
INTERACTION ENTRE UNE BASE DE DONNÉES ET UN PROGRAMME

Objectifs

- Savoir manipuler une base de données à partir d'un programme Python ;
- Savoir manipuler les objets fournis par une bibliothèque.

0 Bibliothèque SQLite3

On peut lire dans documentation officiel de Python¹ la description suivante :

« *SQLite is a C library that provides a lightweight disk-based database that doesn't require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It's also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.* »

Une base **SQLite3** a donc la particularité de ne pas être basée sur un modèle client-serveur, contrairement aux SGBD usuels ; elle est contenue dans un fichier unique. La bibliothèque Python du même nom va nous permettre de créer une interface pour interagir avec les base de données de ce format.

On donne ci-dessous un exemple d'utilisation de la bibliothèque, extrait de la documentation Python, illustrant la manipulation d'une base de donnée **SQLite3** à partir d'un script Python.

```
# Create and fill the table.
con = sqlite3.connect(":memory:")
con.execute("CREATE TABLE lang(name STRING, first_appeared INT)")
data = [
    ("C++", 1985),
    ("Objective-C", 1984),
]
con.executemany("INSERT INTO lang(name, first_appeared) VALUES(?, ?)", data)

# Print the table contents
for row in con.execute("SELECT name, first_appeared FROM lang"):
    print(row)

print("I just deleted", con.execute("DELETE FROM lang").rowcount, "rows")

# Execute the SQL statements in sql_script. sql_script must be a string.
con.executescript("""
    INSERT INTO lang VALUES ("C++", 1985);
    INSERT INTO lang VALUES ("Objective-C", 1984);
""")

# close() is not a shortcut method and it's not called automatically;
# the connection object should be closed manually
con.close()
```

1 Réalisation d'une interface avec Python

On se propose de créer une interface Python capable de gérer les interactions avec la base de données de la médiathèque introduite dans le cours/TP. Elle devra pouvoir répondre aux besoins usuels d'une telle structure comme :

- ajouter ou retirer un livre, un usager ou un emprunt de la base ;
- faire une recherche textuelle sur le titre ou l'auteur des livres de la base ;
- afficher le titre des livres en cours d'emprunt ainsi que le nom des usagers les ayant empruntés ;
- envoyer un courriel de rappel à tous les usagers dont l'emprunt se termine dans moins de trois jours ;
- mettre à jour le numéro de carte d'un usager.

1. <https://docs.python.org/3/library/sqlite3.html>

1.0 Création de la base de données

Pour créer la base de données initiale de la médiathèque, nous allons lire le contenu du fichier `mediatheque.sql`, contenant tous les ordres SQL nécessaires à cet effet, et le stocker dans une chaîne de caractère avec les instructions suivantes :

```
f = open('mediatheque.sql', encoding="utf-8")
ordres_sql = f.read()
f.close()
```

Travail à faire

1. À partir de l'exemple donné ci-dessus, écris une fonction `initialisation_bd()` qui crée une base de données nommée `mediatheque.db` et qui exécute ensuite tous les ordres SQL contenus dans le fichier `mediatheque.sql` pour créer les tables et y insérer les données. (la durée d'exécution peut être relativement longue selon les performances de la machine utilisée...)
2. Vérifie le contenu de la base de données en affichant le titre de tous les livres de la base.

1.1 Création de l'interface

Pour interagir avec la base de données précédemment créée, nous allons définir une classe `Mediatheque` qui possède un attribut `con` qui stockera la référence de l'objet de type `sqlite3.Connection` renvoyé lors de la connexion à la base de données, et qui sera initialisé à `None`.

Travail à faire

1. Définis la classe `Mediatheque` telle que décrite en introduction.
2. Ajoute les méthodes `connexion` et `deconnexion` qui auront la responsabilité de respectivement connecter et déconnecter la base.
 - On s'assurera, dans chaque cas, à mettre à jour la référence de l'attribut `con`.
 - Enfin, on affectera la valeur `True` à l'attribut `con.autocommit` pour auto-valider toutes les insertions dans les tables qui seront effectuées a posteriori.
3. Ajoute une méthode `rechercher_livre(m)` prenant en argument une chaîne de caractères `m` et qui renverra la liste des livres de la base contenant cette chaîne dans son titre.
 - On appellera la méthode `fetchall()` sur l'objet renvoyé par l'appel de `con.execute()` pour récupérer la liste des enregistrements.
 - Si l'on passe en argument une chaîne vide, la méthode doit renvoyer la liste de tous les livres enregistrés dans la base.
4. Ajoute une méthode `ajouter_livre(l)` prenant en argument un tuple `l` contenant toutes les valeurs de l'enregistrement à ajouter dans la table `livre` et qui l'y insérera.
5. Ajoute une méthode `retirer_livre(isbn)` prenant en argument l'isbn du livre à retirer de la table `livre` et le supprimera.
6. En utilisant la méthode `rechercher_livre(m)`, écris un test vérifiant avec l'instruction `assert` que l'ajout puis le retrait d'un livre dans la base laisse le nombre de livres inchangé.
7. De la même manière que dans les questions précédentes, ajoute de nouvelles méthodes permettant de réaliser tous les besoins listés en introduction de la partie 1.