Bancos de dados evolutivos

(migrações)

Live de Python # 211

Design evolutivo de dbs
Migração de schemas
Migrações com Alembic

Titúlos possíveis para essa live!

Essa live não é sobre

Migração de dados
ou
Mudança de banco de dados x para y

IMPORTANTE!!!

Roteiros

1. Migrações

O que são bancos de dados evolutivos?

2. Alembic

Migrações com Python + SQLAlchemy

3. Modelos automáticos

Como gerar modelos a partir de um banco pronto?

4. Migrações automáticas

Como gerar migrações a partir de modelos?

5. Já passei por isso!

Problemas comuns quando aprendemos alembic

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Acássio Anjos, Ademar Peixoto, Adilson Herculano, Alexandre Harano, Alexandre Lima, Alexandre Souza, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Ana Padovan, Andre Azevedo, André Rafael, André Rocha, Aquiles Coutinho, Arnaldo Turque, Aurelio Costa, Bruno Freitas, Bruno Guizi, Bruno Oliveira, Bruno Ramos, Caio Nascimento, Carlos Alipio, Christiano Morais, Clara Battesini, Daniel Freitas, Daniel Haas, Danilo Segura, David Kwast, Delton Porfiro, Dhyeives Rodovalho, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino Aguilar, Diogo Paschoal, Douglas Bastos, Douglas Braga, Douglas Zickuhr, Dutofanim Dutofanim, Emerson Rafael, Érico Andrei, Eugenio Mazzini, Euripedes Borges, Evandro Avellar, Everton Silva, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Fernando Rozas, Flávio Meira, Flavkaze Flavkaze, Franklin Silva, Gabriel Barbosa, Gabriel Simonetto, Geandreson Costa, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Gustavo Dettenborn, Gustavo Suto, Heitor Fernandes, Henrique Junqueira, Hugo Cosme, Igor Taconi, Israel Gomes, Italo Silva, Jair Andrade, Jairo Lenfers, Janael Pinheiro, João Lugão, João Paulo, João Rodrigues, Joelson Sartori, Johnny Tardin, Jonatas Leon, Jonatas Oliveira, Jônatas Silva, José Gomes, Joseíto Júnior, Jose Mazolini, Juan Gutierrez, Juliana Machado, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Leonardo Mello, Leonardo Nazareth, Lucas Mello, Lucas Mendes, Lucas Oliveira, Lucas Polo, Lucas Teixeira, Lucas Valino, Luciano Silva, Luciano Teixeira, Luiz Carlos, Luiz Junior, Luiz Lima, Luiz Paula, Luiz Perciliano, Maicon Pantoja, Maiquel Leonel, Marcelino Pinheiro, Marcelo Matte, Márcio Martignoni, Marco Mello, Marcos Gomes, Marco Yamada, Maria Clara, Marina Passos, Mateus Lisboa, Matheus Cortezi, Matheus Silva, Matheus Vian, Mírian Batista, Murilo Andrade, Murilo Cunha, Murilo Viana, Natan Cervinski, Nicolas Teodosio, Osvaldo Neto, Otávio Barradas, Patricia Minamizawa, Paulo Braga, Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, P Muniz, Priscila Santos, Rafael Lopes, Rafael Rodrigues, Rafael Romão, Ramayana Menezes, Reinaldo Silva, Renato Veirich, Ricardo Silva, Riverfount Riverfount, Robson Maciel, Rodrigo Alves, Rodrigo Freire, Rodrigo Vaccari, Rodrigo Vieira, Rogério Sousa, Ronaldo Silva, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Sara Selis, Thiago Araujo, Thiago Borges, Thiago Bueno, Thiago Curvelo, Thiago Moraes, Thiago Oliveira, Thiago S, Thiago Souza, Tiago Minuzzi, Tony Dias, Valcilon Silva, Valdir Tegon, Victor Wildner, Vinícius Bastos, Vítor Gomes, Vitor Luz, Vladimir Lemos, Walter Reis, Wellington Abreu, Wesley Mendes, William Alves, Willian Lopes, Wilson Neto, Wilson Rocha, Xico Silvério, Yury Barros

