Mapeamento Objeto-Relacional: Implementação utilizando ORM SQLAlchemy + SQLite

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

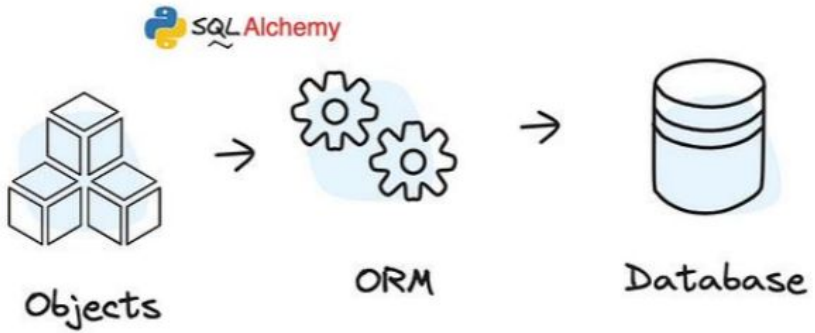
Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Agenda

- ORM
- Principais recursos de um ORM
- Vantagens de um ORM
- Ponte entre objetos e tabelas
- SQLAlchemy
- Principais componentes do SQLAlchemy
- Arquitetura do SQLAlchemy
- Instalação
- Exemplo de implementação
- Níveis de logging do Python
- Logando consultas SQL
- Migrações de banco de dados
- Implementação

ORM

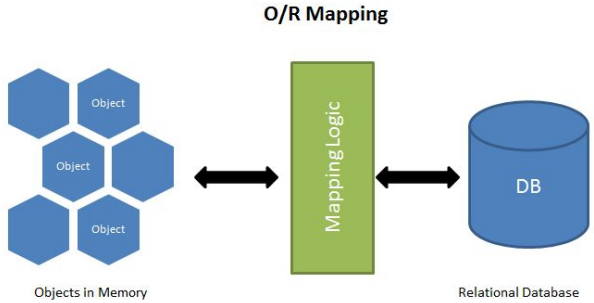
- ORM (Object Relational Mapping) é uma camada que permite conectar a programação orientada a objetos com bancos de dados relacionais, abstraindo os comandos SQL subjacentes.



Principais recursos de um ORM

- **Mapeamento de classes e modelos:** O ORM permite mapear classes e modelos de forma a realizar operações no banco de dados.

- **Tradução entre objetos e tabelas:** O ORM traduz automaticamente as instruções SQL para refletir as mudanças no banco de dados, e transforma os dados recuperados do banco em objetos.



Principais recursos de um ORM

- **Redução da complexidade do código:** O ORM torna o código mais fácil de entender e manter.
- **Maior portabilidade:** O ORM permite que o desenvolvedor trabalhe com diferentes bancos de dados.
- **Recursos avançados de segurança:** O ORM oferece recursos de segurança, como a prevenção de SQL injection.

Exemplos de ORMs

- Prisma, um ORM NodeJS que possui o Prisma Client, Prisma Migrate e Prisma Studio
- Hibernate, uma ferramenta ORM para Java que simplifica o mapeamento entre entidades Java e tabelas de banco de dados
- JPA (Java Persistence API), uma especificação para a persistência de dados em Java
- SQLAlchemy ORM para Python

