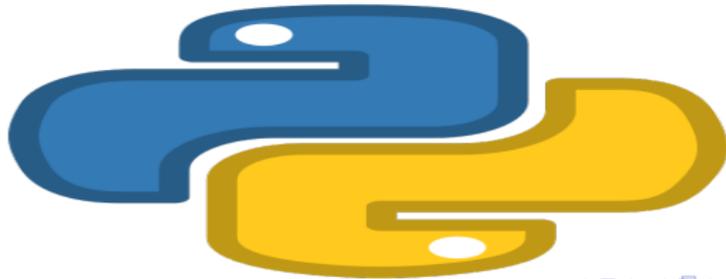


Python : connexion à une base MySQL

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

`elmouelhi.achref@gmail.com`



- 1 Introduction
 - Installation
 - Préparation d'un projet
 - Préparation d'une base de données de test
- 2 Utilisation
 - Connexion
 - SELECT
 - Requêtes préparées
 - INSERT
 - DELETE
- 3 Restructuration du code

Pour se connecter à une base de données

- Il nous faut un driver (qui varie selon le **SGBD** utilisé)
- **SGBD** : **S**ystème de **G**estion de **B**ases de **D**onnées

© Achref EL MOU

Python

Pour se connecter à une base de données

- Il nous faut un driver (qui varie selon le **SGBD** utilisé)
- **SGBD** : **S**ystème de **G**estion de **B**ases de **D**onnées

Driver ?

- Une **API** (interface d'application)
- Facilitant l'accès aux données gérées par un **SGBD**

Packages Python pour se connecter à une base de données MySQL

● MySQL Connector/Python :

- Connecteur officiel fourni par **Oracle** pour **MySQL**.
- Toujours compatible avec les versions récentes de **MySQL** et ne nécessite pas de dépendances externes.

● mysqlclient :

- Très performant grâce à son implémentation en **C**.
- Nécessite la bibliothèque client **MySQL** installée sur le système.

● PyMySQL :

- Bibliothèque écrite en pur **Python**, facile à installer et sans dépendances externes.
- Moins performant que **mysqlclient** et moins riche en fonctionnalités que **MySQL Connector/Python**.

Python

Installation avec PIP

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

Python

Démarche

- Créez un répertoire `python_mysql` dans votre espace de travail
- Lancez **VSC** et allez dans `File > Open Folder...` et choisissez `python_mysql`
- Dans `python_mysql`, créez un fichier `main.py`

Python

Voici le script SQL qui permet de créer la base de données utilisée dans ce cours

```
CREATE DATABASE courspython;

USE courspython;

CREATE TABLE personne (
num INT PRIMARY KEY AUTO_INCREMENT,
nom VARCHAR(30),
prenom VARCHAR(30)
);

SHOW TABLES;

INSERT INTO personne (nom, prenom) VALUES ("Wick", "John"),
("Dalton", "Jack");

SELECT * FROM personne;
```

Quatre étapes

- Établir la connexion avec la base de données
- Créer et exécuter des requêtes **SQL**
- Récupérer le résultat
- Fermer la connexion

Python

Se connecter à la base de données

- le nom de l'hôte sur lequel le serveur **MySQL** est installé (dans notre cas localhost ou 127.0.0.1)
- le port TCP/IP utilisé par **MySQL** (par défaut 3306 ou 3308)
- le nom de la base de données **MySQL**
- le nom d'utilisateur (par défaut `root` pour **MySQL**)
- le mot de passe

Python

Importons le module de connexion

```
import mysql.connector
```

© Achref EL MOUELHI ©

Python

Importons le module de connexion

```
import mysql.connector
```

Établissons la connexion avec la base de données

```
connexion = mysql.connector.connect (  
    host="localhost",  
    user="root",  
    password="",  
    database="courspython",  
    port=3306  
)
```

Python

Importons le module de connexion

```
import mysql.connector
```

Établissons la connexion avec la base de données

```
connexion = mysql.connector.connect (  
    host="localhost",  
    user="root",  
    password="",  
    database="courspython",  
    port=3306  
)
```

Vérifions l'état de la connexion

```
if connexion.is_connected():  
    print("Connexion réussie à la base de données MySQL")
```

Python

Créons la requête

```
request = "SELECT * FROM personne"
```

© Achref EL MOUELHI ©

Python

Créons la requête

```
request = "SELECT * FROM personne"
```

Exécutons la requête

```
curseur = connexion.cursor()  
curseur.execute(request)
```

Python

Créons la requête

```
request = "SELECT * FROM personne"
```

Exécutons la requête

```
curseur = connexion.cursor()  
curseur.execute(request)
```

Récupérons le résultat

```
personnes = curseur.fetchall()
```

Python

Affichons le résultat

```
for personne in personnes:  
    print (personne)
```

```
# résultat  
# (1, 'Wick', 'John')  
# (2, 'Dalton', 'Jack')
```

© Achre

Python

Affichons le résultat

```
for personne in personnes:  
    print (personne)
```

```
# résultat  
# (1, 'Wick', 'John')  
# (2, 'Dalton', 'Jack')
```

