Mapeamento Objeto-Relacional: Migrações de Banco de Dados com Alembic PostgreSQL ⇔ SQLite

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

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

Agenda

- Conceitos básicos de migração de banco de dados
  - O que é Migração?
  - Por que Utilizar Migração em Projetos de Software?
  - Cenários Comuns de Migração entre PostgreSQL e SQLite
  - Cuidados na Migração entre PostgreSQL e SQLite
- Tipos de Migração
- Vantagens e Desvantagens
- Ferramentas Necessárias
- Estratégia de Migração
- Componentes de uma Migração
- Boas Práticas
- Visão geral do Alembic
- Prática
  - Instalação das Ferramentas
  - Configuração do Alembic

Conceitos básicos de migração de banco de dados

O que é migração?

Por que utilizar migração em projetos de software?

Cenários comuns de migração entre PostgreSQL e SQLite

Cuidados na Migração entre PostgreSQL e SQLite

O que é Migração?

- Migração, em termos de banco de dados, refere-se ao processo de modificar a estrutura e/ou os dados de um banco de forma controlada e incremental, mantendo a integridade das informações e evitando erros ou inconsistências.

- **Isso pode incluir:**
  - Alterações no esquema (tabelas, colunas, índices).
  - Modificações nos dados (atualização ou transformação).
  - Mudança de uma plataforma de banco de dados para outra (ex.: PostgreSQL para SQLite).
  - Ferramentas de migração são frequentemente utilizadas para gerenciar essas alterações de forma organizada, permitindo que desenvolvedores mantenham o banco sincronizado com o código da aplicação.

Por que Utilizar Migração em Projetos de Software?

- **Manutenção do Banco de Dados**
  - À medida que o software evolui, o modelo de dados também precisa ser ajustado para refletir novas funcionalidades ou mudanças nas regras de negócio.

- **Controle de Versão do Banco**
  - Assim como o código-fonte, o banco de dados deve ter um histórico de alterações para rastrear o que foi modificado, quando e por quê.

Por que Utilizar Migração em Projetos de Software?

- **Colaboração em Equipe**
  - Em equipes grandes, migrações permitem que todos mantenham seus bancos de dados locais atualizados e consistentes com o ambiente de produção.

- **Automatização e Redução de Erros**
  - Ferramentas de migração automatizam a aplicação de mudanças, reduzindo a possibilidade de erros humanos ao executar scripts SQL manualmente.

- **Rollback Fácil**
  - Em caso de problemas, migrações oferecem suporte para reverter alterações e restaurar o estado anterior do banco de dados.

Cenários Comuns de Migração entre PostgreSQL e SQLite

- PostgreSQL e SQLite são sistemas de gerenciamento de banco de dados populares, mas possuem características distintas. A migração entre eles pode ocorrer em diferentes cenários:

- **Desenvolvimento Local com SQLite e Produção com PostgreSQL**
  - Motivo: SQLite é leve e fácil de configurar, ideal para ambientes de desenvolvimento e testes, enquanto PostgreSQL é mais robusto e adequado para produção.
  - Desafio: Diferenças de tipos de dados, funções SQL e comportamento transacional.

- **Testes Automatizados**
  - Motivo: Projetos podem usar SQLite em memória para rodar testes rapidamente, enquanto a aplicação em produção utiliza PostgreSQL.
  - Desafio: Garantir que os testes simulem adequadamente o ambiente de produção, já que algumas funcionalidades específicas do PostgreSQL não estão disponíveis no SQLite.

Cenários Comuns de Migração entre PostgreSQL e SQLite

- **Migração Temporária para Portabilidade**
  - Motivo: Mover um banco PostgreSQL para SQLite para facilitar a distribuição ou portabilidade do software.
  - Desafio: Conversão de schemas e ajustes de queries SQL para compatibilidade com SQLite.

- **Prototipagem Rápida com SQLite**
  - Motivo: Usar SQLite para criar um protótipo rapidamente e depois migrar para PostgreSQL ao entrar em produção.
  - Desafio: Adaptar o esquema e os dados do protótipo ao banco de produção.

