INTERAÇÃO COM BANCO DE DADOS

Conceitos

O python possui muitas bibliotecas para interagir com diversos bancos de dados. Aqui, o foco será a integração com o PostgreSQL. Isso porque pode ser usado gratuitamente, possui bastante documentação disponível on-line e pode ser aplicado para resolver problemas reais.

Para explicar como o python interage com o PostgreSQL, trabalharemos na seguinte aplicação:

“Desenvolver um programa em que seja possível realizar as operações CRUD para gerenciar uma agenda telefônica que vai tratar de nomes e números de telefones”.

A ideia é desenvolvermos uma aplicação CRUD para que os usuários possam interagir com o sistema para inserir, modificar, consultar e excluir dados de uma agenda telefônica, e que essas operações fiquem sob a responsabilidade do sistema gerenciador de banco de dados.

Dica

Inicialmente, é necessário ter o PostgreSQL instalado. Os exemplos apresentados aqui foram implementados na versão 4.24 do PostgreSQL.

```
CREATE TABLE public."AGENDA"
(
    id integer NOT NULL,
    nome text COLLATE pg_catalog."default" NOT NULL,
    telefone char(12) COLLATE pg_catalog."default" NOT NULL
)
TABLESPACE pg_default;
ALTER TABLE public."AGENDA"
    OWNER to postgres;
```

Para testar se a criação da tabela está correta, pode-se inserir um registro da seguinte maneira:

```
INSERT INTO public."AGENDA"( id, nome, telefone)
VALUES (1, 'teste 1', '02199999999');
```

Em seguida, fazemos uma consulta na tabela:

SELECT * FROM public." AGENDA";

Se tudo funcionar corretamente, aparecerá a seguinte saída:

	id integer	nome text	telefone character (12)
1	1	teste 1	02199999999

Saída da consulta na tabela AGENDA

Agora, vamos estudar o pacote que será utilizado para que o python possa interagir com o PostgreSQL: psycopg2.

Atenção!

O PostgreSQL pode ser integrado ao python usando o módulo psycopg2. Trata-se de um adaptador de banco de dados PostgreSQL. O pacote psycopg2 é simples, rápido e estável.

Para instalá-lo, basta digitar na linha de comando do python:

pip install psycopg2

Para interagir com o banco de dados PostgreSQL com o uso da biblioteca psycopg2, primeiro é necessário criar um objeto Connection, que representa o banco de dados, e, em seguida, pode-se criar um objeto cursor que será bastante útil para executar todas as instruções SQL. Isso será detalhado um pouco mais à frente.

Antes disso, vamos apresentar as principais APIs (rotinas de interface de programação) da psycopg2:

PostgreSQL

psycopg2.connect (database = "NomeDoBancoDeDados", user = "LoginDoUsuário", senha = "SenhaDoBancoDeDados", host = "EndereçoDaAplicação", porta = "NúmeroDaPorta")

A conexão com o banco de dados PostgreSQL é feita com essa API. O banco de dados retorna um objeto de conexão, se o banco de dados for aberto com sucesso. Para aplicações que vão rodar localmente, utiliza-se o endereço de localhost dado por 127.0.0.1". A

porta de comunicação padrão do PostgreSQL é a "5432", mas esse valor pode ser mudado.

Criar um cursor

connection.cursor ()

Esta API cria um cursor que será usado ao longo da programação para interagir com o banco de dados com python.

Executar uma instrução SQL

cursor.execute (sql [, parâmetros opcionais])

Esta rotina é aplicada para executar uma instrução SQL. A instrução SQL pode ser parametrizada.

Por exemplo, seja o trecho de código:

```
nomeDaTabela = 'tabelaExemplo'
cursor.execute(" insert into %s values (%%s, %%s)" % nomeDaTabela,[10, 20])
```

O comando executará a instrução “insert” para inserir os valores 10 e 20 na tabela ‘tabelaExemplo’.