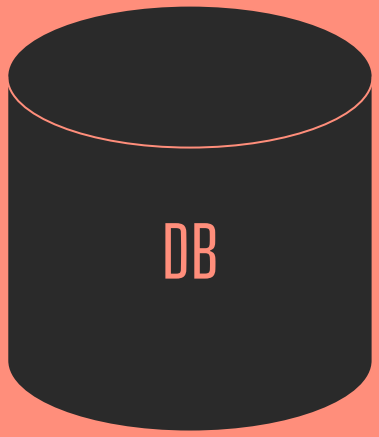
Obrigado você

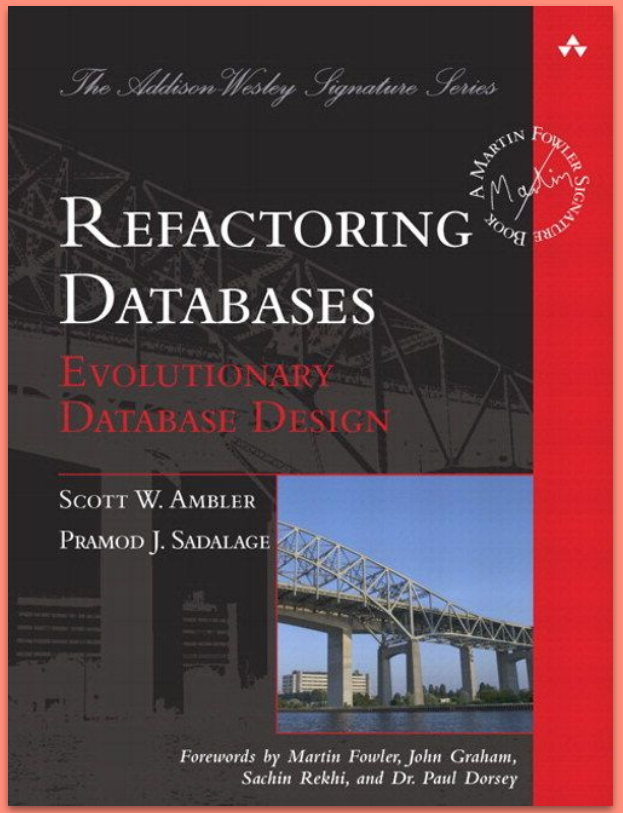
Migrações

Bancos de dados evolutivos

Bancos de dados evolutivos

"O banco deve ser planejado com antecedência"





Agile e os bancos de dados

Com o decorrer de técnicas ágeis, de entrega constante e reformulações inerente de sistemas. O banco de dados deve sofrer diversas "refatorações" com o decorrer do trabalho em um sistema.

Então deve existir uma forma incremental para que o banco deva evoluir em conjunto com a aplicação.