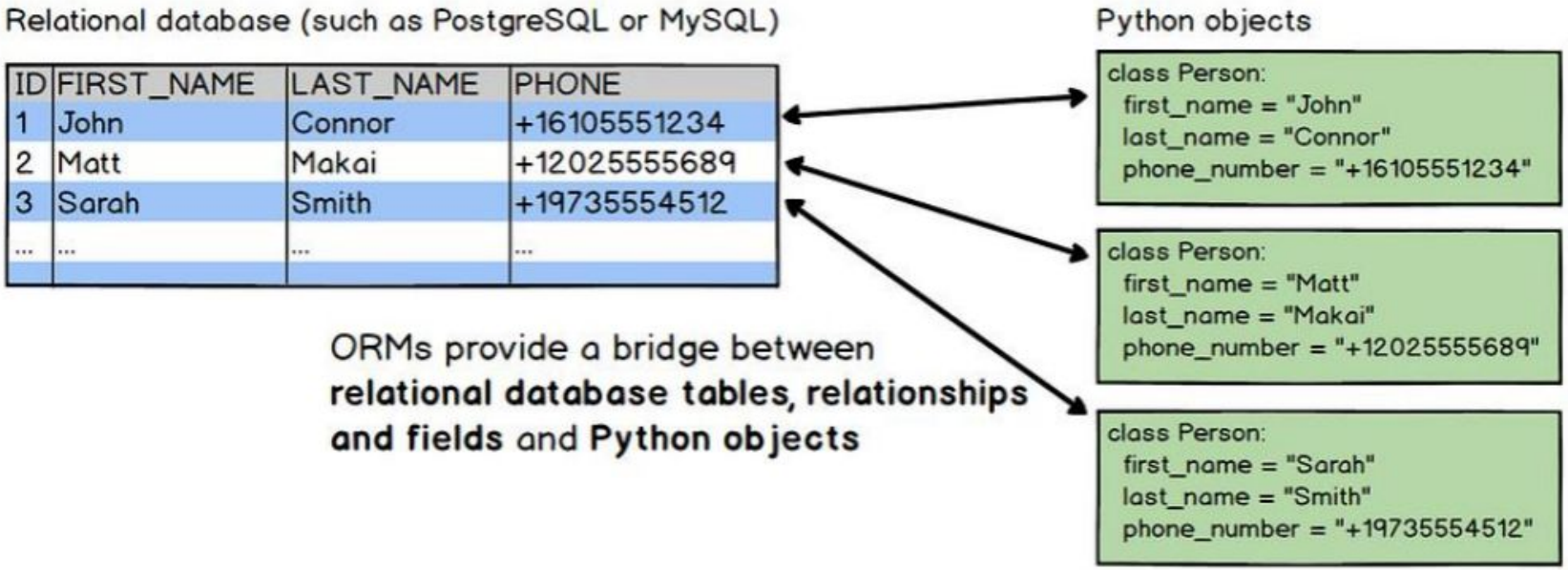
Vantagens de um ORM

- Redução de código repetitivo
  - Não é necessário escrever comandos SQL manualmente.

- Portabilidade
  - O código pode ser mais facilmente adaptado para diferentes bancos de dados.

- Legibilidade
  - Consultas e manipulações de dados se tornam mais intuitivas para os desenvolvedores.

Ponte entre objetos e tabelas



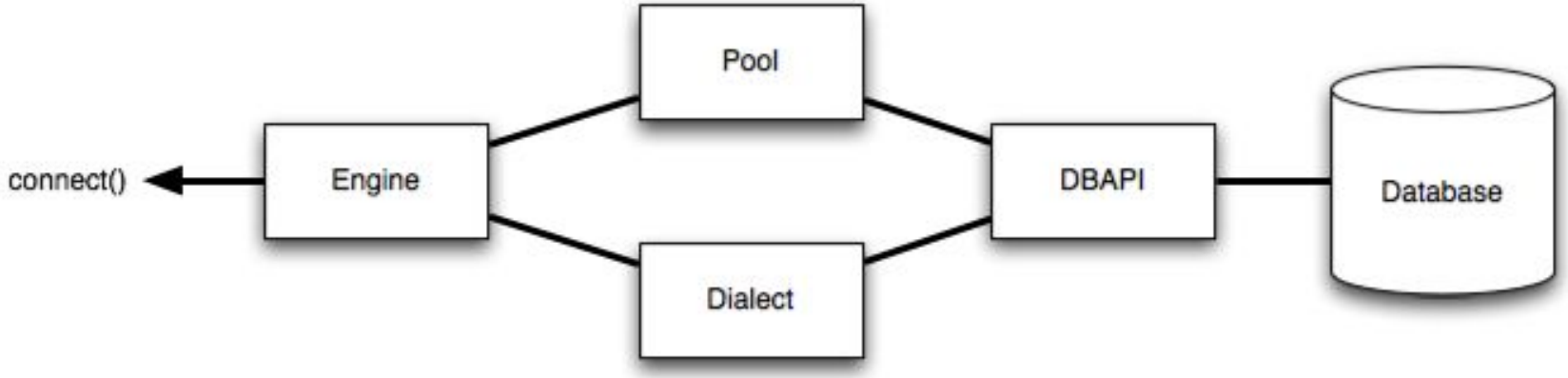
SQLAlchemy

- É uma das bibliotecas ORM mais populares no ecossistema Python.
- Permite a execução de ações em bancos de dados relacionais.
- Simplifica interações com bancos de dados relacionais.

- Suporte nativo a:
  - Microsoft SQL Server
  - MySQL / MariaDB
  - Oracle
  - PostgreSQL
  - SQLite

- Embora seja projetado para bancos relacionais, ele pode ser estendido para trabalhar com bancos NoSQL por meio de adaptações e extensões como MongoAlchemy.

Principais componentes



Principais componentes

- Engine
  - Interface que permite a interação com o banco de dados. Ele lida com conexões e executa consultas.

