|  |  |
| --- | --- |
|  |  |

**Дипломная работа**

тема:

***Сравнение производительности и удобства использования различных ORM (Object-Relational Mapping) библиотек: Django ORM, Tortoise ORM и SQLAlchemy***

2024

# Содержание

[Содержание 1](#_Toc177421558)

[Введение 2](#_Toc177421559)

[Структура проекта 3](#_Toc177421560)

[Структура базы данных 4](#_Toc177421561)

[1. Django\_ORM 5](#_Toc177421562)

[2. SQLAlchemy\_ORM 10](#_Toc177421563)

[3. Tortoise\_ORM 15](#_Toc177421564)

[Заключение и выводы 16](#_Toc177421565)

[Приложение 1. Пример файловой структура приложения 20](#_Toc177421566)

[Приложение 2. Список основных библиотек, используемых в проекте 21](#_Toc177421567)

# Введение

Что такое ORM?

*согласно сайту Википедия:*

**ORM** (англ. Object-Relational Mapping, рус. *объектно-реляционное отображение*, или преобразование) – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных». Существуют как проприетарные, так и свободные реализации этой технологии.

**ORM** другими словами — это способ создания таблиц и отношений между ними с помощью классов в Python. Он также предоставляет систему для создания запросов и управления базой данных с помощью объектно-ориентированного кода вместо чистого SQL.. Причем знание работы с СУБД на прямую требуется минимальное .

## Преимущества:

* Упрощает разработку - устраняет необходимость в применении SQL-кода.
* Сокращает время разработки.
* Снижает затраты на разработку.
* Устраняет различия в SQL для конкретного поставщика – ORM может писать запросы используя синтаксис нужной СУБД, так что вам не придется углубленно изучать язык запросов разных

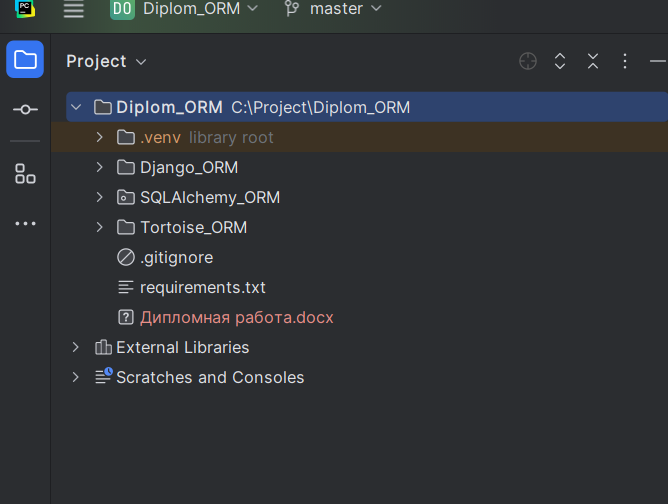
SQL систем.

## Недостатки:

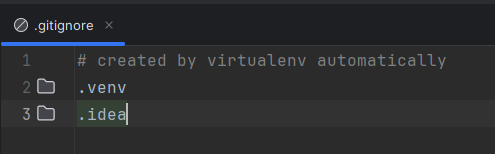
* Разработчики теряют понимание того, что на самом деле делает код - разработчик больше контролирует ситуацию, используя SQL.
* ORM имеет тенденцию быть медленным. Так как является посредником между программой и базой.
* ORM не может конкурировать с SQL-запросами при выполнении очень сложных запросов.

# Структура проекта

Согласно темы дипломной работы необходимо разработать приложения для работы с базами данных с использованием Django\_ORM, Tortoise\_ORM и SQLAlchemy , таким образом в одном проекте я создал три директории с соответствующими названиями и открывая ту или иную директорию, можно будет запускать интересующие нас приложения.



И сразу добавил ”.gitignore”, куда поместил папки ”.idea” и ”.venv” (содержится информация о виртуальном окружении (установленных библиотеках, модулях и так далее). Данное действие необходимо, чтобы эта информация не отправилась в репозиторий на github.



## Django\_ORM

Django – это популярный веб-фреймворк для создания масштабируемых и надежных веб-приложений. Одним из основных компонентов Django является система объектно-реляционного отображения (ORM), которая позволяет разработчикам взаимодействовать с базами данных с помощью кода на Python, а не писать необработанные SQL-запросы.

С помощью Django ORM можно создавать, получать, обновлять и удалять записи в базе данных, используя объекты и методы Python.

