



UNIVERSIDADE
FEDERAL DO PIAUÍ

Universidade Federal do Piauí

Disciplina: Engenharia de Software II

Professor: Armando Soares Sousa

Projeto: Microgram

Equipe:

- **Carlos Eduardo Mendes de Oliveira.**
- **José Pires Gayoso**
- **Fernando Vieira Rosa**
- **Markesley Ramos**
- **Bruno Estrella**

Tutorial 4

Tutorial de como inserir e persistir dados no banco de dados escolhido.

1) Introdução

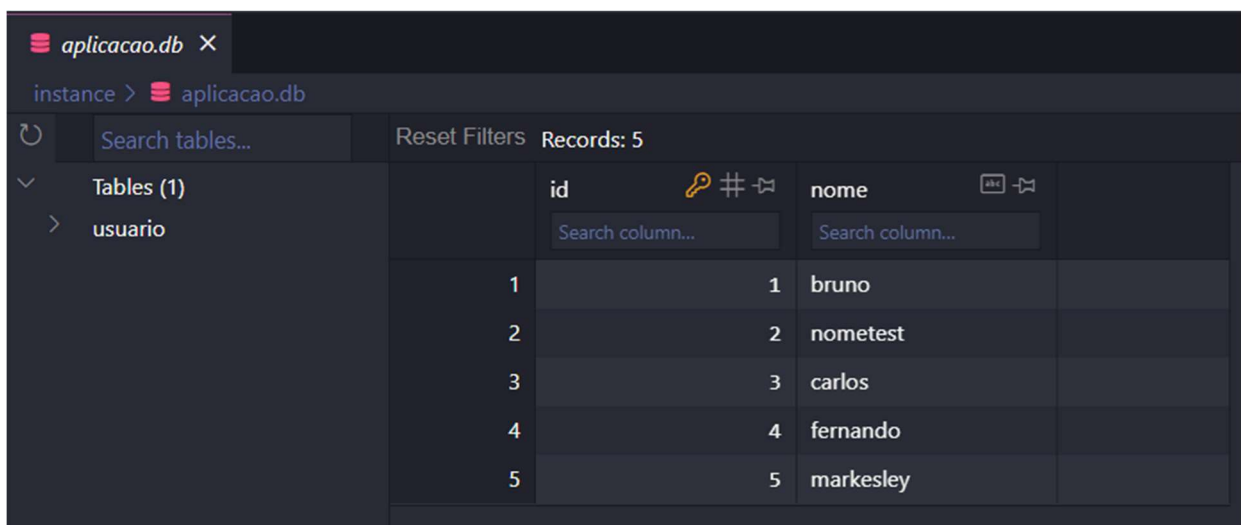
Nesse tutorial usaremos uma aplicação criada usando essas tecnologias como exemplo para facilitar o entendimento. Para ter acesso completo à aplicação de exemplo acesse: https://github.com/BrunoEstrella1707/Tutorial_4

Antes de ver esse tutorial é de suma importância que tenha sido visto o tutorial 3 pois nele

aprendemos a configurar e criar o banco de dados com SQLAlchemy e SQLite. Nesse tutorial usaremos alguns comandos básicos do flask como `render_template` e `redirect`. Também vamos usar outras bibliotecas relacionadas ao flask para que facilite a inserção de dados. Será usada uma extensão do VSCode para visualizarmos os dados do nosso banco. E a parte mais importante serão os comandos do SQLAlchemy para interagirmos com o banco.

