Отчёт

Оптимизация производительности представлений системы

Оглавление

[Цели 3](#_Toc108791083)

[Задачи 3](#_Toc108791084)

[Описание работы 4](#_Toc108791085)

[Представление «Lessons» 4](#_Toc108791086)

[Состояние до изменений 4](#_Toc108791087)

[Внесённые изменения 5](#_Toc108791088)

[Состояние после изменений 5](#_Toc108791089)

[Представление «Products» 5](#_Toc108791090)

[Состояние до изменений 5](#_Toc108791091)

[Внесённые изменения 6](#_Toc108791092)

[Состояние после изменений 6](#_Toc108791093)

[Итоги 6](#_Toc108791094)

# Цели

Использовать django-debug-toolbar

Использовать инструменты оптимизации

# Задачи

Задача состоит в том, чтобы ускорить работу сайта. Критерием будет количество запросов на странице (в большинстве случаев это подходит, иногда с уменьшением количества запросов увеличивается время ответа от страниц, поэтому можно еще обращать внимание на время загрузки страницы)

Пробуем оптимизировать максимальное количество страниц на сайте, чем больше, тем лучше для тренировки. Если какую-то страницу оптимизировать не удается, ничего страшного бывают сложные непонятные ситуации.

Дополнительно приложить отчет в виде таблицы (скрин или документ). В таблице 4 колонки:

Адрес страницы

Кол-во запросов до оптимизации

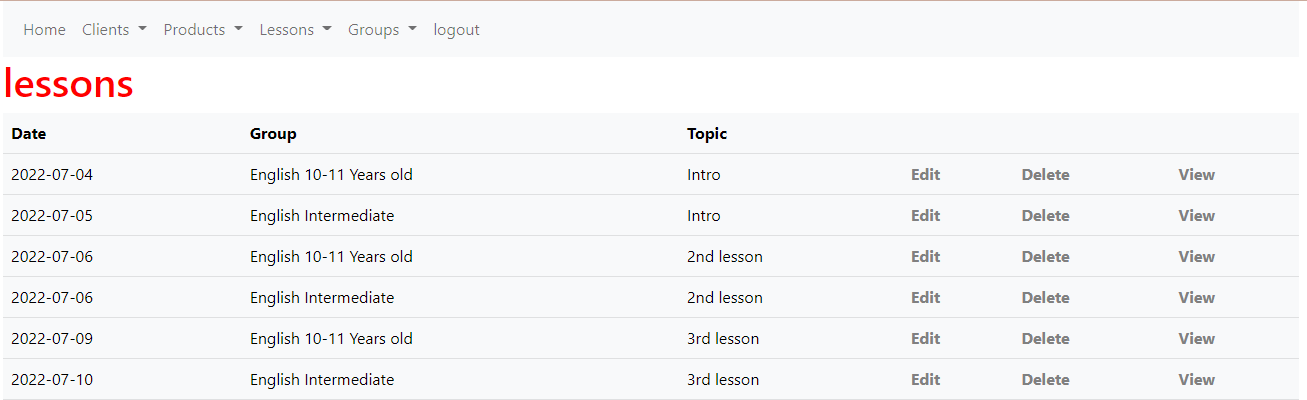
Кол-во запросов после оптимизации

Средство оптимизации (кратко, например select\_related, cached\_property, with, кэширование, любые другие) кэширование - желательно использовать только в крайнем случае и аккуратно

# Описание работы

## Представление «Lessons»

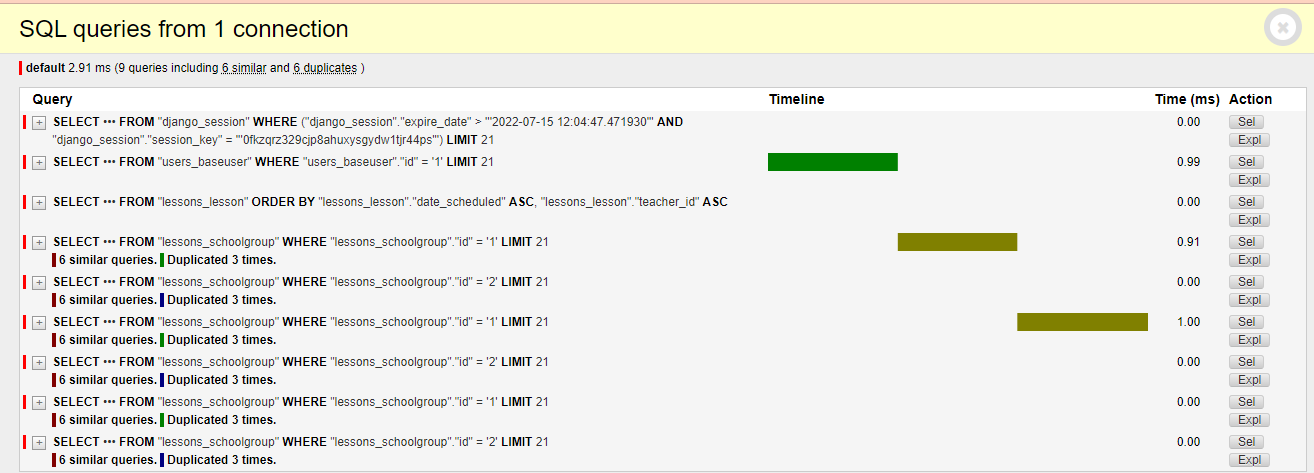
Представление является наследником базового класса ListView, служит для отображения списка проведённых занятий. Содержит информацию о дате занятия, группе, теме занятия, а также инструменты правки, просмотра деталей и удаления занятия.



