


Cours	Cours	
BTS SNIR	Base de données et Python	
2 ^{ème} année	Sqlite et mysql	

Base de données et Python

Vous pouvez stocker des informations dans des fichiers XML mais très rapidement, suivant l'ampleur de votre projet, cette technologie atteindra ses limites que ce soit en terme de performances ou d'utilisation au quotidien. Si plusieurs personnes souhaitent modifier un fichier texte de données en même temps les erreurs seront inévitables. Il existe une solution pour stocker des informations et pouvoir travailler avec : **les bases de données**.

Une base de données c'est quoi?

Une base de données (*database* en anglais) est un conteneur dans lequel il est possible de stocker des données de façon structurée. Cette structure permet au programme informatique connectée à celle-ci de **faire des recherches complexes**.

Un langage standardisé -SQL- est dédié à cette structure et permet aussi bien de faire des recherches mais aussi des modifications ou des suppressions.

Les logiciels de gestion de bases de données les plus utilisées aujourd'hui sont des **SGBDR** - *Système de gestion de base de données relationnelles* -, c'est à dire que les données sont liées les unes aux autres, par exemple on peut définir que si on supprime une information, d'autres informations dépendantes de cette dernière soient elles-aussi automatiquement supprimées. Cela garantit une **cohérence de données**.

Il ne faut donc pas confondre une **base de données** qui est un conteneur et le **SGBDR** qui est un logiciel de gestion de bases de données.

Quels sont les SGBDR les plus connus ?

PostgreSQL	PostgreSQL est la base de données à utiliser pour les gros projets. Stable et très puissant, il permet de gérer des Gigabytes de données sans problème.
MySQL	Mysql est l'un des SGBDR les plus utilisés au monde. Il est gratuit et très puissant. Il possède la double licence GPL et propriétaire depuis son rachat par <i>Sun Microsystem</i> eux-mêmes racheté par <i>Oracle</i> (concurrent direct de MySQL). Le logiciel reste cependant entièrement gratuit et libre. Il répond à une logique client/serveur , c'est à dire que plusieurs clients (ordinateurs distants) peuvent se connecter sur un seul serveur qui héberge les données.
MariaDB	Le créateur de MySQL a créé MariaDB suite au rachat de MySQL pour continuer le projet en open source.
SQLite	SQLite est une bibliothèque écrite en C . SQLite est parfait pour les petits projets. Sa particularité est d'être intégré directement à un programme et ne répond donc pas à la logique client-serveur. Il est le moteur de base de données le plus distribué au monde puisqu'il est intégré à de nombreux logiciels grand public comme Firefox, Skype, Adobe, etc. Le logiciel pèse moins de 300 ko et peut donc être intégré à des projets tournant sur de petites supports comme les smartphones. Souvent aucune installation n'est nécessaire pour l'utiliser.
Oracle	Oracle Database est sous licence propriétaire, c'est à dire payant. Il est souvent utilisé pour les projets à gros budget nécessitant de réaliser des actions complexes.
Microsoft SQL Server	Produit <i>Microsoft</i> ne tourne que sur un OS Windows, payant n'apporte rien de plus que les logiciels concurrents libre de droit. Si vous avez trop d'argent à la limite...

SQLite



Notre premier exemple concernera SQLite.

SQLite a été conçu pour être intégré dans le programme même. Pour des projets plus ambitieux par exemple un projets web, le choix de MySQL serait plus judicieux.

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: "/data/ma_base.db" . Il vous faudra cependant vérifier que le dossier existe avant de l'utiliser.

Il est également possible de travailler avec une base de données de manière temporaire:

```
conn = sqlite3.connect(':memory:')
```

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

```
db.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 manière 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()
```

Gestion des erreurs

Il est recommandé de toujours encadrer les opérations sur des bases de données et d'anticiper des erreurs:

```
import sqlite3

try:

    conn = sqlite3.connect('data/users.db')

    cursor = conn.cursor()

    cursor.execute("""
CREATE TABLE users(

    id INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,

    name TEXT,

    age INTERGER
)
""")

    conn.commit()

except sqlite3.OperationalError:

    print('Erreur la table existe déjà')

except Exception as e:

    print("Erreur")

    conn.rollback()

    # raise e

finally:

    conn.close()
```

Les erreurs que vous pouvez intercepter :

Error

DatabaseError

DataError

IntegrityError

InternalError

NotSupportedError

OperationalError

ProgrammingError

InterfaceError

Warning

MySQL



MySQL est le logiciel idéal pour vos projets de sites web.

Contrairement à SQLite, il est nécessaire de l'installer et de le configurer.

Installation de MySQL pour python

Tout d'abord il vous faudra installer le packet **mysql-server**

```
sudo apt-get install mysql-server
```

Un mot de passe super-utilisateur vous sera demandé lors de l'installation.

Une fois l'installation terminée, il vous faudra créer une base de données. Pour cela vous devez vous connecter à MySQL:

```
mysql -u root -p
```

On vous demandera votre mot de passe.

Une fois la **console MySQL** ouverte, vous pouvez créer votre base de données.

Dans notre cas, nous la nommerons **test1** :

```
mysql> CREATE DATABASE test1;  
  
Query OK, 1 row affected (0.00 sec)
```

Il faudra installer les packets suivants:

```
sudo apt-get install python-pip libmysqlclient-dev python-dev python-mysqldb
```

Et la librairie MySQL:

```
pip install mysql  
  
pip install MySQL-python  
  
pip install mysql-connector-python --allow-external mysql-connector-python
```

Pour python 3 il vous faudra installer le packet **python3-mysql.connector** :

```
sudo apt-get install python3-mysql.connector
```

Connexion au serveur MySQL

```
import mysql.connector

conn = mysql.connector.connect(host="localhost",user="root",password="XXX", database="test1")

cursor = conn.cursor()

conn.close()
```

Cr  er une table de donn  es MySQL

```
cursor.execute("""

CREATE TABLE IF NOT EXISTS visiteurs (

    id int(5) NOT NULL AUTO_INCREMENT,

    name varchar(50) DEFAULT NULL,

    age INTEGER DEFAULT NULL,

    PRIMARY KEY(id)

);

""")
```

Ins  rer des donn  es

Il existe deux mani  res d'ajouter des donn  es dans une table:

```
user = ("olivier", "34")

cursor.execute("""INSERT INTO users (name, age) VALUES(%s, %s)""", user)

user = {"name": "olivier", "age" : "34"}

cursor.execute("""INSERT INTO users (name, age) VALUES(%(name)s, %(age)s)""", user
)
```

Trouver des données avec mysql

La logique est la même que pour SQLite:

```
cursor.execute("""SELECT id, name, age FROM users WHERE id = %s""", ("5", ))
rows = cursor.fetchall()

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

Pensez à consulter l'aide des packages pour approfondir vos connaissances:

```
help(sqlite3)
help(mysql.connector)
```