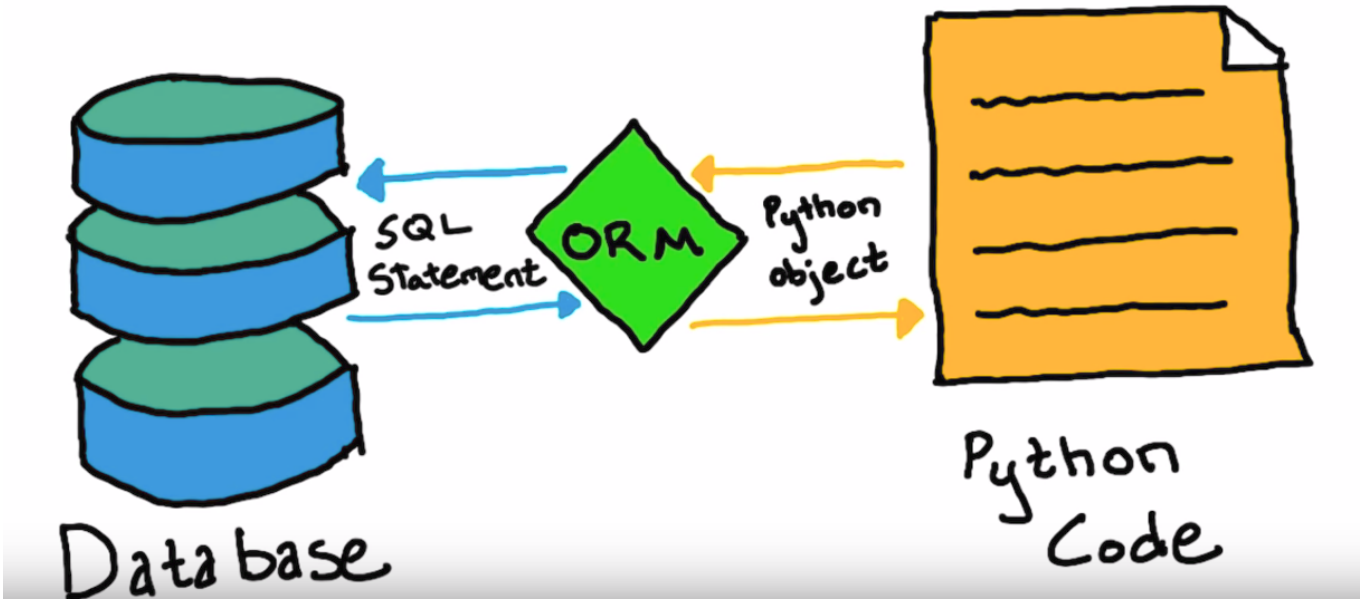
Lesson 28 - ORM

🔷 **Что такое ORM?**

ORM (Object-Relational Mapping) — это способ работы с базой данных через объектно-ориентированное программирование. Вместо написания SQL-запросов напрямую, разработчик взаимодействует с объектами и методами языка программирования.

🔥 **Преимущества ORM:**

✅ Упрощает работу с БД, особенно при сложных связях.
✅ Позволяет писать код без явного SQL.
✅ Поддерживает кросс-базовую совместимость (можно легко менять БД).
✅ Автоматически защищает от SQL-инъекций.



🔷 **SQLAlchemy: мощнейшая ORM для Python**

SQLAlchemy — это самая популярная библиотека для работы с БД в Python. Она поддерживает как **“чистый SQL” (Core API)**, так и **ORM**.

🛠️ **Установка:**

```
pip install sqlalchemy
```

🔷 **1. Создание соединения с БД**

Для PostgreSQL, MySQL и других БД строка подключения будет отличаться. Например, для PostgreSQL:

```
engine = create_engine("postgresql://user:password@localhost:5432/mydb")
```

🔷 2. Определение модели (таблицы)

ORM использует классы для описания таблиц.

```
from sqlalchemy.orm import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    age = Column(Integer)

# Создаём таблицы в БД
Base.metadata.create_all(engine)
```

🔷 3. Работа с данными (CRUD)

Для работы с БД нужен **Session** (сессия).

✅ Создание сессии

```
from sqlalchemy.orm import sessionmaker

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

✅ Добавление данных

```
new_user = User(name="Alice", age=25)
session.add(new_user)
session.commit()
```

✅ Чтение данных

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

✅ Поиск по фильтру

```python
user = session.query(User).filter_by(name="Alice").first()
print(user.id)
```

✅ Обновление данных

```python
user = session.query(User).filter_by(name="Alice").first()
user.age = 30
session.commit()
```

✅ Удаление данных

```python
user = session.query(User).filter_by(name="Alice").first()
session.delete(user)
session.commit()
```

🔷 4. Работа с отношениями

В SQLAlchemy можно описывать связи **(One-To-Many, Many-To-Many и др.)**.

🔗 Один ко многим (One-To-Many)

Пример: у одного пользователя может быть несколько статей.

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

class Article(Base):
    __tablename__ = "articles"

    id = Column(Integer, primary_key=True)
    title = Column(String, nullable=False)
    user_id = Column(Integer, ForeignKey("users.id"))

    user = relationship("User", back_populates="articles")

User.articles = relationship("Article", order_by=Article.id,
back_populates="user")

user = User(name="Bob", age=40)
article1 = Article(title="First Post", user=user)
article2 = Article(title="Second Post", user=user)

session.add(user)
session.commit()
```

🔷 **Итоги**

SQLAlchemy — мощная ORM с поддержкой сложных отношений.

Но **в больших проектах** стоит комбинировать ORM с “чистым” SQL для оптимизации.

ДЗ - 28

```
📌 Задача 1:
Создайте модель User, которая содержит следующие поля:
• id (Integer, Primary Key)
• name (String, уникальный, не NULL)
• email (String, уникальный, не NULL)
• age (Integer, может быть NULL)

Ваши задачи:
1. Создайте таблицу users в базе данных (PostgreSQL).
2. Напишите функцию add_user(name, email, age), которая добавляет нового
пользователя в БД.
3. Напишите функцию get_all_users(), которая возвращает список всех
пользователей.
4. Напишите функцию get_user_by_email(email), которая находит пользователя
по email.
5. Напишите функцию update_user_age(email, new_age), которая обновляет
возраст пользователя.
6. Напишите функцию delete_user(email), которая удаляет пользователя из БД.
```