Executa um comando SQL

cursor.executemany (sql, sequência_de_parâmetros) Esta rotina executa um comando SQL em todas as sequências de parâmetros.

Por exemplo, seja o trecho de código:

```
carros = (
(1, 'Celta', 35000),
(2, 'Fusca', 30000),
(3, 'Fiat Uno', 32000)
)
con = psycopg2.connect(database='BancoExemplo', user='postgres',
      password='postgres')
cursor = con.cursor()
```

query = "INSERT INTO cars (id, nome, preco) VALUES (%s, %s, %s)"

cursor.executemany(query, carros)

O trecho de código começa com uma lista de três carros, na qual cada carro possui um código de identificação, um nome e um preço.

Em seguida, é feita uma conexão com o banco de dados “BancoExemplo”.

Logo depois, é criado o cursor que será usado para realizar as operações sobre o banco de dados.

Por fim, é executada a rotina “executemany”, sendo que ela recebe uma query e uma lista de carros que serão inseridos no banco de dados.

- **cursor.callproc ('NomeDaFunção_Ou_NomeDoProcedimentoArmazenado', [parâmetros IN e OUT,])**
  Esta rotina faz chamada para uma função ou um procedimento armazenado do banco de dados. Os parâmetros IN e OUT correspondem, respectivamente, aos parâmetros de entrada e saída da função ou do procedimento armazenado e devem ser separados por vírgulas.

- **cursor.rowcount**
  Este atributo retorna o número total de linhas do banco de dados que foram modificadas, inseridas ou excluídas pela última instrução de “execute“.

- **connection.commit()**
  Este método confirma a transação atual. É necessário que ele seja chamado ao final de uma sequência de operações sql, pois, caso contrário, tudo o que foi feito desde a última chamada até o “commit” não será visível em outras conexões de banco de dados.

- **connection.rollback()**
  Este método reverte quaisquer mudanças no banco de dados desde a última chamada até o “commit”.

- **connection.close()**
  Este método fecha a conexão com o banco de dados. Ele não chama o “commit” automaticamente. Se a conexão com o banco de dados for fechada sem chamar o “commit” primeiro, as alterações serão perdidas.

- **cursor.fetchone()**
  Este método busca a próxima linha de um conjunto de resultados de consulta, retornando uma única sequência, ou nenhuma, quando não há mais dados disponíveis.

- **cursor.fetchmany([size = cursor.arraysize])**
  Esta rotina busca o próximo conjunto de linhas de um resultado de consulta, retornando uma lista. Uma lista vazia é retornada quando não há mais linhas disponíveis. O método tenta buscar quantas linhas forem indicadas pelo parâmetro “size”.

- **cursor.fetchall()** Esta rotina busca todas as linhas (restantes) de um resultado de consulta, retornando uma lista. Uma lista vazia é retornada quando nenhuma linha está disponível.

Exemplo prático com psycopg2

Criação de uma tabela

Inserção de dados

Seleção de dados

Atualização de dados

Exclusão de dados

Agora, vamos aos exemplos que implementam as operações CRUD.

Criação de tabela

Este primeiro código mostra como criar uma tabela a partir do python. É uma alternativa em relação a criar a tabela usando o PostgreSQL.

```
import psycopg2

conn = psycopg2.connect(database = "postgres", user = "postgres", password = "senha123", host = "127.0.0.1", port = "5432")

print("Conexão com o Banco de Dados aberta com sucesso!")

cur = conn.cursor()

cur.execute('''CREATE TABLE Agenda(ID INT PRIMARY KEY NOT NULL,Nome TEXT NOT NULL,Telefone CHAR(12));''')

print("Tabela criada com sucesso!")

conn.commit()

conn.close()
```

Depois de colocar o programa para executar, se tudo funcionar corretamente, aparecerão as mensagens na tela:

Conexão com o Banco de Dados feita com Sucesso! Tabela criada com sucesso!

Agora, vamos analisar o código.

- Linha 1 - É feita a importação da biblioteca psycopg2.

