Migração de Banco de Dados PostgreSQL ⇔ SQLite

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Agenda

- Configuração Inicial
- Criação de Modelos com SQLAlchemy
- Configuração do Alembic
- Gerar e Aplicar Migrações
- Troca do Banco de Dados
- Alteração no esquema do Banco
    - Atualizar o Modelo
    - Gerar um Script de Migração
    - Verificar o Script de Migração
    - Aplicar a Migração
    - Testar a Nova Estrutura

Configuração Inicial

- Instale as dependências necessárias:

```
pip install fastapi uvicorn sqlalchemy psycopg2 alembic
```

Definir os Modelos com SQLAlchemy

- Crie um arquivo chamado **models.py** para definir os modelos do banco de dados:

```python
from sqlalchemy import Column, Integer, String, create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    email = Column(String, unique=True, index=True)

# Configuração do banco de dados SQLite
SQLITE_DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(SQLITE_DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

def init_db():
    Base.metadata.create_all(bind=engine)
```

Configuração do Alembic

- Inicialize o Alembic no projeto:

```
alembic init migrations
```

- Isso cria uma pasta chamada migrations e o arquivo alembic.ini.
- Ajuste o alembic.ini:
  - No arquivo alembic.ini, configure a URL de conexão para o banco SQLite:

```
sqlalchemy.url = sqlite:///./test.db
```

Configuração do Alembic

- Atualize o arquivo env.py:
  - No arquivo migrations/env.py, importe os modelos e o engine:

```python
from models import Base
from sqlalchemy import engine_from_config
from sqlalchemy import pool
from alembic import context

config = context.config
target_metadata = Base.metadata

def run_migrations_offline():
    context.configure(
        url=config.get_main_option("sqlalchemy.url"),
        target_metadata=target_metadata,
        literal_binds=True,
        dialect_opts={"paramstyle": "named"},
    )

    with context.begin_transaction():
        context.run_migrations()

def run_migrations_online():
    connectable = engine_from_config(
        config.get_section(config.config_ini_section),
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
    )

    with connectable.connect() as connection:
        context.configure(connection=connection, target_metadata=target_metadata)

        with context.begin_transaction():
            context.run_migrations()

if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

Gerar e Aplicar Migrações

- Geração da migração inicial:

```
alembic revision --autogenerate -m "Initial migration"
```

- Isso cria um script de migração na pasta migrations/versions.
- Aplicar a migração:

```
alembic upgrade head
```

- Isso cria as tabelas definidas no modelo no banco SQLite.

Troca para PostgreSQL

- Atualize a URL de conexão no alembic.ini:

```
sqlalchemy.url = postgresql+psycopg2://user:password@localhost/dbname
```

Troca para PostgreSQL

- Migre os dados de SQLite para PostgreSQL
- Use uma ferramenta como **pgloader** ou escreva um script Python para copiar os dados.

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import User, SessionLocal

# Conexão com o PostgreSQL
POSTGRES_DATABASE_URL = "postgresql+psycopg2://postgres:2023@localhost/dbmigration"
postgres_engine = create_engine(POSTGRES_DATABASE_URL)

# Copiar dados de SQLite para PostgreSQL
def migrate_data():
    sqlite_session = SessionLocal()
    postgres_session = sessionmaker(bind=postgres_engine)()

    users = sqlite_session.query(User).all()
    for user in users:
        # Crie uma nova instância do objeto User para desvincular da sessão SQLite
        new_user = User(id=user.id, name=user.name, email=user.email)
        postgres_session.add(new_user)

    postgres_session.commit()

    sqlite_session.close()
    postgres_session.close()

if __name__ == "__main__":
    migrate_data()
```

Troca para PostgreSQL

- Reaplique as migrações no PostgreSQL:

```
alembic upgrade head
```

Aplicação FastAPI

- Crie um arquivo **main.py**:
- Este fluxo permite a migração do banco de dados de SQLite para PostgreSQL com Alembic e a integração com FastAPI para gerenciar os dados.
- A abordagem garante que o banco de dados permaneça consistente e sincronizado durante o desenvolvimento.

```python
from fastapi import FastAPI, Depends
from sqlalchemy.orm import Session
from models import User, SessionLocal, init_db

app = FastAPI()

# Inicializa o banco de dados
init_db()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/users/")
def create_user(name: str, email: str, db: Session = Depends(get_db)):
    user = User(name=name, email=email)
    db.add(user)
    db.commit()
    db.refresh(user)
    return user

@app.get("/users/")
def read_users(skip: int = 0, limit: int = 10, db: Session = Depends(get_db)):
    return db.query(User).offset(skip).limit(limit).all()
```

Alteração no esquema do Banco

Atualizar o Modelo, Gerar um Script de Migração, Verificar o Script de Migração, Aplicar a Migração e Testar a Nova Estrutura

Atualizar o Modelo

- Primeiro, atualize o modelo no arquivo models.py para incluir a nova coluna.

- Por exemplo, se você quiser adicionar uma coluna age:

```
from sqlalchemy import Column, Integer, String,
create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    email = Column(String, unique=True, index=True)
    age = Column(Integer, nullable=True)  # Nova coluna
adicionada
```

Gerar um Script de Migração

- Use o comando abaixo para criar um script de migração com Alembic:

```
alembic revision --autogenerate -m "Add age column to users table"
```

- Este comando gerará um arquivo de migração na pasta migrations/versions.

- Ele detecta automaticamente a nova coluna age e criará o código SQL correspondente.

Verificar o Script de Migração

- Abra o arquivo gerado na pasta migrations/versions e verifique o conteúdo. Ele deve conter algo assim:

```python
from typing import Sequence, Union
from alembic import op
import sqlalchemy as sa

# revision identifiers, used by Alembic.
revision: str = '420910d3e373'
down_revision: Union[str, None] = 'b2484433e6fa'
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None

def upgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column('users', sa.Column('age', sa.Integer(), nullable=True))
    # ### end Alembic commands ###

def downgrade() -> None:
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_column('users', 'age')
    # ### end Alembic commands ###
```

Aplicar a Migração

- Depois de verificar o script, aplique a migração ao banco de dados:

```
alembic upgrade head
```

- Este comando executará o código da função upgrade() no script, adicionando a coluna age à tabela users.

Testar a Nova Estrutura

- Por exemplo, você pode atualizar a rota para criar um usuário e incluir a coluna age:

```python
@app.post("/users/")
def create_user(name: str, email: str, age: int, db: Session = Depends(get_db)):
    user = User(name=name, email=email, age=age)
    db.add(user)
    db.commit()
    db.refresh(user)
    return user
```

- Agora, ao enviar um dado com age, ele será salvo na tabela atualizada.

Referências

- Curso completo de FastAPI por Eduardo Mendes
  - https://fastapidozero.dunossauro.com/
  - https://github.com/dunossauro/fastapi-do-zero
  - Playlist no YouTube

- FastAPI - https://fastapi.tiangolo.com/
- Pydantic - https://pydantic.dev/
- SQLAlchemy - https://www.sqlalchemy.org/
- SQLModel - https://sqlmodel.tiangolo.com

- https://docs.github.com/pt/rest/using-the-rest-api/using-pagination-in-the-rest-api?apiVersion=2022-11-28

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br