Digital College

FORMAÇÃO EM

DATA ANALYTICS

MÓDULO 3:

PYTHON PARA ANÁLISE DE DADOS

UNIDADE 2:

MANIPULAÇÃO DE DADOS COM PYTHON

Sumário

1. Introdução

Manipular dados com Python é uma tarefa essencial para quem trabalha com análise de dados. Existem diversas bibliotecas em Python que facilitam esse processo, como Pandas, Numpy e Matplotlib.

Para começar, você precisa ter seus dados armazenados em algum formato que possa ser lido. Uma vez que você tenha seus dados armazenados, você pode usar a biblioteca Pandas para ler os dados em um DataFrame, que é uma estrutura de dados tabular.

Uma vez que você tenha os dados armazenados em um DataFrame, você pode executar várias operações nele, como filtrar dados, calcular estatísticas, agrupar dados e visualizar gráficos.

Esses são apenas alguns exemplos de como você pode manipular dados com Python usando a biblioteca Pandas. Há muitas outras funcionalidades disponíveis, então vale a pena explorar a documentação oficial do Pandas para ver o que mais você pode fazer.

2. Manipulando dados no PostgreSQL

Existem algumas bibliotecas em Python que você pode usar para acessar bancos de dados PostgreSQL. Algumas das mais populares são:

- Psycopg2: é uma biblioteca PostgreSQL robusta adaptada para Python e que oferece uma interface de baixo nível amplamente utilizada para acessar bancos de dados PostgreSQL diretamente do Python;

- SQLAlchemy: é uma biblioteca Python que permite trabalhar com vários bancos de dados, incluindo PostgreSQL. Ele fornece uma camada de abstração sobre o banco de dados, permitindo que você trabalhe de uma maneira mais orientada a objetos;

- PyGreSQL: é uma biblioteca Python projetada especificamente para trabalhar com bancos de dados PostgreSQL. Ela fornece uma interface simples para executar consultas SQL, bem como vários recursos adicionais, como suporte a transações, suporte a tipos de dados PostgreSQL e muito mais;

- Psql: é um pacote Python que oferece uma maneira fácil de executar comandos psql diretamente do Python. É uma ferramenta útil para executar consultas e comandos de banco de dados diretamente do Python, sem precisar se preocupar em escrever um código SQL completo.

Essas são algumas das bibliotecas Python mais comuns que você pode usar para acessar bancos de dados PostgreSQL. Qualquer uma delas pode ser uma boa opção, dependendo do seu caso de uso específico e das suas necessidades.

3. Utilizando a biblioteca Psycopg2

A biblioteca psycopg2 é uma biblioteca PostgreSQL adaptada para Python, que permite que você se conecte a bancos de dados PostgreSQL diretamente do Python. Ela oferece uma interface de baixo nível para acessar o banco de dados, o que significa que você precisa escrever código SQL para executar consultas e comandos no banco de dados.

Aqui está um exemplo básico de como usar o psycopg2 para se conectar a um banco de dados PostgreSQL e executar uma consulta SQL:

```
import psycopg2

# Conectando ao banco de dados
conn = psycopg2.connect(
    host="localhost",
    database="mydatabase",
    user="myuser",
    password="mypassword"
)

# Criando um cursor
cur = conn.cursor()

# Executando uma consulta SQL
cur.execute("SELECT * FROM mytable")

# Obtendo os resultados
rows = cur.fetchall()

# Exibindo os resultados
for row in rows:
    print(row)

# Fechando a conexão
cur.close()
conn.close()
```

Figura 01

Neste exemplo, estamos nos conectando a um banco de dados PostgreSQL chamado "mydatabase", usando o usuário "myuser" e a senha "mypassword". Em seguida, estamos executando uma consulta SQL simples que seleciona todos os registros da tabela "mytable" e, finalmente, exibindo os resultados.

O psycopg2 é uma biblioteca robusta e amplamente utilizada para conectar-se a bancos de dados PostgreSQL. Ele oferece muitos recursos avançados, como suporte a transações, suporte a tipos de dados PostgreSQL e muito mais. No entanto, como é uma biblioteca de baixo nível, pode exigir um pouco mais de trabalho para usar do que algumas das bibliotecas de nível superior, como o SQLAlchemy.

É importante notar que, ao usar o psycopg2, você precisa ter cuidado para evitar ataques de injeção de SQL. Certifique-se sempre de validar e sanitizar todos os dados de entrada antes de usá-los em consultas SQL.

3.1. Inserindo dados

A inserção de dados é uma das operações básicas em um sistema de gerenciamento de banco de dados. Para realizar essa operação em um banco de dados PostgreSQL usando a biblioteca psycopg2 em Python, precisamos estabelecer uma conexão com o banco de dados, criar um cursor e executar uma consulta SQL INSERT.

