Вглубь запросов

Выполнение запросов. Объект QuerySet и его методы. Связывание моделей. Продолжаем дополнять сайт "About me"

[Почему мы НЕ будем пользоваться интерфейсом администратора](#_c713xnuw3xem)

[Добавляем собственные команды](#_gdgflytq07zg)

[Продвинутые запросы](#_pwm7z5jlzrg4)

[Получение всех объектов](#_gmbay6homdoo)

[Получение объектов через фильтры](#_ku2npds82jqm)

[Связанные модели](#_837jh1a8pgh)

[Рассмотрим на конкретном примере](#_pig737qqydwz)

[Домашнее задание](#_owpsqd9opgcf)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_uvp6qax5r1ok)

# 

# Почему мы НЕ будем пользоваться интерфейсом администратора

Django имеет встроенный интерфейс администратора. Он использует мета-данные модели, чтобы предоставить многофункциональный, готовый к использованию интерфейс для работы с содержимым сайта.

Это неплохой инструмент, тем не менее, мы не будем им пользоваться, на то есть несколько причин:

* Работа с БД через консоль дает более четкие понятия манипуляции с данными, нежели работа через графический интерфейс;
* Любая нестандартная операция с данными через админку требует очень глубокого изучения самой админки и создает больше сложностей, чем преимуществ;
* Я видел много больших проектов на django и ни один из них не использовал встроенную админку, все пишут свою. Причины разные, но практика показывает несостоятельность админки для реальных проектов.

# Добавляем собственные команды

На данный момент основная задача понять как всё работает. Для этого нужно экспериментировать. На первых порах вы часто будете ломать БД, а каждый раз заполнять её данными вручную - задача не из приятных. Чтобы избежать этой рутины, можно создать пару собственных скриптов, заполняющих базу тестовыми данными. Оформим данные скрипты как дополнительные комманды manage.py.

Проект может быть расширен собственными командами для manage.py. Для этого добавим в приложение каталог management/commands. Для каждого **модуля** в этом каталоге, который не начинается с подчёркивания, Django создаст соответствующую команду.

Структура проекта будет подобной:

mainapp/  
 /management  
 /commands  
 fill\_db.py  
 ishell.py  
 /migrations  
 \_\_init\_\_.py  
 \_\_init\_\_.py  
 models.py  
 tests.py  
 views.py

fill\_db - команда, которая будет заполнять БД тестовыми данными  
ishell - запуск интерактивной оболочки для работы с БД, с автоматическим импортом всех моделей

Для **модулей** должно быть соблюдено лишь одно требование - наличие в нём класса Command, который унаследован от BaseCommand или его потомков.

Наши модули будут запускаться без аргументов, поэтому мы их наследуем от NoArgsCommand

**fill\_db.py**

|  |
| --- |
| from django.core.management.base import BaseCommand, CommandError  from mainapp.models import Work, Hobby, Study  from datetime import date  class Command(BaseCommand):  help = 'Fill DB new data'  def handle(self, \*args, \*\*options):  works = [  {'organization': 'ООО "СтройКопай"', 'region': 'Москва', 'site': 'build\_dig.com', 'position': 'digger',  'duties': 'В основном копал...', 'period': 6},  {'organization': 'ИП "Иванушка"', 'region': 'Подмосковье', 'site': '', 'position': 'assistant',  'duties': '...', 'period': 4},  {'organization': 'GeekBrains', 'region': 'Москва', 'site': 'geekbrains.ru', 'position': 'teacher',  'duties': 'Пдготовка и преподавание курсов python/django', 'period': 12},  ]  hobbies = [  {'name': 'tourism'},  {'name': 'programming'},  {'name': 'digging -)'},  ]  studies = [  {'type': 'school', 'number': 36, 'study\_from': date(1990, 9, 1), 'study\_to': date(1998, 6, 1)},  {'type': 'lyceum', 'number': 66, 'study\_from': date(1998, 9, 1), 'study\_to': date(2001, 6, 1)},  {'type': 'university', 'number': 0, 'study\_from': date(2001, 9, 1), 'study\_to': date(2006, 8, 1)},  ]  for work in works:  work = Work(\*\*work)  work.save()  for hobby in hobbies:  hobby = Hobby(\*\*hobby)  hobby.save()  for study in studies:  study = Study(\*\*study)  study.save() |

При выполнении команды python manage.py fill\_db, будет выполнен данный скрипт. Конструктор модели сделан так, что может принимать именованные аргументы. \*\* - операция распаковки словаря.

