Base de données

NSI - Tle
Activité

LYCÉE BOISSY D'ANGLAS
LYCÉE DES MÉTIERS DE LA CRÉATION INDUSTRIELLE

Contenu : - Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel
- Base de données relationnelle

Capacités attendues : Identifier les concepts définissant le modèle relationnel.
Savoir distinguer la structure d'une base de données de son contenu.
Repérer des anomalies dans le schéma d'une base de données.

Introduction

Dans cette activité cours nous allons traiter des différentes manières d'agir sur des bases de données avec Python. L'application de ce traitement des bases nous permettra de tracer un parcours sur OpenStreetMap.

Il existe de nombreux logiciels de gestion de bases de données relationnelle sur le marché comme postgresql, mysql, sqlite ... une base de données relationnelle se gère via du SQL, un langage de requête de bases de données.

Nous allons donc étudier cette gestion de bases de données via Python.

De nombreuses bibliothèques de fonctions ont été développées pour interfacer du code python avec les différents SGBD. Ainsi par exemple psycopg est un paquet Python permettant de gérer une base de données Postgresql en Python.

Nous utiliserons ici SQLite. SQLite est un système de gestion de base de données (SGBD) écrit en C qui sauvegarde la base sous forme d'un fichier multi-plateforme.

SQLite est souvent le moyen le plus rapide et le plus simple de gérer une base de données, et comme Python aime la simplicité il existe un paquet standard Python pour faire l'interface entre Python et SQLite.

La base de données est une notion des années 60. Mais le modèle relationnel date de 70 et les SGBD de 80. Les bases de données relationnelles sont basées sur la théorie des ensembles avec l'utilisation des opérateurs de l'algèbre relationnel (l'union, la différence, produit cartésien, la projection, la sélection, le quotient et la jointure)

Rappel :structure d'une base de données :

Une base de données est composée de "**tables**" contenant des enregistrements. Chaque table est un tableau où les colonnes sont appelées "***champs***" (ou "***Attributs***") et les lignes "***enregistrements***". Dans chaque cellule, on place les "***valeurs***" (données).

Colonne ou Champ ou Attribut

Identificateur	Pièce	Temp mesurée	Consigne	Date	Heure
1	Cuisine	19.1	20	09/02/20	08:00
2	Séjour	20.3	21	09/02/20	12:00
3	Chambre 1	17.5	18	10/02/20	08:00
4	Chambre 2	17.4	18	10/02/20	12:00
5	Salle de bain	20.4	20	11/02/20	08:00

enregistrements

Valeur

1. Utilisation du module Sqlite3 de python

Nous devons tout d'abord importer le module sqlite3 (le 3 renseigne la version de sqlite) dans un nouveau fichier Python créé avec Pyzo, nommé "***Activité1_base_de_donnees.py***".

```
import sqlite3 # import du module sqlite3 qui permet d'interagir avec la base de données
```

a. Se connecter à la base de données

Le paquet sqlite3 contient la méthode ***sqlite3.connect*** qui offre tous les services de connexions à une base de données Sqlite3.

```
ma_connexion = sqlite3.connect('Ma_base.db')
# crée ou ouvre un fichier nommé 'Ma_base.db' qui doit donc se trouver dans le même
# dossier que l'endroit où le fichier Python est placé
```

Attention : on ouvre une connexion avec un fichier, il est donc indispensable de fermer cette connexion à la fin.

```
ma_connexion.close()
```

b. Créer et supprimer des tables

On se propose de créer dans la base de données "***Ma_base***" une table nommée "***Trajet1***" constituée de 4 *champs*, afin d'y enregistrer les données, composées d'un identifiant (***id***), des coordonnées des points gps (***latitude*** et ***longitude***) du parcours du trajet1, ainsi que le commentaire à afficher sur le marqueur (***commentaire***).

