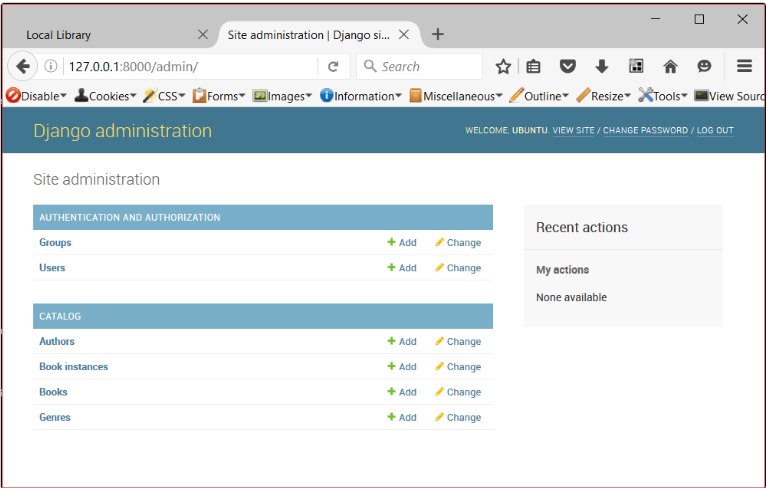
**Практическая работа: Использование моделей**

**Цель:**

Научится определять модели данных и выполнять управление моделью данных в проекте Django.

**Порядок выполнения:**

1. Подумайте о том, какие данные нам нужно хранить, и о взаимоотношениях между разными объектами. Мы знаем, что нам нужно хранить информацию о книгах (название, резюме, автор, язык, на котором написана книга, категория, ISBN) и что у нас может быть несколько доступных экземпляров (с уникальным глобальным идентификатором, статусом доступности и т. Д.). Нам может потребоваться хранить больше информации об авторе, чем просто их имя, и могут быть несколько авторов с одинаковыми или похожими именами. Мы хотим иметь возможность сортировать информацию на основе названия книги, автора, письменного языка и категории.
2. Постройте ER-диаграмму полученной модели данных на любой онлайн платформе, например https://app.diagrams.net/.
3. Откройте проект предыдущей практической работы и активируйте среду окружения.
4. Модели обычно определяются в приложении **models.py**Откройте файл models.py в вашем проекте
   1. Создайте класс Book:
      * Незабываем установить родительский класс(model.Model)
      * Далее опишите поля модели, например поле Название книги:  
        title = models.CharField(max\_length=200)
   2. Аналогично создайте остальные классы.
5. Выполните миграцию БД, чтобы добавить их в свою базу данных:  
   python manage.py makemigrations  
   python manage.py migrate
6. Добавьте некоторые данные о книгах, используя административную панель Django Admin:
   1. Откройте файл admin.py  (**/library/catalog/admin.py**).
   2. Зарегистрируйте модели путем добавления следующих команд:
      * Импортируйте свои модели:  
        from .models import список классов моделей через запятую
      * Зарегистрируйте каждую из них:  
        admin.site.register(Book)
   3. Чтобы войти в админ-панель, нам необходимо иметь учётную запись пользователя. Для просмотра и создания записей, пользователю также понадобится разрешение для управления всеми нашими объектами. Вы можете создать учётную запись "superuser", которая даёт полный доступ к сайту и все необходимые разрешения, используя **manage.py**:  
      python manage.py createsuperuser
   4. Запустите сервер
   5. Для входа в админ-панель откройте ссылку /admin (например http://127.0.0.1:8000/admin) и введите логин и пароль вашего нового суперпользователя.  
      В этой части сайта отображаются все наши модели, сгруппированные по установленному приложению. Вы можете кликнуть на названии модели, чтобы получить список всех связанных записей, далее можете кликнуть на этих записях, для их редактирования . Также можно непосредственно кликнуть на ссылку **Add**, расположенную рядом с каждой моделью, чтобы начать создание записи этого типа.  
      
   6. Кликните на ссылке **Add** справа от Books, чтобы создать новую книгу (появится диалоговое окно как на картинке внизу).
   7. Добавьте несколько книг, авторов и жанров (например, Фэнтези) в ваше приложение. Удостоверьтесь, что каждый автор и жанр включает пару различных книг.

