LINGUAGEM DE PROGRAMAÇÃO APLICADA

AULA 5

Prof. Vinicius Borin

CONVERSA INICIAL

Ao longo desta etapa de estudos, você irá aprender:

- Banco de dados – SQLite3;
- JavaScript Object Notation (JSON);
- Continuando com Design pattern.

TEMA 1 – INTRODUÇÃO AO SQLITE3

O SQLite é um sistema de gerenciamento de banco de dados relacional que é conhecido por sua leveza, facilidade de uso e eficiência. Aqui estão alguns pontos importantes sobre o SQLite 3:

- Zero configuração: uma de suas principais vantagens do SQLite é não requerer configuração ou administração de servidor. Você simplesmente inclui uma biblioteca em seu projeto e está pronto para começar a usar o banco de dados.
- Embarcado: o SQLite é uma biblioteca C que é embutida diretamente em aplicativos, o que significa que todo o banco de dados é armazenado em um único arquivo. Isso torna o SQLite perfeito para aplicativos que precisam de um banco de dados leve e não querem a sobrecarga de um servidor de banco de dados separado.
- Ampla disponibilidade: o SQLite é amplamente suportado e está disponível em praticamente todas as plataformas, incluindo Windows, macOS, Linux e sistemas embarcados. Ele é usado em uma variedade de aplicativos, desde navegadores da web até aplicativos móveis e até mesmo em sistemas embarcados.
- Suporte ao SQL padrão: o SQLite suporta a maioria do SQL padrão, incluindo consultas complexas, junções, transações, gatilhos e subconsultas. Isso o torna muito poderoso e flexível para uma ampla gama de aplicativos.
- Transações ACID: o SQLite oferece suporte a transações de Atomicidade, Consistência, Isolamento, Durabilidade (ACID), o que garante que as

operações de banco de dados sejam confiáveis, mesmo em face de falhas do sistema.

- Extensibilidade: o SQLite permite a criação de funções personalizadas e extensões em C, o que significa que você pode estender suas capacidades para atender às necessidades específicas do seu aplicativo.
- Bom desempenho: embora seja leve, o SQLite é conhecido por oferecer bom desempenho para a maioria dos casos de uso. Ele é otimizado para leitura e gravação concorrentes, tornando-o adequado para aplicativos com requisitos de desempenho moderados.
- Livre e de Código aberto: o SQLite é de código aberto e é distribuído sob os termos da Licença Pública Geral do SQLite, que é uma licença de software livre.

No entanto, é importante notar que o SQLite pode não ser a melhor opção para todos os casos de uso. Por exemplo, ele pode não ser ideal para aplicativos que exigem alta concorrência e escalabilidade, ou para aplicativos que necessitam de recursos avançados de segurança e autenticação. Nesses casos, sistemas de gerenciamento de banco de dados mais robustos como MySQL, PostgreSQL ou Oracle podem ser mais apropriados.

Devido à sua natureza leve e incorporada, o SQLite é amplamente utilizado em aplicativos móveis, tanto para iOS quanto para Android. Ele é usado para armazenar dados localmente no dispositivo, como configurações do aplicativo, dados do usuário, cache e até mesmo para aplicativos que exigem armazenamento de dados offline.

TEMA 2 – PROGRAMANDO COM SQLITE3

Vamos, agora, aprender a criar um código em que nos conectamos a um banco de dados com SQLite3 de uma base de animais.

2.1 Conectando, criando uma tabela e inserindo registros

Aqui está um breve resumo do código:

- Conectando ao banco de dados: essa linha cria uma conexão com o banco de dados SQLite. Se o banco de dados 'animal.db' não existir, ele será criado nesse momento.
- Criando um cursor: um cursor é criado para executar comandos SQL na conexão com o banco de dados.
- Criando a tabela: esse comando SQL cria uma tabela chamada 'dog' com colunas para 'id' (chave primária autoincremental), 'name', 'age', 'weight', 'height' e 'breed'. O IF NOT EXISTS garante que a tabela só será criada se ainda não existir.
- Exemplos de dados para inserir na tabela: uma lista chamada pets_data contém tuplas de dados para serem inseridos na tabela 'dog'.
- Inserindo os dados na tabela: um loop for é usado para inserir os dados de cada animal na tabela 'dog' usando a instrução SQL INSERT INTO. O método cursor.execute() é usado para executar a consulta SQL.
- Commit das alterações: as alterações feitas no banco de dados são confirmadas com conn.commit(). Isso garante que as alterações sejam permanentemente gravadas no banco de dados.
- Fechando a conexão com o banco de dados: a conexão com o banco de dados é fechada usando conn.close() para liberar recursos.
- Imprimindo uma mensagem de sucesso: uma mensagem é impressa para indicar que os registros foram inseridos com sucesso.

```
# Conectando ao banco de dados (cria um novo se não existir)
conn = sqlite3.connect('animal.db')

# Criando um cursor para executar comandos SQL
cursor = conn.cursor()

# Criando a tabela
cursor.execute('''CREATE TABLE IF NOT EXISTS dog (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT,
                    age INTEGER,
```

```
                    weight REAL,
                    height REAL,
                    breed TEXT
                )''')

# Exemplos de dados para inserir na tabela
pets_data = [
    ('Luke', 9, 45, 0.80, 'Pastor Alemao'),
    ('Buddy', 2, 7.2, 0.3, 'Golden Retriever'),
    ('Whiskers', 1, 1.2, 0.15, 'Siamese'),
    ('Max', 4, 5.0, 0.4, 'Labrador'),
]

# Inserindo os dados na tabela
for pet in pets_data:
    cursor.execute("INSERT INTO dog "
                   "(name, age, weight, height, breed) "
                   "VALUES (?, ?, ?, ?, ?)",
                   pet)

# Commit das alterações
conn.commit()

# Fechando a conexão com o banco de dados
conn.close()

print("Registros inseridos com sucesso.")
```

2.2 Listando os dados da tabela

Esse código Python demonstra como recuperar e exibir registros de uma tabela em um banco de dados SQLite. Aqui está uma breve explicação de cada parte:

- Conectando ao banco de dados: essa linha cria uma conexão com o banco de dados SQLite chamado 'animal.db'.
- Criando um cursor: um cursor é criado para executar comandos SQL na conexão com o banco de dados.
- Selecionando todos os registros da tabela: o método execute() é usado para executar a consulta SQL "SELECT * FROM dog", que seleciona

todos os registros da tabela 'dog'. Os resultados da consulta são armazenados em rows usando o método fetchall().

- Mostrando os registros: um loop for é usado para iterar sobre os resultados (rows) e imprimir os valores de cada registro, que incluem ID, nome, idade, peso, altura e raça do animal.
- Fechando a conexão com o banco de dados: a conexão com o banco de dados é fechada usando conn.close() para liberar recursos.

```
# Conectando ao banco de dados
conn = sqlite3.connect('animal.db')

# Criando um cursor para executar comandos SQL
cursor = conn.cursor()

# Selecionando todos os registros da tabela
cursor.execute("SELECT * FROM dog")
rows = cursor.fetchall()

# Mostrando os registros
for row in rows:
    print(f"ID: {row[0]}, "
          f"Name: {row[1]}, "
          f"Age: {row[2]}, "
          f"Weight: {row[3]}, "
          f"Height: {row[4]}, "
          f"Breed: {row[5]}")

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

TEMA 3 – JAVASCRIPT OBJECT NOTATION (JSON)

O JavaScript Object Notation (JSON) é um formato de dados leve e de fácil leitura que é usado para troca de dados entre um servidor e um cliente, e também para armazenamento de dados. Ele é baseado na sintaxe de objetos literais em JavaScript, mas é independente de linguagem, o que significa que pode ser usado com praticamente qualquer linguagem de programação, e com o Python não é diferente.

As características principais do JSON incluem:

- Simplicidade: o JSON é fácil de ler e escrever para humanos, tornando-o ideal para troca de dados entre sistemas.
- Estrutura de Dados: o JSON suporta tipos de dados como strings, números, booleanos, arrays e objetos, permitindo a representação de estruturas de dados complexas.
- Leveza: o JSON é leve em termos de espaço de armazenamento e largura de banda necessária para transmitir os dados.
- Independência de Linguagem: como mencionado anteriormente, o JSON é independente de linguagem, o que significa que pode ser facilmente interpretado e manipulado em diferentes plataformas e linguagens de programação.
- Suporte Universal: JSON é amplamente suportado por muitas linguagens de programação, tornando-o uma escolha popular para troca de dados em aplicativos e serviços web.

Um exemplo simples de JSON seria:

```
{
  "nome": "João",
  "idade": 30,
  "cidade": "São Paulo",
  "casado": false,
  "hobbies": ["futebol", "cinema", "cozinhar"]
}
```

3.1 Criando e manipulando um JSON

Vejamos a descrição do código seguir:

- Definimos um dicionário chamado *data* que contém informações sobre um animal.
- Definimos o caminho para o arquivo, no qual os dados serão armazenados em formato JSON.
- Usamos a função json.dump() para escrever os dados do dicionário no arquivo JSON especificado, com uma formatação indentada de quatro espaços.

- Imprimimos uma mensagem indicando que os dados foram salvos em JSON.
- Usamos a função json.load() para ler os dados do arquivo JSON especificado.
- Imprimimos uma mensagem indicando que os dados foram carregados do JSON.
- Imprimimos os dados carregados do arquivo JSON.

```
import json

# Dados a serem salvos em um arquivo JSON
data = {
    "name": "Luke",
    "age": 9,
    "weight": 45,
    "height": 0.8,
    "breed": "pastor alemão",
}

# Caminho para o arquivo JSON
file_path = "data.json"

# Salvar dados em formato JSON no arquivo
with open(file_path, "w") as json_file:
    json.dump(data, json_file, indent=4)

print("Dados salvos em JSON.")

# Carregar dados de um arquivo JSON
with open(file_path, "r") as json_file:
    loaded_data = json.load(json_file)

print("Dados carregados do JSON:")
print(loaded_data)
```

3.2 Classes em JSON

Vejamos a descrição do código a seguir:

```
import json

class Dog:
    def __init__(self, name, age, weight, height, breed):
        self.name = name
        self.age = age
        self.weight = weight
        self.height = height
        self.breed = breed

# Criando um objeto da classe Dog
dog = Dog("Luke", 9, 45, 0.8, "Pastor Alemão")

# Caminho para o arquivo JSON
file_path = "dog.json"
# Convertendo o objeto para um dicionário
dog_dict = {
    "name": dog.name,
    "age": dog.age,
    "weight": dog.weight,
    "height": dog.height,
    "breed": dog.breed
}
# Salvando o dicionário em formato JSON no arquivo
with open(file_path, "w") as json_file:
    json.dump(dog_dict, json_file, indent=4)

print("Objeto da classe Dog salvo em formato JSON.")
print("Dados salvos em JSON.")

# Carregar dados de um arquivo JSON
with open(file_path, "r") as json_file:
    loaded_data = json.load(json_file)

print("Dados carregados do JSON:")
print(loaded_data)
```

TEMA 4 – PADRÃO DE DESIGN DE ESTRUTURAL: PROXY

Imagine que você está em um consultório médico e precisa marcar uma consulta com o médico. Ao invés de ligar diretamente para ele, você fala com a sua secretária.

A secretária é como um proxy nesse cenário. Ela é responsável por aceitar suas solicitações de consulta, verificar a disponibilidade do médico, agendar a consulta no horário certo e confirmar com você.

Ela gerencia todo o processo de agendamento de consulta, garantindo que tudo seja feito corretamente e no tempo certo, sem que você precise lidar diretamente com o médico. Isso torna o processo mais organizado e eficiente para todos os envolvidos.

```
import sqlite3

# Classe Dog
class Dog:
    def __init__(self, name, age, weight, height, breed):
        self.name = name
        self.age = age
        self.weight = weight
        self.height = height
        self.breed = breed

# Proxy para interagir com o banco de dados SQLite3
class DbProxy:
    def __init__(self, database_path='dogs.db'):
        self.database_path = database_path

    def insert_dog(self, new_dog):
        conn = sqlite3.connect(self.database_path)
        cursor = conn.cursor()

        cursor.execute('''CREATE TABLE IF NOT EXISTS dogs (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            name TEXT,
                            age INTEGER,
                            weight REAL,
                            height REAL,
```