id	latitude	longitude	commentaire

Il faut construire cette table en renseignant les *champs*. Chaque table doit correspondre à une entité réelle, par exemple la table "***Trajet1***" correspond à toutes les informations nécessaires à la définition d'un point gps. Lorsqu'une table est définie, on définit en même temps le type et le nom de chaque *champ*, c'est-à-dire chaque élément décrivant l'entité. Par exemple, lors de la création de la table "***Trajet1***", il faut définir les *champs* "***id***", "***latitude***", "***longitude***" et "***commentaire*** " avec leur type de données respectifs. Pour cela, on met dans une variable "***requete***" les instructions à réaliser. Puis on exécute cette requête par "***cursor.execute(requete)***".

```
#À FAIRE 1
ma_connexion = sqlite3.connect('Ma_base.db')
cursor = ma_connexion.cursor()
# cursor est un object auquel on passe les commandes SQL en vue d'être executées.
requete = '''CREATE TABLE Trajet1(id INTEGER PRIMARY KEY, latitude FLOAT, longitude FLOAT,  commentaire TEXT)'''
cursor.execute(requete)
# execute ne lance pas la commande
# il est important de noter le champ 'id' de type nombre entier qui sert de clef primaire
```

La *clef primaire* d'une table dans une base de données relationnelle sert d'identifiant unique à une entité. Par exemple, la clef primaire sert à connaitre une unique coordonnée en particulier. La latitude est une très mauvaise clef primaire car une même latitude peut être partagée par un même point. Par exemple, le point de coordonnées 45.249350, 4.671999 correspond au parking de la place michelet du lycée Boissy d'Anglas, et le point de coordonnées 45.249350, 4.68 se situe vers Nicolas Motos. Ils ont pourtant la même valeur de latitude… Par habitude, un champ "***id***" est souvent créer pour identifier chaque enregistrement de la table.

```
ma_connexion.commit()
```

La fonction **'*commit*'** permet de lancer la commande, c'est-à-dire que le commit permet de créer véritablement la table "***Trajet1***" et de la sauvegarder dans "***Ma_base.db***". Ne jamais oublier de **'*commit*'** dans les commandes car sinon il ne se passera rien…

```
# Pour supprimer la table il suffit de :
# requete = '''DROP TABLE Trajet1;'''
# cursor.execute(requete)
# ma_connexion.commit()
```

Attention : la commande '***commit***' est une fonction de '***ma_connexion***' et non de '***cursor***'.

c. Insérer des données dans la table

```
#À FAIRE 2
requete = '''INSERT INTO Trajet1 VALUES (1, 45.248952, 4.675639, 'Place Michelet - Annonay');'''
cursor.execute(requete)
```

Chaque ajout de données passera par le mot clef SQL "**INSERT INTO**" suivi du nom de la table dans laquelle on souhaite ajouter des données. Suit alors le mot clef "**VALUES**" suivi des valeurs à ajouter en vérifiant que le type est le bon.

```
ma_connexion.commit()
```

Un point de coordonnées gps a été ajouté à la table.

- Ajouter des données à l'aide de variables :

```
#À FAIRE 3
lat_point = 45.248733
long_point = 4.675899
com_point = ""
requete = '''INSERT INTO Trajet1 VALUES (?,?,?,?);'''
cursor.execute(requete, (2, lat_point, long_point, com_point))
ma_connexion.commit()
```

- Réaliser des insertions plus rapidement :

```
#À FAIRE 4
points = [(3, 45.248001, 4.676494, ""), (4, 45.247654, 4.676781, ""), (5, 45.247890, 4.677469, "")]
# liste de tuples contenant toutes les informations nécessaires pour ajouter les points à la base de données
requete = '''INSERT INTO Trajet1 VALUES (?,?,?,?);'''
cursor.executemany(requete, points)
ma_connexion.commit()
```

- Comptage des id :

```
#À FAIRE 5
requete = '''SELECT * FROM Trajet1;'''
cursor.execute(requete)
last_id = cursor.lastrowid    # last_id contient le dernier id ajouté
print('la dernière ligne que vous venez de renseigner est : %s' % last_id)
```

le dernier id est : 5

- Retrouver le dernier id renseigné dans une base :

```
requete = '''SELECT * FROM Trajet1;'''
cursor.execute(requete)
liste_complete = cursor.fetchall()
#prendre le dernier enregistrement
if len(liste_complete) == 0 :
        id =0
else :
        dernier = liste_complete[len(liste_complete)-1]
        id = dernier[0]
print(id)
```

- Ajouter automatiquement un nouveau point :

```
#À FAIRE 6
requete = '''SELECT * FROM Trajet1;'''
cursor.execute(requete)
id_suivant = cursor.lastrowid + 1
requete = '''INSERT INTO Trajet1 VALUES (?,?,?,?);'''
point_a_inserer = (id_suivant, 45.248069, 4.679126, '')
cursor.execute(requete, point_a_inserer)
ma_connexion.commit()
```

d. Accéder aux données

- Pour sélectionner des données, il faut utiliser la commande SQL "***SELECT***"

```
#À FAIRE 7
requete = '''SELECT * FROM Trajet1;'''
cursor.execute(requete)
premier_point = cursor.fetchone()
print('7- le premier point est : ', premier_point)
```

