Django Starter

Django ORM и административная панель

Introduction

Лазорык Михаил

Software developer, 3 года опыта

mykhailo.lazoryk

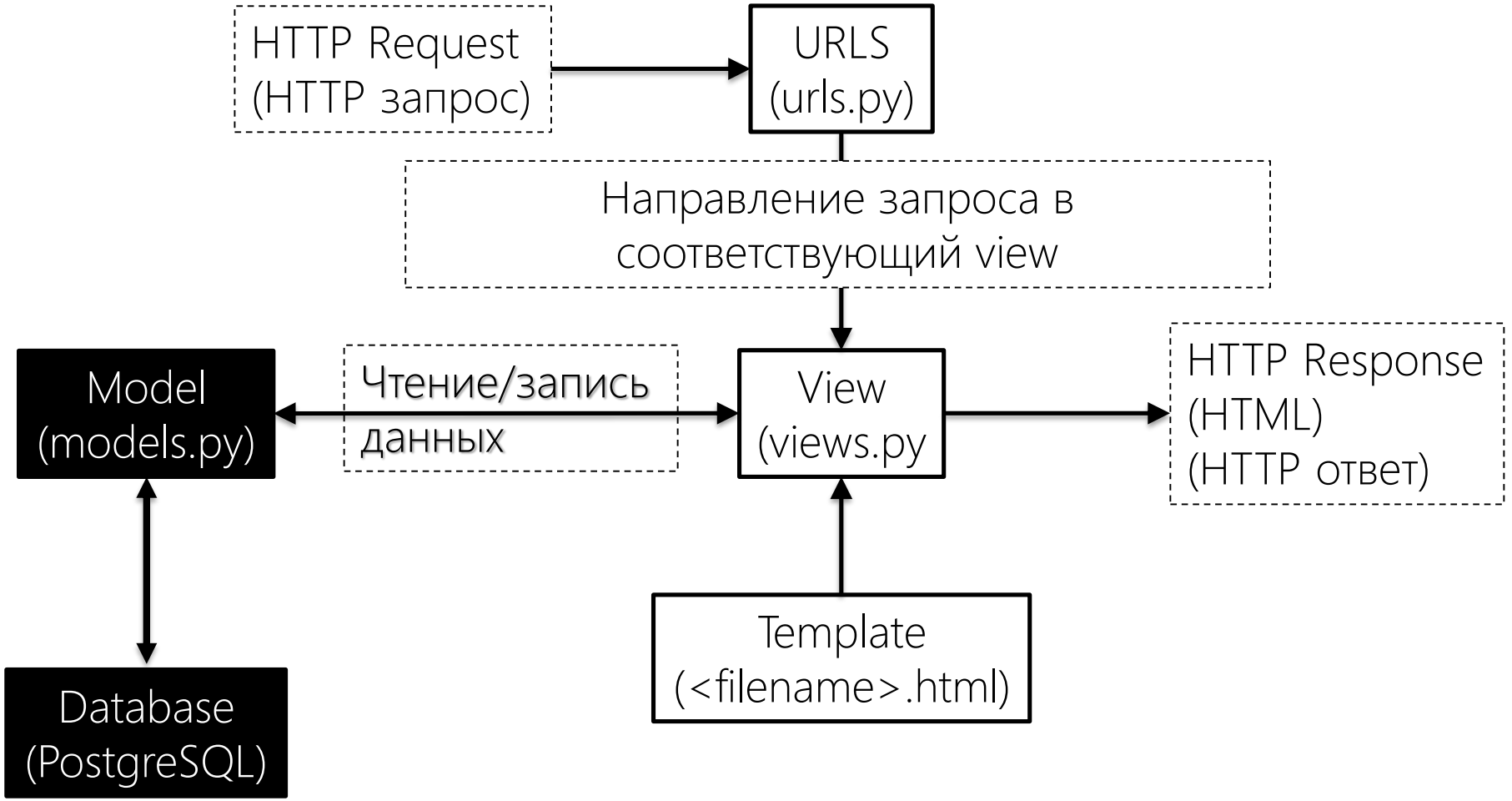
mykhailo-lazoryk

Django ORM и административная панель

План урока

1. QuerySet и работа с ним
2. Создавание записей с помощью моделей
3. Сортировка и фильтрация: детальный разбор
4. Сложные запросы
5. Администратор и админ панель

Структура файлов в реализации MVC в Django (MTV)



Работа с моделями

Вытягивание данных с базы данных происходит через Django ORM, которая возвращает QuerySet.

- 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».

- QuerySet, по сути, — список объектов заданной модели. QuerySet позволяет читать данные из базы данных, фильтровать и изменять их порядок.

