Дипломная работа

на тему:

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

**Автор работы:**

Иванчишина Марина Валерьевна

г. Москва

2024

Оглавление

[1. Введение 3](#_Toc182391656)

[1.1. Обоснование выбора темы 3](#_Toc182391657)

[1.2. Определение цели и задач работы 3](#_Toc182391658)

[2. Основные понятия и определения 3](#_Toc182391659)

[3. Методы и подходы к разработке 5](#_Toc182391660)

[3.1. Обзор ORM-библиотек 5](#_Toc182391661)

[*3.1.1. Django ORM 5*](#_Toc182391662)

[*3.1.2. SQLAlchemy 5*](#_Toc182391663)

[*3.1.3. Tortoise ORM 6*](#_Toc182391664)

[4. Проектирование приложения 6](#_Toc182391665)

[4.1. Обоснование выбора фреймворка Django для разработки веб-приложения на Python 6](#_Toc182391666)

[4.2. Разработка архитектуры 6](#_Toc182391667)

[4.3. Разработка 7](#_Toc182391668)

[5. Анализ и интерпретация результатов 8](#_Toc182391669)

[5.1. Интерпретация результатов 9](#_Toc182391670)

[5.2. Рекомендации по выбору ORM инструмента в зависимости от проекта 10](#_Toc182391671)

[6. Заключение 10](#_Toc182391672)

# Введение

# Обоснование выбора темы

«Кто владеет информацией – тот владеет миром», - сказал Натан Ротшильд, а Уинстон Черчилль добавил: «Кто управляет информацией, тот управляет миром».

Современный мир развивается настолько быстро, что человек – даже группа людей - не успевает накопить, запомнить, усвоить и проанализировать все поступающие ему данные в постоянно увеличивающемся потоке информации. Со временем встал вопрос об упорядоченном хранении информации. Конечно, библиотеки, книжные хранилища тоже можно назвать упорядоченными. Однако, они не могут обеспечить быстрый доступ к необходимым данным. Как ответ на такой запрос стала развиваться теория баз данных.

База данных – совокупность информации, хранимая в соответствии с определённой схемой. Базы данных предназначены для хранения и сбора данных. Определённая схема организации данных позволяет пользователям легко получать доступ к наборам хранимых данных и управлять ими. Наиболее современными и понятными на интуитивном уровне считаются реляционные базы данных.

Для быстрого обращения, использования и анализа данных, хранящихся в реляционных базах, был разработан свой особый язык – SQL (язык структурированных запросов).

В таком потоке информации редко какое приложение не использует базу данных в том или ином виде. По мере роста и усложнения приложения усложняются запросы к базе данных, увеличивается количество запросов. И использовать отдельный – специальный – язык вместо привычного для разработчика стека становится не очень удобным. Как следствие стали разрабатывать инструменты так называемого объектно-реляционного сопоставления, который, по сути, представляют реляционную базу данных как объект, с которым может работать привычный разработчику язык программирования.

# Определение цели и задач работы

Цель работы: создание приложения, визуализирующего скорость выполнения запросов к реляционной базе данных средствами различных ORM-инструментов.

Задачи работы:

1. Выбор фреймворка и обзор библиотек для разработки веб-приложений и работы с базами данных: анализ особенностей и выбор наиболее подходящих инструментов.
2. Разработка критериев сравнения инструментов.
3. Реализация приложений и проведение тестирования.
4. Написание дипломной работы, представляющую собой реализацию веб-приложения, визуализирующего результаты обработки запросов несколькими ORM-инструментами.

Цели и задачи работы направлены на получение представления об инструментах по работе с ORM.

# Основные понятия и определения

Основные понятия, используемые в настоящей работе:

|  |  |
| --- | --- |
| **Active Record (активная запись)** | - шаблон проектирования приложений, один из способов доступа к данным реляционных БД в объектно-ориентированном программировании, при котором таблицы БД обёрнуты в классы, а экземпляр класса представляет собой запись в таблице[[1]](#footnote-1). Active record - шаблон проектирования, в котором объект представляет собой комбинацию данных и логики (методов), связанной с этими данными. |
| **CPU (central processing unit, центральный процессов, центральное обрабатывающее устройство)** | Электронный блок либо интегральная схема, исполняющая машинные инструкции (код программ), главная часть аппаратного обеспечения компьютер или программируемого логического контроллера[[2]](#footnote-2). |
| **Data Mapper (преобразователь данных)** | - это шаблон проектирования, который разделяет объекты предметной области (бизнес-объекты) от схемы БД (отношений)[[3]](#footnote-3). |
| **Django ORM** | - часть фреймворка Django, позволяющая работать с реляционными СУБД с применением технологии ORM. |
| **ORM (объектно-реляционное сопоставление)** | - технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных»[[4]](#footnote-4). |
| **RAM (random access memory, оперативное запоминающее устройство, оперативная память)** | в большинстве случаев энергозависимая часть системы компьютерной памяти, в которой во время работы компьютера хранится выполняемый машинный код (программы), а также входные, выходные и промежуточные данные, обрабатываемые процессором[[5]](#footnote-5). |
| **SQL (Structured Query Language, язык структурированных запросов)** | **-**декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной БД, управляемой соответствующей СУБД. |
| **SQLAlchemy** | - библиотека, позволяющая синхронизировать объекты Python и объекты реляционных СУБД |
| **Tortoise ORM** | - асинхронная библиотека для работы с БД. |
| **База данных (БД)** | - модель некоторой предметной области, представляющая собой совокупность данных, которая характеризует её состояние; хранится в памяти компьютера, организована и доступна для оперирования пользователям в соответствии с определённой моделью данных[[6]](#footnote-6).  В настоящей работе будут под базами данных будут пониматься только базы данных, использующие реляционную модель данных. |
| **Библиотека** | - совокупность компьютерных программ, доступ к которым осуществляется по их именам (или индексам). Могут представлять собой библиотеки объектных модулей или библиотеки программ на исходном языке программирования[[7]](#footnote-7). |
| **СУБД (система управления базами данных)** | **-** комплекс программ, позволяющих создать БД и манипулировать данными (вставлять, обновлять, удалять и выбирать). Обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД[[8]](#footnote-8). |
| **Фреймворк** | - каркас (скелет) приложения, отражающий его структуру, но не содержащий реализации функциональных возможностей[[9]](#footnote-9). |

# Методы и подходы к разработке

# Обзор ORM-библиотек

# Django ORM

Django ORM – часть фреймворка Django, который в общих чертах будет описан ниже при анализе инструментов для разработки веб-приложений. Соответственно, Django ORM «наследует» те же свойства: шаблонизирует более-менее типичные запросы. Но при этом сложные и нестандартные запросы Django ORM не под силу. Следовательно, Django ORM удобен для создания более-менее стандартных баз данных при разработке стандартных же веб-приложений.

Django ORM использует active record, и она отличается большей степенью абстрактности. Как следствие, запросы составлять удобней и легче.

Django ORM легко интегрируется с интерфейсом администратора Django, позволяя легко выполнять операции по созданию, чтению, обновлению и удалению.

Немаловажную роль при выборе того или иного инструменты играет возможность найти поддержку и совет тех, кто использует этот инструмент. Т.к. Django ORM нельзя использовать отдельно от Django, то и сообщество этого инструмента ORM ограничено пользователями фреймворка Django.

Django ORM изначально был разработан для работы с такими распространёнными СУБД, как PostgreSQL, MySQL, SQLite, но в дальнейшем появилась возможность дополнительно настроить её на работу с другими СУБД. Однако, как это уже было сказано, Django ORM можно использовать только в веб-приложениях, разработанных на Django.

# SQLAlchemy

SQLAlchemy – автономная библиотека ORM. Она использует реализацию Data Mapper, где классы Python сопоставляются с таблицами базы данных, но конфигурация сопоставления более явная: Data Mapper действует как промежуточное звено между приложением и базой данных, передавая данные между ними, при этом соединение сохраняется независимым друг от друга. Эта гибкость позволяет разработчикам обрабатывать сложные взаимодействия с базами данных, интегрироваться с различными веб-фреймворками, более эффективно использовать базу данных, составляя сложные SQL-запросы. Одновременно с этим, однако, это усложняет работу с БД при помощи библиотекой.

Сообщество SQLAlchemy не ограничивается веб-разработчиками, т.к. библиотеку можно использовать для реляционного сопоставления не только при веб-разработке.

SQLAlchemy независима – она работает со многими типами СУБД.

# Tortoise ORM

Как пишут создатели данной библиотеки, Tortoise ORM вдохновлена Django ORM, т.е. ей свойственна особенность последней: использование реализации active record. Вместе с тем, при разработке моделей в Tortoise ORM также возможно использоваться Pydantic для валидации вводимых данных. И всё же, использование такого подхода лишает Tortoise ORM той гибкости при запросах к БД, которая свойственна, например, SQLAlchemy.

Основной же особенностью Tortoise ORM является её интеграция с asyncio, что позволяет библиотеке поддерживать асинхронные запросы, увеличивая производительность по сравнению с другими библиотеками, в особенности при больших загрузках приложений запросами пользователей.

