Créer des tables avec SQLModel

Un modèle SQLModel représente une **structure de table** dans une base de données relationnelle. Comme en SQL, on va pouvoir les relier avec des clés étrangères grâce à des relations **one-to-many** ou **many-to-many**.

Les relations entre modèles permettent de structurer la base et de faciliter les requêtes complexes.

Sommaire

Définir un modèle simple

```python
from sqlmodel import SQLModel, Field
from typing import Optional

class User(SQLModel, table=True):
    id: int | None = Field(default=None, primary_key=True)
    username: str
    email: str
    age: int | None = None
```

Un modèle simple permet de décrire les colonnes d'une table comme si c'était une classe Python, en utilisant le typage natif. Chaque champ est directement converti en colonne SQL, avec son type, ses contraintes, et ses valeurs par défaut.

Détails ligne par ligne

> `class User(SQLModel, table=True)`

- `SQLModel` est la classe de base à hériter.
- `table=True` signifie que cette classe correspond à une table dans la base.
- Si `table=True` est absent, le modèle ne sera **pas utilisé pour générer une table** (utile plus tard pour les schémas de validation côté API).

> `id: int | None = Field(default=None, primary_key=True)`

- `id` est défini comme **clé primaire**, et peut être `None` avant insertion (il sera auto-généré par la base).
- `Field(...)` permet de spécifier des options supplémentaires (valeur par défaut, contraintes, etc.).

> `username: str` **et** `email: str`

- Champs obligatoires : aucune valeur par défaut, donc exigés à la création.
- Ce sont des **colonnes typées** dans la table.

> `age: int | None = None`

- Champ **optionnel** : il peut être vide (null en base).
- Typé avec `int | None` ou `Optional[int]` : deux syntaxes équivalentes.

Définir un modèle one-to-many

La relation **one-to-many** est utilisée lorsqu'un élément (ex : une équipe) peut être lié à plusieurs autres éléments (ex : des héros), mais chaque élément "enfant" ne peut être rattaché qu'à **un seul parent**.

Exemple : une équipe avec plusieurs héros

```
from typing import Optional, List
from sqlmodel import SQLModel, Field, Relationship

class Team(SQLModel, table=True):
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str
    heroes: List["Hero"] = Relationship(back_populates="team")

class Hero(SQLModel, table=True):
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str
    team_id: Optional[int] = Field(default=None, foreign_key="team.id")
    team: Optional[Team] = Relationship(back_populates="heroes")
```

Explications clés

- Le champ `team_id` dans `Hero` est une **clé étrangère** vers `Team`.
- Le champ `heroes` dans `Team` est une **liste d'objets Hero** automatiquement remplie.
- `Relationship(back_populates=...)` permet d'établir la **navigation bidirectionnelle** entre les deux classes.
- Cette relation est pratique pour interroger tous les membres d'une équipe, ou connaître l'équipe d'un héros.

Définir un modèle many-to-many

En SQL, les relations **many-to-many** (plusieurs à plusieurs) **n'existent pas directement**. Elles sont toujours implémentées via une **table de liaison** (intermédiaire), contenant deux clés étrangères.

Par exemple : un héros peut appartenir à **plusieurs équipes**, et une équipe peut contenir **plusieurs héros**.

Exemple : des héros dans plusieurs équipes

```
from sqlmodel import SQLModel, Field, Relationship
from typing import Optional, List

class HeroTeamLink(SQLModel, table=True):
    hero_id: Optional[int] = Field(default=None, foreign_key="hero.id",
primary_key=True)
    team_id: Optional[int] = Field(default=None, foreign_key="team.id",
primary_key=True)

class Team(SQLModel, table=True):
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str
    heroes: List["Hero"] = Relationship(back_populates="teams",
link_model=HeroTeamLink)

class Hero(SQLModel, table=True):
    id: Optional[int] = Field(default=None, primary_key=True)
    name: str
    teams: List[Team] = Relationship(back_populates="heroes",
link_model=HeroTeamLink)
```

Explications clés

- `HeroTeamLink` est la **table de liaison** entre les entités `Hero` et `Team`.
- Elle contient **deux clés étrangères**, chacune définie aussi comme **clé primaire**.
- Cette table intermédiaire permet d'ajouter ou supprimer facilement des relations sans affecter les entités principales.
- `link_model=...` est nécessaire pour indiquer le modèle de liaison à SQLModel.
- On peut accéder à tous les héros d'une équipe (`team.heroes`) et à toutes les équipes d'un héros (`hero.teams`), comme si c'était des listes Python.