OBJECTIFS

Objectif Global

Découverte des principaux modes d'interfacer python avec une base de données SQL

Objectifs pédagogiques

- Se connecter à une base de données SQL en python
- Se connecter à une base de données SQL avec pandas
- Se connecter à une base de données SQL avec un ORM

Démarche pédagogique

Dans ce cours, nous allons apprendre à interagir avec une base de données PostgreSQL en Python en utilisant différentes approches : les bibliothèques de base comme psycopg2, des bibliothèques plus avancées comme SQLAlchemy pour une approche ORM, et pandas pour manipuler et analyser les données.

La base de données contient des informations sur les emplacements pour garer et charger des vélos ou trottinettes électriques dans la ville fictive de **Numcity**. En alternant explications détaillées et exercices pratiques, nous allons explorer comment interroger cette base de données et visualiser les données de manière informative à travers des graphiques et des cartes.

Compétences

- Ecrire un script python permettant de se connecter à une base de données
- Mettre en oeuvre un ORM python
- Afficher des données d'une base de données sur une carte

MODALITÉS

Durée

1,5J

ITÉRATION 1

psycopg2

Modalités

- Travail individuel en autonomie

Livrables

Notebook contenant les instructions python permettant de répondre au cahier des charges

Objectifs

Utiliser les outils python de base.

Partie 1 : Connexion à PostgreSQL avec psycopg2

1.1 Qu'est-ce que psycopg2 ?

psycopg2 est une bibliothèque Python qui permet de se connecter et d'interagir avec une base de données PostgreSQL. Elle est idéale pour exécuter des requêtes SQL de manière directe. Elle prend en charge les transactions, permet l'utilisation des curseurs pour des requêtes complexes, et inclut des fonctionnalités avancées telles que la gestion des pools de connexions. Vous pouvez consulter la documentation officielle pour des détails sur les fonctionnalités et les exemples d'utilisation. Pour une introduction détaillée, visitez ce tutoriel en ligne.

1.2 Connexion à la base de données

Code Exemple :

```
import psycopg2

# Connexion à la base de données
conn = psycopg2.connect(
        dbname="numcity",
        user="votre_utilisateur",
        password="votre_mot_de_passe",
        host="localhost",
        port="5432"
)

# Création d'un curseur
cursor = conn.cursor()

# Exécution d'une requête SQL
cursor.execute("SELECT * FROM parking_charging_spots LIMIT 5;")
rows = cursor.fetchall()

# Affichage des résultats
for row in rows:
        print(row)

# Fermeture de la connexion
cursor.close()
conn.close()
```

Ressources supplémentaires :

- Documentation officielle de psycopg2

1.3 Exercice pratique

Pour cet exercice, vous allez créer un notebook Jupyter qui permet de tester des requêtes SQL sur toutes les tables de la base de données numcity.

1. Dans un nouveau notebook Jupyter, connectez-vous à la base de données numcity en utilisant psycopg2.
2. Créez une fonction Python permettant d'exécuter une requête SQL et de retourner les résultats dans un format lisible.
3. Écrivez des requêtes pour interroger chacune des tables disponibles dans votre base de données.
4. Testez vos requêtes et vérifiez les résultats pour vous assurer qu'ils sont corrects.
5. Documentez votre notebook avec des descriptions des tables et des objectifs de chaque requête.

ITÉRATION 2

pandas

Modalités

- Travail individuel en autonomie

Livrables

Notebook contenant les instructions python permettant de répondre au cahier des charges en utilisant la bibliothèque pandas

Objectifs

Utiliser pandas pour s'interfacer avec une base de données.

Partie 2 : Manipuler les données avec pandas

2.1 Importer des données directement dans un DataFrame

pandas est une bibliothèque puissante pour la manipulation et l'analyse des données. Lorsqu'il est utilisé avec des bases de données, pandas permet de charger directement les résultats d'une requête SQL dans un DataFrame. Cela simplifie grandement le traitement des données, notamment pour les opérations telles que le filtrage, l'agrégation et la création de graphiques.

Voici quelques avantages principaux de pandas pour travailler avec des bases de données :

- **Facilité d'utilisation** : Les requêtes SQL peuvent être exécutées directement depuis pandas via read_sql_query.
- **Intégration native avec d'autres bibliothèques** : Les données peuvent facilement être utilisées avec des bibliothèques comme matplotlib, seaborn ou folium pour la visualisation.
- **Puissantes capacités de traitement** : pandas permet de manipuler des données structurées de manière performante.