**Теоретические сведения**

### [Определение модели](https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/Models#%D0%BE%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8)

Модели обычно определяются в приложении **models.py**. Они реализуются как подклассы django.db.models.Model, и могут включать поля, методы и метаданные. В приведённом ниже фрагменте кода показана «типичная» модель, названная MyModelName:

from django.db import models

from django.urls import reverse

class MyModelName(models.Model):

"""Типичный класс модели, производный от класса Model."""

# Поля

my\_field\_name = models.CharField(max\_length=20, help\_text='Введите описание поля')

# …

# Метаданные

class Meta:

ordering = ['-my\_field\_name']

# Methods

def get\_absolute\_url(self):

"""Возвращает URL-адрес для доступа к определенному экземпляру MyModelName."""

return reverse('model-detail-view', args=[str(self.id)])

def \_\_str\_\_(self):

"""Строка для представления объекта MyModelName (например, в административной панели и т.д.)."""

return self.my\_field\_name

В следующих разделах мы подробно рассмотрим каждый элемент внутри модели:

#### Поля

Модель может иметь произвольное количество полей любого типа - каждый представляет столбец данных, который мы хотим сохранить в одной из наших таблиц базы данных. Каждая запись (строка) базы данных будет состоять из одного значения каждого поля. Давайте рассмотрим приведённый выше пример:

my\_field\_name = models.CharField(max\_length=20, help\_text="Введите описание поля")

Наш вышеприведённый пример имеет одно поле, называемое my\_field\_name, типа models.CharField — что означает, что это поле будет содержать строки буквенно-цифровых символов. Типы полей назначаются с использованием определённых классов, которые определяют тип записи, которая используется для хранения данных в базе данных, а также критерии проверки, которые должны использоваться, когда значения получены из формы HTML (то есть, что составляет действительное значение). Типы полей также могут принимать аргументы, которые дополнительно определяют, как поле хранится или может использоваться. В этом случае мы даём нашему полю два аргумента:

* max\_length=20 — Указывает, что максимальная длина значения в этом поле составляет 20 символов.
* help\_text="Введите описание поля" — предоставляет текстовую метку для отображения, чтобы помочь пользователям узнать, какое значение необходимо предоставить, когда это значение должно быть введено пользователем через HTML-форму.

Имя поля используется для обращения к нему в запросах и шаблонах. В полях также есть метка, которая задаётся как аргумент (verbose\_name), либо выводится путём заглавной буквы первой буквы имени переменной поля и замены любых символов подчёркивания пробелом (например, my\_field\_name будет иметь метку по умолчанию My field name).

Порядок, в котором объявляются поля, будет влиять на их порядок по умолчанию, если модель отображается в форме (например, на сайте администратора), хотя это может быть переопределено.

##### Общие аргументы поля

Следующие общие аргументы могут использоваться при объявлении многих / разных типов полей:

* [help\_text](https://docs.djangoproject.com/en/2.1/ref/models/fields/#help-text): Предоставляет текстовую метку для HTML-форм (например, на сайте администратора), как описано выше.
* [verbose\_name](https://docs.djangoproject.com/en/2.1/ref/models/fields/#verbose-name): Удобочитаемое имя для поля, используемого в поле метки. Если не указано, Django выведет по умолчанию подробное название от имени поля.
* [default](https://docs.djangoproject.com/en/2.2/ref/models/fields/#default): Значение по умолчанию для поля. Это может быть значение или вызываемый объект, и в этом случае объект будет вызываться каждый раз, когда создаётся новая запись.
* [null](https://docs.djangoproject.com/en/2.2/ref/models/fields/#null): Если True, Django будет хранить пустые значения как NULL в базе данных для полей, где это уместно (CharField вместо этого сохранит пустую строку). По умолчанию используется значение False.
* [blank](https://docs.djangoproject.com/en/2.2/ref/models/fields/#blank): Если True, поле может быть пустым в ваших формах. По умолчанию используется значение False, что означает, что проверка формы Django заставит вас ввести значение. Это часто используется с null = True, потому что если вы хотите разрешить пустые значения, вы также хотите, чтобы база данных могла представлять их соответствующим образом.
* [choices](https://docs.djangoproject.com/en/2.2/ref/models/fields/#choices): Группа вариантов для этого поля. Если это предусмотрено, по умолчанию соответствующий виджет формы будет полем выбора с этими вариантами вместо стандартного текстового поля.
* [primary\_key](https://docs.djangoproject.com/en/2.2/ref/models/fields/#primary-key): Если True, задаёт текущее поле в качестве первичного ключа для модели (первичный ключ - это специальный столбец базы данных, предназначенный для однозначной идентификации всех разных записей таблицы). Если в качестве первичного ключа не указано поле, Django автоматически добавит для этой цели поле.

