Введение в SQLAlchemy

Основы ORM в Python

Лазар В. И., Козлова Е. Р.

19 февраля 2025 г.

План занятия

1. Что такое SQLAlchemy?
2. Установка и настройка
3. Создание моделей
4. Работа с данными
5. Связи между моделями

Обзор SQLAlchemy

- **SQLAlchemy** — одна из самых популярных ORM-библиотек для Python.
- Позволяет работать с разными СУБД (SQLite, PostgreSQL, MySQL и др.) единым способом.
- Делит функционал на *Core* (работа с SQL на более низком уровне) и *ORM* (объектно-реляционное отображение).
- Упрощает создание и сопровождение проектов: меньше «ручного» SQL, больше объектного подхода.

Зачем нужна ORM?

- **Отделение бизнес-логики** от конкретных SQL-запросов.
- **Кросс-база**: без изменения кода можно подключить другую СУБД.
- **Удобство**: вместо таблиц и строк — привычные классы и объекты.
- **Безопасность**: автоматическая экранизация параметров, меньше риска SQL-инъекций.

Установка

Используйте pip

```
pip install sqlalchemy
```

- Для работы с SQLite хватит `sqlite3` из стандартной библиотеки.
- При необходимости подключать драйверы для других СУБД (например, `psycopg2` для PostgreSQL).

Базовая конфигурация

Пример создания engine и session

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

# Пример для SQLite
engine = create_engine('sqlite:///example.db')

Session = sessionmaker(bind=engine)
session = Session()
```

- **Engine** — «двигатель» для подключения к базе.
- **Session** — объект для взаимодействия с БД (объект транзакции).

Декларативная модель

Пример Python-класса

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    username = Column(String, nullable=False)
    city = Column(String)
```

- Base — базовый класс для всех моделей.
- **Колонки** описываются как атрибуты класса.
- __tablename__ — имя таблицы в БД.

Создание таблиц по моделям

Пример

```
Base.metadata.create_all(engine)
```

- `create_all` проверяет, какие таблицы уже существуют, и создаёт отсутствующие.
- Полезно при первом запуске или при добавлении новых моделей.

Добавление данных

Пример Python-кода

```
new_user = User(username='Ivan', city='Moscow')
session.add(new_user)
session.commit()
```

- `session.add` — подготавливает объект к вставке.
- `commit()` — фиксирует транзакцию (CREATE или UPDATE).

Выборка (SELECT)

Пример

```
users = session.query(User).all()
for user in users:
    print(user.username, user.city)
```

- `query(User).all()` вернёт список всех объектов из таблицы `users`.
- Можно применять `filter`, `order_by`, `limit` и прочие SQL-конструкции.

Обновление данных

Пример

```
user = session.query(User).filter_by(username='Ivan').first()
user.city = 'Saint Petersburg'
session.commit()
```

- При изменении атрибута объекта и вызове `commit()` SQLAlchemy сформирует и выполнит `UPDATE`-запрос.

Удаление данных

Пример

```
user_to_delete = session.query(User).filter_by(username='Ivan')
session.delete(user_to_delete)
session.commit()
```

- `session.delete` готовит объект к удалению.
- `commit()` выполняет DELETE-запрос.

Отношение One-to-Many

```
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship

class Department(Base):
    __tablename__ = 'departments'
    id = Column(Integer, primary_key=True)
    name = Column(String)

class Employee(Base):
    __tablename__ = 'employees'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    department_id = Column(
                Integer, ForeignKey('departments.id'))
    department = relationship(
                'Department', backref='employees')
```

Отношение One-to-Many

- ForeignKey в Employee указывает на departments.id.
- relationship позволяет обращаться к Employee.department.
- backref='employees' даёт доступ к списку сотрудников из Department.

Отношение One-to-One

- В SQLAlchemy нет специального ключевого слова «One-to-One».
- Реализуется как `One-to-Many` с дополнительным уникальным ограничением или через `uselist=False`.

Пример с `uselist=False`

```
class Person(Base):
    __tablename__ = 'persons'
    id = Column(Integer, primary_key=True)
    name = Column(String)

class Passport(Base):
    __tablename__ = 'passports'
    id = Column(Integer, primary_key=True)
    person_id = Column(Integer,
        ForeignKey('persons.id'), unique=True)
    person = relationship('Person', backref='passport',
                                    uselist=False)
```

Отношение One-to-One

- `unique=True` указывает, что `person_id` уникален — не может указывать на несколько записей (строк).
- `uselist=False` даёт понять, что ожидается единственный объект.

Отношение Many-to-Many (через ассоциативную таблицу)

Пример

```
association_table = Table('association', Base.metadata,
    Column('student_id', Integer, ForeignKey('students.id')),
    Column('course_id', Integer, ForeignKey('courses.id'))
)
```

Отношение Many-to-Many (через ассоциативную таблицу)

Пример

```
class Student(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    courses = relationship('Course',
                secondary=association_table,
                back_populates='students')
class Course(Base):
    __tablename__ = 'courses'
    id = Column(Integer, primary_key=True)
    title = Column(String)
    students = relationship('Student',
                secondary=association_table,
                back_populates='courses')
```

Отношение Many-to-Many (через ассоциативную таблицу)

- `association_table` — вспомогательная таблица с полями `student_id` и `course_id`.
- `relationship(..., secondary=...)` связывает через таблицу.
- `back_populates` указывает на связное поле в другой модели.

Общая идея проекта

Система управления проектами и задачами (консольное приложение):

- Написать на Python (используя SQLAlchemy) мини-систему для работы с `Project`, `User`, `Profile`, `Task`.
- Обязательно задействовать:
  - Все основные операции с БД (CRUD: Create, Read, Update, Delete).
  - Различные типы связей: One-to-One, One-to-Many, Many-to-Many.

Требования к проекту (1/2)

- Модели (таблицы) и связи:
  1. **User (пользователь)**
     - `id`, `username`, `email`
     - Связь One-to-One с `Profile`
     - Связь Many-to-Many с `Project`
  2. **Profile (профиль пользователя)**
     - `id`, `bio`, `phone`, `user_id` (уникальный ForeignKey)
  3. **Project (проект)**
     - `id`, `title`, `description`
     - Связь One-to-Many с `Task`
     - Связь Many-to-Many с `User`
  4. **Task (задача)**
     - `id`, `title`, `status`, `project_id` (ForeignKey)

Требования к проекту (2/2)

- **CRUD-операции:**
  - Создание новых записей (`User`, `Project`, `Task`, `Profile`).
  - Чтение (просмотр) списков и деталей (вывести всех пользователей, задачи проекта и т. п.).
  - Обновление (изменить `email` пользователя, статус задачи, описание проекта и т. д.).
  - Удаление (убрать ненужного пользователя, проект или задачу).
- **Пример функционала**: «Добавить проект», «Добавить задачу», «Вывести пользователей», «Назначить пользователя на проект» (Many-to-Many), «Добавить профиль к пользователю» (One-to-One).
- **Демонстрация работы связей**:
  - `user.profile`, `project.tasks`, `user.projects` и т. п. (в зависимости от настроенного `relationship`).