EDC - Graduação em Engenharia de Computação
Projeto de Bloco: Fundamentos de Dados

TP5 - Web Scraping com Python

Professor: Alcione Santos Dolavale.
Aluna: Letícia Ferreira Augusto Alves Gomes.

Instituto Infnet.
Brasil, 10 de março de 2025.
Professor: Alcione Santos Dolavale.
Aluna: Letícia Gomes.
Projeto Bloco: Fundamentos Dados.

Sumário

Introdução

◆Breve Descrição

Este projeto foi feito a partir das ferramentas: **SQLite3** e **Python**, com dados de eventos, contendo as tabelas:

- **tb_evento:** 6 dados de eventos;
- **tb_dado_evento:** 6 dados de eventos;
- **tb_metadado:** 6 metadados de eventos;

◆Localização dos Dados

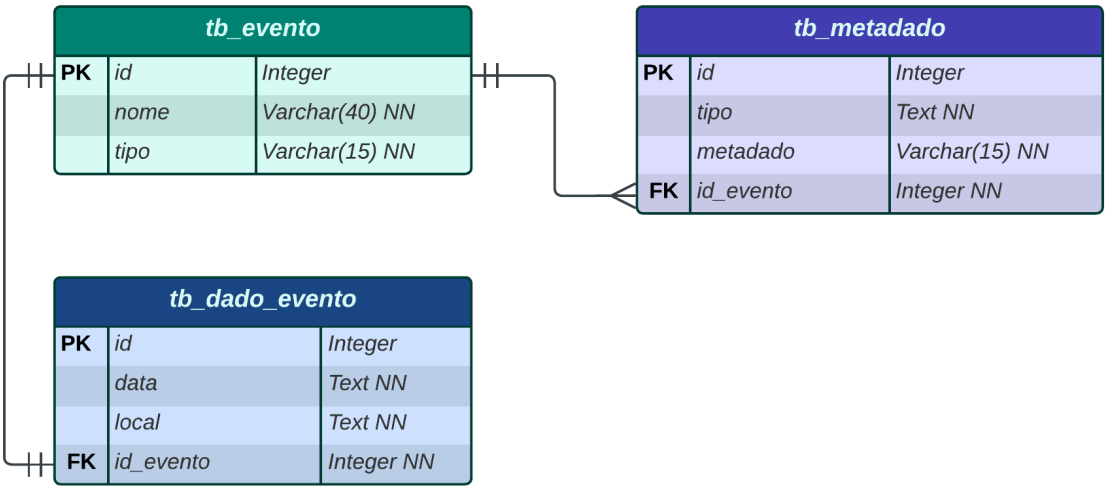
Os dados criados e consultas foram separados entre os arquivos: main.py, conexão_db.py, criacao_tabelas.py, web_scraping.py, crud.py e modelos.py

- **Sites utilizados para web scraping:**

  https://www.metallica.com/tour/2025-04-26-toronto-ontario-canada.html
  https://www.linkinpark.com/tour
  https://www.ironmaiden.com/tour/run-for-your-lives-world-tour/
  https://www.sampaingressos.com.br/afonso+padilha+stand+up+comedy+no+marte+hall+sp+marte+hall
  https://www.sampaingressos.com.br/mateus+solano+em+o+figurante+teatro+renaissance
  https://bit.ly/54o-festival-de-inverno-de-campos-do-jordao

◆Estrutura Diagrama ER

Diagrama **Entidade-Relacionamento** dos dados utilizados.



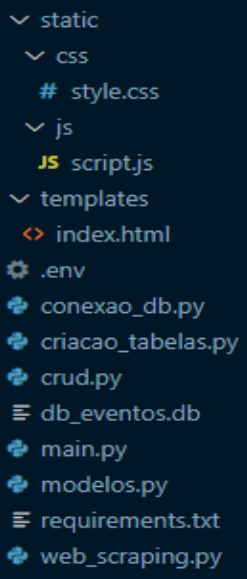
Análise do Código

A seguir estão informações sobre todo o projeto desenvolvido pela aluna.

