06

Integrando aplicação e banco de dados

Transcrição

Antes de prosseguirmos, vamos dar uma olhada nos módulos que instalamos no vídeo anterior, começando pelo `prepara_banco.py` :

```
import MySQLdb
print('Conectando...')
conn = MySQLdb.connect(user='root', passwd='admin', host='127.0.0.1', port=3306)

# Descomente se quiser desfazer o banco...
conn.cursor().execute("DROP DATABASE `jogoteca`;")
conn.commit()

criar_tabelas = '''SET NAMES utf8;
    CREATE DATABASE `jogoteca` /*!40100 DEFAULT CHARACTER SET utf8 COLLATE utf8_bin */;
    USE `jogoteca`;
    CREATE TABLE `jogo` (
      `id` int(11) NOT NULL AUTO_INCREMENT,
      `nome` varchar(50) COLLATE utf8_bin NOT NULL,
      `categoria` varchar(40) COLLATE utf8_bin NOT NULL,
      `console` varchar(20) NOT NULL,
      PRIMARY KEY (`id`)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
    CREATE TABLE `usuario` (
      `id` varchar(8) COLLATE utf8_bin NOT NULL,
      `nome` varchar(20) COLLATE utf8_bin NOT NULL,
      `senha` varchar(8) COLLATE utf8_bin NOT NULL,
      PRIMARY KEY (`id`)
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;'''

conn.cursor().execute(criar_tabelas)

# inserindo usuarios
cursor = conn.cursor()
cursor.executemany(
      'INSERT INTO jogoteca.usuario (id, nome, senha) VALUES (%s, %s, %s)',
      [
            ('luan', 'Luan Marques', 'flask'),
            ('nico', 'Nico', '7a1'),
            ('danilo', 'Danilo', 'vegas')
      ])

cursor.execute('select * from jogoteca.usuario')
print(' -------------  Usuários:  -------------')
for user in cursor.fetchall():
    print(user[1])

# inserindo jogos
cursor.executemany(
      'INSERT INTO jogoteca.jogo (nome, categoria, console) VALUES (%s, %s, %s)',
      [
            ('God of War 4', 'Acao', 'PS4'),
```

```
            ('NBA 2k18', 'Esporte', 'Xbox One'),
            ('Rayman Legends', 'Indie', 'PS4'),
            ('Super Mario RPG', 'RPG', 'SNES'),
            ('Super Mario Kart', 'Corrida', 'SNES'),
            ('Fire Emblem Echoes', 'Estrategia', '3DS'),
    ])

cursor.execute('select * from jogoteca.jogo')
print(' -------------  Jogos:  -------------')
for jogo in cursor.fetchall():
    print(jogo[1])

# commitando senão nada tem efeito
conn.commit()
cursor.close()
```

Perceba que, em `CREATE TABLE 'jogo'` , também estamos criando uma `id` para o jogo. Dessa forma, toda tabela terá um identificador único - já que o nome de um jogo pode se repetir, por exemplo. Esse `id` tem uma propriedade especial no banco, chamada `AUTO INCREMENT` : ele começa com o valor `1` e vai incrementando conforme são criados novos jogos.

Agora veremos o módulo `dao.py` :

```
from jogoteca import Jogo, Usuario

SQL_DELETA_JOGO = 'delete from jogo where id = %s'
SQL_JOGO_POR_ID = 'SELECT id, nome, categoria, console from jogo where id = %s'
SQL_USUARIO_POR_ID = 'SELECT id, nome, senha from usuario where id = %s'
SQL_ATUALIZA_JOGO = 'UPDATE jogo SET nome=%s, categoria=%s, console=%s where id = %s'
SQL_BUSCA_JOGOS = 'SELECT id, nome, categoria, console from jogo'
SQL_CRIA_JOGO = 'INSERT into jogo (nome, categoria, console) values (%s, %s, %s)'


class JogoDao:
    def __init__(self, db):
        self.__db = db

    def salvar(self, jogo):
        cursor = self.__db.connection.cursor()

        if (jogo.id):
            cursor.execute(SQL_ATUALIZA_JOGO, (jogo.nome, jogo.categoria, jogo.console, jogo.id)
        else:
            cursor.execute(SQL_CRIA_JOGO, (jogo.nome, jogo.categoria, jogo.console))
            jogo.id = cursor.lastrowid
        self.__db.connection.commit()
        return jogo

    def listar(self):
        cursor = self.__db.connection.cursor()
        cursor.execute(SQL_BUSCA_JOGOS)
        jogos = traduz_jogos(cursor.fetchall())
        return jogos

    def busca_por_id(self, id):
```

```
        cursor = self.__db.connection.cursor()
        cursor.execute(SQL_JOGO_POR_ID, (id,))
        tupla = cursor.fetchone()
        return Jogo(tupla[1], tupla[2], tupla[3], id=tupla[0])

    def deletar(self, id):
        self.__db.connection.cursor().execute(SQL_DELETA_JOGO, (id, ))
        self.__db.connection.commit()


class UsuarioDao:
    def __init__(self, db):
        self.__db = db

    def buscar_por_id(self, id):
        cursor = self.__db.connection.cursor()
        cursor.execute(SQL_USUARIO_POR_ID, (id,))
        dados = cursor.fetchone()
        usuario = traduz_usuario(dados) if dados else None
        return usuario


def traduz_jogos(jogos):
    def cria_jogo_com_tupla(tupla):
        return Jogo(tupla[1], tupla[2], tupla[3], id=tupla[0])
    return list(map(cria_jogo_com_tupla, jogos))


def traduz_usuario(tupla):
    return Usuario(tupla[0], tupla[1], tupla[2])
```