QuerySet может быть создан, отфильтрован, нарезан и, как правило, передан без фактического запроса к базе данных.

Работа с моделями

QuerySets – “ленивы”, создание QuerySet не выполняет запросов к базе данных. Вы можете добавлять фильтры хоть весь день и Django не выполнит ни один запрос, пока QuerySet не вычислен.

```
>>> q = Entry.objects.filter(headline__startswith="What")
>>> q = q.filter(pub_date__lte=datetime.date.today())
>>> q = q.exclude(body_text__icontains="food")
>>> print(q)
```

Глядя на это можно подумать, что было выполнено три запроса в базу данных. На самом деле был выполнен один запрос - в последней строке (print(q)). Результат QuerySet не будет получен из базы данных, пока вы не “попросите” об этом. Когда вы делаете это, QuerySet вычисляется запросом к базе данных.

Когда вычисляются QuerySets

- Итерация. QuerySet является итеративным, и он выполняет свой запрос к базе данных при первой итерации по нему.
- Срезы. Как объяснено в Ограничении QuerySet, QuerySet может быть нарезан, используя синтаксис Python для срезов массивов. Срез QuerySet обычно возвращает другой, не вычисленный QuerySet, но Django выполнит запрос к базе данных, если вы используете параметр «step» синтаксиса среза, и вернет список. Срез QuerySet, который был вычислен, также возвращает список.
- repr(). QuerySet вычисляется, когда вы вызываете repr() для него. Это удобно для интерактивного интерпретатора Python, поэтому вы можете сразу увидеть свои результаты при интерактивном использовании API.
- len(). QuerySet вычисляется, когда вы вызываете len() для него. Это возвращает длину списка результатов.
- list(). Принудительное вычисление QuerySet, путем вызова list() для него.
- bool(). Тестирование QuerySet в логическом контексте, например, с использованием bool(), or, and или оператора if, вызовет выполнение запроса. Если есть хотя бы один результат, QuerySet равен True, иначе - False.
- Pickling/кэширование. При этих операциях будет выполнен запрос к базе данных.

Методы, которые возвращают новый QuerySet

- *filter()* - возвращает новый QuerySet, содержащий объекты, которые соответствуют заданным параметрам поиска.
- *exclude()* - возвращает новый QuerySet, содержащий объекты, которые не соответствуют указанным параметрам поиска.
- *order_by()* - По умолчанию результаты, возвращаемые QuerySet, упорядочиваются с помощью кортежа, заданного параметром ordering в классе Meta модели. Вы можете переопределить это для каждого QuerySet, используя метод order_by.
- *reverse()* - Используйте метод reverse() для изменения порядка, в котором возвращаются элементы набора запросов. Вызов reverse() во второй раз восстанавливает порядок в нормальном направлении.
- *distinct()* - Возвращает новый QuerySet, который использует SELECT DISTINCT в своем SQL-запросе. Это исключает повторяющиеся строки из результатов запроса.
- *values()* - Возвращает QuerySet, который возвращает словари, а не экземпляры модели, когда используется как итеративный.

Методы, которые возвращают новый QuerySet

- *values_list()* - это похоже на values(), за исключением того, что вместо возврата словарей он возвращает кортежи при повторении. Каждый кортеж содержит значение из соответствующего поля или выражения, переданное в вызов values_list() - поэтому первый элемент является первым полем и т.д.

- *dates()* - возвращает QuerySet, который вычисляет список объектов datetime.date, представляющих все доступные даты определенного вида в QuerySet.

- *datetimes()* - возвращает QuerySet, который оценивает список объектов datetime.datetime, представляющих все доступные даты определенного вида в содержимом QuerySet.

- *none()* - вызов none() создаст набор запросов, который никогда не вернет никаких объектов, и при доступе к результатам запрос не будет выполнен. Набор запросов qs.none() является экземпляром EmptyQuerySet.

- *all()* - возвращает копию текущего QuerySet (или подкласса QuerySet). Это может быть полезно в ситуациях, когда вы захотите передать либо менеджер модели, либо QuerySet и выполнить дальнейшую фильтрацию по результату. После вызова all() для любого объекта у вас обязательно будет QuerySet для работы.

