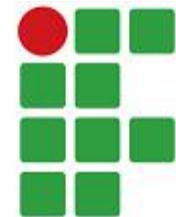


PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática

Lucas Sampaio Leite



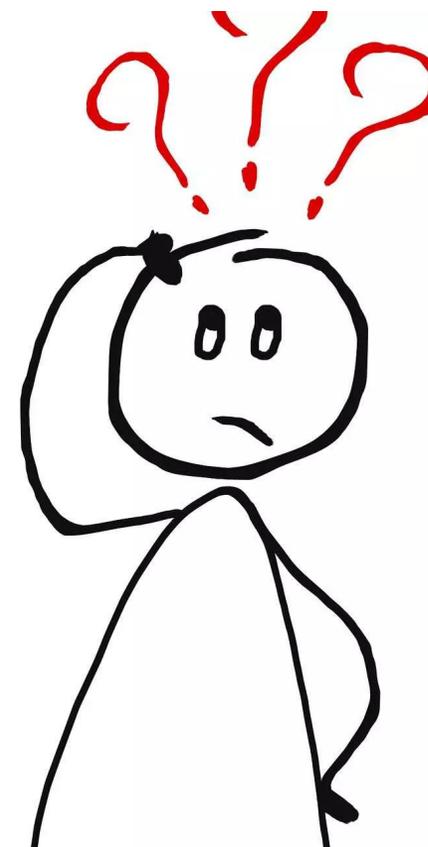
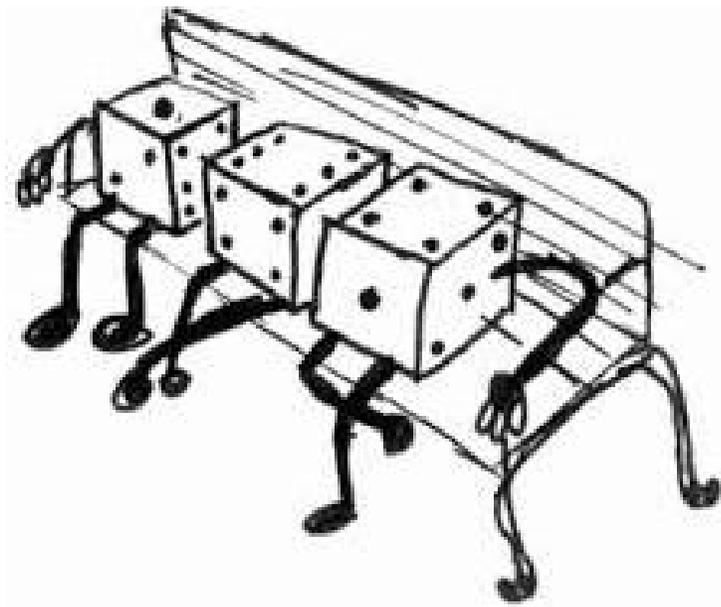
**INSTITUTO
FEDERAL**

Baiano

DB-API

- A DB-API (ou Python Database API Specification) é uma especificação padrão definida em Python para permitir que desenvolvedores se conectem e interajam com bancos de dados relacionais de forma consistente, independentemente do banco de dados usado (como SQLite, PostgreSQL, MySQL, etc.).
- Essa especificação é formalmente chamada de PEP 249 (Python Enhancement Proposal 249), e define uma interface comum para bibliotecas que oferecem acesso a bancos de dados relacionais.

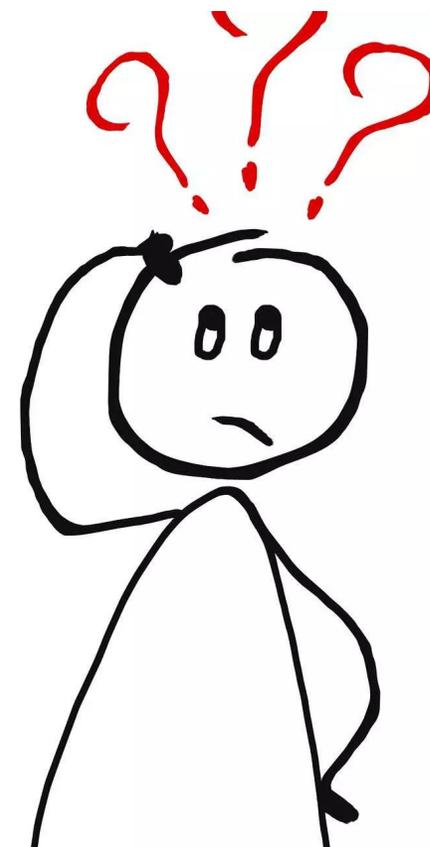
O que é um banco de dados?



O que é um banco de dados?

- Um banco de dados é uma coleção organizada de dados, que pode ser facilmente acessada, gerenciada e atualizada.
- Ele é usado para armazenar informações de forma estruturada, permitindo que programas e usuários realizem consultas, inserções, atualizações e remoções de forma eficiente.

O que é um SGBD?



O que é um SGBD?

- Um SGBD (Sistema de Gerenciamento de Banco de Dados) é um software responsável por criar, organizar, manipular e proteger os dados em um banco de dados. Ele funciona como um intermediário entre o usuário (ou um programa) e o banco de dados físico (os arquivos que armazenam os dados).
 - Imagine o banco de dados como uma biblioteca (onde os livros são os dados), e o SGBD é o bibliotecário que sabe onde tudo está, controla quem pode pegar livros, evita bagunça e garante que tudo seja devolvido corretamente.

Tipos de SGBDs

Tipo	Exemplo	Característica principal
Relacional (SQL)	MySQL, PostgreSQL, SQLite	Usa tabelas, colunas e SQL para consultas
Não relacional (NoSQL)	MongoDB, Redis	Usa documentos, pares chave-valor, etc
Em memória	Redis, H2	Muito rápido, mas volátil
Distribuído	Cassandra, Amazon DynamoDB	Dados espalhados por vários servidores

SQLite

- SQLite é um sistema de gerenciamento de banco de dados relacional gratuito e de código aberto, desenvolvido em linguagem C. Ele é leve, rápido e embutido, ou seja, não requer um servidor separado para operar.
- No Python, o SQLite já está integrado por meio do módulo padrão sqlite3, o que permite criar e manipular bancos de dados com facilidade, sem a necessidade de instalar bibliotecas adicionais.

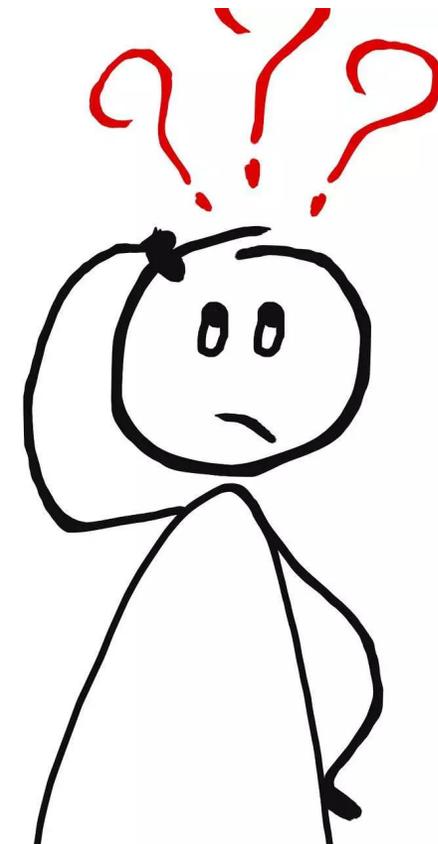
Verificando a instalação do SQLite

- No terminal ou o prompt de comando e digite python
- No console do Python, importar o módulo sqlite3:

```
import sqlite3  
print(sqlite3.sqlite_version)
```

- Se o módulo estiver disponível (o que é o padrão), ele irá mostrar a versão do SQLite.

