Programação de Sistemas para Internet

Prof. Romerito Campos

Plano de Aula

- Objetivo: *Acesso ao banco de dados com flask_sqlalchemy*

Conteúdos

- Instalação e Configurações
- Definição de modelos
- Criação do banco de dados
- Consultas básicas

Instalação

Instalação

- A instalação do Flask-SQLAlchemy é feita como a seguir:

```
pip install flask-sqlalchmey
```

- Ao instalar a extensão Flask-SQLAlchemy, você também instala o SQLAlchemy.
- A extensão será uma camada de softaware que realizará operações usando SQLAlchemy.
- A ideia é simplificar o uso do SQLAlchemy.

Configuração

Configuração

- O Exemplo 01 contém uma aplicação que utiliza a extesão Flask-SQLAlchemy
- O projeto está organizado da seguinte forma:

```
case1/
├── database/
│   ├── __init__.py
│   ├── models.py
│   └── utils.py
├── templates/
└── app.py
```

- A pasta `database` é um pacote que contém as principais configurações para uso do Flask-SQLAlchemy.
- O arquivo `app.py` contém algumas configurações necessárias para uso do banco de dados e também as rotas da aplicação.
- Os passos a seguir devem ser realizados:
  - Inicializar a extensão
  - Configurar a extensão
  - Criar o banco
  - Realizar consultas

Inicializar a extensão

Inicializar a extensão

- Esta etapa, no Exemplo 01, foi realizada no arquivo `database/__init__.py`

```
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import DeclarativeBase

class Base(DeclarativeBase):
    pass

db = SQLAlchemy(model_class=Base)
```

- No código anterior, utilizamos a classe `SQLAlchemy` da extensão Flask-SQLAlchemy para definir uma instância do banco (variável `db`)
- Além disso, utilizamos o modo declarativo para definir os modelos do projeto.
- Definimos uma classe `Base`, assim como é feito com SQLAlchemy puro.
- Ao criarmos o objeto `db`, indicamos o modelo de classe a ser usado.
    - `SQLAlchemy(model_class=Base)`

- O objeto `db` dá acesso a duas informações importantes para o restante da configuração e manuseio da conexão com banco.
- Primeiro, podemos acessar o modelo de classe que está sendo usando no SQLAlchemy
  - `db.Model`: será aplicado na criação de modelos
- Segundo, podemos acessar a variável `db.session` que permite executar consultas em geral.

Configura a extensão

Configurar a extensão

- A configuração da extesão envolve:
  - definir a URI do banco
  - vincular a aplicação ao SQLAlchemy
- Estas duas etapas são realizadas com o objeto `app` que está no arquivo `app.py`
- Portanto, vamos importar a variável `db` do pacote database no arquivo `app.py`.

- O código abaixo é um resumo das etapas descritas no slide anterior (Veja Exemplo 01 na íntegra):

```python
# resumo de app.py
from flask import Flask
from database import db

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///projeto.db'
```

- Configurar a `URI` do banco é indicar qual dialeto será usado(SQLITE neste caso) e indicar onde o banco ficará.
  - Neste caso, uma pasta chamada `instance` será criada e o arquivo `projeto.db` ficará dentro dela. Como mudar.

- A segunda ação é vincular a aplicação (`app`) ao SQLAlchemy:

```
# continuação do arquivo app.py
db.init_app(app)
```

- A linha de código acima faz parte do arquivo `app.py`. Indicamos nela o vínculo entre o banco de dados e a aplicação.
  - Neste caso, a URI definida nas configurações do `app` será considerada para gerar o banco.
- O mesmo raciocínio aplica-se ao MYSQL e outros dialetos que o SQLAlchemy suporta.

Definição de Modelos

Definição de Modelos

- A definição de modelos tem uma pequena diferença em relação ao que é feito utilizando SQLAlchemy puro. Veja o exemplo:

```
class User(db.Model):
    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str]
    email: Mapped[str] = mapped_column(unique=True)
```

- Utilizamos `db.Model` como modelo base para a classe `User`.
- Neste exemplo, não utilizamos o atributo `__tablename__` para indicar o nome da tabela gerada no banco.

