Banco de dados

Guilherme Arthur de Carvalho

Analista de sistemas

https://linktr.ee/decarvalhogui

Objetivo Geral

Vamos abordar os conceitos básicos de banco de dados e como podemos interagir com eles usando a DB API em Python.

Pré-requisitos

- ✓ Python e VSCode
- ✓ SQLite

Conteúdo

- ❑ Introdução aos Bancos de dados Relacionais
- ❑ Python DB API
- ❑ Boas práticas em consultas SQL
- ❑ Gerenciando transações

Introdução aos Bancos de dados Relacionais

Banco de dados

O que é um banco de dados?

Os bancos de dados são coleções organizadas de dados, geralmente armazenados e acessados eletronicamente a partir de um sistema de computador.

Tipos de Bancos de dados

Existem vários tipos de bancos de dados, incluindo relacionais, não relacionais, orientados a objetos e muito mais. O tipo mais comum é o banco de dados relacional, que organiza os dados em tabelas.

O papel do SGBD

Os sistemas de Gerenciamento de Banco de Dados (SGBD) são softwares que interagem com o usuário, outras aplicações e o próprio banco de dados para capturar e analisar os dados. Existem muitos SGBDs disponíveis no merdaco, alguns dos mais populares incluem: MySQL, PostgreSQL, SQLite, Oracle Database, Microsoft SQL Server e MariaDB.

Introdução aos Banco de Dados Relacionais

Um banco de dados relacional é um tipo de banco de dados que organiza os dados em tabelas. Cada tabela é composta de linhas, que representam registros individuais, e colunas, que representam campos de dados.

Tabelas

Em um banco de dados relacional, uma tabela é uma estrutura que organiza os dados em linhas e colunas, semelhante a uma planilha.

Cada linha representa um registro distinto e cada coluna representa um tipo de informação, chamado de campo. Por exemplo, uma tabela 'Clientes' pode ter campos como 'ID', 'nome', 'email' e 'telefone'.

Chaves primárias

Cada tabela em um banco de dados relacional deve ter uma chave primária. A chave primária é uma coluna (ou conjunto de colunas) cujo valor é único para cada registro. Isso garante que cada registro na tabela possa ser identificado de maneira única. Por exemplo, na tabela 'Clientes', o campo 'ID' pode ser a chave primária.

Chaves estrangeiras

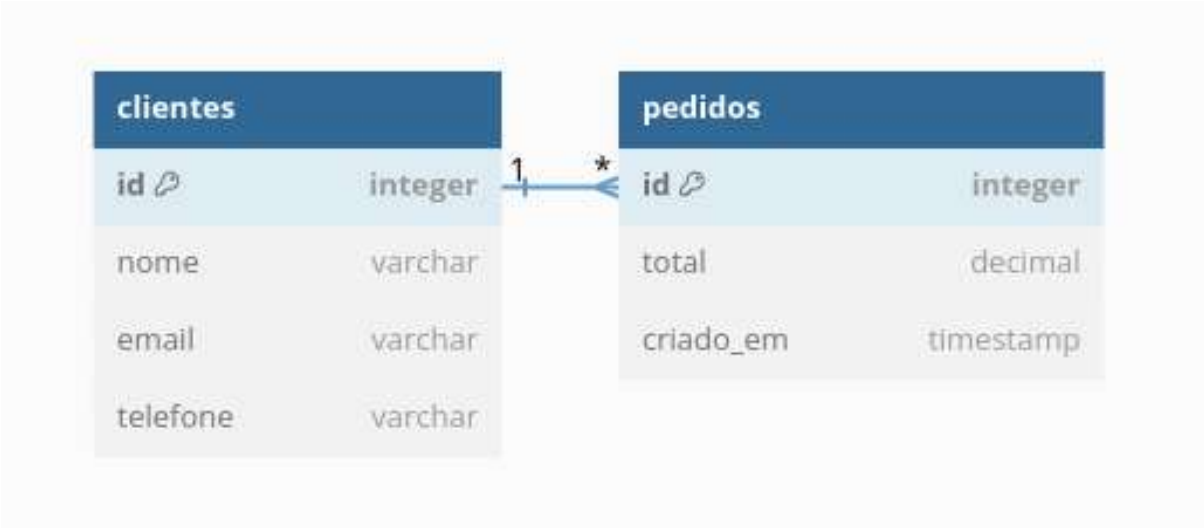
As chaves estrangeiras são usadas para estabelecer relações entre as tabelas. Uma chave estrangeira é um campo em uma tabela que corresponde à chave primária de outra tabela.