Кроме того, Django ORM предоставляет QuerySet API, который позволяет выполнять запросы к вашей базе данных. QuerySet –  это **набор методов для работы с коллекцией объектов из модели базы данных**. С его помощью можно получать, обновлять и удалять записи из базы данных.

Работа в Джанго начинается с его установки, можно в консоли написать:

* python -m pip install Django
* django-admin startproject {PAKET\_NAME} . (точка позволяет установить проект джанго в текущей директории)
* python manage.py startapp {name\_app} (создание приложений)
* manage.py — исполняет различные команды, в частности запускает приложение;
* settings.py — отвечает за настройки проекта; (мы должны настроить проект. Добавить приложение в INSTALLED\_APPS , путь в TEMPLATES, подключение к СУБД DATABASES у меня по умолчанию SQLlite .

Основные файлы принципа MVT это:

* .models.py – где хранятся модели наших сущностей связанные с СУБД.
* views.py - это то место, где мы размещается «логику» работы нашего приложения. Оно запрашивает информацию из модели, которую мы создали ранее, и передаёт её в шаблон.
* Templates -шаблоны для страниц сайта .

При необходимости можно создать своего суперюзера, командой в консоли:

python manage.py createsuperuser

Добавить модели в админ панель и там напрямую вносить данные.

admin.py – admin.site.register(ваша\_модель)

В таком формате довольно удобно управлять базой данных, где наглядно выбирается объект и вставляется туда, куда нужно, сохраняя при этом связи.

Создание таблиц базы данных:

python manage.py makemigrations Создает запросы на создание таблиц

python manage.py migrate инициализация БД

Приложение готово к работе.

Создается постоянная сессия (связь) с базой данных . Так же Джанго дает работать в интерактивной консоли для формирования запросов и работы с БД

python manage.py shell

from app.models import User добавили модель

User.objects.all() Сколько объектов у модели возвращает список

User.objects.create(name="Matvey",balance= 10.50,age= 25) создание

User.objects.filter(name="Matvey") отбор

User.objects.filter(name="Matvey").update(name="MAMAMA") отобрали поменяли

User.objects.count() количество

f = User.objects.get(id=2) присваиваем значение и с ним работаем

f.delete() например удаляем

а= User.objects.all()

a.delete() удалили всё

если есть связи между таблицами

from app.models import Task добавили еще одну модель

Task.objects.get(id=1).user.set((User.objects.get(id=1),))

метод set(objects) удобно применять при связи многие\_ко\_многим .

## SQLAlchemy\_ORM

SQLAlchemy — это Python-библиотека, которая позволяет работать с реляционными базами данных с помощью ORM. Реляционные базы данных хранят информацию в виде связанных между собой таблиц. К таким базам относят Oracle, Microsoft SQL Server, PostgreSQL и другие.

Всё благодаря тому, что библиотека состоит из двух компонентов — SQLAlchemy Core и SQLAlchemy ORM.

* SQLAlchemy Core — это как бы абстракция над SQL. Компонент умеет общаться с SQL и превращать Python-код в понятные для баз данных запросы с помощью SQL Expression Language. При этом он понимает, с каким именно видом хранилища данных взаимодействует.
* SQLAlchemy ORM — удобный для разработчика интерфейс для управления базами данных. Компонент позволяет работать с ними и коммитить запросы

Будем использовать FastAPI.

FastAPI — это современный, быстрый (высокопроизводительный) ,асинхронный в отличии от Django веб-фреймворк .

SQLAlchemy не имеет своей библиотеки миграции ,будем использовать Alembic

* Инициализация Alembic в проекте.

alembic init alembic {app/migrations}

* Настроить URL базы данных.

alembic.ini и заменить строку sqlalchemy.url = driver://user:pass@localhost/dbname на свой URL базы данных.

* В env.py

from app.backend.db import Base добавим базу

from app.models import User,Task добавим модели

target\_metadata = Base.metadata

Создание таблиц и БД:

Можно без Alembic

Base.metadata.create\_all(bind=engine) Создаст таблицы если не созданы. Но отслеживать изменения не будет

**Alembic**

* Сгенерировать скрипт миграции.

alembic revision --autogenerate -m "Initial migration".

Alembic проанализирует изменения в модели и сгенерирует скрипт миграции в директории alembic/versions.

* Применить миграцию.