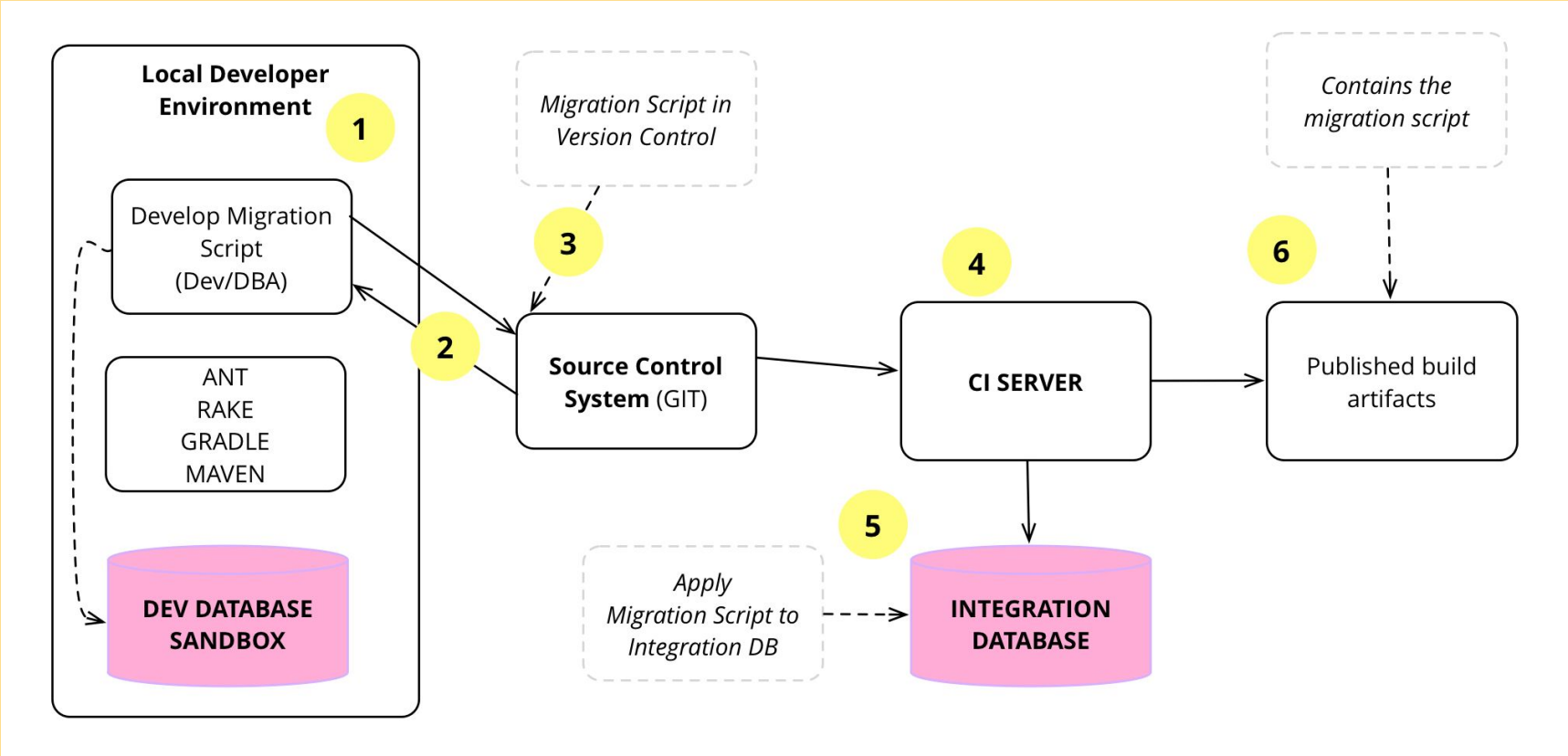
Versionamento do estado do banco

Isso deve ser feito de uma forma em que deve ser possível fazer rollback tanto na aplicação, quanto no banco. E isso deve ser versionado junto com o software. No git, por exemplo.

Isso pode trazer diversas vantagens:

- Todas as alterações do banco estão em um único lugar
- Se um problema ocorrer, podemos voltar o estado do banco e do app
- Não existe dessincronia entre o app e o banco
- Criação de ambientes diferentes de produção com o mesmo "schema"

Interação contínua com o banco



https://martinfowler.com/articles/evodb.html

Pontos a serem levados em consideração

- DBAs + Devs = <3
- Migrações versionadas
- Toda alteração deve ser uma migração
- Cada pessoa pode reconstruir o ambiente do banco de dados
- Bibliotecas podem fazer as migrações de forma automática
  - Integração contínua pode acontecer no banco também
- Migrações podem adicionar mais coisas do que schema
  - Dados de teste
  - População de tabelas padrão
    - Usuário administrativo?
  - Seeds
    - Dados necessários para o banco operar

Migrações com python

Alembic

O que é o Alembic?

O Alembic foi inicialmente desenvolvido por Mike Bayers, mesmo criador do SQLAlchemy. Teve sua primeira versão lançada em novembro de 2011. 5 anos após a primeira versão do SQLAlchemy, em 2006.

- Pode trabalhar em toda a camada de DLL
- Fornece scripts de migração de schemas para upgrades e downgrades
- Suporte a geração de SQL (offline)
- API minimalista

https://techspot.zzzeek.org/

pip install alembic

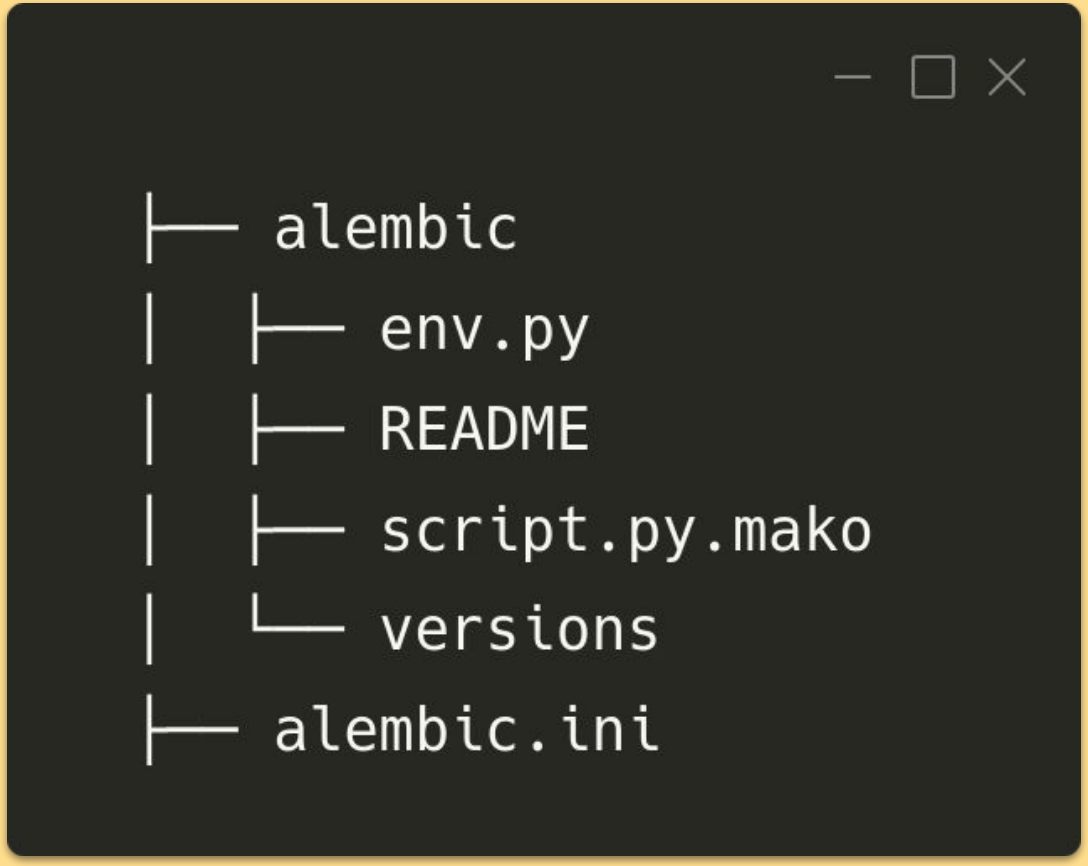
Bora instalar!