Com esse módulo, poderemos inserir um jogo novo na nossa aplicação, sem que seja necessário salvá-lo rigidamente no nosso servidor. Nele, temos um método `salvar()` dentro da classe `JogoDao`. Esse método coloca os dados no banco de dados (além de atualizar o banco, entre outras funções que utilizaremos no futuro).

Passando os valores `jogo.nome`, `jogo.categoria` e `jogo.console` para *query* `SQL_CRIA_JOGO`, conseguimos criar um novo jogo na lista e, ao final, retornamos o jogo criado.

Para que isso seja feito, no começo do código, estamos importando a classe `Jogo` e a classe `Usuario` do módulo `jogoteca`.

Vamos observar como estamos operando a criação de novos jogos no projeto **jogoteca**:

```
@app.route('/criar', methods=['POST',])
def criar():
    nome = request.form['nome']
    categoria = request.form['categoria']
    console = request.form['console']
    jogo = Jogo(nome, categoria, console)
    lista.append(jogo)
    return redirect(url_for('index'))
```

A função `criar()` pega um objeto `jogo` e o coloca em uma lista por meio do `append()`. A partir de agora, vamos mudar essa bordagem e salvar esses dados no `dao.py`. Para isso, o primeiro passo é importarmos `JogoDao`, que é o nosso módulo de jogos para banco de dados:

```
from dao import JogoDao
```

Para utilizarmos `JogoDao`, precisaremos inicializá-lo. O `__init__` da classe `JogoDao` espera um parâmetro `db`, que é um banco de dados instanciado na nossa aplicação.

A criação desse banco de dados é bem simples. Primeiro, importaremos a classe `MySQL` de `flask_mysqldb`. Em seguida, vamos criar um objeto dessa classe, que chamaremos de `db`.

```
db = MySQL(app)
```

Na própria aplicação do Flask, é possível passar todas as configurações que servirão para conectar ao banco de dados - as mesmas informações que utilizamos em `prepara_banco.py`.

```
app.config['MYSQL_HOST'] = "0.0.0.0"
app.config['MYSQL_USER'] = "root"
app.config['MYSQL_PASSWORD'] = "admin"
app.config['MYSQL_DB'] = "jogoteca"
app.config['MYSQL_PORT'] = 3306
```

Podemos simplesmente passar a nossa aplicação (`app`), que tem essas configuração, para o `MySQL()`. Isso é um facilitador que dispensa a necessidade de inicializar o MySQL e criar uma conexão, pois já temos uma instância latente e funcional do banco de dados.

Então, criaremos um objeto `jogo_dao`, passando uma instância de `JogoDao()` com o nosso banco de dados `db` (que é o MySQL).

```
from flask import Flask, render_template, request, redirect, session, flash, url_for
from flask_mysqldb import MySQL

from dao import JogoDao

app = Flask(__name__)
app.secret_key = 'alura'

app.config['MYSQL_HOST'] = "0.0.0.0"
app.config['MYSQL_USER'] = "root"
app.config['MYSQL_PASSWORD'] = "admin"
app.config['MYSQL_DB'] = "jogoteca"
app.config['MYSQL_PORT'] = 3306

db = MySQL(app)

jogo_dao = JogoDao(db)
```

