SQLAlchemy 이해하기

철학부터 실전 마이그레이션까지 배우는 데이터베이스 툴킷

👋발표자 : 채다현

이 세미나에서 다룰 내용

1. SQLAlchemy의 정의와 철학
2. Core / ORM 구조와 비교
3. 1.x → 2.x 버전 변화
4. SQL 표현식 예제
5. Alembic 마이그레이션

1. SQLAlchemy란?

- Python에서 가장 널리 사용되는 ORM + SQL Toolkit
- 다양한 DBMS 지원 (MySQL, PostgreSQL 등)
- ORM + Core 두 가지 방식 제공
- SQL도, 객체지향도 모두 잡을 수 있는 라이브러리

왜 SQLAlchemy를 써야 할까?

- 표준화된 DB 접근 방식
- ORM + Core로 유연한 설계
- 생산성과 성능의 균형
- 다양한 실무 적용 사례

2. SQLAlchemy의 철학

> “SQLAlchemy는 ORM이 아니라 SQL Toolkit이다. - Michael Bayer”

- 데이터베이스를 제어하기 위한 강력한 툴킷(toolkit)
  - ORM, Core, 엔진 등 구성 요소를 자유롭게 조합
  - 복잡함을 감추지 않고, 개발자에게 제어권을 준다.
- 추상화보다 통제에 집중
  - 복잡한 쿼리도 직접 명시적으로 작성 가능
- ORM과 Core를 모두 지원
  - 고수준 ORM과 저수준 SQL 표현식을 모두 활용 가능

Django ORM vs SQLAlchemy ORM

```
# Django ORM 예시
user = User.objects.filter(name="Alice").first()

# SQLAlchemy ORM 예시 (2.x 스타일)
stmt = select(User).where(User.name == "Alice")
with Session(engine) as session:
    user = session.execute(stmt).scalar_one_or_none()
```

3. SQLAlchemy 1.x vs 2.x

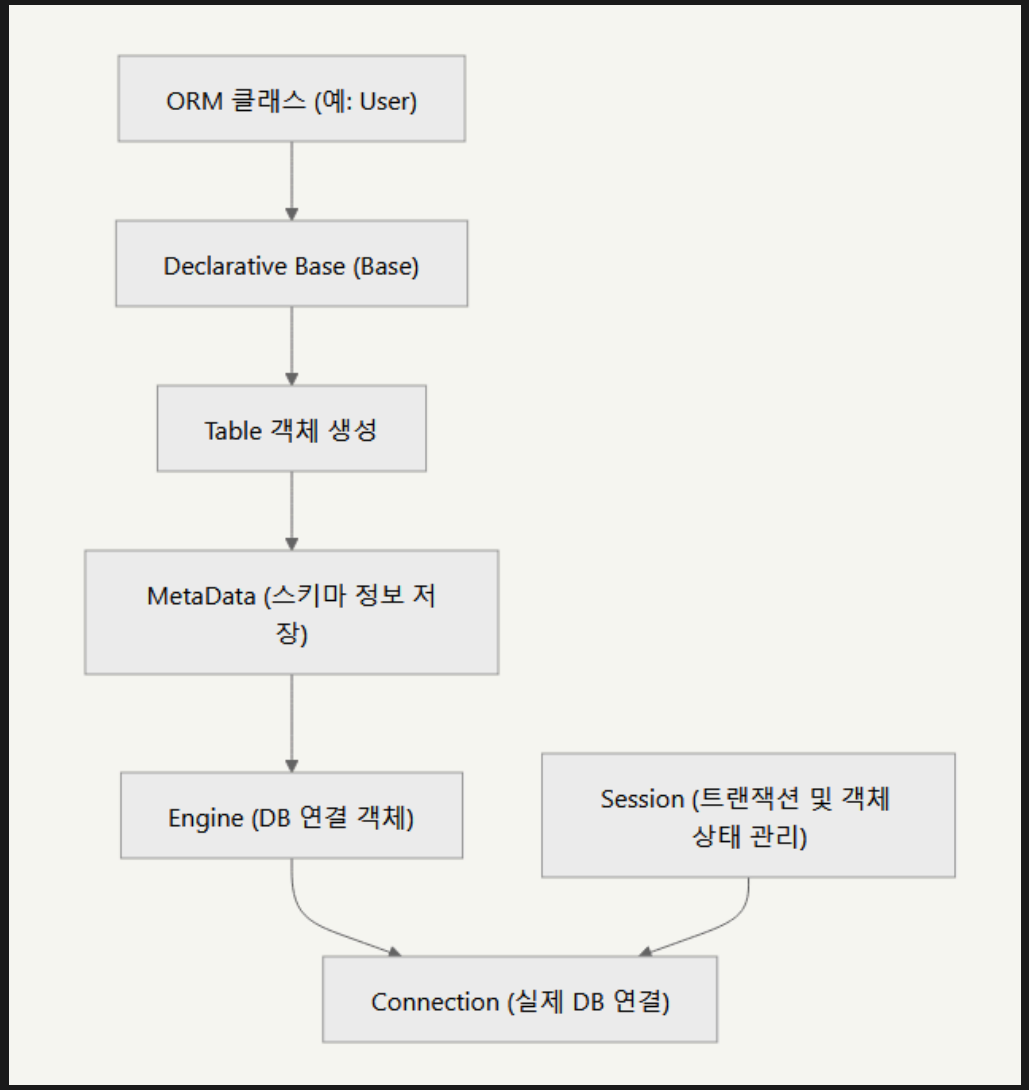
항목	SQLAlchemy 1.x	SQLAlchemy 2.x (Modern Style)
세션 사용	수동 생성 / close 필요	with Session(...) 명시적 관리
ORM 쿼리	session.query()	select(...) + session.execute()
결과 추출 방식	.first(), .all() 등	.scalars(), .scalar_one(), .fetchall()
Insert 방식	add(), merge()	insert() + execute()
Async 지원	거의 없음	공식 async 지원 (AsyncSession)
사용 철학	ORM 중심, 암시적 스타일	Core 중심, 명시적 + 함수형 스타일