inicializando o alembic [0]

Para inicializar as migrações no seu projeto você precisa executar no terminal:

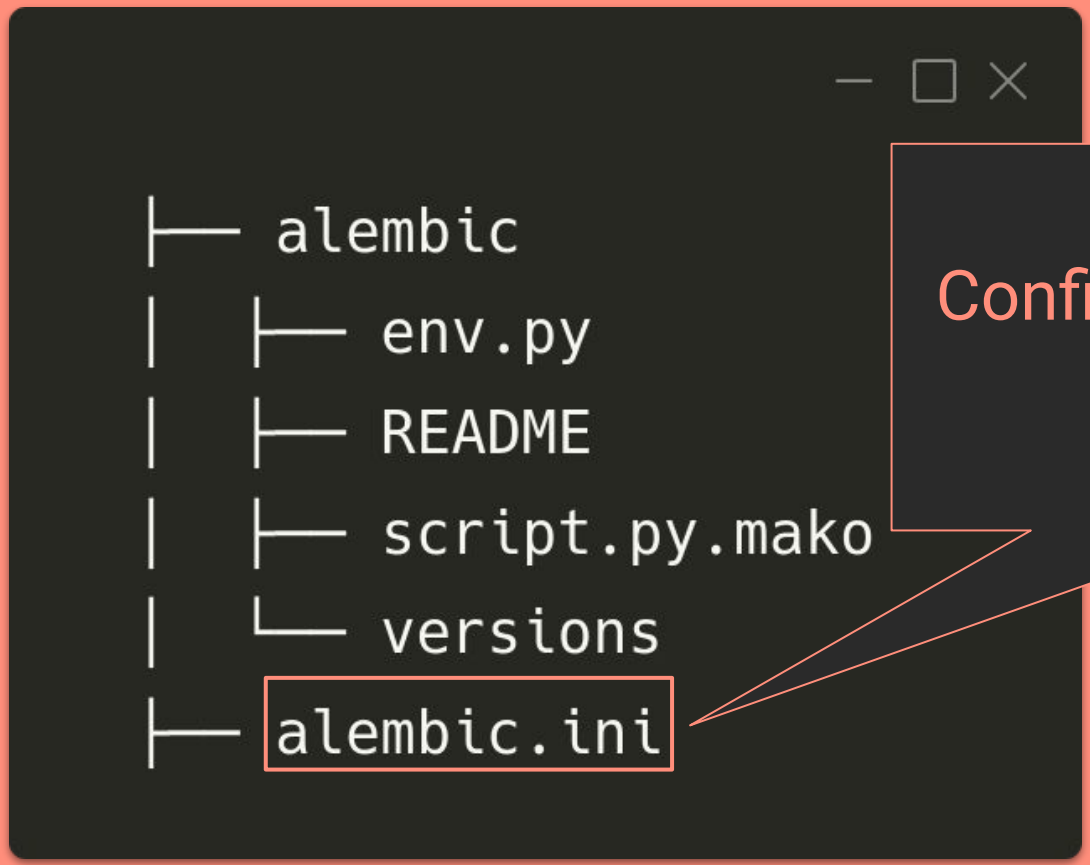
```
alembic init <nome> # pode ser qualquer nome
alembic init alembic
```

Inicializando o alembic [1]



Arquivos gerados pela migração

O arquivo alembic.ini



Configurações gerais do alembic

Configurações gerais

```
# A generic, single database configuration.

[alembic]
script_location = alembic
prepend_sys_path = .
version_path_separator = os
sqlalchemy.url = driver://user:pass@localhost/dbname
```

Configurações gerais

Pasta dos arquivos da migração.
alembic init <nome>

```
# A generic, single database configuration.

[alembic]
script_location = alembic
prepend_sys_path = .
version_path_separator = os
sqlalchemy.url = driver://user:pass@localhost/dbname
```

Configurações gerais

onde está a pasta `alembic`

```
# A generic, single database configuration.

[alembic]
script_location = alembic
prepend_sys_path = .
version_path_separator = os
sqlalchemy.url = driver://user:pass@localhost/dbname
```

