УДК 004.652

ОПТИМИЗАЦИЯ РАБОТЫ С ДАННЫМИ В ВЕБ-ПРИЛОЖЕНИЯХ: АНАЛИЗ ИНСТРУМЕНТОВ БАЗ ДАННЫХ В DJANGO.

Ефремова В.А., студент гр. ИТб-201, IV курс

Научный руководитель: Асанов С.А., старший преподаватель

Кузбасский государственный технический университет
имени Т.Ф. Горбачева

г. Кемерово

Веб-разработка стала неотъемлемой частью современного мира, и для упрощения процесса создания веб-приложений и веб-сайтов разработчики обращаются к использованию фреймворков, представляющих собой комплексные наборы инструментов и библиотек. Фреймворк Django для Python является одним из наиболее популярных инструментов в этой области. В данной статье мы рассмотрим фреймворк Django и проанализируем его возможности работы с базами данных, а также сравним различные подходы к работе с базами данных в рамках данного фреймворка.

Фреймворк Django славится готовыми модулями и надстройки, которые сильно упрощают работу. Он помогает быстрее создавать веб-приложения, чем нежели писать код с нуля и включает все необходимые инструменты для создания веб-приложения, а также предоставляет множество компонентов, начиная от аутентификация пользователей и заканчивая системой управления базами данных.

Фреймворк Django основан на архитектурном принципе MVT (Model-View-Template, модель – представление – шаблон). В соответствии с этим подходом код приложения делится на данные (модели), логику (представления) и пользовательский интерфейс (шаблоны).

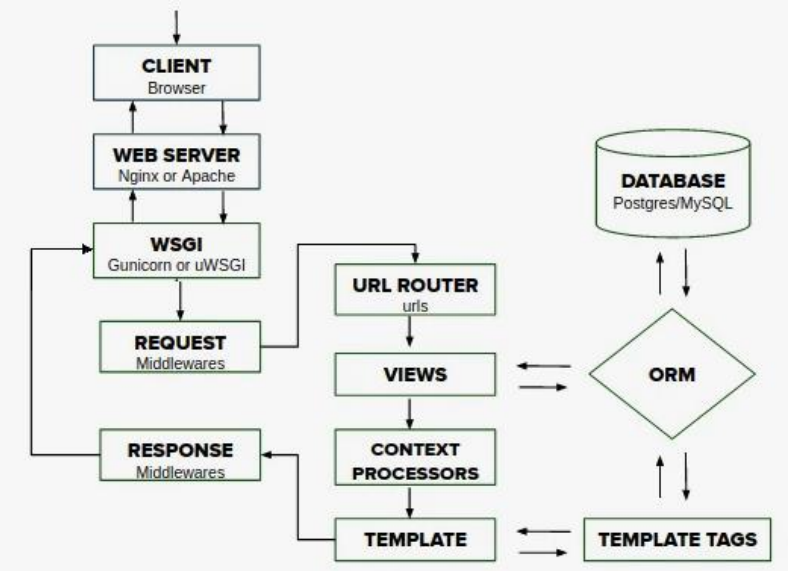

Рисунок 1 – Структурная схема архитектуры Django.

Одним из основных компонентов работы с базой данных в нём является использование модели данных. Модель представляет собой класс Python, который определяет структуру данных приложения и способы их хранения в базе данных. Django обеспечивает поддержку различных типов баз данных, включая SQLite, PostgreSQL, MySQL и другие.

Конечно, для создания баз данных в Python можно воспользоваться так называемой «ручной» работой с базами данных, для которой можно использовать библиотеки наподобие SQLite3 или SQLAlchemy. С SQLite3 можно напрямую взаимодействовать с базами данных SQLite, используя SQL запросы и обрабатывая результаты.

Преимущества данной работы:

1. Гибкость: При использовании прямых SQL запросов вы имеете полный контроль над тем, что происходит в базе данных. Это может быть полезно в случаях, когда вам нужно написать сложные или оптимизированные запросы.
2. Производительность: В некоторых случаях прямые SQL запросы могут быть более эффективными по сравнению с методами работы с БД в Django, особенно при выполнении сложных операций или запросов.

```python
import sqlite3

# Подключение к базе данных
conn = sqlite3.connect('example.db')
cursor = conn.cursor()
# Создание таблицы
cursor.execute('''CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY, name TEXT)''')
# Вставка данных
cursor.execute("INSERT INTO users (name) VALUES ('Victoria')")
conn.commit()
# Получение данных
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()
for row in rows:
    print(row)
```

Рисунок 2 – Пример работы с базой данных SQLite3.

«Ручная» работа с базами данных даёт больше контроля, но требует написания большего объёма кода для выполнения простых задач, в отличие от Django, преимуществами которого являются:

1. Универсальность: Существует огромное количество дополнительных модулей, которые позволяют реализовать на Django веб-приложения с любой функциональностью и для любой сферы.
2. Оптимизация производительности: Встроенные механизмы кэширования, обработки статических файлов и оптимизации запросов позволяют настроить Django приложение на максимальную производительность.
3. Простота интеграции: Возможность интеграции с чем угодно.
4. Безопасность: Надежная встроенная защита от внедрения SQL-кода, подделки межсайтовых запросов.
5. Скорость разработки: На процесс разработки приложений и веб-сайтов требуется меньше времени, чем при использовании других платформ – благодаря первоклассной документации и огромному количеству готовых модулей.