Ressources supplémentaires :

- Documentation officielle de pandas
- Tutoriel sur pandas et SQL
- Guide complet sur pandas

Code Exemple :

```
import pandas as pd
 import psycopg2

 # Connexion à la base de données
 conn = psycopg2.connect(
        dbname="numcity",
        user="votre_utilisateur",
        password="votre_mot_de_passe",
        host="localhost",
        port="5432"
)
```

```
# Charger des données dans un DataFrame
query = "SELECT * FROM parking_charging_spots;"
df = pd.read_sql_query(query, conn)

# Afficher un aperçu des données
print(df.head())

# Fermer la connexion
conn.close()
```

2.2 Analyse et visualisation

Vous pouvez utiliser des fonctions pandas pour analyser les données et des bibliothèques comme matplotlib et seaborn pour visualiser les résultats.

Code Exemple :

```
import matplotlib.pyplot as plt

# Compter les types de stationnements
type_counts = df['type'].value_counts()

# Création d'un graphique
type_counts.plot(kind='bar')
plt.title("Nombre de parkings par type")
plt.xlabel("Type de parking")
plt.ylabel("Nombre")
plt.show()
```

2.3 Exercice pratique

Pour cet exercice, vous allez créer un notebook Jupyter permettant de tester les requêtes sur toutes les tables de la base de données numcity en utilisant pandas.

1. Dans un nouveau notebook Jupyter, connectez-vous à la base de données numcity en utilisant psycopg2 ou une autre méthode de connexion compatible.
2. Utilisez la fonction read_sql_query de pandas pour charger les données de chaque table dans des DataFrames distincts.
3. Documentez les colonnes principales et les types de données disponibles dans chaque DataFrame, et présentez un échantillon des lignes avec head().

4. Réalisez des requêtes et des analyses sur ces DataFrames pour obtenir des informations telles que :
   - Le nombre d'emplacements disponibles par type (vélo, trottinette, mixte).
   - La répartition des emplacements par quartier.
5. Créez un graphique en barres à partir des résultats des analyses précédentes pour illustrer les données.

Bonus : Ajoutez des sections dans le notebook pour expliquer chaque étape et inclure des descriptions des analyses réalisées.

ITÉRATION 3

SQLAlchemy

Modalités

- Travail individuel en autonomie

Livrables

Notebook contenant les instructions python permettant de répondre au cahier des charges en utilisant la bibliothèque SQLAlchemy

Objectifs

Utiliser un ORM python pour s'interfacer avec une base de données SQL.

Partie 3 : Une approche avancée avec SQLAlchemy

3.1 Introduction à SQLAlchemy

Un ORM (Object Relational Mapping) est un outil qui permet de mapper des bases de données relationnelles à des objets Python. Cela signifie que vous pouvez manipuler les tables et les enregistrements de votre base de données comme s'il s'agissait de classes et d'instances Python. Cela rend l'interaction avec la base plus intuitive et moins sujette aux erreurs causées par des requêtes SQL complexes.

SQLAlchemy est l'un des ORM les plus populaires pour Python. Il offre une flexibilité remarquable pour interagir avec des bases de données, tout en étant suffisamment performant pour des cas d'utilisation complexes. SQLAlchemy se compose de deux parties principales :

1. **Core SQLAlchemy** : Une interface pour écrire des requêtes SQL directement, en utilisant des outils programmatiques.
2. **SQLAlchemy ORM** : Un mécanisme permettant de mapper des tables à des classes Python et d'utiliser des objets Python pour interagir avec la base.

Voici quelques avantages clés de l'utilisation de SQLAlchemy :

- **Abstraction des requêtes SQL** : Vous pouvez écrire moins de code SQL brut, ce qui simplifie la maintenance.
- **Validation des types** : SQLAlchemy assure que vos données sont cohérentes avec les types définis dans votre modèle.
- **Support multi-base de données** : Il peut fonctionner avec différents moteurs comme PostgreSQL, MySQL, SQLite, etc.
- **Gestion des relations** : SQLAlchemy gère les relations entre les tables grâce à des outils simples comme relationship.

Ressources supplémentaires :

- Documentation officielle de SQLAlchemy
- Tutoriel SQLAlchemy pour débutants
- Comparaison Core vs ORM SQLAlchemy

En combinant puissance et simplicité, SQLAlchemy est un excellent choix pour les projets qui nécessitent une gestion avancée des bases de données.

3.2 Connexion et exécution de requêtes

Code Exemple :

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

# Connexion avec SQLAlchemy
engine = create_engine('postgresql://votre_utilisateur:votre_mot_de_passe@localhost/numcity')
Session = sessionmaker(bind=engine)
session = Session()

# Exemple de requête
result = session.execute("SELECT * FROM parking_charging_spots LIMIT 5;")
for row in result:
    print(row)

# Fermeture de la session
session.close()
```

3.3 ORM : Définir une table comme classe Python

Code Exemple :

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class ParkingSpot(Base):
    __tablename__ = 'parking_charging_spots'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    type = Column(String)

# Exemple de requête avec ORM
for spot in session.query(ParkingSpot).filter_by(type='bike'):
    print(spot.name)
```

3.4 Exercice pratique

Pour cet exercice, vous allez créer un notebook Jupyter permettant de refaire toutes les requêtes vues dans les exercices précédents en utilisant SQLAlchemy.

