R308 : Consolidation de la programmation

Chapitre 5 : Python et les BDD :

1 Rappel : Lecture écriture de fichiers en python :

Le stockage d'information dans des fichiers est une opération très courante.
Il existe deux types de fichiers : fichier texte et fichier binaires
Pour des raisons de facilité de manipulation, nous n'étudierons que les fichiers textes.

1.1 Ouverture d'un fichier :

On utilisera la fonction open()

1. Les différents modes d'ouverture :

Caractère	Description
"r"	Lecture (par défaut)
"w"	Écriture (avec remise à zéro)
"x"	Création exclusive (erreur si fichier déjà existant)
"a"	Écriture (avec ajout à la fin)
"b"	Mode binaire
"t"	Mode texte (par défaut)
"+"	Mise à jour (lecture et écriture)

Mode par défaut : "tr"

2. Méthode d'ouverture de fichier :

```
open(file, mode= 'r', encoding=None)
```

Exemple :

```
fichier = open(nom_fichier, "tr", encoding="utf-8")
```

3. Chemin relatif ou absolu :

Le répertoire par défaut est celui du fichier répertoire courant
../ permet de remonter d'un niveau dans l'arborescence
/ définit le chemin absolue

4. La gestion des erreurs :

La gestion des erreurs se fera en utilisant la clause try-except
Il est possible d'utiliser des erreurs spécifiques :

```
    FileNotFoundError : le fichier n'existe pas ;
    FileExistsError   : le fichier existe déjà ;
    PermissionError   : le programme n'a pas les droits d'accès nécessaires sur le fichier ;
    IsADirectoryError : le fichier est un dossier.
```

5. Utilisation du mot clé with :

Le mot-clé "with" permet une gestion plus sécurisée de l'ouverture des fichier (fermeture en cas d'erreur)

```
      with open(nom_fichier, "r") as fichier:
            ...
```

1.2 Lectures des données d'un fichier texte :

Voici le fichier exemple :
animaux.txt

```
girafe;rose;2
tigre;rouge;3
singe;jaune;1
éléphant;bleu;4
```

1. Boucle for :

```
for ligne in fichier:
	print(ligne, end="")	// avec lecture du saut de ligne en fin de ligne
```

Affichage :

```
girafe;rose;2
tigre;rouge;3
singe;jaune;1
éléphant;bleu;4
```

2. Méthode readlines() :

La méthode readlines() retourne une liste contenant toutes les lignes du fichier

```
liste:list[str] = fichier.readlines()
print(liste)
```

Affichage :

```
['girafe;rose;2\n', 'tigre;rouge;3\n', 'singe;jaune;1\n', 'éléphant;bleu;4\n']
```

3. Méthodes read() :

La méthode read() retourne une chaîne de caractères contenant toutes les lignes du fichier

```
str_multi_lignes = fichier.read()
print(str_multi_lignes)		// avec lecture du saut de ligne en fin de ligne
```

Affichage (une seule chaîne de caractères !):

```
girafe;rose;2
tigre;rouge;3
singe;jaune;1
éléphant;bleu;4
```

4. Méthode readline() :

La méthode readline() permet de lire le fichier ligne par ligne

```
ligne = fichier.readline()
	while ligne != "":
		print(ligne, end="")
		ligne = fichier.readline()
```

Affichage :

```
girafe;rose;2
tigre;rouge;3
singe;jaune;1
éléphant;bleu;4
```

5. Écriture dans un fichier :

La méthode write() permet d'écrire dans un fichier.

```
fichier.write(f"poulet{SEPARATEUR}noir{SEPARATEUR}6\n")
```

1.3 Fermeture de fichier :

Attention, cette opération est indispensable lorsque le fichier n'est plus utilisé … sauf si vous utilisez la structure « with »

1.4 Rappel sur la méthode split() des chaînes de caractères :

La méthode split() permet de découper (diviser) une chaîne de caractères en fonction d'un séparateur.

```
SEPARATEUR: str = ';'

with open("fichier.txt") as fichier:
	print(fichier.__class__)
	for ligne in fichier:
		ligne = ligne.replace('\n','') # suppression du saut de ligne à la fin
		print(ligne)
		liste_champs:list[str] = ligne.split(SEPARATEUR)
		print(liste_champs)
```

