**EXERCICE: Gestion des données d'étudiants avec SQLite**

**Objectif :**

Créer un programme en Python qui permet de mettre à jour les informations d'un étudiant dans une base de données SQLite.

**Instructions :**

1. **Connexion à la base de données :**
   * Établissez une connexion à une base de données SQLite nommée madb.db.
   * Si la base de données n'existe pas, elle sera créée automatiquement.
2. **Table student :**
   * Créez une table student si elle n'existe pas déjà. Cette table doit contenir les colonnes suivantes :
     + id : Identifiant unique pour chaque étudiant (clé primaire, auto-incrémentée).
     + name : Nom de l'étudiant (texte, obligatoire).
     + email : Email de l'étudiant (texte, obligatoire).
3. **Saisie des informations :**
   * Demandez à l'utilisateur de saisir les informations suivantes :
     + id : L'identifiant de l'étudiant dont les informations doivent être mises à jour.
     + name : Le nouveau nom de l'étudiant.
     + email : Le nouvel email de l'étudiant.
4. **Mise à jour des informations :**
   * Mettez à jour le nom et l'email de l'étudiant correspondant à l'identifiant saisi dans la base de données.
5. **Affichage des résultats :**
   * Après la mise à jour, affichez les informations de tous les étudiants présents dans la table student.
6. **Validation et fermeture :**
   * Validez les changements (utilisez conn.commit()).
   * Fermez la connexion à la base de données.

Syntax de set :

UPDATE table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

Les méthodes `fetch` (comme `fetchall()` et `fetchone()`) et la boucle `for` sont utilisées en Python pour récupérer et traiter les résultats d'une requête SQL exécutée avec un curseur (`cursor`). Elles servent des objectifs similaires, mais elles sont utilisées de manières différentes et ont des particularités distinctes.

### 1. Utilisation des méthodes `fetch`

Les méthodes `fetch` sont utilisées pour récupérer les résultats d'une requête SQL à partir d'un curseur.

- \*\*`fetchall()`\*\* :

- Récupère \*\*toutes\*\* les lignes du résultat de la requête et les renvoie sous forme d'une liste de tuples.

- Chaque tuple représente une ligne de la table, et les éléments du tuple représentent les colonnes.

- Exemple :

```python

result = cursor.execute("SELECT \* FROM student")

rows = result.fetchall() # Récupère toutes les lignes

print(rows)

```

Ici, `rows` est une liste contenant toutes les lignes de la table `student`.

- \*\*`fetchone()`\*\* :

- Récupère \*\*une seule\*\* ligne du résultat de la requête à la fois.

- Si tu appelles plusieurs fois `fetchone()`, tu obtiens les lignes suivantes une par une.

- Exemple :

```python

row = cursor.fetchone() # Récupère la première ligne

print(row)

```

### 2. Utilisation de la boucle `for`

La boucle `for` est utilisée pour \*\*itérer\*\* directement sur l'objet `cursor` qui est un itérable. Chaque itération renvoie une ligne du résultat de la requête.

- Lorsque tu utilises une boucle `for` sur un curseur, il \*\*récupère automatiquement les résultats ligne par ligne\*\* et tu peux les traiter directement dans la boucle.

- Exemple :

```python

result = cursor.execute("SELECT \* FROM student")

for row in result:

print(row)

```

Ici, la boucle `for` parcourt chaque ligne de la table `student` et l'imprime.

### Différences clés

1. \*\*Mémoire\*\* :

- \*\*`fetchall()`\*\* charge toutes les lignes en mémoire à la fois, ce qui peut être inefficace si le résultat de la requête contient beaucoup de données.

- \*\*La boucle `for`\*\* sur le curseur est plus efficace en termes de mémoire, car elle traite une ligne à la fois.

2. \*\*Accès aux données\*\* :

- \*\*`fetchall()`\*\* renvoie toutes les lignes sous forme d'une liste, ce qui te permet de les parcourir ou de les accéder par index (comme `rows[0]` pour la première ligne).

- \*\*La boucle `for`\*\* ne te donne accès à chaque ligne qu'une fois. Tu ne peux pas "revenir en arrière" sans relancer la requête ou stocker les résultats.

3. \*\*Cas d'utilisation\*\* :

- Utilise \*\*`fetchall()`\*\* si tu as besoin de toutes les données en mémoire en même temps pour des traitements ultérieurs.

- Utilise \*\*la boucle `for`\*\* si tu veux traiter chaque ligne de manière séquentielle sans avoir à charger toutes les données en mémoire.

### Résumé

- \*\*`fetchall()`\*\* : Récupère toutes les lignes en une seule fois et les stocke dans une liste. Utile si tu as besoin d'accéder de manière aléatoire aux données.

- \*\*Boucle `for` sur un curseur\*\* : Traite les lignes une par une, ce qui est plus efficace pour les grandes quantités de données et pour les traitements en streaming ou en séquence.