UAA12 – Manipuler les Bases de données en python et flask

1 TP1 : Manipuler les bases de données avec python

1.1 Test1 : Prise en main

1. Dans ton dossier uaa12_web, crée un **uaa12_python_flask_sqlite**, puis un sous-dossier **tp1_python_sqlite_test1**
2. Dans ce dossier, crée un fichier **test1-1_create.py** et suis le tuto **[1-AppPy]** Apprendre Python **p279-284** « Création de la base de données - Objets « connexion » et « curseur »

Remarque :

Que se passe-t-il si tu exécutes plusieurs fois ton programme ?
Pour pouvoir lancer indéfiniment ton script, utilise l'instruction try except :

```
try:
 cur.execute("CREATE TABLE membres(age INTEGER, nom TEXT, taille REAL)")
except:
 pass
```

Code test1-1_create.py

```
# Création de la bd - objets connexion et curseur p282-283
# Initialisation
##########################################################################
import sqlite3
fdb="bd_test.db"    # fichier de la bd
                # SQLite mémorise toutes les tables d'une BD dans un seul fichier que l'on paut sauvegarder où on veut
oconn=sqlite3.connect(fdb)  #objet-connexion: interface entre le prog et la bd
                # connexion au fichier de BD (et création s'il n'existe pas encore)
cur=oconn.cursor()  # curseur: mémoire tampon pour dialoguer avec la BD.
                # les données en cours de traitement et les opérations effectuées sont mémorisées temporairement
                # avant leur transfert définitif ds la BD
```

```
                # Cela permet d'annuler si nécessaire des opérations inédéquates et de revenir en arrière


# Requêtes SQL
#############################################################################
# Création d'une nouvelle table
try:
 cur.execute("CREATE TABLE membres(age INTEGER, nom TEXT, taille REAL)")

 # Entrer les enregistrements:
 cur.execute("INSERT INTO membres VALUES(21, 'Dupont', 1.83)")
 cur.execute("INSERT INTO membres VALUES(15, 'Blumar', 1.57)")
 cur.execute("INSERT INTO membres VALUES(18, 'Ozemir', 1.69)")
except:
 pass

# Syntaxe générale: cur.execute("requete sql")
# Attention, execute() crée les données dans la mémore tampon cursor() mais pas ds la bd
# (ce qui permet de l'annuler si besoin)

# Sauvegarde et fermeture
#############################################################################
oconn.commit()      # Le transfert dans la base de données est fait par commit()
cur.close()         # fermeture du curseur
oconn.close         # fermeture de la connexion
```

3. Dans ce même dossier, crée un fichier **test1-2_connexion.py** et suis le tuto **[1-AppPy] p284-286** « Connexion à une base de données existante »

Code test1-2_connexion.py

```
#############################################################################
# Connexion à une bd existante p284
```

```
# Initialisation
import sqlite3
oconn=sqlite3.connect("bd_test.db")
cur=oconn.cursor()

# Requêtes SQL
print("Série de tupples")
cur.execute("SELECT * FROM membres")    # sélection des enregistrements
for l in cur:                           # affichage des enregistrements
      print(l)                          # série de tuples
print("curseur ap un select *",cur)

print("Liste de  tupples")
cur.execute("SELECT * FROM membres")
print(list(cur))

print("Liste de tupples avec fetchall()")
cur.execute("SELECT * FROM membres")
print(cur.fetchall())

print("Insertion de données par des var python")
cur.execute("INSERT INTO membres VALUES(18, 'Ricard', 1.75)")

data=[(17,"Durand",1.74),(22,"Berger",1.71),(20,"Weber",1.65)]
for tu in data:
      print("tu: ",tu)
      cur.execute("INSERT INTO membres VALUES(?,?,?)",tu) # insertion avec formatage de la requête par execute()

print("Modif un enregistrement")
cur.execute("UPDATE membres SET nom='Gerart' WHERE nom ='Ricard'")
cur.execute("SELECT * FROM membres")
for l in cur:
```

```
    print(l)


print("Supprimer un enr")
cur.execute("DELETE FROM membres WHERE nom='Gerart'")
cur.execute("SELECT * FROM membres")
for l in cur:
    print(l)


# Sauvegarde et fermeture
oconn.commit()
cur.close()
oconn.close
```

4. Dans ce même dossier, crée un fichier **test1-3_exo16-1_create.py** et fais l'exercice **[1-AppPy] exercice 16.1 p286**

1.2 Test2 : Filmographie

1. Dans **uaa12_python_flask_sqlite**, crée un dossier **tp1_python_sqlite_test2**
2. Reprends les consignes et ton code sql du « **TP3 – filmographie » de l'UAA7 BD**

Tables des films

f_id	titre	realisateur	annee
1	Les évadés	Darabont	1994
2	Le parrain	Coppola	1972
3	Le parrain 2	Coppola	1974
4	L'odyssée de Pi	Ang Lee	2013
5	Chocolat	Hallstrom	2000
6	Scarface	De Palma	1983
7	Rango	Verbinski	2011