Affichage :

```
girafe;rose;2
['girafe', 'rose', '2']
tigre;rouge;3
['tigre', 'rouge', '3']
singe;jaune;1
['singe', 'jaune', '1']
éléphant;bleu;4
['éléphant', 'bleu', '4']
```

2 Introduction :

2.1 Les SGBD (Système de gestion de Base de Données):

1. SQLite :

SQLite est une bibliothèque écrite en langage C qui propose un moteur de base de données relationnelle accessible par le langage SQL. Il s'agit d'un développement libre de droits.

Contrairement aux serveurs de bases de données traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégrée aux programmes.

SQLite permet de gérer des petites bases de données (inférieure à 1Go) et remplace avantageusement la gestion de fichiers.

SQLite est un moteur de base de données très léger, couramment utilisé pour les systèmes embarqués (smartphone par exemple).

Il ne permet pas toutefois de gérer beaucoup d'utilisateurs.

2. MySQL :

MySQL est un système de gestion de bases de données relationnelles (SGBDR). Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés, autant par le grand public (applications web principalement) que par des professionnels.

MySQL est conçu pour gérer des bases de données importantes (quelques Go) et permet de nombreux accès.

3. PostgreSQL :

PostgreSQL est un système de gestion de base de données relationnelle et objet (SGBDRO) libre.
Il permets de gérer des bases de données de plusieurs téra octets

4. Oracle et autre :

Des solutions propriétaires

3 Quelques liens :

1. Installation de sqlitebrowser

https://sqlitebrowser.org/dl/
installation sous debian :

```
sudo apt-get update
sudo apt-get install sqlitebrowser
```

2. Vidéo 1 :

https://www.youtube.com/watch?v=K0zF1GiPrxY

3. video 2

https://www.youtube.com/watch?v=gwcOg2x_8uc

4. Pour charger des données depuis en fichier csv

https://tube.ac-lyon.fr/w/85399ea5-bba0-428b-9470-2d3bb41b7de1

5. BDD en ligne :

https://extendsclass.com/sqlite-browser.html

4 Outils utilisés :

Système de Gestion de Bases de Données : SQLite3
Outil graphique : DB-Browser pour SQLite
Module SQLite3 de python

5 Exemple à partir d'un exercice de 1ere année de BDD : cinémathèque

t_producteurs

id	nom	prenom
1	EASTWOOD	Clint
2	BURTON	Tim
3	SPIELBERG	Steven

t_films

id	titre	annee	genre	id_producteur
1	Impitoyable	1992	western	1
2	Gran torino	2008	drame	1
3	Ed Wood	1994	drame	2
4	Batman	1994	action	2
5	Big Eyes	NULL	comédie	2

t_salles

id	nom	3D	capacite
1	salle 1	oui	300
2	salle 2	non	175
3	salle 3	non	150

t_seances

id	id_film	id_salle	date	heure
1	1	1	10/10/13	18.0
2	2	1	10/10/13	20.0
3	2	2	11/10/13	21.0
4	4	2	12/10/13	18.0
5	5	2	12/10/13	20.5

5.1 Remarques :

Chaque table possède une clé primaire auto-incrémentée dont le nom est « id »
L'heure des séances est un réel codé en centième d'heure : 20h30 => 20,5
La date est une chaîne de caractère

5.2 Quelques requêtes :

1. Requête avec une réponse unique :

Obtenir nom et prénom d'un producteur a partir de son identifiant :

```
SELECT t_producteurs.nom, t_producteurs.prenom
      FROM t_producteurs
      WHERE id = 1;
```

	nom	prenom
1	EASTWOOD	Clint

2. Identifiant d'un producteur a partir de son nom :

```
SELECT t_producteurs .id
      FROM t_producteurs
      WHERE nom = "SPIELBERG"
```