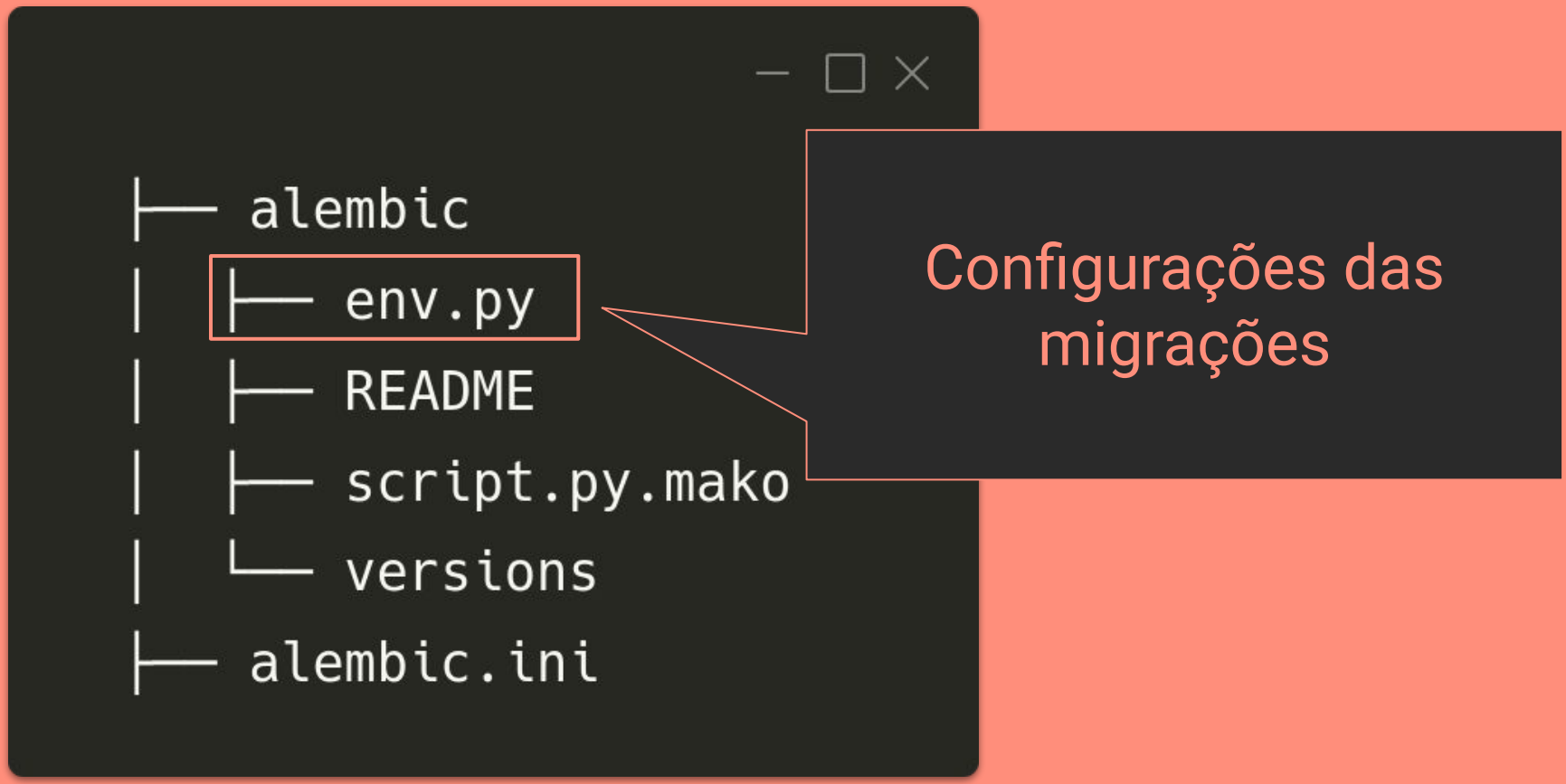
Configurações gerais

URI do banco de dados
ALTERAR ISSO

```
# A generic, single database [illegible]ation.

[alembic]
script_location = alembic
prepend_sys_path = .
version_path_separator = os
sqlalchemy.url = driver://user:pass@localhost/dbname
```

O arquivo env.py



Arquivo env.py

```python
# env.py
from sqlalchemy import engine_from_co
from sqlalchemy import pool

from alembic import context

config = context.config
# ...
target_metadata = None

# ...

def run_migrations_offline() -> None:
    ...

def run_migrations_online() -> None:
    ...
```

Chave [alembic] do arquivo alembic.ini

Arquivo env.py

```python
# env.py
from sqlalchemy import engine_from_
from sqlalchemy import pool

from alembic import context

config = context.config
# ...
target_metadata = None

# ...

def run_migrations_offline() -> None:
    ...

def run_migrations_online() -> None:
    ...
```

