

Урок 3. Синхронное и асинхронное подключение

Весь код по этой теме вы сможете изучить в GitLab.

Рассмотрим синхронное и асинхронное подключение к СУБД PostgreSQL.

Ранее при работе с ORM мы использовали синхронное подключение к SQLite. Сейчас настроим аналогичное подключение к PostgreSQL, разберём особенности и познакомимся с асинхронным подключением. В документации к ORM есть целый раздел, который посвящён настройке подключения к PostgreSQL. Ссылку на него вы можете найти под видео.

В первую очередь относительно SQLite меняется драйвер для подключения к базе. Наиболее популярный вариант — `psycopg2`. В документации указано, что его основные особенности:

- полная реализация спецификации Python DB API 2.0 (PEP 249), который обсуждался в модулях по SQLAlchemy;
- безопасность потоков — несколько из них могут использовать одно и то же соединение.

Настроим подключение и разберёмся с особенностями работы ORM с PostgreSQL.

SEQUENCE

PostgreSQL поддерживает последовательности. SQLAlchemy по умолчанию использует их как средства для создания новых значений первичного ключа на основе целых чисел. SEQUENCE — это объект, который генерирует числовые значения в определённой последовательности в соответствии с заданной спецификацией.

Одно из основных предназначений SEQUENCE — формирование значений для столбца идентификаторов в таблицах.

Identity

Конструкция Identity позволяет объявлять столбец как столбец идентификаторов. Его значение автоматически генерируется сервером базы данных с использованием возрастающей или убывающей последовательности.

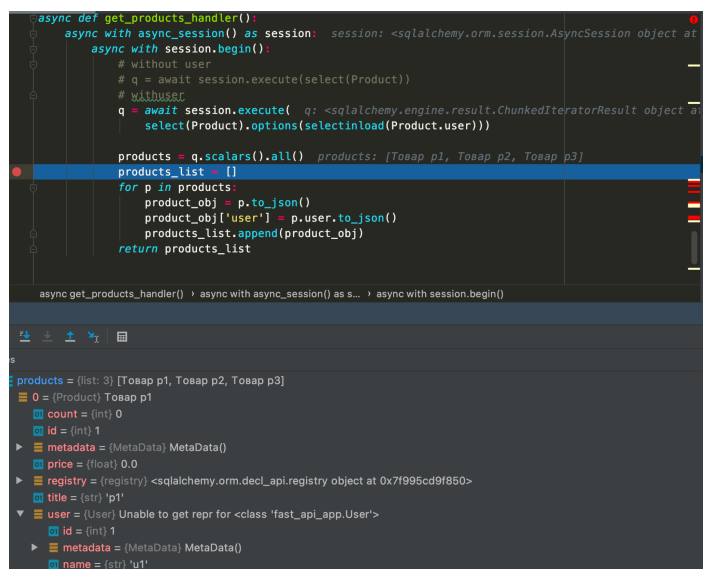
Конструкция разделяет большую часть своих возможностей управления поведением базы данных с Sequence.

Мы разобрали документацию ORM, но существуют и другие расширения, которые поддерживаются при работе с СУБД PostgreSQL. Например, полнотекстовый поиск. С ними вы можете познакомиться самостоятельно, прочитав документацию по ссылке под видео.

Разберём асинхронную работу с СУБД.

Для этого потребуется новый объект при создании сессии — `AsyncSession`. Работая с ним, нужно учитывать, что при асинхронной работе появляются проблемы с «ленивой» подгрузкой данных связанных таблиц ORM. То есть нельзя по умолчанию обратиться к ним через полученный объект.

При создании сессии через объект `sessionmaker` необходимо указать атрибут `expire_on_commit = False`. При значении `= True` срок действия всех экземпляров сессии полностью истекает после каждого `commit()`. Весь доступ к объектам после завершённой транзакции загружается из последнего состояния базы данных. В случае `= False` — не придётся после каждого коммита запрашивать данные заново. При асинхронном подключении это необходимо, чтобы гарантировать, что данные будут доступны даже после транзакции в сессии.



```
async def get_products_handler():
    async with async_session() as session:
        async with session.begin():
            # without user
            # q = await session.execute(select(Product))
            # with user
            q = await session.execute(
                q: <sqlalchemy.engine.result.ChunkedIteratorResult object at 0x7f995cd9f850>
                select(Product).options(selectinload(Product.user)))

            products = q.scalars().all()
            products_list = []
            for p in products:
                product_obj = p.to_json()
                product_obj['user'] = p.user.to_json()
                products_list.append(product_obj)
            return products_list

async get_products_handler() -> async with async_session() as s... -> async with session.begin()
```

The screenshot shows a debugger window with the following output:

```
products = (list: 3) [Товар p1, Товар p2, Товар p3]
0 = (Product) Товар p1
count = (int) 0
id = (int) 1
metadata = (MetaData) MetaData()
price = (float) 0.0
registry = (registry) <sqlalchemy.orm.decl_api.registry object at 0x7f995cd9f850>
title = (str) 'p1'
user = (User) Unable to get repr for <class 'fast_api_app.User'>
id = (int) 1
metadata = (MetaData) MetaData()
name = (str) 'u1'
```

Все особенности работы с асинхронным подключением представлены в документации ORM. Ссылку на неё можно найти под видео.

В этом видео мы обсудили синхронную и асинхронную работу с СУБД. На примере flask-приложения рассмотрели синхронный вариант работы с ORM, а на примере fast-api — асинхронный. В следующем видео мы разберём, что такое миграция БД, и познакомимся с полезными инструментами для работы с миграциями.