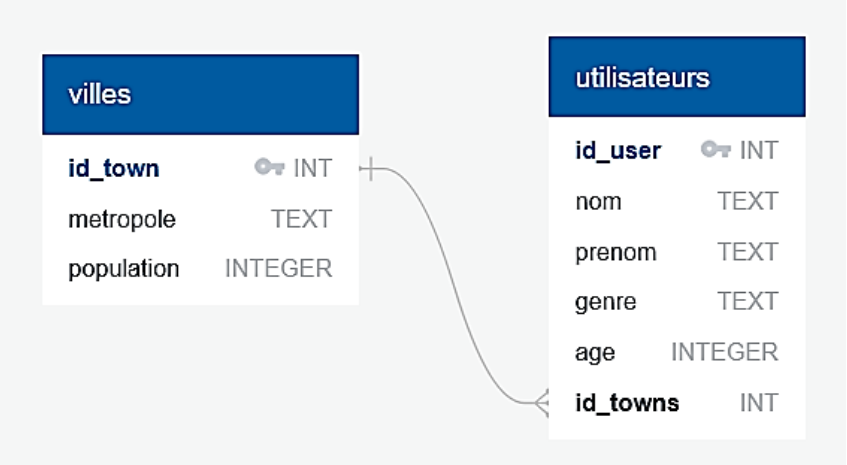
Tp_2 - Requêtes Sql avec sqlite3- python

Dans le Tp_1 précédent, nous avons créé les tables ***villes*** et ***utilisateurs*** ci-dessous. Nous les avons remplis et avons exécuté des requêtes SQL de sélection afin d'en exploiter les données.

Ce travail avait été réalisé par l'intermédiaire du logiciel *DB Browser* qui permet de créer une base de données, d'exécuter des requêtes Sql et de visualiser le contenu des tables.

On se propose à présent, dans ce Tp_2, de réaliser les mêmes opérations en exécutant les mêmes requêtes Sql cette fois-ci dans un code python. On utilisera ici uniquement *DB Browser*, pour vérifier l'état de la base de données créée.

⇨ Sur Pyzo, ouvrir un nouveau fichier nommé *freeBranlyPython.py* .



1. CREATION DES RELATION *VILLES* ET *UTILISATEURS* :

⇨ Dans ce fichier, écrire et compléter le script python ci-dessous qui permet une fois complété, de créer les 2 tables *villes* et *utilisateurs*.

connexion est un objet lié à la base de données

Si le fichier n'existe pas, une base de données à ce nom est créée automatiquement

curseur est un objet de la classe sqlite3

commit() permet d'appliquer les modifications sur la bdd

Comme se script crée les tables, si on les crée une seconde fois, cela provoque des erreurs. Pour les prévenir, on supprime ici les tables qui seront recréées ensuite.

```
import sqlite3
connexion = sqlite3.connect("freeBranlyPython.db")
curseur = connexion.cursor()
curseur.execute("PRAGMA foreign_keys = ON") # Active les clés étrangères

# Suppression des relations table et utilisateur si elles existent
curseur.executescript("""
                      DROP TABLE IF EXISTS utilisateurs   ;
                      DROP TABLE IF EXISTS villes   ;
                      """)
connexion.commit()

# Création de la table villes
curseur.executescript("""
                      CREATE TABLE villes  (
                                            id_town INTEGER PRIMARY KEY,
                                            metropole TEXT,
                                            population INTEGER
                                           ) ;
                      """)
connexion.commit()

# Création de la table utilisateurs



# Fermeture du lien python - fichier
connexion.close()
```

2. REMPLISSAGE DES RELATION *VILLES* ET *UTILISATEURS* :

⇨ Compléter encore ce fichier, afin d'insérer les enregistrements ci-dessous dans les tables ***villes*** et ***utilisateurs*** .

Table : villes

	id_town	metropole	population
	Filtre	Filtre	Filtre
1	1	LYON	513000
2	2	PARIS	2161000
3	3	MARSEILLE	861000
4	4	MACON	33000

Table : utilisateurs

	id_user	nom	prenom	genre	age	id_towns
	Filtre	Filtre	Filtre	Filtre	Filtre	Filtre
1	1	Chouhan	Jean	homme	50	1
2	2	Durand	Louis	homme	83	1
3	3	GranJean	Alice	femme	45	2
4	4	Bobet	Louison	homme	27	3
5	5	Champin	Arnaud	homme	23	1

⇨ Vérifier avec le logiciel *DB Browser* que le contenu des tables est bien celui-donné ci-dessus.

A partir de ce point, on n'utilisera plus DB Browser et on pourra donc fermer ce logiciel. Toutes les requêtes de lecture données dans la suite seront réalisées par votre script python.

3. REQUETES DE LECTURE :

Pour récupérer dans une liste tous les enregistrements de la table villes, on exécute la requête Sql suivante :

On exécute la méthode *execute()* sur l'objet *curseur*

```
# Exécution des requêtes de sélection
curseur.execute("""
                    SELECT * FROM villes;
                """)
resultat = curseur.fetchall()
```

On exécute la méthode *fetchall()* sur l'objet *curseur*

La variable ***resultat*** qui est retournée par la méthode *fetchall()* est une liste de ***tuples*** :

```
[(1, 'LYON', 513000), (2, 'PARIS', 2161000), (3, 'MARSEILLE', 861000), (4, 'MACON', 33000)]
```

Pour lire par exemple la population de Paris, on peut exécuter dans la console :

```
>>> resultat[1][2]
2161000
```

Ainsi, si on demande tous les attributs de tous les enregistrements de *villes* , on obtient la liste :

	id_town	metropole	population
1	1	LYON	513000
2	2	PARIS	2161000
3	3	MARSEILLE	861000
4	4	MACON	33000

Liste requête 1.

```
[(1, 'LYON', 513000), (2, 'PARIS', 2161000), (3, 'MARSEILLE', 861000), (4, 'MACON', 33000)]
```

Les copies d'écrans demandées dans la suite seront insérées directement dans ce fichier .docx .