Et enfin fermons la connexion

```
connexion.close()  
curseur.close()
```

Python

Les instructions d'accès aux données peuvent lever une exception. Ajoutons donc `try ... except ... finally`

```
import mysql.connector
from mysql.connector import Error

try:
    connexion = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="courspython",
        port=3306
    )
    request = "SELECT * FROM Personne"

    curseur = connexion.cursor()
    curseur.execute(request)

    personnes = curseur.fetchall()
    for personne in personnes:
        print(personne)

except Error as e:
    print("Exception : ", e)

finally:
    if (connexion.is_connected()):
        connexion.close()
        curseur.close()
    print("La connexion à MySQL est désormais fermée")
```

Python

Pour afficher certains champs

```
for personne in personnes:  
    print(personne[1], personne[2])  
  
# affiche  
# Wick John  
# Dalton Jack  
# La connexion à MySQL est désormais fermée
```

Python

Les instructions d'accès aux données peuvent lever une exception. Ajoutons donc `try ... except ... finally`

```
import mysql.connector
from mysql.connector import Error

try:
    connexion = mysql.connector.connect (
        host="localhost",
        user="root",
        password="",
        database="courspython",
        port=3308
    )
    request = "SELECT * FROM Personne"

    curseur = connexion.cursor(dictionary=True)
    curseur.execute(request)

    personnes = curseur.fetchall()
    for personne in personnes:
        print (personne['prenom'], personne['nom'])

except Error as e:
    print ("Exception : ", e)

finally:
    if (connexion.is_connected()):
        connexion.close()
        curseur.close()
    print ("La connexion à MySQL est désormais fermée")
```

Python

Pour indiquer le nombre de tuples à récupérer, on utilise
`fetchmany(nombre)`

```
personnes = curseur.fetchmany(1)

for personne in personnes:
    print(personne[1], personne[2])

# affiche
# Wick John
```

Python

Pour récupérer seulement le premier tuple, on utilise `fetchone()`

```
personne = curseur.fetchone()

print(personne[1], personne[2])

# affiche
# Wick John
```

Python

Pour récupérer un tuple selon la valeur d'une colonne (num = 2 par exemple)

```
request = "SELECT * FROM Personne WHERE num = 2"
```

```
curseur = connexion.cursor()
```

```
curseur.execute(request)
```

```
personne = curseur.fetchone()
```

```
print(personne[1], personne[2])
```

```
# affiche
```

```
# Dalton Jack
```

Python

Ou en utilisant les requêtes paramétrées

```
request = "SELECT * FROM Personne WHERE num = %s"

curseur = connexion.cursor()

values = [2]
curseur.execute(request, values)

personne = curseur.fetchone()

print(personne[1], personne[2])

# affiche
# Dalton Jack
```

Python

Pour faire une insertion

```
request = "INSERT INTO personne (nom, prenom) VALUES (%s, %s)"

tuple = ('muller', 'thomas')
curseur = connexion.cursor()
curseur.execute(request, tuple)

connexion.commit()
print("Tuple inséré avec succès dans la table personne")
```

Remarques

- `connexion.commit()` est indispensable pour valider la requête
- Pour annuler la requête, on peut utiliser `connexion.rollback()`

Python

Pour récupérer la valeur de la clé primaire auto-générée

```
request = "INSERT INTO personne (nom, prenom) VALUES (%s, %s)"

tuple = ('muller', 'thomas')
curseur = connexion.cursor()
curseur.execute(request, tuple)

connexion.commit()
print("Tuple inséré avec succès avec id = ", curseur.lastrowid)
```

Python

Pour une insertion multiple

```
request = "INSERT INTO personne (nom, prenom) VALUES (%s, %s)"

tuples = [('muller', 'thomas'), ('ribery', 'franc')]
curseur = connexion.cursor()
curseur.executemany(request, tuples)

connexion.commit()
print(curseur.rowcount, "tuple(s) inséré(s) avec succès")
```

Python

Exercice

Pour terminer le CRUD, faites `UPDATE` et `DELETE`.

