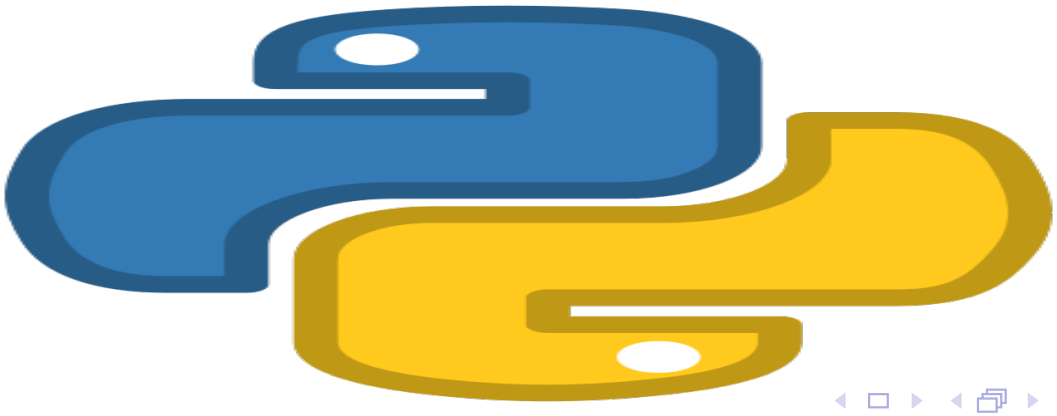
Python : connexion à une base **MySQL**

Achref El Mouelhi

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

elmouelhi.achref@gmail.com



Plan

Python

Pour se connecter à une base de données

- Il nous faut un driver (qui varie selon le SGBD utilisé)
- SGBD : Système de Gestion de Bases de Données

Python

Pour se connecter à une base de données

- Il nous faut un driver (qui varie selon le SGBD utilisé)
- SGBD : Système de Gestion de Bases de Données

Driver ?

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

Python

Téléchargement et installation

- Définissez une variable d'environnement pour `pip` (défini dans `C:\Users\elmou\AppData\Local\Programs\Python\Python38-32\Scripts`)
- Redémarrez la console
- Lancez la commande `pip install mysql-connector-python`

Python

Démarche

- Créez un répertoire `cours-python-mysql` dans votre espace de travail
- Lancez **VSC** et allez dans `File > Open Folder...` et choisissez `cours-python-mysql`
- Dans `cours-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;
```

Python

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
```

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=3308
)
```

Python

Créons la requête

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

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')
```

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=3308
  )
  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()

# virgule obligatoire
tuple = (2,)
curseur.execute(request, tuple)

personne = curseur.fetchone()

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

# affiche
# Dalton Jack
```

Python

Pour sélectionner seulement quelques champs d'un tuple

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

curseur = connexion.cursor()

# virgule obligatoire
tuple = (2,)
curseur.execute(request, tuple)

nom = curseur.fetchone()

print(nom[0])

# affiche
# Dalton
```

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")
```

Python

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 dans la table
   personne 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
   dans la table personne", curseur.lastrowid)
```