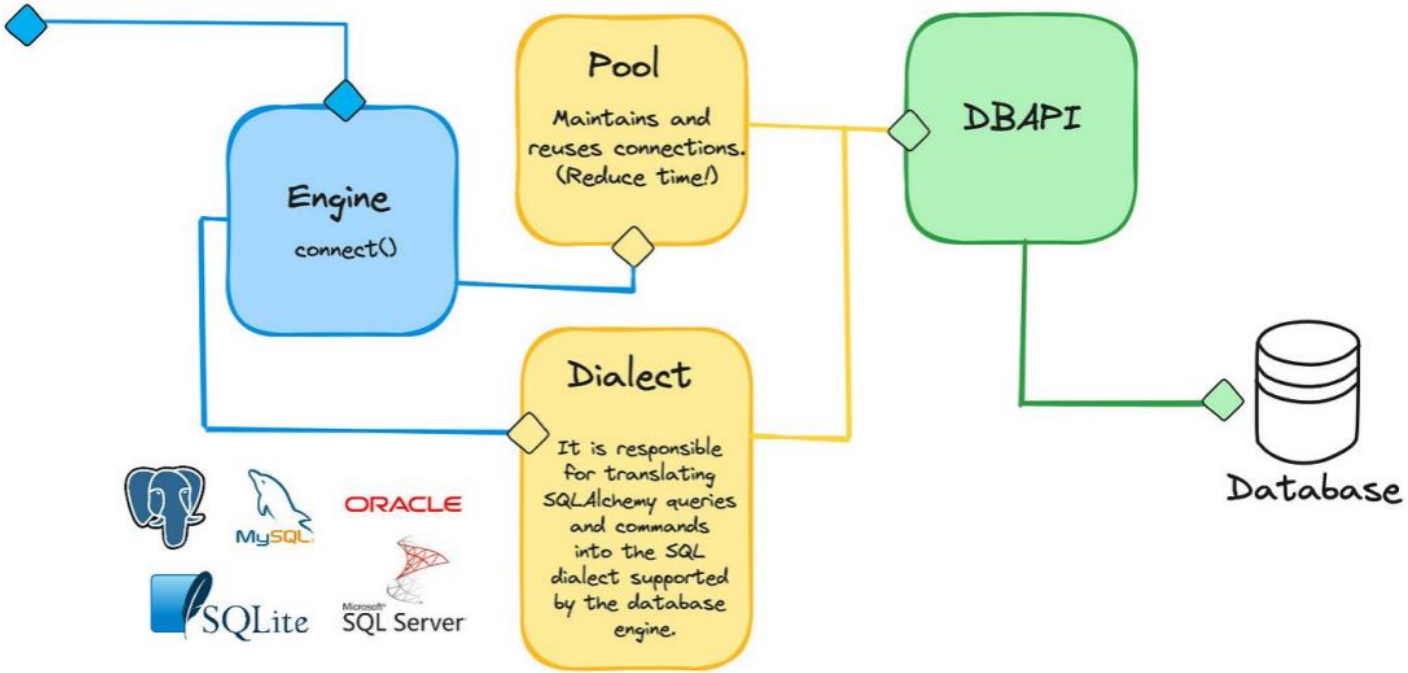
- Pool
  - Coleção de conexões que permite reutilizar conexões e melhorar o desempenho da consulta reduzindo o tempo.

Principais componentes

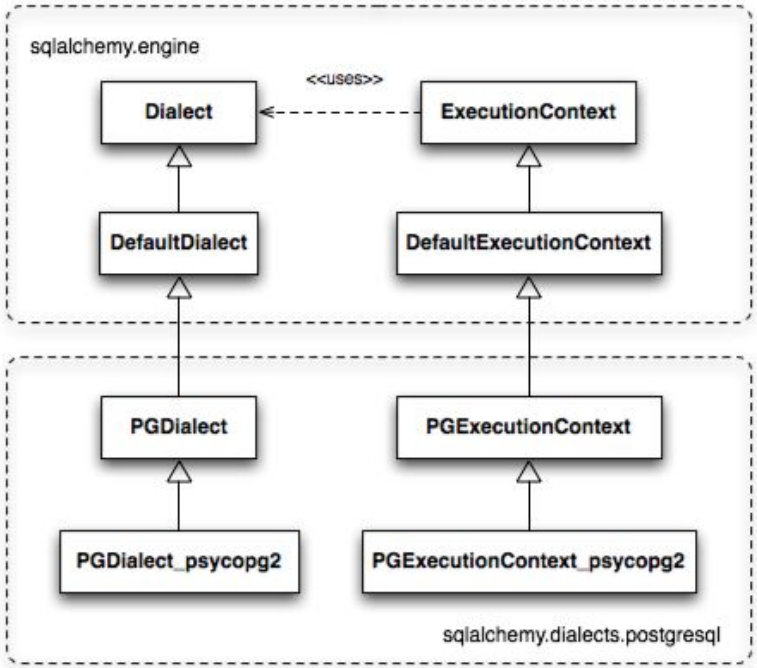
- Dialeto
  - Componente que permite a interação com o banco de dados.
  - Cada dialeto é projetado para interagir e traduzir consultas para um banco de dados.
  - Dialetos nativos: SQLite, MySQL, MariaDB, PostgreSQL, SQL Server e Oracle.
  - Existem também dialetos externos.