Ici, la réponse est un tuple…

- Pour afficher les informations concernant le premier point :

```
#À FAIRE 8
print("8- Le premier point dans la base est : %s de latitude et %s de longitude" % (premier_point[1], premier_point [2]))
# l'indice du tuple commence à 0 mais est réservé à l'id de l'enregistrement
```

Le premier point est : (1, 45.248952, 4.675639, 'Place Michelet')

- Pour afficher tous les enregistrements :

```
#À FAIRE 9
requete = '''SELECT * FROM Trajet1;'''
cursor.execute(requete)
```

```
points_recuperes = cursor.fetchall()
for point in points_recuperes :
        print ("9- Le point n° %s est de %s de latitude et de %s de longitudes. Le commentaire attaché est : %s" % (point[0], point[1], point[2], point[3]))
        # point correspond à un enregistrement de la table
        # point[indice] correspond au champ correspondant du point en question
```

9- Le point n° 1 est de 45.248952 de latitude et de 4.675639 de longitudes. Le commentaire attaché est : Place Michelet - Annonay
9- Le point n° 2 est de 45.248733 de latitude et de 4.675899 de longitudes. Le commentaire attaché est :
9- Le point n° 3 est de 45.248001 de latitude et de 4.676494 de longitudes. Le commentaire attaché est :
9- Le point n° 4 est de 45.247654 de latitude et de 4.676781 de longitudes. Le commentaire attaché est :
9- Le point n° 5 est de 45.24789 de latitude et de 4.677469 de longitudes. Le commentaire attaché est :
9- Le point n° 6 est de 45.248069 de latitude et de 4.679126 de longitudes. Le commentaire attaché est :

e. Mettre à jour et supprimer des enregistrements

Les mots clefs utilisés sont "***UPDATE***" et "***DELETE***"

On va renseigner une valeur erronée dans la base, puis la modifier :

```
#À FAIRE 10
requete = '''INSERT INTO Trajet1 VALUES (?,?,?,?);'''
cursor.execute(requete, (7, 49.787890, 4.673219, 'erreur'))
ma_connexion.commit()
# visualisation de cette valeur à l'id 7 :
requete = '''SELECT latitude, longitude, commentaire FROM Trajet1 WHERE id=? '''
cursor.execute(requete, (7, ))
print(cursor.fetchone())
```

(49.78789, 4.673219, 'erreur')

```
#À FAIRE 11
# ces coordonnées sont erronées et on va les remplacer
requete = '''UPDATE Trajet1 SET latitude = ?, longitude = ?, commentaire = ? WHERE id = ? '''
cursor.execute(requete, (45.247890, 4.677469, 'valeur changée !', 7))
ma_connexion.commit()
# affichage du résultat
requete = '''SELECT latitude, longitude, commentaire FROM Trajet1 WHERE id=? '''
cursor.execute(requete, (7, ))
print(curseur.fetchone())
```

(45.24789, 4.677469, 'valeur changée !')

On s'aperçoit que la mise est jour est assez classique : Juste après le mot clef "***UPDATE***" on définit la table à mettre à jour. Ensuite vient le mot clef "***SET***" après lequel on définit les champs à mettre à jour puis le mot clef "***WHERE***" permettant de dire quel enregistrement doit être mis à jour.

Pour supprimer le dernier enregistrement : la requête est du type '''DELETE FROM Trajet1 WHERE id = ?;'''
Récupérer le dernier id, afin de lancer une requête permettant de supprimer le dernier enregistrement. Afficher le résultat.

```
#À FAIRE 12
```

Vous devriez obtenir :
(45.248069, 4.679126, '')

Supprimer tous les enregistrements d'une base :

```
#À FAIRE 13
requete = '''DELETE FROM Trajet1;'''
cursor.execute(requete)
requete = '''SELECT * FROM Trajet1 ;'''
cursor.execute(requete)
point = cursor.fetchone()
print('point : ', point)
```

Vous devriez obtenir :
Point : None
Le contenu de la table a été entièrement supprimé.

