Qu'est-ce que SQLModel ?

Sommaire

Contexte

Dans une application web moderne (comme avec FastAPI), on a souvent besoin de :

- définir des **modèles de données** pour la base SQL,
- valider des **données entrantes** (via des formulaires ou requêtes JSON),
- structurer des **réponses sortantes**.

Habituellement, on combine deux bibliothèques :

- **SQLAlchemy** → communication avec la base de données
- **Pydantic** → validation, sérialisation/désérialisation de données

Mais cela implique souvent **une duplication** : un modèle SQL + un modèle Pydantic. 👉 **SQLModel unifie tout ça**.

SQLAlchemy + Pydantic

☑ SQLAlchemy

ORM robuste et reconnu, SQLAlchemy permet :

- de créer des tables relationnelles avec Python,
- d'écrire des requêtes SQL à l'aide d'une API orientée objet.

Mais il nécessite des **déclarations manuelles** pour beaucoup d'éléments, et il ne sait pas gérer la validation JSON/API.

☑ Pydantic

Utilisé avec FastAPI ou seul, il permet de :

- **valider** les entrées JSON (type, format, champs requis…),
- **sérialiser/désérialiser** les objets Python ↔ JSON.

Mais il ne permet **ni stockage**, ni **accès à une base SQL**.

SQLModel = les deux réunis

SQLModel est une surcouche qui combine SQLAlchemy et Pydantic dans un même modèle.

- Basé sur les **annotations de type Python**
- Inspiré des **dataclasses**
- Fournit un modèle unique qui :
    - fonctionne avec une base SQL (via SQLAlchemy)
    - est compatible avec FastAPI (via Pydantic)

Exemple de modèle simple

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

class UserBase(SQLModel):
    username: str
    email: str

class User(UserBase, table=True):
    id: Optional[int] = Field(default=None, primary_key=True)
```

- `table=True` : indique que `User` est **persisté dans la base de données**
- `UserBase` pourra servir plus tard pour la validation FastAPI (ex : POST ou PUT)

> ℹ Dans ce cours, on se concentrera uniquement sur la partie **table SQL** (modélisation BDD). La gestion des **schémas de validation** (`UserCreate`, `UserRead`, etc.) sera abordée plus tard, dans le chapitre FastAPI.

Avantages

Avantage	Explication
🎯 Un modèle unique	Évite la redondance entre ORM et validation
🔡 Typé	Compatible avec les outils d'autocomplétion et type-checking
⚡ Simple et moderne	Élimine le boilerplate de SQLAlchemy
🧩 Pensé pour FastAPI	Fonctionne naturellement avec les endpoints API

À retenir

- SQLModel **fusionne SQLAlchemy et Pydantic** pour écrire des modèles simples, robustes et typés.
- Il permet une **intégration transparente avec les bases SQL** et prépare le terrain pour les **API REST**.
- Ici, on se concentre uniquement sur la **couche BDD** (tables, relations, sessions).