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

Continuação…

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

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

Agenda

- Implementação
  - projeto
  - equipe
  - membro
  - tarefa
- Relacionamentos
- back_populates em detalhes
- Código

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.

back_populates em detalhes

- back_populates: Esse parâmetro cria um vínculo bidirecional entre duas classes, permitindo acessar dados relacionados em ambas as direções do relacionamento.

- O valor atribuído a back_populates é o nome do atributo na outra classe que completa o vínculo.

Quando usar o plural ou o singular

- A escolha entre plural e singular depende da natureza do relacionamento:

- **Relacionamento 1:N (Um para Muitos):**
  - Classe que possui o "um":
    - O atributo correspondente geralmente é singular.
    - Exemplo: equipe na classe Projeto (um projeto pertence a uma única equipe).

  - Classe que possui o "muitos":
    - O atributo correspondente geralmente é plural.
    - Exemplo: projetos na classe Equipe (uma equipe pode ter vários projetos).

Quando usar o plural ou o singular

- **Relacionamento N:1 (Muitos para Um):**
  - **Classe que possui o "muitos":**
    - O atributo correspondente é singular.
    - Exemplo: equipe na classe Membro (um membro pertence a uma única equipe).

- **Classe que possui o "um":**
    - O atributo correspondente é plural.
    - Exemplo: membros na classe Equipe (uma equipe pode ter vários membros).

Exemplo aplicado

projetos: List["Projeto"] = Relationship(back_populates="equipe")

- **Contexto**:
  - Está na classe Equipe.
  - Indica que uma equipe pode estar associada a vários projetos.

- **Plural**:
  - O atributo projetos está no plural porque uma equipe pode ter vários projetos (relação 1:N).

- **back_populates="equipe"**:
  - Refere-se ao atributo equipe na classe Projeto, que é singular, porque cada projeto pertence a uma única equipe.

Exemplo aplicado

equipe: Optional[Equipe] = Relationship(back_populates="membros")

- **Contexto**:
  - Está na classe Membro.
  - Indica que um membro pertence a uma única equipe.

- **Singular**:
  - O atributo equipe está no singular porque um membro só pode estar associado a uma única equipe (relação N:1).

- **back_populates="membros"**:
  - Refere-se ao atributo membros na classe Equipe, que é plural, porque uma equipe pode ter vários membros.

Exemplo aplicado

- **Regra geral para plural ou singular com back_populates:**
  - Use o plural para representar uma coleção de objetos relacionados (lado do "muitos").
  - Use o singular para representar uma única relação (lado do "um").
- **Exemplo**
  - Plural (List[...]): Quando o relacionamento indica múltiplas entidades (lado do "muitos").
  - Singular (Optional[...]): Quando o relacionamento indica uma única entidade (lado do "um").

Classe Equipe

```python
# Definição da classe `Equipe`
class Equipe(SQLModel, table=True):
    """
    Representa uma equipe no sistema. Cada equipe pode ter múltiplos membros e projetos associados.
    """
    id: int = Field(default=None, primary_key=True)  # Campo `id` é a chave primária da tabela.
    nome: str  # Nome da equipe (obrigatório).
    descricao: Optional[str] = None  # Descrição da equipe (opcional).
    membros: List["Membro"] = Relationship(back_populates="equipe")
    # Relacionamento 1:N com `Membro`. Cada equipe pode ter vários membros.
    projetos: List["Projeto"] = Relationship(back_populates="equipe")
    # Relacionamento 1:N com `Projeto`. Cada equipe pode ter vários projetos.
```

Atributos:

- **id**:
  - Tipo: int
  - É uma chave primária (primary_key=True).
  - Identifica unicamente cada equipe.
- **nome**:
  - Tipo: str
  - Campo obrigatório que armazena o nome da equipe.
- **descricao**:
  - Tipo: Optional[str]
  - Campo opcional que pode conter uma descrição da equipe.
- **membros**:
  - Relacionamento 1:N com a classe Membro:
    - Uma equipe pode ter vários membros.
    - O back_populates="equipe" permite a sincronização bidirecional com o atributo equipe da classe Membro.
- **projetos**:
  - Relacionamento 1:N com a classe Projeto:
    - Uma equipe pode ter vários projetos.
    - O back_populates="equipe" sincroniza com o atributo equipe da classe Projeto.

Classe Membro

```python
# Definição da classe `Membro`
class Membro(SQLModel, table=True):
    """
    Representa um membro de uma equipe. Cada membro pertence a uma única equipe.
    """
    id: int = Field(default=None, primary_key=True)  # Campo `id` é a chave primária da tabela.
    nome: str  # Nome do membro (obrigatório).
    equipe_id: int = Field(foreign_key="equipe.id")
    # Campo que armazena a chave estrangeira da equipe à qual o membro pertence.
    equipe: Optional[Equipe] = Relationship(back_populates="membros")
    # Relacionamento N:1 com `Equipe`. Cada membro pertence a uma equipe.
```

Atributos:

- **id**:
  - Tipo: int
  - É uma chave primária (primary_key=True).
  - Identifica unicamente cada membro.
- **nome**:
  - Tipo: str
  - Campo obrigatório que armazena o nome do membro.
- **equipe_id**:
  - Tipo: int
  - Representa uma **chave estrangeira** (foreign_key="equipe.id") que liga o membro à sua equipe.
- **equipe**:
  - Relacionamento N:1 com a classe Equipe:
    - Cada membro pertence a uma única equipe.
    - Sincroniza com o atributo membros da classe Equipe.