- Linha 2 - É feita a conexão com o banco de dados. Observe os parâmetros da função “connect”, pois é necessário que você crie um banco no PostgreSQL com usuário e senha, conforme escrito na função.
- Linha 3 - É exibida uma mensagem de sucesso para o usuário.
- Linha 4 - É criado o cursor que vai permitir realizar operações no banco de dados.
- Linha 5 - Executa o comando SQL para criar a tabela “Agenda” com os campos “ID”, “Nome” e “Telefone”.
- Linha 9 - É exibida uma mensagem de criação da tabela com sucesso.
- Linha 10 - É executada a função “commit” para confirmar a execução das operações SQL.
- Linha 11 - Por fim, é fechada a conexão com o banco de dados.

Inserção de dados na tabela

O exemplo desse código mostra como inserir um registro em uma tabela a partir do python usando a biblioteca psycopg2.

O exemplo desse código mostra como inserir um registro em uma tabela a partir do python usando a biblioteca psycopg2.

```
import psycopg2

conn = psycopg2.connect(database = " postgres", user="postgres" , password=" senha123" , host="127.0.0.1", port="5432" )

print ("Conexão com o Banco de Dados aberta com sucesso!")

cur=conn.cursor()

cur.execute("""INSERT INTO public."AGENDA" ("id", "nome" , "telefone" ) VALUES (1, 'Pessoa 1' , '02199999999' )""")

conn.commit()

print("Inserção realizada com sucesso!"); 8conn.close()
```

As mensagens que vão aparecer depois da execução do programa são:

Conexão com o Banco de Dados feita com Sucesso! Inserção realizada com sucesso!

Vamos analisar o código.

- Linha 1 a 4 - São realizadas as mesmas operações do exemplo anterior: Importação da biblioteca “psycopg2”, abrir a conexão com o banco de dados “postgres” e impressão da mensagem “Conexão aberta com sucesso!”.

- Linha 5 - É executado o comando SQL para inserir dados na tabela AGENDA.No caso, o registro é formado pelos seguintes dados: O campo “id” recebe o valor 1, o campo “nome” recebe o valor ‘Pessoa 1’ e, por fim, o campo “telefone” recebe o valor ‘02199999999’.
  Essa linha tem mais algumas questões que merecem destaque: O uso de aspas simples e duplas.
  No caso do banco de dados PostgreSQL, o nome da tabela e dos campos deve estar entre aspas duplas, por causa disso é que o comando insert possui três aspas duplas logo no início e no final. Sendo assim, muita atenção com isso, pois existem algumas variações conforme o sistema gerenciador de banco de dados escolhido.
- Linha 6 a 8 - Do mesmo modo como foi realizado no exemplo de criação da tabela “Agenda”, são realizadas as seguintes operações: “commit” das operações do banco de dados, fechamento da conexão com o banco de dados, impressão na linha de comando da mensagem “Inserção realizada com sucesso!”.

Seleção de dados na tabela

Antes de descrever o exemplo de seleção de dados, já podemos perceber algo em comum em todas as operações dos códigos para trabalhar com banco de dados no início do código:

Importação da biblioteca “psycopg2”.

Abertura da conexão com o banco de dados “postgres”.

E no final do código:

“Commit” das operações realizadas no banco de dados para confirmar a execução delas. Esse comando é obrigatório.

Fechamento da conexão com o banco de dados.

Agora, vamos analisar o código que mostra como selecionar um registro em uma tabela a partir da biblioteca psycopg2 do python.

```
import psycopg2

conn = psycopg2.connect(database = " postgres", user="postgres" , password=" senha123" , host="127.0.0.1", port="5432" )

print ("Conexão com o Banco de Dados aberta com sucesso!")

cur=conn.cursor()

cur.execute("""select * from public."AGENDA" where "id"=1""")

registro=cur.fetchone()

print(registro) 8 conn.commit() 9 print("Seleção realizada com sucesso!");
```

conn.close()

Depois de colocar o programa para executar, se tudo funcionar corretamente, aparecerão as mensagens na tela:

Conexão com o Banco de Dados feita com Sucesso!
(1, 'Pessoa 1', '02199999999 ')
Tabela criada com sucesso!

