Bases de données et Python

SQLite3 est un SGBD facile à manipuler. Il est inclus dans python sans besoin de configuration et il est transactionnel. Il est très rapide et léger et la base de données est entièrement stockée dans un seul fichier. Il est utilisé dans plusieurs applications comme une manière interne de stockage. Python inclus un module appelé "sqlite3" pour permettre l'utilisation des bases de données. Ce module est gérable par SQL suivant les spécifications de DB-API 2.0.

Utiliser le module SQLite :

Pour importer le module SQLite:

```
import sqlite3
```

Créer une base de données avec SQLite :

Là aussi pour créer une base de données avec SQLite, rien de plus simple:

```
conn = sqlite3.connect('ma_base.db')
```

Lorsque vous exécuterez votre programme vous remarquerez que si la base n'existe pas encore, un fichier sera créé dans le dossier de votre programme. Et si celui-ci existe déjà il sera réutilisé. Vous pouvez bien évidemment choisir l'emplacement de votre base de données en renseignant un path, exemple: "C :/data/ma_base.db" . Il vous faudra cependant vérifier que le dossier existe avant de l'utiliser.

Lorsque le travail que vous attendiez est terminé, pensez à fermer la connexion vers la base:

```
conn.close()
```

Créer une table avec SQLite :

Voici un exemple de création de table:

```
cursor = conn.cursor()
cursor.execute("""
CREATE TABLE IF NOT EXISTS users(
     id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,
     name TEXT,
     age INTERGER
)
""")
conn.commit()
```

Supprimer une table avec SQLite:

```
cursor = conn.cursor()

cursor.execute("""

DROP TABLE users

""")

conn.commit()
```

Insérer des données :

Il existe plusieurs manières d'insérer des données, la plus simple étant celle-ci:

```
cursor.execute("""

INSERT INTO users(name, age) VALUES(?, ?)""", ("olivier", 30))
```

Vous pouvez passer par un dictionnaire:

```
data = {"name" : "olivier", "age" : 30}

cursor.execute("""

INSERT INTO users(name, age) VALUES(:name, :age)""", data)
```

Vous pouvez récupérer l'id de la ligne que vous venez d'insérer de cette manière:

```
id = cursor.lastrowid

print('dernier id: %d' % id)
```

Il est également possible de faire plusieurs insert en une seule fois avec la fonction executemany:

```
users = []

users.append(("olivier", 30))

users.append(("jean-louis", 90))

cursor.executemany("""

INSERT INTO users(name, age) VALUES(?, ?)

""", users)
```

Récupérer des données

Vous pouvez récupérer la première ligne correspondant à votre recherche à l'aide de la fonction **fetchone**.

```
cursor.execute("""SELECT name, age FROM users""")

user1 = cursor.fetchone()

print(user1)
```

Le résultat est un tuple:

```
('olivier', 30)
```

Vous pouvez récupérer plusieurs données de la même recherche en utilisant la fonction **fetchall()**.

```
cursor.execute("""SELECT id, name, age FROM users""")

rows = cursor.fetchall()

for row in rows:

    print('{0} : {1} - {2}'.format(row[0], row[1], row[2]))
```

L'objet curseur fonctionne comme un itérateur, invoquant la méthode **fetchall()** automatiquement:

```
cursor.execute("""SELECT id, name, age FROM users""")

for row in cursor:

    print('{0} : {1}, {2}'.format(row[0], row[1], row[2]))
```

Pour la recherche spécifique, on utilise la même logique vu précédemment:

```
id = 2

cursor.execute("""SELECT id, name FROM users WHERE id=?""", (id,))

response = cursor.fetchone()
```

Modifier des entrées

Pour modifier des entrées:

```
cursor.execute("""UPDATE users SET age = ? WHERE id = 2""", (31,))
```

SQLite transactions : rollback

Pour revenir au dernier **commit**, utilisez la méthode **rollback**.

```
conn.rollback()
```