Opérations CRUD (Create, Read, Update, Delete) avec Python et MySQL

Introduction

Les opérations **CRUD** (**Create**, **Read**, **Update**, **Delete**) sont les bases de la gestion des données dans une base de données. Elles permettent :

- **Create :** D'ajouter de nouvelles données.
- **Read :** De récupérer et afficher les données existantes.
- **Update :** De modifier des données existantes.
- **Delete :** De supprimer des données.

Pourquoi utiliser une base de données avec Python?

1. **Automatisation des tâches :**
   - Une base de données permet de centraliser les informations et d'y accéder automatiquement via des scripts.
   - Elle simplifie la gestion des grandes quantités de données.

2. **Sécurité :**
   - Les bases de données offrent des fonctionnalités pour sécuriser l'accès aux informations (authentification, autorisations).
   - Les données sont mieux protégées que dans des fichiers plats (comme les fichiers texte).

3. **Performance :**
   - Les bases de données sont optimisées pour rechercher, insérer, mettre à jour et supprimer rapidement des données.
   - Elles supportent de grandes quantités de données.

4. **Organisation des données :**
   - Une structure relationnelle (avec plusieurs tables liées) garantit la cohérence et réduit la redondance des informations.

Partie 1. Connexion à MySQL

1. Installation

Pour **interagir** avec la base de données, nous utilisons **mysql-connector-python**. Si ce module n'est pas encore installé, **installez-le** avec la commande suivante via **pip**:

```
pip install mysql-connector-python
```

2. Comment fonctionne la librairie ?

1. **Connexion à la base de données**
   - Vous devez fournir les **informations de connexion** (hôte, utilisateur, mot de passe, base de données).
   - Si la connexion est établie avec succès, vous obtenez un **objet connexion**.
2. **Interaction avec la base**
   - Une fois connecté, vous utilisez un **cursor** pour exécuter des **requêtes SQL**. Un **cursor** est un objet qui interagit avec la base de données et manipule les données.
   - Le curseur peut exécuter des requêtes comme **SELECT**, **INSERT**, **UPDATE**, **DELETE**.
3. **Gestion des résultats**
   - Après une requête (comme **SELECT**), vous pouvez récupérer les résultats via des méthodes comme **.fetchall( )** ou **.fetchone( )**.
4. **Fermeture de la connexion**
   - Il est crucial de **fermer** la connexion pour **libérer** les ressources après les opérations.

3. Connexion à la base de données

Afin d'interagir avec une base de données MySQL, il faut d'abord établir une connexion avec la base de données. Cette connexion est essentielle car elle agit comme un pont entre le script Python et le serveur MySQL.

Étapes pour créer une connexion à la base de données MySQL

1. **Importer le module mysql.connector :** La **bibliothèque** nécessaire pour interagir avec MySQL.
2. **Créer une connexion :** Fournir les **informations** nécessaires pour se connecter (hôte, utilisateur, mot de passe, base de données).
3. **Gérer les erreurs :** Utiliser des blocs **try-except** pour s'assurer que la connexion est gérée proprement et que toute erreur est capturée.

```
# Étape 1 : Importer la bibliothèque

import mysql.connector
from mysql.connector import Error  # Pour capturer les erreurs

try:
    # Connexion à la base de données
    connection = mysql.connector.connect(
        host="localhost",          # Adresse du serveur MySQL
        user="user",               # Nom d'utilisateur
        password=« yourpassword",  # Mot de passe de l'utilisateur
        database="yourdatabase"    # Nom de la base de données
    )

    # Vérifier si la connexion est établie
    if connection.is_connected():
        print("Connexion réussie à la base de données MySQL")

except Error as e:
    # En cas d'erreur
    print(f"Erreur lors de la connexion à la base de données : {e}")
```

Explication du code

- **mysql.connector.connect :** C'est la fonction principale pour établir une **connexion** avec la base de données. Elle prend plusieurs **arguments** :
  - **host** : L'adresse du serveur où MySQL est hébergé (souvent localhost ou une adresse IP).
  - **user** : Le nom de l'utilisateur MySQL.
  - **password** : Le mot de passe de cet utilisateur.
  - **database** : La base de données spécifique que vous souhaitez utiliser.

- **is_connected :** Permet de vérifier si la connexion a été établie avec **succès**.

- **Gestion des erreurs avec try...except :** Si une **erreur** se produit (par exemple, mauvais mot de passe ou serveur indisponible), elle est **capturée** dans le bloc **except**.

On peut créer une fonction qui retournera la connexion afin d'effectuer des requêtes plus tard.
Étape suivante : Utilisation du Cursor après la connexion

Une fois la connexion établie, vous pouvez créer un **cursor** à partir de l'objet connexion pour interagir avec la base de données.

---

3. Qu'est-ce qu'un cursor et comment cela fonctionne ?

Un **cursor** (curseur) est un **objet** fourni par la bibliothèque **mysql-connector-python**. Il sert de point d'entrée pour **interagir** avec la base. Pensez à lui comme un "pointeur" permettant d'exécuter des **commandes SQL** et de **récupérer** des résultats.

