

# Mapeamento Objeto-Relacional: MySQL e Implementação com ORM SQLAlchemy e SQLModel

QXD0099 - Desenvolvimento de Software para Persistência

**Universidade Federal do Ceará - *Campus* Quixadá**

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



# Agenda

- MySQL
- Fundamentos do SQLAlchemy
- Por que usar SQLAlchemy e SQLAlchemyModel
- Conexão com MySQL
- Configuração do Ambiente
- Criando Modelos com SQLAlchemyModel
- CRUD com SQLAlchemy e SQLAlchemyModel
- Exemplo

# MySQL

- O MySQL é um sistema de gerenciamento de banco de dados relacional amplamente utilizado por sua facilidade de uso, confiabilidade e desempenho.



```
mysql> ALTER TABLE minttec ADD last_name VARCHAR(20) AFTER first_name;
Query OK, 0 rows affected (0.47 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> show columns from minttec;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int  | YES  |     | NULL    |       |
| first_name | varchar(15) | YES  |     | NULL    |       |
| last_name | varchar(20) | YES  |     | NULL    |       |
| email | varchar(20) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

# Por que MySQL?

- **Facilidade de Uso:** O MySQL é conhecido por sua facilidade de uso, com uma sintaxe SQL intuitiva e uma interface amigável que permite aos usuários iniciantes começarem rapidamente.
- **Confiabilidade:** Assim como o PostgreSQL, o MySQL é altamente confiável e tolerante a falhas, garantindo que seus dados estejam seguros e acessíveis mesmo em situações adversas.

# Por que MySQL?

- **Escalabilidade:** O MySQL oferece suporte a escalabilidade vertical e horizontal, permitindo que você dimensione seu banco de dados de acordo com as necessidades do seu aplicativo à medida que ele cresce.
- **Comunidade Ativa:** Assim como o PostgreSQL, o MySQL possui uma comunidade ativa de desenvolvedores e usuários que contribuem para seu desenvolvimento contínuo, fornecendo suporte e recursos adicionais.

# Instalando e Configurando o MySQL

- O MySQL está disponível para uma variedade de sistemas operacionais e pode ser baixado gratuitamente em <https://dev.mysql.com/downloads/>.
- Após a instalação, você pode configurar e iniciar o MySQL no seu sistema. Durante o processo de instalação, você será solicitado a definir uma senha para o usuário administrador (root).

# Criando um Banco de Dados

- No MySQL, um banco de dados é um contêiner para armazenar e organizar dados. Para criar um novo banco de dados, você pode usar o utilitário de linha de comando `mysql` ou uma interface gráfica como o MySQL Workbench.
- Para criar um novo banco de dados chamado “meubanco”, você pode executar o seguinte comando:

```
CREATE DATABASE meubanco;
```

# Criando Tabelas e Inserindo Dados

- No MySQL, assim como em outros bancos de dados relacionais, você define a estrutura dos dados usando tabelas. Aqui está um exemplo de criação de uma tabela chamada “usuarios” com três colunas: id, nome e idade:

```
CREATE TABLE usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255),
  idade INT
);
```

- Agora que a tabela está criada, você pode inserir dados nela. Por exemplo, para adicionar um usuário à tabela “usuarios,” você pode executar a seguinte instrução SQL:

```
INSERT INTO usuarios (nome, idade) VALUES ('Hugo', 30);
```



# SQLModel

- Biblioteca que combina e simplifica recursos do Pydantic e do SQLAlchemy para criar modelos de dados que podem ser usados tanto para validação quanto para interagir com bancos de dados.
- Projetada para ser intuitiva, especialmente em aplicações implementadas com FastAPI.
- Instalação no projeto:
  - `pip install sqlmodel`



# Fundamentos do SQLAlchemy

- Diferença entre SQLAlchemy e SQLModel:
  - SQLModel simplifica o uso de SQLAlchemy ao integrar tipagem Pydantic.
  - Tabelas e modelos de validação compartilhados.

 **SQLModel**



**Pydantic**

**SQLAlchemy**

# Fundamentos do SQLAlchemy

- Conceitos básicos:
  - SQLAlchemy para definir modelos que podem ser tabelas ou esquemas de dados.
  - Field para configurar propriedades das colunas.
  - Configuração de relações.



*SQL databases in Python,  
designed for simplicity, compatibility, and robustness.*

# Conexão com MySQL

- Instalação do driver MySQL:
  - `pip install mysql-connector-python`
- Configurar o URL de conexão

