Aula 10

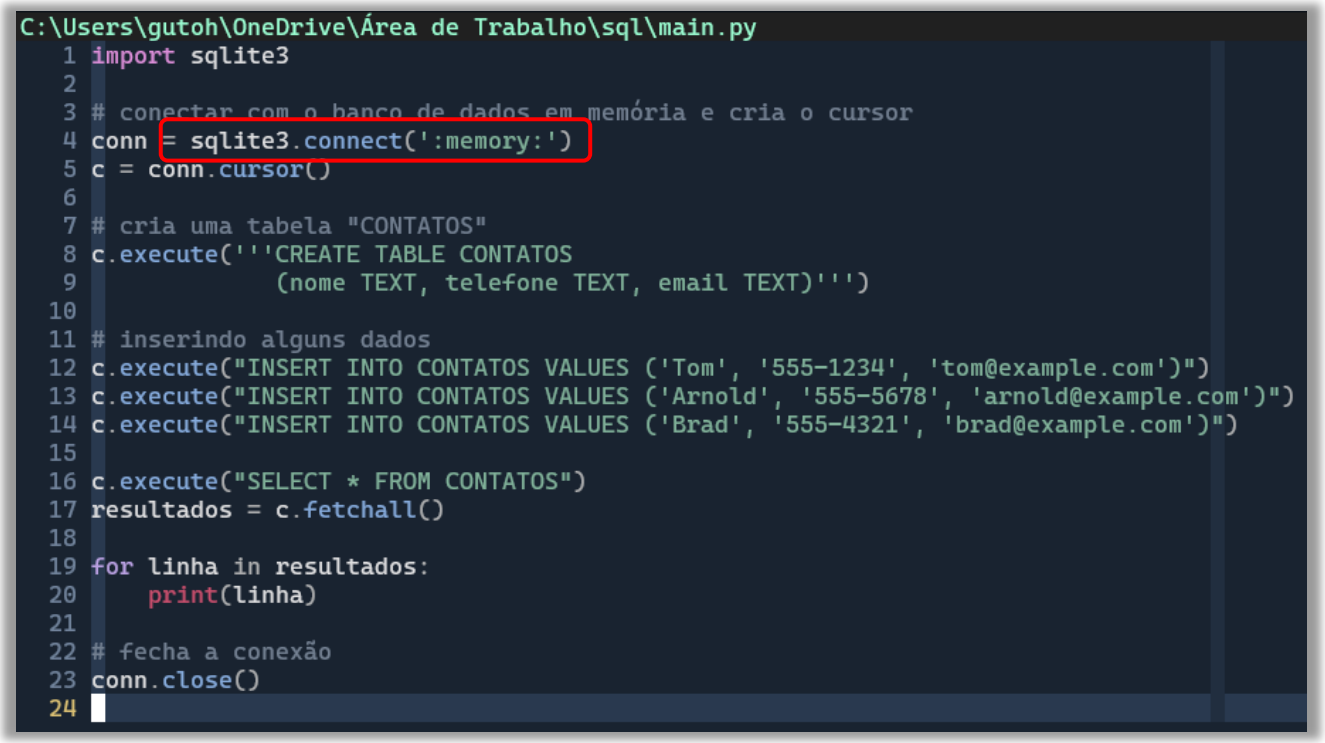
SQL

Banco em Memória

O banco que estávamos criando até o momento estava sendo salvo em disco, mas há outra opção de uso. Também podemos criar um banco de dados de forma temporária, existindo apenas na memória do computador.

Para isso, ao criarmos uma conexão com o banco, em vez de passarmos o nome do banco, passamos o comando ':memory'.

Veja o exemplo abaixo:



Na linha 4, temos a criação do banco em memória. Repare que, ao longo do código não é feito commit. Isso porque o commit aqui não tem sentido, já que as alterações ficarão restritas à memória RAM que, quando encerrado o programa, será apagado.

Usar um banco de dados em memória pode ser útil em situações em que você precisa armazenar temporariamente dados que não precisam ser preservados depois que o programa é encerrado. Por exemplo, se você precisar realizar cálculos complexos que envolvam grandes quantidades de dados, pode ser mais rápido e mais eficiente criar um banco de dados em memória para armazenar temporariamente os dados de entrada e saída.