Для этого нужно выполнить команду alembic upgrade head.

Далее, чтобы запустить сервер напишем команду:

python –m uvicorn main:app –reload (перезапуск сервера при внесенных изменениях)

http://127.0.0.1:8000/docs - попадаем в Swagger, в котором предлагается формировать или редактировать интересующие нас изделия.

Примеры запросов.

from sqlalchemy import select

statement = select(User).where(User.name == name) Создание запроса

db\_object = session.scalars(statement).one() и уже поиск в базе

statement - наш запрос, select выбирает все объекты из таблицы "users"

а можно добавлять через модель

def create\_db(db:Session,creates\_user):  
 db\_user = User(username=creates\_user.username,firstname=creates\_user.firstname,  
 lastname=creates\_user.lastname,age=creates\_user.age)  
 db.add(db\_user)  
 db.commit()  
 return "Successful"

Основные методы **sqlalchemy :**

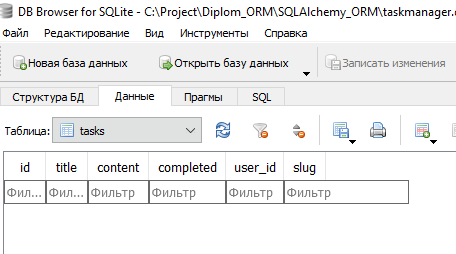
execute() пронимает запрос

query() принимает модель

методов много и они разнообразны :

| **Метод** | **Описание** |
| --- | --- |
| *all()* | Возвращает результат запроса (объект Query) в виде списка |
| *count()* | Возвращает общее количество записей в запросе |
| *first()* | Возвращает первый результат из запроса или None, если записей нет |
| *scalar()* | Возвращает первую колонку первой записи или None, если результат пустой. Если записей несколько, то бросает исключение MultipleResultsFound |
| *one* | Возвращает одну запись. Если их несколько, бросает исключение MutlipleResultsFound. Если данных нет, бросает NoResultFound |
| *get(pk)* | Возвращает объект по первичному ключу (pk) или None, если объект не был найден |
| *filter(\*criterion)* | Возвращает экземпляр Query после применения оператора WHERE |
| *limit(limit)* | Возвращает экземпляр Query после применения оператора LIMIT |
| *offset(offset)* | Возвращает экземпляр Query после применения оператора OFFSET |
| *order\_by(\*criterion)* | Возвращает экземпляр Query после применения оператора ORDER BY |
| *join(\*props, \*\*kwargs)* | Возвращает экземпляр Query после создания SQL INNER JOIN |
| *outerjoin(\*props, \*\*kwargs)* | Возвращает экземпляр Query после создания *SQL LEFT OUTER JOIN* |
| *group\_by(\*criterion)* | Возвращает экземпляр Query после добавления оператора GROUP BY к запросу |
| *having(criterion)* | Возвращает экземпляр Query после добавления оператора HAVING |

Наша база данных со всеми таблицами создаётся в taskmenedger.db, который мы можем открыть в обычном DB Browser и посмотреть всю структуру там.



## Tortoise\_ORM

**Tortoise ORM — это простой в использовании asyncio ORM (Object Relational Mapper), созданный на основе Django.**

* **Асинхронность**. Библиотека полностью асинхронная, что делает её идеальной для современных асинхронных фреймворков, таких как FastAPI .
* **Простота использования**. Синтаксис интуитивный и простой в настройке, похож на Django ORM.
* **Лёгкость**. Библиотека минималистичная, фокусируется на основных функциях.
* **Гибкость**. Поддерживает несколько баз данных, включая SQLite, PostgreSQL и MySQL

Разработчики указывают, что Tortoise ORM — все еще молодая библиотека, и могут появляться изменения, **ломающие существующий код** при обновлении. Стоит иметь это в виду при использовании ее в своих проектах.

Чтобы начать работать с tortoise ORM, нам нужно:

Подключиться к базе данных.

from tortoise import Tortoise

async def connectToDatabase():

await Tortoise.init(db\_url='sqlite://db.sqlite3',modules={'models': ['app.models'] )

Генерация схем с помощью tortoise

from tortoise import Tortoise, run\_async

from database.connectToDatabase import connectToDatabas

async def main():

await connectToDatabase()

await Tortoise.generate\_schemas()

if \_\_name\_\_ == '\_\_main\_\_':

run\_async(main())

либо через

@asynccontextmanager

async def lifespan(app: FastAPI):

await init\_db()

await Tortoise.generate\_schemas()

yield

await Tortoise.close\_connections()

pass

app = FastAPI(title='Dnevnik', lifespan=**lifespan**)

Lifespan в FastAPI — это функция, которая отвечает за жизненный цикл приложения.