- DB-API
  - Interface que fornece métodos para permitir a comunicação entre o Python e o banco de dados.

SQLAlchemy


Pool
Maintains and reuses connections. (Reduce time!)
DBAPI
Engine
connect()
Dialect
It is responsible for translating SQLAlchemy queries and commands into the SQL dialect supported by the database engine.
Database
MySQL
ORACLE
SQLite
Microsoft SQL Server

Principais componentes

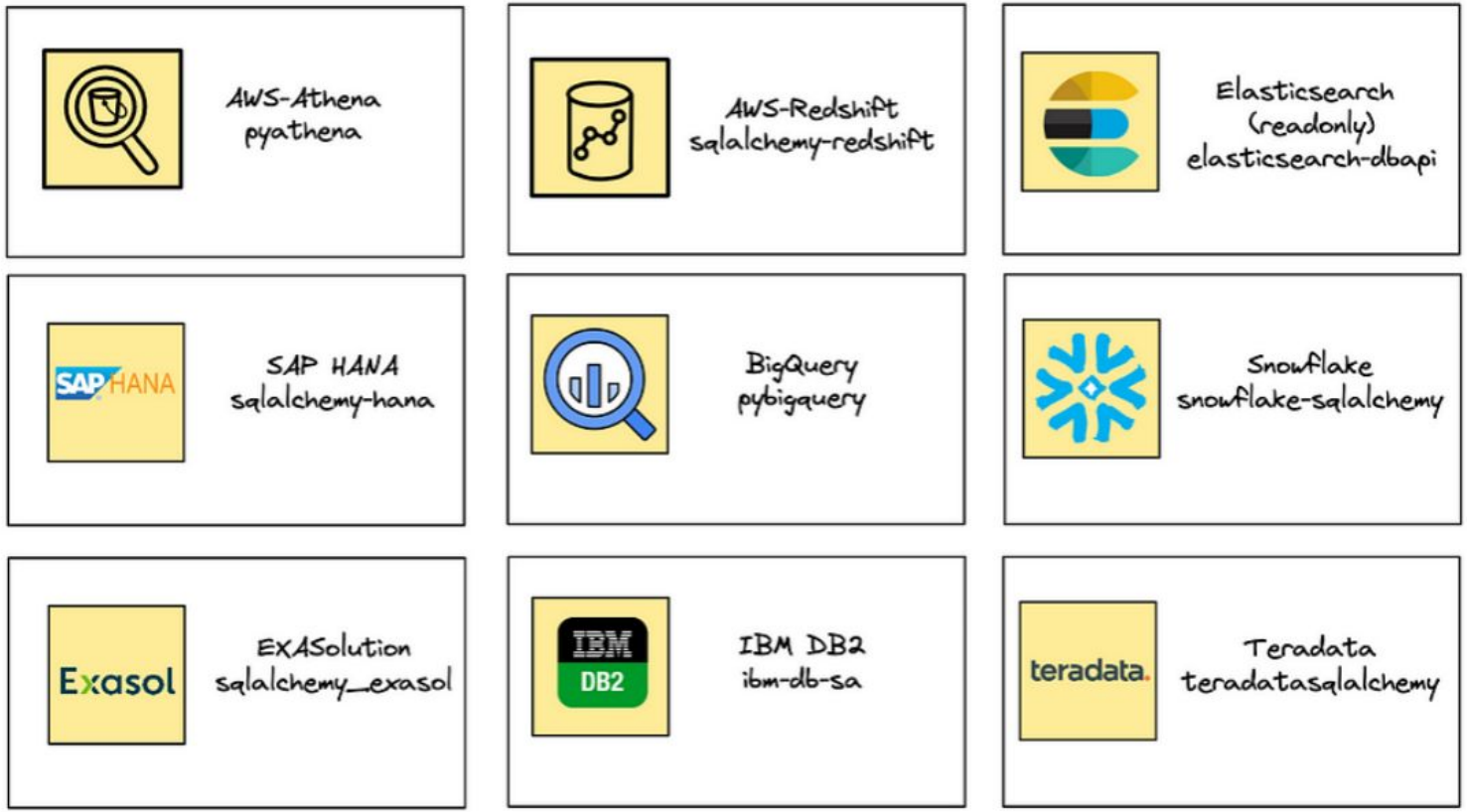


- O relacionamento entre Dialect e ExecutionContext quando usado com o dialeto psycopg2.
- A classe PGDialect fornece comportamentos que são específicos para o uso do banco de dados PostgreSQL, como o tipo de dado ARRAY e catálogos de esquema.
- A classe PGDialect_psycopg2 então fornece comportamentos específicos para o psycopg2 DBAPI, incluindo manipuladores de dados unicode e comportamento do cursor do lado do servidor.