Classes

As classes são uma ferramenta poderosas para trabalhar com o banco de dados. Podemos ter uma classe para gerenciar o banco de dados, outra para os contatos (usando como exemplo uma agenda).

Veja como fica um modelo abaixo:

Também podemos usar classes do Python para gerenciar melhor nosso código:

C:\Users\gutoh\OneDrive\Área de Trabalho\sql\main.py

```python
import sqlite3

class Agenda:
    """Classe para gerenciar a agenda e a conexão com o BD."""
    def __init__(self, nome_banco:str=':memory'):
        self.conn = sqlite3.connect(nome_banco)
        self.cur = self.conn.cursor()

    def fecha(self):
        self.conn.close()

    def commit(self):
        self.conn.commit()

    def executa(self, query:str, parametros=None):
        if parametros:
            self.cur.execute(query, parametros)
        else:
            self.cur.execute(query)

    def busca_todos(self, query, parametros=None):
        self.executa(query, parametros)
        return self.cur.fetchall()

    def busca_um(self, query, parametros=None):
        self.executa(query, parametros)
        return self.cur.fetchone()

    def cria_tabela(self):
        """Função para criar a tabela."""
        query = """
            CREATE TABLE IF NOT EXISTS AGENDA
                (id INTEGER PRIMARY KEY, nome TEXT,
                 endereco TEXT, telefone TEXT)"""
        self.executa(query)

```

C:\Users\gutoh\OneDrive\Área de Trabalho\sql\main.py

```python
class Contato:
    """Classe representando cada contato da agenda."""
    def __init__(self, nome:str, endereco:str, telefone:str):
        self.id:int = 0
        self.nome:str = nome
        self.endereco:str = endereco
        self.telefone:str = telefone

    def cria_registro(self, banco:Agenda):
        """função para salvar o contato atual na agenda"""
        query:str = "INSERT INTO AGENDA (nome, endereco, telefone) VALUES (?, ?, ?)"
        banco.executa(query, (self.nome, self.endereco, self.telefone))
        banco.commit()

    def __str__(self):
        return f'Nome: {self.nome}\nEndereço: {self.endereco}\nTelefone: {self.telefone}\n'

    @classmethod
    def busca_contato(cls, banco:Agenda):
        """função para buscar um contato na agenda e retornar ele como objeto"""
        print('Digite o id do contato a ser buscado :\n')
        id = input(' >> ')

        query = 'SELECT nome, endereco, telefone FROM AGENDA WHERE id = ?'
        resultado = banco.busca_um(query, (id,))

        if resultado:
            return Contato(resultado[0], resultado[1], resultado[2])

    @staticmethod
    def busca_todos(banco):
        query = "SELECT * FROM AGENDA"
        database.executa(query)
        resultados = database.busca_todos(query)
        contatos = []
        for item in resultados:
            contato = Contato(item[1], item[2], item[3])
            contatos.append(contato)
        return contatos

```

```
C:\Users\gutoh\OneDrive\Área de Trabalho\sql\main.py
78 if __name__ == '__main__':
79     database = Agenda('agenda.db')
80     database.cria_tabela()
81
82     contato_1 = Contato('Tom', 'POA', '123456789')
83     contato_1.cria_registro(database)
84
85     contato_2 = Contato('Arnold', 'Canoas', '987654321')
86     contato_2.cria_registro(database)
87
88     contato_buscado = Contato.busca_contato(database)
89     print(contato_buscado)
90
91     contatos = Contato.busca_todos(database)
92     for contato in contatos:
93         print(contato)
94
95     database.fecha()
96
```

No código acima, temos uma classe chamada Agenda que é responsável por receber os códigos SQL (queries) e então executar no banco de dados. Se nenhum nome for especificado para a criação da agenda, ela será criada e armazenada apenas na memória RAM da máquina.
Ela também tem uma função responsável por criar a tabela AGENDA, caso ela ainda não exista.

A classe Contato tem os métodos com as queries responsáveis por buscar e inserir os contatos na tabela. Ela tem dois métodos especiais. Um método de classe para buscar um contato com base no seu id e um método estático para buscar todos os contatos da agenda e retornar uma lista de objetos do tipo Contato.