Para inserir um registro em uma tabela em um banco de dados PostgreSQL, podemos usar a seguinte consulta SQL:

```
INSERT INTO nome_da_tabela (coluna1, coluna2, ...) VALUES (valor1, valor2,
```

Figura 02

Para executar essa consulta SQL usando a biblioteca psycopg2, precisamos estabelecer uma conexão com o banco de dados e criar um cursor. Em seguida, podemos executar a consulta usando o método execute() do cursor e, finalmente, confirmar a transação usando o método commit() da conexão. O código Python para realizar a inserção de dados em uma tabela em um banco de dados PostgreSQL usando a biblioteca psycopg2 é mostrado abaixo:

```
import psycopg2

# Estabelecendo a conexão com o banco de dados
conn = psycopg2.connect(
    host="localhost",
    database="nome_do_banco_de_dados",
    user="nome_de_usuario",
    password="senha_do_usuario"
)

# Criando um cursor
cur = conn.cursor()

# Executando a consulta SQL INSERT
cur.execute("INSERT INTO nome_da_tabela (coluna1, coluna2, ...) VALUES (va

# Confirmar a transação
conn.commit()

# Fechando o cursor e a conexão com o banco de dados
cur.close()
conn.close()
```

Figura 03

Agora vamos ver um exemplo prático de como inserir um registro em uma tabela em um banco de dados PostgreSQL usando a biblioteca psycopg2 em Python:

```python
import psycopg2

# Estabelecendo a conexão com o banco de dados
conn = psycopg2.connect(
    host="localhost",
    database="mydatabase",
    user="myuser",
    password="mypassword"
)

# Criando um cursor
cur = conn.cursor()

# Executando a consulta SQL INSERT
cur.execute("INSERT INTO usuarios (nome, idade) VALUES ('João', 30);")

# Confirmar a transação
conn.commit()

# Fechando o cursor e a conexão com o banco de dados
cur.close()
conn.close()
```

Figura 04

Neste exemplo, estamos inserindo um novo registro na tabela "usuarios" com nome "João" e idade "30". Note que após executar a consulta SQL INSERT, estamos confirmando a transação usando o método commit() da conexão. Isso garante que as alterações sejam permanentemente armazenadas no banco de dados.

3.2. Alterando dados

A alteração de dados é uma operação importante em um sistema de gerenciamento de banco de dados, pois permite atualizar registros existentes em uma tabela. Para realizar essa operação em um banco de dados PostgreSQL usando a biblioteca psycopg2 em Python, precisamos estabelecer uma conexão com o banco de dados, criar um cursor e executar uma consulta SQL UPDATE.

Para alterar um registro em uma tabela em um banco de dados PostgreSQL, podemos usar a seguinte consulta SQL:

```
UPDATE nome_da_tabela SET coluna1 = valor1, coluna2 = valor2, ... WHERE cor
```

Figura 05

Neste exemplo, "nome_da_tabela" é o nome da tabela onde queremos alterar os dados; "coluna1", "coluna2", ... são as colunas da tabela que queremos alterar; e "valor1", "valor2", ... são os novos valores que queremos atribuir a essas colunas. A cláusula WHERE é usada para especificar quais registros devem ser alterados.

Para executar essa instrução SQL usando a biblioteca psycopg2, precisamos estabelecer uma conexão com o banco de dados e criar um cursor. Em seguida, podemos executar a instrução usando o método execute() do cursor e, finalmente, confirmar a transação usando o método commit() da conexão. O código Python para realizar a alteração de dados em uma tabela em um banco de dados PostgreSQL usando a biblioteca psycopg2 é mostrado abaixo:

```
import psycopg2

# Estabelecendo a conexão com o banco de dados
conn = psycopg2.connect(
    host="localhost",
    database="nome_do_banco_de_dados",
    user="nome_de_usuario",
    password="senha_do_usuario"
)

# Criando um cursor
cur = conn.cursor()

# Executando a consulta SQL UPDATE
cur.execute("UPDATE nome_da_tabela SET coluna1 = valor1, coluna2 = valor2,

# Confirmar a transação
conn.commit()

# Fechando o cursor e a conexão com o banco de dados
cur.close()
conn.close()
```

Figura 06

Agora vamos ver um exemplo prático de como alterar um registro em uma tabela em um banco de dados PostgreSQL usando a biblioteca psycopg2 em Python:

```
import psycopg2

# Estabelecendo a conexão com o banco de dados
conn = psycopg2.connect(
    host="localhost",
    database="mydatabase",
    user="myuser",
    password="mypassword"
)

# Criando um cursor
cur = conn.cursor()

# Executando a consulta SQL UPDATE
cur.execute("UPDATE usuarios SET idade = 31 WHERE nome = 'João';")

# Confirmar a transação
conn.commit()

# Fechando o cursor e a conexão com o banco de dados
cur.close()
conn.close()
```

Figura 07

Neste exemplo, estamos alterando o registro na tabela "usuarios" com nome "João" para atribuir um novo valor à coluna "idade". Note que a cláusula WHERE é usada para especificar qual registro deve ser alterado. Depois de executar a consulta SQL UPDATE, estamos confirmando a transação usando o método commit() da conexão. Isso garante que as alterações sejam permanentemente armazenadas no banco de dados.

3.3. Excluindo dados

A exclusão de dados é uma operação importante em um sistema de gerenciamento de banco de dados, pois permite remover registros existentes de uma tabela. Para realizar essa operação em um banco de dados PostgreSQL usando a biblioteca psycopg2 em Python, precisamos estabelecer uma conexão com o banco de dados, criar um cursor e executar uma consulta SQL DELETE.