◆ Diretório Raiz do Projeto

O projeto foi desenvolvido com a estrutura abaixo:

Diretório Raiz



Letícia_Ferreira_Augusto_Alves_Gomes__PB__TP5/

- ***static*** ← Pasta onde os arquivos estáticos estão **armazenados**;
  - ***css*** ← Pasta onde os arquivos ***.CSV*** estão **armazenados**;
    - ***style.css*** ← arquivo de estilização;;
  - ***js*** ← Pasta onde os arquivos ***.CSV*** estão **armazenados**;
    - ***script.js*** ← script de javascript;
- ***templates*** ← Pasta onde o arquivo index.html está **armazenados**;
  - ***index.html*** ← arquivo index;
- ***.env*** ← Arquivo onde a chave do Gemini está armazenada;
- ***conexao_db.py*** ← Arquivo que gerencia a **conexão** com o banco de dados do **SQLite3**.
- ***criacao_tabelas.py*** ← Arquivo que cria tabelas no banco de dados do **SQLite3**.
- ***crud.py*** ← Arquivo que gerencia na **criação** e **população** dos dados no **SQLite3**;
- **db_eventos.db** ← Arquivo do banco de dados do **SQLite3**;
- ***main.py*** ← Arquivo onde todas as **consultas** foram desenvolvidas;
- ***modelos.py*** ← Arquivo que armazena **classes** utilizadas nas **consultas**.
- **requirements.txt** ← Arquivo com as **dependências** a serem **instaladas**.
- **web_scraping.py** ← Arquivo que realiza o **web scraping** em páginas **HTML**.

◆ web_scraping.py

• Importação de bibliotecas

1. **Biblioteca `bs4 BeautifulSoup`:**
   Biblioteca utilizada para raspagem de dados HTML.

2. **Biblioteca `requests`:**
   Biblioteca utilizada para fazer requisições para obter o conteúdo HTML.

3. **Biblioteca `os`:**
   Biblioteca utilizada para manipular arquivos e diretórios.

4. **Biblioteca `google.generativeai`:**
   Biblioteca utilizada para usar a API do Google Gemini.

5. **Biblioteca `google.api_core.exceptions`:**
   Biblioteca utilizada para tratar erros da API do Google Gemini.

6. **Biblioteca `dotenv`:**
   Biblioteca utilizada para carregar variáveis de ambiente.

7. **Biblioteca `dateutil.parser`:**
   Biblioteca utilizada para converter strings de datas em datetime.

8. **Biblioteca `time`:**
   Biblioteca utilizada para a manipulação do tempo.

Importação do Bibliotecas

```python
from bs4 import BeautifulSoup
import requests
import os
import google.generativeai as genai
import google.api_core.exceptions
from dotenv import load_dotenv
from dateutil.parser import parse
import time
```

• Listas Inicializadas

Inicialização de listas:

Listas inicializadas

```python
eventos, dados_eventos, metadados= [], [], []
```

- ## Inicialização do Gemini

Esta função carrega a chave da API do **Google Gemini** e configura o modelo.

Inicializando Gemini

```
load_dotenv()

GEMINI_KEY = os.getenv('GEMINI_KEY')
genai.configure(api_key=GEMINI_KEY)

model = genai.GenerativeModel(model_name='gemini-2.0-flash')
```

- ## Função achar_url()

Esta função faz com que a **URL** de um site se torne **manipulável** no **Python**:

Função *achar_url*()

```
def achar_url(url):
    html = requests.get(url).content
    return BeautifulSoup(html, 'html.parser')
```

- ## Função achar_texto()

Esta função **localiza** o **texto** da **tag HTML**, a partir de seu **atributo**(refere-se ao tipo de atributo do HTML), **nome** (valor do atributo HTML caso não seja False), **tipo** (caso haja, refere-se a decisão entre find_all() ou find()) e **restricao**(caso haja, refere-se a um nome específico de uma tag HTML).

Função *achar_texto*()

```
def achar_texto(atributo, nome, tipo=None, restricao=None):
    if (tipo):
        nomes, c = '', 0
        for attr in soup.find_all(attrs={atributo:nome}):
            if restricao == attr.get_text(strip=True):
                return attr
            if c > 0: nomes += ' '
            c+=1
            nomes += attr.get_text(strip=True)
        return nomes
    return soup.find(attrs={atributo:nome})
```

- ## Função achar_informacoes()

Esta função, a partir do **Google Gemini**, encontra o **tipo** e o **nome** de um **evento** com base em sua **URL** e **prompt**. Caso o **número de solicitações** da API esteja **temporariamente** no **limite**, a função irá esperar **5 segundos** e tentar outra vez.

Função *achar_informacoes*()

```
def achar_informacoes(url):
    try:
        prompt = 'Defina o tipo de evento (música, teatro, arte, stand-up etc.) a partir da URL especificada em apenas uma palavra.'
        tipo = model.generate_content([url, prompt]).text.strip()

        prompt = '''Analise a url lendo as informações e responda apenas o nome do evento
        (caso seja show (não clássico), responda como o nome da banda), sem qualquer outra informação adicional'''
        nome = model.generate_content([url, prompt]).text.strip()

        return tipo, nome

    except (google.api_core.exceptions.ResourceExhausted):
        time.sleep(5)
        return achar_informacoes(url)
```

- ## Função padronizar_data()

Esta função padroniza a data para o formato: AAAA-MM-DD.