Модели Hobby, Study вы должны были создать самостоятельно, и они могут отличаться от моих, если это так, просто исправьте данные согласно своей структуре.

Так выглядят мои модели

**models.py**

|  |
| --- |
| class Work(models.Model):  organization = models.CharField(verbose\_name='Организация', max\_length=32, unique=True)  region = models.CharField(verbose\_name='Регион', max\_length=32, blank=True)  site = models.CharField(verbose\_name='Сайт', max\_length=64, blank=True)  position = models.CharField(verbose\_name='Должность', max\_length=16)  duties = models.TextField(verbose\_name='Обязанности')  period = models.PositiveIntegerField(verbose\_name='Время работы', default=1)  class Hobby(models.Model):  name = models.CharField(verbose\_name='Название', unique=True, max\_length=32)  class Study(models.Model):  type = models.CharField(verbose\_name='Тип заведения', max\_length=16)  number = models.PositiveIntegerField(verbose\_name='Номер заведения')  address = models.CharField(verbose\_name='Адрес', default='no\_address', max\_length=32)  study\_from = models.DateField(verbose\_name='Учился с')  study\_to = models.DateField(verbose\_name='Учился до') |

Некоторым полям я добавил атрибут unique=True, чтобы сделать их уникальными, для предотвращения копий одинаковых объектов в базе. Если внимательно посмотрите, то заметите, что модель для учебного заведения явно реализована не удачно. Например, если вы учились в университете, то его не получится корректно задать (у ВУЗов, как правило, нет номеров, но есть названия). Это мы исправим чуть позже.

**ishell.py**

|  |
| --- |
| from django.core.management.base import BaseCommand  from IPython.terminal.embed import InteractiveShellEmbed  from django.db.models import Model  class Command(BaseCommand):  def handle(self, \*args, \*\*options):  modules = ('mainapp.models',)  for module in modules:  m = \_\_import\_\_(module, fromlist='non-empty')  for a in m.\_\_dict\_\_:  v = getattr(m, a)  if hasattr(v, '\_\_base\_\_') and issubclass(v, Model):  globals()[a] = v  InteractiveShellEmbed()() |

Этот скрипт запускает интерактивную консоль (аналогично python manage.py shell), дополнительно импортирую все классы моделей. Если данный код кажется вам непонятным, почитайте дополнительно по функциям getattr(), hasattr() и issubclass().

# Продвинутые запросы

Несколько удобных инструментов создано, база заполнена данными, время посмотреть, как работают запросы, и какие возможности предоставляет ORM-django.

Для получения объектов из базы данных создается QuerySet через Manager модели. Запомните, менеджер вызывается у объекта класса Модели, а не у его экземпляров.

QuerySet представляет выборку объектов из базы данных. Он может не содержать или содержать один или несколько фильтров – критерии для ограничения выборки по определенным параметрам. В терминах SQL (для тех, кто знаком с SQL), QuerySet - это оператор SELECT, а фильтры - условия, такие как WHERE или LIMIT.

Вы получаете QuerySet, используя Manager. Каждая модель содержит как минимум один Manager, и он называется objects по умолчанию. Обратиться к нему можно непосредственно через класс модели:

Work.objects или Hobby.objects

Manager - главный источник QuerySet для модели. Например, Work.objects.all() вернет QuerySet, который содержит все объекты Blog из базы данных.

## Получение всех объектов

Самый простой способ получить объекты из таблицы - это получить их все. Для этого используйте метод all() менеджера(Manager):

|  |
| --- |
| >>> works =Work.objects.all() |

Метод all() возвращает QuerySet всех объектов в базе данных.