Para excluir um registro de uma tabela em um banco de dados PostgreSQL, podemos usar a seguinte consulta SQL:

```
DELETE FROM nome_da_tabela WHERE condicao;
```

Figura 08

Neste exemplo, "nome_da_tabela" é o nome da tabela da qual queremos excluir os dados, e "condicao" é usado para especificar quais registros devem ser excluídos.

Para executar essa consulta SQL usando a biblioteca psycopg2, precisamos estabelecer uma conexão com o banco de dados e criar um cursor. Em seguida, podemos executar a consulta usando o método execute() do cursor e, finalmente, confirmar a transação usando o método commit() da conexão. O código Python para realizar a exclusão de dados em uma tabela em um banco de dados PostgreSQL usando a biblioteca psycopg2 é mostrado abaixo:

```
import psycopg2

# Estabelecendo a conexão com o banco de dados
conn = psycopg2.connect(
    host="localhost",
    database="nome_do_banco_de_dados",
    user="nome_de_usuario",
    password="senha_do_usuario"
)

# Criando um cursor
cur = conn.cursor()

# Executando a consulta SQL DELETE
cur.execute("DELETE FROM nome_da_tabela WHERE condicao;")

# Confirmar a transação
conn.commit()

# Fechando o cursor e a conexão com o banco de dados
cur.close()
conn.close()
```

Figura 09

Agora vamos ver um exemplo prático de como excluir um registro de uma tabela em um banco de dados PostgreSQL usando a biblioteca psycopg2 em Python:

```python
import psycopg2

# Estabelecendo a conexão com o banco de dados
conn = psycopg2.connect(
    host="localhost",
    database="mydatabase",
    user="myuser",
    password="mypassword"
)

# Criando um cursor
cur = conn.cursor()

# Executando a consulta SQL DELETE
cur.execute("DELETE FROM usuarios WHERE nome = 'João';")

# Confirmar a transação
conn.commit()

# Fechando o cursor e a conexão com o banco de dados
cur.close()
conn.close()
```

Figura 10

Neste exemplo, estamos excluindo o registro da tabela "usuarios" com nome "João". Note que a cláusula WHERE é usada para especificar qual registro deve ser excluído. Depois de executar a consulta SQL DELETE, estamos confirmando a transação usando o método commit() da conexão. Isso garante que as alterações sejam permanentemente armazenadas no banco de dados.

4. Utilizando a biblioteca SQLAlchemy

A biblioteca SQLAlchemy é uma biblioteca Python de Mapeamento Objeto-Relacional (ORM) que permite interagir com bancos de dados relacionais de forma mais fácil e intuitiva. Com o SQLAlchemy, você pode escrever código Python que pode interagir com bancos de dados relacionais como se fossem objetos Python, em vez de ter que escrever código SQL manualmente.

O SQLAlchemy é uma biblioteca bastante completa, que oferece várias ferramentas e recursos avançados, como suporte a transações, gerenciamento de conexão, geração de esquema etc. Além disso, ele suporta vários bancos de dados relacionais, incluindo PostgreSQL, MySQL, SQLite e Oracle, entre outros.

Aqui está um exemplo básico de como usar o SQLAlchemy para se conectar a um banco de dados PostgreSQL e executar uma consulta SQL:

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Definindo o modelo
Base = declarative_base()

class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    age = Column(Integer)