Supprimer la table :

```
#À FAIRE 14
requete = '''DROP TABLE Trajet1;'''
cursor.execute(requete)
```

Vérifier à l'aide "***DB Browser for sqlite***" si la table a bien été supprimée.

Récapitulatif SQL :

- Tous les noms changés seront marqués entre des dièses *##*
- Connection : *#nom_base_de_donnees#* **= sqlite3.connect**(*#':memory:' ou 'nom_db.db'#*)
- Toujours passer par un "cursor" pour lancer des commandes SQL
- Créer une table : **'''** **CREATE TABLE** *#nom_table#* **(** *#nom_champs# #TYPE_champs# #PRIMARY KEY#* **, ...); '''**
- Insérer des données dans la table : **'''INSERT INTO** *#nom_table#* **VALUES** (*#valeur ou ?#*)**;''',** *#si ? alors mettre ici la valeur correspondante#*
- Sélection des données : **'''SELECT** *#nom_champs ou *# #* **FROM** *#nom_table#* **WHERE** *#comparaison pour filtrage des enregistrements#***;'''**
- Modifier une ou des données : **'''UPDATE** *#nom_table#* **SET** (*#valeur ou ?#*)**;''',** *#si ? alors mettre ici la valeur correspondante#*
- Suppression d'une donnée : '''**DELETE FROM** *#nom_table#* **WHERE** *#comparaison pour filtrage des enregistrements#***;'''**
- Suppression de toute la table : '''**DELETE FROM** *#nom_table#***;'''**
- Suppression d'une table : '''**DROP TABLE** *#nom_table#***;'''**

En récapitulant, une requête :
SELECT ***** | *expression* [, *expression* ...] **FROM** *nom_table1* **WHERE** [*condition logique*]

2. Utilisation du module "*folium*" – génération de page html contenant une carte OpenStreetMap

Pour placer les points sur une carte, nous allons utiliser le module "***folium***" de python qui permet de générer une page html contenant une carte *OpenStreetMap*. Sur celle-ci, on peut venir rajouter des repères, des marqueurs, tracer des lignes reliant plusieurs points, etc…
Pour utiliser ce module, il faut vérifier qu'il soit installé dans python. Dans la fenêtre du shell, taper la commande "***pip install folium***" *et valider.*
Pour générer une carte avec la place Michelet au centre de celle-ci :

```
import folium       # utilisation du module folium

# créer la carte nommée ma_carte avec les coordonnées du point centrale de celle-ci et de la valeur du zoom
ma_carte = folium.Map(location=[45.249337, 4.672245],zoom_start=18)

# générer la carte dans une page html
ma_carte.save('ma_page_de_carte.html')
```

Tester votre code, en nommant le fichier : "***Activité2_generation_carte_OSM.py***". Ouvrir le fichier "***ma_page_de_carte.html***" généré situé dans le même dossier que le fichier python enregistré préalablement. Si on veut l'ouvrir directement dans le navigateur dès l'exécution du programm python, rajouter la ligne "***webbrowser.open('ma_page_de_carte.html')*** " et "***import webbrowser***" en tête de programme (commande "***pip install geopy***" si ce module n'est pas installé…).

Ajouter un marqueur sur la carte avec une fenêtre d'information (pop-up qui s'ouvre après clic sur le marqueur) :

```
folium.Marker([45.249337, 4.672245], popup = "Point 1 du Trajet1", icon = folium.Icon(color='green')).add_to(ma_carte)
# le texte de la fenêtre pop-up peut être changé ainsi que la couleur du marqueur (color='')
```