Есть много других вариантов - вы можете просмотреть [full list of field options here](https://docs.djangoproject.com/en/2.2/ref/models/fields/" \t "_blank).

##### Общие типы полей

Следующие общие аргументы могут использоваться при объявлении многих / разных типов полей:

* [CharField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.CharField) Используется для определения строк фиксированной длины от короткой до средней. Вы должны указать max\_length для хранения данных.
* [TextField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.TextField) используется для больших строк произвольной длины. Вы можете указать max\_length для поля, но это используется только тогда, когда поле отображается в формах (оно не применяется на уровне базы данных).
* [IntegerField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.IntegerField) это поле для хранения значений (целого числа) и для проверки введённых значений в виде целых чисел в формах.
* [DateField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.DateField) и [DateTimeField](https://docs.djangoproject.com/en/2.2/ref/models/fields/" \l "django.db.models.DateTimeField" \t "_blank) используются для хранения / представления дат и информации о дате / времени (как Python datetime.date и datetime.datetime, соответственно). Эти поля могут дополнительно объявлять (взаимоисключающие) параметры auto\_now=True (для установки поля на текущую дату каждый раз, когда модель сохраняется), auto\_now\_add (только для установки даты, когда модель была впервые создана) и по умолчанию (чтобы установить дату по умолчанию, которую пользователь может переустановить).
* [EmailField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.EmailField) используется для хранения и проверки адресов электронной почты.
* [FileField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.FileField) и [ImageField](https://docs.djangoproject.com/en/2.2/ref/models/fields/" \l "django.db.models.ImageField" \t "_blank) используются для загрузки файлов и изображений соответственно ( ImageField просто добавляет дополнительную проверку, что загруженный файл является изображением). Они имеют параметры для определения того, как и где хранятся загруженные файлы.
* [AutoField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.AutoField) - это особый тип IntegerField, который автоматически увеличивается. Первичный ключ этого типа автоматически добавляется в вашу модель, если вы явно не укажете его.
* [ForeignKey](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.ForeignKey) Используется для указания отношения «один ко многим» к другой модели базы данных (например, автомобиль имеет одного производителя, но производитель может делать много автомобилей). «Одна» сторона отношения - это модель, содержащая ключ.
* [ManyToManyField](https://docs.djangoproject.com/en/2.2/ref/models/fields/#django.db.models.ManyToManyField) используется для определения отношения «многие ко многим» (например, книга может иметь несколько жанров, и каждый жанр может содержать несколько книг). В нашем приложении для библиотек мы будем использовать их аналогично ForeignKeys, но их можно использовать более сложными способами для описания отношений между группами. Они имеют параметр on\_delete, чтобы определить, что происходит, когда связанная запись удаляется (например, значение models.SET\_NULL просто установило бы значение NULL)

Существует много других типов полей, включая поля для разных типов чисел (большие целые числа, малые целые числа, дробные), логические значения, URL-адреса, slugs, уникальные идентификаторы и другие «связанные с временем» сведения (продолжительность, время и т. д.). Вы можете просмотреть [full list here](https://docs.djangoproject.com/en/1.10/ref/models/fields/" \l "field-types" \t "_blank).

#### Метаданные

Вы можете объявить метаданные на уровне модели для своей модели, объявив класс Meta, как показано на рисунке.

class Meta:

ordering = ["-my\_field\_name"]

...

Одной из наиболее полезных функций этих метаданных является управление сортировка записей, возвращаемых при запросе типа модели. Вы можете сделать это, указав соответствия названия полей для сортировки, как показано выше. Порядок будет зависеть от типа поля (поля символов отсортированы в алфавитном порядке, а поля даты отсортированы в хронологическом порядке). Как показано выше, вы можете префикс имени поля минус-символом (-), чтобы изменить порядок сортировки.

