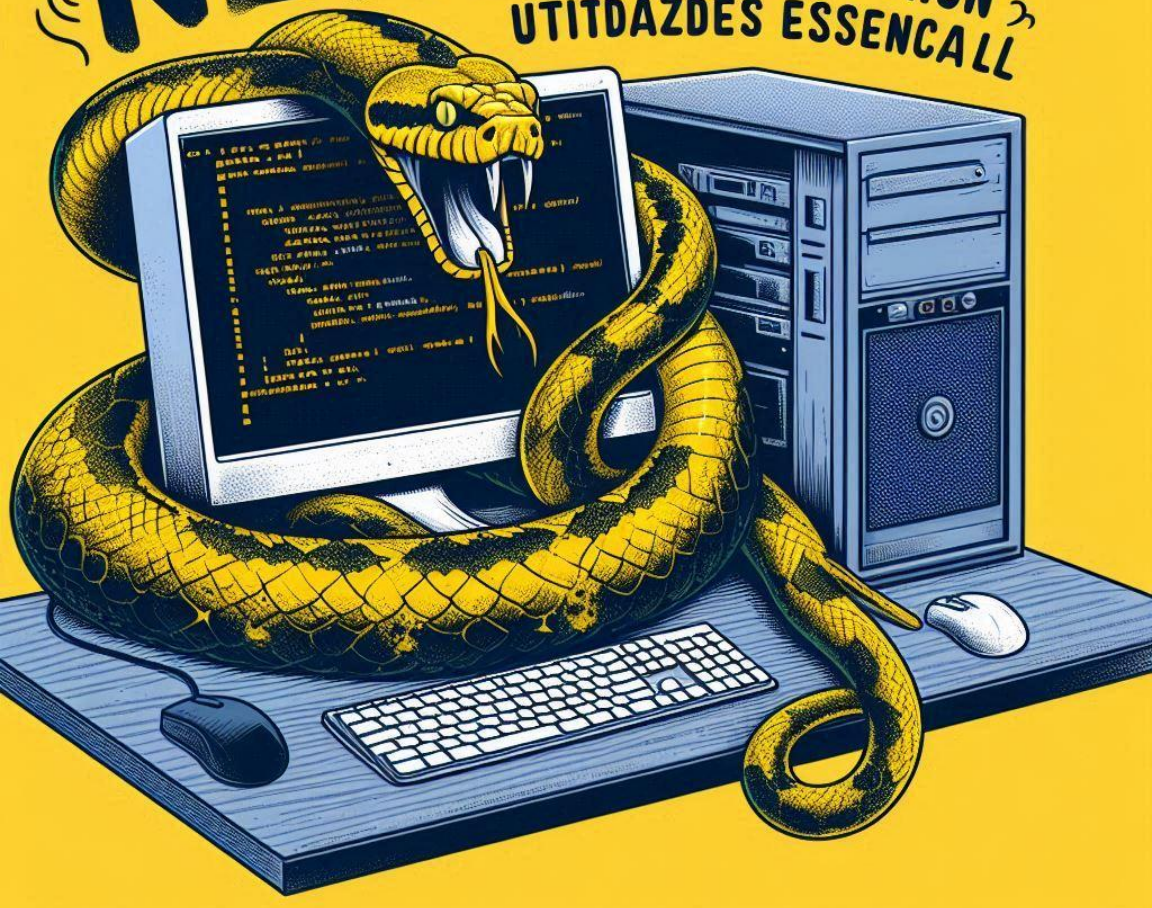
NERRDO COM PYTHON

UTITDAZDES ESSENCALL



Por Agatha G. Prieto

1 Conectando ao BD com Python

Neste capítulo, vamos explorar como conectar Python a um banco de dados MySQL, alterar dados existentes e incluir novos registros. Vamos usar a biblioteca “mysql.conector” para realizar essas tarefas de forma simples e eficiente.

