database.py

```
"""Database configuration"""
from sqlalchemy import create_engine
from sqlalchemy.orm import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "sqlite:///./movies.db"

# # Créer un moteur de base de données (engine) qui établit la connexion
avec notre base SQLite (movies.db).
engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)

# Définir SessionLocal, qui permet de créer des sessions pour interagir
avec la base de données.
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Définir Base, qui servira de classe de base pour nos modèles SQLAlchemy.
Base = declarative_base()

# # Optionnel : pour exécuter une vérification de la connexion à la base de
données
# # (peut être utile pour le débogage ou la configuration initiale).
# if __name__ == "__main__":
#     try:
#         with engine.connect() as conn:
#             print("Connexion à la base de données réussie.")
#     except Exception as e:
#         print(f"Erreur de connexion : {e}")
```

Voici une explication claire et simple de ce que font les trois instructions, avec un focus sur **les arguments** :

1. `create_engine(...)`

```
engine = create_engine(
    SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False}
)
```

Cette ligne **crée un moteur de base de données SQLAlchemy** qui va permettre à ton application Python d'interagir avec la base SQLite.

Explication des arguments :

- `SQLALCHEMY_DATABASE_URL` : c'est l'URL de connexion à ta base. Exemple ici :

```
"sqlite:///./movies.db"
```

> Cela veut dire : utiliser SQLite et se connecter à un fichier nommé `movies.db` situé dans le même dossier que ce fichier Python.

- **`connect_args={"check_same_thread": False}`** :
  - SQLite, par défaut, **interdit l'utilisation de la même connexion dans plusieurs threads**.
  - Or, FastAPI (et d'autres frameworks web) peuvent utiliser du **multithreading** pour gérer plusieurs requêtes en parallèle.
  - Donc `check_same_thread=False` **désactive cette restriction**.
  - Attention : À utiliser uniquement si **tu gères bien les sessions SQLAlchemy** (ce que fait FastAPI avec dépendances `Depends()`).

---

2. `sessionmaker(...)`

```
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
```

`sessionmaker` est une **fabrique de sessions**. Tu l'utilises pour créer des sessions qui vont te permettre de lire/écrire dans la base de données.

Explication des arguments :

- **`autocommit=False`** :
  - Cela signifie que **tu dois valider les transactions manuellement** (avec `.commit()`).
  - C'est plus sûr : tu peux rollback en cas d'erreur.
- **`autoflush=False`** :
  - Si c'était `True`, SQLAlchemy enverrait automatiquement les changements en base **avant certaines requêtes SELECT**.
  - Ici, on veut plus de contrôle. Donc on met `False` : les changements sont flushés **manuellement ou au moment du commit**.
- **`bind=engine`** :
  - Lie la session à l'**engine** que tu as créé plus haut.
  - Ainsi, toutes les sessions créées avec `SessionLocal()` vont utiliser la base `movies.db`.

Exemple d'utilisation de `SessionLocal` :

```
db = SessionLocal()
try:
    movies = db.query(Movie).all()
```

```
    finally:
        db.close()
```

---

3. `declarative_base()`

```
from sqlalchemy.orm import declarative_base

Base = declarative_base()
```

Cette ligne crée une **classe de base** nommée `Base` à partir de laquelle **tous tes modèles (tables)** vont hériter.

Pourquoi c'est utile ?

Lorsque tu définis une classe comme ceci :

```
class Movie(Base):
    __tablename__ = "movies"
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, index=True)
```

Tu es en train de créer :

- une **classe Python** (`Movie`)
- qui est **liée à une table SQL** (`movies`)
- avec des **colonnes** (`id`, `title`…)

Mais pour que SQLAlchemy comprenne que `Movie` doit être **une table dans la base de données**, il faut qu'elle hérite d'une **classe spéciale**, et c'est justement ce que `Base = declarative_base()` fournit.

En résumé :

Élément	Rôle
`declarative_base()`	Crée une superclasse `Base`
`Base`	Sert de base à tous tes modèles SQLAlchemy
Classe qui hérite de `Base`	Devient une table dans la base de données via la **declarative mapping**