TP 14 : Interaction entre bases de données et Python

Dans ce TP, on va voir comment manipuler une base de données en utilisant Python, ce qui permettra d'automatiser via des boucles l'exécution de requêtes, pour extraire des données rapidement. On verra ensuite comment créer des bases de données.

1 Interaction d'une base de données avec Python

Vous trouverez sur le site web un script permettant un dialogue interactif avec la base de données, voici quelques explications. Le module permettant d'intégrer un SGBD à un environnement Python s'appelle `sqlite3` ; une fois ce dernier importé, on se connecte à une base de données par l'intermédiaire de la fonction `connect`, en précisant en paramètre un chemin d'accès vers la base de données. Par exemple, pour utiliser la base de données `communes_francaises.sqlite` du dernier TP on écrira :

```
import sqlite3 as sql
conn=sql.connect("communes_francaises.sqlite") #Changer le chemin ou utiliser os.chdir
```

L'objet `conn` est désormais en place, et vous allez pouvoir dialoguer avec lui à l'aide du langage SQL. On va d'abord mettre en place ce que l'on appelle un curseur. Il s'agit d'une sorte de tampon mémoire intermédiaire, destiné à mémoriser temporairement les données en cours de traitement, ainsi que les opérations que vous effectuez sur elles, avant leur transfert définitif dans la base de données. Cette technique permet donc d'annuler si nécessaire une ou plusieurs opérations qui se seraient révélées inadéquates (dans le cas où on modifie la base de données), et de revenir en arrière dans le traitement, sans que la base de données n'en soit affectée.

```
cur=conn.cursor()
```

Une fois le curseur créé, la méthode *execute* du curseur permet de transmettre des requêtes rédigées en SQL sous forme de chaîne de caractères :

```
cur.execute("SELECT * FROM communes WHERE dep_id=6 ORDER BY population DESC")
```

Dans le cas où l'on effectue des requêtes d'extraction de données dans la base (`SELECT ...`), on peut parcourir le curseur comme un itérable (et par exemple afficher les résultats)

```
for resultat in cur:
    print(resultat)
```

Ceci va vider le curseur, et l'on peut récupérer les résultats de la requête. Si l'on avait effectué plusieurs requêtes avant de « vider » le curseur, les résultats se seraient « enfilés » dans le curseur.

Enfin, si des modifications ont été effectuées sur la BDD (ce qu'on verra plus tard), il faut appliquer la méthode commit à la connexion créée pour qu'elles deviennent définitives. On peut ensuite refermer le curseur et la connexion :

```
conn.commit() #utile si une modification a eu lieu
cur.close()
conn.close()
```

Le code que vous trouverez sur le site web est le suivant :

```
import sqlite3 as sql
conn=sql.connect("communes_francaises.sqlite") #changer le répertoire
cur=conn.cursor()

while True:
    requete=input("Entrez une requête SQL ou STOP pour arrêter : ")
    if requete=="STOP":
        break
```

```
    try:
        cur.execute(requete)
        for result in cur:
            print(result)
    except:
        print("Requête incorrecte")

cur.close()
conn.close()
```

Quelques explications/rappels :

- `input` permet de récupérer ce que l'utilisateur tape au clavier, et renvoie le résultat comme une chaîne de caractères ;
- `break` permet d'interrompre une boucle (ici on sort donc de la boucle `while` uniquement si « STOP » est tapé au clavier) ;
- On peut utiliser `try` et `except` pour qu'un code puisse se poursuivre même s'il y a des erreurs : si vous tapez une requête syntaxiquement incorrecte, une erreur se produit (non visible) et le bloc `except` est exécuté. S'il n'y a pas d'erreur lors de l'évaluation du bloc `try`, le bloc `except` n'est pas exécuté.

Rappelons la structure de la base de données `communes_francaises.sqlite`.
Pour la table depts :

- id (un entier) la clé primaire de l'enregistrement ;
- code (une chaîne de caractères) le code administratif du département ;
- nom (une chaîne de caractères) le nom du département ;
- chef_lieu (un entier) l'identifiant du chef-lieu du département (il référence la clé primaire de la table `communes`).