Fonctionnement du curseur :

1. **Création :** Vous créez un curseur via la méthode **.cursor( )** de l'objet connexion.
2. **Exécution des requêtes :** Le curseur est utilisé pour exécuter des requêtes SQL avec **.execute( )**.
3. **Récupération des résultats :** Après une requête **SELECT**, vous utilisez des méthodes comme **.fetchone( )**, **.fetchall( )** pour lire les données.
4. **Fermeture :** Une fois que vous avez fini avec le curseur, vous devez le fermer avec **.close()**.

Partie 2. Les opérations de CRUD

Les opérations **CRUD** sont les **quatre** actions **principales** que vous effectuez sur une base de données :

- **CREATE (Créer) :** Insérer des données dans une table.
- **READ (Lire) :** Lire ou interroger des données dans une table.
- **UPDATE (Mettre à jour) :** Modifier des données existantes.
- **DELETE (Supprimer) :** Effacer des données.

Lorsque vous travaillez avec un **curseur MySQL** en Python, vous utilisez des **méthodes** pour exécuter des **requêtes** SQL et gérer les données telles que : **execute( ), fetchall( ), fetchone( ), executemany( ), lastrowid( ), rowcount( ), commit( ), close( )**.

---

1. execute( )

La méthode **.execute()** est utilisée pour exécuter une commande SQL. Elle peut être utilisée pour :

- Insérer des données (**INSERT**),
- Lire des données (**SELECT**),
- Mettre à jour des données (**UPDATE**),
- Supprimer des données (**DELETE**),
- Créer ou modifier des tables (**CREATE**, **ALTER**, **DROP**).

Syntaxe :

```
cursor.execute(sql_query, parameters)
```

- **sql_query** : La commande SQL à exécuter.
- **parameters** (optionnel) : Les valeurs dynamiques à insérer dans la requête (remplace les placeholders %s).

Exemple :

```
# Requête avec des placeholders pour éviter les injections SQL
sql_insert = "INSERT INTO users (name, email, age) VALUES (%s, %s, %s)"
values = ("Alice", "alice@example.com", 30)
cursor.execute(sql_insert, values)
```

2. fetchall( )

La méthode **.fetchall( )** récupère **toutes les lignes** du résultat d'une requête SQL.

Quand l'utiliser ?

- Utilisez-la lorsque vous voulez récupérer plusieurs lignes ou toutes les lignes d'une table.
- Elle retourne les résultats sous forme de liste de tuples.

Exemple :

```
# Exécuter une requête SELECT
cursor.execute("SELECT * FROM users")

# Récupérer tous les résultats
results = cursor.fetchall()

# Parcourir les résultats
for row in results:
    print(row)
```

Sortie :

```
(1, 'Alice', 'alice@example.com', 30)
(2, 'Bob', 'bob@example.com', 25)
```

3. fetchone( )

La méthode **.fetchone( )** récupère **une seule ligne** du résultat d'une requête SQL. Si aucune ligne n'est disponible, elle retourne **None**.

Quand l'utiliser ?

- Utilisez-la lorsque vous savez que la requête retourne une seule ligne (par exemple, rechercher un utilisateur par son ID).
- Cela économise de la mémoire si vous n'avez besoin que d'une ligne.

Exemple :

```
# Requête SELECT pour un utilisateur spécifique
cursor.execute("SELECT * FROM users WHERE id = %s", (1,))

# Récupérer une seule ligne
user = cursor.fetchone()

# Afficher l'utilisateur
print(user)
```

Sortie :

```
(1, 'Alice', 'alice@example.com', 30)
```

4. executemany( )

La méthode **.executemany( )** permet d'exécuter une requête SQL **plusieurs fois** avec différentes valeurs.

Quand l'utiliser ?

Utilisez-la pour insérer ou mettre à jour plusieurs lignes en une seule commande.

Exemple :

```
# Requête avec placeholders
sql_insert = "INSERT INTO users (name, email, age) VALUES (%s, %s, %s)"

# Liste de valeurs
values = [
    ("Alice", "alice@example.com", 30),
    ("Bob", "bob@example.com", 25),
    ("Charlie", "charlie@example.com", 28)
]

# Exécuter la requête pour plusieurs valeurs
cursor.executemany(sql_insert, values)

# Valider les modifications
connection.commit()
```

5. commit( )

La méthode .commit() est utilisée pour **valider les modifications** apportées à la base de données (comme un **INSERT**, **UPDATE** ou **DELETE**).

Pourquoi est-ce nécessaire ?

- **Par défaut**, les changements apportés à la base de données (insertion, mise à jour, suppression) **sont temporaires** jusqu'à ce que vous appeliez **.commit( )**.
- Sans **.commit( )**, les modifications seront **annulées** lorsque la connexion sera fermée.