Função *padronizar_data*()

```
def padronizar_data(data):
    return parse(data, fuzzy=True).strftime('%Y-%m-%d')
```

- ## Função padronizar_local()

Esta função padroniza o local no formato "local, cidade, país" com ajuda do Gemini IA. Caso o **número de solicitações** da API esteja **temporariamente** no **limite**, a função irá esperar **5 segundos** e tentar outra vez.

Função `padronizar_local()`

```
def padronizar_local(local):
    try:
        prompt='''Pesquise o local e padronize(em português) apenas
    organizando no formato "nome do local, cidade, país",
        sem qualquer informação adicional'''

        return model.generate_content([local, prompt]).text.strip()

    except (google.api_core.exceptions.ResourceExhausted):
        time.sleep(5)
        return padronizar_local(local)
```

- ## Função definir_ambiente()

Esta função define o **tipo** de **ambiente** com base no **local** e no **clima** do local na **data**. Caso o **número de solicitações** da API esteja **temporariamente** no **limite**, a função irá esperar **5 segundos** e tentar outra vez.

Função *`definir_ambiente`*()

```
def definir_ambiente(local, data):
    try:
        prompt = f'''
       Analise o local e informações, considere também o clima previsto na
       data do evento ({data}).
       Depois responda apenas com expressão "Ar Livre" ou "Fechado" de acordo
       com  a probabilidade do evento, sem qualquer outra informação
       adicional.
        '''
        response = model.generate_content([local, prompt])

        return response.text.strip()

    except (google.api_core.exceptions.ResourceExhausted, ValueError):
        time.sleep(5)
        return definir_ambiente(local, data)
```

- ## Função add_evento()

Esta função **adiciona** um **dicionário contendo id, nome** e **tipo** para a lista **eventos.**

Função *add_evento*()

```
def add_evento(nome, tipo):
    eventos.append({'id':len(eventos)+1, 'nome': nome, 'tipo':tipo})
```

- ## Função add_dado_evento()

Esta função **adiciona** um **dicionário** contendo **id, id do evento, data** e **local** para a lista **dado_evento.**

Função *add_dado_evento*()

```
def add_dado_evento(id_evento, data, local):
    dados_eventos.append({'id':len(dados_eventos)+1, 'id_evento':id_evento,
    'data': data, 'local':local})
```

- ## Função add_metadado()

Esta função **adiciona** um **dicionário** contendo **id, id do evento, url** e **ambiente** para a lista **dado_evento**.

Função *add_metadado*()

```
def add_metadado(id_evento, tipo, metadado):
    metadados.append({'id':len(metadados)+1,'id_evento':id_evento,
    'tipo':tipo, 'metadado':metadado})
```

- ## Parâmetros para as Funções

Depois que cada **nome url, local, data, tipo** e **ambiente** forem encontrados com as **funções, os mesmos** são **adicionados** para as **listas correspondentes**:

Parâmetros

```
soup = achar_url(url)
local = padronizar_local(achar_texto(atributo, nome).get_text(strip=True) + ',
    ' + achar_texto(atributo, nome, tipo))
data = padronizar_data(achar_texto(atributo, nome).get_text(strip=True))
tipo, nome = achar_informacoes(url)
ambiente = definir_ambiente(local, data)

add_evento(nome, tipo)
add_dado_evento(len(eventos), data, local)
add_metadado(len(eventos), tipo, metadado)
```

◆ conexao_db.py

• Importação do SQLite3

A biblioteca `sqlite3` foi utilizada para a **conexão** entre o **projeto** e o banco de dados **SQLite3.**

Importação do SQLite3

```
import sqlite3
```

• Função conectar()

A função **`conectar()`** estabelece uma **conexão ativa** com o banco de dados **SQLite3** a partir do nome do base de dados. Caso não haja nenhum erro, a função retornará a instância para **conexão ativa** armazenada na variável **conn**, senão a **conexão não** será **estabelecida**.

Função *conectar*()

```
def conectar():
    try:
        conn = sqlite3.connect('db_empresaTech.db')
        return conn
    except Exception as ex:
        print(ex)
        exit()
```

• Função desconectar()

A função **`desconectar()`** **salva** as alterações e encerra a **conexão ativa** estabelecida pela função **`conectar()`** — caso ela esteja **ativa** — com o banco de dados **SQLite3**.

Função *desconectar*()

```
def desconectar(conn):
    if (conn):
        conn.commit()
        conn.close()
```

◆ modelos.py

• Sequências de Escape ANSI

Estas **variáveis** armazenam **sequências de escape ANSI** para formatar a **cor** das **strings** da **classe Metadado**.