Pour la table communes :

- id (un entier) la clé primaire de l'enregistrement ;
- code (une chaîne de caractères) le code administratif de la commune ;
- postal (une chaîne de caractères) le code postal de la commune ;
- nom (une chaîne de caractères) le nom de la commune ;
- superficie (un flottant) la superficie de la commune, en hectares ;
- altitude (un entier) l'altitude de la commune, en mètres ;
- population (un flottant) la population d'une commune, en millier d'habitants ;
- dep_id (un entier) l'identifiant du département où se trouve la commune (référence la clé primaire de la table `depts`)

Question 1. Utilisez le script interactif pour effectuer quelques requêtes de recherche du TP précédent, par exemple :

- afficher toutes les entrées de la table `depts` ;
- donner le nombre d'entrées de la table `communes` ;
- donner pour chaque département le nombre d'habitants (utiliser un GROUP BY) ;
- afficher la liste des chefs-lieu de France (utiliser une jointure entre les champs `chef_lieu` de `depts` et `id` de `communes`).

Vous pouvez refaire d'abord ces requêtes avec SQLite Studio si vous n'êtes pas sûrs de vous.

2 Génération de requêtes par des fonctions Python

L'intérêt majeur de l'utilisation du module `sqlite3` est de pouvoir écrire des fonctions, qui vont prendre en entrée un certain paramètre, générer une requête dépendant du paramètre, exécuter la requête, récupérer le résultat, et le renvoyer : l'utilisateur peut donc faire usage des fonctions pour récupérer des données dans la base, sans écrire la moidre requête ! Bien sûr, c'est nous qui allons écrire ces fonctions. On rappelle que les requêtes doivent être fournies à la méthode execute sous forme de chaînes de caractères. On fera donc usage de :

- `str(x)` pour convertir `x` (un entier, flottant...) en chaîne de caractères ;
- `+` pour concaténer des chaînes de caractères.

Les questions qui suivent demandent d'écrire des requêtes qui produisent un résultat, avec un seul attribut (sauf à la question 6). Il est facile de récupérer le résultat en question à partir du curseur :

— une fois la requête effectuée, `L=list(cur)` permet d'obtenir une liste à 1 seul élément (elle en aurait contenu plusieurs si la requête fournissait plusieurs résultats) ;

— `L[0]` est cet élément : c'est un tuple à un seul élément (si la requête produisait bien un résultat à un seul attribut) ;

— `L[0][0]` est donc ce résultat. On pourra utiliser `int` pour convertir en retour ce résultat en entier...

Question 2. Écrire une fonction `chef_lieu(c)` prenant en entrée une chaîne de caractères `c` et renvoyant le nom de la commune qui est le chef lieu du département dont le code administratif est `c`. (On affichera une erreur si `c` ne correspond à aucun code administratif).

```
>>> chef_lieu("83")
'TOULON'
>>> chef_lieu("06")
'NICE'
>>> chef_lieu("2A")
'AJACCIO'
>>> chef_lieu("979")
code erroné
```

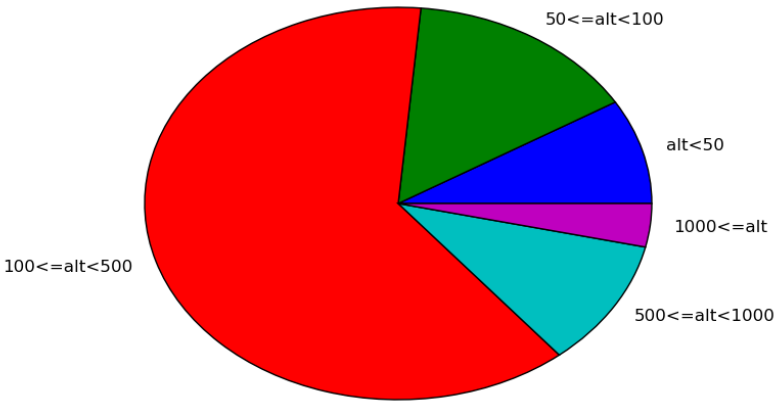
Attention : la requête à exécuter doit contenir des guillemets autour du code de département. On rappelle l'on peut faire usage de guillemets simples et doubles en SQLite comme en Python. Avant d'écrire la fonction, on cherchera à écrire correctement une requête récupérant le bon résultat.