# Проектирование приложения

# Обоснование выбора фреймворка Django для разработки веб-приложения на Python

Поскольку предполагается, что сравнение ORM-инструментов будет проводиться в виде веб-приложения, то следует выбрать и инструмент для разработки такого вида приложения.

Среди библиотек инструментов ORM, предлагаемых к сравнению, есть Django ORM. Данная библиотека является встроенной и неотъемлемой частью фреймворка Django, и инструменты библиотеки органично и неразрывно связаны с ним. Как следствие, Django ORM работает только вместе с самим фреймворком.

В отличие от Django ORM две другие библиотеки – SQLAlchemy и Tortoise ORM – могут работать с другими фреймворками для разработки веб-приложений. У них нет таких ограничений, как у Django ORM.

Так как предполагается сравнение именно библиотек ORM, а не работы различных фреймворков или архитектур сайтов, разработанных при помощи того или иного стека, то имеет смысл выбрать один фреймворк для разработки целевого веб-приложения. А раз одна из библиотек имеет жёсткие ограничения по работе, то и выбор особо невелик.

Поэтому для реализации веб-приложения по визуализации различий работы разных библиотек ORM будет выбран фреймворк Django.

# Разработка архитектуры

Целью работы является сравнение работы библиотек ORM. В связи с этим схема приложения должна содержать:

- базу данных, с которой библиотека ORM будет работать;

- фронтенд для визуализации результатов сравнения и

- бэкенд, который и будет делать основную часть работы приложения.

Существуют различные базы данных. ORM предполагает, что в проекте должна быть задействована СУБД, использующая реляционную модель, поддерживающая язык запросов SQL. Самым очевидным и простым вариантом может быть SQLite. Использование различных СУБД в проекте представляется неоправданным, т.к. сравниваются именно библиотеки по работе с СУБД, а не сами СУБД.

Для разработки фронтенда используются встроенные инструменты фреймворка Django: создается HTML-страница с визуализации результатов сравнения в виде таблицы. Для более показательного представления данных возможно использование CSS – языка описания внешнего вида веб-страницы.

Разрабатывать интерфейс с каким-либо расширенным функционалом не представляется целесообразным, т.к. не предполагается сравнение функциональности библиотек ORM при обработке запросов от внешних пользователей.

Разработка бэкенда представляет собой основную часть работы, функционала приложения, т.к. работа ORM и организация взаимодействия с базами данных является частью именно разработки бэкенда.

Архитектура не предусматривает тестирования и логирования, т.к. в приложение участие пользователя сведено к минимуму и практически исключены ошибки ввода данных.

# Разработка

Разработка была разделена на несколько этапов, которые, в основном, соотносятся с архитектурой веб-приложения.

Первым – подготовительным – этапом можно назвать первоначальную настройку проекта с использованием фреймворка Django: импортирование фреймворка, создание приложения, прописывание маршрутов HTML-страницы, базы данных, регистрация приложения.

Второй этап – создание и заполнение базы данных. Здесь были созданы модели ORM. На стадии первичной разработки было использовано «автозаполнение» 2 таблиц. Между записями не были настроены связи. В последующем предполагается воспользоваться одной из «тренировочных» баз данных, взятой из открытых источников.

Третий этап – реализация серверной логики. На данном этапе были реализованы 3 подхода к обработке 9 основных типов запросов к базе данных: с использованием встроенной Django ORM, SQLALchemy и Tortoise. Для более очевидного сравнения производительности библиотек каждый из запросов повторяется по 100 раз. В результатах представлено суммарное время выполнения каждого запроса. Также реализована передача результатов на фронтенд. Позднее был добавлен сбор информации о загрузке CPU и использовании RAM при каждом из запросов – также с выводом результатов на фронтенд.

Четвертый этап – фронтенд. Создана HTML-страница с использованием стилей CSS для визуализации результатов обработки в виде таблицы, наглядно демонстрирующей скорость обработки запросов (производительность) при помощи каждой из сравниваемых библиотек.