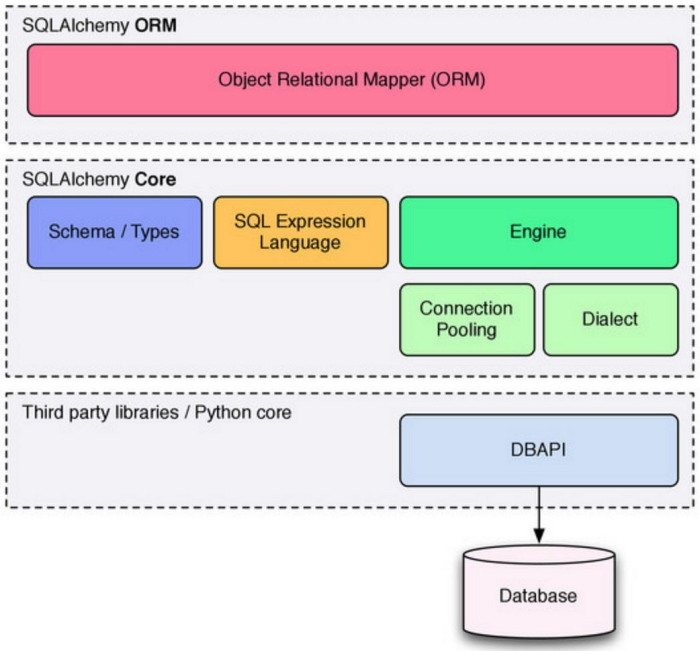
External dialects

https://docs.sqlalchemy.org/en/20/dialects/

There are 30 different dialects among the most popular are these



Arquitetura do SQLAlchemy



- O objetivo central do SQLAlchemy de fornecer uma abordagem de kit de ferramentas é que ele expõe cada camada de interação do banco de dados como uma API rica, dividindo a tarefa em duas categorias principais conhecidas como Core e ORM .

- O Core inclui interação com Python Database API (DBAPI), renderização de instruções SQL textuais entendidas pelo banco de dados e gerenciamento de esquema. Todos esses recursos são apresentados como APIs públicas.

- O ORM, ou mapeador objeto-relacional, é então uma biblioteca específica construída sobre o Core. O ORM fornecido com o SQLAlchemy é apenas uma de qualquer número de camadas possíveis de abstração de objetos que podem ser construídas sobre o Core, e muitos desenvolvedores e organizações constroem seus aplicativos sobre o Core diretamente.

Instalação

- *pip install sqlalchemy*
- Verificando versão instalada:

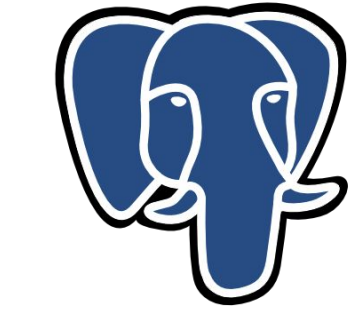
```
import sqlalchemy
sqlalchemy.__version__

ou

python -c "import sqlalchemy; print(sqlalchemy.__version__)"
```

Instalação

- PostgreSQL:
  - pip install psycopg2
- MySQL:
  - pip install pymysql
- MariaDB:
  - pip install mariadb
- SQLite
  - Já vem no sqlalchemy



Exemplo (models.py)

```python
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column
from sqlalchemy import String

# Classe base para o ORM
class Base(DeclarativeBase):
    pass

# Definição de uma tabela como classe
class Aluno(Base):
    __tablename__ = 'alunos'

    id: Mapped[int] = mapped_column(primary_key=True)
    nome: Mapped[str] = mapped_column(String(50))
    apelido: Mapped[str | None]  # Campo opcional

    # Método para representar os atributos do objeto
    def __repr__(self) -> str:
        return f"Aluno(id={self.id}, nome={self.nome}, apelido={self.apelido})"
```

- **Classe Base**: É a base do mapeamento ORM, necessária para o SQLAlchemy identificar que as classes derivadas representam tabelas no banco de dados.
- **Classe Aluno**:
  - Representa a tabela alunos no banco de dados.
  - Cada atributo corresponde a uma coluna na tabela.
  - O atributo apelido é opcional, representado por str | None.
- **Método __repr__**: Fornece uma representação legível da instância da classe, útil para depuração e exibição.