Методы, которые возвращают новый QuerySet

- *union()* - использует оператор SQL UNION для объединения результатов двух или более QuerySet'ов.
- *select_related()* - возвращает QuerySet, который будет «следовать» отношениям внешнего ключа, выбирая дополнительные данные связанного объекта при выполнении своего запроса. Это повышение производительности, которое приводит к одному более сложному запросу, но означает, что дальнейшее использование отношений внешнего ключа не потребует запросов к базе данных.
- *extra()* - иногда синтаксис запроса Django сам по себе не может легко выразить сложное предложение WHERE. Для этих крайних случаев Django предоставляет модификатор extra() QuerySet - ловушку для вставки определенных предложений в SQL, генерируемый QuerySet.
- *get()* - возвращает объект, соответствующий заданным параметрам поиска, который должен быть в указанном формате.
- *create()* - удобный метод для создания объекта и сохранения всего за один шаг.
- *get_or_create()* - удобный метод для поиска объекта с указанным kwargs (может быть пустым, если в вашей модели есть значения по умолчанию для всех полей), создавая его при необходимости.
- *delete()* - удалить QuerySet.
- *update()* - обновить QuerySet.

Создание записей с помощью моделей

Для представления данных таблицы в виде объектов Python, Django использует интуитивно понятную систему: класс модели представляет **таблицу**, а экземпляр модели - **запись в этой таблице**.

Чтобы создать объект, создайте экземпляр класса модели, указав необходимые поля в аргументах, и вызовите метод **save**(), чтобы сохранить его в базе данных.

```
>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatles news.')
>>> b.save()
```

В результате выполнения этого кода будет создан **INSERT** SQL-запрос. Django не выполняет запросов к базе данных, пока не будет вызван метод **save**().
Метод **save**() ничего не возвращает.

Создание записей с помощью моделей

Пример использования метода create:

```
p = Person.objects.create(first_name="Bruce", last_name="Springsteen")
```

Bulk_create позволяет сохранить в базе данных множество объектов одним запросом:

```
>>> Entry.objects.bulk_create([ Entry(headline="Django 1.0 Released"), Entry(headline="Django 1.1 Announced"), ])
```

Важные нюансы:

- Метод модели save() не будет вызван, и сигналы pre_save и post_save не будут вызваны.
- Не работает с дочерними моделями при multi-table наследовании.
- Если первичный ключ модели это AutoField, его значение не будет получено и атрибут первичного ключа не будет установлен, как это делает метод save() .
- Не работает со связями многое-ко-многим.

Обновление записей с помощью моделей

Для сохранения изменений в объект, который уже существует в базе данных, используется **save**().

В данном примере изменяется название объекта b5 модели Blog и обновляется запись в базе данных:

```
>>> b5.name = 'New name'
>>> b5.save()
```

В результате выполнения этого кода будет создан **UPDATE SQL** запрос. Django не выполняет каких либо запросов к базе данных, пока не будет вызван метод save().

Обновление **ForeignKey** работает так же, как и сохранение обычных полей.

Фильтры

Фильтры полей – это “операторы” для составления условий SQL WHERE. Они задаются как именованные аргументы для метода **filter**(), **exclude**() и **get**() в QuerySet.

Фильтры полей выглядят как **field__lookuptype=value**. Используется двойное подчеркивание.

Поля, указанные при фильтрации, должны быть полями модели. Есть одно исключение - для поля ForeignKey можно указать поле с суффиксом _id. В этом случае необходимо передать значение первичного ключа связанной модели.

При передаче неверного именованного аргумента, будет вызвано исключение **TypeError**.

Примеры фильтров

- **exact** - точное совпадение. Если передано значение None, оно будет интерпретировано как SQL NULL.
- **contains** - регистрозависимая проверка на вхождение.
- **in** - проверяет на вхождение в список значений.
- **gt** - больше чем.
- **gte** - больше чем или равно.
- **lt** - меньше чем.
- **lte** - меньше чем или равно.
- **startswith** - регистрозависимая проверка того, что поле начинается с указанного значения.
- **endswith** - регистрозависимая проверка того, что поле оканчивается с указанного значения.
- **range** - проверка на вхождение в диапазон (включающий).

Примеры фильтров

- **search** - полнотекстовый поиск, который использует преимущества полнотекстового индекса. Работает как и contains, но значительно быстрее, благодаря полнотекстовому индексу.
- **year, month, day, week_day, hour, minute, second** - проверка времени для поля времени.
- **isnull** - принимает True или False, что соответствует SQL запросу IS NULL и IS NOT NULL, соответственно.
- **regex** - регистрозависимая проверка регулярным выражением.

Django предлагает удобный и понятный интерфейс для фильтрации по связанным объектам, самостоятельно заботясь о JOIN в SQL. Для фильтра по полю из связанных моделей используются имена связывающих полей, разделенных двойным нижним подчеркиванием, пока не будет достигнуто нужное поле.