Conectando ao BD com Python

Configurando o Ambiente

Antes de começar, precisamos instalar a biblioteca “mysql.conector”. Você pode fazer isso usando o seguinte comando:

Untitled-1

```
pip install mysql-connector-python
```

Conectando ao BD

Vamos iniciar com a conexão ao banco de dados. Para isso, precisamos das credenciais de acesso, como nome do host, nome do usuário, senha e nome do banco de dados.

Untitled-1

```
import mysql.connector

# Configurações do banco de dados
config = {
    'user': 'seu_usuario',
    'password': 'sua_senha',
    'host': 'localhost',
    'database': 'nome_do_banco'
}

# Conectando ao banco de dados
conn = mysql.connector.connect(**config)
```

Untitled-1

```python
# Verificando se a conexão foi bem-sucedida
if conn.is_connected():
    print('Conexão estabelecida com sucesso!')
else:
    print('Falha na conexão.')
```

Inserindo Dados

Agora que estamos conectados ao banco de dados, vamos inserir alguns dados em uma tabela. Suponha que temos uma tabela chamada 'clientes' com as colunas 'id', 'nome', e 'email'.

Untitled-1

```python
# Cursor para executar comandos SQL
cursor = conn.cursor()

# Comando SQL para inserir dados
sql = "INSERT INTO clientes (nome, email) VALUES (%s, %s)"
valores = ("João Silva", "joao.silva@example.com")

# Executando o comando SQL
cursor.execute(sql, valores)

# Confirmando a inserção
conn.commit()

print(cursor.rowcount, "registro(s) inserido(s).")
```

Atualizando Dados

Para atualizar dados existentes, usamos o comando SQL 'UPDATE'. Vamos atualizar o email de um cliente específico.

Untitled-1

```python
# Comando SQL para atualizar dados
sql = "UPDATE clientes SET email = %s WHERE nome = %s"
valores = ("joao.novoemail@example.com", "João Silva")

# Executando o comando SQL
cursor.execute(sql, valores)

# Confirmando a atualização
conn.commit()

print(cursor.rowcount, "registro(s) atualizado(s).")
```

Deletando Dados

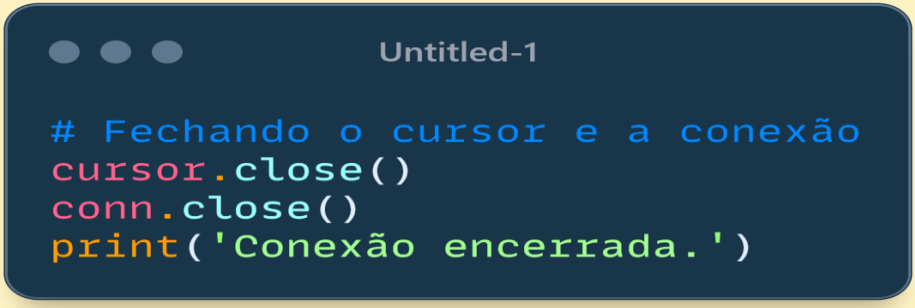
Para deletar dados de uma tabela, usamos o comando SQL 'DELETE'. Vamos deletar o cliente com o nome "João Silva".

Untitled-1

```python
# Comando SQL para deletar dados
sql = "DELETE FROM clientes WHERE nome = %s"
valores = ("João Silva",)

# Executando o comando SQL
cursor.execute(sql, valores)

# Confirmando a exclusão
conn.commit()

print(cursor.rowcount, "registro(s) deletado(s).")
```

Fechando a Conexão

Após realizar todas as operações, é importante fechar a conexão com o banco de dados para liberar recursos.



Resumo

Neste capítulo, aprendemos como:

- Conectar Python a um banco de dados MySQL
- Inserir novos registros
- Atualizar registros existentes
- Deletar registros

No próximo capítulo, vamos explorar como realizar consultas avançadas e trabalhar com os resultados de forma eficiente.

2
Consultas Avançadas com Python

Neste capítulo, vamos aprender como realizar consultas avançadas em um banco de dados MySQL usando Python. Também veremos como manipular e utilizar os resultados dessas consultas de forma prática.

