Persistência de Dados

Matheus Sehnem de Oliveira
Universidade do Estado de Santa Catarina

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Persistência de Dados

- Nesta seção apresentaremos uma biblioteca para interagir com o banco de dados **PostgreSQL**;
- Para isso, utilizaremos uma biblioteca da comunidade: o psycopg2;
- Serão apresentados apenas conceitos básicos aqui: a realização de consultas (select) e a inserção de novos objetos;
- As demais consultas de update e delete são análogas a esses dois exemplos.

Seções

Instalando pip e psycopg2

- Para instalar a biblioteca, utilizaremos o pacote **pip**. Para instalá-lo basta executar o comando: **py get-pip.py**
- Em seguida, execute o comando: **pip install psycopg2**.

Seções

UDESC

Singleton para a Conexão - Exemplo

- Para manipularmos as interações com o banco de dados, vamos criar uma classe para se conectar ao banco de dados.
- Chamaremos tal classe de Connection, sendo esta um Singleton.

```
import psycopg2 as postgres

class Connection:

    __instance = None
    __connection = None

    def __new__(cls):
        if cls.__instance is None:
            cls.__instance = super().__new__(cls)
            cls.__connection = postgres.connect(
                host='localhost', database='NOME_DB', user='postgres', password='SUA_SENHA')
        return cls.__instance
```

Singleton para a Conexão

- Note que o atributo privado **connection** foi criado dentro do construtor do Singleton, invocando o método **connect()** da biblioteca psycopg2.
- Na invocação desse método deve ser informado o host, a database, o usuário e a senha.
- Caso seu banco de dados se encontre em uma porta diferente da 5432 (*default*), passe o atributo **port** do tipo numérico, logo após o host.

Seções

Métodos da classe Connection

- Criaremos três métodos na classe Connection:
  - Um para fechar a conexão com o banco (é recomendado utilizar quando o programa for encerrado);
  - Outro para obter um cursor para realizar consultas ao banco;
  - E um método para realizar *commits* no banco, isto é, persistir as informações na memória.

Método cursor()

- Tal método será necessário para executar as consultas no banco.
- Como o atributo **connection** é privado, precisamos retornar uma referência a chamada do método **cursor()** do atributo:

```
def cursor(self):
    return self.__connection__.cursor()
```

Método commit()

- O método **commit()** segue o mesmo princípio.
- Ele é executado para cada consulta que necessitar escrever dados no banco de dados (inserts, updates, deletes, etc.).
- Como ele apenas solicita ao banco de dados que os dados devem ser persistidos, não é necessário retornar nada:

```
def commit(self):
    self.__connection__.commit()
```

Método close()

- Já o método **close()** irá encerrar a comunicação com o banco:

```
def close(self):
    return self.__connection__.close()
```

Seções

Classe Pessoa e tabela pessoas - Exemplo

- Vamos criar uma classe Pessoa contendo dois atributos: um id e um nome.
- Além de criarmos a tabela no banco, criaremos também uma sequência para gerar novos id's:

```
create table pessoas(
    id int primary key,
    nome varchar(256)
);

create sequence id_pessoas;
```

Classe Pessoa e tabela pessoas - Exemplo

- Para a classe Pessoa, vamos definir no método **___init___()** esses dois atributos;
- Criaremos um método **nome()** que receberá o nome atribuído a pessoa e retornará a referência do objeto;
- E também o get desse atributo:

```
class Pessoa:

def __init__(self):
    self.__id = -1
    self.__nome = ''

def nome(self, nome):
    self.__nome = nome
    return self

def getNome(self):
    return self.__nome

def id(self, id):
    self.__id = id
    return self
```

Classe Pessoa e tabela pessoas - Exemplo

- Para o id, a principio definiremos apenas o método **id()** responsável por setar o valor do atributo e retornar a referência do objeto.

Seções

Método fromTupla() e __repr__()

- As consultas utilizando a biblioteca psycopg2 retornam tuplas;
- Uma tupla é um conjunto ordenado de n elementos que podem ser acessado indexadamente pela posição;
- Após ser instanciada, uma tupla não pode ser alterada.
- Portanto, ao executarmos o comando **select * from pessoas**, teremos n linhas contendo id e nome, respectivamente;
- Criaremos um método na classe Pessoa que recebe uma tupla e seta os valores do id e nome de acordo com a informação contida na tupla, retornando-o ao final:

```
def fromTupla(self, tupla):
    self.__id = tupla[0]
    self.__nome = tupla[1]
    return self
```

Método fromTupla() e __repr__()