2) Inserindo Dados

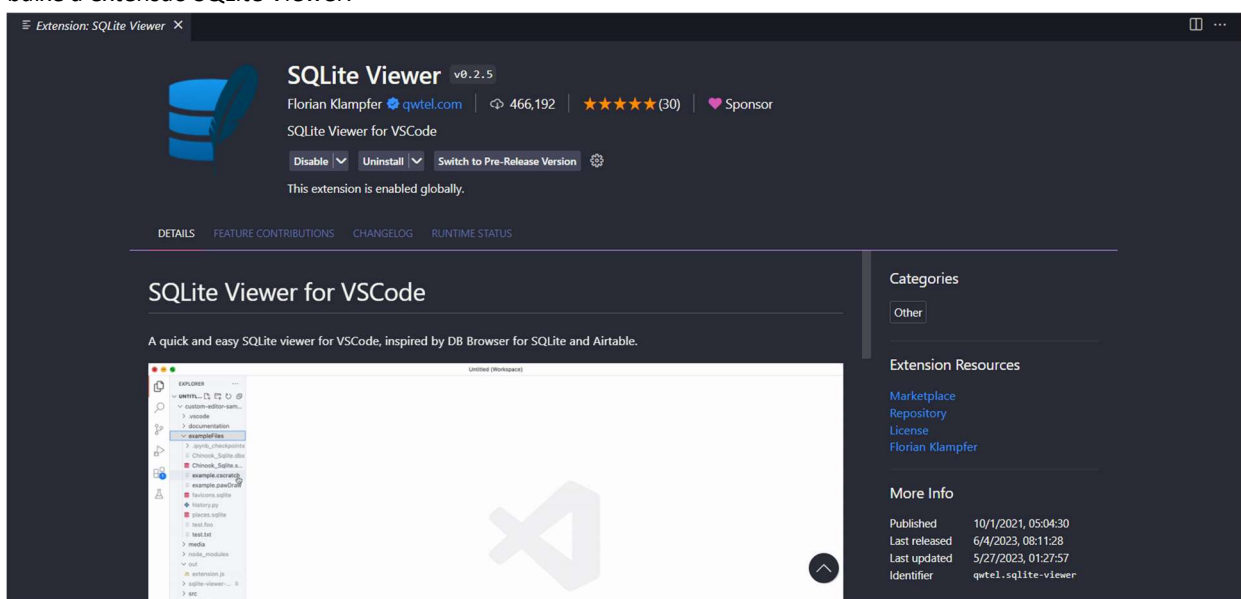
- Após a configuração e criação do banco de dados precisamos inserir dados que podem ser guardados nele. Diferentemente do tutorial 3, usaremos uma tabela onde estão registrados vários usuários.



The screenshot shows the SQLite Viewer interface within VS Code. It displays a table named 'usuario' with 5 records. The table has two columns: 'id' and 'nome'. The records are as follows:

	id	nome
1	1	bruno
2	2	nometest
3	3	carlos
4	4	fernando
5	5	markesley

Obs: Note que a instância do banco está sendo visualizado pelo próprio editor VSCode. Para que isso seja possível baixe a extensão SQLite Viewer.



The screenshot shows the SQLite Viewer extension page in the VS Code Marketplace. The extension is developed by Florian Klampfer and has a rating of 5 stars (30 reviews). It is described as a quick and easy SQLite viewer for VSCode, inspired by DB Browser for SQLite and Airtable. The page includes details, feature contributions, changelog, and runtime status. The extension is currently enabled globally.

SQLite Viewer v0.2.5
Florian Klampfer | qwtel.com | 466,192 | 5 stars (30) | Sponsor
SQLite Viewer for VSCode
Disable | Uninstall | Switch to Pre-Release Version | Settings
This extension is enabled globally.

SQLite Viewer for VSCode
A quick and easy SQLite viewer for VSCode, inspired by DB Browser for SQLite and Airtable.

Categories
Other

Extension Resources
[Marketplace](#)
[Repository](#)
[License](#)
[Florian Klampfer](#)

More Info
Published: 10/1/2021, 05:04:30
Last released: 6/4/2023, 08:11:28
Last updated: 5/27/2023, 01:27:57
Identifier: qwtel.sqlite-viewer

- Agora, usaremos as bibliotecas wtforms e flaskwtfl para criar os formulários de registro

dos nomes. No arquivo de rotas do projeto o formulário `FormularioDeNomes` é instanciado e passado como parâmetro para o template HTML para que possamos usar os dados preenchidos. O `FormularioDeNomes` contém um campo do tipo `String`, que servirá para inserirmos o nome que desejamos inserir no banco, e um campo do tipo `Submit`, que servirá como um botão de confirmação.

```
from flask_wtf import FlaskForm
from wtforms import StringField, SubmitField
from wtforms.validators import DataRequired

class FormularioDeNomes(FlaskForm):

    nome = StringField(Label='Nome: ', validators=[DataRequired()])
    submit = SubmitField(Label='Registrar')
```