Consultas Avançadas com Python

Realizando Consultas Simples

Para realizar uma consulta simples, usamos o comando SQL 'SELECT'. Vamos buscar todos os clientes da tabela clientes.

Untitled-1

```python
# Conectando ao banco de dados
conn = mysql.connector.connect(**config)
cursor = conn.cursor()

# Comando SQL para buscar todos os clientes
sql = "SELECT * FROM clientes"

# Executando a consulta
cursor.execute(sql)

# Obtendo os resultados
resultados = cursor.fetchall()

# Exibindo os resultados
for cliente in resultados:
    print(cliente)

# Fechando o cursor e a conexão
cursor.close()
conn.close()
```

Consultas com Filtros

Podemos filtrar os resultados usando a cláusula 'WHERE'. Vamos buscar os clientes cujo nome começa com 'J'.

Untitled-1

```python
# Conectando ao banco de dados
conn = mysql.connector.connect(**config)
cursor = conn.cursor()

# Comando SQL para buscar clientes cujo nome começa com 'J'
sql = "SELECT * FROM clientes WHERE nome LIKE 'J%'"

# Executando a consulta
cursor.execute(sql)

# Obtendo os resultados
resultados = cursor.fetchall()

# Exibindo os resultados
for cliente in resultados:
    print(cliente)

# Fechando o cursor e a conexão
cursor.close()
conn.close()
```

Consultas com Ordenação

Podemos ordenar os resultados usando a cláusula 'ORDER BY'. Vamos buscar todos os clientes e ordená-los pelo nome.

Untitled-1

```python
# Conectando ao banco de dados
conn = mysql.connector.connect(**config)
cursor = conn.cursor()

# Comando SQL para buscar todos os clientes ordenados pelo nome
sql = "SELECT * FROM clientes ORDER BY nome"
```

Untitled-1

```
# Executando a consulta
cursor.execute(sql)

# Obtendo os resultados
resultados = cursor.fetchall()

# Exibindo os resultados
for cliente in resultados:
    print(cliente)

# Fechando o cursor e a conexão
cursor.close()
conn.close()
```

Consultas com Limite

Podemos limitar o número de resultados usando a cláusula 'LIMIT'. Vamos buscar os 5 primeiros clientes.

Untitled-1

```
# Conectando ao banco de dados
conn = mysql.connector.connect(**config)
cursor = conn.cursor()

# Comando SQL para buscar os 5 primeiros clientes
sql = "SELECT * FROM clientes LIMIT 5"

# Executando a consulta
cursor.execute(sql)

# Obtendo os resultados
resultados = cursor.fetchall()
```

Untitled-1

```python
# Exibindo os resultados
for cliente in resultados:
    print(cliente)

# Fechando o cursor e a conexão
cursor.close()
conn.close()
```

Manipulando os Resultados

Os resultados das consultas podem ser manipulados de diversas formas. Vamos calcular a quantidade total de clientes

Untitled-1

```python
# Conectando ao banco de dados
conn = mysql.connector.connect(**config)
cursor = conn.cursor()

# Comando SQL para contar o número de clientes
sql = "SELECT COUNT(*) FROM clientes"

# Executando a consulta
cursor.execute(sql)

# Obtendo o resultado
total_clientes = cursor.fetchone()[0]

# Exibindo o total de clientes
print(f'Total de clientes: {total_clientes}')

# Fechando o cursor e a conexão
cursor.close()
conn.close()
```

Consultas com Junção de Tabelas

Vamos realizar uma consulta que envolve a junção de duas tabelas. Suponha que temos uma tabela pedidos que registra os pedidos feitos pelos clientes. Vamos buscar os pedidos juntamente com os nomes dos clientes.