- **Substituição de Banco de Dados**
  - Motivo: Migrar de SQLite para PostgreSQL quando o projeto cresce e necessita de mais recursos (ex.: suporte a conexões simultâneas, replicação, segurança avançada).
  - Desafio: Transferir dados, reconfigurar a aplicação e ajustar queries SQL específicas.

Cuidados na Migração entre PostgreSQL e SQLite

- **Tipos de Dados**
  - PostgreSQL oferece mais tipos de dados (ex.: JSON, arrays, UUID), enquanto SQLite tem um sistema de tipagem mais flexível.
  - Ajustes podem ser necessários para tipos como SERIAL (PostgreSQL) e INTEGER PRIMARY KEY (SQLite).

- **Diferenças em Funções e Sintaxe**
  - Funções como NOW() ou CURRENT_TIMESTAMP podem se comportar de forma diferente.
  - Queries SQL que funcionam em um banco podem precisar de ajustes no outro.

Cuidados na Migração entre PostgreSQL e SQLite

- **Chaves Estrangeiras**
  - O suporte a chaves estrangeiras é habilitado por padrão no PostgreSQL, mas deve ser ativado explicitamente no SQLite (PRAGMA foreign_keys = ON).

- **Indexação e Performance**
  - Os métodos de indexação podem variar entre os dois bancos, afetando a performance das queries.

- **Ferramentas de Migração**
  - Ferramentas como pgloader podem ajudar a transferir dados entre PostgreSQL e SQLite.
  - Scripts personalizados podem ser necessários para adaptar schemas ou corrigir inconsistências.

Tipos de Migração

- **Estrutural:** Alterações no esquema (ex.: criar ou remover tabelas/colunas, modificar tipos de dados).

- **Dados:** Modificação dos dados existentes (ex.: atualizar valores, migrar para novos formatos).

- **Plataforma:** Migração entre diferentes sistemas de gerenciamento de banco de dados (ex.: de MySQL para PostgreSQL).

Vantagens e Desvantagens

- **Vantagens:**
  - Versionamento: Histórico claro das mudanças no banco de dados.
  - Automação: Reduz erros manuais ao alterar esquemas.
  - Facilidade em rollback: Possibilidade de reverter rapidamente uma migração problemática.
  - Consistência: Garante que todos os ambientes (dev, teste, produção) estejam sincronizados.

- **Desvantagens:**
  - Complexidade inicial: Requer configuração e aprendizado inicial.
  - Dependência do Alembic: Pode dificultar migrações manuais.
  - Limitações de autogenerate: Em casos complexos, ajustes manuais podem ser necessários nos scripts gerados.

Ferramentas Necessárias

- **Alembic:** Usado em projetos Python, funciona bem com SQLAlchemy.
- **Flyway e Liquibase:** Ferramentas independentes de linguagem para controle de versão do banco.
- **Django Migrations:** Sistema integrado ao framework Django.
- **Rails Active Record Migrations:** Parte do Ruby on Rails.

Estratégia de Migração

- **Versão Inicial:** Define o estado inicial do banco.
- **Scripts Incrementais:** Cada alteração no banco é registrada em um script de migração.
- **Rollback/Reversão:** Permite desfazer alterações em caso de erro.
- **Automatização:** Uso de ferramentas para criar e aplicar migrações automaticamente.

Componentes de uma Migração

- **Script de Migração:** Arquivo contendo instruções SQL ou código para alterar o banco.

- **Versão do Banco:** Identificador que relaciona o estado atual do banco com o código.

- **Comandos Básicos:**
  - **up** ou **migrate**: Aplica as mudanças no banco.
  - **down** ou **rollback**: Reverte mudanças aplicadas.

Boas Práticas

- **Testar em Ambiente de Desenvolvimento** antes de aplicar em produção.
- **Fazer Backup do Banco** antes de qualquer migração.
- **Documentar as Alterações** para facilitar o entendimento e manutenção.
- **Sincronizar Código e Banco**: Certificar-se de que a aplicação e o banco estão na mesma versão.

Visão Geral do Alembic

O que é Alembic?