Chaves estrangeiras

Por exemplo, em uma tabela 'Pedidos', podemos ter um campo 'ClienteID' que seja uma chave estrangeira apontando para a chave primária da tabela 'Clientes'. Isso cria um relacionamento entre 'Pedidos' e 'Clientes', permitindo que cada pedido seja associado a um cliente específico.

Relacionamento entre tabelas

Os bancos de dados relacionais permitem estabelecer relações entre tabelas. As relações podem ser 'um para um', 'um para muitos', ou 'muitos para muitos'. Estas relações permitem efetuar consultas complexas que unem dados de várias tabelas.


clientes
id integer
nome varchar
email varchar
telefone varchar
1
*
pedidos
id integer
total decimal
criado_em timestamp

[15]

SQL (Structured Query Language)

O SQL é a linguagem usada para interagir com bancos de dados relacionais. Com SQL, podemos criar tabelas, inserir, atualizar e deletar registros, bem como executar consultas para buscar dados.

Exemplo de código:

```sql
-- Cria um novo banco de dados
CREATE DATABASE loja;
-- Cria uma tabela para armazenar informações de produtos
CREATE TABLE produtos (id INTEGER PRIMARY KEY AUTOINCREMENT, nome VARCHAR(100), preco DECIMAL);
-- Inclui um novo produto
INSERT INTO produtos (nome, preco) VALUES ('Curso de Python', 250.00);
-- Lista todos os produtos
SELECT * FROM produtos;
-- Atualiza o produto com id informado
UPDATE produtos SET nome='Curso de Python para iniciantes' WHERE id = 1;
-- Exclui o produto com id informado
DELETE FROM produtos WHERE id = 1;
```

Python DB API

Banco de dados

Conectando-se a um Banco de dados

A primeira etapa para trabalhar com um banco de dados é estabelecer uma conexão. Vamos ver como podemos fazer isso usando Python DB API.

Exemplo de código:

```python
import sqlite3
con = sqlite3.connect('meu_banco_de_dados.db')
```

Inserindo registros

Inserir registros em um banco de dados é uma operação comum. Com a Python DB API, usamos a operação INSERT do SQL para isso.

Exemplo de código:

```
data = (4, 'abcd')
cursor.execute('INSERT INTO minha_tabela VALUES (?, ?)', data)
conn.commit()
```

Atualizando registros

A operação UPDATE do SQL é usada para modificar registros existentes. É importante ser específico ao usar o UPDATE para evitar alterar mais registros do que o planejado.

Exemplo de código:

```
data = ('abcde', 4)
cursor.execute('UPDATE minha_tabela SET name = ? WHERE id = ?', data)
con.commit()
```

Deletando registros

A operação DELETE do SQL é usada para remover registros. Novamente, precisamos ser específicos ao usar o DELETE para evitar remover mais registros do que o planejado.

Exemplo de código:

```
id = 4
cursor.execute('DELETE FROM minha_tabela WHERE id = ?', (id,))
con.commit()
```

Operações em lote

Operações em lote são úteis quando precisamos inserir muitos registros de uma vez. Com Python DB API, podemos usar o método 'executemany()' para isso.

Exemplo de código:

```
data = [(5, 'abcde'), (6, 'abcdef'), (7, 'abcdefg')]
cursor.executemany('INSERT INTO minha_tabela VALUES (?, ?)', data)
con.commit()
```

Consultas com único resultado

O método 'fetchone()' pode ser usado para recuperar um único registro de resultado. Ele retorna o próximo registro na lista de resultados ou 'None' se não houver mais resultados.