# Criando uma conexão
engine = create_engine('postgresql://myuser:mypassword@localhost/mydatabase

# Criando as tabelas
Base.metadata.create_all(engine)

# Criando uma sessão
Session = sessionmaker(bind=engine)
session = Session()

# Executando uma consulta
users = session.query(User).all()

# Exibindo os resultados
for user in users:
    print(user.id, user.name, user.age)
```

Figura 11

Neste exemplo, estamos definindo um modelo de usuário, usando classes Python, com a ajuda da classe declarative_base() do SQLAlchemy. Em seguida, estamos criando uma conexão com o banco de dados PostgreSQL usando a função create_engine(), que recebe a URL de conexão com o banco de dados. Depois, estamos criando as tabelas correspondentes ao nosso modelo, usando o método metadata.create_all().

Logo depois, estamos criando uma sessão para interagir com o banco de dados, usando a classe sessionmaker(). E, finalmente, estamos executando uma consulta SQL para buscar todos os usuários do banco de dados, usando o método query() da sessão.

O SQLAlchemy é uma biblioteca bastante poderosa e flexível, mas pode ter um pouco de curva de aprendizado para quem nunca usou um ORM antes. No entanto, ele pode facilitar muito o trabalho com bancos de dados relacionais, permitindo que você escreva código Python mais legível e manutenível.

4.1. Inserindo dados

Para inserir dados em uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy em Python, precisamos criar uma sessão, definir o modelo da tabela, criar uma instância da classe modelo preenchendo os valores dos campos e, finalmente, chamar o método add() na sessão para salvar a instância no banco de dados.

O SQLAlchemy é uma biblioteca de Mapeamento Objeto-Relacional (ORM) em Python que permite interagir com bancos de dados relacionais por meio de objetos Python. Com o SQLAlchemy, podemos definir modelos de tabela usando classes Python e interagir com os registros do banco de dados usando essas classes. Para inserir registros em uma tabela, podemos criar uma instância da classe modelo preenchendo os valores dos campos e, em seguida, usar o método add() em uma sessão do SQLAlchemy para salvar a instância no banco de dados.
Para inserir registros em uma tabela em um banco de dados PostgreSQL usando o SQLAlchemy, precisamos seguir os seguintes passos:

- Criar uma sessão usando o método sessionmaker() do SQLAlchemy e a conexão com o banco de dados;
- Definir o modelo da tabela usando uma classe Python que herda da classe declarative_base() do SQLAlchemy. Essa classe deve conter a definição da tabela e seus campos;
- Criar uma instância da classe modelo preenchendo os valores dos campos;
- Chamar o método add() na sessão para salvar a instância no banco de dados;
- Confirmar a transação usando o método commit() da sessão.

O código Python para inserir um registro em uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy é mostrado abaixo:

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Criando uma conexão com o banco de dados
engine = create_engine('postgresql://myuser:mypassword@localhost/mydatabase')

# Criando uma sessão
Session = sessionmaker(bind=engine)
session = Session()

# Definindo o modelo da tabela
Base = declarative_base()
class Usuario(Base):
    __tablename__ = 'usuarios'
    id = Column(Integer, primary_key=True)
    nome = Column(String)
    email = Column(String)

# Criando uma instância do modelo e preenchendo os valores dos campos
novo_usuario = Usuario(nome='João', email='joao@example.com')

# Salvando a instância no banco de dados
session.add(novo_usuario)

# Confirmar a transação
session.commit()

# Fechando a sessão
session.close()
```

Figura 12

Agora vamos ver um exemplo prático de como inserir um registro em uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy em Python:

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Criando uma conexão com o banco de dados
engine = create_engine('postgresql://myuser:mypassword@localhost/mydatabase

# Criando uma sessão
Session = sessionmaker(bind=engine)
session = Session()

# Definindo o modelo da tabela
Base = declarative_base()
class Usuario(Base):
    __tablename__ = 'usuarios'
    id = Column(Integer, primary_key=True)
    nome = Column(String)
    email = Column(String)

# Criando uma instância do modelo e preenchendo os valores dos campos
novo_usuario = Usuario(nome='Jo
```

Figura 13

4.2. Alterando dados

Para alterar dados em uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy em Python, precisamos criar uma sessão, buscar o registro a ser alterado na tabela, atualizar os valores dos campos desejados e, finalmente, chamar o método commit() na sessão para salvar as alterações no banco de dados.

O SQLAlchemy é uma biblioteca de Mapeamento Objeto-Relacional (ORM) em Python que permite interagir com bancos de dados relacionais por meio de objetos Python. Com o SQLAlchemy, podemos definir modelos de tabela usando classes Python e interagir com os registros do banco de dados usando essas classes. Para atualizar registros em uma tabela, podemos buscar o registro a ser alterado na tabela, atualizar os valores dos campos desejados e, em seguida, usar o método commit() em uma sessão do SQLAlchemy para salvar as alterações no banco de dados.

Para alterar registros em uma tabela em um banco de dados PostgreSQL usando o SQLAlchemy, precisamos seguir os seguintes passos:

- Criar uma sessão usando o método sessionmaker() do SQLAlchemy e a conexão com o banco de dados;
- Definir o modelo da tabela usando uma classe Python que herda da classe declarative_base() do SQLAlchemy. Essa classe deve conter a definição da tabela e seus campos;
- Buscar o registro a ser alterado na tabela usando o método query() da sessão e a chave primária da tabela;
- Atualizar os valores dos campos desejados na instância do modelo;
- Chamar o método commit() na sessão para salvar as alterações no banco de dados.

O código Python para alterar um registro em uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy é mostrado abaixo:

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Criando uma conexão com o banco de dados
engine = create_engine('postgresql://myuser:mypassword@localhost/mydatabase'

# Criando uma sessão
Session = sessionmaker(bind=engine)
session = Session()

# Definindo o modelo da tabela
Base = declarative_base()
class Usuario(Base):
    __tablename__ = 'usuarios'
    id = Column(Integer, primary_key=True)
    nome = Column(String)
    email = Column(String)

# Buscando o registro a ser alterado na tabela
usuario = session.query(Usuario).get(1)

# Atualizando os valores dos campos desejados na instância do modelo
usuario.nome = 'João Silva'
usuario.email = 'joao.silva@example.com'

# Salvando as alterações no banco de dados
session.commit()

# Fechando a sessão
session.close()
```

Figura 14

Agora vamos ver um exemplo prático de como alterar um registro em uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy em Python:

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Criando uma conexão com o banco de dados
engine = create_engine('postgresql://myuser:mypassword@localhost/mydatabase

