QRH : Base de données / Base de données spatiales

Installation de PostgreSQL / PostGIS

Linux (Manjaro)

Pour installer PostreSQL et PostGIS sur Manjaro, il est nécessaire d'installer les paquets suivants :

```
postgresql
postgresql-docs
postgis
```

Fichiers d'installation pour PostgreSQL / PostGIS sur Manjaro.[1]

Il peut être utile d'installer DBeaver ou pgAdmin pour faciliter la gestion des bases [2]. Une fois Snap installé et configuré, on peut executer la commande

```
sudo snap install dbeaver-ce
```

Pour installer la version community de Dbeaver]

Initialisation du Cluster et ajout d'utilisateurs

Par défaut, sous Manjaro, le Cluster de base de données n'est pas initialisé. De même seul l'utilisateur **postgres** est créé. Il est donc nécessaire d'initialiser le Cluster et de créer de nouveaux utilisateurs

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

Pour démarrer ou arreter le serveur, on utilise les commandes :

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

Il est bien sûr possible de démarrer le serveur automatiquement au démararrage de la session :

```
systemctl enable postgresql
```

Pour ajouter des utilisateurs, on utilise la commande suivante, qui créer un utilisateur nommé **lambda** ayant les droits de créer des bases de données :

Tout d'abord, on se connecte avec l'utilisateur **postgres** qui est le seul utilisateur ayant les droits pour se connecter à PostgreSQL, par défaut :

```
su - postgres
psql
```

Si la commande est executée correctement, l'utilisateur est maintenant connecté en tant qu'utilisateur **postgres** sur le shell de PostgreSQL (psql).

On peut alors utiliser les commandes suivantes pour créer un utilisateur nommé **lambda** ayant pour mot de passe **omega** :

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

Création d'un shéma et modification des droits de l'utilisateur

Sur les versions récentes de PostgreSQL (à partir de la version 15 ?), il est nécessaire de créer des schémas pour des questions de droits d'accès. Un schéma peut regrouper plusieurs tables et permet de mieux organiser ces données.

Pour créer un schéma, il faut utiliser la commande :

```
CREATE SCHEMA myschema;
```

Il faut ensuite donner les droits aux utilisateurs de ce schéma :

- Il faut se connecter avec un super-utilsateur, ici **postgres** sur la base **eAIP_France** :

```
psql -U postgres -d eAIP_France
```

- Rendre l'utilisateur **omega** propriétaire du schéma **schema_name** :

```
ALTER SCHEMA schema_name OWNER TO omega;
```

- Il peut être utile voir nécessaire de donner des droits à l'utilsateur **omega**. Il est par exemple possible de lui donner les droits de superutilisateur :

```
ALTER USER omega WITH SUPERUSER;
```

Création d'une base de données

Pour créer une base de données, on utilise (en dehors du shell de PostgreSQL) :

```
createdb *nom_de_la_base*
```

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

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

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

Utilisation de PostgreSQL et PostGIS

Création d'une 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 des données spatiales avec PostgreSQL, il faut activer l'extension PostGIS dans la base de données utilisée.

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

Insertion de données spatiale

On peut ensuite insérer des données spatiale de la façon suivante :

```
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**

Exemple d'utilisation : calcul de cap et distance entre deux aéroports

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.coordo, c2.coordo)) + 360
    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, de Paris Orly (LFPO, ORY) vers Marseille Provence (LFML, MRS), on obtient bien le cap réciproque et la même distance :

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

Exemple d'utilisation : Trouver les points les plus proches d'un point d'intérêt

On peut déterminer les points les plus proches d'un autre point, de la façon suivante :

```
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)
```

Utilisation de GeoAlchemy 2 avec SQLAlchemy v1

Note : La syntaxe risque de changer avec le passage de SQLAlchemy en v2 (à suivre de près donc)

GeoAlchemy est une extension de SQLAlchemy permettant l'utilisation de données spatiales

Exemple d'utilisation : Distance et cap entre deux aéroports

Voir la machine virtuelle Manjaro pour l'ensemble du code.

Fichier **main.py**

```
"""
Demonstration de GeoAlchemy 2
"""
__author__ = 'Damien GABRIEL'

import psycopg2
import sqlalchemy as sa
from sqlalchemy import create_engine, text
from sqlalchemy.orm import sessionmaker
from Airport import Airport

if __name__ == '__main__':
    engine = create_engine('postgresql://pgm:pgm@localhost/postgis', echo=False)
    insp = sa.inspect(engine)
    if insp.has_table("airport"):
        print("Airport table exists")
        Airport.__table__.drop(engine)
    Airport.__table__.create(engine)
    Session = sessionmaker(bind=engine)
    session = Session()
```

```
    Marseille_Provence = Airport(name='Marseille Provence', icao='LFML', iata='MRS',
                                 coordinates='POINT(5.21500 43.43667)')
    Paris_Orly = Airport(name='Paris Orly', icao='LFPO', iata='ORY',
                         coordinates='POINT(2.37958 48.72328)')
    session.add(Marseille_Provence)
    session.add(Paris_Orly)
    session.commit()
    # textual query should now be explicit text!
    text_query = text("""
        SELECT ST_DISTANCE(c1.coordinates,
        c2.coordinates) / 1000 as "Distance (km)",
        CASE WHEN degrees(ST_AZIMUTH(c1.coordinates, c2.coordinates)) < 0
        THEN degrees(ST_AZIMUTH(c1.coordinates, c2.coordinates)) + 360
        ELSE
        degrees(ST_AZIMUTH(c1.coordinates, c2.coordinates))
        END AS "Cap"
        FROM Airport c1, Airport c2
        WHERE c1.name = 'Marseille Provence' AND c2.name = 'Paris Orly';
        -- WHERE c1.icao = 'LFML' AND c2.icao = 'LFPO';
        """)
    distance_cap = session.execute(text_query)
    results_as_dict = distance_cap.mappings().all()
    print(results_as_dict)
```

Fichier **Airport.py**