O que são bancos de dados relacionais?



O que são bancos de dados relacionais?

- Um banco de dados relacional é um tipo de banco de dados que organiza os dados em tabelas (também chamadas de "entidades"), com linhas e colunas, de forma semelhante a uma planilha.
 - Baseado no modelo relacional proposto por Edgar F. Codd em 1970.
- Estrutura básica:
 - Tabelas (entidades): cada tabela representa um tipo de dado (ex: Clientes, Pedidos).
 - Linhas (registros): cada linha representa uma entrada específica.
 - Colunas (atributos): cada coluna representa uma característica (ex: nome, idade).

O que são bancos de dados relacionais?

- Exemplo: tabela Clientes

id_cliente	nome	cidade
1	Ana	Recife - PE
2	João	Salvador - BA
3	Maria	Senhor do Bonfim - BA

Chave primária

- Uma chave primária é um campo (ou conjunto de campos) em uma tabela de banco de dados que é usado para identificar de maneira única cada registro dessa tabela.
- A chave primária garante que cada linha na tabela seja única e não haja duplicação de dados.
- Características:
 - Uniqueness (unicidade).
 - Não pode ser nulo.
 - Indexação automática.

Chave estrangeira

- Uma chave estrangeira (ou foreign key) é um campo (ou conjunto de campos) em uma tabela de banco de dados que estabelece um vínculo com a chave primária de outra tabela.
- Ela é usada para garantir a integridade referencial, ou seja, assegurar que os dados em uma tabela estejam corretamente relacionados com os dados de outra.

Chave estrangeira

- Exemplo: tabela Clientes

id_cliente	nome	cidade
1	Ana	Recife - PE
2	João	Salvador - BA
3	Maria	Senhor do Bonfim - BA

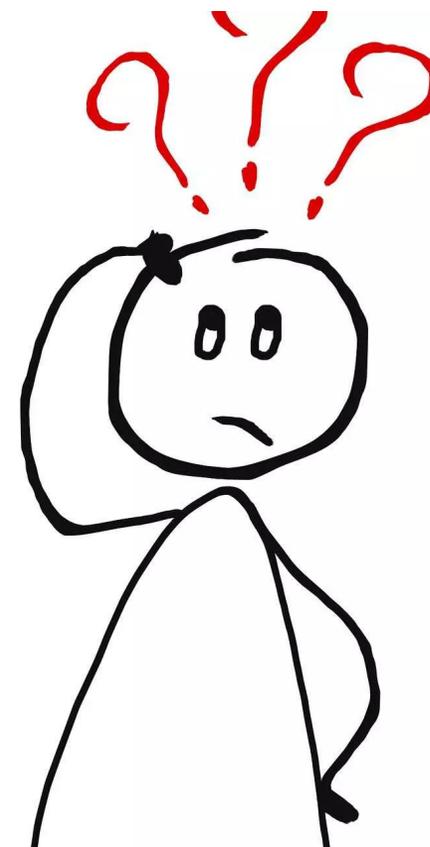
- Exemplo: tabela Pedidos

id_pedido	id_cliente(FK)	data
1	1	2025-01-01
2	2	2025-01-02

Chave estrangeira

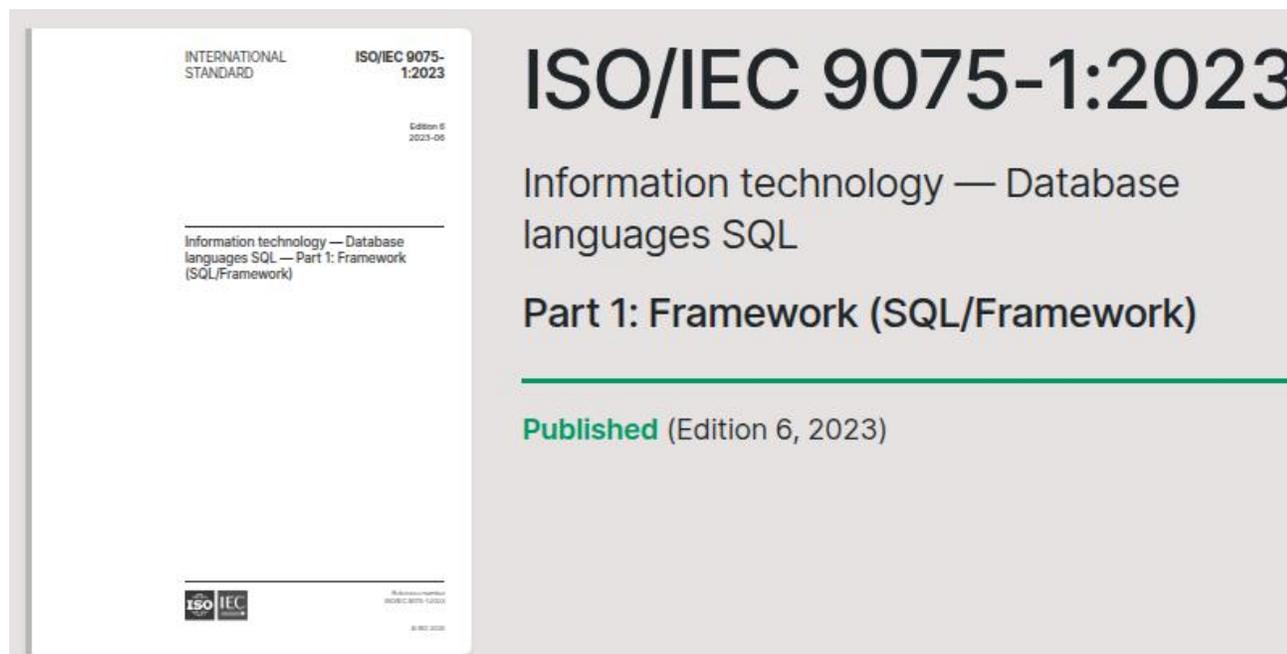
- Benefícios da chave estrangeira:
 - Evita dados "órfãos" (por exemplo, pedidos vinculados a clientes que não existem).
 - Estabelece relações entre tabelas (essencial em bancos relacionais).
 - Pode ser usada para ações automáticas (como ON DELETE CASCADE).

O que é SQL?



O que é SQL?

- SQL (Structured Query Language) é uma linguagem de consulta estruturada usada para criar, manipular e consultar bancos de dados relacionais.
- É a linguagem padrão para se comunicar com sistemas de gerenciamento de banco de dados como MySQL, PostgreSQL, SQL Server, Oracle, entre outros.



