مدل

مدل‌ها *(Models)* در *ORM*ها نقش اساسی را ایفا می‌کنند. هر مدل به‌صورت یک کلاس پایتون تعریف می‌شود و به عنوان یک جدول در پایگاه داده می باشد. *SQLAlchemy* این امکان را فراهم می‌کند که ساختار جداول پایگاه داده را نیز در قالب این کلاس‌ها تعریف کنیم. با استفاده از این کلاس‌ها، *SQLAlchemy* می‌تواند به‌صورت خودکار جداول متناظر را در پایگاه داده شما ایجاد کند. به این ترتیب، تمام تعاملات (مانند خواندن، نوشتن، به‌روزرسانی و حذف داده) از طریق اشیاء پایتونی و به شکل انتزاعی انجام می‌پذیرد و شما را از نوشتن مستقیم کوئری‌های *SQL* در بسیاری از موارد بی‌نیاز می‌کند.

تعریف مدل پایه

برای تعریف مدل‌ها با استفاده از روش *Declarative* در *SQLAlchemy* (که روش پیشنهادی و مدرن است)، نیاز است که کلاس‌های مدل شما از یک کلاس پایه‌ی مشخص ارث‌بری کنند. این کلاس پایه، `DeclarativeBase` نام دارد و از ماژول `sqlalchemy.orm` قابل‌دسترسی است.

Copy Python

```python
from sqlalchemy.orm import DeclarativeBase
import sqlalchemy as db

class Base(DeclarativeBase):
    pass
```

شاید بپرسید که چرا یک کلاس خالی به نام `Base` تعریف می‌کنیم و از آن ارث‌بری می‌کنیم، در حالی که می‌توان مستقیماً از `DeclarativeBase` ارث‌بری کرد؟

دلیل اصلی این کار، ایجاد یک نقطه‌ی مرجع واحد برای تمام مدل‌ها و اعمال رفتارهای مشترک بین آن‌ها است.

اگر بخواهیم یک متد یا ویژگی خاص را به تمام مدل‌های خود اضافه کنیم، کافی است آن را یک بار در کلاس `Base` تعریف کنیم. تمام مدل‌های مشتق‌شده به‌صورت خودکار از آن بهره‌مند خواهند شد. در مجموع، اگر در آینده نیاز به تغییرات سراسری در نحوه‌ی عملکرد مدل‌ها داشته باشیم، می‌توانیم این تغییرات را در `Base` اعمال کنیم و تمام مدل‌ها تحت‌تأثیر قرار بگیرند.

تعریف ستون‌ها

در نسخه‌های جدید *SQLAlchemy*، روش پیشنهادی برای تعریف ستون‌های جدول در مدل‌ها استفاده از ترکیب `Mapped` و *Type Hint* است.

- **`Mapped[Type]`** : این سازه (از ماژول `sqlalchemy.orm`) به *SQLAlchemy* اطلاع می‌دهد که یک ویژگی در کلاس مدل، نماینده‌ی یک ستون در جدول پایگاه داده است و نوع داده‌ی پایتونی متناظر آن چیست (`Type`).
- **`mapped_column(...)`** : این تابع (از ماژول `sqlalchemy.orm`) برای ارائه‌ی تنظیمات بیشتر و دقیق‌تر مربوط به ستون در پایگاه داده استفاده می‌شود. در حالی که `Mapped[Type]` نوع داده‌ی پایتونی را مشخص می‌کند، `mapped_column` جزئیات مربوط به ستون پایگاه داده مانند نام ستون (اگر با نام ویژگی متفاوت باشد)، نوع داده‌ی *SQL*، کلید اصلی بودن، ویژگی `autoincrement` ، پیش‌فرض‌ها، قابلیت *Null* بودن و ... را تعیین می‌کند.