```
"""
Demonstration de l'utilisation de GeoAlchemy 2
Définition de la table Airport
"""
__author__ = "Damien GABRIEL"

from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String
from geoalchemy2 import Geography

Base = declarative_base()


class Airport(Base):
    __tablename__ = 'airport'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    icao = Column(String)
    iata = Column(String)
    coordinates = Column(Geography('POINT', srid=4326))
```

Les dépendances du projets sont :

Fichier **requirements.txt**

```
psycopg2~=2.9.5
SQLAlchemy~=1.4.46
setuptools~=65.5.1
GeoAlchemy2~=0.12.5
```

Le résultat de la requête est :

```
[{'Distance (km)': 627.09856039227, 'Cap': 340.54449384096057}]
```

Utilisation de GeoAlchemy 2 avec SQLAlchemy v2

Pour le moment, je n'ai vu qu'un seul changement à faire pour que le programme soit compatible avec SQLAlchemy v2.

Il faut déclarer la requete textuelle explicitement comme étant du texte.

Voir le fichier main modifié :

Fichier **main.py**

```
"""
Demonstration de GeoAlchemy 2
"""
__author__ = 'Damien GABRIEL'

import psycopg2
import sqlalchemy as sa
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from Airport import Airport

if __name__ == '__main__':
    engine = create_engine('postgresql://pgm:pgm@localhost/demo_geography',
echo=False)
    insp = sa.inspect(engine)
    if insp.has_table("airport"):
        print("Airport table exists")
        Airport.__table__.drop(engine)
    Airport.__table__.create(engine)
    Session = sessionmaker(bind=engine)
    session = Session()
    Marseille_Provence = Airport(name='Marseille Provence', icao='LFML', iata='MRS',
                                 coordinates='POINT(5.21500 43.43667)')
```

```
    Paris_Orly = Airport(name='Paris Orly', icao='LFPO', iata='ORY',
                         coordinates='POINT(2.37958 48.72328)')
    session.add(Marseille_Provence)
    session.add(Paris_Orly)
    session.commit()
    distance_cap = session.execute("""
    SELECT ST_DISTANCE(c1.coordinates,
        c2.coordinates) / 1000 as "Distance (km)",
        CASE WHEN degrees(ST_AZIMUTH(c1.coordinates, c2.coordinates)) < 0
        THEN degrees(ST_AZIMUTH(c1.coordinates, c2.coordinates)) + 360
        ELSE
        degrees(ST_AZIMUTH(c1.coordinates, c2.coordinates))
        END AS "Cap"
    FROM Airport c1, Airport c2
    WHERE c1.name = 'Marseille Provence' AND c2.name = 'Paris Orly';
    -- WHERE c1.icao = 'LFML' AND c2.icao = 'LFPO';
        """)
    results_as_dict = distance_cap.mappings().all()
    print(results_as_dict)
```

Avec les nouvelles déclarations pour la définition des classes, le fichier Airport.py devient

Fichier **Airport.py**

```
"""
Demonstration de l'utilisation de GeoAlchemy 2
Définition de la table Airport
"""
__author__ = "Damien GABRIEL"

from sqlalchemy import Column, Integer, String
from geoalchemy2 import Geography
from sqlalchemy.orm import Mapped, DeclarativeBase, mapped_column


class Base(DeclarativeBase):
    pass


class Airport(Base):
    __tablename__ = 'airport'
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str] = mapped_column(String(40))
    icao: Mapped[str] = mapped_column(String(4))
    iata: Mapped[str] = mapped_column(String(3))
    coordinates: Mapped[Geography] = mapped_column(Geography('POINT', srid=4326))
```

Les dépendances, mise à jour, du projets sont :

Fichier **requirements.txt**

```
psycopg2~=2.9.5
SQLAlchemy~=2.0.12
setuptools~=65.5.1
GeoAlchemy2~=0.12.5
```

Mise à jour de PostgreSQL

Lors de mise à jour (particulièrement de de Linux), il peut être nécessaire de mettre à jour PostgreSQL. Dans mon cas, il s'agit de passer PostgreSQL de la version 14 à 15 sous Manjaro.

Les liens de référence sont suivant:

- Lien vers le Wiki d'Arch Linux sur le sujet
- Lien vers la documentation officielle de PostgreSQL
- https://postgresql-tutorial.com/postgresql-how-to-grant-superuser-privileges/
- https://postgresql-tutorial.com/postgresql-how-to-change-the-owner-of-a-schema/
- https://dba.stackexchange.com/questions/209179/how-to-assign-privileges-on-a-postgresql-schema-to-a-user

[1] Pour installer les paquets depuis un fichier utiliser la commande : pacman -S $(cat yourfilename | cut -d' ' -f1), cf lien vers la response sur Unix StackExchange

[2] Installation de DBeaver via Snap sous Manjaro : Lien site snapcraft.io