Exemple :

```
# Insertion d'une nouvelle ligne
cursor.execute("INSERT INTO users (name, email, age) VALUES (%s, %s,
%s)", ("Alice", "alice@example.com", 30))

# Valider l'insertion
connection.commit()
```

6. lastrowid

La propriété **.lastrowid** renvoie l'**ID de la dernière ligne insérée** dans une table avec une clé primaire **auto-incrémentée**.

Quand l'utiliser ?

Utilisez-la pour connaître l'ID généré automatiquement lors d'une insertion.

Exemple :

```
cursor.execute("INSERT INTO users (name, email, age) VALUES (%s, %s,
%s)", ("David", "david@example.com", 40))

print("ID de la nouvelle ligne :", cursor.lastrowid)
```

Sortie :

```
ID de la nouvelle ligne : 4
```

7. rowcount( )

La propriété **.rowcount** renvoie le nombre de lignes affectées par la **dernière requête SQL** (par exemple, le nombre de lignes insérées, mises à jour ou supprimées).

Exemple :

```
cursor.execute("UPDATE users SET age = %s WHERE age > %s", (30, 25))
print(f"Lignes mises à jour : {cursor.rowcount}")
```

Sortie :

```
Lignes mises à jour: 2
```

8. close( )

La méthode **.close( )** ferme le curseur une fois que vous avez terminé d'exécuter les requêtes.

Pourquoi est-ce nécessaire ?

- Elle **libère** les ressources associées au curseur.
- Bien qu'elle ne soit pas obligatoire, c'est une **bonne pratique**.

Exemple :

```
cursor.close()
```

Récapitulatif des méthodes

Méthode	Description
.execute()	Exécute une requête SQL.
.fetchall()	Récupère toutes les lignes du résultat d'une requête.
.fetchone()	Récupère une seule ligne du résultat d'une requête.
.executemany()	Exécute une requête SQL plusieurs fois avec des valeurs différentes.
.commit()	Valide les modifications dans la base de données.
.close()	Ferme le curseur ou la connexion.
.lastrowid	Renvoie l'ID de la dernière ligne insérée.
.rowcount	Renvoie le nombre de lignes affectées par la dernière requête SQL.

TP : Gestion clients

1. Créez la base de données « **gestion_clients** » ainsi que la table « **clients** » à partir du script ci-dessous
2. Créez un nouveau dossier « **python_mysql** »,
3. À l'intérieur du dossier « **python_mysql** » créez un fichier « **app.py** »,

---

1. Structure de la base de données

Nous allons utiliser une table appelée **clients**, qui représente une liste de clients d'une entreprise.

Table clients

Colonne	Type
id_client	INT (Primary Key, AUTO_INCREMENT)
nom	VARCHAR(30)
email	VARCHAR(50)
telephone	VARCHAR(10)
ville	VARCHAR(30)

---

2 . Création de la base de données

Script SQL

```sql
-- Créer la base de données
CREATE DATABASE gestion_clients;

-- Utiliser la base de données
USE gestion_clients;

-- Table "clients"
CREATE TABLE clients (
    id_client INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(30) NOT NULL,
    email VARCHAR(50) NOT NULL,
    telephone VARCHAR(10),
    ville VARCHAR(30)
);

-- Insérer des données
INSERT INTO clients (nom, email, telephone, ville) VALUES
('Alice Dupont', 'alice.dupont@email.com', '0601234567', 'Paris'),
('Bob Martin', 'bob.martin@email.com', '0612345678', 'Lyon'),
('Charlie Durand', 'charlie.durand@email.com', '0623456789', 'Marseille');
```

3. Créer un utilisateur de base de données

Créez un **utilisateur** afin de se connecter à la base de données en lui attribuant seulement les **autorisations nécessaires**.

4. Connexion à la base de données

Créez la **fonction de connexion** à la base de données en veillant à **traiter** les éventuelles **erreurs** de connexions.

5. Création d'un menu

Créez un **menu** permettant de sélectionner les choix suivant :

- **Lire** les données
- **Ajouter** des données
- **Mettre à jour** des données
- **Supprimer** des données

6. Afficher les données

Créez une fonction permettant de **lire tous les enregistrements** de la table « clients » et assignez la au **menu 1**.

7. Insérer des données

Créez une fonction permettant **d'ajouter** un enregistrement dans la table « clients » et assignez la au **menu 2**.

8. Mettre à jour des données

Créez une fonction permettant de **modifier** toutes les informations d'un enregistrement de la table « clients » et assignez la au **menu 3**.

9. Supprimer des données

Créez une fonction permettant de **supprimer** un enregistrement de la table « clients » et assignez la au **menu 4**.

10. Rechercher des données

Ajouter un menu « **Rechercher** » puis créez une fonction permettant de **rechercher** un client sur la base de son nom **ou** de son email **ou** de son numéro de téléphone **ou** de sa ville, puis assignez cette fonction à votre nouveau menu.

Limites de l'affichage brut :

- **Difficile à lire :** Les données ne sont pas présentées sous forme de tableau.
- **Pas de contexte :** On ne sait pas quelle colonne correspond à quelle donnée.
- **Peu professionnel :** Ce format n'est pas adapté à une interface utilisateur ou à un rapport.