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

## Připojení

Mysql database beží defaultně na portu 3306.

Pro přihlášení na školní počítač do mysql:

* USERNAME: root
* PASSWORD: student
* HOSTNAME: 127.0.0.1
* PORT: 3306

**pip install mysql-connector-python**

**import** **mysql.connector**

**# Vytvoření připojení k databázi**

cnx = mysql.connector.connect(user='jmeno\_uzivatele', password='heslo',

host='adresa\_databaze',database='jmeno\_databaze')

**# Vytvoření kurzoru pro práci s databází**

cursor = cnx.cursor()

table\_name=”user”

query = (f"CREATE TABLE IF NOT EXISTS {table\_name} (id INT PRIMARY KEY AUTO\_INCREMENT, jmeno VARCHAR(255), vek INT)")

cursor.execute(query)

# pokud bysme chtěli ukončit connection

cursor.close()

cnx.close()

## Ukládání načítání dat

# pokud bysme chtěli ukončit connection

\_\_INSERT\_USER\_SQL = 'insert into user(jmeno, vek) values (%s,%s)'

\_\_GET\_PRODUCT\_SQL = 'select \* from user'

cursor.execute(\_\_INSERT\_PRODUCT\_SQL,(“Pepa”,”34”))

cursor.execute(\_\_GET\_PRODUCT\_SQL)

for value in self.cursor:

print(value)

## Mapování entit v OOP

### Fasáda:

Fasáda (anglicky Facade) je softwarový návrhový vzor, který slouží ke zjednodušení komunikace mezi uživatelem a systémem. Použití je výhodné, pokud je tento systém příliš komplexní (obsahuje mnoho tříd a vazeb) pro splnění dané oblasti úloh, jež po něm vyžadují uživatelé. Fasáda je způsob nahrazení velkého počtu rozhraní subsystémů, sjednoceným rozhraním, které bude zaštitovat všechna rozhraní subsystémů.

import mysql.connector

class UserDatabase:

def \_\_init\_\_(self, host, user, password, database):

self.connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database

)

self.cursor = self.connection.cursor()

def create\_user(self, name, email):

sql = "INSERT INTO users (name, email) VALUES (%s, %s)"

values = (name, email)

self.cursor.execute(sql, values)

self.connection.commit()

return self.cursor.lastrowid

def get\_user(self, id):

sql = "SELECT \* FROM users WHERE id = %s"

values = (id,)

self.cursor.execute(sql, values)

return self.cursor.fetchone()

def update\_user(self, id, name=None, email=None):

sql = "UPDATE users SET "

values = []

if name is not None:

sql += "name = %s, "

values.append(name)

if email is not None:

sql += "email = %s, "

values.append(email)

sql = sql[:-2] + " WHERE id = %s"

values.append(id)

self.cursor.execute(sql, values)

self.connection.commit()

return self.cursor.rowcount > 0

def delete\_user(self, id):

sql = "DELETE FROM users WHERE id = %s"

values = (id,)

self.cursor.execute(sql, values)

self.connection.commit()

return self.cursor.rowcount > 0

def \_\_del\_\_(self):

self.connection.close()

## Repository

Repository je podobná jako DAO, ale pojmenováním metod je blíž kolekci. Dokonce může mít DAO uvnitř sebe a využívat ho. Typické je třeba přetížení hranatých závorek [] apod.

import mysql.connector

class UserRepository:

def \_\_init\_\_(self, host, user, password, database):

self.connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database

)

def get(self, id):

sql = "SELECT \* FROM users WHERE id = %s"

values = (id,)

cursor = self.connection.cursor(dictionary=True)

cursor.execute(sql, values)

user = cursor.fetchone()

cursor.close()

return user

def find(self, name):

sql = "SELECT \* FROM users WHERE name LIKE %s"

values = (f"%{name}%",)

cursor = self.connection.cursor(dictionary=True)

cursor.execute(sql, values)

users = cursor.fetchall()

cursor.close()

return users

def add(self, name, email):

sql = "INSERT INTO users (name, email) VALUES (%s, %s)"

values = (name, email)

cursor = self.connection.cursor()

cursor.execute(sql, values)

user\_id = cursor.lastrowid

self.connection.commit()

cursor.close()

return user\_id

def update(self, id, name, email):

sql = "UPDATE users SET name = %s, email = %s WHERE id = %s"

values = (name, email, id)

cursor = self.connection.cursor()

cursor.execute(sql, values)

self.connection.commit()

cursor.close()

def delete(self, id):

sql = "DELETE FROM users WHERE id = %s"

values = (id,)

cursor = self.connection.cursor()

cursor.execute(sql, values)

self.connection.commit()

cursor.close()

def \_\_del\_\_(self):

self.connection.close()

### Table Gateway

Table Gateway pattern se používá pro práci s celými tabulkami v databázi. Tento vzor vytváří objekt pro každou tabulku v databázi a poskytuje metody pro čtení a zápis dat do této tabulky. Objekt Table Gateway může také poskytnout metody pro vytváření, mazání a aktualizaci záznamů v této tabulce.