Classe Projeto

```python
# Definição da classe `Projeto`
class Projeto(SQLModel, table=True):
    """
    Representa um projeto de uma equipe. Cada projeto pertence a uma única equipe.
    """
    id: int = Field(default=None, primary_key=True)  # Campo `id` é a chave primária da tabela.
    nome: str  # Nome do projeto (obrigatório).
    descricao: Optional[str] = None  # Descrição do projeto (opcional).
    equipe_id: int = Field(foreign_key="equipe.id")
    # Campo que armazena a chave estrangeira da equipe à qual o projeto pertence.

    equipe: Optional[Equipe] = Relationship(back_populates="projetos")
    # Relacionamento N:1 com `Equipe`. Cada projeto pertence a uma equipe.

    tarefas: List["Tarefa"] = Relationship(back_populates="projeto")
    # Relacionamento 1:N com `Tarefa`. Cada projeto pode ter várias tarefas.
```

Atributos:

- **id**:
  - Tipo: int
  - É uma chave primária.
- **nome**:
  - Tipo: str
  - Campo obrigatório que armazena o nome do projeto.
- **descricao**:
  - Tipo: Optional[str]
  - Campo opcional que descreve o projeto.
- **equipe_id**:
  - Tipo: int
  - Representa uma chave estrangeira para a equipe proprietária do projeto.
- **equipe**:
  - Relacionamento N:1 com Equipe:
    - Cada projeto pertence a uma equipe.
- **tarefas**:
  - Relacionamento 1:N com Tarefa:
    - Um projeto pode ter várias tarefas.

Classe Tarefa

```python
# Definição da classe `Tarefa`
class Tarefa(SQLModel, table=True):
    """
    Representa uma tarefa dentro de um projeto. Cada tarefa pertence a um único projeto.
    """
    id: int = Field(default=None, primary_key=True)  # Campo `id` é a chave primária da tabela.
    descricao: str  # Descrição da tarefa (obrigatória).
    projeto_id: int = Field(foreign_key="projeto.id")
    # Campo que armazena a chave estrangeira do projeto ao qual a tarefa pertence.
    projeto: Optional[Projeto] = Relationship(back_populates="tarefas")
    # Relacionamento N:1 com `Projeto`. Cada tarefa pertence a um projeto.
```

Atributos:

- **id**:
  - Tipo: int
  - É uma chave primária.
- **descricao**:
  - Tipo: str
  - Campo obrigatório que detalha a tarefa.
- **projeto_id**:
  - Tipo: int
  - Chave estrangeira que liga a tarefa ao projeto correspondente.
- **projeto**:
  - Relacionamento N:1 com a classe Projeto:
    - Cada tarefa pertence a um único projeto.

Classe Membership

```python
# Definição da classe `Membership`
class Membership(SQLModel, table=True):
    """
    Representa um relacionamento entre membro e equipe.
    Essa classe é usada para criar um vínculo entre membro e equipe em uma relação N:M.
    """
    id: int = Field(default=None, primary_key=True)  # Campo `id` é a chave primária da tabela.
    membro_id: int = Field(foreign_key="membro.id")
    # Campo que armazena a chave estrangeira do membro associado.
    equipe_id: int = Field(foreign_key="equipe.id")
    # Campo que armazena a chave estrangeira da equipe associada.
```

Atributos:

- **id**:
  - Tipo: int
  - É uma chave primária.
- **membro_id**:
  - Tipo: int
  - Chave estrangeira que referencia o membro associado.
- **equipe_id**:
  - Tipo: int
  - Chave estrangeira que referencia a equipe associada.

Relacionamentos em Resumo

- **Equipe e Membro:**
  - Relacionamento 1:N:
  - Uma equipe pode ter vários membros.
  - Cada membro pertence a uma única equipe.

- **Equipe e Projeto:**
  - Relacionamento 1:N:
  - Uma equipe pode ter vários projetos.
  - Cada projeto pertence a uma única equipe.

Relacionamentos em Resumo

- **Projeto e Tarefa:**
  - Relacionamento 1:N:
  - Um projeto pode ter várias tarefas.
  - Cada tarefa pertence a um único projeto.

- **Equipe e Membro via Membership:**
  - Relacionamento N:M (usando uma tabela associativa):
  - Um membro pode participar de várias equipes.
  - Uma equipe pode incluir vários membros.


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

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

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