مثال: فرض کنید می‌خواهیم جدولی به نام `books` برای ذخیره‌ی اطلاعات کتاب‌ها در پایگاه داده ایجاد کنیم. این جدول باید شامل ستون‌های زیر باشد:

- **شناسه‌ی کتاب (`id`):** یک عدد صحیح یکتا برای شناسایی هر کتاب.
- **عنوان (`title`):** عنوان کتاب (متن).
- **نویسنده (`author`):** نام نویسنده کتاب (متن).
- **تاریخ انتشار (`publication_date`):** تاریخ انتشار کتاب (تاریخ).

```python
from sqlalchemy.orm import DeclarativeBase
import sqlalchemy as db
from sqlalchemy.orm import Mapped, mapped_column

class Base(DeclarativeBase):
    pass

class Book(Base):
    __tablename__ = 'books'
    id: Mapped[int] = mapped_column(primary_key=True, autoincrement=True)
    title: Mapped[str]
```

```
12      author: Mapped[str]
13      publication_date: Mapped[db.Date]
```

- خط ۱۰ نام جدولی که این مدل به آن نگاشت می‌شود را مشخص می‌کند.
- خط ۱۲ ستون `id` را تعریف می‌کند. `Mapped[int]` نشان می‌دهد که نوع پایتونی این ستون `int` است. `mapped_column` با آرگومان‌های `primary_key=True` و `autoincrement=True` به *SQLAlchemy* می‌گوید که این ستون کلید اصلی جدول است و مقادیر آن باید به‌صورت خودکار توسط پایگاه داده تولید شوند.
- خطوط ۱۳ و ۱۴ ستون‌های `title` و `author` را تعریف می‌کنند. `Mapped[str]` نشان می‌دهد که نوع پایتونی این ستون‌ها `str` (رشته) است. از آن‌جایی که نیاز به تنظیمات پیشرفته‌ای نداریم، نیازی به استفاده از `mapped_column` نیست و *SQLAlchemy* نوع مناسب *SQL* (مثلا `VARCHAR`) را استنتاج می‌کند.
- خط ۱۵ ستون `publication_date` را تعریف می‌کند. `Mapped[db.Date]` (که `db` همان `sqlalchemy` است) نشان می‌دهد که نوع پایتونی این ستون، شیء تاریخ *SQLAlchemy* است که به نوع تاریخ در پایگاه داده نگاشت می‌شود.

نکته: در تعریف ستون‌ها با `Mapped`، اگر نوع پایتونی به تنهایی برای *SQLAlchemy* کافی نباشد یا نیاز به تنظیمات خاصی داشته باشید، باید از `mapped_column` استفاده کنید. **مثال:** برای تعریف یک ستون عددی که *Null*پذیر نیست و مقدار پیش‌فرض دارد، ممکن است بنویسید:

```
1  count: Mapped[int] = mapped_column(nullable=False, default=0)
```

ایجاد جداول در پایگاه داده

پس از تعریف مدل‌ها، *SQLAlchemy* به‌صورت خودکار جداول متناظر را در پایگاه داده شما ایجاد **نمی‌کند**. برای انجام این کار، باید از متد `create_all` استفاده کنید. این متد بر روی شیء `metadata` که به کلاس پایه‌ی مدل‌های شما متصل است، در دسترس قرار دارد.

برای استفاده از `create_all` ، نیاز به یک *Engine* دارید که در درس‌نامه‌های قبل نحوه‌ی ایجاد آن را یاد گرفته‌اید. اکنون باید یک شیء *Engine* به نام `engine` داشته باشید.

مثال:

```
from sqlalchemy import create_engine
from your_app.models import Base

engine = create_engine("sqlite:///database.db")

Base.metadata.create_all(engine)
```

متد `create_all(engine)` تمام مدل‌هایی را که از `Base` ارث‌بری کرده‌اند، بررسی می‌کند. برای هر مدل، اگر جدول متناظر با `__tablename__` آن در پایگاه داده متصل به `engine` وجود نداشته باشد، دستورات *SQL* لازم برای ایجاد آن جدول اجرا می‌شود. اگر جداول از قبل وجود داشته باشند، هیچ تغییری اعمال نمی‌شود و خطایی رخ نمی‌دهد.