<https://www.iso.org/standard/76584.html>

Principais funcionalidades do SQL

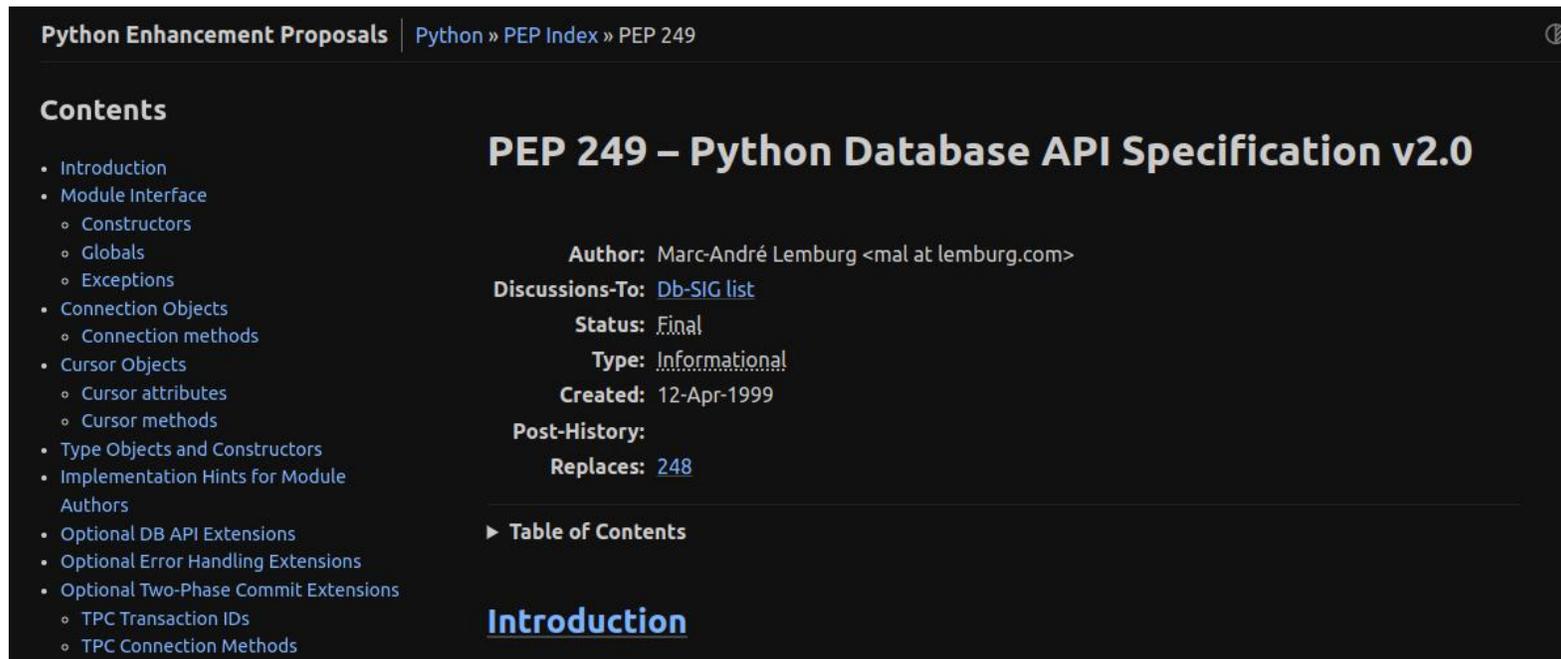
- Definir estruturas de dados
 - Categoria: DDL (Data Definition Language)
 - Comandos: CREATE, ALTER, DROP, TRUNCATE
- Manipular dados
 - Categoria: DML (Data Manipulation Language)
 - Comandos: INSERT, UPDATE, DELETE, MERGE
- Consultar dados
 - Categoria: DQL (Data Query Language)
 - Comando: SELECT

Principais funcionalidades do SQL

- Controlar permissões
 - Categoria: DCL (Data Control Language)
 - Comandos: GRANT e REVOKE
- Controlar transações
 - Categoria: TCL (Transaction Control Language)
 - Comandos: COMMIT, ROLLBACK, SAVEPOINT

Python DB-API

- Permite que diferentes SGBDs (como SQLite, MySQL, PostgreSQL e Oracle) sejam acessados de forma padronizada, usando a mesma estrutura de comandos em Python, mesmo que os sistemas de banco de dados por trás sejam diferentes.
- Documentação: <https://peps.python.org/pep-0249/>



Python Enhancement Proposals | Python » PEP Index » PEP 249

Contents

- Introduction
- Module Interface
 - Constructors
 - Globals
 - Exceptions
- Connection Objects
 - Connection methods
- Cursor Objects
 - Cursor attributes
 - Cursor methods
- Type Objects and Constructors
- Implementation Hints For Module Authors
- Optional DB API Extensions
- Optional Error Handling Extensions
- Optional Two-Phase Commit Extensions
 - TPC Transaction IDs
 - TPC Connection Methods

PEP 249 – Python Database API Specification v2.0

Author: Marc-André Lemburg <mal at lemburg.com>
Discussions-To: [Db-SIG list](#)
Status: Final
Type: Informational
Created: 12-Apr-1999
Post-History:
Replaces: [248](#)

► [Table of Contents](#)

[Introduction](#)

Conectando-se a um banco de dados

- Conectar ao banco de dados significa estabelecer uma comunicação entre um programa (como um script Python) e um sistema de gerenciamento de banco de dados (SGBD), como MySQL, PostgreSQL, SQL Server ou SQLite.
- Exemplo SQLite:

```
import sqlite3  
  
conn = sqlite3.connect("nome_do_banco")
```



<https://docs.python.org/3/library/sqlite3.html>

Conectando-se a um banco de dados

- Exemplo de conexão com MySQL e PostgreSQL:

```
import mysql.connector

conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='sua_senha',
    database='nome_do_banco'
)
```



<https://dev.mysql.com/doc/connector-python/en/>

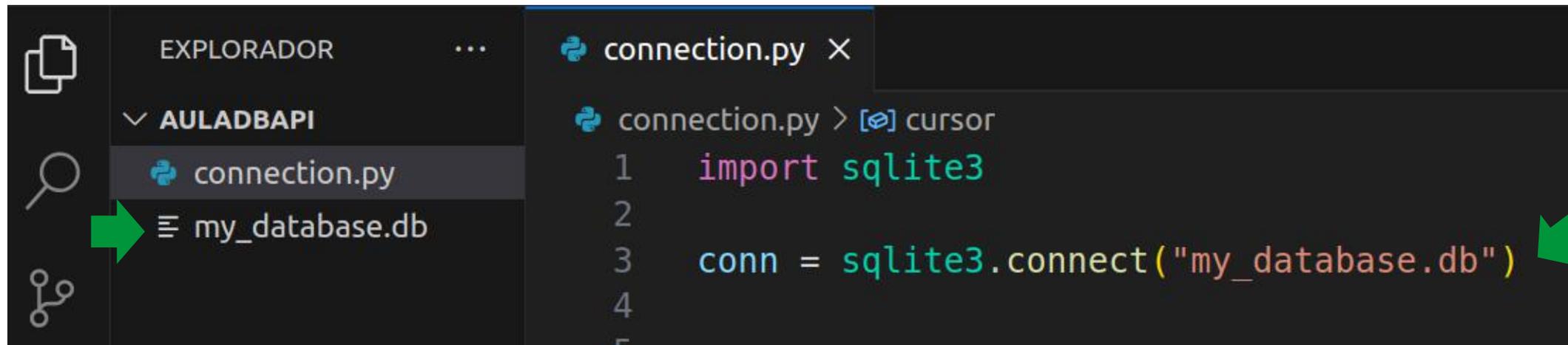
```
import psycopg2

conn = psycopg2.connect(
    host="localhost",
    port=5432,
    database="nome_do_banco",
    user="root",
    password="sua_senha"
)
```