# Criando uma sessão
Session = sessionmaker(bind=engine)
session = Session()

# Definindo o modelo da tabela
Base = declarative_base()
class Usuario(Base):
    __tablename__ = 'usuarios'
    id = Column(Integer, primary_key=True)
    nome = Column(String)
    email
```

Figura 15

4.3. Excluindo dados

Para excluir dados de uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy em Python, precisamos criar uma sessão, definir o modelo da tabela, usar a sessão para consultar os registros que desejamos excluir e, finalmente, chamar o método delete() na sessão para excluir os registros selecionados.

Para excluir registros de uma tabela em um banco de dados PostgreSQL usando o SQLAlchemy, precisamos seguir os seguintes passos:

- Criar uma sessão usando o método sessionmaker() do SQLAlchemy e a conexão com o banco de dados;
- Definir o modelo da tabela usando uma classe Python que herda da classe declarative_base() do SQLAlchemy. Essa classe deve conter a definição da tabela e seus campos;
- Consultar os registros que desejamos excluir usando a sessão e o método filter(). Podemos especificar os critérios de seleção usando a sintaxe de filtro do SQLAlchemy;
- Chamar o método delete() na sessão para excluir os registros selecionados;
- Confirmar a transação usando o método commit() da sessão.

O código Python para excluir registros de uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy é mostrado abaixo:

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Criando uma conexão com o banco de dados
engine = create_engine('postgresql://myuser:mypassword@localhost/mydatabase')

# Criando uma sessão
Session = sessionmaker(bind=engine)
session = Session()

# Definindo o modelo da tabela
Base = declarative_base()
class Usuario(Base):
    __tablename__ = 'usuarios'
    id = Column(Integer, primary_key=True)
    nome = Column(String)
    email = Column(String)

# Criando uma instância do modelo e preenchendo os valores dos campos
novo_usuario = Usuario(nome='João', email='joao@example.com')

# Salvando a instância no banco de dados
session.add(novo_usuario)

# Confirmar a transação
session.commit()

# Fechando a sessão
session.close()
```

Figura 16

Agora vamos ver um exemplo prático de como excluir registros de uma tabela em um banco de dados PostgreSQL usando a biblioteca SQLAlchemy em Python:

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.orm import sessionmaker
from sqlalchemy.ext.declarative import declarative_base

# Criando uma conexão com o banco de dados
engine = create_engine('postgresql://myuser:mypassword@localhost/mydatabase

# Criando uma sessão
Session = sessionmaker(bind=engine)
session = Session()

# Definindo o modelo da tabela
Base = declarative_base()
class Usuario(Base):
    __tablename__ = 'usuarios'
    id = Column(Integer, primary_key=True)
    nome = Column(String)
    email = Column(String)

# Criando uma instância do modelo e preenchendo os valores dos campos
novo_usuario = Usuario(nome='Jo
```

Figura 17

5. Utilizando a biblioteca PyGreSQL

O PyGreSQL é uma biblioteca de banco de dados PostgreSQL de baixo nível para Python que fornece uma interface Python para o banco de dados PostgreSQL. Com o PyGreSQL, podemos nos conectar a um banco de dados PostgreSQL, executar consultas SQL e manipular dados. É uma alternativa ao psycopg2 e também oferece suporte a recursos avançados, como transações, bloqueio e controle de concorrência.

O PyGreSQL é uma biblioteca Python que fornece acesso ao PostgreSQL usando o protocolo nativo do PostgreSQL. É uma biblioteca de baixo nível que permite executar consultas SQL e manipular dados de maneira eficiente. O PyGreSQL usa uma interface de objeto simples e oferece suporte a recursos avançados, como transações, bloqueio e controle de concorrência.

O PyGreSQL suporta a maioria dos tipos de dados do PostgreSQL, incluindo tipos de dados personalizados, e também oferece suporte a recursos avançados, como conexões persistentes e transações. O PyGreSQL é uma alternativa ao psycopg2 e é amplamente utilizado em aplicativos Python que se conectam ao PostgreSQL. Para usar o PyGreSQL, precisamos instalar a biblioteca usando o gerenciador de pacotes pip e, em seguida, criar uma conexão com o banco de dados usando as credenciais de conexão. Podemos executar consultas SQL na conexão usando o método query() da conexão.

O código Python para se conectar a um banco de dados PostgreSQL usando o PyGreSQL é mostrado abaixo:

```python
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd='

# Executando uma consulta SQL
result = conn.query("SELECT * FROM minha_tabela")

# Obtendo os resultados da consulta
for row in result:
    print(row)
```

Figura 18

Agora vamos ver um exemplo prático de como conectar-se a um banco de dados PostgreSQL e executar uma consulta SQL usando o PyGreSQL em Python:

```python
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd='