4. SQLAlchemy 컴포넌트

구성요소	설명
Engine	데이터베이스 연결을 관리하는 객체
Session	ORM의 트랜잭션과 객체 상태 관리
MetaData	테이블 및 스키마 정보를 저장하는 객체
Table / Column	SQL 표현을 위한 테이블과 컬럼 정의
Declarative Base	ORM 클래스 생성을 위한 기본 클래스

- ORM 클래스는 Declarative Base를 통해 정의
  - 내부적으로 Table과 MetaData로 변환되며, Engine과 Session을 통해 실제 DB와 통신

SQLAlchemy 컴포넌트 흐름도



```python
# sqlalchemy_2x_example
from sqlalchemy import create_engine, Column, Integer, String, select
from sqlalchemy.orm import declarative_base, Session

# 1. Base 정의
Base = declarative_base()

# 2. ORM 클래스 정의
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    email = Column(String)

# 3. Engine 설정
engine = create_engine("sqlite:///example_2x.db", echo=True, future=True)

# 4. 테이블 생성
Base.metadata.create_all(engine)

# 5. 세션 내에서 회원 생성 및 조회
with Session(engine) as session:
    # 회원 생성
    new_user = User(name="홍길동", email="hong@example.com")
    session.add(new_user)
    session.commit()

    # 회원 조회 (select + scalars)
    stmt = select(User).where(User.name == "홍길동")
    user = session.execute(stmt).scalars().first()

    print(f"조회된 사용자: {user.name}, {user.email}")
```

5. SQLAlchemy Core 와 ORM

ORM 방식

```
session.query(User).filter_by(name='Alice').first()
```

Core 방식

```
stmt = select(user_table).where(user_table.c.name == 'Alice')
conn.execute(stmt)
```

5.1. SQLAlchemy Core 소개 및 사용법

- SQL 표현식을 Python으로 작성
- ORM보다 저수준, 세밀한 제어 가능
- SQL에 익숙한 개발자에게 유리

```
user_table = Table("user", metadata,
    Column("id", Integer, primary_key=True),
    Column("name", String)
)

with engine.connect() as conn:
    conn.execute(insert(user_table).values(name="Alice"))
    result = conn.execute(select(user_table))
```

5.2. SQLAlchemy ORM 소개 및 사용법

- Python 클래스를 테이블로 추상화
- Session 객체로 데이터 관리
- 객체지향 개발자에게 유리

```
class User(Base):
    __tablename__ = "user"
    id = Column(Integer, primary_key=True)
    name = Column(String)

user = User(name="Bob")
session.add(user)
session.commit()

user = session.query(User).filter_by(name="Bob").first()
```

5.3. Core vs ORM 비교

항목	Core	ORM
작성 방식	SQL 표현식	객체지향 추상화
유연성	높음	쉬움
속도	느릴 수 있음	빠름
학습 곡선	SQL 지식 필요	낮음
장점	복잡한 쿼리 제어 가능	빠른 개발 속도
단점	장황하고 verbose 할 수 있음	쿼리 추적 어려움

5.4. SQLAlchemy Core 와 ORM 동시사용

- SQLAlchemy는 Core와 ORM을 혼합하여 사용하는 것을 권장
- 복잡한 쿼리는 Core 스타일(select(), join())로 작성하고, ORM 객체와 함께 사용 가능

```
from sqlalchemy import select
from sqlalchemy.orm import Session

# Core 스타일 쿼리: ORM 모델(Employee)을 대상으로 조건 지정
stmt = (
    select(Employee)
    .where(Employee.status == "100")              # 재직자만
    .where(Employee.organization_seq == 10)       # 조직 ID = 10
)

# 2.x 스타일의 명시적 세션 관리
with Session(engine) as session:
    employees = session.execute(stmt).scalars().all()

    # 결과: 조직 10번에 속한 재직자 목록 (Employee 객체 리스트)
```

6. Alembic 마이그레이션

- SQLAlchemy의 공식 마이그레이션 도구
- DB 스키마 변경을 버전 관리

```
alembic init alembic
alembic revision --autogenerate -m "create table"
alembic upgrade head
```

참고자료

- SQLAlchemy 공식 문서
- SQLAlchemy
- Alembic 마이그레이션 실습 예제 링크