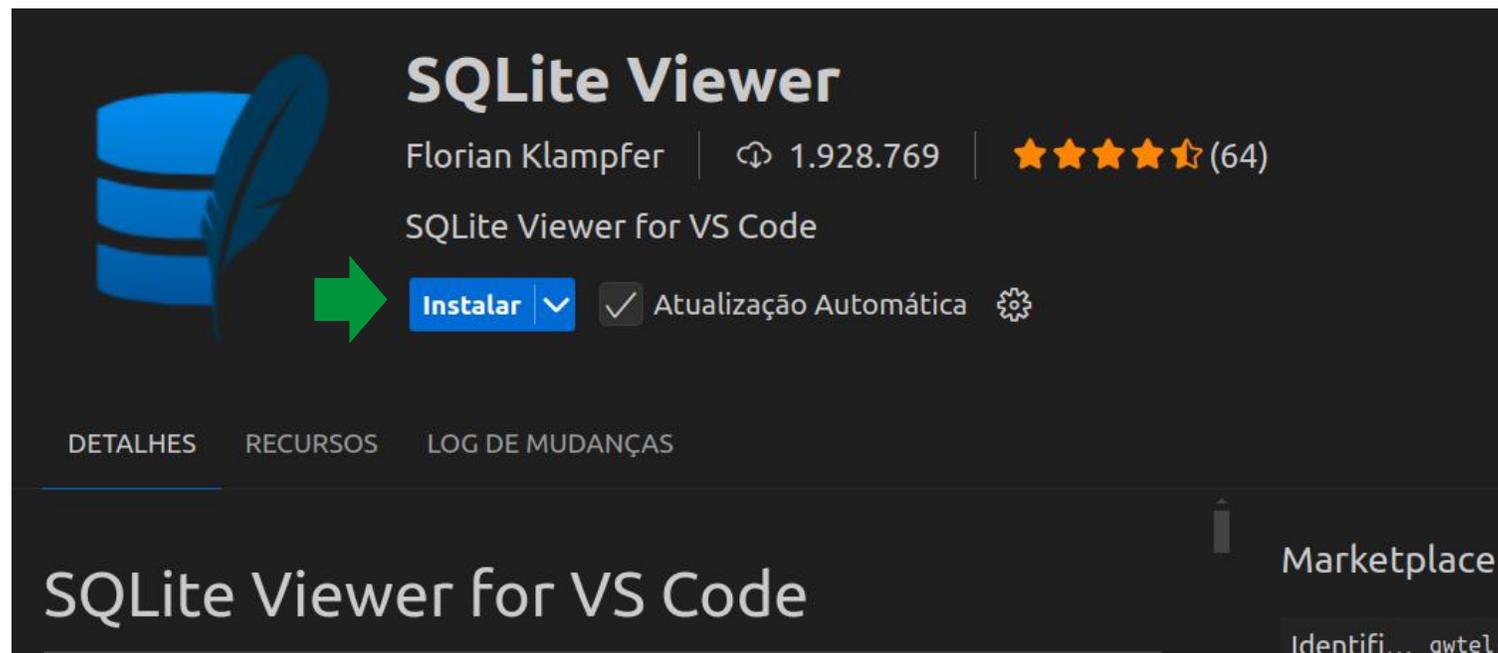
<https://www.psycopg.org/docs/>

Visualizando o banco no VSCode



```
connection.py x
connection.py > [?] cursor
1  import sqlite3
2
3  conn = sqlite3.connect("my_database.db")
4
5
```

- Extensão:



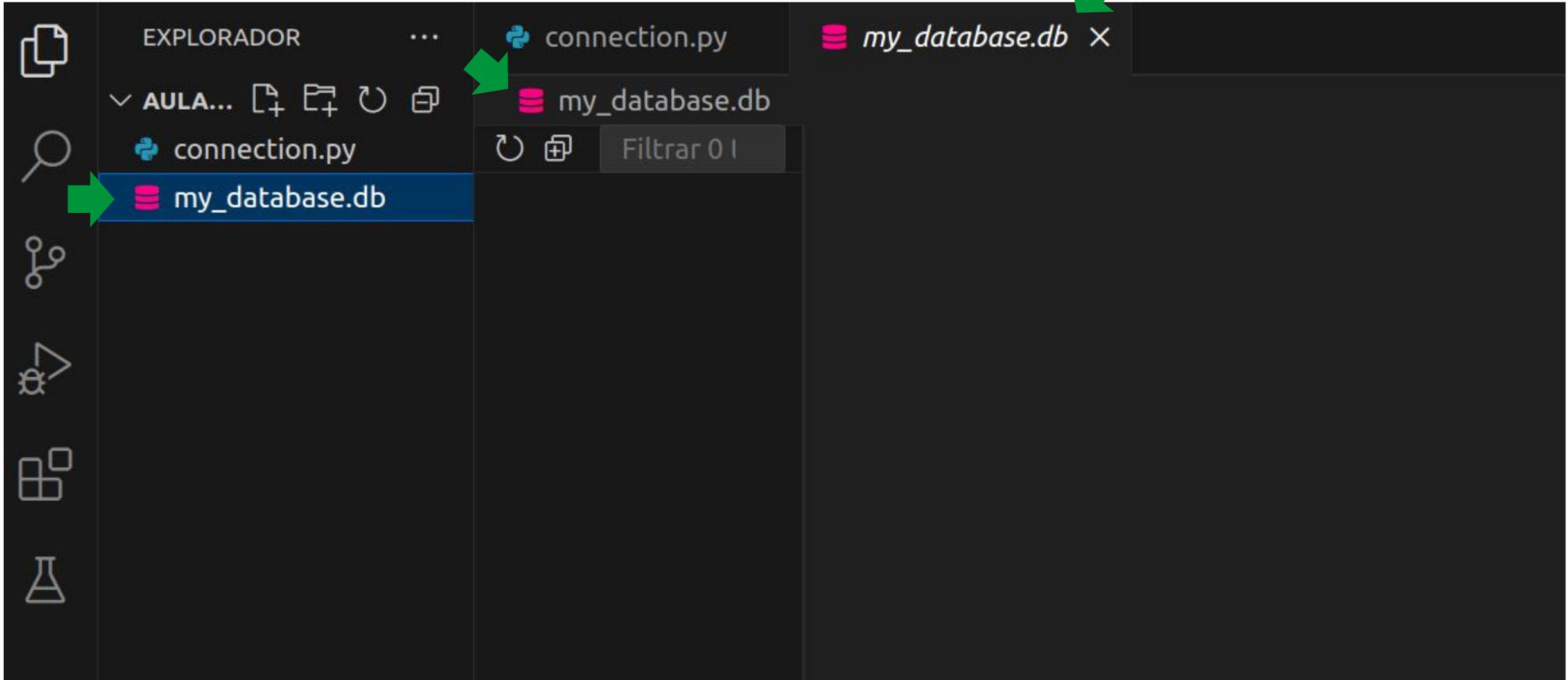
SQLite Viewer
Florian Klampfer | 1.928.769 | ★★★★★ (64)
SQLite Viewer for VS Code