Например, если мы решили сортировать книги по умолчанию:

ordering = ["title", "-pubdate"]

Книги будут отсортированы по алфавиту по названию, от A-Z, а затем по дате публикации внутри каждого названия, от самого нового до самого старого.

Другим распространённым атрибутом является verbose\_name, подробное имя для класса в единственной и множественной форме:

verbose\_name = "BetterName"

Другие полезные атрибуты позволяют создавать и применять новые «разрешения доступа» для модели (разрешения по умолчанию применяются автоматически), разрешить упорядочение на основе другого поля или объявить, что класс является «абстрактным» (базовый класс, для которого вы не можете создавать записи, и вместо этого будет создан для создания других моделей). Многие другие параметры метаданных управляют тем, какая база данных должна использоваться для модели и как хранятся данные (это действительно полезно, если вам нужно сопоставить модель с существующей базой данных). Полный список опций метаданных доступен здесь: [Model metadata options](https://docs.djangoproject.com/en/2.2/ref/models/options/" \l "model-meta-options" \t "_blank) (Django документация).

#### Методы

Модель также может иметь методы. Минимально в каждой модели вы должны определить стандартный метод класса для Python \_\_str \_\_ (), чтобы вернуть удобочитаемую строку для каждого объекта. Эта строка используется для представления отдельных записей на сайте администрирования (и в любом другом месте, где вам нужно обратиться к экземпляру модели). Часто это возвращает поле названия или имени из модели.

def \_\_str\_\_(self):

return self.field\_name

Другим распространённым методом включения в модели Django является get\_absolute\_url (), который возвращает URL-адрес для отображения отдельных записей модели на веб-сайте (если вы определяете этот метод, тогда Django автоматически добавит кнопку «Просмотр на сайте» на экранах редактирования записей модели на сайте администратора). Типичный шаблон для get\_absolute\_url () показан ниже.

def get\_absolute\_url(self):

"""

Returns the url to access a particular instance of the model.

"""

return reverse('model-detail-view', args=[str(self.id)])

**Примечание:** Предполагается, что вы будете использовать URL-адреса, например / myapplication / mymodelname / 2, для отображения отдельных записей для вашей модели (где «2» - это идентификатор для определённой записи), вам нужно будет создать URL-карту, чтобы передать ответ и идентификатор «Образцовое представление модели» (которое будет выполнять работу, необходимую для отображения записи). Вышеуказанная функция reverse () может «перевернуть» ваш URL-адрес (в приведённом выше примере с именем «model-detail-view»), чтобы создать URL-адрес правильного формата.

Конечно, для выполнения этой работы вам всё равно придётся писать сопоставление URL-адрес, просмотр и шаблон!

Вы также можете определить любые другие методы, которые вам нравятся, и вызывать их из вашего кода или шаблонов (при условии, что они не принимают никаких параметров).

### Управление моделью

После того, как вы определили свои классы моделей, вы можете использовать их для создания, обновления или удаления записей и для запуска запросов для получения всех записей или отдельных подмножеств записей. Мы покажем вам, как это сделать в учебнике, когда мы определяем наши представления, с кратким обзором.

#### Создание и изменение записей

Чтобы создать запись, вы можете определить экземпляр модели, а затем вызвать метод save ().

# Create a new record using the model's constructor.

a\_record = MyModelName(my\_field\_name="Instance #1")

# Save the object into the database.

a\_record.save()

**Примечание:** Если вы не указали какое-либо поле в качестве primary\_key, новая запись будет выдаваться автоматически, с идентификатором имени поля. Вы можете запросить это поле после сохранения указанной выше записи, и оно будет иметь значение 1.

Вы можете получить доступ к полям в этой новой записи с использованием синтаксиса точек и изменить значения. Вы должны вызвать save (), чтобы сохранить изменённые значения в базе данных.