Tables des acteurs

a_id	nom
1	Johnny Deep
2	Al Pacino
3	Suraj Sharma
4	Brad Pitt
5	Edward Norton

Tables des filmographies

fg_fk_acteur_id	fg_fk_film_id	role	salaire
1	5	Roux	5000
1	7	Rango	10000
2	2	Michael Corleone	10000
2	3	Michael Corleone	20000
2	6	Tony Montana	15000
3	4	Pi	20000

3. Crée 2 fichiers **test2_create_insert.py** et **test2_setup.py**:
   - ✓ **test2_create_insert.py** contient les créations des tables et les insertions des données. Teste-le en essayant de lire les données par le terminal sql.
   - ✓ **test2_setup.py** contient la lecture du contenu de chaque table et des 3 jointes

Résultat dans le terminal

```
acteurs       filmographie  films
a_id                  nom
--------------------  --------------------
1                     Johnny Deep
2                     Al Pacino
3                     Suraj Sharma
4                     Brad Pitt
5                     Edward Norton
f_id                  titre                 realisateur           annee
--------------------  --------------------  --------------------  --------------------
1                     Les évadés            Darabont              1994
2                     Le parrain            Coppola               1972
3                     Le parrain 2          Coppola               1974
4                     L'odysséede Pi        Ang Lee               2013
5                     Chocolat              Hallstrom             2000
6                     Scarface              De Palma              1983
7                     Rango                 Verbinski             2011
fg_fk_acteur_id       fg_fk_film_id         role                  salaire
--------------------  --------------------  --------------------  --------------------
1                     5                     Roux                  5000
1                     7                     Rango                 10000
2                     2                     Michael Corleone      10000
2                     3                     Michael Corleone      20000
2                     6                     Tony Montana          15000
3                     4                     Pi                    20000
a_id                nom                 f_id                titre               realisateur         annee  fg_fk_acteur_id  fg_fk_film_id  role                salaire
------------------  ------------------  ------------------  ------------------  ------------------  -----  ---------------  -------------  ----------------  -------
1                   Johnny Deep         5                   Chocolat            Hallstrom           2000   1                5              Roux              5000
1                   Johnny Deep         7                   Rango               Verbinski           2011   1                7              Rango             10000
2                   Al Pacino           2                   Le parrain          Coppola             1972   2                2              Michael Corleone  10000
2                   Al Pacino           3                   Le parrain 2        Coppola             1974   2                3              Michael Corleone  20000
2                   Al Pacino           6                   Scarface            De Palma            1983   2                6              Tony Montana      15000
3                   Suraj Sharma        4                   L'odysséede Pi      Ang Lee             2013   3                4              Pi                20000
sqlite>
```

1.3 Test3 : Injection SQL

- ✓ Reprends l'exercice **test1-3_exo16-1_create.py** de **[1-AppPy] exercice 16.1 p286**. Renomme-le tp8_bd_3_exo16-1_injections_sql.py.

On a vu que l'on pouvait écrire des requêtes paramétrées grâce au caractère **?**, et c'est vraiment ainsi qu'il faut procéder ! Mais à la place d'écrire :

```
req ="INSERT INTO compositeurs (comp, a_naiss) VALUES (?, ?)"
```

on aurait pu être tenté d'écrire :

```
req = "INSERT INTO notes VALUES ('" + nom + "','" + aNais + "')"
```

ou

```
req = f"INSERT INTO compositeurs VALUES ('{nom}',{aNais})"
```

Cette 2 dernières façons de faire, en construisant notre requête SQL comme une chaîne de caractère classique ou une chaîne formatée, introduit une faille de sécurité critique pour notre application, qui devient vulnérable à ce que l'on appelle une **injection sql**.

- ✓ Simplifie ton pour ne garder que la création de la table compositeurs (nom et année de naissance), la boucle d'insertion et la lecture des données de la table.

Code simplifié

```
try:
    req ="CREATE TABLE compositeurs(comp TEXT, a_naiss INTEGER)"
    cur.execute(req)
except:
    pass


while True :
    nom = input("Nom du compositeur (<Enter> pour terminer) : ")
    if nom =='':
        break
    aNais = input("Année de naissance : ")
    req = "INSERT INTO notes VALUES ('" + nom + "','" + aNais + "')"
    print(req)
    cur.executescript(req)


cur.execute("SELECT * FROM compositeurs")
print(cur.fetchall())
```

- ✓ Teste-le avec les données suivantes et observe le contenu de la base de données :

Résultat

```
Nom du compositeur (<Enter> pour terminer) : roro', 2024); DROP TABLE compositeurs; --
Année de naissance : 3000
```

Explication

La requête formulée est :

```
INSERT INTO notes VALUES ('roro', 2024); DROP TABLE compositeurs; --',3000)
```

Dans un 1er temps, le couple ('roro', 2024) a été inséré.
Puis l'ordre a été donné de détruire la table
Le reste de code qui n'est pas correct est ignoré car – est le symbole du commentaire en sqlite.
Evidemment, ce code a été écrit pour être spécifiquement vulnérable à l'injection SQL. Il suffit de remplacer cur.executescript(req) par cur.execute() pour que le code reste fonctionnel et refuse l'injection (1 seule requête SQL peut être exécutée avec execute(req))