Metadados do database declarativo

Vamos usar na geração automática

Arquivo env.py

```python
# env.py
from sqlalchemy import engine_from_
from sqlalchemy import pool

from alembic import context

config = context.config
# ...
target_metadata = None

# ...

def run_migrations_offline() -> None:
    ...

def run_migrations_online() -> None:
    ...
```

Funções que aplicam as migrações

Vamos editar isso quando necessário

Gerando nossa primeira migração

Agora que entendemos a estrutura do alembic. Pordemos gerar nossa primeira migração!

```
alembic revision -m "primeira"
Generating <path>/alembic/versions/96be538716d4_primeira.py ...  done
```

Gerando nossa primeira migração

Agora que entendemos a estrutura do alembic. Pordemos gerar nossa primeira migração!

```
alembic revision -m "primeira"
Generating <path>/alembic/versions/96be538716d4_primeira.py ...  done
```

As migrações ficam na pasta versions

Gerando nossa primeira migração

Agora que entendemos a estrutura do alembic. Pordemos gerar nossa primeira migração!

```
alembic revision -m "primeira"
Generating <path>/alembic/versions/96be538716d4_primeira.py ...  done
```

Identificador da migração

```
"""primeira  # Nome da migração

Revision ID: 96be538716d4  # GUID da migração
Revises:
Create Date: 2022-07-17 16:54:39.976681

"""
from alembic import op    # op = Operations
import sqlalchemy as sa

revision = '96be538716d4'  # GUID da migração
down_revision = None       # GUID da migração anterior
branch_labels = None       # GUID da próxima migração
depends_on = None


def upgrade() -> None:  # Função que aplica essa migração
    pass


def downgrade() -> None:  # Função que desaplica essa migração
    pass
```

Api de operações

A API de operações funciona para os comandos de **DDL** (Data Definition Language):

- Criação de tabelas (CREATE TABLE)
- Alteração de tabelas (ALTER TABLE)
- Deleção de tabelas (DROP TABLE)

Essas operações devem ser feitas dentro das funções de upgrade() e downgrade()

Exemplo das operações de DDL

```python
from alembic import op
# ...

def upgrade() -> None:
    op.create_table(
        '<nome_da_tabela>',
        ...
    )

    op.add_column(
        '<nome_da_table>',
        ...
    )


def downgrade() -> None:
    op.drop_table('<nome_da_tabela>')
    op.drop_column('<nome_da_tabela>', '<nome_da_coluna>')
```

Objetos no SQLAlchemy Core

Os tipos de dados e as abstrações necessárias para executar a API de operações estão presentes no SQLAlchemy core

https://youtu.be/rBIksyGY4_E

Implementando a nossa primeira migração

```python
from alembic import op
import sqlalchemy as sa

# ...

def upgrade() -> None:
    op.create_table(
        'pessoa',
        sa.Column('id', sa.Integer(), primary_key=True),
        sa.Column('nome', sa.String(length=50), nullable=False),
        sa.Column('email', sa.String(length=50), nullable=False)
    )
```

Downgrade da migração

```python
from alembic import op
import sqlalchemy as sa

# ...

def downgrade() -> None:
    op.drop_table('pessoa')
```

Configuração do banco de dados no alembic

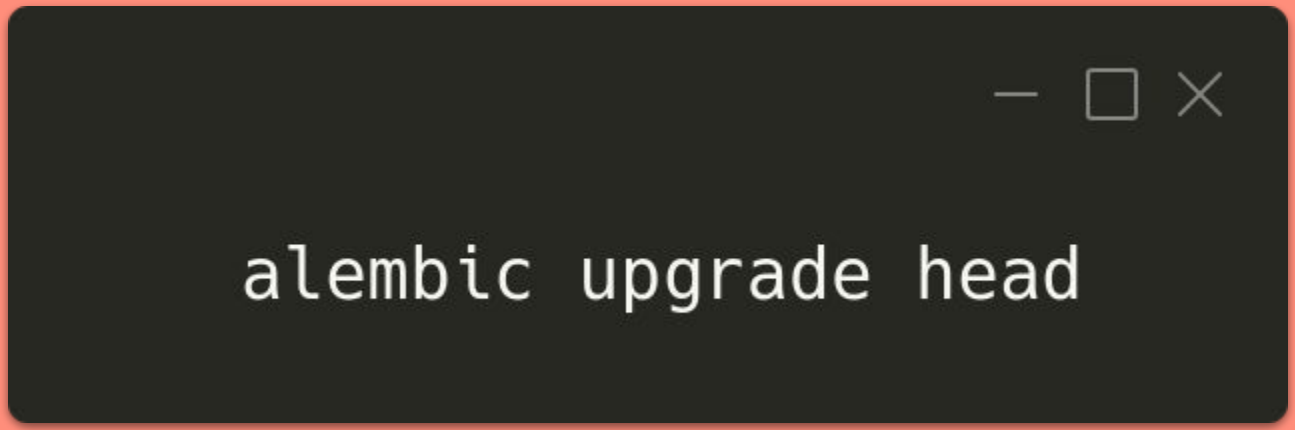
Antes de aplicar a migração de fato, devemos dizer ao Alembic onde está nosso banco de dados. Para isso precisamos passar a URI do banco no **alembic.ini**

```
[alembic]
sqlalchemy.url = sqlite:///live_migrations.db
```

Vamos usar SQLite para diminuir a complexidade da live*

Poderia ser qualquer outra URI de um banco que SQLAlchemy oferece suporte

Agora sim, MIGRANDO!!!