# Executando uma consulta SQL
result = conn.query("SELECT * FROM usuarios")

# Obtendo os resultados da consulta
for row in result.getresult():
    print(row)

# Fechando a conexão
conn.close()
```

Figura 19

Neste exemplo, estamos nos conectando a um banco de dados chamado "mydatabase" em um servidor local, usando um nome de usuário e uma senha válidos. Em seguida, estamos executando uma consulta SQL simples para recuperar todos os registros da tabela "usuarios". Finalmente, estamos iterando sobre os resultados da consulta e imprimindo cada linha. É importante fechar a conexão quando não precisamos mais dela para liberar recursos.

5.1. Inserindo dados

Para inserir dados em uma tabela do PostgreSQL usando o PyGreSQL em Python, podemos usar o método insert() da conexão. O método insert() recebe dois parâmetros: o nome da tabela onde os dados serão inseridos e um dicionário com as colunas e valores a serem inseridos.

O PyGreSQL usa o protocolo nativo do PostgreSQL para se comunicar com o banco de dados, tornando a inserção de dados muito eficiente. Ao usar o método insert() da conexão PyGreSQL, podemos inserir novas linhas em uma tabela do PostgreSQL com facilidade. O PyGreSQL também oferece suporte a transações, garantindo que as alterações de dados sejam confirmadas apenas quando a transação for confirmada.

Para inserir dados em uma tabela do PostgreSQL usando o PyGreSQL, precisamos primeiro criar uma conexão com o banco de dados usando as credenciais de conexão. Em seguida, podemos usar o método insert() para inserir novas linhas em uma tabela. O método insert() recebe dois parâmetros: o nome da tabela onde os dados serão inseridos e um dicionário com as colunas e valores a serem inseridos.

O código Python para inserir dados em uma tabela do PostgreSQL usando o PyGreSQL é mostrado abaixo:

```
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd='m

# Inserindo uma nova linha na tabela usuarios
data = {'nome': 'João', 'idade': 30, 'cidade': 'São Paulo'}
conn.insert('usuarios', data)

# Confirmando a transação
conn.commit()

# Fechando a conexão
conn.close()
```

Figura 20

Neste exemplo, estamos nos conectando a um banco de dados chamado "mydatabase" em um servidor local, usando um nome de usuário e uma senha válidos. Em seguida, estamos inserindo uma nova linha na tabela "usuarios" com os valores 'João', '30' e 'São Paulo' para as colunas 'nome', 'idade' e 'cidade', respectivamente. Finalmente, estamos confirmando a transação e fechando a conexão. É importante confirmar a transação para garantir que as alterações de dados sejam confirmadas no banco de dados.

Agora vamos ver um exemplo prático de como inserir dados em uma tabela do PostgreSQL usando o PyGreSQL em Python:

```
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd='

# Inserindo uma nova linha na tabela usuarios
data = {'nome': 'Maria', 'idade': 25, 'cidade': 'Rio de Janeiro'}
conn.insert('usuarios', data)

# Confirmando a transação
conn.commit()

# Fechando a conexão
conn.close()
```

Figura 21

Neste exemplo, estamos nos conectando a um banco de dados chamado "mydatabase" em um servidor local, usando um nome de usuário e uma senha válidos. Em seguida, estamos inserindo uma nova linha na tabela "usuarios" com os valores 'Maria', 25 e 'Rio de Janeiro' para as colunas 'nome', 'idade' e 'cidade', respectivamente. Finalmente, estamos confirmando a transação e fechando a conexão.

5.2. Alterando dados

Para atualizar dados em uma tabela do PostgreSQL usando o PyGreSQL em Python, podemos usar o método update() da conexão. O método update() recebe três parâmetros: o nome da tabela a ser atualizada, um dicionário com os novos valores para cada coluna e uma condição que determina quais linhas serão atualizadas.

O PyGreSQL usa o protocolo nativo do PostgreSQL para se comunicar com o banco de dados, tornando a atualização de dados muito eficiente. Ao usar o método update() da conexão PyGreSQL, podemos atualizar facilmente as linhas existentes em uma tabela do PostgreSQL. O PyGreSQL também oferece suporte a transações, garantindo que as alterações de dados sejam confirmadas apenas quando a transação for confirmada.

Para atualizar dados em uma tabela do PostgreSQL usando o PyGreSQL, precisamos primeiro criar uma conexão com o banco de dados usando as credenciais de conexão. Em seguida, podemos usar o método update() para atualizar as linhas na tabela. O método update() recebe três parâmetros: o nome da tabela a ser atualizada, um dicionário com os novos valores para cada coluna e uma condição que determina quais linhas serão atualizadas.

O código Python para atualizar dados em uma tabela do PostgreSQL usando o PyGreSQL é mostrado abaixo:

```
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd='

# Atualizando a idade de usuários com nome João
data = {'idade': 40}
where = "nome = 'João'"
conn.update('usuarios', data, where)

# Confirmando a transação
conn.commit()

# Fechando a conexão
conn.close()
```

Figura 22

Neste exemplo, estamos nos conectando a um banco de dados chamado "mydatabase" em um servidor local, usando um nome de usuário e senha válidos. Em seguida, estamos atualizando a idade dos usuários com nome 'João' para 40 na tabela "usuarios". Finalmente, estamos confirmando a transação e fechando a conexão. É importante confirmar a transação para garantir que as alterações de dados sejam confirmadas no banco de dados.

Agora vamos ver um exemplo prático de como atualizar dados em uma tabela do PostgreSQL usando o PyGreSQL em Python:

```
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd='

# Atualizando a idade de usuários com nome Maria
data = {'idade': 30}
where = "nome = 'Maria'"
conn.update('usuarios', data, where)