# Access model field values using Python attributes.

print(a\_record.id) #should return 1 for the first record.

print(a\_record.my\_field\_name) # should print 'Instance #1'

# Change record by modifying the fields, then calling save().

a\_record.my\_field\_name="New Instance Name"

a\_record.save()

#### Поиск записей

Вы можете искать записи, соответствующие определённым критериям, используя атрибут объектов модели (предоставляемый базовым классом).

**Примечание:** Объяснение того, как искать записи, используя «абстрактную» модель и имена полей, может быть немного запутанным. В приведённом ниже обсуждении мы будем ссылаться на модель книги с полями названия и жанра, где жанр также является моделью с единственным именем в поле.

Мы можем получить все записи для модели как объект QuerySet, используя objects.all(). QuerySet - это итерируемый объект, означающий, что он содержит несколько объектов, которые мы можем перебирать / прокручивать.

all\_books = Book.objects.all()

Метод filter() Django позволяет отфильтровать возвращаемый QuerySet для соответствия указанному текстовому или числовому полю по конкретным критериям. Например, чтобы отфильтровать книги, содержащие слово «wild» («дикие») в заголовке, а затем подсчитать их, мы могли бы сделать следующее.

wild\_books = Book.objects.filter(title\_\_contains='wild')

number\_wild\_books = Book.objects.filter(title\_\_contains='wild').count()

Соответствующие поля и тип соответствия определяются в имени параметра фильтра, используя формат: field\_name\_\_match\_type (обратите внимание на двойное подчёркивание между заголовком выше). Выше мы фильтруем заголовок с учётом регистра. Есть много других типов совпадений, которые вы можете сделать: icontains (без учёта регистра), iexact (точное совпадение без учёта регистра), exact (точное совпадение с учётом регистра ) и in, gt (больше), startswith и т. д [смотреть полный список](https://docs.djangoproject.com/en/2.2/ref/models/querysets/#field-lookups)(Django Docs, [EN]).

В некоторых случаях вам нужно будет фильтровать поле, которое определяет отношение «один ко многим» к другой модели (например, ForeignKey). В этом случае вы можете «индексировать» поля в связанной модели с дополнительными двойными подчёркиваниями. Так, например, чтобы фильтровать книги с определённым жанровым рисунком, вам нужно будет указывать имя в поле жанра, как показано ниже:

books\_containing\_genre = Book.objects.filter(genre\_\_name\_\_icontains='fiction') # Will match on: Fiction, Science fiction, non-fiction etc.

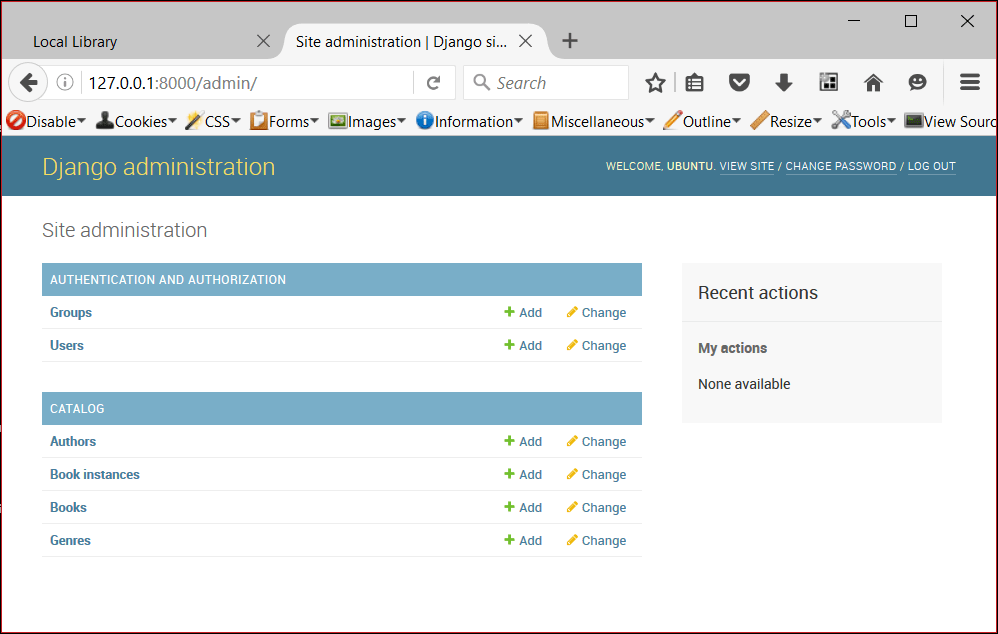
**Примечание:** Вы можете использовать символы подчёркивания (\_\_) для навигации по многим уровням отношений (ForeignKey / ManyToManyField) по своему усмотрению. Например, книга, имеющая разные типы, определяемая с использованием дополнительной связи «обложка», может иметь имя параметра: type\_\_cover\_\_name\_\_exact = 'hard'.