В остальном работа с этой ORM очень похожа на Dgango ORM .

Методы и способы интуитивно понятны любому знакомому с Dgango ORM.

# Заключение и выводы

Итак, подводя итоги опять же повторим, что же такое ORM?

**ORM** (Object-Relational Mapping) — это технология, которая позволяет работать с базами данных, используя объектно-ориентированный подход. ORM создает объекты, которые представляют данные в базе данных, и позволяет работать с ними так же, как с обычными объектами в коде.

**Преимущества ORM**

***Простота:*** ORM предоставляет простой интерфейс для работы с базой данных, который может быть понятным любому программисту. ORM скрывает сложности SQL-запросов, позволяя работать с данными на более высоком уровне абстракции.

***Переносимость:*** ORM может работать с различными СУБД, что делает его более переносимым, чем SQL. Это позволяет разработчикам легко переносить свое приложение на другую СУБД без изменения кода.

***Сопровождаемость:*** ORM может значительно упростить сопровождение приложения, так как изменения в структуре базы данных могут быть внесены непосредственно в код ORM, а не в каждый SQL-запрос.

***Безопасность:*** ORM может предотвратить SQL-инъекции, поскольку ORM автоматически экранирует данные, которые передаются в базу данных.

**Недостатки ORM**

***Сложность:*** ORM может быть сложным для понимания, особенно для новых разработчиков. ORM требует определенных знаний и опыта, чтобы использовать его эффективно.

***Производительность:*** ORM может быть менее эффективным, чем работа с SQL напрямую. ORM должен обрабатывать запросы и преобразовывать их в SQL, что может замедлить производительность.

***Ограничения:*** ORM может иметь ограничения в отношении того, какие запросы могут быть выполнены. В случае, когда нужно выполнить сложный запрос или использовать специфичные функции базы данных, может потребоваться написание SQL-запроса напрямую.

**РЕЗУЛЬТАТЫ СРАВНЕНИЯ**

Django ORM является частью фреймворка Django и естественно не является конкурентом остальным ORM.

SQLAlchemy является намного более гибким инструментом и намного более сложным.

Может удовлетворить почти все потребности разработчика. Следующая ступень работа с СУБД напрямую.

Tortoise ORM достаточно молодая библиотека. Вобрала в себя удобство и простоту работы от Dgango ORM .Но отличается от нее асинхронностью и возможностью использоваться в разных проектах не на основе Dgango.

**Таким образом:**

Django ORM стоит особняком ,и не соперничает с другими.

SQLAlchemy решит почти все возможные задачи. Для продвинутых разработчиков дает возможность использовать более гибкие решения.

Tortoise ORM подходит для любителей Dgango и начинающих, а также для простых задач .

# 

# Приложение . Список основных библиотек, используемых в проекте

Django==5.1.1  
django-json-widget==2.0.1  
fastapi==0.114.2  
uvicorn==0.30.6  
Jinja2==3.1.4  
pydantic==2.9.0  
SQLAlchemy==2.0.34  
SQLAlchemy-Utils==0.41.2  
tortoise-orm==0.21.6  
tortoise~=0.1.1

Некоторые методы QuerySet API:

* **filter()**.  Фильтрует набор данных и возвращает выборку, которая соответствует фильтру, также в виде объекта QuerySet.
* **exclude()**. Фильтрует набор данных и возвращает выборку, которая не соответствует фильтру, в виде объекта QuerySet.
* **order\_by()**. Сортирует набор данных и возвращает отсортированную выборку в виде объекта QuerySet.
* **distinct()**. Возвращает объект QuerySet, который содержит только уникальные объекты.
* **all()**.  Возвращает объект QuerySet, который содержит все объекты из таблицы.
* **get()**. Возвращает один объект модели, который соответствует указанным ключевым аргументам, или вызывает исключение, если совпадения не найдено.
* **create()**. Создаёт новый объект и сохраняет его в базе данных.
* **update()**. Обновляет одно или несколько полей у всех объектов в наборе QuerySet.
* **delete()**. Удаляет все объекты в наборе QuerySet.

Полный список методов можно найти в документации