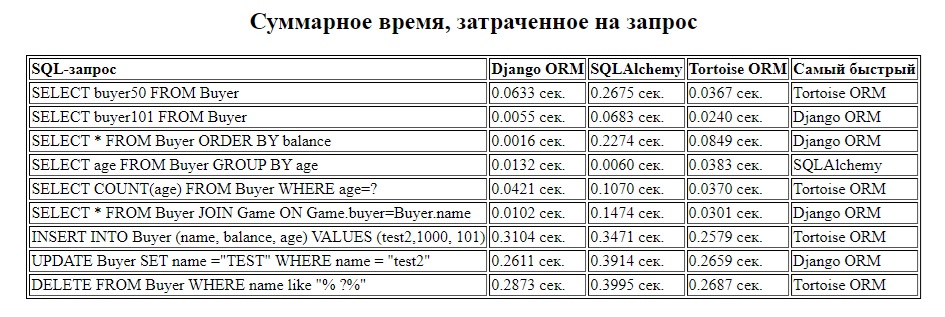


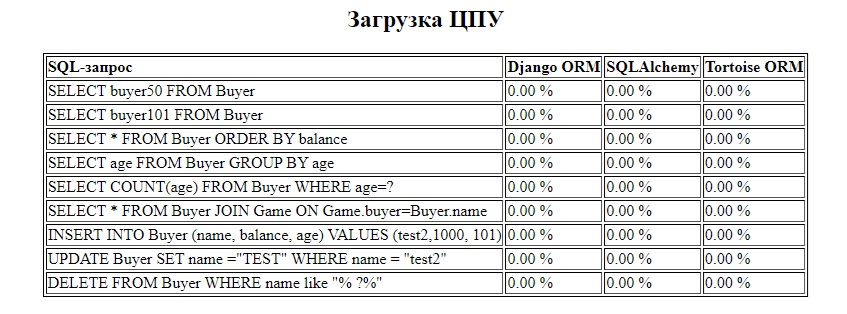
Рисунок . Файловая структура проекта

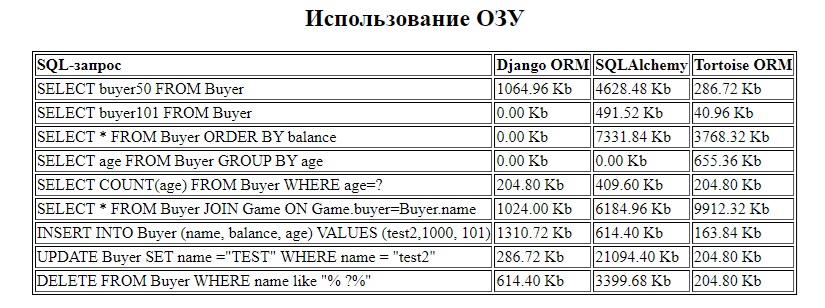
# Анализ и интерпретация результатов

Результаты работы программы

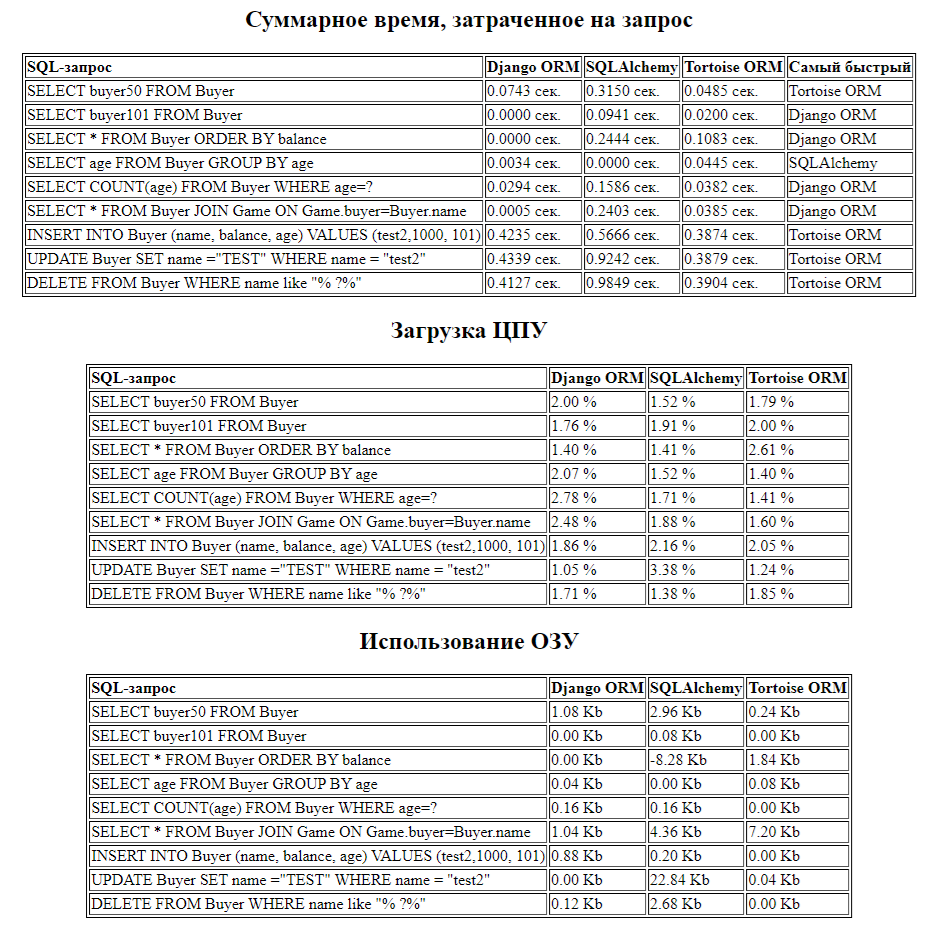
Запуск 1 (каждый запрос был сделан 100 раз, интервал для подсчета загрузки ЦПУ = 0):