Exemplo (settings.py)

```
DATABASE_URL="sqlite:///exemplo-orm.db"
```

- Define a URL de conexão com um banco de dados SQLite chamado exemplo-orm.db. Essa URL é usada para configurar a conexão com o banco de dados ao criar um engine no SQLAlchemy.

Exemplo (database.py)

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import Session, sessionmaker
from models import Base
import settings

# Configuração do banco de dados
engine = create_engine(settings.DATABASE_URL)

# Criar a(s) tabela(s) no banco de dados
Base.metadata.create_all(engine)

def get_session() -> Session:
    Session = sessionmaker(bind=engine)
    return Session()
```

- Cria uma conexão com o banco de dados usando create_engine.
- Cria as tabelas definidas nos modelos ORM (Base.metadata.create_all).
- Define uma função get_session que retorna uma sessão para realizar operações no banco de dados.

Exemplo (exemplo_orm.py)

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Base, Aluno
from database import get_session

# Obter uma sessão para interagir com o banco de dados
session = get_session()

# Inserir novos registros
try:
    session.add(Aluno(nome='Maria', apelido='Mari'))
    session.add(Aluno(nome='João'))
    session.commit()
except Exception as e:
    session.rollback()
    print(f'Erro: {e}')

# Consultar registros
alunos = session.query(Aluno).all()
for aluno in alunos:
    print(aluno)
```

- **Obter Sessão**: Uma sessão é criada com get_session() para interagir com o banco.
- **Inserir Dados**: Adiciona dois registros na tabela Aluno e confirma a transação com session.commit(). Em caso de erro, realiza um rollback e exibe a mensagem de erro.
- **Consultar Dados**: Recupera todos os registros da tabela Aluno usando session.query(Aluno).all() e os exibe com um loop for.

Níveis de logging do Python

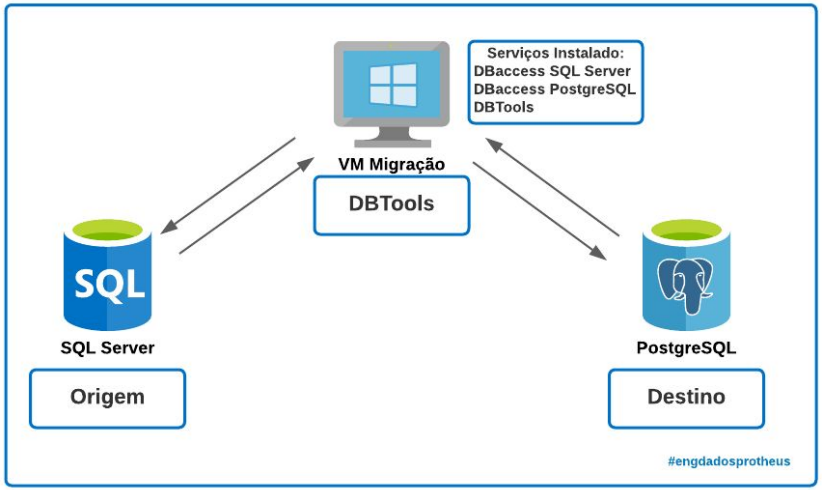
Nível	Valor Numérico	Descrição
DEBUG	10	Mensagens detalhadas para diagnóstico, geralmente usadas no desenvolvimento.
INFO	20	Informações gerais sobre o funcionamento do programa.
WARNING **(default)**	30	Indica que algo inesperado aconteceu ou pode causar problemas no futuro.
ERROR	40	Indica que ocorreu um erro, mas o programa continua executando.
CRITICAL	50	Um erro sério indicando que o programa pode não ser capaz de continuar