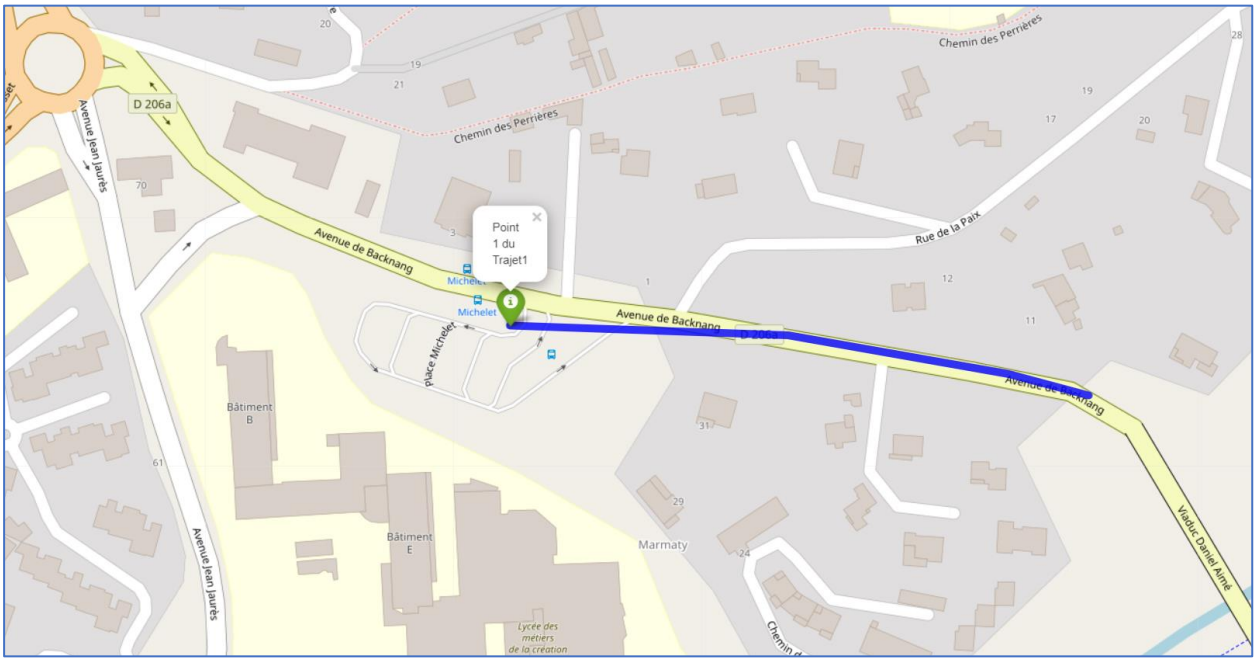
Tracer une ligne entre plusieurs points stockés dans une liste :

```
# création d'une variable liste de liste de points
stock_points = [45.249350, 4.671999], [45.249297, 4.673788], [45.249146, 4.675011], [45.249063, 4.675451]

folium.PolyLine(stock_points, color="blue", weight=8, opacity=0.8).add_to(ma_carte)
# le format des coordonnées sont des réels (float)
# la couleur peut être changée : paramètre color
# épaisseur de la ligne : paramètre weight

# générer la carte dans une page html
ma_carte.save('ma_page_de_carte.html')
```

Pour visualiser le résultat, il faut se rendre dans le dossier du programme et ouvrir le fichier "***ma_page_de_carte.html***" :



Sauvegarder ce programme.

3. Activités

1- Réaliser un programme qui récupère les points de la base de données de la partie 1 et trace le Trajet sur une carte d'une page html. Ce dernier sera sous la forme d'un marqueur, avec le commentaire à afficher dans le pop-up. Pour vous aider, reprendre le programme de l'activité1 où l'on supprimera au préalable les exercices "**À FAIRE 12** à **14**"… Nommer votre fichier "***Activite3_generation_carte_OSM.py***". Voir annexe 2 sur les listes…

2- Réaliser 2 programmes python. Un devra renseigner la base de données avec les données rentrées au clavier par l'utilisateur. Le 2ème générera la carte OSM en allant lire les données dans la base de données.
Pour le premier programme, l'utilisateur renseignera les points un à un, grâce au module graphique Tkinter sous la forme "***latitude, longitude, commentaire***" en respectant la casse. La gestion des fenêtres est proposée dans le fichier "***renseignement_base.py***" qu'il s'agira de compléter dans la fonction valider, afin d'écrire dans la base de données les coordonnées latitude et longitude et les commentaires éventuels des points.