```
mysql+mysqlconnector://username:password@host/database
```

# Configuração do Ambiente

- Instalar dependências
  - `pip install sqlalchemy sqlmodel mysql-connector-python`
- Criar um banco de dados `school_db`.
- Criar um usuário com permissões

```
CREATE DATABASE school_db;
CREATE USER 'student'@'localhost' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON school_db.* TO 'student'@'localhost';
```

# Criando Modelos com SQLAlchemy

```
from sqlalchemy import SQLAlchemy, Field, create_engine

class Student(SQLAlchemy, table=True):
    id: int = Field(default=None, primary_key=True)
    name: str
    age: int
    grade: str

database_url =
"mysql+mysqlconnector://student:password@localhost/school
_db"
engine = create_engine(database_url, echo=True)

# Criar tabelas no banco
SQLAlchemy.metadata.create_all(engine)
```

- **SQLAlchemy:** Define o modelo Student mapeado para a tabela.
- **Field:** Define propriedades da tabela.
- **create\_engine:** Configura a conexão com o MySQL.
- **metadata.create\_all:** Cria as tabelas no banco.

# CRUD com SQLAlchemy e SQLModel - Inserir Dados

```
from sqlmodel import Session

with Session(engine) as session:
    student = Student(name="Alice",
                       age=20, grade="A")
    session.add(student)
    session.commit()
    session.refresh(student)
    print(student)
```

- **Session:** A sessão é responsável por interagir com o banco de dados. É o meio para realizar operações como inserir, consultar, atualizar e deletar registros.
- **Criação do objeto student:** Um novo objeto da classe Student é instanciado com valores:
  - name="Alice", age=20, e grade="A".
- **session.add(student):** Adiciona o objeto student ao contexto da sessão. Ele ainda não está persistido no banco de dados até que o commit seja executado.
- **session.commit():** Aplica as mudanças no banco de dados. Neste momento, a transação é finalizada, e o registro é salvo no banco.
- **session.refresh(student):** Atualiza o objeto student com os dados mais recentes do banco de dados, incluindo qualquer valor gerado automaticamente, como um ID.
- **print(student):** Exibe os dados do objeto student, agora com as informações persistidas.

# CRUD com SQLAlchemy e SQLModel - Consultar Dados

```
with Session(engine) as session:
    students =
    session.query(Student).all()
    for student in students:
        print(student)
```

- **session.query(Student):** Realiza uma consulta na tabela associada à classe Student. Ele cria uma query que busca todos os registros dessa tabela.
- **all():** Executa a query e retorna todos os registros encontrados como uma lista de objetos.
- **for student in students:** Itera sobre a lista de objetos retornada pela query.
- **print(student):** Exibe cada objeto student, mostrando os dados de cada aluno encontrado no banco.



# CRUD com SQLAlchemy e SQLModel - Atualizar Dados

```
with Session(engine) as session:
    student = session.get(Student, 1)
    if student:
        student.grade = "A+"
        session.add(student)
        session.commit()
```

- **session.get(Student, 1):** Busca um registro específico da tabela Student com base na chave primária (neste caso, o valor 1).
  - Se o registro existe, ele é carregado no objeto student.
- **Verificação if student:** Garante que o objeto student foi encontrado antes de prosseguir com a atualização.
- **student.grade = "A+":** Modifica o campo grade do objeto student.
- **session.add(student):** Adiciona o objeto atualizado ao contexto da sessão. Isso indica ao SQLAlchemy que o objeto sofreu alterações e precisa ser persistido no banco.
- **session.commit():** Aplica as alterações no banco de dados.

# CRUD com SQLAlchemy e SQLModel - Deletar Dados