Существует гораздо больше возможностей для запросов, включая обратные поиски от связанных моделей, цепочки фильтров, возврат меньшего набора значений и т. д. Для получения дополнительной информации см. [Making queries](https://docs.djangoproject.com/en/2.2/topics/db/queries/" \t "_blank) (Django Docs, [EN]).

## **Вход в админ-панель и её использование**

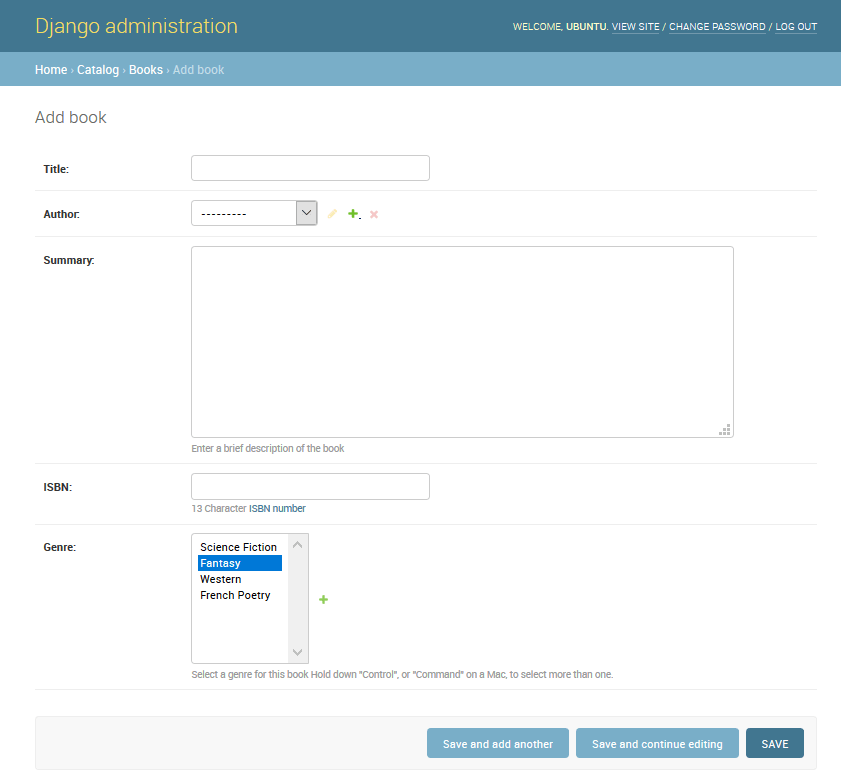
Для входа в админ-панель откройте ссылку /admin (например [http://127.0.0.1:8000/admin](http://127.0.0.1:8000/admin/)) и введите логин и пароль вашего нового суперпользователя (вас перенаправят на login-страницу и потом обратно на /admin после ввода всех деталей).

В этой части сайта отображаются все наши модели, сгруппированные по установленному приложению. Вы можете кликнуть на названии модели, чтобы получить список всех связанных записей, далее можете кликнуть на этих записях, для их редактирования . Также можно непосредственно кликнуть на ссылку **Add**, расположенную рядом с каждой моделью, чтобы начать создание записи этого типа.