- Por fim, criaremos o método __repr__() para podermos exibir os resultados buscados do banco:

```
def __repr__(self):
    return u'{}: {}'.format(self.__id, self.__nome)
```

Seções

Classe DAO

- Para manter o padrão, vamos criar um DAO para persistir e consultar as informações do banco.
- Importaremos as classes Connection e Pessoa criadas anteriormente e definiremos as SQL's dentro do método **___init___()**.
- Utilizaremos como exemplo três consultas:
  - **select * from pessoas:** buscará todas as informações da tabela pessoas, retornando uma lista de tuplas;
  - **select nextval('id_pessoas'):** retornará o valor seguinte a sequência **id_pessoas**, utilizaremos para inserir novas entradas na tabela;
  - **insert into pessoas values (X,Y):** inserirá uma nova tupla na tabela, onde X receberá o valor do id e Y do nome da nova pessoa inserida.

Classe DAO

```
from connection import Connection
from pessoa import Pessoa

class PessosDAO:

    def __init__(self):
        self.__sqlSelectAll = 'select * from pessoas'
        self.__sqlSelectNewId = "select nextval('id_pessoas')"
        self.__sqlInsert = "insert into pessoas values ({},'{}')"
```

- Note que para a consulta de inserção, deixamos a string preparada para ser substituida pelos valores desejados, no caso o id e o nome, respectivamente.

Seções

Método selectAll()

- Este método irá pegar a instância do Singleton Connection e em seguida buscar o cursor, utilizando-o para executar a SQL de seleção descrita anteriormente.
- Ao final, a variável **result** irá armazenar o resultado da chamada do método **fetchall()**, que retorna todas as tuplas da tabela **pessoas**.
- Após isso, é necessário iterar sob a variável **result**, utilizando o método **fromTupla()** da classe Pessoa para instanciar um novo objeto Pessoa para armazenar os dados da tupla **i**.
- Por fim, é retornada a lista de pessoas buscadas na tabela **pessoas**.

```
def selectAll(self) -> list:
    con = Connection()
    cursor = con.cursor()
    cursor.execute(self.__sqlSelectAll)
    result = cursor.fetchall()
    pessoas = []
    for i in result:
        pessoas.append(Pessoa().fromTupla(i))
    return pessoas
```

Método selectAll()

- A API disponibilizada pela biblioteca traz três métodos que podem ser utilizados para buscar as informações referentes a consulta (todos na forma de tuplas):
  - **fetchone():** requisita para o cursor apenas a primeira linha de todos os resultados da consulta SQL.
  - **fetchmany(n):** traz a quantidade *n* de linhas do resultado da consulta SQL.
  - **fetchall():** retorna todas as linhas restantes da consulta ou, caso nenhum dos métodos anteriores tenham sido utilizados, traz o resultado completo da consulta.

Seções

Método selectNewId()

- Para gerar um novo id, podemos utilizar o método **fetchone()** descrito anteriormente de forma similar ao método **selectAll()**:

```
def __selectNewId(self) -> int:
    con = Connection()
    cursor = con.cursor()
    cursor.execute(self.__sqlSelectNewId)
    result = cursor.fetchone()
    return resul[0]
```

- Note que a variável **result** traz uma linha pois utilizamos o método **fetchone()**;
- Como sabemos que a consulta que enviamos ao **execute** retorna apenas um valor (o próximo da sequência), então podemos retornar o 0-ésimo elemento de **result**.

Seções

Método insertPessoa()

- Por fim, implementaremos o método que recebe um objeto do tipo Pessoa e o persiste no banco.
- Para isso, é necessário primeiro requisitar um novo id ao banco.
- Após, o mesmo processo feito nos métodos anteriores é repetido: pega-se a instância da classe Connection e uma referência para o cursor.
- Feito isso, basta executar o comando SQL, substituindo os valores entre chaves da SQL original, pelos desejados, utilizando o método **format()** da string.

```
def insertPessoa(self, pessoa):
    id = self.__selectNewId()
    con = Connection()
    cursor = con.cursor()
    cursor.execute(self.__sqlInsert.format(id, pessoa.getNome()))
    con.commit()
```

- É importante lembrar de fazer a chamada do método **commit()** ao final. Caso contrário o objeto não será persistido no banco!

Referencias

KUWAKI, F. B. T. F. **Programação Orientada a Objetos em Python: O básico**. 2020.

KUWAKI, V. T. F. Modelo de slides udesc lattex. In: . [S.l.]: Disponível em: <https://github.com/takeofriedrich/slidesUdescLattex>. Acesso em: 26 nov. 2021.

UDESC

Duvidas:
Matheus Sehnem de Oliveira
matheus.sehnem@outlook.com
github.com/matheusrnk