Criação do formulário.

```
from tutorial import app
from tutorial import db
from flask import render_template, redirect, url_for
from tutorial.models import Usuario
from tutorial.forms import FormularioDeNomes

@app.route('/', methods=['GET', 'POST'])
def page_home():
    form = FormularioDeNomes()
    if form.validate_on_submit():
        user = Usuario(
            nome = form.nome.data
        )
        db.session.add(user)
        db.session.commit()
        return redirect(url_for('page_home'))

    return render_template("home.html", form=form)
```

Instanciando e passando o formulário como parâmetro.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Página para registrar nomes:</h1>
  <form method='POST' class='form-register', style="color: ■whitesmoke;">

    {{ form.hidden_tag() }}

    {{ form.nome.label() }}
    {{ form.nome(class='form-control', placeholder='Insira seu nome: ') }}

    <br>
    <div class="d-grid">
      {{ form.submit(class='btn btn-lg btn-success') }}
    </div>

    <br>
    <div class="checkbox mb-3">
      <a href="{{url_for('page_show')}}">Mostrar nomes cadastrados</a>
    </div>
  </form>

```

Template que cria o formulário e insere os campos na página HTML.

- Nessa etapa já podemos cadastrar os nomes para que sejam inseridos no Banco de Dados. No template mostrado acima, são adicionados os campos nome e submit do formulário, além da hidden tag que servirá contra proteção contra CSRF. Usamos o método POST para inserir os dados. Observe como os campos do formulário foram adicionados na página:

Página para registrar nomes:

Nome:

[Mostrar nomes cadastrados](#)

- Agora basta adicionarmos os nomes e clicar no botão registrar. Na rota que utilizamos para criar essa página, existe um condicional que verifica se o botão foi apertado ou não, o `validate_on_submit`. Dentro desse condicional, ocorre a lógica de inserção dos dados. Um objeto do tipo `Usuario` é criado e tem seu nome preenchido com o valor passado no campo nome do formulário. Após a criação do objeto `user`, ele é passado como parâmetro na função `db.session.add()`. Depois de adicionarmos o objeto, basta que façamos um `commit` e o usuário será adicionado no banco.

```
if form.validate_on_submit():
    user = Usuario(
        nome = form.nome.data
    )
    db.session.add(user)
    db.session.commit()
    return redirect(url_for('page_home'))
```

3) Recuperando Dados

- Agora iremos recuperar os dados da tabela e os apresentaremos. Para isso foi criada outra rota com uma página que irá mostrar os dados. Na página de rotas, faremos uma consulta na tabela de nomes. Usamos a função `Item.query.all()`, essa função acessa a tabela e retorna todos os registros dela. Note que retornamos todos os registros mas caso quiséssemos retornar apenas alguns, deveríamos usar `Item.query.filter_by()`, passando os parâmetros de como fariíamos a consulta.

```
@app.route('/show')
def page_show():
    itens = Usuario.query.all()
    return render_template('show.html', itens=itens)
```

O retorno da consulta é passado para o template.

- Por fim apresentamos os nomes usando um laço `for` que percorre esses nomes. Cada nome é apresentado como item de uma lista.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Show</title>
</head>
<body>
  <h1>Página para mostrar nomes cadastrados:</h1>
  {% for item in itens %}
    <ul>
      <li>{{item.nome}}</li>
    </ul>
  {% endfor %}
  <br>
  <div class="checkbox mb-3">
    <a href="{{url_for('page_home')}}">Cadastrar mais nomes</a>
  </div>
</body>
</html>

```

Página para mostrar nomes cadastrados:

- bruno
- nometest
- carlos
- fernando
- markesley

[Cadastrar mais nomes](#)

Referência para mais detalhes e/ou dúvidas:

- <https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/>