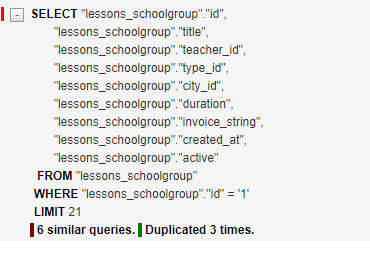
**Рис.1. Представление «Занятия»**

### Состояние до изменений

Форма анализа SQL запросов показывает, наличие повторяющихся запросов. Количество повторений запросов напрямую зависит от количества записей в таблице lessons.



**Рис.2. Список SQL выполняемых для отображения страницы до оптимизации**



**Рис.3. Повторяющийся запрос**

Класс «Lesson» имеет два внешних ключа «teacher» и «group», каждый раз представление обращается к базе, делая запрос по этим ключам, что несёт за собой дополнительные расходы ресурсов сервера и увеличение времени, необходимого на отображение результата.

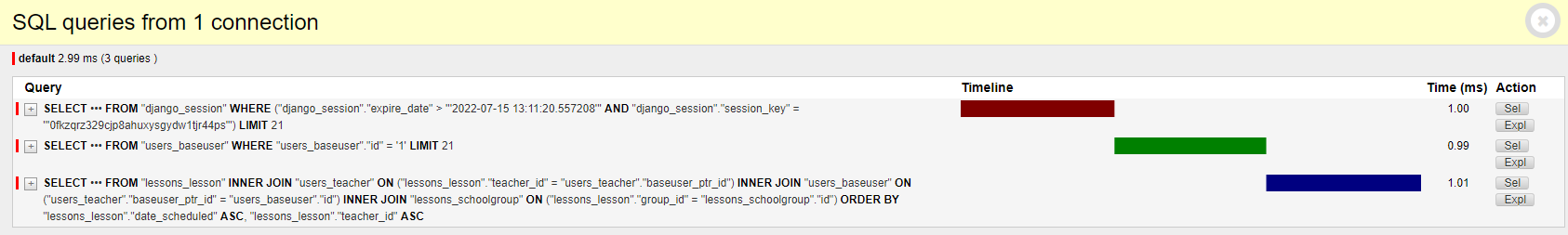
### Внесённые изменения

В классе «LessonListView» переопределили метод «get\_queryset» использовав метод «select\_related» от полученного «queryset» с тем, чтобы он подгружал в себя данные из связанных классов.

|  |
| --- |
| class LessonListView(ListView):  model = Lesson  page\_title = "lessons"  def get\_queryset(self):  qs = super().get\_queryset()  qs = qs.select\_related("group", "teacher")  return qs |

**Таб.1. Изменения в классе «LessonListView»**

### Состояние после изменений



**Рис.4. Список SQL выполняемых для отображения страницы после оптимизации**

Налицо улучшения в виде сокращения повторяющихся запросов к базе до нуля. Косвенно, можем сказать, что и скорость «отдачи» страницы увеличилась, в два раза, если исключить из оценки первые два запроса, связанные с сессиями и пользователем.