## Получение объектов через фильтры

QuerySet, возвращенный Manager, описывает все объекты в таблице базы данных. Обычно вам нужно выбрать только часть всех объектов. Для этого вы можете изменить QuerySet, добавив условия фильтрации. Два самых простых метода изменить QuerySet - это:

|  |
| --- |
| filter(\*\*kwargs) |

Возвращает новый QuerySet, который содержит объекты, удовлетворяющие параметрам фильтрации.

|  |
| --- |
| exclude(\*\*kwargs) |

Возвращает новый QuerySet, содержащий объекты, которые не удовлетворяют параметрам фильтрации.

Например, для получения всех мест работы только в Москве:

|  |
| --- |
| >>> Work.objects.filter(region='Москва') |

Это аналогично:

|  |
| --- |
| >>> Work.objects.all().filter(region='Москва') |

Результат изменения QuerySet - это новый QuerySet и можно использовать цепочки фильтров.

Получаем все места работы в Москве, исключая места с сайтом geekbrains.ru

|  |
| --- |
| >>> Work.objects.filter(region='Москва').exclude(site='geekbrains.ru') |

После каждого изменения QuerySet, вы получаете новый QuerySet, который никак не связан с предыдущим QuerySet. Каждый раз создается отдельный QuerySet, который может быть сохранен и использован.

Очень важным является следующее свойство QuerySet: 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.

|  |
| --- |
| >>> works = Work.objects.filter(region='Москва') >>> works = works..exclude(site='geekbrains.ru') >>> print(works) |

Первые две инструкции не выполняют запроса к БД, и только когда вы пытаетесь использовать данные (в примере отобразить на экране), запрос будет выполнен.

Описание всех методов QuerySet [здесь](http://djbook.ru/rel1.7/ref/models/querysets.html#methods-that-return-new-querysets).

# Связанные модели

Основное преимущество реляционных баз данных - возможность добавлять связи для таблиц. Django предоставляет возможность использовать три самых используемых типа связей: многое-к-одному, многое-ко-многому и один-к-одному.

По умолчанию Django для каждой модели добавляем такое поле:

|  |
| --- |
| id = models.AutoField(primary\_key=True) |

Это автоинкрементный первичный ключ. Каждая модель должна иметь хотя бы одно поле с primary\_key=True. С помощью этого ключа и происходит связывание.

## Рассмотрим на конкретном примере

У нас есть организации, в которых мы работали. Сейчас мы храним только два атрибута об организации (её название и регион размещения), и храним мы эти данные в модели места работы вперемешку с должностью и временем работы…Не очень хорошо получается.

Отделим организации, в которых мы работали, в отдельные объекты с собственными атрибутами, а в местах работы (Work) будем хранить ссылки на объекты этих организаций. После небольших преобразований получаем:

models.py (остальная часть без изменений)

|  |
| --- |
| class Organization(models.Model):  name = models.CharField(verbose\_name='Название', max\_length=32)  region = models.CharField(verbose\_name='Регион', max\_length=32)  tax\_id = models.IntegerField(verbose\_name='ИНН')  site = models.CharField(verbose\_name='Сайт', max\_length=64, blank=True)  class Work(models.Model):  organization = models.ForeignKey(Organization, verbose\_name='Организация')  position = models.CharField(verbose\_name='Должность', max\_length=16)  duties = models.TextField(verbose\_name='Обязанности')  period = models.PositiveIntegerField(verbose\_name='Время работы', default=1)  …  … |

Для определения связи многое-к-одному используется поле ForeignKey. Вы используете его так же, как другие типы полей, добавляя как атрибут в модель.

Для ForeignKey необходимо указать обязательный позиционный аргумент: класс связанной модели. Django автоматически построит связи, вам об этом не нужно заботиться.

Обратите внимание: теперь все данные, касающиеся самой организации, хранятся в соответствующей модели.