⇨ Donner ci-dessous une copie d'écran de la liste obtenue si on demande le nom des villes dont la population est supérieure à 500 000 habitants :

	metropole
1	LYON
2	PARIS
3	MARSEILLE

Liste requête 2.

⇨ Donner une copie d'écran de la liste obtenue si on demande le nom des villes qui ont un « R » dans ce nom :

	metropole
1	PARIS
2	MARSEILLE

Liste requête 3.

⇨ Donner une copie d'écran de la liste obtenue si on demande le prénom et l'âge des utilisateurs de sexe masculin :

	prenom	age
1	Jean	50
2	Louis	83
3	Louison	27
4	Arnaud	23

Liste requête 4.

⇨ Donner une copie d'écran de la liste obtenue si on demande la somme de tous les âges.

	somme_age
1	228

Liste requête 5.

⇨ Donner une copie d'écran de la liste obtenue si on demande le nombre d'enregistrements de la table *utilisateurs*

	nb
1	5

Liste requête 6.

⇨ Donner une copie d'écran de la liste obtenue si on demande la moyenne des âges des utilisateurs

	ageMoyen
1	45.6

Liste requête 7.

⇨ Donner une copie d'écran de la liste obtenue si on demande l'âge maximum des utilisateurs :

	age_maxi
1	83

Liste requête 8.

⇨ Donner une copie d'écran de la liste obtenue si on désire demande les noms et prénoms des clients qui habitent LYON.

	nom	prenom
1	Chouhan	Jean
2	Durand	Louis
3	Champin	Arnaud

Liste requête 10.

⇨ Donner une copie d'écran de la liste obtenue si on demande les nom, prenom et nombre d'habitants de la ville dans laquelle vit Durand :

	nom	prenom	population
1	Durand	Louis	513000

Liste requête 11.

⇨ Donner une copie d'écran de la liste obtenue si on demande les nom, prénom et ville des utilisateurs qui vivent dans une ville de plus de 1 000 000 habitants :

	nom	prenom	metropole
1	GranJean	Alice	PARIS

Liste requête 12.

⇨ Donner une copie d'écran de la liste obtenue si on demande tous les attributs de la table *villes* avec en plus les attributs des *utilisateurs*, pour tous les enregistrements de *villes* :

	id_town	metropole	population	id_user	nom	prenom	genre	age	id_towns
1	1	LYON	513000	5	Champin	Arnaud	homme	23	1
2	1	LYON	513000	1	Chouhan	Jean	homme	50	1
3	1	LYON	513000	2	Durand	Louis	homme	83	1
4	2	PARIS	2161000	3	GranJean	Alice	femme	45	2
5	3	MARSEILLE	861000	4	Bobet	Louison	homme	27	3
6	4	MACON	33000	NULL	NULL	NULL	NULL	NULL	NULL

Liste requête 13.

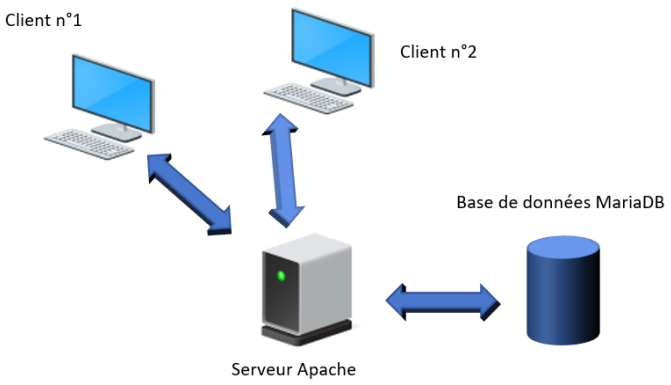
⇨ Donner une copie d'écran de la liste obtenue si on demande tous les attributs de la table villes avec en plus les attributs des utilisateurs, pour seulement les enregistrements de *villes* répertoriés dans *utilisateurs* :

	id_town	metropole	population	id_user	nom	prenom	genre	age	id_towns
1	1	LYON	513000	1	Chouhan	Jean	homme	50	1
2	1	LYON	513000	2	Durand	Louis	homme	83	1
3	2	PARIS	2161000	3	GranJean	Alice	femme	45	2
4	3	MARSEILLE	861000	4	Bobet	Louison	homme	27	3
5	1	LYON	513000	5	Champin	Arnaud	homme	23	1

Requête 14.

4. CONCLUSION :

On a vu dans cette activité qu'il était possible de gérer une bdd directement avec un code python. C'est exactement ce qui se passe sur un serveur qui héberge un site internet. Ces serveurs peuvent être gérés en python ou le plus souvent avec d'autres langages (php, javascript, ruby, …). Les requêtes sql sont construites en fonction des demandes qui émanent du client. La base de données hébergée sur le serveur retourne alors rapidement les informations demandées pour construire ensuite une réponse (fichier html ou code JS) qui est aussitôt envoyée au client.



⇨ N'oubliez pas d'uploader les fichiers *freeBranlyPython.py et freeBranlyPython.docx* sur nsibranly.fr .