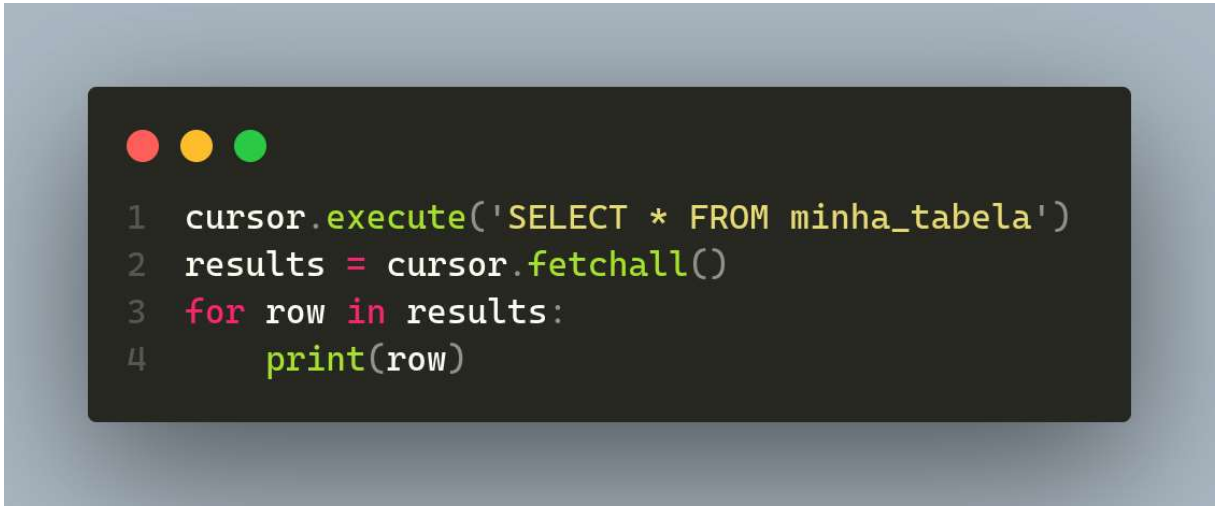
Exemplo de código:

```
1  cursor.execute('SELECT * FROM minha_tabela WHERE id = 1')
2  result = cursor.fetchone()
3  print(result)
```

Consultas com múltiplos resultado

O método 'fetchall()' pode ser usado para recuperar todos os registros de resultados de uma vez. Ele retorna uma lista de registros ou uma lista vazia se não houver mais resultados.

Exemplo de código:



```python
cursor.execute('SELECT * FROM minha_tabela')
results = cursor.fetchall()
for row in results:
    print(row)
```

Trabalhando com resultados de consulta

Os resultados das consultas são retornados como tuplas por padrão. Se a tupla não atender as nossas necessidades podemos usar a classe 'sqlite3.Row' ou uma 'row_factory' customizada.

Exemplo de código:

```
cursor.row_factory = sqlite3.Row
cursor.execute('SELECT * FROM minha_tabela WHERE id = 1')
result = cursor.fetchone()
print(dict(result))
```

Boas práticas em consultas SQL

Banco de dados

Introdução

Ao escrever consultas SQL em Python, é importante seguir boas práticas para garantir a segurança e a eficiência do seu código.

Pensando em segurança

Uma dessas práticas é evitar a concatenação de strings nas consultas e usar consultas parametrizadas. Isso não apenas melhora a legibilidade do código, mas também ajuda a prevenir ataques de injeção SQL.