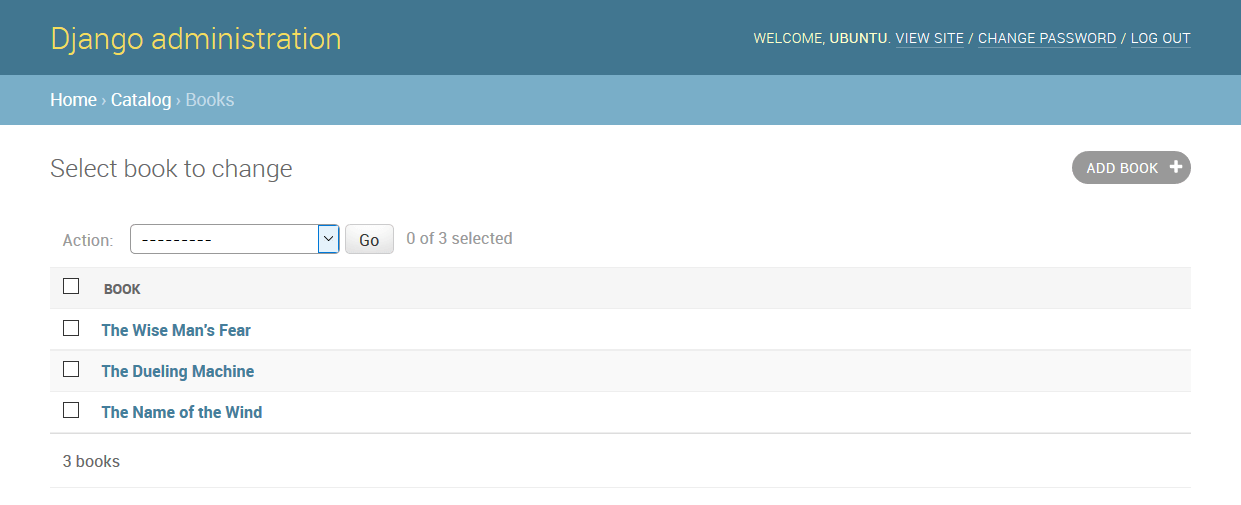
Кликните на ссылке **Add** справа от Books, чтобы создать новую книгу (появится диалоговое окно как на картинке внизу). Заметьте, что заголовок каждого поля - это тип используемого виджета, и help\_text (если есть) совпадает со значением, которое вы указали в модели.

Введите значение для полей. Вы можете создавать новых авторов или жанры, нажимая на значок "+ ", расположенный рядом с соответствующим полем (или выберите существующее значение из списков, если вы уже создали их). Когда вы закончили, нажмите на **SAVE,** **Save and add another**, или **Save and continue editing,** чтобы сохранить записи.



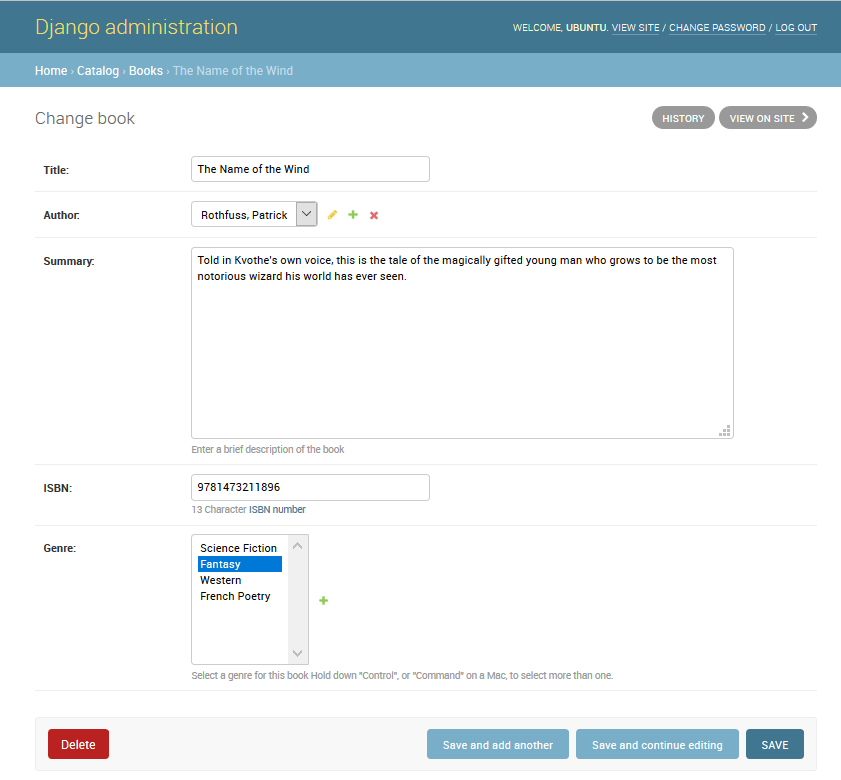
**Примечание:** А сейчас, хотелось бы, чтобы вы добавили несколько книг, авторов и жанров (например, Фэнтези) в ваше приложение. Удостоверьтесь, что каждый автор и жанр включает пару различных книг (позже, когда мы реализуем представления "list" и "detail", это сделает их более интересными).