Checando se a migração aconteceu

```
sqlite3 live_migrations.db

sqlite> .tables
alembic_version pessoa
sqlite> .schema
CREATE TABLE alembic_version (
    version_num VARCHAR(32) NOT NULL,
    CONSTRAINT alembic_version_pkc PRIMARY KEY (version_num)
);

CREATE TABLE pessoa (
    id INTEGER,
    nome VARCHAR(50) NOT NULL,
    email VARCHAR(50) NOT NULL
);
```

Checando se a migração aconteceu

```
sqlite3 live_migrations.db

sqlite> .tables
alembic_version pessoa
sqlite> .schema
CREATE TABLE alembic_version (
    version_num VARCHAR(32) NOT NULL,
    CONSTRAINT alembic_version_pkc PRIMARY KEY (version_num)
);

CREATE TABLE pessoa (
    id INTEGER,
    nome VARCHAR(50) NOT NULL,
    email VARCHAR(50) NOT NULL
);
```

O que está armazenado na tabela do alembic?

```
sqlite> SELECT * FROM alembic_version;
96be538716d4  # O valor da revisão
```

Desfazendo e aplicando migrações

O alembic conta com uma boa funcionalidade de histórico de migrações e como progredir de uma migração para outra

```
alembic history
alembic history --indicate-current

alembic upgrade <ID>
alembic upgrade head
alembic upgrade +1

alembic downgrade <ID>
alembic downgrade base
alembic downgrade -1
```

SQLA CodeDen

Geração de modelos automáticos

Já tenho um Banco e agora?

Um outro cenário comum é começar um projeto com um banco já existente ou ter começado um projeto sem modelos evolutivos.

Para fazer isso, podemos fazer um "reflection" do banco de dados em código.

Para criar os modelos que já existem no banco para criarmos a primeira migração, podemos usar o sqlacodegen

pip install sqlacodegen

Instalação

SQLACodeGen

Usando a URI do banco, o codegen vai olhar sua base de dados e contruir os modelos usando o SQLAlchemy ORM para você:

```
sqlacodegen <DB_URI>
sqlacodegen sqlite:///live_migrations.db
```

Código gerado

```python
# Código gerado pelo SQACodeGen
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import declarative_base

Base = declarative_base()


class Pessoa(Base):
    __tablename__ = 'pessoa'

    id = Column(Integer, primary_key=True)
    nome = Column(String(50), nullable=False)
    email = Column(String(50), nullable=False)
```

Mas eu não uso o ORM!

O código gerado automaticamente pode ter os 3 formatos nativos do SQLAlchemy:

- Core - tables
- ORM - declarative
- Dataclasses - dataclasses

Mas para isso, precisamos da versão 3 do SQLAcodegen. Ela está em **release candidate** nos dia da live (jul/2022)

pip install sqlacodegen==3.0.0rc1

Gerando diferentes modelos

```
# Core
sqlacodegen --generator tables sqlite:///live_migrations.db

# ORM (default)
sqlacodegen --generator declarative sqlite:///live_migrations.db

# Dataclasses (1.4+)
sqlacodegen --generator dataclasses sqlite:///live_migrations.db
```

Como gerar migraçãos pelos modelos?

Migrações Automaticas

Usando os metadados

```python
# Código gerado pelo SQACodeGen
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import declarative_base

Base = declarative_base()
```

```python
# env.py
from sqlalchemy import engine_from_config
from sqlalchemy import pool

from alembic import context

config = context.config
# ...
target_metadata = None
```

Aplicando ao arquivo

```python
# env.py
from models import Base
# ...
target_metadata = Base.metadata
```

Alterando o modelo

```python
# models.py
class Pessoa(Base):
    # ...
    senha = Column(String(50), nullable=False)
```

Uma migração auto gerada

```
alembic revision --autogenerate \
    -m "campo de senha na tabela pessoa"
```

O resultado

```
"""campo de senha na tabela pessoa

Revision ID: e3e118fe4d4b
Revises: 96be538716d4
Create Date: 2022-07-17 19:22:55.442104

"""
# ...
def upgrade() -> None:  # Adicionando a coluna senha
    # ### commands auto generated by Alembic - please adjust! ###
    op.add_column('pessoa', sa.Column('senha', sa.String(length=50), nullable=False))
    # ### end Alembic commands ###


def downgrade() -> None:  # Rmovendo a coluna senha
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_column('pessoa', 'senha')
    # ### end Alembic commands ###
```

Problemas comuns

Já passei por isso

Sem acesso ao banco de produção!