Question 3. Écrire de même une fonction `nombre_communes_dep(c)` prenant en entrée un code administratif et renvoyant le nombre de communes (c'est un entier) dans le département de code `c`, affichant une erreur sinon.

```
>>> nombre_communes_dep("06")
163
>>> nombre_communes_dep("2B")
236
```

Question 4. Écrire une fonction `nombre_communes_alt(mini,maxi)`, prenant en entrée deux entiers et retournant le nombre de communes à une altitude située entre `mini` inclus et `maxi` exclus.

```
>>> nombre_communes_alt(100,400)
20968
>>> nombre_communes_alt(0,10000)
36613
>>> nombre_communes_alt(100,100)
0
```



Question 5. *Camembert.* La fonction pie de `matplotlib.pyplot` permet le tracé d'un diagramme circulaire (« camembert » en français, « tarte » en anglais). Consulter l'aide (`help(plt.pie)` avec `matplotlib.pyplot` importé sous le nom `plt` comme d'habitude) associée à cette fonction, puis tracer un diagramme circulaire dans lequel seront représentés :

— le nombre de communes situées à une altitude inférieure à 50 m ;

— le nombre de communes situées entre 50 et 100 m d'altitude ;

— le nombre de communes situées entre 100 et 500 m d'altitude ;

— le nombre de communes situées entre 500 et 1000 m d'altitude ;

— le nombre de communes situées à plus de 1000 m d'altitude. (On rappelle que l'Everest culmine à 8848 mètres.)

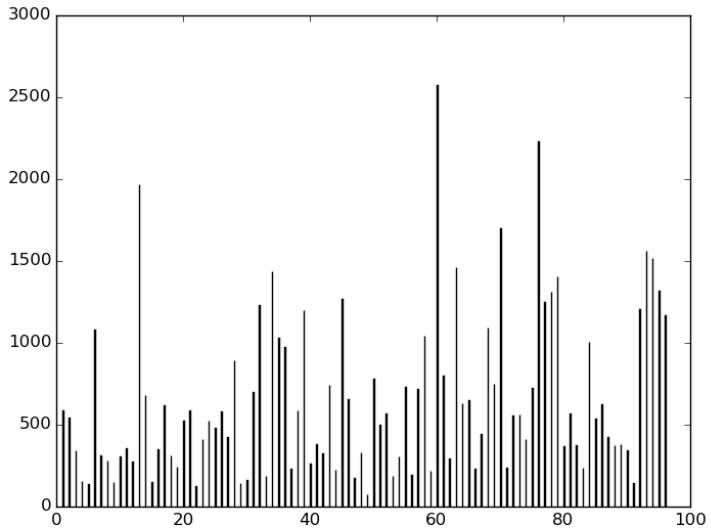
On utilisera bien sûr la fonction précédente.

Question 6. *Histogramme.* De même, la fonction `plt.bar` permet de réaliser un histogramme (diagramme en « rectangles »). Il s'utilise par exemple comme suit :

```
x = [1,2,3,4,5,6,7,8,9,10]
hauteurs_barres = [8,12,8,5,4,3,2,1,0,0]
largeur_barres = 0.1
plt.bar(x, hauteurs_barres, largeur_barres)
plt.show()
```

(Évidemment, `x` et `hauteurs_barres` doivent être deux listes de même taille)

On peut convertir le résultat d'une requête en liste (`cur.execute(requete)` puis `list(cur)`). Le résultat est une liste de tuples, chaque tuple étant associé à une ligne du résultat de la requête. Les types des éléments du tuple sont les mêmes que les attributs de la table associée au résultat de la requête. En exécutant une seule requête, réalisez un histogramme représentant les populations des différents départements par milliers d'habitants.