Untitled-1

```
# Conectando ao banco de dados
conn = mysql.connector.connect(**config)
cursor = conn.cursor()

# Comando SQL para buscar pedidos com os nomes dos clientes
sql = """
SELECT pedidos.id, clientes.nome, pedidos.data_pedido
FROM pedidos
JOIN clientes ON pedidos.cliente_id = clientes.id
"""
# Executando a consulta
cursor.execute(sql)

# Obtendo os resultados
resultados = cursor.fetchall()

# Exibindo os resultados
for pedido in resultados:
    print(pedido)

# Fechando o cursor e a conexão
cursor.close()
conn.close()
```

Resumo

Neste capítulo, aprendemos como:

- Realizar consultas simples e avançadas
- Filtrar, ordenar e limitar resultados
- Manipular os resultados das consultas
- Realizar junção de tabelas

No próximo capítulo, vamos explorar como tratar erros e exceções ao trabalhar com bancos de dados em Python.

3 Inserindo Dados de uma Planilha no Banco de Dados

Neste capítulo, vamos aprender como ler dados de uma planilha Excel e inseri-los em um banco de dados MySQL. Para isso, usaremos a biblioteca ‘pandas’ para manipular a planilha e a biblioteca ‘mysql.conector’ para a conexão com o banco de dados.

Inserindo Dados de uma Planilha no Banco de Dados

Configurando o Ambiente

Primeiro, precisamos instalar as bibliotecas necessárias. Podemos fazer isso usando o seguinte comando:

Untitled-1

```
pip install pandas mysql-connector-python openpyxl
```

Lendo Dados da Planilha

Vamos começar lendo os dados de uma planilha Excel usando a biblioteca pandas. Suponha que temos uma planilha chamada 'clientes.xlsx' com as colunas 'nome' e 'email'.

Untitled-1

```
import pandas as pd

# Lendo a planilha Excel
file_path = "caminho/para/sua/planilha/clientes.xlsx"
df = pd.read_excel(file_path)

# Exibindo os dados da planilha
print(df.head())
```

Conectando ao Banco de Dados

Em seguida, vamos configurar a conexão com o banco de dados MySQL.

Untitled-1

```python
import mysql.connector

# Configurações do banco de dados
config = {
    'user': 'seu_usuario',
    'password': 'sua_senha',
    'host': 'localhost',
    'database': 'nome_do_banco'
}

# Conectando ao banco de dados
conn = mysql.connector.connect(**config)

# Verificando se a conexão foi bem-sucedida
if conn.is_connected():
    print('Conexão estabelecida com sucesso!')
else:
    print('Falha na conexão.')
```

Inserindo Dados no Banco de Dados

Agora, vamos inserir os dados da planilha na tabela clientes do banco de dados.

Untitled-1

```python
# Cursor para executar comandos SQL
cursor = conn.cursor()
```

Resumo

Neste capítulo, aprendemos como:

- Ler dados de uma planilha Excel usando pandas
- Conectar ao banco de dados MySQL
- Inserir dados da planilha em uma tabela do banco de dados

No próximo capítulo, vamos explorar como atualizar dados no banco de dados com base nas informações de uma planilha.

4 Enviando Emails com Python

Neste capítulo, vamos aprender como enviar emails usando Python, incluindo a anexação de arquivos. Para isso, utilizaremos a biblioteca smtplib para enviar emails e a biblioteca email para criar o conteúdo do email.

Enviando Emails com Python

Configurando o Ambiente

Antes de começar, precisamos garantir que temos as bibliotecas necessárias. As bibliotecas 'smtplib' e 'email' são incluídas na biblioteca padrão do Python, então não é necessário instalá-las separadamente.

Enviando um Email Simples

Vamos começar enviando um email simples. Para isso, precisaremos das credenciais do servidor de email (SMTP), como servidor, porta, email e senha.

