Komunikace s databázovým systémem - Připojení, Ukládání a načítání dat, Mapování entit v OOP

Připojení
Základní požadavky k tomu, abychom se mohli vzdáleně připojit na databázi pomocí Pythonu.

Nutné parametry k připojení do databáze:

USERNAME - uživatel
PASSWORD - heslo
HOSTNAME - adresa
PORT - 3306 pro MYSQL
DATABASE - název konkrétní databáze na serveru

Postup:

1. instalace balíčku, mysql

   pip install mysql-connector-python

2. Import pro připojení

```
import mysql.connector

connection = mysql.connector.connect(
   host = "localhost",
   user = "root",
   passwd = "<PASSWORD>",
   database = "mydatabase"
   port = 3306
)
```

Údaje o připojení můžeme zadat přímo v kódu, nebo vytvořit .env soubor pro snažší konfiguraci. Pro .env soubory je nutné ještě dodatečně nainstalovat knihovnu python-dotenv

```
import os

import mysql.connector
from dotenv import load_dotenv

load_dotenv()

conn = mysql.connector.connect(
   host=os.getenv("host"),
   user=os.getenv("user"),
```

```
    password=os.getenv("password"),
    database=os.getenv("database")
    port = os.getenv("port")
)
```

Ukládání a načítání dat
Databáze používáme pro ukládání, čtení a správu dat. Díky tomu, že ukládáme data do databáze, tak naše data jsou bezpečněji a vhodněji uložená, než třeba v obyčejném souboru. Ve srovnání s ukládáním do souborů, databáze má několik výhod oproti souborům - Rychlost (indexy), možnost multiuživatelského připojení a lepší strukturovanost a integrovanost dat (tabulky, datové typy, relace….)

Máme 2 základní způsoby, jak s databází pracovat:

1. příme SQL dotazy v kódu
2. pomocí SQLAlchemy (ORM) - přístup přes třídy a objekty

Pro spuštění sql příkazů potřebujeme tzv. kurzor, který se vytváří ze spojení. Je dobrý si ten kurzor přestavit jako takovej prst, kterým se ukazuje na místo někde v databázi, se kterým chcem pracovat. Tento nástroj nám umožňuje:
Provádět SQL příkazy (execute…)
Získávat výstupy příkazů (fetchall)
Zapisovat změny (Commit)

cursor = connection.cursor()
(nezapomínat uzavírat kurzor a připojení)

Create

```
# Create - vytvoreni tabulky

cursor = connection.cursor()

table_name = 'user'
cursor.execute("""
    CREATE TABLE IF NOT EXISTS studenti (
        id INT AUTO_INCREMENT PRIMARY KEY,
        jmeno VARCHAR(50) NOT NULL,
        vek INT
    )
""")

cursor.close()
connection.close
```

Vložení dat

```
# Vlozeni dat

cursor = connection.cursor()

insert_query = "Insert into studenti (jmeno, vek) values (%s, %s)"
values = ("Karel", 13)

cursor.execute(insert_query, values)
connection.commit()
cursor.close()
connection.close()
```

Načtení dat

```
# Vlozeni dat

cursor = connection.cursor()

select_query = "select * from studenti"
result = cursor.fetchall()

for studenti in result:
   print(studenti)

cursor.close()
connection.close()
```

Mapování entit v OOP
Možnost abstrakce jednotlivých tabulek v databázi lze napodobit jako třídy. Každý sloupec v tabulce je atribut, každý řádek v tabulce odpovídá jednomu objektu třídy. Umožňuje nám to přirozenější práci s daty, lepší organizaci kódu, oddělení logiky, lehká údržba…..

SQLAlchemy - Převádí databázové tabulky na Python objekty, Nahrazuje složité SQL dotazy jednoduchými Python příkazy, Automaticky řeší vztahy mezi tabulkami
pip install SQLAlchemy

2 vrstvy
Core - nízkoúrovňová část, pracuje s přímo s SQL
ORM - vysokoúrovňová část, objektově-relační mapování

Importování
create_engine - slouží pro navázaní spojení s databází, všechna komunikace probíha přes engine
declarative_base - základní třída, ze které dědí ostatní třídy
sessionmaker - komunikační most mezi námi a databází

from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import declarative_base, sessionmaker

```python
# Import - přesně jak je v obrázku
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import declarative_base, sessionmaker

# 1. Engine - připojení k databázi
engine = create_engine('sqlite:///skola.db')  # SQLite pro jednoduchost

# 2. Base - základ pro všechny modely
Base = declarative_base()


# 3. Model - třída reprezentující tabulku
class Student(Base):
    __tablename__ = 'studenti'

    id = Column(Integer, primary_key=True)
    jmeno = Column(String)
    vek = Column(Integer)


# 4. Vytvoření tabulek
Base.metadata.create_all(engine)
```

```python
# 5. Session - komunikační most
Session = sessionmaker(bind=engine)
session = Session()

# 6. Použití - přidání studenta
novy_student = Student(jmeno="Jan", vek=18)
session.add(novy_student)
session.commit()

# 7. Jednoduchý dotaz
studenti = session.query(Student).all()
for student in studenti:
   print(f"{student.jmeno} je {student.vek} let starý")
```

Návrhové vzory pro práci s databází
Návrhové vzory ulehčují prací s databází, zpřehledňují kód, odděleují logiku.

<u>DAO (data access object)</u>
Dvě třídy:
Model (Student) - obsahuje data a bussiness logiku
DAO (StudentDAO) - obsahuje všechny databázové operace pro daný model, typicky create(), read(), update(), delete()

```python
# Čistý model - neví nic o databázi
class Student:
   def __init__(self, id=None, jmeno=None, prijmeni=None, vek=None):
       self.id = id
       self.jmeno = jmeno
       self.prijmeni = prijmeni
       self.vek = vek

   def je_plnolety(self):
       return self.vek >= 18


# DAO - stará se o všechny databázové operace
class StudentDAO:
   def __init__(self, connection):
       self.conn = connection

   def create(self, student):
       """Vytvoří nového studenta v databázi"""
       cursor = self.conn.cursor()
       cursor.execute("""
           INSERT INTO studenti (jmeno, prijmeni, vek)
           VALUES (%s, %s, %s)
       """, (student.jmeno, student.prijmeni, student.vek))
```

```
        self.conn.commit()
        student.id = cursor.lastrowid
        return student
```