Запуск 2 (каждый запрос был сделан 100 раз, интервал для подсчета загрузки ЦПУ = 0,1с):



# Интерпретация результатов

Как видно по результатам нескольких запусков программы, что среди лидеров по скорости обработки запросов к БД Django ORM и Tortoise ORM, в то время как SQLAlchemy лучше справляется со «сложными» запросами (запросы с группировкой, выборка с объединением) при отсутствии связанных полей.

При этом нужно учитывать то, что БД создана посредством Django ORM, данная БД использует модели именно этой ORM, т.е. является как бы «знакомой и родной» БД для Django ORM. Также следует учесть, что создатели Tortoise ORM при её разработке были «вдохновлены» Django ORM.

Каких-то особенностей или закономерностей в нагрузке на процессоры (CPU) и память (RAM) выявить не удалось. Возможно, следует более детально оценить результаты исследований на бóльшем массиве данных с более разветвленной структурой БД. Однако, работа с запросами к БД, – по крайне мере, не сложными – не предполагает особых вычислений, что, в свою очередь, не влечёт какой-то значимой разницы в нагрузке на CPU и RAM.

# Рекомендации по выбору ORM инструмента в зависимости от проекта

Выбор того или иного инструмента зависит от исходных данных, целей и задач, которые нужно достичь.

Если планируется разработка небольшого веб-приложения с небольшим количеством пользователей (а, следовательно, и запросов к несложной и неразветвленной БД), то использование встроенной в фреймворк разработки веб-приложений Django ORM можно назвать вполне адекватным выбором: его достаточно легко использовать, разработать модели, подключить БД. Такое приложение не потребует больших ресурсов (технических) на обработку запросов.

При масштабировании веб-приложения уже следует более тщательно подходить к выбору ORM инструмента: проанализировать количество обращений, технические возможности аппаратуры (не каждая машина может поддержать асинхронные обращения на должном уровне), а также архитектуру БД (объем данных, отношения между таблицами, количество таблиц).

Также следует отдать предпочтение не встроенному ORM инструменту (в данном случае, SQLAlchemy, Tortoise ORM), если предстоит работать с уже существующим проектом (БД), т.к. встроенный может «не понять» структуры БД, и придется переделывать архитектуру БД.

# Заключение

Проектирование и разработка приложения для сравнения производительности 3 ORM были завершены в соответствии с заданием. Приложение визуализирует собранную и обработанную информацию и даёт первичные выводы.

На основе собранной информации можно сделать определённые выводы для принятия решения об использовании той или иной библиотеки для работы с БД разрабатываемого приложения.

При реализации была попытка также собрать данные по загрузке аппаратных мощностей, однако интерпретировать эти данные на настоящем этапе не представляется возможным. При необходимости, в будущем, можно будет постараться реализовать сбор более точных данных.

1. https://ru.wikipedia.org/wiki/ActiveRecord. [↑](#footnote-ref-1)
2. https://ru.wikipedia.org/wiki/Центральный\_процессор [↑](#footnote-ref-2)
3. https://teletype.in/@konkin86/orm-pattern [↑](#footnote-ref-3)
4. https://ru.wikipedia.org/wiki/ORM [↑](#footnote-ref-4)
5. https://ru.wikipedia.org/wiki/Оперативная\_память [↑](#footnote-ref-5)
6. Большая российская энциклопедия (https://bigenc.ru/c/baza-dannykh-eadd27). [↑](#footnote-ref-6)
7. Большая российская энциклопедия (https://bigenc.ru/c/biblioteka-programm-e53a02). [↑](#footnote-ref-7)
8. Большая российская энциклопедия (https://bigenc.ru/c/sistema-upravleniia-bazami-dannykh-98ed12). [↑](#footnote-ref-8)
9. Большая российская энциклопедия (https://bigenc.ru/c/freimvork-bea961). [↑](#footnote-ref-9)