Como ele auxilia nas migrações de banco de dados?

Integração com SQLAlchemy.

O que é Alembic?

- **Alembic** é uma ferramenta de migração de banco de dados para aplicações Python que utilizam **SQLAlchemy** como ORM (Object-Relational Mapping).

- Ela permite gerenciar alterações no esquema do banco de dados de forma incremental e controlada, rastreando mudanças no modelo de dados ao longo do tempo.

- Alembic é amplamente utilizado em projetos que requerem a evolução do banco de dados em paralelo ao desenvolvimento do código da aplicação.

Como Alembic Auxilia nas Migrações de Banco de Dados?

- **Controle de Versão do Esquema**
  - Alembic mantém um histórico de alterações feitas no esquema do banco de dados. Cada mudança é registrada como uma revisão em scripts de migração, facilitando a rastreabilidade.

- **Aplicação Incremental de Alterações**
  - As migrações são aplicadas de forma incremental, permitindo que o banco de dados seja atualizado gradualmente para refletir as alterações no modelo.

Como Alembic Auxilia nas Migrações de Banco de Dados?

- **Rollback de Alterações**
  - Em caso de erro, é possível reverter facilmente uma migração, retornando o banco de dados ao estado anterior.

- **Automatização**
  - Alembic pode gerar scripts de migração automaticamente com base nas alterações detectadas no modelo do SQLAlchemy, reduzindo o esforço manual.

- **Sincronização entre Equipes e Ambientes**
  - Permite que diferentes desenvolvedores mantenham seus bancos de dados locais em sincronia com o ambiente de produção.

Principais Funcionalidades do Alembic

- **Criação de Scripts de Migração**
  - Geração de scripts de migração a partir de alterações no modelo SQLAlchemy:

```
alembic revision --autogenerate -m "Descrição da migração"
```

- **Aplicação de Migrações**
  - Atualiza o banco de dados para a versão mais recente:

```
alembic upgrade head
```

Principais Funcionalidades do Alembic

- **Rollback de Migrações**
  - Reverte uma ou mais migrações:

```
alembic downgrade -1
```

- **Configuração Personalizável**
  - O arquivo alembic.ini e a pasta env.py permitem configurar o comportamento do Alembic, como conexão com o banco e diretórios de scripts.

Integração com SQLAlchemy

- **Conexão com o Banco de Dados**
  - Alembic utiliza a string de conexão definida em alembic.ini ou no arquivo env.py, que pode ser a mesma usada pelo SQLAlchemy na aplicação.
    - Exemplo de string de conexão no alembic.ini:

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

Integração com SQLAlchemy

- **Detecção Automática de Alterações no Modelo**
  - Quando configurado corretamente, Alembic pode inspecionar os modelos definidos no SQLAlchemy e gerar scripts de migração automaticamente:

```
alembic revision --autogenerate -m "Nova migração"
```

- **Execução de Scripts SQL Customizados**
  - Além de alterações automáticas, Alembic permite adicionar comandos SQL personalizados nos scripts de migração gerados, oferecendo flexibilidade para atender a requisitos específicos.

Exemplo de Fluxo com Alembic e SQLAlchemy

- **Inicializar o Alembic no Projeto**
  - Isso cria a estrutura básica para gerenciar migrações.

```
alembic init migrations
```

Exemplo de Fluxo com Alembic e SQLAlchemy

- **Definir os Modelos no SQLAlchemy**

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

Base = declarative_base()

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

Exemplo de Fluxo com Alembic e SQLAlchemy

- **Gerar uma Migração Inicial**

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

- **Aplicar a Migração**

```
alembic upgrade head
```

Exemplo de Fluxo com Alembic e SQLAlchemy

- **Adicionar uma Nova Coluna ao Modelo**

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

- **Gerar e Aplicar uma Nova Migração**

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

Prática

- Configuração Inicial
  - *pip install fastapi uvicorn sqlalchemy psycopg2 alembic*
- Criação de Modelos com SQLAlchemy
- Configuração do Alembic
- Gerar e Aplicar Migrações
- Troca do Banco de Dados


LET'S
CODE

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