Выделяют несколько методов работы с базами данных в Django. Рассмотрим некоторых из них:

1. Django ORM (Object-Relational Mapping, объектно-реляционное отображение) – таблицы в БД описываются Python классами, в которых описаны все атрибуты, а выполнение запроса к БД – это вызов методов данного класса. Каждая запись в таблице – объект данного класса и автоматическое создание таблиц и связей между ними в базе данных на основе классов из модели.
2. Raw SQL Queries – возможность выполнения прямых SQL запросов к базе данных с помощью методов `django.db.connection.cursor()`. Этот

способ полезен, когда нужно выполнить сложные или оптимизированные SQL запросы, которые неудобно или невозможно сформулировать через ORM.

3. Stored Procedures – возможность использования хранимых процедур базы данных для выполнения сложных вычислений или операций, которые лучше выполнять на стороне базы данных.
4. Raw SQL Manager – выполнения сырых SQL запросов к базе данных на уровне моделей.

Сравнение преимуществ и недостатков каждого метода приведено в следующей таблице:

Таблица 1 – Сравнение методов работы с БД в Django

Название метода	Преимущества	Недостатки
1. Django ORM	Упрощение разработки: ORM позволяет работать с базой данных на уровне объектов, что делает код более читаемым и понятным Безопасность: предотвращение возможности создания уязвимостей из-за некорректного формирования SQL запросов. Переносимость: ORM позволяет абстрагироваться от конкретной СУБД. Вы можете легко изменить используемую базу данных, не затрагивая основной код вашего приложения.	Производительность: ORM иногда может быть менее эффективным, чем написание оптимизированных SQL запросов, особенно при выполнении сложных операций или взаимодействии с большим объемом данных. Ограничения: ORM может быть ограничивающим в случае нестандартных или сложных запросов, которые сложно или невозможно выразить через ORM.
2. Raw SQL Queries	Эффективность: Позволяет писать оптимизированные SQL запросы, что может повысить производительность при выполнении сложных или специфических операций.	Усложняет разработку: Требует знания SQL и может увеличить сложность кода и поддержку. Безопасность: Может быть уязвимым к SQL инъекциям, если не используются параметризованные запросы.

	Гибкость: Даёт полный контроль над выполнением запросов и позволяет использовать специфичные функции баз данных.	
3. Stored Procedures	Повышение производительности: Хранимые процедуры могут уменьшить сетевой трафик и увеличить производительность. Централизация логики: Позволяет централизовать бизнес-логику на стороне базы данных.	Поддерживаемость: Хранимые процедуры могут быть сложны для отладки и поддержки. Зависимость от конкретной СУБД: Хранимые процедуры могут сделать ваше приложение зависимым от конкретной СУБД.
Raw SQL Manager	Такие же, как и у Raw SQL Queries	

Не смотря на приведённые преимущества и недостатки, выбор конкретного способа зависит от конкретной задачи и предпочтений веб-разработчика. Однако, в большинстве случаев ORM в Django является предпочтительным способом работы с базами данных из-за своей удобной абстракции и интеграции с фреймворком.

В контексте Django, ORM (Object-Relational Mapping) является наиболее востребованным и автоматизированным подходом к работе с базами данных. Он предоставляет удобный интерфейс для выполнения операций CRUD (Create, Read, Update, Delete) без необходимости писать ручные SQL запросы.

```python
from django.db import models
from django.shortcuts import reverse


class Person(models.Model):
    choices = (
            ('M', 'Male'),
            ('F', 'Female')
        )
    name = models.CharField(max_length=140)
    sex = models.CharField(max_length=1, choices=choices)
    age = models.IntegerField()

    def get_absolute_url(self):
        return reverse('myurl', kwargs={'id': self.id, 'name': self.name}
```

Рисунок 3 – Пример использования Django ORM.

Таким образом, главное преимущество Django заключается в использовании постоянного соединения с базой данных, что улучшает производительность, позволяя не создавать новое подключение к базе данных при каждом запросе, а преимущества Django ORM по сравнению с другими методами работы с базами данных включают автоматическую генерацию миграций для изменения структуры базы данных, безопасность (защиту от SQL инъекций) и простоту использования. ORM уменьшает необходимость вручную писать SQL запросы и упрощает разработку. Так что, если нет конкретной необходимости использовать более низкоуровневые подходы, то Django ORM будет наиболее востребованным и автоматизированным способом работы с базами данных в Django.

Список литературы:

1. Продвинутые концепции Django ORM: [Электронный ресурс] URL: https://www.almabetter.com/bytes/tutorials/django/advanced-django-orm-concepts (дата обращения: 26.03.2024)
2. Django – что это такое для чего нужно: обзор и установка фреймворка для Python: [Электронный ресурс] URL: https://proglib.io/p/kurs-django-chast-1-django-chto-eto-obzor-i-ustanovka-freymvorka-struktura-proekta-2023-07-25?ysclid=lufsjh2zhj505365891 (дата обращения: 26.03.2024)
3. ORM и основы работы с базами данных: [Электронный ресурс] URL: https://proglib.io/p/kurs-django-chast-2-orm-i-osnovy-raboty-s-bazami-dannyh-2024-01-29?ysclid=lufk6fiinj927384836 (дата обращения: 27.03.2024)
4. Документация Django: [Электронный ресурс] URL: https://docs.djangoproject.com/en/5.0/topics/install/ (дата обращения 28.03.2024)
5. Дронов В. А. Django 2.1. Практика создания веб-сайтов на Python. - СПб.: БХВ-Петербург, 2019. – 672 с. – ISBN 978-5-9775-4058-2.