[Instalar](#) Atualização Automática

DETALHES RECURSOS LOG DE MUDANÇAS

SQLite Viewer for VS Code Marketplace

Visualizando o banco no VSCode



Criando tabelas

- Para criar uma tabela usando a Python DB-API com SQLite, é necessário um cursor, que é o objeto responsável por executar comandos SQL dentro de uma conexão com o banco de dados.

```
import sqlite3

conn = sqlite3.connect("my_database.db")
cursor = conn.cursor()

cursor.execute("""
    CREATE TABLE clientes (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nome VARCHAR(100),
        cidade VARCHAR(100)
    );
""")
```

Liberando o cursor e fechando a conexão

- `cursor.close()` → libera os recursos associados ao cursor.
- `conn.close()` → fecha a conexão com o banco, liberando a memória e garantindo que nenhum acesso fique aberto indevidamente.

Deixar cursores ou conexões abertas pode causar vazamentos de memória, travamento de arquivos (no SQLite), ou até limite excedido de conexões (em bancos como PostgreSQL).

Modificado o script com boas práticas

```
import sqlite3

conn = sqlite3.connect("my_database.db")
cursor = conn.cursor()

cursor.execute("""
    CREATE TABLE IF NOT EXISTS clientes (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nome VARCHAR(100),
        cidade VARCHAR(100)
    );
""")

cursor.close()
conn.close()
```



Modificado o script com boas práticas

```
import sqlite3

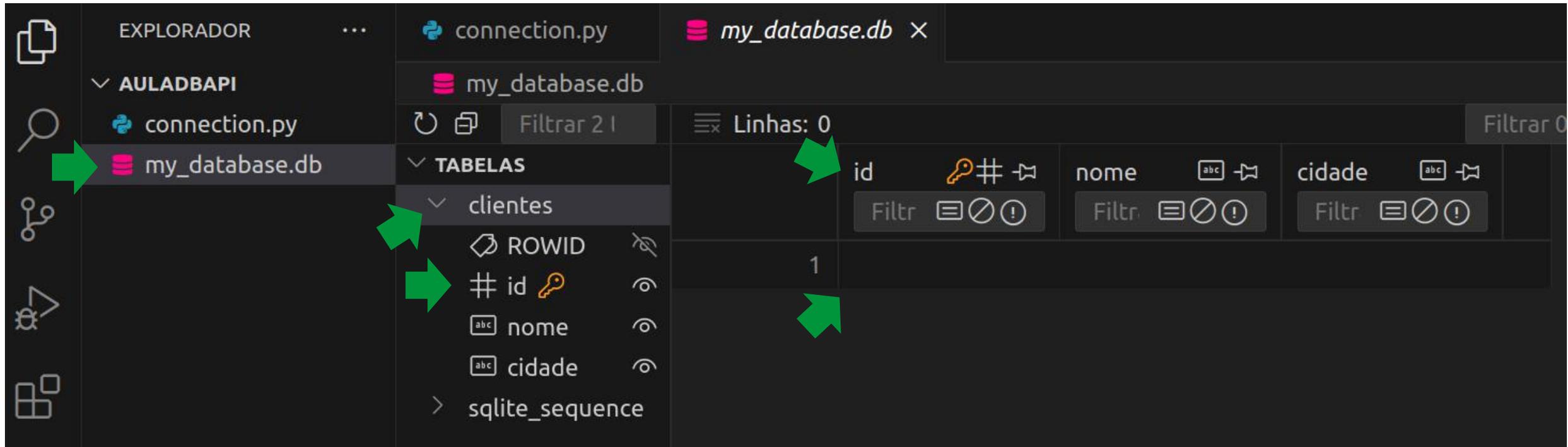
conn = sqlite3.connect("my_database.db")
cursor = conn.cursor()

cursor.execute("""
    CREATE TABLE IF NOT EXISTS clientes (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        nome VARCHAR(100),
        cidade VARCHAR(100)
    );
""")

cursor.close()
conn.close()
```

O IF NOT EXISTS é uma cláusula do SQL usada para evitar erro na criação de uma estrutura (como uma tabela) que já existe no banco de dados.

Visualizando a tabela no VSCode



The screenshot shows the VS Code interface with a database connection named 'my_database.db' open. The Explorer sidebar on the left shows the project structure, including 'AULADBAPI' and 'my_database.db'. The 'my_database.db' folder is expanded to show a table named 'clientes'. The table structure is displayed in the main editor area, showing columns: 'id' (primary key), 'nome', and 'cidade'. The table is currently empty, with 'Linhas: 0' displayed above the columns. A single row with the value '1' is visible in the 'id' column. Green arrows point to the 'my_database.db' folder in the Explorer, the 'clientes' table in the table list, the 'id' column header, and the value '1' in the table data.

id	nome	cidade
1		

Inserindo registros

- Em SQL, usamos o comando INSERT INTO para isso. No contexto do Python com sqlite3 (seguindo a Python DB API), essa inserção é feita com o cursor.
- Exemplo de comando SQL: **INSERT INTO clientes (nome, cidade) VALUES ('Lucas', 'Recife -PE')**

- Usando DB-API:

```
data = ("Lucas", "Recife - PE")
cursor.execute("""
    INSERT INTO clientes (nome, cidade)
    VALUES (?, ?)
""", data)
```

Inserindo registros

- Em SQL, usamos o comando INSERT INTO para isso. No contexto do Python com sqlite3 (seguindo a Python DB API), essa inserção é feita com o cursor.
- Exemplo de comando SQL: **INSERT INTO clientes (nome, cidade) VALUES ('Lucas', 'Recife -PE')**

- Usando DB-API:

```
data = ("Lucas", "Recife - PE")
cursor.execute("""
    INSERT INTO clientes (nome, cidade)
    VALUES (?, ?)
""", data)
```



Consultas parametrizadas →
forma segura de passar dados a
comandos SQL.

Inserindo registros

- Em SQL, usamos o comando INSERT INTO para isso. No contexto do Python com sqlite3 (seguindo a Python DB API), essa inserção é feita com o cursor.
- Exemplo de comando SQL: **INSERT INTO clientes (nome, cidade) VALUES ('Lucas', 'Recife -PE')**

- Usando DB-API:

```
data = ("Lucas", "Recife - PE")
cursor.execute("""
    INSERT INTO clientes (nome, cidade)
    VALUES (?, ?)
""", data)
```

Essa inserção ainda não foi gravada permanentemente no banco. Ela está em uma transação pendente. É necessário o comando `conn.commit()` para que o SQLite confirme a transação.