Logando consultas SQL

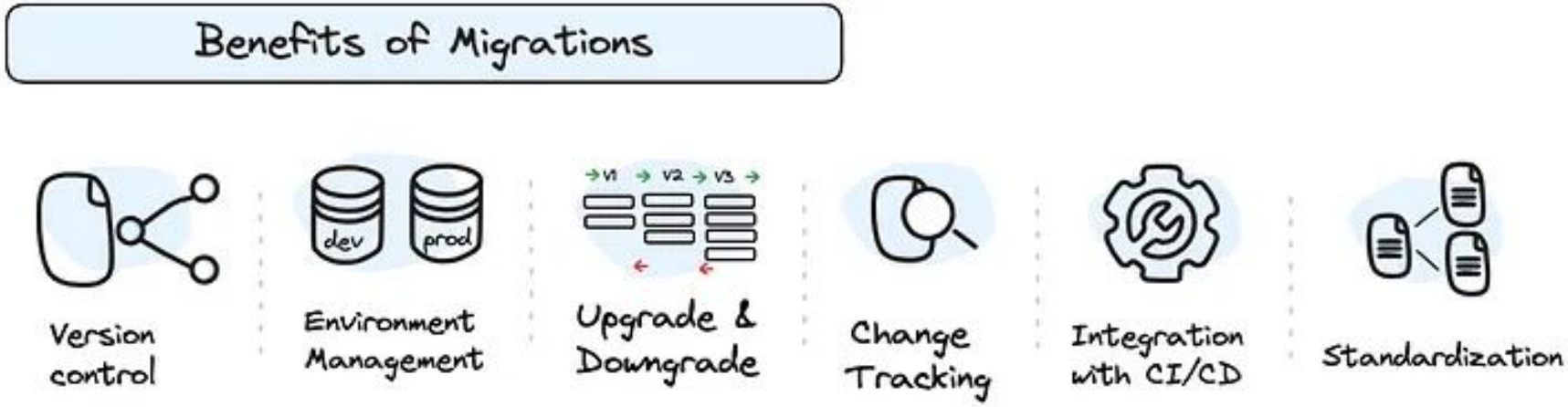
```
logging.basicConfig()
logging.getLogger("sqlalchemy.engine").setLevel(logging.INFO)
```

Migrações de banco de dados

- Processo que permite modificar a estrutura do banco de dados.
- Criadas para manter a consistência e a integridade



Quais são os benefícios de usar migrações?



Quais são os benefícios de usar migrações?

- Controle de versão : evita intervenção manual no banco de dados mantendo o controle sobre as versões do esquema.

- Gerenciamento de Ambientes: Facilita a criação de novos ambientes através da aplicação de migrações, permitindo fácil reprodução de configurações específicas e mantendo a coerência entre elas.

- Upgrade e Downgrade : Outro benefício é a capacidade não apenas de aplicar mudanças, mas também de revertê-las. Isso fornece flexibilidade e segurança no gerenciamento de banco de dados.

Quais são os benefícios de usar migrações?

- Auditoria : Alembic-audit é outra biblioteca que pode ser implementada para manter um registro cronológico das alterações feitas no banco de dados, facilitando a rastreabilidade.

- Integração CI/CD : Integra-se facilmente aos pipelines de CI/CD para aplicar alterações no banco de dados automaticamente, simplificando e garantindo consistência na implantação de aplicativos.

- Padronização : Esta implementação permite um desenvolvimento mais limpo, estruturado e coerente para definir e aplicar mudanças no esquema do banco de dados. Ao usar modelos, a reutilização de scripts é promovida, garantindo um gerenciamento eficiente e consistente das mudanças no banco de dados.

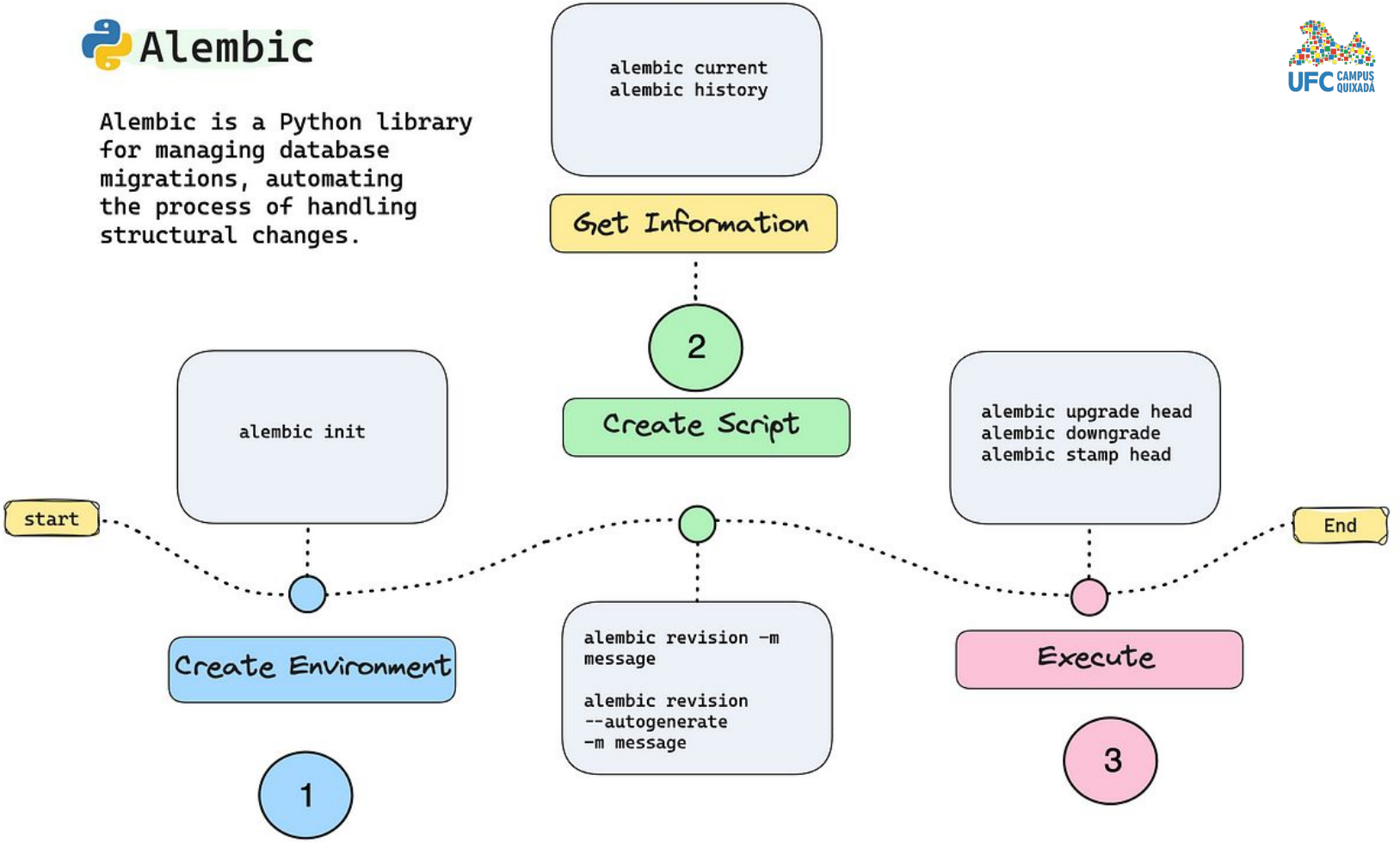
O que é Alembic?