```
with Session(engine) as session:
    student = session.get(Student, 1)
    if student:
        session.delete(student)
        session.commit()
```

- **session.get(Student, 1):** Busca o registro da tabela Student com chave primária igual a 1.
- **Verificação if student:** Confirma que o objeto student foi encontrado.
- **session.delete(student):** Marca o objeto student para ser removido do banco de dados.
- **session.commit():** Aplica a remoção no banco de dados. O registro é deletado permanentemente.

# Resumo das Operações CRUD

- Create (Inserir) → `session.add()` seguido de `session.commit()` para salvar o registro.
- Read (Consultar) → `session.query()` para buscar registros, usando `.all()` ou métodos como `filter`.
- Update (Atualizar) → Buscar o registro com `session.get()`, modificar os valores e aplicar com `session.commit()`.
- Delete (Deletar) → Buscar o registro com `session.get()`, usar `session.delete()` e confirmar com `session.commit()`.

# Resumo das Operações CRUD

- O SQLAlchemy e SQLModel trabalham de forma integrada para gerenciar os dados com um banco relacional, proporcionando uma interface mais fácil e orientada a objetos.

# models.py

```
from sqlmodel import SQLModel, Field

class Aluno(SQLModel, table=True):
    __tablename__ = 'alunos'
    id: int | None = Field(default=None, primary_key=True)
    nome: str
    apelido: str | None = Field(default=None)
```

# .env

- Arquivo .env
- `pip install python-dotenv`

```
DATABASE_URL=mysql+mysqlconnector://student:password@localhost/school_db
```

# database.py

```
from sqlalchemy import create_engine, Session, SQLModel
from dotenv import load_dotenv
import logging
import os

# Carregar variáveis do arquivo .env
load_dotenv("db.env")

# Configurar o logger
logging.basicConfig()
logging.getLogger("sqlalchemy.engine").setLevel(logging.INFO)

# Configuração do banco de dados
engine = create_engine(os.getenv("DATABASE_URL"))

# Criar a(s) tabela(s) no banco de dados
# Inicializa o banco de dados
def create_db_and_tables() -> None:
    SQLModel.metadata.create_all(engine)

def get_session() -> Session:
    return Session(engine)
```

# main.py

```
from models import Aluno
from sqlmodel import select
from database import get_session, create_db_and_tables

create_db_and_tables()

# Obter uma sessão para interagir com o banco de dados
with get_session() as session:
    # Inserir novos registros
    try:
        session.add(Aluno(nome='Maria', apelido='Mari'))
        session.add(Aluno(nome='João'))
        # Consultar registros
        alunos = session.exec(select(Aluno)).all()
        for aluno in alunos:
            print(aluno)
        session.commit()
    except Exception as e:
        session.rollback()
        print(f'Erro: {e}')
```



## Adicionando o FastAPI

# main2.py (1/2)

```
@app.get("/")
def home():
    return {"msg": "Olá, mundo!"}

@app.post("/alunos", response_model=Aluno)
def inserir_aluno(aluno: Aluno, session: Session = Depends(get_session)) -> Aluno:
    session.add(aluno)
    session.commit()
    session.refresh(aluno)
    return aluno

@app.get("/alunos", response_model=list[Aluno])
def listar_alunos(session: Session = Depends(get_session)) -> list[Aluno]:
    alunos = session.exec(select(Aluno)).all()
    return alunos
```

# Implementação

- **Projeto:** Representa um projeto, com dados básicos como nome e descrição.
- **Equipe:** Representa uma equipe que gerencia os projetos.
- **Membro:** Representa os membros de uma equipe.
- **Tarefa:** Representa as tarefas relacionadas a um projeto.
- **Participação (Membership):** Relação muitos-para-muitos entre Membro e Projeto.

# Relacionamentos

- **Relacionamento 1:n entre Equipe e Projeto:**
  - Uma equipe pode ter vários projetos.
  - Representado pelo campo `equipe_id` na entidade Projeto e pela lista projetos na entidade Equipe.
- **Relacionamento n:m entre Membro e Projeto:**
  - Um membro pode participar de vários projetos, e um projeto pode ter vários membros.
  - Implementado por meio da tabela associativa Membership

# Relacionamentos

- **Relacionamento 1:n entre Projeto e Tarefa:**
  - Um projeto pode ter várias tarefas associadas.
  - Representado pelo campo projeto\_id na entidade Tarefa e pela lista tarefas na entidade Projeto.



# 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/>
- SQLAlchemyModel - <https://sqlmodel.tiangolo.com>



# Obrigado! Dúvidas?



**Universidade Federal do Ceará - *Campus* Quixadá**

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