Inserindo registros

- Em SQL, usamos o comando INSERT INTO para isso. No contexto do Python com sqlite3 (seguindo a Python DB API), essa inserção é feita com o cursor.
- Exemplo de comando SQL: **INSERT INTO clientes (nome, cidade) VALUES ('Lucas', 'Recife -PE')**

- Usando DB-API:

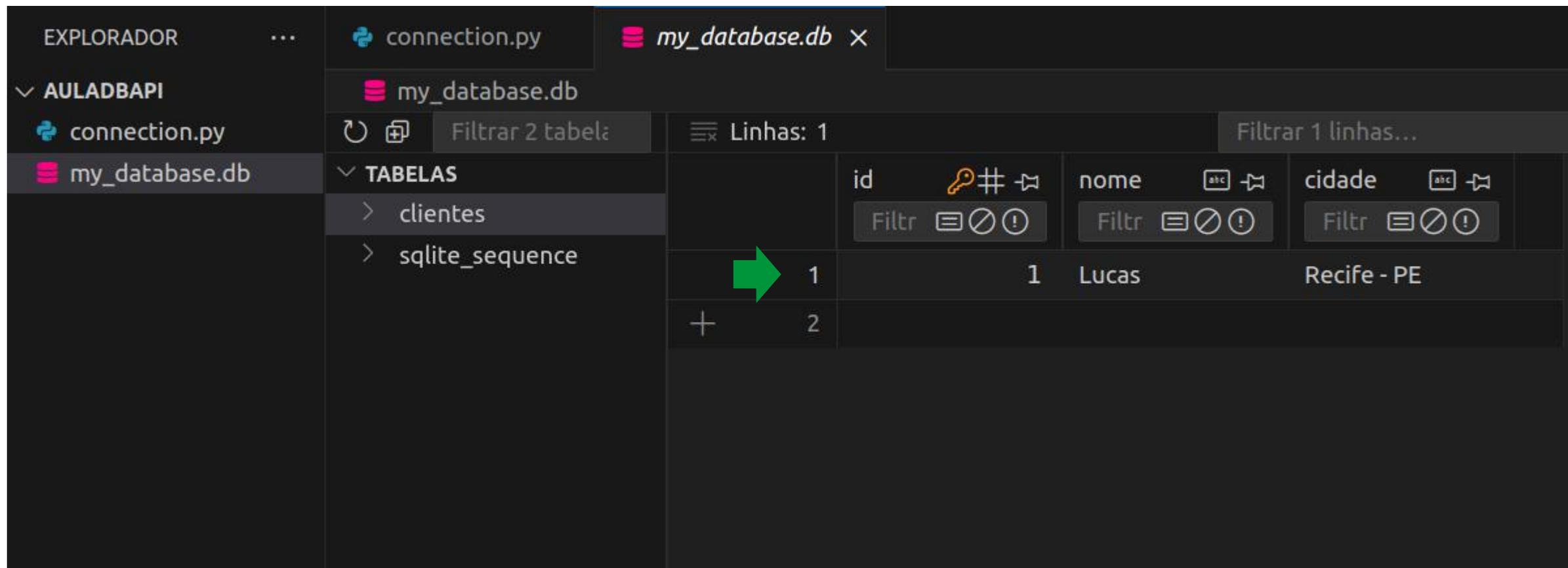
```
data = ("Lucas", "Recife - PE")
cursor.execute("""
    INSERT INTO clientes (nome, cidade)
    VALUES (?, ?)
""", data)
conn.commit()
```

Reflexão: Cada commit envolve operações de escrita em disco. Esse é um dos processos mais custosos em termos de tempo.

Confirmando transações

- O SQLite (e outros SGBDs) trabalha com o conceito de transação.
 - O comando `conn.commit()` confirma (salva) todas as alterações feitas na transação atual no banco de dados.
 - Enquanto você insere, atualiza ou deleta registros, essas alterações ficam em memória (não permanentes) até que você use `commit()`.
 - Ao chamar `commit()`, você finaliza a transação atual e grava definitivamente as mudanças no arquivo do banco.
- Sem o `commit()`, as alterações serão descartadas ao fechar a conexão.

Inserindo registros



The screenshot shows a database management tool interface. On the left, there is a sidebar with 'EXPLORADOR' and a tree view containing 'AULADBAPI', 'connection.py', and 'my_database.db'. The main area displays the 'my_database.db' database with a 'TABELAS' section containing 'clientes' and 'sqlite_sequence'. The 'clientes' table is selected, and its data is shown in a table view. The table has three columns: 'id', 'nome', and 'cidade'. The first row contains the values '1', 'Lucas', and 'Recife - PE'. A green arrow points to the 'id' column of the first row. The interface also shows a 'Linhas: 1' indicator and a 'Filtrar 1 linhas...' search bar.

	id	nome	cidade
1	1	Lucas	Recife - PE
+	2		

Atualizando registros

- O UPDATE é usado para alterar dados existentes em uma tabela do banco de dados.
- Ele modifica um ou mais campos de um ou mais registros, mantendo o mesmo id (ou chave primária).
- É importante ser específico ao usar o UPDATE para evitar alterar mais registros que o planejado.
- Exemplo de comando SQL: **UPDATE clientes SET nome = 'Lucas', cidade = 'Senhor do Bonfim - BA' WHERE id = 1;**

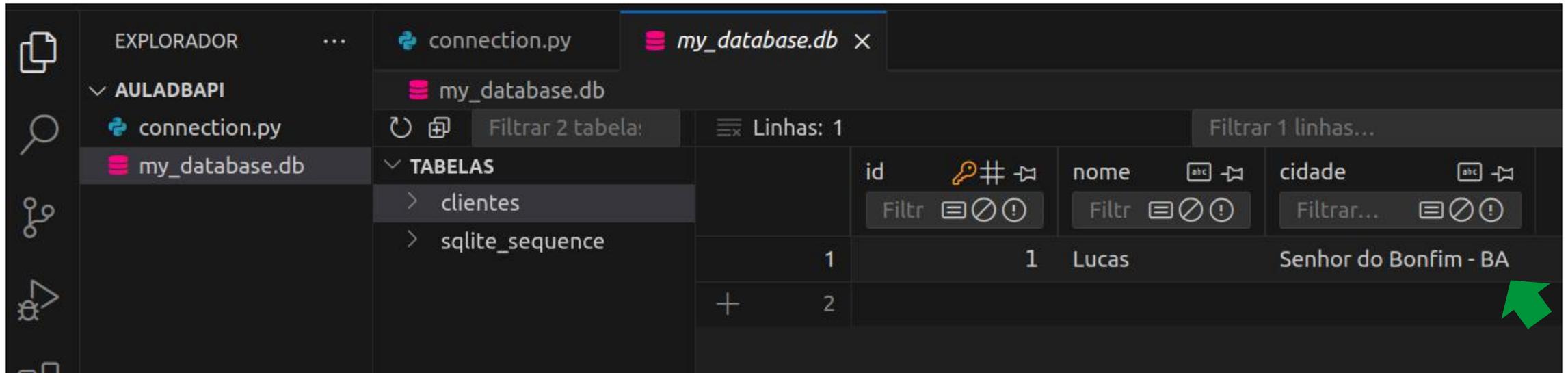
Atualizando registros

- Exemplo de comando SQL: **UPDATE clientes SET nome = 'Lucas', cidade = 'Senhor do Bonfim - BA' WHERE id = 1;**

- Usando DB-API:

```
data = ("Lucas", "Senhor do Bonfim - BA", 1)
cursor.execute("""
    UPDATE clientes
    SET nome = ?, cidade = ?
    WHERE id = ?
""", data)
```

Atualizando registros



The screenshot shows a database management tool interface. On the left, there is a sidebar with a tree view under 'AULADBAPI' containing 'connection.py' and 'my_database.db'. The main area displays the 'my_database.db' database with a table named 'clientes'. The table has columns 'id', 'nome', and 'cidade'. The first row contains the values '1', 'Lucas', and 'Senhor do Bonfim - BA'. A green arrow points to the 'Senhor do Bonfim - BA' cell in the 'cidade' column.