3 D'une base de données à un fichier

Question 7. Écrire dans un fichier `departements.txt` le nom de tous les départements de la table depts précédé de leur code administratif, et suivi de leur population en milliers d'habitants (un département par ligne). La première ligne est : `01, AIN, 588.8000000000008`. (On pourra aussi utiliser `round(x,1)` pour couper au dixième et obtenir plutôt 588.8.) On rappelle ici les fonctions principales pour travailler avec un fichier dans Python :

fonction	desciption
`f=open(nom_du_fichier,'r')`	Ouvre le fichier `nom_de_fichier` (donné sous la forme d'une chaîne de caractères indiquant son emplacement) en lecture (`r` comme read). Le fichier doit exister et seule la lecture est autorisée.
`f=open(nom_du_fichier,'w')`	Ouvre le fichier `nom_de_fichier` en écriture (`w` comme write). Si le fichier n'existe pas, il est créé, sinon il est écrasé (vidé avant utilisation).
`f=open(nom_du_fichier,'a')`	Ouvre le fichier `nom_de_fichier` en ajout (`a` comme append). Identique au mode `'w'`, sauf que si le fichier existe, il n'est pas écrasé et ce qu'on écrit est ajouté à partir de la fin du fichier.
`f.close()`	Sur un fichier ouvert comme précédemment, le ferme. Cette ligne est impérative pour les fichiers ouverts en écriture, puisque le fichier n'est réellement écrit complètement qu'à la fermeture.
`f.readlines()`	Lit tout le fichier et stocke le résultat dans une liste de chaînes de caractères, chaque élément correspondant à une ligne. Attention, le saut de ligne `\n` est présent à la fin de chaque chaîne.
`f.write(s)`	Écrit la chaîne `s` à la suite du fichier.

4 Création d'une base de données à une seule table (à partir d'un fichier)

Nous allons voir dans cette section comment créer une base de données, et insérer des valeurs dedans. Fermez la table `communes_francaises`.
Voici un exemple de script pour demander la création d'une nouvelle table, qui est sensée gérer les membres d'une association (le code est téléchargeable comme `exemple_code.py`) :

```
conn=sql.connect("assoc.sqlite")
cur=conn.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS membres (id INTEGER PRIMARY KEY AUTOINCREMENT, age INTEGER,
nom TEXT, taille REAL)")

cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'DUPOND',1.83)")
cur.execute("INSERT INTO membres(age,nom,taille) VALUES(15,'DURAND',1.57)")
cur.execute("INSERT INTO membres(age,nom,taille) VALUES(18,'DUGOMMIER',1.69)")

conn.commit()
cur.close()
conn.close()
```

Quelques explications :

- On crée une table « membres » à 4 attributs. Notez le champ « id » de type nombre entier qui sert de clef primaire : ceci est spécifié, et ce champ est auto-incrémenté, ce qui signifie que lorsqu'on rentre de nouveaux membres dans la base, on n'a pas à préciser nous-même le champ `id`.
- Le "IF NOT EXISTS" est optionnel mais conseillé car il évite un message d'erreur au cas ou la TABLE existe déjà.
- Dans la rédaction, on fait suivre CREATE TABLE du nom de la table, puis entre parenthèses les noms de chaque champs suivi du type de données et séparés par des virgules.
- L'insertion de membres dans la base se fait de lui-même, à l'aide de INSERT INTO. L'exemple parle de lui-même.
- Après l'insertion, les enregistrements sont dans le tampon du curseur, mais ils n'ont pas encore été transférés véritablement dans la base de données. On pourrait donc tout annuler si nécessaire. Le transfert dans la base de données est déclenché par la méthode `commit()` de l'objet `conn`.
- On peut refermer le curseur et la connexion ensuite.