```
                            breed TEXT
                        )''')

        cursor.execute("INSERT INTO dogs "
                       "(name, age, weight, height, breed) "
                       "VALUES (?, ?, ?, ?, ?)",
                       (new_dog.name,
                        new_dog.age,
                        new_dog.weight,
                        new_dog.height,
                        new_dog.breed))

        conn.commit()
        conn.close()
        print("Dog inserted successfully.")

    def get_all_dogs(self):
        conn = sqlite3.connect(self.database_path)
        cursor = conn.cursor()

        cursor.execute("SELECT * FROM dogs")
        rows = cursor.fetchall()

        dogs = []
        for row in rows:
            dogs.append(Dog(row[1], row[2], row[3], row[4], row[5]))

        conn.close()
        return dogs


# Criando um objeto Dog
dog = Dog("Luke", 10, 45, 0.8, "Pastor Alemão")

# Usando o Proxy para inserir o objeto no banco de dados
proxy = DbProxy()
proxy.insert_dog(dog)

# Usando o Proxy para buscar todos os cachorros no banco de dados
all_dogs = proxy.get_all_dogs()
for d in all_dogs:
```

```
    print(f"Name: {d.name}, "
          f"Age: {d.age}, "
          f"Weight: {d.weight}, "
          f"Height: {d.height}, "
          f"Breed: {d.breed}")
```

TEMA 5 – PADRÃO DE DESIGN ESTRUTURAL: ADAPTER

Imagine que você tem um fone de ouvido com fio que apresenta um conector de áudio P2 (aquele padrão comum de 3,5 mm), mas o seu computador ou smartphone só tem portas USB. Como você pode usar seu fone de ouvido com o dispositivo USB? É aí que entra o adaptador!

O adaptador é como um pequeno dispositivo que transforma o conector de áudio P2 do seu fone de ouvido em algo que pode se comunicar com a porta USB do seu computador ou smartphone. Ele faz com que o fone de ouvido com fio e o dispositivo USB se entendam e funcionem juntos, mesmo que tenham sido feitos com padrões diferentes.

Em programação, é bastante comum usarmos um adapter para converter, por exemplo, um padrão de armazenamento de arquivos para outro, como converter um arquivo *Extensible Markup Language* (XML) em um JSON, ou então um JSON para um SQLite, ou vice-versa. Vejamos o exemplo:

```
import sqlite3
import json


class JsonSqliteAdapter:
    def __init__(self, db_name):
        self.db_name = db_name

    def connect(self):
        self.conn = sqlite3.connect(self.db_name)
        self.conn.execute("CREATE TABLE IF NOT EXISTS data "
                          "(id INTEGER PRIMARY KEY, json_data TEXT)")

    def save_data(self, data):
        json_data = json.dumps(data)
        self.conn.execute("INSERT INTO data (json_data) "
                          "VALUES (?)",
```

```
                            (json_data,))
        self.conn.commit()

    def load_data(self):
        cursor = self.conn.execute("SELECT json_data FROM data")
        json_data_list = [row[0] for row in cursor.fetchall()]
        data_list = [json.loads(json_data) for json_data in
json_data_list]
        return data_list

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


# Exemplo de uso
if __name__ == "__main__":
    adapter = JsonSqliteAdapter("data.db")
    adapter.connect()

    data_to_save = [{"name": "Alice", "age": 25}, {"name": "Bob",
"age": 30}]
    adapter.save_data(data_to_save)

    loaded_data = adapter.load_data()
    for item in loaded_data:
        print(item)

    adapter.close()
```

FINALIZANDO

Ao longo desta etapa de estudos, você aprendeu:

- Banco de dados – SQLite3;
- JavaScript Object Notation (JSON);
- Design pattern: proxy;
- Design pattern: adapter.

REFERÊNCIAS

PUGA, S.; RISSETI, G. **Lógica de programação e estrutura de Dados**. 3. ed. São Paulo: Pearson, 2016.

ROSEWOOD, E. **Programação orientada a objetos em Python**: dos fundamentos às técnicas avançadas.

ZHART, D. **Design patter**. Refactoring Guru. Disponível em: <https://refactoring.guru/design-patterns>. Acesso em: 17 mar. 2024.