Esercizio FastAPI & SQLAlchemy

Modellazione Relazionale Avanzata

Corso FastAPI per Principianti

Descrizione dello Scenario

Si vuole realizzare il backend per un sistema di gestione progetti di una Software House. Il sistema deve gestire i dipendenti, i progetti a cui lavorano, i task specifici di ogni progetto e le tecnologie utilizzate.

L'obiettivo è implementare l'API utilizzando **FastAPI**, **SQLAlchemy** (con database MySQL o SQLite) e **Pydantic**.

1 Specifiche del Database (Data Modeling)

Il database deve essere composto da 5 entità (tabelle). Di seguito sono descritte le tabelle e le relazioni richieste.

1. Tabelle Principali

- **Team**: Rappresenta un gruppo di lavoro (es. "Backend", "Frontend", "DevOps").
  - `id`: Integer, Primary Key.
  - `name`: String (unico).
  - `location`: String (es. "Ufficio A", "Remoto").
- **Developer** (Sviluppatore):
  - `id`: Integer, Primary Key.
  - `nickname`: String (unico).
  - `email`: String.
  - **Relazione 1**: Ogni sviluppatore appartiene a **un solo** Team.
- **Project** (Progetto):
  - `id`: Integer, Primary Key.
  - `name`: String (es. "Sito E-Commerce").
  - `deadline`: String (o Date).
  - **Relazione 2**: Ogni progetto ha un **Tech Lead** (che è un Developer). Questo è un collegamento *Uno-a-Molti* (Un Developer può essere lead di più progetti).
- **Task** (Compito):
  - `id`: Integer, Primary Key.
  - `description`: String.
  - `is_completed`: Boolean (default False).

  - **Relazione 3**: Ogni task appartiene a **un solo** Project.

- **Technology** (Tecnologia):
  - `id`: Integer, Primary Key.
  - `name`: String (es. "Python", "React", "Docker").
  - **Relazione 4 (Molti-a-Molti)**: Un Project usa molte Technology, e una Technology è usata in molti Project.

2 Schema delle Relazioni

Riepilogo Relazioni

1. **Team** ↔ **Developer**: Uno a Molti (1 Team ha N Developers).
2. **Developer** ↔ **Project**: Uno a Molti (1 Developer "Tech Lead" gestisce N Projects).
3. **Project** ↔ **Task**: Uno a Molti (1 Project ha N Tasks).
4. **Project** ↔ **Technology**: **Molti a Molti** (Richiede Tabella di Associazione).

3 Requisiti dell'Esercizio

3.1 1. Configurazione e Modelli (`database.py`, `models.py`)

- Configura la connessione al database.
- Definisci le classi SQLAlchemy per tutte le entità sopra descritte.
- Implementa correttamente le `ForeignKey` e le `relationship`.
- **Attenzione:** Per la relazione Project-Technology, dovrai creare la tabella di associazione (es. `project_technologies`).

3.2 2. Schemi Pydantic (`schemas.py`)

Crea gli schemi Pydantic per la validazione dei dati. Assicurati di gestire le relazioni annidate:

- Quando richiedo un **Project**, voglio vedere:
  - Il nome del Tech Lead.
  - La lista dei **Task** associati.
  - La lista delle **Technology** usate (oggetti completi, non solo ID).
- Quando creo un **Project** (Input), devo poter passare una lista di ID per le tecnologie (es. `technology_ids: [1, 3]`).

3.3 3. Logica CRUD (`crud.py`)

Implementa le seguenti funzioni:

- `create_team`, `create_developer`, `create_technology`.
- `create_project`: Deve gestire l'inserimento della relazione Molti-a-Molti ricevendo una lista di ID tecnologie.
- `get_project`: Deve restituire il progetto con tutte le relazioni caricate (Join).

3.4 4. API Endpoints (`main.py`)

Esponi le seguenti rotte:

- `POST /teams/`, `POST /developers/`, `POST /technologies/`
- `POST /projects/`: Accetta il JSON per creare un progetto.
- `POST /projects/{id}/tasks/`: Aggiunge un task a un progetto esistente.
- `GET /projects/`: Restituisce la lista dei progetti con tutti i dettagli annidati.

Suggerimento per la tabella di associazione

Ricorda che la tabella "ponte" per la relazione molti-a-molti non è solitamente una classe Python, ma un oggetto `Table`:

```
project_technologies = Table(
    "project_technologies",
    Base.metadata,
    Column("project_id", Integer, ForeignKey("projects.id")),
    Column("tech_id", Integer, ForeignKey("technologies.id"))
)
```