

RELATIONSHIP

ВВЕДЕНИЕ

Задачи урока

- Научиться использовать связи в таблицах, где есть связь один ко многим

В SQLAlchemy тип данных **Relationship** используется для определения связей между таблицами в базе данных. Этот тип данных позволяет определить отношения между моделями данных и упрощает работу с связанными данными

Связь в SQLAlchemy может быть односторонней или двусторонней, и она может представлять отношения "один-к-одному", "один-ко-многим" или "многие-ко-многим"

Для определения связей между моделями в SQLAlchemy используется функция **relationship()**. Эта функция принимает различные аргументы и параметры, которые позволяют настроить связь между моделями

Типичные примеры связей в SQLAlchemy

01

Связь "Один-к-Одному"

Означает, что каждая запись в одной таблице (таблице А) связана лишь с одной записью в другой таблице (таблице В), и наоборот. Примером может быть связь между таблицами "Пользователь" и "Адрес". Каждый пользователь имеет только один адрес, и каждый адрес принадлежит только одному пользователю

```
...
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    address_id = Column(Integer, ForeignKey('addresses.id'))
    address = relationship('Address', backref='user')
```

```
class Address(Base):
    __tablename__ = 'addresses'
    id = Column(Integer, primary_key=True)
    street = Column(String)
    city = Column(String)
```

В этом примере класс **User** имеет связь "Один-к-Одному" с классом **Address**. Связь устанавливается через атрибут `address`, используя функцию `relationship()`. Аргумент `backref` указывает имя обратной ссылки, через которое можно получить доступ от **Address** к связанным записям **User**

02

Связь "Один-ко-Многим"

Означает, что каждая запись в одной таблице (таблице А) может быть связана с несколькими записями в другой таблице (таблице В), но каждая запись в таблице В может быть связана только с одной записью в таблице А. Примером может быть связь между таблицами "Пользователь" и "Пост". Каждый пользователь может иметь несколько постов, но каждый пост принадлежит только одному пользователю

```
...
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    posts = relationship('Post', backref='user')

class Post(Base):
    __tablename__ = 'posts'
    id = Column(Integer, primary_key=True)
    title = Column(String)
    content = Column(String)
    user_id = Column(Integer, ForeignKey('users.id'))
```

В этом примере класс **User** имеет связь "Один-ко-Многим" с классом **Post**. Связь устанавливается через атрибут `posts`, используя функцию `relationship()`. Аргумент `backref` указывает имя обратной ссылки, через которое можно получить доступ от **Post** к связанным записям **User**

03

Связь "Многие-ко-Многим"

Означает, что каждая запись в одной таблице (таблице А) может быть связана с несколькими записями в другой таблице (таблице В), и наоборот. Примером может быть связь между таблицами "Пользователь" и "Роль". Каждый пользователь может иметь несколько ролей, и каждая роль может быть у пользователя

Для установки связи "Многие-ко-Многим" в SQLAlchemy требуется использовать дополнительную таблицу, называемую связующей таблицей. Эта таблица содержит внешние ключи к таблицам А и В, и она отображает отношение между ними

```
...
from sqlalchemy import Table, Column, Integer, ForeignKey
from sqlalchemy.orm import relationship
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

association_table = Table('user_roles', Base.metadata,
    Column('user_id', Integer, ForeignKey('users.id')),
    Column('role_id', Integer, ForeignKey('roles.id'))
)

class User(Base):
    __tablename__ = 'users'

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

    roles = relationship("Role", secondary=association_table,
        back_populates="users")

class Role(Base):
    __tablename__ = 'roles'

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

    users = relationship("User", secondary=association_table,
        back_populates="roles")
```

В этом примере класс `User` имеет связь "Многие-ко-Многим" с классом `Role`. Связь устанавливается через атрибут `roles`, используя функцию `relationship()`. Параметр `secondary` указывает таблицу, которая служит связующей таблицей для связи "Многие-ко-Многим" `User` и `Role`. Аргумент `backref` указывает имя обратной ссылки, через которое можно получить доступ от `Role` к связанным записям `User`

Это лишь несколько примеров применения **Relationship** в **SQLAlchemy**, и он предлагает гибкий и мощный способ определения связей между таблицами в базе данных

ИТОГИ

- ✓ Если у нас есть связь один ко многим, мы можем использовать **relationship** для связи с другой таблицей, и тогда при выборе данных нам не нужно будет добавлять **join** к запросу
- ✓ Если у нас есть связь многие ко многим, то мы можем использовать справочник и этот справочник использовать в **relationship**