План урока

- Взглянем на виды Баз Данных (БД) и инструменты для работы с БД через Python
- Установим PostgreSQL и pgAdmin
- Опишем модели таблиц через SQLAlchemy и научимся управлять состоянием БД через Alembic
- Заполним таблицы тестовыми данными

Виды баз данных

Бэкенд немыслим без хранилища данных, в качестве хранилища может выступать практически что угодно — список или словарик внутри нашей Python программы, простой текстовый файл на компьютере, базы данных внутри оперативной памяти и базы данных на жестком диске.

Хранение важных данных в программе Python или внутри оперативной памяти хоть и позволяет быстро к ним обращаться, имеет один огромный минус — при выключении программы или выключении компьютера/сервера все данные навсегда стираются. Если мы хотим надежно хранить наши данные и иметь возможность к ним обращаться даже после перезагрузки компьютера/сервера, то необходимо использовать базы данных (БД) с хранением данных на жестком диске.

БД можно разделить на два типа: реляционные (в формате таблиц) и нереляционные (обычно в формате JSON — аналоге словаря в Python, грубо говоря).

Давайте рассмотрим их отличия на простых примерах.

В реляционных БД обычно хранят данные одинаковой структуры, которая не меняется со временем. Например, мы можем хранить таблицу пользователей, у каждого из которых есть лишь два поля: имя и фамилия.

Пример таблицы в реляционной (SQL) БД

id

first_name

last_name

1

Артём

Шумейко

2

Гвидо

ван Россум

3

Sebastián

Ramírez

Пример записей в нереляционной (NoSQL) БД. Самый распространенный пример — это интернет-магазин с товарами, у каждого из которых есть свое уникальное свойство. Например, у часов есть характеристика защита от влаги и ширина ремешка, в то время как у печенек нету таких свойств, зато есть другие — срок годности и форма печенек.

Пример записей в NoSQL БД

```
{
    "products": [
        {
            "product_id": 1,
            "name": "Печенье ручной работы",
            "product_type": "печенье",
            "price": 505,
            "details": {
                "expiry_date": 280,
                "form": "Фигурное"
            }
        },
        {
            "product_id": 2,
            "name": "Часы Cacasio B1408",
```

```
            "product_type": "часы наручные",
            "price": 5990,
            "details": {
                "diameter": 44,
                "strap_width": 30
            }
        },
    ]
}
```

Работа с базой данных из Python. Сырые SQL запросы vs ORM

Здесь и далее будет рассматриваться и описываться работа с реляционными базами данных, а именно с PostgreSQL. С нереляционными БД также можно работать из Python, для этого есть специальные библиотеки, например, pymongo для работы с MongoDB. Для работы с определенной базой данных необходима определенная библиотека, заточенная именно под эту базу данных, например для работы с MySQL может понадобиться библиотека PyMySQL, а для работы с PostgreSQL можно выбрать среди psycopg2, asyncpg, aiopg. Для синхронной работы с Postgres используется psycopg2, для асинхронной — обычно asyncpg, реже aiopg.

Важно понимать, что все эти библиотеки делают одно и тоже — отправляют сформированный вами запрос в базу данных. Разница в том, какой интерфейс (какие классы и функции) для создания запросов они предоставляют разработчику и предоставляют ли они возможность исполнять запросы асинхронно.

Несмотря на то, что многие знают и любят писать сырые SQL запросы через Python, порой гораздо удобнее писать их на "питонячьем" языке, то есть используя так называемый ORM. ORM — object-relational mapping или объектно-реляционное отображение — это способ взаимодействия приложения с базой данных посредством синтаксиса языка, на котором написано приложение. Объекты — это представление таблиц (реляций) на языке программирования (в нашем случае Python). Например, для простой выборки данных из таблицы `users` вместо запроса

```
query = "SELECT * FROM users"
```

на языке SQL мы можем использовать синтаксис языка Python для написания точно такого же запроса через

```
query = select(Users)
```

Понятное дело, что под капотом функции select и класса Users находится указание на язык SQL. Для Python есть одна невероятно популярная библиотека, которая дает

возможность писать SQL запросы, используя конструкции языка Python — она называется SQLAlchemy. SQLAlchemy сама не отправляет запросы в БД, для этих целей она использует уже готовые библиотеки, называемые драйверами, например, psycopg2 или asyncpg. SQLAlchemy дает возможность формировать как сырые SQL запросы, так и писать через ORM.

Про знание ORM и, в частности, SQLAlchemy спрашивают почти на каждом собеседовании на начинающего backend разработчика.

Плюсы SQLAlchemy:

1. При использовании Алхимии можно работать с таблицами как с объектами на "питонячем" языке
2. Алхимия защищает нас от SQL-инъекций
3. Алхимия позволяет абстрагироваться от конкретной СУБД при написании запросов. Это позволяет сменить используемую базу данных, например, с MySQL на Postgres без больших трудностей

Если вы пришли из Django, то вы, конечно знакомы с ORM, но переход на ORM Алхимии может протекать не так быстро, как вы ожидаете, ввиду больших отличий в названиях функций/методов и способах их применения.

Миграции

Еще одно преимущество работы с базой данных через SQLAlchemy — возможность создания миграций для БД. Миграция — это набор команд, которые изменяют структуру БД, например, добавляют/удаляют таблицу, добавляют/удаляют столбец, изменяют тип столбца. Давайте взглянем на пример миграции, в которой в базу данных добавляется таблица с названием `users` и столбцами `id, email, hashed_password`, задаются их типы, прописываются первичные ключи и так далее. Здесь `sa` — это SQLAlchemy.

```
def upgrade() -> None:
    op.create_table(
        "users",
        sa.Column("id", sa.Integer(), nullable=False),
        sa.Column("email", sa.String(), nullable=False),
        sa.Column("hashed_password", sa.String(), nullable=False),
        sa.PrimaryKeyConstraint("id"),
    )
```

Это обычная функция на языке Python. Под капотом эта команда преобразуется в SQL запрос, который будет отправлен в БД.

```
CREATE TABLE users (
    id int PRIMARY KEY,
    email text NOT NULL,
    hashed_password text NOT NULL
);
```

Для того, чтобы изменять состояние БД через Python, используется библиотека Alembic, написанная автором Алхимии.

1. Миграции позволяют управлять состоянием БД через Python, не прикасаясь к написанию SQL запросов
2. Всегда можно откатиться к нужному состоянию БД, быстро и качественно

Для миграций в Python чаще всего используется библиотека Alembic. Таким образом, в данном курсе используются самая популярная связка библиотек для работы с реляционными БД: SQLAlchemy + Alembic.