Vamos analisar os trechos mais importantes do código.

- Linha 5 - É feita a consulta na tabela “Agenda” pelo registro com “id” igual a 1, por meio do comando Select do SQL.
- Linha 6 - É executado o método “fetchone” que recupera exatamente um registro do “cursor” e atribui para a variável “registro”.
- Linha 7 - É impresso na linha de comando o resultado da consulta armazenado na variável “registro”.

Atualização de dados na tabela

Este exemplo mostra como atualizar os registros de uma tabela a partir do python.

```
import psycopg2

conn = psycopg2.connect(database = " postgres", user="postgres" , password="senha123" , host="127.0.0.1" , port="5432" )

print ("Conexão com o Banco de Dados aberta com sucesso!")

cur=conn.cursor()

print("Consulta antes da atualização")

cur.execute("""select * from public."AGENDA" where "id"=1""")

registro=cur.fetchone()

print(registro)

#Atualização de um único registro

cur.execute("""Update public."AGENDA" set "telefone"='02188888888' where "id"=1""")

conn.commit()

print("Registro Atualizado com sucesso! ")

cur = conn.cursor()

print(" Consulta depois da atualização")
```

```
cur.execute("""select * from public."AGENDA" where "id"=1""")

registro=cur.fetchone()

print(registro)

conn.commit()

print("Seleção realizada com sucesso!");

conn.close()
```

Este código possui três partes distintas, que são:

Parte 1

Uma consulta antes da atualização que mostra os dados do registro antes de serem modificados.

Parte 2

A atualização do registro que vai modificar os dados.

Parte 3

Uma consulta depois da atualização do registro que mostra como ficaram os dados do registro depois de serem atualizados.

Atenção!

A linha 10 é a mais importante deste código. É nela que é executado o comando “update” do sql, que atualizará o dado do campo “telefone” do registro, cujo campo “id” contenha o valor “1”.

Perceba, ainda, que é associado o comando “commit” para o comando “update” do sql na linha 11.

Exclusão de dados na tabela

Por fim, vamos ver o exemplo para excluir um registro de uma tabela.

```
import psycopg2

conn = psycopg2.connect(database = " postgres", user="postgres" , password="senha123" , host="127.0.0.1" , port="5432" )

print ("Conexão com o Banco de Dados aberta com sucesso!")

cur=conn.cursor()

cur.execute("""Delete from public."AGENDA" where "id"=1""")
```

```
conn.commit()

cont=cur.rowcount

print(cont, "Registro excluído com sucesso!")

print("Exclusão realizada com sucesso!");

conn.close()
```

Depois de colocar o programa para executar, se tudo funcionar corretamente, aparecerão as mensagens na tela:

Conexão com o Banco de Dados aberta com Sucesso!
1 Registro excluído com sucesso!Exclusão realizada
com sucesso!

Vamos analisar as partes mais importantes do código.

- Linha 5 - É feita a consulta na tabela “Agenda” pelo registro com “id” igual a 1, por meio do comando Select do SQL.É executado o comando “delete” do sql que excluirá o registro cujo campo “id” seja igual a “1”.
- Linha 7 - A propriedade “rowcount” do “cursor” retorna a quantidade de registros que foram excluídos da tabela “Agenda”.
- Linha 8 - É impresso na linha de comando o total de registros que foram excluídos.

Saiba mais

Além da biblioteca psycopg2, existem outras bibliotecas para trabalhar com vários sistemas gerenciadores de banco de dados. Por exemplo:

- pyMySQL: Biblioteca que faz interface com o MySQL.
- cx_Oracle: Biblioteca que faz interface com o Oracle.
- PySqlite: Biblioteca que faz interface com o SQLite.
- PyMongo: Biblioteca que faz interface com o mongodb, que é um banco de dados NoSQL.