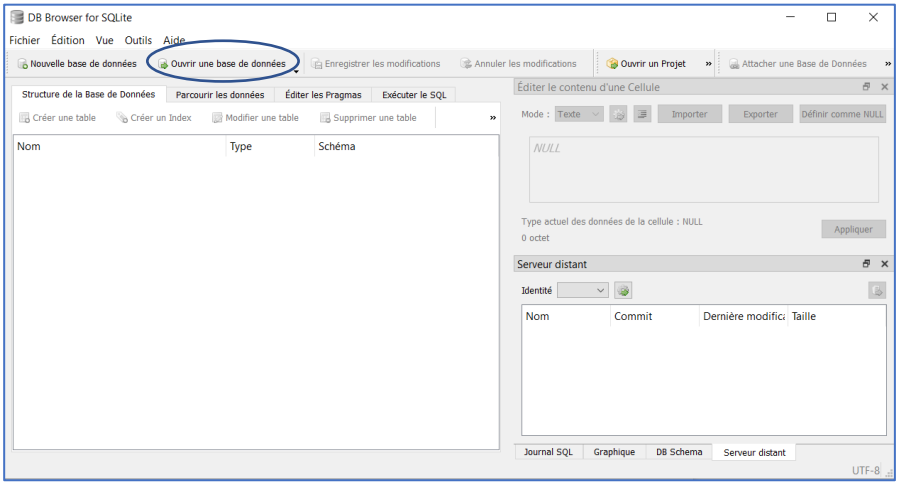
3- Pour aller plus loin…

À chaque saisie de nouvel enregistrement de données dans la base, on veut regénérer la carte. Il faudra alors détecter s'il y a un nouvel enregistrement et relancer ainsi la création de la carte contenant le nouveau point en déplaçant le marqueur sur le dernier point. Modifier vos programmes précédents pour obtenir le fonctionnement souhaité.

4. ANNEXE 1 : Utilisation de DB Browser for SQLite

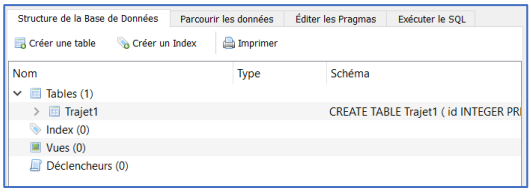
Le programme DB Browser for SQLite permet de pouvoir gérer les bases SQLite. Nous utiliserons sa version portable, disponible sur le réseau (*V:\SQLite_Browser_Portable*).

Pour ouvrir la base, cliquer sur "***Ouvrir une base***"

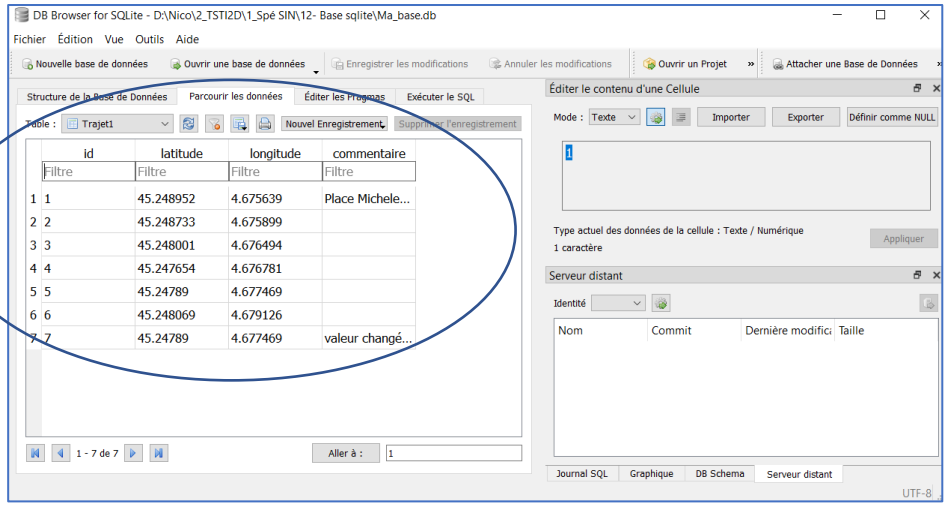


Sélectionner la base créée et valider

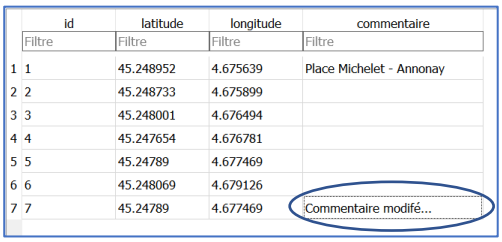
Dans l'onglet, on retrouve la structure de la base. Ici, il n'y a qu'une table, nommée "***Trajet1***".