Por questão de segurança, as vezes não podemos aplicar as migrações no banco de produção. Ou então, elas só podem ser feitas por DBAs. Ou então a alteração pode fazer o projeto parar de funcionar por alguns momentos.

Migrações Offline:

```
alembic upgrade +1 --sql
```

```
CREATE TABLE alembic_version (
    version_num VARCHAR(32) NOT NULL,
    CONSTRAINT alembic_version_pkc PRIMARY KEY (version_num)
);

CREATE TABLE pessoa (
    id INTEGER NOT NULL,
    nome VARCHAR(50) NOT NULL,
    email VARCHAR(50) NOT NULL,
    PRIMARY KEY (id)
);

INSERT INTO alembic_version (version_num) VALUES ('96be538716d4');
```

Comparação de tipos

Por padrão, as migrações não comparam campos já existentes. Então, se precisar alterar um campo de uma tabela, pode ter problemas!

```python
# env.py
def run_migrations_XXX() -> None:
    # ...
    with connectable.connect() as connection:
        context.configure(
            connection=connection,
            target_metadata=target_metadata,
            compare_type=True  # Adicionar essa linha
        )
```

Quebrando a aplicação por alguns segundos!

Se você fizer uma migração online e um acesso for feito nesse meio tempo, a pessoa pode pegar um período instável do banco. Por ele vai fazer operações linha a linha. Para fazer todas as modifinções de uma vez, usamos as **operações de batch**:

```python
def upgrade() -> None:
    with op.batch_alter_table('pessoa', schema=None) as batch_op:
        batch_op.alter_column('nome',
               existing_type=sa.VARCHAR(length=50),
               type_=sa.String(length=100),
               existing_nullable=False)
```

Quebrando a aplicação por alguns segundos!

```python
def upgrade() -> None:
    with op.batch_alter_table('pessoa', schema=None) as batch_op:
        batch_op.alter_column('nome',
               existing_type=sa.VARCHAR(length=50),
               type_=sa.String(length=100),
               existing_nullable=False)


def downgrade() -> None:
    with op.batch_alter_table('pessoa', schema=None) as batch_op:
        batch_op.alter_column('nome',
               existing_type=sa.String(length=100),
               type_=sa.VARCHAR(length=50),
               existing_nullable=False)
```

Quebrando a aplicação por alguns segundos!

```python
# env.py
def run_migrations_XXX() -> None:
    # ...
    with connectable.connect() as connection:
        context.configure(
            connection=connection,
            target_metadata=target_metadata,
            compare_type=True,
            render_as_batch=True  # Adicionar essa linha
        )
```

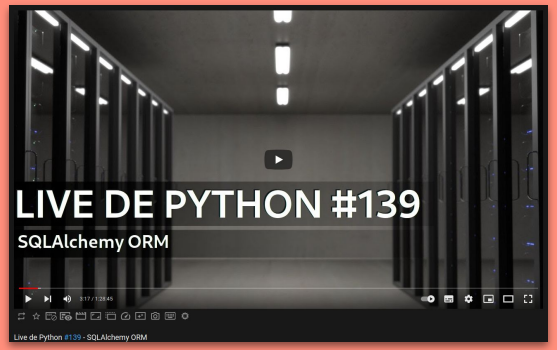
```python
def upgrade() -> None:
    with op.batch_alter_table('pessoa',
        batch_op.alter_column('nome',
               existing_type=sa.VARCHAR(
               type_=sa.String(length=1(
               existing_nullable=False)


def downgrade() -> None:
    with op.batch_alter_table('pessoa', schema=None) as batch_op:
        batch_op.alter_column('nome',
               existing_type=sa.String(length=100),
               type_=sa.VARCHAR(length=50),
               existing_nullable=False)
```

Lives que podem te auxiliar a prosseguir!

https://youtu.be/rBIksyGY4_E

https://youtu.be/nwJYbbJfJpg



https://youtu.be/UgaybOYMKS0

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3

Links de referência

Referencial teórico:

- [texto] DBs evolutivos: https://martinfowler.com/articles/evodb.html
- [livro] Refatoração de bancos de dados: https://bitlybr.com/bzbOL

Alembic:

- Documentação do Alembic: https://alembic.sqlalchemy.org/
- Operators: https://alembic.sqlalchemy.org/en/latest/ops.html
- Blog do Mike Bayer: https://techspot.zzzeek.org/

Lives de Python anteriores:

- SQLAlchemy core #11: https://youtu.be/rBIksyGY4_E
- SQLAlchemy ORM #139: https://youtu.be/UgaybOYMKS0
- SQLAlchemy 1.4+ #166: https://youtu.be/nwJYbbJfJpg