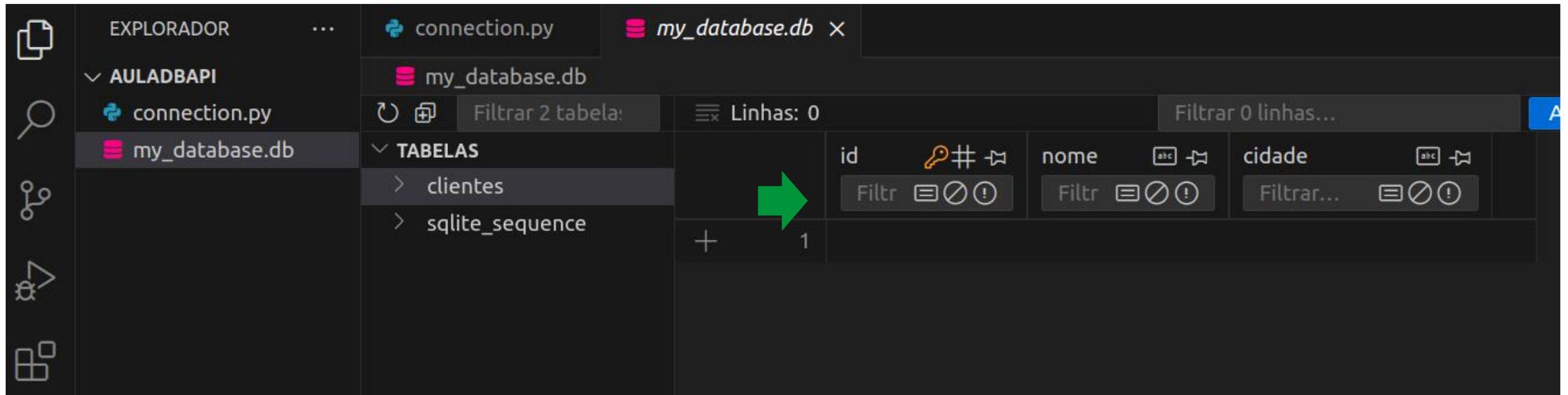
id	nome	cidade
1	Lucas	Senhor do Bonfim - BA
+	2	

Removendo registros

- O comando DELETE serve para remover linhas (registros) de uma tabela em um banco de dados. Ele não remove a estrutura da tabela, apenas dados específicos.
- Exemplo de comando SQL: **DELETE FROM clientes WHERE id = 1;**
- Usando DB-API:

```
data = (1,)  
cursor.execute("""  
    DELETE FROM clientes  
    WHERE id = ?  
""", data)
```

Removendo registros



The screenshot shows a database management tool interface. The left sidebar displays a tree view with the following structure:

- EXPLORADOR
 - AULADBAPI
 - connection.py
 - my_database.db

The main area shows the 'my_database.db' database selected. The 'TABELAS' (Tables) section is expanded, showing two tables: 'clientes' and 'sqlite_sequence'. The 'clientes' table is selected, and its data is displayed in a table view. The table has three columns: 'id', 'nome', and 'cidade'. The 'id' column is the primary key. The table contains one record with the value '1' in the 'id' column. A green arrow points to the '1' in the 'id' column.

id	nome	cidade
1		

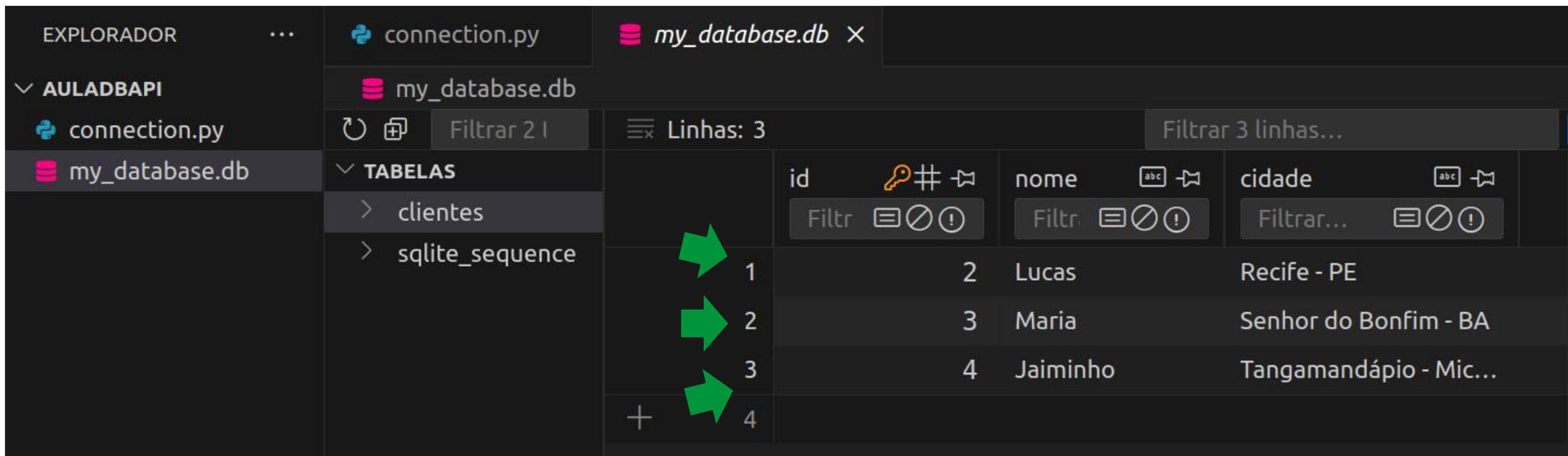
Inserir registros em lote

- Exemplo de comando SQL: `INSERT INTO clientes (nome, cidade) VALUES ('Lucas', 'Recife - PE'), ('Maria', 'Senhor do Bonfim - BA'), ('Jaiminho', 'Tangamandápio - Michoacán');`

- Usando DB-API:

```
data = [ ("Lucas", "Recife - PE"),  
         ("Maria", "Senhor do Bonfim - BA"),  
         ("Jaiminho", "Tangamandápio - Michoacán") ]  
cursor.executemany( """  
    INSERT INTO clientes (nome, cidade)  
    VALUES (?, ?)  
    """ , data)
```

Inserir registros em lote



EXPLORADOR ... connection.py my_database.db x

my_database.db

connection.py Filtrar 2 l Linhas: 3 Filtrar 3 linhas...

my_database.db

TABELAS

- clientes
- sqlite_sequence

	id	nome	cidade
1	2	Lucas	Recife - PE
2	3	Maria	Senhor do Bonfim - BA
3	4	Jaiminho	Tangamandápio - Mic...
+	4		