После того, когда книги добавлены, для перехода на главную страницу админ-панели кликните на ссылке **Home** в верхней части страницы. Потом кликните на ссылке **Books** для отображения текущего списка книг (или на одной из других ссылок, чтобы увидеть список соответствующей модели). После добавления нескольких книг список может выглядеть наподобие скриншота ниже. Отображается название каждой из книг. Его возвращает метод \_\_str\_\_() в модели Book, созданной в предыдущей статье.



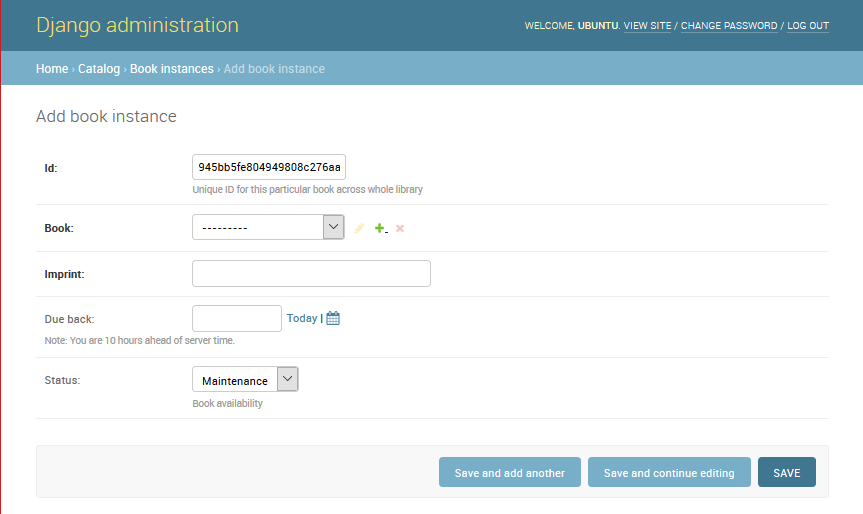
Для удаления книги из этого списка выберите чекбокс рядом с ней и действие delete... из выпадающего списка Action, а затем нажмите кнопку **Go**. Также можно добавить новую книгу, нажав на кнопку **ADD BOOK**.

Вы можете редактировать книгу, кликнув по ссылке с её названием. Страница редактирования книги, приведённая ниже, практически идентична странице добавления новой книги. Основные отличия - это заголовок страницы (Change book) и наличие кнопок **Delete**, **HISTORY** и **VIEW ON SITE.** Последняя присутствует, так как мы определили метод get\_absolute\_url() в нашей модели.



Теперь перейдите назад на страницу **Home** (используя ссылку Home в навигационной цепочке вверху страницы) и просмотрите списки **Author** и **Genre**. В них уже должно быть несколько элементов, созданных при добавлении новых книг. Если хотите, добавьте ещё.

Однако у вас не будет ни одного экземпляра книги, потому что они не создаются из модели Book (хотя можно создать книгу из модели BookInstance — такова природа поля ForeignKey). Для отображения страницы Add book instance (см. рисунок ниже) вернитесь на страницу Home и нажмите кнопку **Add**. Обратите внимание на длинный уникальный Id для идентификации конкретного экземпляра книги в библиотеке.



Создайте несколько экземпляров для каждой из ваших книг. Установите статус Available (доступен) для некоторых экземпляров и On loan (выдан) для остальных. Если статус экземпляра **not** Available (недоступен), то также установите дату возврата (Due back).

Вот и все! Вы изучили как запустить и использовать админ-панель. Также были созданы записи для Book, BookInstance, Genre и Author, которые можно будет использовать после создания наших собственных представлений и шаблонов.