diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Connexion et Requêtage de Base de Données avec MySQL/MariaDB

Sommaire

Introduction à MySQL/MariaDB

MySQL et MariaDB sont des systèmes de gestion de base de données relationnelle (SGBDR) open source largement utilisés dans l'industrie. Ils offrent une solution robuste pour le stockage, la gestion et la récupération de données de manière efficace.

Caractéristiques principales

1. **Performance élevée :** Réputé pour sa rapidité d'exécution, ce qui en fait un choix populaire pour les applications nécessitant des opérations rapides sur la base de données.
2. **Fiabilité :** Ces bases de données sont connues pour leur stabilité et leur fiabilité, ce qui en fait un choix privilégié pour les applications critiques.
3. **Open Source :** Logiciels open source, ce qui signifie que vous pouvez les utiliser, les modifier et les distribuer librement.

> **mysql et mariadb**
>
> MariaDB est un fork de MySQL créé pour maintenir une alternative open source après l'acquisition de MySQL par Oracle. Les deux partagent une base commune mais diffèrent en termes de licence, communauté, et fonctionnalités.

Installation de MySQL/MariaDB

Pour installer MySQL avec Python, vous pouvez utiliser le module `mysql-connector-python`. Utilisez la commande suivante dans votre terminal :

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

Assurez-vous également d'avoir MySQL/MariaDB installé localement ou configurez l'accès à une base de données distante.

Utilisation Python pour MySQL/MariaDB

L'utilisation d'une API Python simplifie grandement les opérations sur une base de données MySQL/MariaDB. Dans cette section, nous explorerons l'utilisation du module `mysql-connector-python` comme une API Python pour interagir avec la base de données de manière plus efficace.

1. Établissement de la Connexion

Avant de pouvoir effectuer des opérations sur la base de données, nous devons établir une connexion en utilisant le module `mysql.connector`.

```
import mysql.connector

# Établir la connexion
conn = mysql.connector.connect(
    host="localhost",
    user="votre_utilisateur",
    password="votre_mot_de_passe"
)
```

2. Création d'un Curseur

Après s'être connecté à notre base de donnée, on doit créer un curseur. Un curseur est un objet qui nous permet d'envoyer des requêtes SQL à la base de données et de récupérer les résultats.

```
# Créer un objet curseur
cursor = conn.cursor()
```

3. Création de Bases de Données et de Tables

Pour commencer nos requêtes, nous devons créer une base de données et une table pour stocker nos données. Voici comment vous pouvez le faire en utilisant Python et le module `mysql-connector-python`.

N'oubliez pas de changer le `ma_base_de_donnees` et `ma_table`.

```
# Créer une base de données
cursor.execute("CREATE DATABASE IF NOT EXISTS ma_base_de_donnees")

# Utiliser la base de données
cursor.execute("USE ma_base_de_donnees")
```

```
# Créer une table
cursor.execute("""
    CREATE TABLE IF NOT EXISTS ma_table (
        id INT AUTO_INCREMENT PRIMARY KEY,
        nom VARCHAR(255),
        age INT
    )
""")

# Valider et appliquer les changements
conn.commit()
```

Le `commit()` permet de mettre à jour la BDD. Si on ne le fait pas, les modification resteront en local.

> **Importance du `commit`**
>
> Si vous faites des modifications sans commit, avec un select pour verrez les modifications apportées. Sauf que si vous lancez une autre session, les modifications ne seront pas confirmées.

4. Utilisation d'Opérations CRUD

- **Création (Create) :**

```
# Insérer des données dans la table
cursor.execute("INSERT INTO ma_table (nom, age) VALUES ('John Doe', 25)")
conn.commit()
```

- **Lecture (Read) :**

```
# Sélectionner toutes les données de la table
cursor.execute("SELECT * FROM ma_table")
result = cursor.fetchall()
for row in result:
    print(row)
```

- **Mise à Jour (Update) :**

```
# Mettre à jour des données dans la table
cursor.execute("UPDATE ma_table SET age = 26 WHERE nom = 'John Doe'")
conn.commit()
```

- **Suppression (Delete) :**

```
# Supprimer des données de la table
cursor.execute("DELETE FROM ma_table WHERE nom = 'John Doe'")
conn.commit()
```

5. Fermeture de la Connexion

N'oubliez pas de fermer la connexion une fois que vous avez terminé vos opérations.

```
# Fermer le curseur
cursor.close()

# Fermer la connexion
conn.close()
```

Gestion des Erreurs et Bonnes Pratiques

Lors de l'accès à une base de données, il est crucial de mettre en place une gestion des erreurs efficace pour anticiper et traiter les problèmes potentiels. De plus, l'application de bonnes pratiques contribue à assurer la stabilité, la sécurité et la performance de votre code.

1. Gestion des Erreurs

La gestion des erreurs dans vos opérations de base de données est essentielle pour maintenir la robustesse de votre application. Utilisez des blocs `try/except` pour capturer et gérer les exceptions potentielles.

On peut ajouter le `database=` si on connais déjà notre base cible. Cela évitera faire par la suite un `USE <database>`.

```
import mysql.connector

try:
    ...

except mysql.connector.Error as err:
    # Gérer les erreurs spécifiques à MySQL
    print(f"Erreur MySQL : {err}")

except Exception as e:
    # Gérer d'autres erreurs
    print(f"Une erreur s'est produite : {e}")

finally:
    # Assurez-vous de fermer le cursor, que l'opération réussisse ou échoue.
 if 'cursor' in locals() and cursor is not None:
        cursor.close()
 # Pareil pour la connexion.
    if 'conn' in locals() and conn.is_connected():
        conn.close()
```

2. Utilisation de Transactions

Les transactions sont un moyen de regrouper plusieurs opérations en une seule unité, garantissant que toutes réussissent ou échouent ensemble. Utilisez des transactions pour maintenir la cohérence des données.

```
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="votre_utilisateur",
    password="votre_mot_de_passe",
    database="ma_base_de_donnees"
)

# Commencer une transaction
conn.start_transaction()

# Exécuter des opérations CRUD
cursor = conn.cursor()
cursor.execute("UPDATE ma_table SET age = 26 WHERE nom = 'John Doe'")

# Valider la transaction
conn.commit()
```

> **Rollback**
>
> Le rollback permet de revenir en arrière, on l'utilise dans le `except` pour annuler les toutes les modifications si une d'entre elles échouent.
>
> Cela nous permet de ne pas avoir de MAJ partielle des données.

3. Utilisation de Paramètres dans les Requêtes

Évitez l'injection SQL en utilisant des paramètres dans vos requêtes plutôt que d'incorporer des valeurs directement dans la chaîne SQL.

```
# Mauvaise pratique (risque d'injection SQL)
nom = "John Doe"
cursor.execute(f"SELECT * FROM ma_table WHERE nom = '{nom}'")

# Bonne pratique (utilisation de paramètres)
nom = "John Doe"
cursor.execute("SELECT * FROM ma_table WHERE nom = %s", (nom,))
```

4. Fermeture des Ressources Correctement

Assurez-vous de fermer correctement toutes les ressources, comme les connexions et les curseurs, pour éviter les fuites de ressources et garantir une utilisation efficace des ressources système.

```
# Fermer le curseur
cursor.close()

# Fermer la connexion
conn.close()
```

5. Sécurité et Autorisations

Appliquez le principe du moindre privilège en définissant des autorisations spécifiques pour les utilisateurs de la base de données. Évitez d'utiliser des comptes ayant des droits excessifs.

6. Sauvegardes Régulières

Réalisez des sauvegardes régulières de votre base de données pour prévenir la perte de données en cas de problème.