- Alembic é uma biblioteca Python que permite migrações de banco de dados controladas e automatizadas.

- Esta biblioteca utiliza ` SQLAlchemy ` e permite o gerenciamento de alterações no esquema do banco de dados por meio de scripts, que descrevem as modificações e podem ser aplicadas automaticamente.

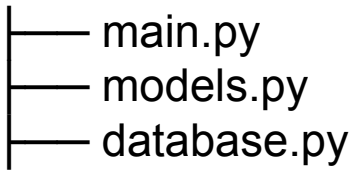
Alembic

Alembic is a Python library for managing database migrations, automating the process of handling structural changes.



Implementação com FastAPI + SQLAlchemy + SQLite

- O modelo que vamos implementar será para um sistema de gestão de alunos e cursos:
  - Aluno: id, nome, email.
  - Curso: id, nome, descrição.
  - Inscrição: id, aluno_id, curso_id
    - (representa uma relação muitos-para-muitos entre Aluno e Curso).



Implementação

- O SQLite não requer a instalação de um servidor separado, pois ele armazena os dados diretamente em um arquivo no disco. No caso, o arquivo escola.db criado pelo código SQLAlchemy será o banco de dados SQLite.

Classe Aluno

- __tablename__: Define o nome da tabela no banco como alunos.

- Atributos:
  - id: Chave primária, identificador único para cada aluno.
  - nome: Nome do aluno, obrigatório (nullable=False).
  - email: Endereço de email do aluno, obrigatório e único.

- Relacionamento:
  - Um aluno pode estar inscrito em vários cursos, representado pelo relacionamento com a tabela Inscricao usando relationship.

Classe Curso

- __tablename__: Define o nome da tabela no banco como cursos.

- Atributos:
  - id: Chave primária, identificador único para cada curso.
  - nome: Nome do curso, obrigatório (nullable=False).
  - descricao: Descrição opcional do curso.

- Relacionamento:
  - Um curso pode ter vários alunos inscritos, representado pelo relacionamento com a tabela Inscricao.

Classe Inscricao

- __tablename__: Define o nome da tabela no banco como inscricoes.

- Atributos:
  - id: Chave primária, identificador único para cada inscrição.
  - aluno_id: Chave estrangeira que referencia o id na tabela alunos.
  - curso_id: Chave estrangeira que referencia o id na tabela cursos.

- Relacionamentos:
  - aluno: Relacionamento que conecta esta inscrição ao respectivo aluno.
  - curso: Relacionamento que conecta esta inscrição ao respectivo curso.


LET'S
CODE

Referências

- SQLAlchemy Documentation — SQLAlchemy 2.0 Documentation
- Glossary — SQLAlchemy 2.0 Documentation
- Simplify Database Migrations using Python with Alembic | by Romina Mendez | Medium

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

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