Exercícios para Praticar

1. Crie uma classe chamada "Cliente" com atributos de nome, idade e telefone.
    a. Crie uma tabela chamada "Clientes" com colunas correspondentes aos atributos da classe "Cliente" usando SQLite3.
    b. Crie um método na classe "Cliente" que insira um novo cliente na tabela "Clientes" usando SQLite3.
    c. Crie um método na classe "Cliente" que atualize os dados de um cliente na tabela "Clientes" usando SQLite3.
    d. Crie um método na classe "Cliente" que exclua um cliente da tabela "Clientes" usando SQLite3.
    e. Crie um método na classe "Cliente" que retorne todos os clientes da tabela "Clientes" usando SQLite3.
2. Crie uma classe chamada "Produto" com atributos de nome, preço e quantidade.
    a. Crie uma tabela chamada "Produtos" com colunas correspondentes aos atributos da classe "Produto" usando SQLite3.
    b. Crie um método na classe "Produto" que insira um novo produto na tabela "Produtos" usando SQLite3.
    c. Crie um método na classe "Produto" que atualize os dados de um produto na tabela "Produtos" usando SQLite3.
    d. Crie um método na classe "Produto" que exclua um produto da tabela "Produtos" usando SQLite3.
    e. Crie um método na classe "Produto" que retorne todos os produtos da tabela "Produtos" usando SQLite3.
3. Crie uma classe chamada "Pedido" com atributos de número, cliente e valor.
    a. Crie uma tabela chamada "Pedidos" com colunas correspondentes aos atributos da classe "Pedido" usando SQLite3.

    b. Crie um método na classe "Pedido" que insira um novo pedido na tabela "Pedidos" usando SQLite3.
    c. Crie um método na classe "Pedido" que atualize os dados de um pedido na tabela "Pedidos" usando SQLite3.
    d. Crie um método na classe "Pedido" que exclua um pedido da tabela "Pedidos" usando SQLite3.
    e. Crie um método na classe "Pedido" que retorne todos os pedidos da tabela "Pedidos" usando SQLite3.
4. Crie uma classe chamada "Funcionario" com atributos de nome, cargo e salario.
    a. Crie uma tabela chamada "Funcionarios" com colunas correspondentes aos atributos da classe "Funcionario" usando SQLite3.
    b. Crie uma classe chamada "Livro" com atributos de título, autor e ano de publicação.
    c. Crie uma tabela chamada "Livros" com colunas correspondentes aos atributos da classe "Livro" usando SQLite3.
    d. Crie um método na classe "Livro" que insira um novo livro na tabela "Livros" usando SQLite3.
    e. Crie um método na classe "Livro" que atualize os dados de um livro na tabela "Livros" usando SQLite3.
    f. Crie um método na classe "Livro" que exclua um livro da tabela "Livros" usando SQLite3.
    g. Crie um método na classe "Livro" que retorne todos os livros da tabela "Livros" usando SQLite3.
5. Crie uma classe chamada "Estudante" com atributos de nome, idade e curso.
    a. Crie uma tabela chamada "Estudantes" com colunas correspondentes aos atributos da classe "Estudante" usando SQLite3.
    b. Crie um método na classe "Estudante" que insira um novo estudante na tabela "Estudantes" usando SQLite3.
    c. Crie um método na classe "Estudante" que atualize os dados de um estudante na tabela "Estudantes" usando SQLite3.
    d. Crie um método na classe "Estudante" que exclua um estudante da tabela "Estudantes" usando SQLite3.
    e. Crie um método na classe "Estudante" que retorne todos os estudantes da tabela "Estudantes" usando SQLite3.
6. Crie uma classe chamada "Veiculo" com atributos de marca, modelo e ano.
    a. Crie uma tabela chamada "Veiculos" com colunas correspondentes aos atributos da classe "Veiculo" usando SQLite3.
    b. Crie um método na classe "Veiculo" que insira um novo veículo na tabela "Veiculos" usando SQLite3.
    c. Crie um método na classe "Veiculo" que atualize os dados de um veículo na tabela "Veiculos" usando SQLite3.
    d. Crie um método na classe "Veiculo" que exclua um veículo da tabela "Veiculos" usando SQLite3.
    e. Crie um método na classe "Veiculo" que retorne todos os veículos da tabela "Veiculos" usando SQLite3.