import mysql.connector

class UserGateway:

def \_\_init\_\_(self, host, user, password, database):

self.connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database

)

self.cursor = self.connection.cursor()

def get\_all\_users(self):

sql = "SELECT \* FROM users"

self.cursor.execute(sql)

return self.cursor.fetchall()

def get\_user\_by\_id(self, id):

sql = "SELECT \* FROM users WHERE id = %s"

values = (id,)

self.cursor.execute(sql, values)

return self.cursor.fetchone()

def get\_user\_by\_email(self, email):

sql = "SELECT \* FROM users WHERE email = %s"

values = (email,)

self.cursor.execute(sql, values)

return self.cursor.fetchone()

def insert\_user(self, name, email):

sql = "INSERT INTO users (name, email) VALUES (%s, %s)"

values = (name, email)

self.cursor.execute(sql, values)

self.connection.commit()

return self.cursor.lastrowid

def update\_user(self, id, name=None, email=None):

sql = "UPDATE users SET "

values = []

if name is not None:

sql += "name = %s, "

values.append(name)

if email is not None:

sql += "email = %s, "

values.append(email)

sql = sql[:-2] + " WHERE id = %s"

values.append(id)

self.cursor.execute(sql, values)

self.connection.commit()

return self.cursor.rowcount > 0

def delete\_user(self, id):

sql = "DELETE FROM users WHERE id = %s"

values = (id,)

self.cursor.execute(sql, values)

self.connection.commit()

return self.cursor.rowcount > 0

def \_\_del\_\_(self):

self.connection.close()

### Row Gateway

Row Gateway pattern se používá pro práci s řádky (záznamy) v databázi. Tento vzor vytváří objekt pro každý záznam v databázi a poskytuje metody pro čtení a zápis jednotlivých sloupců v řádku. Objekt Row Gateway může také poskytnout metody pro vytváření, mazání a aktualizaci řádků v databázi.

import mysql.connector

class UserRowGateway:

def \_\_init\_\_(self, host, user, password, database, id):

self.connection = mysql.connector.connect(

host=host,

user=user,

password=password,

database=database

)

self.id = id

def get\_name(self):

sql = "SELECT name FROM users WHERE id = %s"

values = (self.id,)

cursor = self.connection.cursor()

cursor.execute(sql, values)

name = cursor.fetchone()[0]

cursor.close()

return name

def set\_name(self, name):

sql = "UPDATE users SET name = %s WHERE id = %s"

values = (name, self.id)

cursor = self.connection.cursor()

cursor.execute(sql, values)

self.connection.commit()

cursor.close()

def get\_email(self):

sql = "SELECT email FROM users WHERE id = %s"

values = (self.id,)

cursor = self.connection.cursor()

cursor.execute(sql, values)

email = cursor.fetchone()[0]

cursor.close()

return email

def set\_email(self, email):

sql = "UPDATE users SET email = %s WHERE id = %s"

values = (email, self.id)

cursor = self.connection.cursor()

cursor.execute(sql, values)

self.connection.commit()

cursor.close()

def delete(self):

sql = "DELETE FROM users WHERE id = %s"

values = (self.id,)

cursor = self.connection.cursor()

cursor.execute(sql, values)

self.connection.commit()

cursor.close()

def \_\_del\_\_(self):

self.connection.close()

Table Gateway vs Row Gateway

Hlavní rozdíl mezi Row Gateway a Table Gateway spočívá tedy v tom, že Row Gateway pracuje s jedním řádkem v tabulce a poskytuje metody pro manipulaci s ním, zatímco Table Gateway pracuje s celou tabulkou najednou a poskytuje metody pro manipulaci s celými daty v této tabulce. Při použití jednoho nebo druhého vzoru závisí na konkrétních požadavcích aplikace a datové struktury v databázi.

## Sql injection

SQL Injection je technika útoku, kdy útočník využívá nezabezpečených vstupních polí v webových aplikacích, aby injektoval kód SQL do databáze a získal neoprávněný přístup k datům. Jedním z nejběžnějších způsobů, jak se bránit proti SQL Injection, je sanitizace vstupů.

Sanitizace vstupů spočívá v ověření a čištění vstupů od nebezpečného kódu a nepřípustných znaků. Jeden z nástrojů, který může být použit pro sanitizaci vstupů, je regulární výraz (regex).

import re

# Sanitize input string to avoid SQL injection

def sanitize\_input(input\_str):

sanitized\_str = re.sub(r'[;\'\"]', '', input\_str) # Remove semicolons, single quotes and double quotes

return sanitized\_str

# Example usage

user\_input = "John'; DROP TABLE users;--" # Input containing SQL injection attempt

sanitized\_input = sanitize\_input(user\_input) # Sanitize input

sql\_query = "SELECT \* FROM users WHERE username = '{0}'".format(sanitized\_input) # Generate SQL query with sanitized input