Question 8. Récupérez le fichier `membres_associations.txt` et insérez les nouvelles entrées dans la base à partir d'un script Python. Vous pouvez laisser Dupont, Durand, Dugommier. On rappelle que `s.split(',')` permet de séparer une chaîne de caractères `s` autour du caractère `,` : une liste est renvoyée.

Question 9. Une fois la base créée, récupérer les informations suivantes :

- nombre d'entrées dans la base ?
- moyenne d'âge ?

— taille moyenne ?

Voici les principales instructions utilisables pour manipuler une BDD :

Instruction	Effet produit
`conn = sqlite3.connect(fichierDonnees)`	Ouvre conn qui permet d'accéder à la Bdd
`cur =conn.cursor()`	Crée un curseur cur qui agit sur la Bdd et permet de récupérer les résultats des requêtes
`conn.commit()`	Exécute toutes les modifs sur la Bdd
`cur.close()`	Ferme le curseur `cur`
`conn.close()`	Ferme la connexion `conn`
`cur.execute("CREATE TABLE membres ( age INTEGER, nom TEXT, taille REAL)")`	Crée une table en indiquant les noms et types des champs
`cur.execute("CREATE TABLE IF NOT EXISTS membres (age INTEGER, nom TEXT, taille REAL)")`	Idem mais teste si cette table existe
`cur.execute("INSERT INTO membres(age,nom,taille) VALUES(21,'Dupont',1.83)")`	Insère dans la table et dans les champs cités les données suivantes
`cur.execute("SELECT * FROM membres")`	Un exemple de `SELECT`
`cur.execute("UPDATE membres set nom ='Gerart' WHERE nom='Ricard'")`	On remplace le nom `'Ricard'` de la table membre par le nom `'Gerart'`
`cur.execute("DELETE FROM membres WHERE nom='Machin'")`	Supprime l'enregistrement vérifiant le critère
`cur.execute("DROP TABLE nom_de_la_table")`	On supprime la table `nom_de_la_table`

Question 10. Modifiez la base en suivant les instructions suivantes :

— DUMAS a quitté l'association. Supprimez-le de la base.

— LUZ n'a pas 43 ans, mais 28. Mettez l'information à jour.

5 Création d'une vraie base

Nous allons créer une base de données, concernant des élèves de l'université à qui certains cours sont proposés. Les tables et leurs attributs sont les suivants.

- Table etudiant :
  - id : un entier (clé primaire) ;
  - nom : une chaîne de caractères ;
  - prenom : une chaîne de caractères ;
- Table cours :
  - id : un entier (clé primaire) ;
  - nom : une chaîne de caractères ;
- Table cours_suivi :
  - id_etu : un entier (clé étrangère) ;
  - id_cours : un entier (clé étrangère) ;

  On impose que le couple (id_etu, id_cours) forme une clé primaire.

Question 11. Créez la base de données `etu.sqlite` (il suffit de créer une connexion vers `etu.sqlite`)

Question 12. Après avoir ouvert une connexion `conn` et un curseur `cur`, créez les tables etudiant et cours. (On ne précisera pas AUTOINCREMENT pour les clés primaires, mais on précisera qu'elles sont des clés primaires). Remplissez-les à l'aide des deux fichiers `etudiant.txt` et `cours.txt`.

Question 13. Voici comment créer la table `cours_suivi` :

```
cur.execute("CREATE TABLE IF NOT EXISTS cours_suivi (id_etu INTEGER, id_cours INTEGER,
  PRIMARY KEY (id_etu,id_cours), FOREIGN KEY(id_etu) REFERENCES etudiant(id),
 FOREIGN KEY(id_cours) REFERENCES cours(id))")
```

Voici des explications : on indique que id_etu et id_cours sont des clés étrangères vers les tables en question, et on impose de plus que le couple composé des deux clés forme une clé primaire [1]. Remplissez la table à l'aide du fichier `cours_suivi.txt`.

1. En l'absence d'autres attributs, cela paraît un peu inutile car il ne peut y avoir de doublons dans la base. Néanmoins cela prendrait tout son sens si on rajoutait un champ comme la note obtenue sur ce cours, par exemple.