Untitled-1

```
import smtplib
from email.mime.text import MIMEText

# Configurações do servidor SMTP
smtp_server = "smtp.seuprovedor.com"
port = 587
sender_email = "seu_email@example.com"
password = "sua_senha"
receiver_email = "destinatario@example.com"

# Criação do conteúdo do email
subject = "Assunto do Email"
body = "Este é o corpo do email."

# Criação do objeto MIMEText
msg = MIMEText(body)
msg['Subject'] = subject
msg['From'] = sender_email
msg['To'] = receiver_email
```

Untitled-1

```python
# Envio do email
try:
    server = smtplib.SMTP(smtp_server, port)
    server.starttls()  # Segurança
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, msg.as_string())
    print("Email enviado com sucesso!")
except Exception as e:
    print(f"Erro ao enviar email: {e}")
finally:
    server.quit()
```

Enviando um Email com Anexo

Para enviar um email com anexo, precisamos usar o módulo MIMEMultipart e adicionar o arquivo ao email.

Untitled-1

```python
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders

# Configurações do servidor SMTP
smtp_server = "smtp.seuprovedor.com"
port = 587
sender_email = "seu_email@example.com"
password = "sua_senha"
receiver_email = "destinatario@example.com"
```

Untitled-1

```python
# Criação do conteúdo do email
subject = "Assunto do Email com Anexo"
body = "Este é o corpo do email com anexo."

# Criação do objeto MIMEMultipart
msg = MIMEMultipart()
msg['From'] = sender_email
msg['To'] = receiver_email
msg['Subject'] = subject

# Anexando o corpo do email
msg.attach(MIMEText(body, 'plain'))

# Anexando o arquivo
filename = "documento.pdf"  # Nome do arquivo a ser anexado
filepath = "/caminho/para/o/arquivo/documento.pdf"  # Caminho para o arquivo

with open(filepath, "rb") as attachment:
    part = MIMEBase("application", "octet-stream")
    part.set_payload(attachment.read())

# Codificando em base64
encoders.encode_base64(part)
part.add_header(
    "Content-Disposition",
    f"attachment; filename= {filename}",
)

# Anexando o arquivo ao email
msg.attach(part)

# Envio do email
try:
    server = smtplib.SMTP(smtp_server, port)
    server.starttls()  # Segurança
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, msg.as_string())
    print("Email com anexo enviado com sucesso!")
except Exception as e:
    print(f"Erro ao enviar email: {e}")
finally:
    server.quit()
```

Resumo

Neste capítulo, aprendemos como:

- Enviar um email simples usando Python
- Anexar arquivos a um email e enviá-lo

No próximo capítulo, vamos explorar como personalizar o conteúdo do email, incluindo HTML e imagens embutidas.

5 Automação e Scripting com Python

Neste capítulo, vamos explorar como usar Python para automatizar tarefas comuns e repetitivas, como renomear arquivos em massa, enviar notificações e realizar backups automáticos. Python é uma ferramenta poderosa para scripting devido à sua simplicidade e a vasta quantidade de bibliotecas disponíveis.

Automação e Scripting com Python

Renomeando Arquivos em Massa

Um dos exemplos clássicos de automação é renomear arquivos em massa. Vamos usar a biblioteca os para realizar essa tarefa.

Untitled-1

```python
import os

# Diretório contendo os arquivos
diretorio = "/caminho/para/o/diretorio"

# Prefixo a ser adicionado aos arquivos
prefixo = "novo_"

# Iterando sobre os arquivos no diretório
for nome_arquivo in os.listdir(diretorio):
    # Novo nome do arquivo
    novo_nome = prefixo + nome_arquivo
    # Caminho completo do arquivo antigo e novo
    caminho_antigo = os.path.join(diretorio, nome_arquivo)
    caminho_novo = os.path.join(diretorio, novo_nome)
    # Renomeando o arquivo
    os.rename(caminho_antigo, caminho_novo)

print("Arquivos renomeados com sucesso!")
```

Enviando Notificações

Podemos usar a biblioteca 'plyer' para enviar notificações no desktop. Isso é útil para alertar sobre a conclusão de tarefas ou eventos importantes.

Untitled-1

```python
from plyer import notification

# Enviando uma notificação
notification.notify(
    title='Tarefa Concluída',
    message='A automação de renomear arquivos foi concluída com sucesso!',
    app_name='Automação com Python'
)
```

Realizando Backups Automáticos

Automatizar backups de arquivos é uma prática importante para garantir a segurança dos dados. Vamos copiar arquivos de um diretório para outro usando a biblioteca shutil.