3. Requête avec une réponse multiple :

id et heure des films commençant après 20 h

```
SELECT t_seances.id_film, t_seances.heure
      FROM t_seances
      WHERE t_seances.heure >= 20.0
```

	id	heure
1	2	20.0
2	3	21.0
3	5	20.5

4. Requête avec une réponse multiple, requête imbriquée :

Afficher les films d'un réalisateur a partir de son nom

```
SELECT t_films.titre
	FROM t_films
	WHERE t_films.id_producteur
	in (
		SELECT t_producteurs.id
			FROM t_producteurs
			WHERE t_producteurs.nom = "EASTWOOD"
	)
;
```

	titre
1	Impitoyable
2	Gran Torino

5. Requête avec une réponse multiple et jointure :

Titre, date et heure des films commençant après 20 h

```
SELECT t_films.titre, t_seances.date, t_seances.heure
	FROM t_films INNER JOIN t_seances
		ON t_seances.id_film = t_films.id
	WHERE t_seances.heure >= 20.0;
```

	titre	date	heure
1	Gran Torino	10/10/13	20.0
2	Gran Torino	11/10/13	21.0
3	Big Eyes	12/10/13	20.5

6. Requête avec une réponse multiple et requête imbriquée :

Titre, date et heure des films commençant après 20 h

```
SELECT t_films.titre, t_seances.date, t_seances.heure
	FROM t_films, t_seances
	WHERE t_films.id
		in (
			SELECT t_seances.id_film
			FROM t_seances
			WHERE t_seances.heure >= 20.0)
		AND t_seances.id_film = t_films.id
```

7. Requête d’insertion :

Ajouter un producteur.

```
INSERT INTO t_producteurs (nom, prenom) VALUES ("OZON", "Francois");
```

5.3 Programmation en python :

1. Requête avec une réponse unique :

nom et prénom d'un producteur a partir d'un id avec saisie de l’identifiant :

```
SELECT nom, prenom FROM t_producteurs WHERE id = 1
```

```
import sqlite3
if __name__ == "__main__":
	# déclaration des variables
	nom_bdd:str
	connecteur:sqlite3.Connection
	curseur:sqlite3.Cursor
	requete: str
	reponse:tuple # pour réponse unique
	reponses:list[tuple] # pour réponses mutiples

	nom_bdd = "cine2.sqlite3"
```

```
      # connecteur à la BDD
      connecteur = sqlite3.connect(nom_bdd)

      # création d'un curseur (pour exécuter les requètes)
      curseur = connecteur.cursor()

      # ecriture de la requète
      requete = """SELECT t_producteurs.nom, t_producteurs.prenom
                  FROM t_producteurs
                  WHERE  t_producteurs.id = 1;"""
      print(requete)

      # exécution de la requète
      curseur.execute(requete)

      # lecture du résultat unique
      reponse = curseur.fetchone()
      print(reponse)
```

Remarque1 : important d'inclure une clause try / except pour fermer la BDD dans tous les cas

```
try:

except Exception as exc:
      print(f"ERREUR {exc}")
      connection.rollback() # revenir au dernier enregistrement
finally:
      connection.close()
```

2. Requête avec une réponse multiple :

id et heure des films commençant après 20 h

```
SELECT t_seances.id, t_seances.heure
      FROM t_seances
      WHERE t_seances.eure >= 20.0
```

```
# construction de la requête
requete: str = """SELECT id_film, heure
                  FROM t_seances
                  WHERE heure >= 20;"""
# affichage de la requête (pour vérification)
print(requete)

# appel de l'API pour l'exécution de la requête
cursor.execute(requete)

# extraction de l'information (l'information est une liste de tuples)
reponses = resultat.fetchall() # fetchall() pour une réponse multiple

# affichage des informations
print(liste)
for rep in reponses:
      id_film, heure = rep
      print (f"{id_film} {heure}h")
```

3. Requête d'insertion :

Ajouter un producteur dans la table t_producteurs :

```
# saisie du nom
nom = input("nom du producteur : ")
# saisie du prenom
prenom = input ("prenom du producteur : ")

# construction de la requête
requete = f"INSERT INTO t_producteurs (nom, prenom) VALUES ('{nom}', '{prenom}');"

# affichage de la requête (pour vérification)
print(requete)

# appel de l'API pour l'exécution de la requête
cursor.execute(requete)

# sauvegarde de la BDD (indispensable)
connection.commit()
```

4. Fermeture de la connexion :

```
connection.close()
```