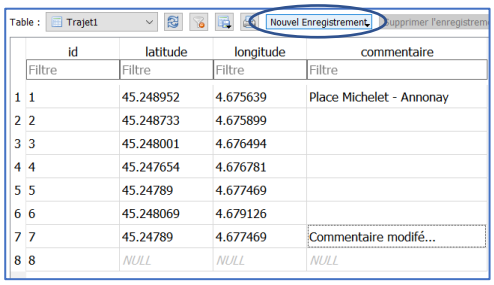
Pour voir les données, cliquer sur l'onglet Parcourir les données. Les enregistrements apparaissent avec les différents champs (colonnes de la structure de donnée) :



On peut modifier une donnée en double cliquant sur celle-ci :



Pour ajouter ou supprimer une ligne :



	id	latitude	longitude	commentaire
	Filtre	Filtre	Filtre	Filtre
1	1	45.248952	4.675639	Place Michelet - Annonay
2	2	45.248733	4.675899	
3	3	45.248001	4.676494	
4	4	45.247654	4.676781	
5	5	45.24789	4.677469	
6	6	45.248069	4.679126	
7	7	45.24789	4.677469	Commentaire modifé...
8	8	NULL	NULL	NULL

Ici l'enregistrement avec l'**id 8** a été créé On peut alors compléter les données en double-cliquant dessus. Pour supprimer la ligne, cliquer sur "***Supprimer l'enregistrement***".

Pour valider les modifications et écrire dans la base, cliquer sur "***Enregistrer les modifications***".

5. ANNEXE 2 : Travail sur liste

- Création d'une liste vide :

```
ma_liste_de_points = []
```

- Récupération de tous les champs de chaque entrée de la base :

```
requete = '''SELECT * FROM Trajet1 ;'''
cursor.execute(requete)
ma_liste_de_points = cursor.fetchone()
print(ma_liste_de_points)
```

[(0, 45.248952, 4.675639, 'Place Michelet'), (1, 45.248733, 4.675899, 'rien'), (2, 45.248001, 4.676494, 'rien'), (3, 45.247654, 4.676781, 'rien'), (4, 45.24789, 4.677469, 'bus3'), (5, 45.247785, 4.678495, 'rien'), (6, 45.248069, 4.679126, 'rien')]

Ici, ***ma_liste_de_points*** est une liste de liste, contenant tous les champs de tous les points de la base de données. Elle est du type :
ma_liste_de_points = [(id0, lat0, lon0, comm0), (id1, lat1, lon1, comm1), (id2, lat2, lon2, comm2), (id3, lat3, lon3, comm3), etc…]

- Afficher le premier élément de la liste :

```
print(ma_liste_de_points[0])
```

(0, 45.248952, 4.675639, 'Place Michelet')]

- Insérer des objets dans une liste :

```
ma_liste_de_points.append(7, 45.248069, 4.679126, 'rien')
print(ma_liste_de_points)
```

[(0, 45.248952, 4.675639, 'Place Michelet'), (1, 45.248733, 4.675899, 'rien'), (2, 45.248001, 4.676494, 'rien'), (3, 45.247654, 4.676781, 'rien'), (4, 45.24789, 4.677469, 'bus3'), (5, 45.247785, 4.678495, 'rien'), (6, 45.248069, 4.679126, 'rien'), (7, 45.248069, 4.679126, 'rien')]

- Créer une liste avec que les champs ***latitude*** et ***longitude*** :
  Afin de pouvoir utiliser la fonction Polyline du module folium, il faut utiliser une liste de liste avec uniquement la latitude et la longitude des points à relier. Il va falloir alors supprimer les champs id et commentaire. Pour cela, on peut créer une nouvelle liste ne contenant que ces deux champs.

```
liste_de_coordonnees = []
for point in ma_liste_de_points :
        #chaque "point" comporte l'id, latitude, longitude et commentaire
        #constituer la liste de coordonnées en prenant que les 2 premières coordonnées à la position 1 et 2 dans "point" :
        liste_coordonnees.append(point[1], point[2])
print(liste_de_coordonnees)
```

[(45.248952, 4.675639), (45.248733, 4.675899), (45.248001, 4.676494), (45.247654, 4.676781), (45.24789, 4.677469), (45.247785, 4.678495), (45.248069, 4.679126), (45.248922, 4.679126)]

  Cette liste peut-être alors utilisée dans la fonction Polyline du module folium…

- Récupérer le commentaire du dernier point de la liste de points afin de l'utiliser :

```
#Récupération du commentaire du dernier point de la liste de points :
dernier_point_comm = ma_liste_de_points[len(ma_liste_de_points)-1]
dernier_point_comm = dernier_point_comm[3]
```