1. Dans un nouveau notebook Jupyter, configurez une connexion à la base de données numcity en utilisant SQLAlchemy.
2. Créez une classe ORM représentant la table parking_charging_spots.
3. Réalisez des requêtes similaires à celles des exercices précédents, notamment :
   - Afficher tous les emplacements disponibles.
   - Filtrer les emplacements selon leur type (vélo, trottinette, etc.).
   - Trouver les emplacements situés dans un quartier particulier.
4. Documentez chaque requête dans le notebook en expliquant ce qu'elle fait et les résultats obtenus.
5. Pour chaque requête, affichez les résultats dans un format lisible et utilisez des visualisations pertinentes si possible (par exemple, des graphiques).

Bonus : Ajoutez une section dans le notebook expliquant les avantages d'utiliser un ORM comme SQLAlchemy pour ces types d'opérations.

ITÉRATION 4

Cartographie

Modalités

- Travail individuel en autonomie

Livrables

Notebooks mis à jour pour afficher des cartes

Objectifs

Utiliser la bibliothèque folium pour créer des cartes

Partie 4 : Visualisation avancée avec folium

4.1 Introduction à folium

folium est une bibliothèque Python puissante pour créer des cartes interactives directement dans vos notebooks Jupyter. Elle permet de superposer des données sur des cartes en utilisant des marqueurs, des polygones ou d'autres objets visuels. Cette bibliothèque est particulièrement utile pour les analyses géospatiales et les présentations interactives des données.

Voici quelques caractéristiques principales de folium :

- **Facilité d'utilisation** : Avec quelques lignes de code, vous pouvez créer des cartes interactives enrichies.
- **Compatibilité avec d'autres bibliothèques** : Elle fonctionne bien avec des outils comme pandas pour visualiser des données géographiques stockées dans des DataFrames.
- **Personnalisation avancée** : folium prend en charge des options avancées, comme les tuiles de carte personnalisées, les pop-ups interactifs et les filtres.
- **Présentation conviviale** : Les cartes générées peuvent être sauvegardées sous forme de fichiers HTML pour être partagées facilement.

Ressources supplémentaires :

- Documentation officielle de folium
- Tutoriel complet sur folium
- Exemples avancés avec folium

4.2 Créer une carte avec des données de PostgreSQL

Code Exemple :

```
import folium

 # Charger les données nécessaires
 bike_spots = df[df['type'] == 'bike']

 # Créer une carte
 m = folium.Map(location=[48.8566, 2.3522], zoom_start=12)

 # Ajouter des marqueurs
```

```
for _, row in bike_spots.iterrows():
        folium.Marker(
        location=[row['latitude'], row['longitude']],
        popup=row['name']
        ).add_to(m)

# Afficher la carte
m
```

4.3 Exercice pratique

Pour cet exercice, vous allez étendre les notebooks créés précédemment en y ajoutant des visualisations interactives avec folium.

1. Dans le notebook Jupyter, commencez par charger les données de la base de données numcity en utilisant pandas ou SQLAlchemy, selon le contexte des exercices précédents.
2. Créez une carte de base centrée sur les coordonnées principales de la ville de Numcity.
3. Ajoutez des marqueurs pour afficher les emplacements des points de recharge pour les trottinettes électriques.
4. Créez une légende ou des pop-ups pour chaque marqueur afin d'afficher des informations comme le nom de l'emplacement et le type de service disponible.
5. Ajoutez une fonctionnalité permettant de filtrer les emplacements sur la carte selon leur type (vélo, trottinette, mixte).
6. Documentez chaque étape dans votre notebook avec des explications et des visualisations pour illustrer les résultats.

Bonus : Sauvegardez votre carte en tant que fichier HTML et ajoutez un bouton interactif pour recharger différents jeux de données si besoin.

Conclusion

Ce cours vous a permis de découvrir différentes méthodes pour interagir avec une base de données PostgreSQL en Python. Vous avez utilisé psycopg2 pour une interaction directe, pandas pour une analyse rapide des données, et SQLAlchemy pour une approche ORM avancée. Enfin, vous avez visualisé les données géographiques avec folium.

Prochaines étapes :

- Intégrer ces techniques dans un projet complet.
- Explorer d'autres bibliothèques comme GeoPandas pour des analyses géospatiales avancées. Vous pouvez consulter la documentation officielle de GeoPandas pour explorer ses fonctionnalités et découvrir comment elle simplifie le traitement et l'analyse des données géographiques en Python.

Si vous avez des questions, n'hésitez pas à les poser !