Cores para formatação

```
azul, negrito, reset = '\x1b[38;5;117m', '\033[1m', '\x1b[22m'
```

• Classe Metadado

Esta classe **inicializa** todos os dados de **Metadado** que foram **instanciados** com **palavras-chave**:

Dado	Tipo
**dados	*dict*

Inicialização da Classe `Metadado`

```
class Metadado:
    def __init__(self, **dados):
        self.dados = dados
```

Esta esta função **encontra** o tamanho **máximo** de **caracteres** que uma lista pode **ocupar** a partir de uma **lista de dicionário** e **palavras-chave**.

Função *`encontrarTamanho`*`()` da Classe `Metadado`

```
    def encontrarTamanho(self, lista, cabecalho):
        return max(len(str(dado.dados.get(cabecalho, ''))) for dado in
lista) + 4
```

Esta esta função retorna uma **tabela** dos **dados** em formato de **string**, identificando o **cabeçalho** dos dicionários com o **método .keys()**, depois um **loop** é iniciado para encontrar o **tamanho** de cada **dicionário** a partir da **lista de dicionários** e as **respectivas palavras-chave** com a função `encontrarTamanho()`.

Depois os cabeçalhos da tela são **formatados** com seus respectivos **tamanhos**. A **quantidade** de **caracteres armazenada** na no **dicionario tamanhos** é **somada**. A lista começa a ser **formatada** e **passada** para a lista **resultado** até ser **retornada.**

Função *formatar()* da Classe `Metadado`

```
def formatar(self, lista):
     cabecalhos = lista[0].dados.keys()
     tamanhos = {cabecalho: self.encontrarTamanho(lista, cabecalho) for cabecalho
 in cabecalhos}
     cabeçalhos_formatados = '| ' + ' | '.join(f'{cabecalho:{tamanhos[cabecalho]}}'
                                              for cabecalho in cabecalhos) + ' |'
     qtdCaracteres = sum([tamanhos[nome]+3 for nome in cabecalhos])-1
     resultado = []
     resultado.append(f'{negrito}{azul} ┌{"─"*qtdCaracteres}┐ \n{
                      cabeçalhos_formatados}\n ├{"─"*qtdCaracteres}┤ {reset}')
      for ocorrencia in lista:
         campos = '| ' + ' | '.join(f'{str(ocorrencia.dados.get(nome,"")):{
                                    tamanhos[nome]}}' for nome in cabecalhos) + '
 |'
             resultado.append(campos)
      resultado.append(f' └{"─"*qtdCaracteres}┘ ')
      return '\n'.join(resultado)
```

◆ crud.py

- ### Importação de Módulos

1. **Módulo conexao_db:**
   Arquivo que intermedia a conexão entre o **Python** e o **SQLite3.**

2. **Módulo modelos:**
   Arquivo que ajuda a **estruturar** o código de maneira **modular**.

3. **Biblioteca `StaticFiles`:**
   Biblioteca utilizada para **configurar** a **rota** de **arquivos estáticos**.

4. **Biblioteca `fastapi`:**
   Biblioteca utilizada para construir a **API** com **Python**.

5. **Biblioteca `fastapi.templating`:**
   Biblioteca utilizada no **gerenciamento** de templates **HTML**.

Importação de Módulos

```
from conexao_db import *
from modelos import *
from starlette.staticfiles import StaticFiles
from fastapi import FastAPI, Request
from fastapi.templating import Jinja2Templates
```

- ### Gerenciando Conexão

1. **app:**
   Esta **variável** representa uma **instância** com o **fastapi**.

2. **conn:**
   Esta **variável** representa o **objeto de conexão** com o **banco de dados**.

3. **cursor:**
   Esta variável representa o **objeto** utilizado para **manipular comandos** de dados **SQL.**

Gerenciando Conexão

```
app = FastAPI()
conn = conectar()
cursor = conn.cursor()
```

- ## Configurando Rotas

  Configurando **rota** para as **pastas \templates** e **\static**

Gerenciando Conexão

```
templates = Jinja2Templates(directory="templates")
app.mount("/static", StaticFiles(directory="static"),
name="static")
```

- ## Função consultar_tabelas()

  Esta função realiza **consultas SQL** a partir da **query** passada como **parâmetro**:

```
def consultar_tabelas(comando, arquivo=None):
```

Este bloco é utilizado para tratar na **captura de erros** caso ocorram:

```
try:
   pass
except Exception as ex:
   print(ex)
```

Estas variáveis são **inicializadas** como uma **lista**:

```
lista, armazenamento = [], []
```

