03

Persistindo informações no MySQL

Transcrição

No curso anterior, vimos que, toda vez que reiniciávamos o servidor, as informações inseridas depois que a aplicação tinha sido iniciada eram perdidas - e não é isso que queremos, certo?

Para resolvermos isso, é necessário armazenar esses dados em um **banco de dados**, onde vamos persistí-los. Mas que banco iremos usar? Nesse curso, usaremos um MySQL - um dos bancos mais usados hoje em dia no mercado, e que irá simplificar um pouco nossa maneira de trabalhar.

Já que não iremos nos aprofundar muito em banco de dados usando o Python, vamos simplesmente integrar esse banco de dados à nossa aplicação. Para que não seja necessário digitar muito código, algumas coisas serão disponibilizadas para você.

Inicialmente, na nossa aplicação, temos duas classes que podem ser representadas no banco de dados: `Jogo` e `Usuario`. Para guardarmos os objetos dessas classes no banco de dados, precisaremos fazer algumas operações, como `create` (criar), `retrieve` (recuperar), `update` e `delete`.

A união dessas operações forma o acrônimo CRUD, que consiste em um conjunto de abstrações que estamos armazenando no banco de dados.

Para essa integração, usaremos uma das várias extensões do Flask, chamada **Flask-MySQLdb**, na versão 0.2.0. Lembra como se instala um pacote no Python? No terminal do **PyCharm**, usaremos o comando `pip3 install flask_mysqldb==0.2.0`. Dessa forma, baixaremos a versão correta do pacote.

Para usarmos o MySQLdb, precisamos primeiro importá-lo. Em seguida, teremos que criar uma conexão, o que é feito acessando o objeto `MySQLdb` e usando a função `conn()` ("*connect*").

Essa função recebe:

- `user` e `passwd` (usuário e senha, o que é comum em um banco de dados)
- `db`, o nome do banco de dados criado
- `host` e `post` - no caso, `127.0.0.1` e `3306`, respectivamente

> Dica: na verdade, estivermos rodando o `host` e a porta padrão, não é necessário passar esses valores. Se você tiver outra configuração de banco, basta adaptar esses parâmetros para conectá-lo.

A conexão permite que façamos algumas operações. A seguir, por exemplo, estamos fazendo um `CREATE`, que é uma inserção no banco de dados - nesse caso, na tabela de `usuario`.

```
cursor = conn.cursor()
cursor.execute('INSERT INTO usuario (id, nome, senha) '
        'VALUES (%s, %s, %s)'
        , ('luan', 'Luan Marques, 'flask')))
conn.commit()
```

Primeiramente, chamamos um `conn.cursor()` para criarmos um `cursor`. Quem executa as nossas *queries* não é a nossa conexão diretamente, mas um cursor.

Em seguida, passamos para o `execute()` do `cursor` todas as *queries* que precisamos executar - nesse caso, um `INSERT` com os campos e valores que queremos inserir nesses campos.

Perceba que utilizamos a notação de interpolação `%` e passamos os valores `s`, de string - ou seja, estamos interpolando strings dentro dessa função. Essas strings, nesse `execute()`, devem ser passadas dentro de uma **tupla** (entre parênteses `()`).

A primeira string é a `id` (nome de usuário), que atribuímos `luan`; a segunda é o `nome` próprio do usuário, que definimos como `Luan Marques`; e a última é a `senha`, que no caso é `flask`.

No final desse `execute()`, nada acontece ainda, pois para fechar a transação com o banco de dados é necessário usar a função `conn.commit()`. Assim os dados serão colocados no banco de dados.

Mas e se quisermos, por exemplo, inserir mais de um registro no banco? Para isso, temos a função `cursor.executemany()`. Com ele, podemos passar uma lista de tuplas, cada uma representando um registro que será inserido no banco de dados, tudo com uma só *query*:

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

Se quisermos mostrar o que está no banco de dados, também usaremos o `execute()`, passando a *query* que queremos executar (por exemplo, `select nome from usuario`):

```
cursor.execute('select nome from usuario')
for usuario in cursor.fetchall():
    print(usuario[0])
```

Para cada dado sendo recebido do banco (`for usuario in cursor`), estamos executando a função `fetchall()`. Sempre que quisermos fazer uma query ou select, primeiro usamos a função `execute()`, preenchemos os dados no `cursor` e depois fazemos `fetchall()` para recuperar esses dados e mostrá-los dentro do `cursor`.

O `cursor` tem, dentro dele, uma lista de tuplas, e cada tupla é um registro no banco de dados. Nesse caso, `usuario` seria uma tupla com todos os dados de `usuario` (`id`, `nome` e `senha`), e estamos retornando somente o `nome`.

Para que nossa aplicação se integre com o banco de dados, você vai precisar de dois módulos que podem ser baixados aqui (https://s3.amazonaws.com/caelum-online-public/739-python-flask2/01/download.zip): **prepara_banco.py**, que prepara o banco de dados, e **dao.py**, que serve para acessar os dados, além de ter algumas funções que irão nos ajudar durante o curso.

> DAO vem de "*Data Access Object*", e é um padrão de objeto utilizado para isolar os códigos relacionados à lógica do banco de dados e o código da aplicação.

Esses arquivos devem ser copiados para dentro do projeto **jogoteca**.

Como estamos trabalhando com um banco relacional, é necessário traduzir os dados de banco para objeto e vice-versa. No arquivo `dao.py`, temos duas funções de tradução:

```
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 o script instalado, é necessário executar o arquivo `prepara_banco.py` para verificar se a criação das tabelas (`usuario` e `jogo`) está funcionando. Em caso positivo, você deve receber a seguinte mensagem, que imprime o nome dos usuários e dos jogos incluídos em cada tabela:

```
C:\Users\rodrigo\PycharmProjects\jogoteca\venv\Scripts\python.exe C:/Users/rodrigo/PycharmProje
Conectando...
 ------------  Usuários:  -------------
Danilo
Luan Marques
Nico
 ------------  Jogos:  -------------
God of War 4
NBA 2k18
Rayman Legends
Super Mario RPG
Super Mario Kart
Fire Emblem Echoes

Process finished with exit code 0
```

Se você tiver algum problema na preparação do banco de dados, consulte o nosso fórum (https://cursos.alura.com.br/forum/curso-flask-upload-persistencia-javascript-jquery/todos/)!

A seguir, vamos continuar integrando nossa aplicação ao banco de dados. Até lá!