- É opcional utilizar o atributo `__tablename__` quando adotamos o Flask-SQLALchemy.
  - No caso do modelo `User`, a tabela terá o nome `user`
- O exemplo a seguir define explicitamente o nome da tabela:

```
class Book(db.Model):
    __tablename__ = 'books'
    id: Mapped[int] = mapped_column(primary_key=True)
    title: Mapped[str] = mapped_column(unique=True)
```

- **Observe que o mapeamento dos atributos da tabela é realizado da mesma maneira que o SQLAlchemy**

Criação das tabelas

Criação das tabelas

- A criação das tabelas é realizada no arquivo `app.py`
- Veja o seguinte trecho de código do Exemplo 01

```
# trecho de código
with app.app_context():
    db.create_all()
```

- Este bloco é importante e requer atenção. Se você executar apenas a linha `db.create_all()` sem o bloco `with`, haverá um erro.

- A explicação para uso de `with` e `app_context()` é a seguinte:
  - a aplicação possui configurações (`SQLALCHEMY_DATABASE_URI`)
  - ficar importando o app em outros módulos pode levar a referência circular(erros)
- Portanto, o objeto `db` vai criar o banco e usar a a configuração `SQLALCHEMY_DATABASE_URI`.
- Desta maneira, precisamos executar o `db.create_all()` dentro do contexto da aplicação e ter acesso as suas configurações.
- Também podemos executar `db.create_all()` dentro do contexto de um `request` (na rota).

- Por fim, um detalhe importante é a necessidade de importar os modelos para o mesmo local onde `db.create_all()` é executado.
- Resumo do código `app.py` do Exemplo 01

```
from flask import Flask
#obrigatório
from database.models import *
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///projeto.db'
# necessário registrar no sqlalchemy
db.init_app(app)

with app.app_context():
    db.create_all()
```

Realizar consultas

Realizar consultas

Select

- A realiação de consultas ao banco é feita por meio do objeto `db`.
- A rota `listar` mostra um `SELECT * FROM users`.

```
users = db.session.execute(db.select(User)).scalars()
```

- Observe que acessamos a sessão a partir de db: `db.session`
- Também usamos a função `select()` que é oferecida pela instância `db`. Ela permite criar consultas: `db.select(User)`.

Insert

- Seja `INSERT INTO users(name, emial) values ('zé', 'ze@ze)`.
  - Para executar esta operação basta

```
#trecho de código que pode ser incluído em uma rota
user = User('zé', 'ze@ze')
db.session.add(user)
db.session.commit()
```

- A operação de inserção requer que seja feito o *commit* para consolidar a mudança no banco.

Remoção

- A remoção de um objeto também é realizada via sessão.
- O trecho de código abaixo é parte da rota `delete`:

```
user = db.get_or_404(User, id)
db.session.delete(user)
db.session.commit ()
return redirect(url_for('listar'))
```

- Busca-se o registro com base no `id`. Remove-o e realiza o commit.

UPDATE

- A atualização de dados no banco é tarefa comum.
- Em SQL utilizamos, `UPDATE users SET name='jose' WHERE id=1`.

```
user = db.get_or_404(User,1)
user.name = 'jose'
db.session.commit()
```

- O código acima é usado para recuparar os dados de `id=1` e guardar uma variável `user` (objeto). Altera-se a propriedade `name`. Em seguida, faz-se o commit.

Consultas complexas

- A rota `lista_subs` utiliza string de consulta para listar os nomes das pessoas registradas no banco.
  - Se o texto passado na string de consulta estiver no nome, então o nome aparece na lista resultante.

```
exemplo
http://localhost:5000/listar_subs?nome=jos
```

- O código para este caso aplica uma restrição ao select.

- Veja a rota `listar_subs`. O código abaixo é apenas a busca:

```
users = db.session.execute(db.select(User).where(User.name.contains(nome))).scalars()
```

- Observe que executamos na função `execute` a seguinte declaração:

```
db.select(User).where(User.name.contains(nome))
```

- Esta declaração vai gerar o seguinte SQL

```
SELECT user.id, user.name, user.email
FROM user
WHERE (user.name LIKE '%' || 'jos' || '%')
```

- Se você já vem utilizando o `select()` do pacote SQLAlchemy padrão, notará que a API é a mesma.

```
#trecho de código de exemplo  usando sessão
from sqlalchemy import select

nome='jos'
declaracao = select(User).where(User.name.contains(nome))
resultado = session.execute(declaracao).scalars()
```

- Neste exemplo, teremos o mesmo resultado que foi obtido com flask-SQLAlchemy. **Portanto, dominar a API do SQLAlchemy permite usar a extensão facilmente**.