Este trecho **executa** a **query** que foi passada como **parâmetro**:

```
cursor.execute(comando)
```

Esta variável **armazena** os **resultados encontrados** do banco de dados:

```
informacoes = cursor.fetchall()
```

Esta variável tem uma **"list comprehension"** que **busca os nomes das colunas** e as **armazena** em formato de **lista**

```
nomeColunas = [nome[0] for nome in cursor.description]
```

Um **loop for** é **inicializado** para passar por cada **dado** dentro de **informações**:

```
for informacao in informacoes:
```

Cada dado será passado como um **dicionário — nomeColunas** e **informacao** foram **combinados** com **zip() —** e adicionado na variável **lista** que **contém a classe dados** e também na variável **armazenamento** retornar para uma **lista normal**.

```
dados = {coluna: resultado for coluna, resultado in zip(nomeColunas, informacao)}
lista.append(Metadado(**dados))
armazenamento.append(dados)
```

Após os blocos acima serem **executados**, a função irá **retornar** a resposta da **classe Metadado** com a função **formatar()** na variável **lista**, e a variável **armazdenamento**:

```
return Metadado().formatar(lista) + '\n', armazenamento
```

Função `consultar_tabelas()`

```
def consultar_tabelas(comando, tp):
     lista, armazenamento = [], []
    cursor.execute(comando)
    informacoes = cursor.fetchall()
    nomeColunas = [nome[0] for nome in cursor.description]
    try:
        for informacao in informacoes:
            dados = {coluna: resultado for coluna, resultado in
zip(nomeColunas, informacao)}
            lista.append(Metadado(**dados))
            armazenamento.append(dados)

    except Exception as ex:
        print(ex)

    return Metadado().formatar(lista) + '\n', armazenamento
```

- ## Função criar_tabelas()

Esta função **cria tabelas SQL** a partir da **query** passada como **parâmetro**:

```
def criar_tabelas(comando):
```

Este bloco é utilizado para **tratar na captura de erros** caso ocorram:

```
try:
   pass
except Exception as ex:
   print(ex)
```

Este trecho **executa** a **query** que foi passada como **parâmetro**:

```
cursor.execute(comando)
```

Função `criar_tabelas()`

```
def criar_tabelas(comando):
    try:
        cursor.execute(comando)

    except Exception as ex:
        print(ex)
```

- ## Função criar_insert()

  Esta função **cria INSERTs** nas **tabelas SQL** a partir da **query** passada como **parâmetro** e seus devidos **valores**:

  ```
  def criar_insert(comando, valores):
  ```

  Este bloco é utilizado para **tratar na captura de erros** caso ocorram:

  ```
  try:
     pass
  except Exception as ex:
     print(ex)
  ```

  Este trecho **executa** a **query** que foi passada como parâmetro **múltiplas vezes**:

  ```
  cursor.executemany(comando, valores)
  ```

Função `criar_insert()`

```
def criar_insert(comando, valores):
    try:
        cursor.executemany(comando, valores)

    except Exception as ex:
        print(ex)
```

◆ criacao_tabelas.py

- ### Importação de Módulo

1. **Módulo crud:**
   Arquivo que é responsável por executar **comandos SQL**.

2. **Módulo web_scraping:**
   Arquivo que é responsável por armazenar dados **HTML**.

Importação de Módulos

```
from crud import *
from web_scraping import eventos, dados_eventos, metadados
```

- ### Função executar_tabelas()

Esta função é responsável pela **execução das tabelas SQL**:

Função executar_tabelas()

```
def executar_tabelas():
```

Este bloco manda a **query** que cria a tabela tb_evento como **parâmetro** da função **criar_tabelas()**:

Criação da tabela tb_evento

```
    criar_tabelas('''
        CREATE TABLE IF NOT EXISTS tb_evento (

            id    INTEGER PRIMARY KEY AUTOINCREMENT,
            nome  VARCHAR(40) NOT NULL,
            tipo  VARCHAR(15) NOT NULL
        );
    ''')
```

Este bloco manda a **query** que cria a tabela **tb_dado_evento** como **parâmetro** da função **criar_tabelas()**:

Criação da tabela tb_dado_evento

```
criar_tabelas('''
    CREATE TABLE IF NOT EXISTS tb_dado_evento (

        id          INTEGER PRIMARY KEY AUTOINCREMENT,
        data        TEXT NOT NULL,
        local       TEXT NOT NULL,
        id_evento   INTEGER NOT NULL,

        FOREIGN KEY (id_evento) REFERENCES tb_evento(id)
    );
''')
```

Este bloco manda a **query** que cria a tabela **tb_metadado** como **parâmetro** da função **criar_tabelas()**:

Criação da tabela tb_metadado

```
criar_tabelas('''
    CREATE TABLE IF NOT EXISTS tb_metadado (

        id          INTEGER PRIMARY KEY AUTOINCREMENT,
        tipo        VARCHAR(15) NOT NULL,
        metadado    TEXT NOT NULL,
        id_evento   INTEGER NOT NULL,

        FOREIGN KEY (id_evento) REFERENCES tb_evento(id)
    );
''')
```

- Inserção de Dados

Este bloco manda a **query** que adiciona todos os dados **armazenados** na lista **eventos** na tabela **tb_evento** que foi **criada anteriormente:**

Inserção de dados na tabela tb_evento

```
    criar_insert('INSERT OR IGNORE INTO tb_evento(id, nome, tipo)
VALUES(:id, :nome, :tipo);',
                 [{'id':evento['id'],
                   'nome':evento['nome'],
                   'tipo':evento['tipo']} for evento in eventos])
```

Este bloco manda a **query** que adiciona todos os dados **armazenados** na lista **dados_eventos** na tabela **tb_dado_evento** que foi **criada anteriormente:**

Inserção de dados na tabela tb_dado_evento

```
    criar_insert('INSERT OR IGNORE INTO tb_dado_evento(id, data,
local, id_evento) VALUES(:id, :data, :local, :id_evento);',
                 [{'id':dado_evento['id'],
                   'data':dado_evento['data'],
                   'local':dado_evento['local'],
                   'id_evento':dado_evento['id_evento']} for
dado_evento in dados_eventos])
```

Este bloco manda a **query** que adiciona todos os dados **armazenados** na lista **metadados** na tabela **tb_metados** que foi **criada anteriormente:**

Inserção de dados na tabela tb_metadado

```
    criar_insert('INSERT OR IGNORE INTO tb_metadado(id, tipo,
metadado, id_evento) VALUES(:id, :tipo, :metadado, :id_evento);',
                 [{'id':metadado['id'],
                   'tipo':metadado['tipo'],
                   'metadado':metadado['metadado'],
                   'id_evento':metadado['id_evento']} for metadado in
metadados])
```

◆ main.py

• Importação de Módulos

1. **Módulo crud:**
   Arquivo que é responsável por **executar comandos SQL.**

2. **Módulo criacao_tabelas:**
   Caso não tenha dados em db_eventos.db o módulo criacao_tabelas é chamado, seu papel é ser responsável pela **criação e população** das tabelas **SQL, em seguida** a função `executar_tabelas()` foi chamada.

Importação de Módulos

```
from crud import *
import os
if (not os.path.getsize('db_eventos.db') > 0):
    from criacao_tabelas import *
    executar_tabelas()
```

• Sequências de Escape ANSI

Esta **variável** armazena **sequências de escape ANSI** para formatar a **cor** das **strings** como amarelo.

Sequências de Escape ANSI

```
amarelo = '\x1b[38;5;229m'
```

• Dados API

Este trecho **inicia** o **endpoint padrão**.

```
@app.get(f'/')
```

Neste trecho **uma função assíncrona** é inicializada. Após os dados serem **armazenados aos seus respectivos TPs**, a função irá **retorná-los** como **requisição** para o **arquivo HTML**:

Função assíncrona name()

```
async def name(request: Request):
    return templates.TemplateResponse("index.html", {"request": request,
                                                     "tp5_1": tp5_1[1],
                                                     "tp5_2": tp5_2[1],
                                                     "tp5_3": tp5_3[1],
                                                     "tp5_4": tp5_4[1],
                                                     "tp5_5": tp5_5[1]})
```

- TP5.1

Este trecho representa uma **consulta SQL** para saber a **listagem** dos **eventos** e seus **respectivos dados**. Para realizar a consulta foi necessário selecionar o dados:

Tabela	Dados
tb_evento	*nome, tipo*
tb_dado_evento	*data, local*

Mostrar todos os eventos com suas datas, localização, e tipo de evento

```
print(f'{amarelo}TP5.1 - Mostrar todos os eventos com suas datas,
localização, e tipo de evento.')
tp5_1 = consultar_tabelas("""
        SELECT E.nome, E.tipo , D.data, D.local
        FROM tb_evento       E
        JOIN tb_dado_evento  D  ON D.id_evento = E.id
        ORDER BY D.data;
    """)
print(tp5_1)
```