Consultas de um único resultado

- Uma consulta SQL é uma solicitação feita a um banco de dados para recuperar dados.
- Para realizar uma consulta SQL que retorna um único registro usando Python e a DB API, você pode utilizar o método `fetchone()`, que retorna a primeira linha dos resultados.
 - Para garantir que o resultado seja um único, o SQL precisa estar bem estruturado.
 - Se não houver nenhum resultado, ele retorna `None`.
- Exemplo de comando SQL: **`SELECT * FROM clientes WHERE id = 2;`**

Consultas de um único resultado

- Exemplo de comando SQL: `SELECT * FROM clientes WHERE id = 2;`
- Usando DB-API:

```
data = (2,)\n\ncursor.execute("""\n    SELECT * FROM clientes\n    WHERE id = ?\n    """, data)\n\ncliente = cursor.fetchone()\nprint(cliente)
```

Consultas de um único resultado

- Exemplo de comando SQL: **SELECT * FROM clientes WHERE id = 2;**
- Usando DB-API:

```
data = (2, )  
  
cursor.execute("""  
    SELECT * FROM clientes  
    WHERE id = ?  
""", data)  
  
cliente = cursor.fetchone()  
print(cliente)
```

A consultar pode retornar campos específicos.

A condição pode considerar outros campos.

Consultas de um único resultado

Linhas: 3 Filtrar 3 linhas...

	id   	nome  	cidade  
1	2	Lucas	Recife - PE
2	3	Maria	Senhor do Bonfim - BA
3	4	Jaiminho	Tangamandápio - Mic...
+ 4			

- Usando DB-API:

```
data = (2,)\n\ncursor.execute("""\n    SELECT * FROM clientes\n    WHERE id = ?\n    """, data)\n\ncliente = cursor.fetchone()\nprint(cliente)
```



```
(2, 'Lucas', 'Recife - PE')
```

Consultas de um único resultado

Linhas: 3 Filtrar 3 linhas...

	id	nome	cidade
1	2	Lucas	Recife - PE
2	3	Maria	Senhor do Bonfim - BA
3	4	Jaiminho	Tangamandápio - Mic...
+	4		

- Usando DB-API:

```
data = (2,)  
  
cursor.execute("""  
    SELECT nome FROM clientes  
    WHERE id = ?  
""", data)  
  
cliente = cursor.fetchone()  
print(cliente)
```



('Lucas',)

Consultas de um único resultado

Linhas: 3 Filtrar 3 linhas...

	id  # 	nome  	cidade  
1	2	Lucas	Recife - PE
2	3	Maria	Senhor do Bonfim - BA
3	4	Jaiminho	Tangamandápio - Mic...
+ 4			

- Usando DB-API:

```
data = (1,)  
  
cursor.execute("""  
    SELECT * FROM clientes  
    WHERE id = ?  
""", data)  
  
cliente = cursor.fetchone()  
print(cliente)
```



None

Consultas de múltiplos resultados

- O método `fetchall()` é utilizado após uma consulta SQL para recuperar todos os registros retornados.
- Ele retorna os dados como uma lista de tuplas, onde cada tupla representa uma linha do resultado.
- Exemplo de comando SQL: **`SELECT * FROM clientes;`**
- Usando DB-API:

```
cursor.execute("""
    SELECT * FROM clientes
""")

clientes = cursor.fetchall()
print(clientes)
```

Consultas de múltiplos resultados

Linhas: 3 Filtrar 3 linhas...

	id	nome	cidade
1	2	Lucas	Recife - PE
2	3	Maria	Senhor do Bonfim - BA
3	4	Jaiminho	Tangamandápio - Mic...
+	4		

- Usando DB-API:

```
cursor.execute("""
    SELECT * FROM clientes
""")

clientes = cursor.fetchall()
print(clientes)
```



```
[(2, 'Lucas', 'Recife - PE'), (3, 'Maria', 'Senhor do Bonfim - BA'), (4, 'Jaiminho', 'Tangamandápio - Michoacán')]
```

Consultas de múltiplos resultados

Linhas: 3 Filtrar 3 linhas...

	id	nome	cidade
1	2	Lucas	Recife - PE
2	3	Maria	Senhor do Bonfim - BA
3	4	Jaiminho	Tangamandápio - Mic...
+	4		

- Usando DB-API:

```
cursor.execute("""
    SELECT * FROM clientes
    ORDER BY nome
""")

clientes = cursor.fetchall()
print(clientes)
```



```
[(4, 'Jaiminho', 'Tangamandápio - Michoacán'), (2, 'Lucas', 'Recife - PE'), (3, 'Maria', 'Senhor do Bonfim - BA')]
```

Exercício

1. Crie uma Classe ClienteDB segundo o template ao lado: Esta classe encapsula todas as operações relacionadas ao banco de dados, criação do banco, criação da tabela, inserção, atualização, exclusão e consulta de clientes.
2. Teste os métodos criados.

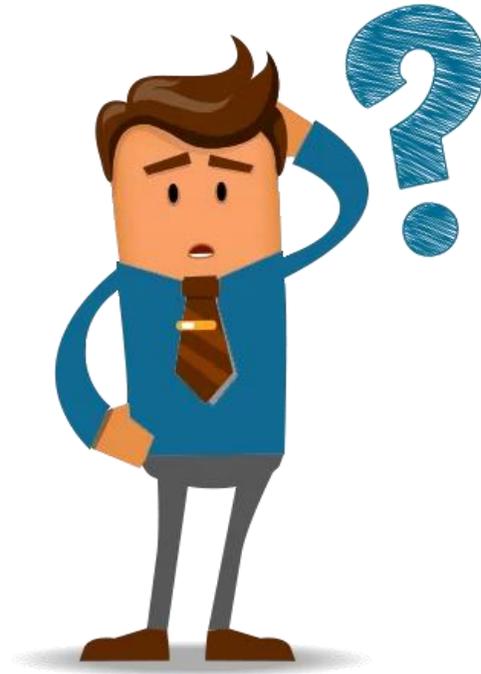
```
import sqlite3
```

```
class ClienteDB:  
    def __init__(self, db_name):  
        self.conn = sqlite3.connect(db_name)  
        self.cursor = self.conn.cursor()  
        self._criar_tabela()  
  
    def _criar_tabela(self):  
        pass  
  
    def inserir_cliente(self, nome, cidade):  
        pass  
  
    def atualizar_cliente(self, id, nome, cidade):  
        pass  
  
    def deletar_cliente(self, id):  
        pass  
  
    def consultar_cliente(self, id):  
        pass  
  
    def consultar_todos_clientes(self):  
        pass  
  
    def fechar_conexao(self):  
        pass
```

Próximos passos...

- Aprofundar SQL para melhor utilização da DB-API;
- Integração da DB-API com Frameworks: Entender como a DB-API é utilizada internamente para melhorar a eficiência e o controle sobre as consultas SQL.

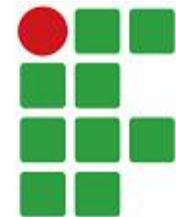
Dúvidas



PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática

Lucas Sampaio Leite



**INSTITUTO
FEDERAL**

Baiano