Сортировка

Для сортировки используется метод order_by:

- По умолчанию, результат возвращаемый QuerySet, отсортирован по полям, указанным в аргументе ordering класса Meta модели. Вы можете переопределить сортировку, используя метод order_by.
- Знак “минус” в "-pub_date" указывает на “нисходящую” сортировку. Сортировка по возрастанию подразумевается по умолчанию.
- Чтобы отсортировать случайно, используйте "?". Заметка: запрос с order_by('?') может быть медленным и сильно нагружать базу данных, зависит от типа базы данных, которые используются.
- Для сортировки по полю из другой модели, используйте синтаксис, аналогичный тому, который используется при фильтрации по полям связанной модели.
- Нет способа указать должна ли сортировка учитывать регистр. Поэтому Django возвращает результат в таком порядке, в каком его вернула используемая база данных. Можно отсортировать по полю, преобразовав значение в нижний регистр, используя Lower.
- Если вы не хотите использовать сортировку, указанную по умолчанию, выполните метод order_by() без аргументов.

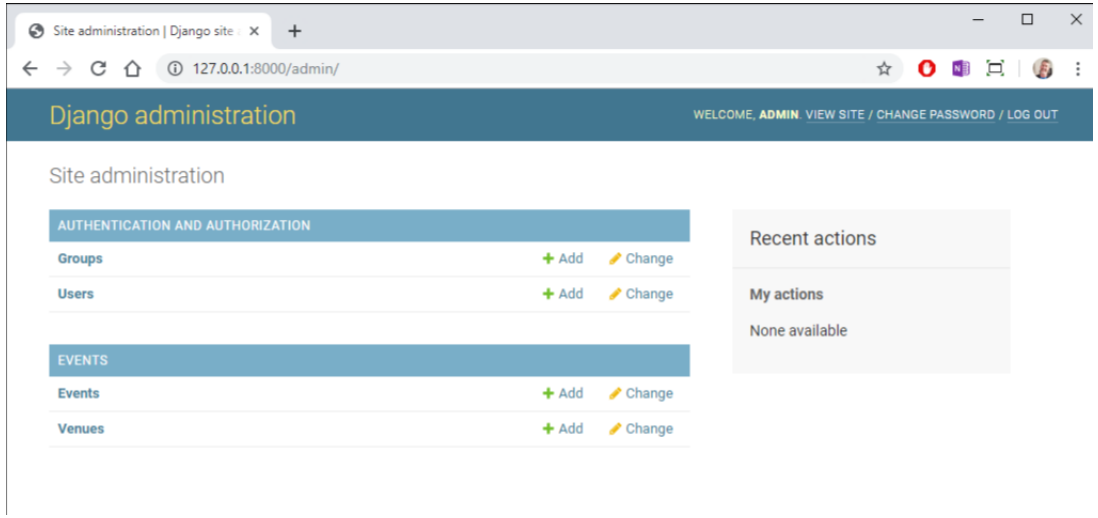
Сложные запросы

Именованные аргументы функции **filter()** и др. – объединяются оператором "AND". Если нужны более сложные запросы (например, запросы с оператором OR), можно использовать объекты Q.

- **Объект Q** (django.db.models.Q) – объект, используемый для инкапсуляции множества именованных аргументов для фильтрации. Аргументы определяются так же.
- Объекты Q могут быть объединены операторами & и |, при этом будет создан новый объект Q.
- Вы можете комбинировать различные объекты Q с операторами & и |, и использовать скобки. Можно использовать оператор ~ для отрицания (NOT) в запросе.
- Каждый метод для фильтрации, который принимает именованные аргументы (например, filter(), exclude(), get()), также может принимать объекты Q. Если вы передадите несколько объектов Q как аргументы, они будут объединены оператором "AND".
- Вы можете использовать одновременно объекты Q и именованные аргументы. Все аргументы (будь то именованные аргументы или объекты Q) объединяются оператором "AND". Однако, если присутствует объект Q, он должен следовать перед именованными аргументами.

Администратор и админ панель

Одна из сильных сторон Django – это автоматический интерфейс администратора. Он использует мета-данные модели, чтобы предоставить многофункциональный, готовый к использованию интерфейс для работы с содержимым сайта.



Администратор и админ панель

Перспективы расширения возможностей администратора:

- Настройка отображения списков.
- Регистрация класса ModelAdmin.
- Добавление фильтров списка.
- Формирование макета с подробным представлением.
- Встроенное редактирование связанных записей.
- Изменение внешнего вида.

План урока

1. QuerySet и работа с ним
2. Создавание записей с помощью моделей
3. Сортировка и фильтрация: детальный разбор
4. Сложные запросы
5. Администратор и админ панель

Проверка знаний

TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!

Лазорык Михаил
Software developer

tieto

django

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association