Exemplo de código:

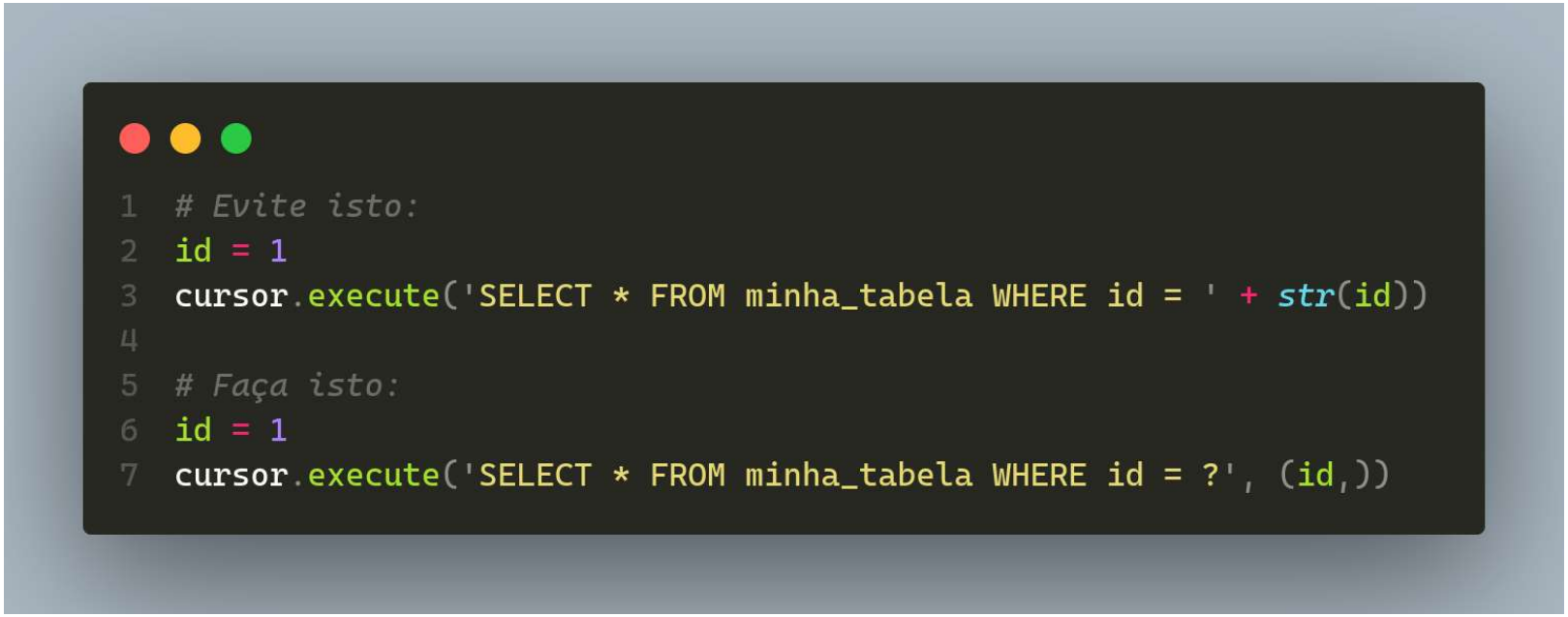
```python
# Evite isto:
id = 1
cursor.execute('SELECT * FROM minha_tabela WHERE id = ' + str(id))

# Faça isto:
id = 1
cursor.execute('SELECT * FROM minha_tabela WHERE id = ?', (id,))
```

Injeção SQL: Uma brecha de segurança

Uma dessas práticas é evitar a concatenação de strings nas consultas e usar consultas parametrizadas. Isso não apenas melhora a legibilidade do código, mas também ajuda a prevenir ataques de injeção SQL.

Exemplo de código:



```python
# Evite isto:
id = 1
cursor.execute('SELECT * FROM minha_tabela WHERE id = ' + str(id))

# Faça isto:
id = 1
cursor.execute('SELECT * FROM minha_tabela WHERE id = ?', (id,))
```

Gerenciando Transações

Banco de dados

Introdução

A DB API também nos permite gerenciar transações, o que é crucial para manter a integridade dos dados.

Exemplo de código:

```python
try:
    cursor.execute('INSERT INTO minha_tabela VALUES (?, ?)', (1, 'abc'))
    conn.commit()
except Exception as e:
    print(f"Ocorreu um erro: {e}")
    conn.rollback()
```

Dúvidas?

> Fórum/Artigos - https://web.dio.me/articles

Exemplo de código:

Insira sua imagem dentro deste espaço
(retire o retângulo azul, ele deverá ser utilizado somente para referência)