Python

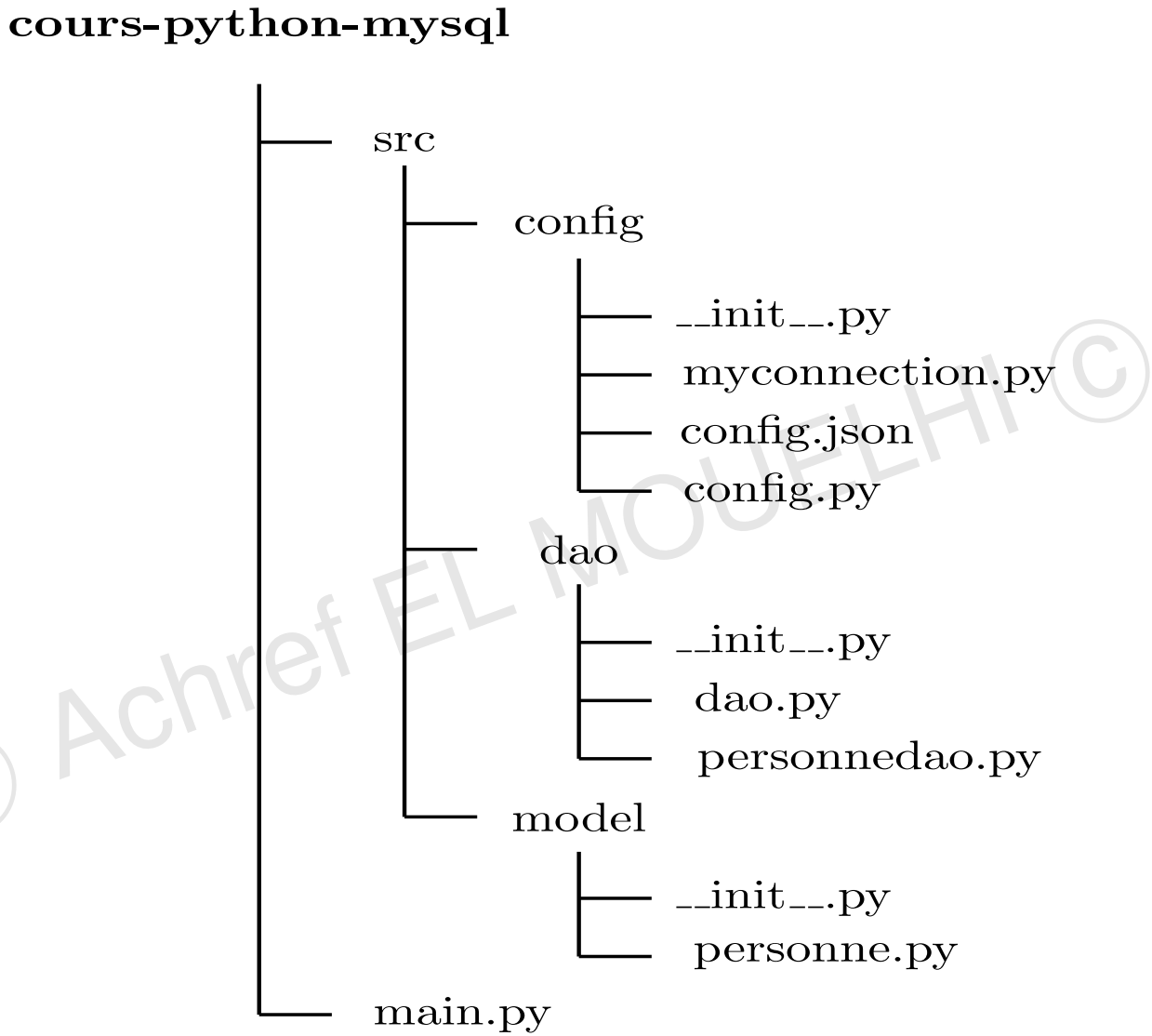
Exercice

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

Python

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)



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": 3308
    }
}
```

Python

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

```
import json

config = {}

with open('src/config/config.json') as f:
    config = json.load(f)
```

Python

La classe `MyConnection`

```
import mysql.connector as pymysql
from mysql.connector import Error
from .config import config


class MyConnection:

    __connection = None
    __cursor = None

    def __init__(self):
        __db_config = config['mysql']
        self.__connection = pymysql.connect(host=__db_config['host'],
                                            user = __db_config['user'],
                                            password = __db_config['password'],
                                            db = __db_config['db'],
                                            port=__db_config['port'])
        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:
        if num > 0:
           self._num = num
        else:
            self._num = 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
    @prenom.deleter
    def prenom(self) -> str:
        del self._prenom
    def __str__(self) -> str:
        return str(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 findAll(self) -> Iterable:
        pass
    @abstractmethod
    def findById(self, t: int) -> T:
        pass
    @abstractmethod
    def update(self, t: T) -> T:
        pass
    @abstractmethod
    def remove(self, t: T) -> None:
        pass
```

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

```
from ..config.myconnection import MyConnection
from ..model.personne import Personne
from .dao import Dao

class PersonneDao (Dao[Personne]):
    __db = None

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

    def findAll(self):
        return self.__db.query("SELECT * FROM personne", None).fetchall
            ()
    def save(self, personne: Personne) -> Personne:
        pass

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

Python

Le `main.py` pour tester toutes ces classes

```
from mysql.connector import Error
from src.dao.personnedao import PersonneDao
from src.model.personne import Personne

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

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

Python

Remarque

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