Мы изменили структуру ваших моделей, изменим скрипт заполнения БД

fill\_db.py

|  |
| --- |
| class Command(NoArgsCommand):  help = 'Fill DB new data'  def handle\_noargs(self, \*\*options):  organizations = [  {'name': 'ООО "СтройКопай"', 'region': 'Москва', 'tax\_id': 123456, 'site': 'build\_dig.com'},  {'name': 'ИП "Иванушка"', 'region': 'Подмосковье', 'tax\_id': 666122, 'site': ''},  {'name': 'GeekBrains', 'region': 'Москва', 'tax\_id': 123456, 'site': 'geekbrains.ru'},  ]  works = [  {'organization': 'ООО "СтройКопай"', 'position': 'digger',  'duties': 'В основном копал...', 'period': 6},  {'organization': 'ИП "Иванушка"', 'position': 'assistant',  'duties': '...', 'period': 4},  {'organization': 'GeekBrains', 'position': 'teacher',  'duties': 'Подготовка и преподавание курсов python/django', 'period': 12},  ]  hobbies = [  {'name': 'tourism'},  {'name': 'programming'},  {'name': 'digging -)'},  ]  studies = [  {'type': 'school', 'number': 36, 'study\_from': date(1990, 9, 1), 'study\_to': date(1998, 6, 1)},  {'type': 'lyceum', 'number': 66, 'study\_from': date(1998, 9, 1), 'study\_to': date(2001, 6, 1)},  {'type': 'university', 'number': 0, 'study\_from': date(2001, 9, 1), 'study\_to': date(2006, 8, 1)},  ]  for organization in organizations:  organization = Organization(\*\*organization)  organization.save()  for work in works:  org\_name = work["organization"]  # Получаем организацию по имени  organization = Organization.objects.get(name=org\_name)  # Заменяем название организации объектом  work['organization'] = organization  work = Work(\*\*work)  work.save()  for hobby in hobbies:  hobby = Hobby(\*\*hobby)  hobby.save()  for study in studies:  study = Study(\*\*study)  study.save() |

При создании объекта с полем ForeignKey в качестве значения этому полю надо передавать ссылку на экземпляр соответствующего класса. Метод менеджера get() возвращает только один объект или выбрасывает исключение, если объект не найден. Подробнее про get() [здесь](http://djbook.ru/rel1.4/topics/db/queries.html#retrieving-a-single-object-with-get).

# Домашнее задание

Это последнее ДЗ по доработке проекта About Me. Начиная со следующего занятия, мы начнем более серьёзный проект:

1. В своей копии проекта отделить модель Организации, добавив еще пару-тройку полей (например, адрес и номер телефона). Пересоздать БД заново, и наполнить её информацией, используя скрипт.
2. На странице отображения мест работы, имена организаций сделать в виде гиперссылок, при переходе по которым должна показываться карточка данной организации с полной информацией об организации. Не забудте сделать отдельный шаблон, для отображения информации об организации.  
   P.S. Чтобы реализовать правильные url-адреса для организаций почитайте [этот](http://djbook.ru/rel1.7/topics/http/urls.html) раздел djangobook.
3. \*(усложненное) На странице “Работа” добавьте флаг(check-box), который позволит отбражать только последние три места работы. Если у вас три или меньше места работы, добавьте несколько произвольных мест для проверки данного фильтра.

Все проблемы и пути их решения подробно обсудим на следующем занятии. После следующего занятия я выложу свою полную реализацию проекта About Me.

# Дополнительные материалы

Все то, о чем сказано в методичке, но подробнее:

1. [manage скрипты](http://webnewage.org/2008/02/05/komandovat-paradom-budet-django/)
2. [Выполнение запросов](http://djbook.ru/rel1.7/topics/db/queries.html#making-queries)
3. [Методы QuerySet](http://djbook.ru/rel1.7/ref/models/querysets.html#methods-that-return-new-querysets)

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Собственные команды manage.py](http://djbook.ru/rel1.5/howto/custom-management-commands.html)
2. [Перевод документации](http://djbook.ru/)