Untitled-1

```python
import shutil

# Diretório de origem e destino
origem = "/caminho/para/o/diretorio/origem"
destino = "/caminho/para/o/diretorio/destino"

# Copiando arquivos do diretório de origem para o destino
for nome_arquivo in os.listdir(origem):
    caminho_origem = os.path.join(origem, nome_arquivo)
    caminho_destino = os.path.join(destino, nome_arquivo)
    shutil.copy(caminho_origem, caminho_destino)

print("Backup concluído com sucesso!")
```

Automatizando Envio de Emails com Relatórios

Vamos enviar um relatório por email automaticamente, combinando habilidades que aprendemos nos capítulos anteriores.

Untitled-1

```python
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
import pandas as pd

# Lendo os dados do relatório
file_path = "caminho/para/relatorio.xlsx"
df = pd.read_excel(file_path)

# Configurações do servidor SMTP
smtp_server = "smtp.seuprovedor.com"
port = 587
sender_email = "seu_email@example.com"
password = "sua_senha"
receiver_email = "destinatario@example.com"

# Criação do conteúdo do email
subject = "Relatório Semanal"
body = "Segue em anexo o relatório semanal."
```

Untitled-1

```python
# Criação do objeto MIMEMultipart
msg = MIMEMultipart()
msg['From'] = sender_email
msg['To'] = receiver_email
msg['Subject'] = subject

# Anexando o corpo do email
msg.attach(MIMEText(body, 'plain'))

# Anexando o arquivo
filename = "relatorio.xlsx"
with open(file_path, "rb") as attachment:
    part = MIMEBase("application", "octet-stream")
    part.set_payload(attachment.read())

# Codificando em base64
encoders.encode_base64(part)
part.add_header(
    "Content-Disposition",
    f"attachment; filename= {filename}",
)

# Anexando o arquivo ao email
msg.attach(part)

# Envio do email
try:
    server = smtplib.SMTP(smtp_server, port)
    server.starttls()  # Segurança
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, msg.as_string())
    print("Email com relatório enviado com sucesso!")
except Exception as e:
    print(f"Erro ao enviar email: {e}")
finally:
    server.quit()
```

Resumo

Neste capítulo, aprendemos como:

- Renomear arquivos em massa
- Enviar notificações no desktop
- Realizar backups automáticos
- Enviar relatórios por email automaticamente

No próximo capítulo, vamos explorar como interagir com APIs para automatizar a coleta e manipulação de dados.

Agradecimentos

Escrever este eBook foi uma experiência incrível e gratificante. Poder compartilhar conhecimento sobre Python e suas aplicações práticas é algo que sempre me motivou, e ver esse projeto ganhar vida é extremamente satisfatório.

Gostaria de expressar minha profunda gratidão à DIO e ao Santander, que proporcionaram o desafio que deu origem a este eBook. Este desafio foi uma oportunidade única para expandir meus conhecimentos, aplicar habilidades de maneira prática e contribuir para a comunidade de desenvolvedores.

Este eBook reflete uma parte significativa do meu dia a dia, pois trabalhar com Python e explorar suas inúmeras possibilidades é uma paixão constante. Foi essa paixão que me inspirou a escrever sobre o tema, na esperança de ajudar outros a descobrir o poder e a simplicidade dessa linguagem.

Agradeço a todos os leitores por dedicarem seu tempo e interesse. Espero que este eBook tenha sido útil e inspirador para vocês, assim como foi para mim ao escrevê-lo. Que ele sirva como um guia prático e fácil de seguir para todos que desejam aprimorar suas habilidades em Python e explorar o vasto mundo da automação e do desenvolvimento de software.

Continuem aprendendo, explorando e criando. O conhecimento é uma jornada contínua e sempre há novas descobertas e inovações esperando por nós.

Obrigado e boa sorte em suas futuras aventuras com Python!