Agora, na nossa função `criar()`, ao invés de utilizarmos `.append()`, vamos chamar a função `salvar()` para o objeto `jogo_dao`, passando um `jogo`:

```
@app.route('/criar', methods=['POST',])
def criar():
    nome = request.form['nome']
    categoria = request.form['categoria']
    console = request.form['console']
    jogo = Jogo(nome, categoria, console)
    jogo_dao.salvar(jogo)
    return redirect(url_for('index'))
```

Se rodarmos a aplicação para testar, receberemos um erro:

```
Traceback (most recent call last):
  File "C:/Users/rodrigo/PycharmProjects/jogoteca/jogoteca.py", line 3, in <module>
    from dao import JogoDao
  File "C:\Users\rodrigo\PycharmProjects\jogoteca\dao.py", line 1, in <module>
    from jogoteca import Jogo, Usuario
  File "C:\Users\rodrigo\PycharmProjects\jogoteca\jogoteca.py", line 3, in <module>
    from dao import JogoDao
ImportError: cannot import name 'JogoDao'
```

Não estamos conseguindo importar o nome `JogoDao`. Isso acontece porque essa é uma classe dentro de um módulo, e esse módulo está importando... da própria `jogoteca`! Ou seja, as nossas classes estão se importando de maneira cíclica, e o próprio Python não sabe o que fazer com essa informação, o que gera um problema.

Nós precisamos ter as classes `Jogo` e `Usuario` no `dao.py`, mas sem fazermos essa importação cíclica da `jogoteca`. Para isso, começaremos a dividir melhor a nossa aplicação.

Ou seja, removeremos essas duas classes da `jogoteca` e criaremos um módulo específico para elas, chamado `models`, clicando com o botão direito no projeto **jogoteca** e em seguida em "New > Python File".

```
class Jogo:
    def __init__(self, nome, categoria, console):
        self.nome = nome
        self.categoria = categoria
        self.console = console


class Usuario:
    def __init__(self, id, nome, senha):
        self.id = id
        self.nome = nome
        self.senha = senha
```

Para consertarmos o código de `jogoteca.py` e de `dao.py`, importaremos essas classes de `models`:

```
from models import Jogo, Usuario
```

Com tudo isso implementado, vamos tentar a criação de um novo jogo em http://127.0.0.1:5000/novo (http://127.0.0.1:5000/novo). Nós seremos redirecionados para http://127.0.0.1:5000/criar (http://127.0.0.1:5000/criar) e receberemos o seguinte erro:

> **builtins.AttributeError**
>
> AttributeError: 'Jogo' object has no attribute 'id'

Faltou

Quando a classe `JogoDao` cria um jogo, ela está passando somente os valores padrão (`nome` , `categoria` e `console`). Na função `salvar()` , ele precisa pegar o `id` do jogo que foi criado e passar para dentro desse objeto:

```
    def salvar(self, jogo):
        cursor = self.__db.connection.cursor()

        if (jogo.id):
            cursor.execute(SQL_ATUALIZA_JOGO, (jogo.nome, jogo.categoria, jogo.console, jogo.id)
        else:
            cursor.execute(SQL_CRIA_JOGO, (jogo.nome, jogo.categoria, jogo.console))
            jogo.id = cursor.lastrowid
```

Porém, estamos fazendo `jogo.id` sem existir um `id` .

No módulo `prepara_banco.py` , vemos que, quando criamos a tabela `jogo` , cada jogo tem um `id` que começa com o valor `1` e vai aumentando de acordo com a quantidade de jogos criados. Isso porque o `nome` do jogo pode ser duplicado, mas eles terão identificadores diferentes.

Portanto, precisaremos modificar a classe `Jogo` para incluir essa propriedade, porém sem quebrar o resto do nosso código. Uma maneira de fazermos isso é tornar o `id` opcional, passando o valor `None` . Dessa forma, sempre teremos o `id` , mas não terá problema se não recebermos um.

```
class Jogo:
    def __init__(self, nome, categoria, console, id=None):
        self.id = id
        self.nome = nome
        self.categoria = categoria
        self.console = console
```

Dessa forma, não teremos problemas com códigos que já estão trabalhando com `Jogo` , mas que não têm `id` . Acessando novamente a página http://127.0.0.1:5000/novo (http://127.0.0.1:5000/novo) da nossa aplicação, conseguiremos criar um jogo sem nenhum erro... porém, o resultado não será mostrado na tela. Ainda falta recuperarmos os dados do banco de dados, e é isso que faremos no próximo vídeo. Até lá!