## Представление «Products»

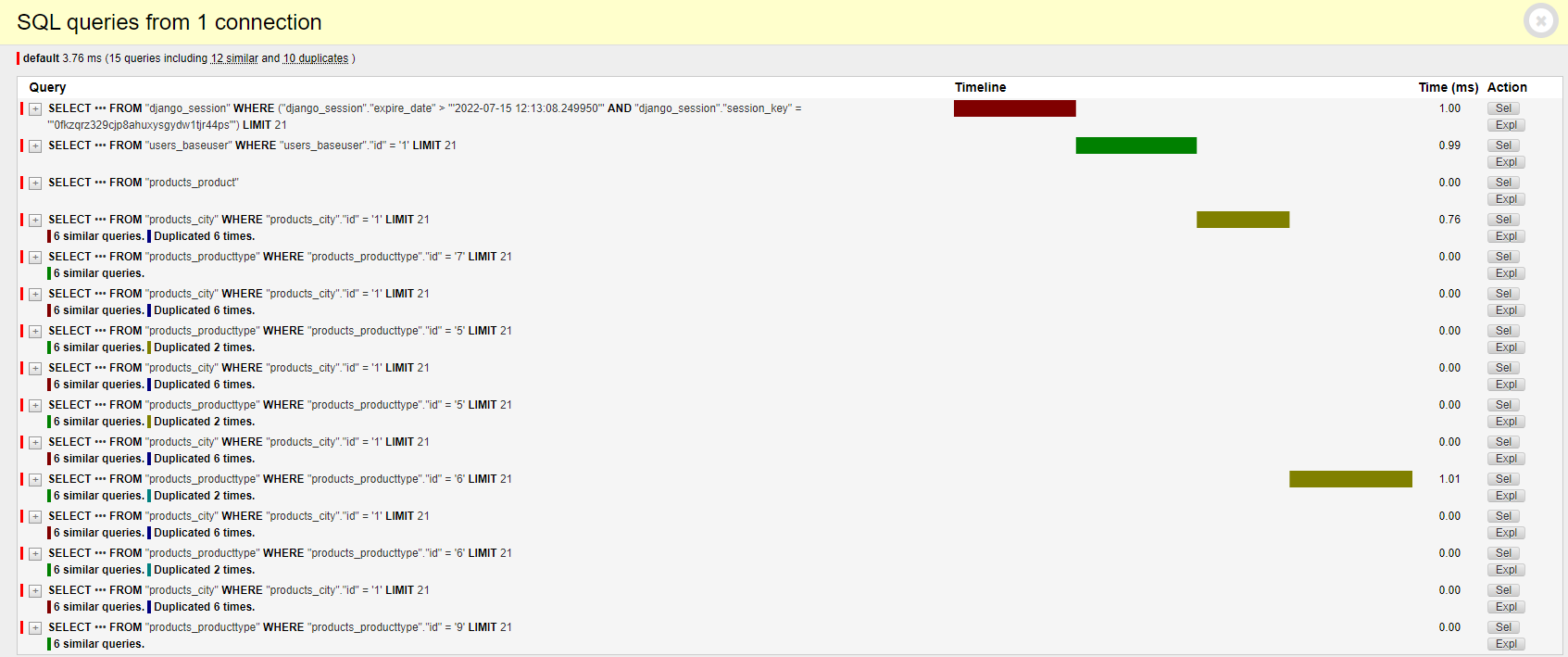
Отображает продукты, доступные в системе. Также содержит инструменты правки, вывода детальной информации и удаления продукта.

Изображение выглядит как текст

Автоматически созданное описание

**Рис.5. Представление «Продукты»**

### Состояние до изменений



**Рис.6. Список SQL выполняемых для отображения страницы до оптимизации**

|  |  |
| --- | --- |
| Изображение выглядит как текст  Автоматически созданное описание | Изображение выглядит как текст  Автоматически созданное описание |

**Рис.7. Повторяющиеся запросы**

### Внесённые изменения

В классе «LessonListView» переопределили метод «get\_queryset» использовав метод «select\_related» от полученного «queryset» с тем, чтобы он подгружал в себя данные из связанных классов.

|  |
| --- |
| class ProductListView(ListView):  model = Product  def get\_queryset(self):  qs = super().get\_queryset()  qs = qs.select\_related("city", "type")  return qs |

**Таб.2. Изменения в классе «ProductListView»**

### Состояние после изменений

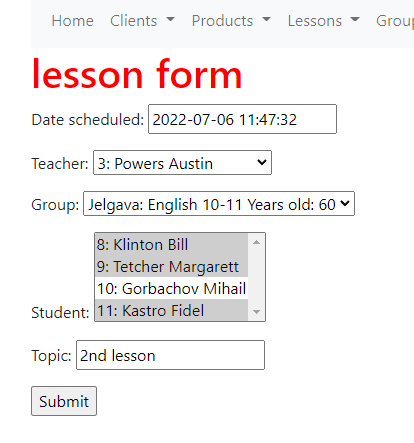
Изображение выглядит как текст

Автоматически созданное описание

**Рис.8. Список SQL выполняемых для отображения страницы после оптимизации**

# Форма правки информации о занятии

Данная форма служит для редактирования лекций. Имеет поля установки даты занятия, выбора преподавателя, группы, присутствовавших учеников и темы урока.



**Рис. 9. Форма правки занятия**

Список запросов к базе показал дублирующиеся запросы к таблице «products\_city». Напрямую в классе «Lesson» мы не связываемся с продуктом и городом. Но город и продукт присутствуют в классе «ScoolGroup» и связанным с ним и «Product». Т. е. имея три группы мы собираем для каждой из них группы, а для этих групп собираем продукты, доступные для каждой группы. Но и в этом случае мы ищем продукты не по городу, а по группе.

Отсюда вопрос- как обрабатывать подобные ситуации? В представлении мы не можем использовать prefetch \_selected, prefetch\_related так как они будут ссылаться на необъявленные атрибуты класса «city».

Скорее всего придётся менять структуру класссов.

Изображение выглядит как текст

Автоматически созданное описание

**Рис.8. Список SQL выполняемых для отображения страницы после оптимизации**

Изображение выглядит как текст

Автоматически созданное описание

**Рис.7. Повторяющийся запрос**

# Итоги

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Адрес страницы | Кол-во запросов до оптимизации | Кол-во запросов после оптимизации | Средство оптимизации |
| 1 | http://localhost:8000/ | 9 | 3 | select\_related добавлен в представление |
| 2 | http://localhost:8000/products/products | 15 | 3 | select\_related добавлен в представление |
| 3 | http://localhost:8000/update-lesson/1/ | 10 | 10 | Решение не найдено. |