# Confirmando a transação
conn.commit()

# Fechando a conexão
conn.close()
```

Figura 23

Neste exemplo, estamos nos conectando a um banco de dados chamado "mydatabase" em um servidor local, usando um nome de usuário e uma senha válidos. Em seguida, estamos atualizando a idade dos usuários com nome 'Maria' para 30 na tabela "usuarios". Finalmente, estamos confirmando a transação e fechando a conexão.

5.3. Excluindo dados

Para excluir dados de uma tabela do PostgreSQL usando o PyGreSQL em Python, podemos usar o método query() da conexão para executar uma instrução SQL de exclusão. A instrução SQL de exclusão pode ser construída usando a cláusula DELETE com uma condição que determina quais linhas serão excluídas.

O PyGreSQL usa o protocolo nativo do PostgreSQL para se comunicar com o banco de dados, tornando a exclusão de dados muito eficiente. Ao usar o método query() da conexão PyGreSQL, podemos executar facilmente instruções SQL de exclusão em uma tabela do PostgreSQL. O PyGreSQL também oferece suporte a transações, garantindo que as alterações de dados sejam confirmadas apenas quando a transação for confirmada.

Para excluir dados de uma tabela do PostgreSQL usando o PyGreSQL, precisamos primeiro criar uma conexão com o banco de dados usando as credenciais de conexão. Em seguida, podemos construir uma instrução SQL de exclusão que usa a cláusula DELETE com uma condição que determina quais linhas serão excluídas. Por fim, executamos a instrução SQL usando o método query() da conexão PyGreSQL.

O código Python para excluir dados de uma tabela do PostgreSQL usando o PyGreSQL é mostrado abaixo:

```
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd=

# Excluindo usuários com idade maior que 50
where = "idade > 50"
conn.query("DELETE FROM usuarios WHERE " + where)

# Confirmando a transação
conn.commit()

# Fechando a conexão
conn.close()
```

Figura 24

Neste exemplo, estamos nos conectando a um banco de dados chamado "mydatabase" em um servidor local, usando um nome de usuário e uma senha válidos. Em seguida, estamos excluindo usuários com idade maior que 50 na tabela "usuarios" usando a instrução SQL DELETE. Finalmente, estamos confirmando a transação e fechando a conexão. É importante confirmar a transação para garantir que as alterações de dados sejam confirmadas no banco de dados.

Agora vamos ver um exemplo prático de como excluir dados de uma tabela do PostgreSQL usando o PyGreSQL em Python:

```
import pg

# Conectando ao banco de dados
conn = pg.DB(dbname='mydatabase', host='localhost', user='myuser', passwd='

# Excluindo usuários com idade menor que 25
where = "idade < 25"
conn.query("DELETE FROM usuarios WHERE " + where)

# Confirmando a transação
conn.commit()

# Fechando a conexão
conn.close()
```

Figura 25

Neste exemplo, estamos nos conectando a um banco de dados chamado "mydatabase" em um servidor local, usando um nome de usuário e uma senha válidos. Em seguida, estamos excluindo usuários com idade menor que 25 na tabela "usuarios" usando a instrução SQL DELETE. Finalmente, estamos confirmando a transação e fechando a conexão.

6. Utilizando a biblioteca Psql

O pacote Psql é uma ferramenta conveniente para executar comandos Psql diretamente do Python. Ele oferece uma maneira fácil de realizar consultas e comandos de banco de dados, sem precisar escrever um código SQL completo.

O Psql é construído em cima da biblioteca psycopg2 e usa suas classes e funções internamente para se comunicar com o banco de dados PostgreSQL. É uma opção interessante para aqueles que desejam uma abordagem mais simples e direta para lidar com as consultas SQL.

A seguir, apresento um exemplo teórico e prático de como utilizar o pacote Psql para realizar uma consulta simples em um banco de dados PostgreSQL:

- Instalando o pacote Psql:

```
pip install psql
```

- Importando o pacote:

```
from psql import Psql
```

- Criando uma conexão com o banco de dados:

```
db = Psql(database='nome_do_banco', user='usuario', password='senha',
host='localhost', port=5432)
```

- Executando uma consulta:

```
result = db.query("SELECT * FROM tabela")
```

- Exibindo o resultado da consulta:

```
for row in result:
    print(row)
```

O exemplo acima é um exemplo básico, mas é possível realizar uma ampla variedade de consultas e comandos SQL utilizando o pacote Psql. É importante lembrar que, embora o Psql facilite a comunicação com o banco de dados, ainda é necessário ter um conhecimento mínimo sobre SQL para realizar as consultas e comandos necessários.

6.1. Inserções com o pacote Psql

Para realizar inserções de dados utilizando o pacote Psql, você pode utilizar o método execute da classe Psql. O método 'execute' executa uma instrução SQL diretamente no banco de dados.

A seguir, apresento um exemplo tanto teórico quanto prático de como utilizar o pacote Psql para realizar uma inserção em um banco de dados PostgreSQL:

- Importando o pacote:

```
from psql import Psql
```

- Criando uma conexão com o banco de dados:

```
db = Psql(database='nome_do_banco', user='usuario', password='senha', host='localhost', port=5432)
```

- Criando uma instrução SQL para a inserção:

```
sql = "INSERT INTO tabela (coluna1, coluna2) VALUES (%s, %s)"
```

- Executando a instrução SQL com os valores desejados:

```
valores = ('valor_coluna1', 'valor_coluna2')
db.execute(sql, valores)
```

No exemplo acima, a variável sql contém a instrução SQL para a inserção de valores em uma tabela do banco de dados. O %s é um marcador de posição que será substituído pelos valores reais que serão inseridos na tabela.

A variável 'valores' contém os valores que serão inseridos nas colunas coluna1 e coluna2, respectivamente. Esses valores são passados como um tuple para o método 'execute'.

Após executar o método 'execute', a inserção é realizada no banco de dados. Você pode confirmar a inserção verificando os dados na tabela correspondente.

6.2. Alterações com o pacote Psql

Para realizar alterações de dados utilizando o pacote Psql, você pode utilizar o método 'execute' da classe Psql. O método 'execute' executa uma instrução SQL diretamente no banco de dados.
A seguir, apresento um exemplo tanto teórico quanto prático de como utilizar o pacote Psql para realizar uma alteração em um banco de dados PostgreSQL:

- **Importando o pacote:**

```
from psql import Psql
```

- **Criando uma conexão com o banco de dados:**

```
db = Psql(database='nome_do_banco', user='usuario', password='senha', host='localhost', port=5432)
```

- **Criando uma instrução SQL para a alteração:**

```
sql = "UPDATE tabela SET coluna1 = %s WHERE coluna2 = %s"
```

- **Executando a instrução SQL com os valores desejados:**

```
valores = ('novo_valor_coluna1', 'valor_coluna2')
db.execute(sql, valores)
```

No exemplo acima, a variável sql contém a instrução SQL para a alteração de valores em uma tabela do banco de dados. O %s é um marcador de posição que será substituído pelos valores reais que serão utilizados na alteração.

A variável 'valores' contém os novos valores que serão inseridos na coluna coluna1, onde a coluna coluna2 possuir o valor 'valor_coluna2'. Esses valores são passados como um tuple para o método 'execute'.

Após executar o método 'execute', a alteração é realizada no banco de dados. Você pode confirmar a alteração verificando os dados na tabela correspondente.

6.3. Exclusões com o Psql

Para realizar exclusões de dados utilizando o pacote Psql, você pode utilizar o método 'execute' da classe Psql. O método 'execute' executa uma instrução SQL diretamente no banco de dados.
A seguir, apresento um exemplo tanto teórico quanto prático de como utilizar o pacote Psql para realizar uma exclusão em um banco de dados PostgreSQL:

- **Importando o pacote:**

```
from psql import Psql
```

- **Criando uma conexão com o banco de dados:**

```
db = Psql(database='nome_do_banco', user='usuario', password='senha', host='localhost', port=5432)
```

- **Criando uma instrução SQL para a exclusão:**

```
sql = "DELETE FROM tabela WHERE coluna = %s"
```

- **Executando a instrução SQL com o valor desejado:**

```
valor = 'valor_coluna'
db.execute(sql, valor)
```

No exemplo acima, a variável sql contém a instrução SQL para a exclusão de valores em uma tabela do banco de dados. O %s é um marcador de posição que será substituído pelo valor real que será utilizado na exclusão.

A variável 'valor' contém o valor que será utilizado para encontrar os registros que serão excluídos. Esse valor é passado como um argumento para o método 'execute'.

Após executar o método 'execute', a exclusão é realizada no banco de dados. Você pode confirmar a exclusão verificando os dados na tabela correspondente.

Referências

BANIN, Sérgio Luiz. **Python 3 - Conceitos e Aplicações - Uma Abordagem Didática [BV:MB]**. São Paulo: Érica, 2018.

FORBELLONE, André L. V.; Eberspacher, Henri F**. Lógica de Programação: a construção de algoritmos e estruturas de dados [BV:PE]**. 3. ed. São Paulo: Editora Pearson, 2005.

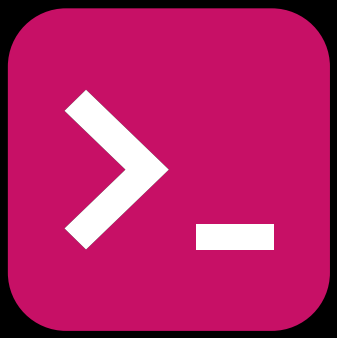
LIMA, Janssen dos Reis. **Consumindo a API do Zabbix com Python [BV:PE]**. Rio de Janeiro: Editora Brasport, 2016.

PERKOVIC, Ljubomir. **Introdução à Computação Usando Python - Um Foco no Desenvolvimento de Aplicações [BV:MB]**. 1ª ed. Rio de Janeiro: LTC, 2016.

SEBESTA,Robert W. **Conceitos de Linguagens de Programação**. 11. ed. Porto Alegre: Grupo A, 2011.

TUCKER, Allen; NOONAN, Robert. **Linguagens de Programação: Princípios e Paradigmas**. Porto Alegre: Grupo A, 2009.

VLADISHEV, A. **Consumindo a API do Zabbix com Python**. Rio de Janeiro: Brasport, 2016.



Digital College

ENSINO DE HABILIDADES DIGITAIS

@digitalcollegebr

/school/digitalcollegebr

/digitalcollegebr

digitalcollege.com.br