Printscreen do resultado da consulta

```
TP5.1 - Mostrar todos os eventos com suas datas, localização, e tipo de evento.
```

nome	tipo	data	local
Afonso Padilha - Stand Up Comedy	Stand-up	2025-03-20	Marte Hall, São Paulo, Brasil
O Figurante	Teatro	2025-03-27	Teatro Renaissance, São Paulo, Brasil
Metallica	Música	2025-04-26	Rogers Centre, Ontario, Canadá
Iron Maiden	Música	2025-06-05	Trondheim Rocks, Trondheim, Noruega
54º Festival de Inverno de Campos do Jordão	Música	2025-06-10	Parque Capivari, Campos do Jordão, Brasil
Linkin Park	Música	2025-08-08	Scotiabank Arena, Toronto, Canadá

- TP5.2

Este trecho representa uma **consulta SQL** para mostrar os **dois eventos** mais **próximos** de **começar**. Para realizar a consulta foi necessário selecionar o dados:

Tabela	Dados
tb_evento	*nome*
tb_dado_evento	*local, data*

Mostrar os dados dos 2 eventos mais próximos de iniciar.

```
print(f'{amarelo}TP5.2 - Mostrar os dados dos 2 eventos mais próximos de iniciar. ')
tp5_2  = consultar_tabelas("""
        SELECT E.nome, D.local, D.data
        FROM tb_evento      E
        JOIN tb_dado_evento D  ON D.id_evento = E.id
        ORDER BY D.data LIMIT 2;
    """)
print(tp5_2)
```

Printscreen do resultado da consulta

```
TP5.2 - Mostrar os dados dos 2 eventos mais próximos de iniciar.
```

nome	local	data
Afonso Padilha - Stand Up Comedy	Marte Hall, São Paulo, Brasil	2025-03-20
O Figurante	Teatro Renaissance, São Paulo, Brasil	2025-03-27

- **TP5.3**

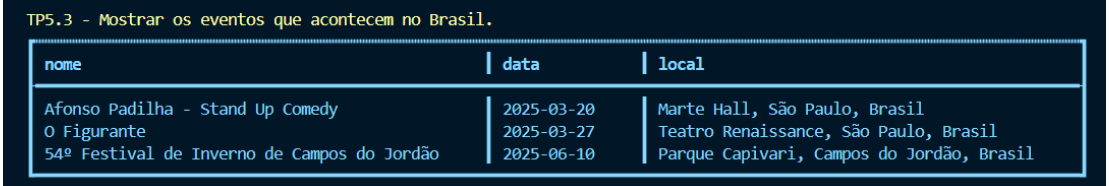
Este trecho representa uma **consulta SQL** para saber quais eventos acontecem no Brasil. Para realizar a consulta foi necessário selecionar o dados:

Tabela	Dados
tb_evento	*nome*
tb_dado_evento	*data, local*

Mostrar os eventos que acontecem no Brasil.

```
print(f'{amarelo}TP5.3 - Mostrar os eventos que acontecem no
Brasil. ')
tp5_3  = consultar_tabelas("""
        SELECT E.nome, D.data, D.local
        FROM tb_evento       E
        JOIN tb_dado_evento  D  ON D.id_evento = E.id
        WHERE D.local LIKE '%Brasil';
    """)
print(tp5_3)
```

Printscreen do resultado da consulta



- **TP5.4**

Este trecho representa uma **consulta SQL** para saber quais eventos são ao ar livre. Para realizar a consulta foi necessário selecionar o dados:

Tabela	Dados
tb_evento	*nome, tipo*
tb_dado_evento	*data, local*
tb_metadado	*metadado*

Mostrar todos os eventos que são ao ar livre

```
print(f'{amarelo}TP5.4 - Mostrar todos os eventos que são ao ar livre. ')
tp5_4  = consultar_tabelas("""
        SELECT E.nome, E.tipo, D.data, D.local, M.metadado AS ambiente
        FROM tb_evento       E
        JOIN tb_dado_evento  D  ON D.id_evento = E.id
        JOIN tb_metadado     M  ON M.id_evento = E.id
        WHERE M.metadado = 'Ar Livre';
    """, "tp5.4")
print(tp5_4[0])
```

Printscreen do resultado da consulta

TP5.4 - Mostrar todos os eventos que são ao ar livre.

nome	tipo	data	local	ambiente
Iron Maiden	Música	2025-06-05	Trondheim Rocks, Trondheim, Noruega	Ar Livre
54º Festival de Inverno de Campos do Jordão	Música	2025-06-10	Parque Capivari, Campos do Jordão, Brasil	Ar Livre

- **TP5.5**

Este trecho representa uma **consulta SQL** para saber qual foi o **projeto** que teve o **maior número** de **dependentes**. Para realizar a consulta foi necessário selecionar o dados:

Tabela	Dados
tb_evento	*nome*
tb_dado_evento	*tipo, metadado*

Metadados por Evento

```
print(f'{amarelo}TP5.5 -  Mostrar todos os Metadados por evento.')
tp5_5  = consultar_tabelas("""
        SELECT E.nome, GROUP_CONCAT(M.tipo, ', ') AS Chaves,
GROUP_CONCAT(M.metadado, ', ') AS Metadados
        FROM      tb_evento   E
        LEFT JOIN tb_metadado M ON M.id_evento = E.id
        GROUP BY E.id;
    """, "tp5.5")
print(tp5_5[0])
```

Printscreen do resultado da consulta

```
TP5.5 -  Mostrar todos os Metadados por evento.
```

nome	Chaves	Metadados
Metallica	ambiente, url	Fechado, https://www.metallica.com/tour/2025-04-26-toronto-ontario-canada.html
Linkin Park	ambiente, url	Fechado, https://www.linkinpark.com/tour
Iron Maiden	ambiente, url	Ar Livre, https://www.ironmaiden.com/tour/run-for-your-lives-world-tour/
Afonso Padilha - Stand-Up Comedy	ambiente, url	Fechado, https://bit.ly/afonso-padilha-stand-up-comedy
O Figurante	ambiente, url	Fechado, https://bit.ly/mateus-solano-em-o-figurante
Festival de Inverno de Campos do Jordão	ambiente, url	Ar Livre, https://bit.ly/54o-festival-de-inverno-de-campos-do-jordao

- Desconectar Base de Dados

Após todo o **código** ser **executado** o banco tem sua **conexão encerrada**.

Encerramento de conexão com MySQL

```
desconectar(conn)
```

◆ requirements.txt

- dependências

Este arquivo representa as dependências a serem instaladas.

requirements.txt

```
fastapi
jinja2
beautifulsoup4
requests
google-generativeai
google-api-core
python-dotenv
python-dateutil
starlette
```

◆ templates

- index.html

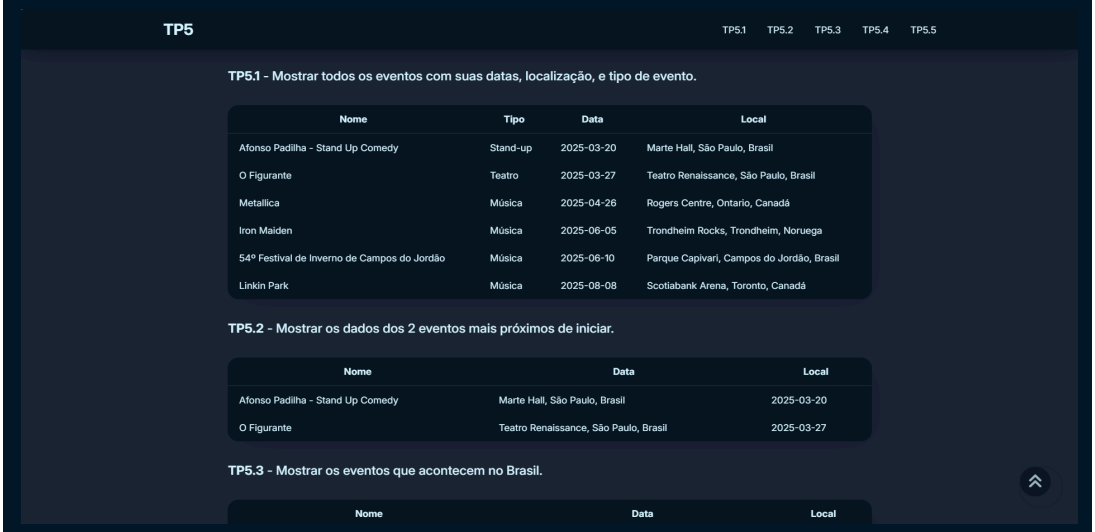
Este trecho do **HTML** representa os dados dos **comandos sql** sendo passados para uma **tabela HTML**.

A partir da estrutura abaixo — utilizando a **sintaxe** de **templating** do **Jinja2** —, foi possivel **integrar** os dados da **consulta do python** para o **arquivo HTML**, mudando apenas os valores das tags th e td, e o nome de cada lista de tp.

index.html

```
<table>
  <thead>
    <tr>
      <th>Nome</th>
      <th>Tipo</th>
      <th>Data</th>
      <th>Local</th>
    </tr>
  </thead>
  <tbody>
    {% for tp in tp5_1 %}
    <tr>
      <td>{{tp.nome}}</td>
      <td>{{tp.tipo}}</td>
      <td>{{tp.data}}</td>
      <td>{{tp.local}}</td>
    </tr>
    {% endfor %}
  </tbody>
</table>
```

Resultado Final



Responsivo