© Achre... HI ©

Python

Pour faire une suppression

```
import mysql.connector
from mysql.connector import Error
connexion = None
try:
    connexion = mysql.connector.connect(
        host="localhost",
        user="root",
        password="",
        database="courspython",
        port=3306
    )

    print('connexion établie avec succès')
    sql = "DELETE FROM personne WHERE nom=%s"
    params = ('Mitroglou',)
    curseur = connexion.cursor()
    curseur.execute(sql, params)

    connexion.commit()
    print(f"Tuple supprimé avec succès ")

except Error as e:
    print(f"Problème de connexion : {e}")

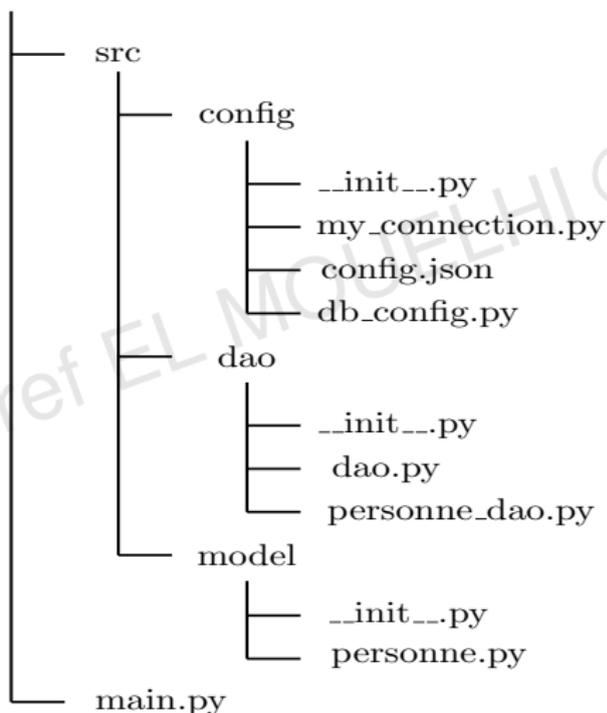
finally:
    if connexion != None and connexion.is_connected():
        connexion.close()
        curseur.close()
    print(f"Connexion terminée avec MySQL")
```

Organisation du code

- Placer toutes les données (nomBaseDeDonnées, nomUtilisateur, motDePasse...) dans un fichier **JSON** et les charger dans une classe `MyConnection`
- Pour chaque table de la base de données, définir une classe **Python** (appelée `model`) ayant comme attributs les colonnes de cette table
- Placer tout le code correspondant à l'accès aux données (de la base de données) dans des nouvelles classes (qui constitueront la couche **DAO : Data Access Object**)

Python

cours-python-mysql



Arborescence du projet (.__init__.py n'est plus nécessaire depuis **Python 3.3**)

Python

Le fichier `config.json`

```
{
  "mysql": {
    "host": "localhost",
    "user": "root",
    "password": "",
    "db": "courspython",
    "port": 3306
  }
}
```

Python

Le fichier `db_config.py` qui charge les données définies dans `config.json`

```
import json

def load():
    with open('src/config/config.json') as f:
        data = json.load(f)
        return data['mysql']
```

Python

La classe MyConnection

```
import mysql.connector as pymysql
from src.config.db_config import load

class MyConnection:

    __connection = None
    __cursor = None

    def __init__(self):
        data_source = load()
        self.__connection = pymysql.connect(**data_source)
        self.__cursor = self.__connection.cursor()

    def query(self, query, params):
        self.__cursor.execute(query, params)
        return self.__cursor

    def close(self):
        self.__connection.close()
```

La classe `Personne`

```
class Personne:
    def __init__(self, num: int = 0, nom: str = '', prenom: str = ''):
        self.num = num
        self._nom = nom
        self._prenom = prenom

    @property
    def num(self) -> int:
        return self._num

    @num.setter
    def num(self, num) -> None:
        self._num = num if num > 0 else 0

    @property
    def nom(self) -> str:
        return self._nom

    @nom.setter
    def nom(self, nom) -> None:
        self._nom = nom

    @property
    def prenom(self) -> str:
        return self._prenom

    @prenom.setter
    def prenom(self, prenom) -> None:
        self._prenom = prenom

    def __str__(self) -> str:
        return f'{self._num} {self._prenom} {self._nom}'
```

Python

La classe abstraite `Dao`

```
from typing import TypeVar, Generic, Iterable
from abc import ABC, abstractmethod

T = TypeVar('T')

class Dao (Generic[T], ABC):

    @abstractmethod
    def save(self, t: T) -> T:
        pass

    @abstractmethod
    def find_all(self) -> Iterable:
        pass

    @abstractmethod
    def find_by_id(self, t: int) -> T:
        pass

    @abstractmethod
    def update(self, t: T) -> T:
        pass

    @abstractmethod
    def remove(self, t: T) -> None:
        pass
```

Python

La classe `PersonneDao` qui implémente `Dao[Personne]`

```
from ..config.my_connection import MyConnection
from ..model.personne import Personne
from src.dao.dao import Dao

class PersonneDao (Dao[Personne]):

    __db = None

    def __init__(self):
        self.__db = MyConnection()

    def find_all(self):
        donnees_recuperees = self.__db.query('SELECT * FROM personne', None).fetchall()
        personnes = [Personne(num, nom, prenom) for num, nom, prenom in donnees_recuperees]
        return personnes

    def save(self, personne: Personne) -> Personne:
        pass

    def find_by_id(self, t: int) -> Personne:
        pass

    def update(self, t: Personne) -> Personne:
        pass

    def remove(self, t: Personne) -> None:
        pass
```

Le `main.py` pour tester toutes ces classes

```
from mysql.connector import Error
from src.dao.personne_dao import PersonneDao

try:
    personneDao = PersonneDao()
    for personne in personneDao.find_all():
        print(personne)

except Error as e:
    print("Exception : ", e)
```

Remarque

N'oublions pas d'implémenter les quatre autres méthodes de la classe abstraite `Dao`