

## Содержание

1.	Создание БД и нового проекта .....	2
2.	Настройка соединения с БД.....	2
3.	Проверка соединения с БД.....	3
4.	Первое приложение .....	3
5.	Описание моделей .....	3
6.	Установка модели и создание БД.....	4
7.	Доступ к данным.....	6
8.	Строковое представление модели .....	7
9.	Изменение данных .....	8
10.	Селекция/Изменение данных.....	9
11.	Удаление данных .....	10
12.	Привязка к проекту.....	11
13.	Задание.....	13
	Краткий справочник .....	14
	Определения модели.....	14
	Универсальные опции полей .....	15
	Отношения.....	15

## 1. Создание БД и нового проекта

Сначала, как обычно, создадим новый проект:

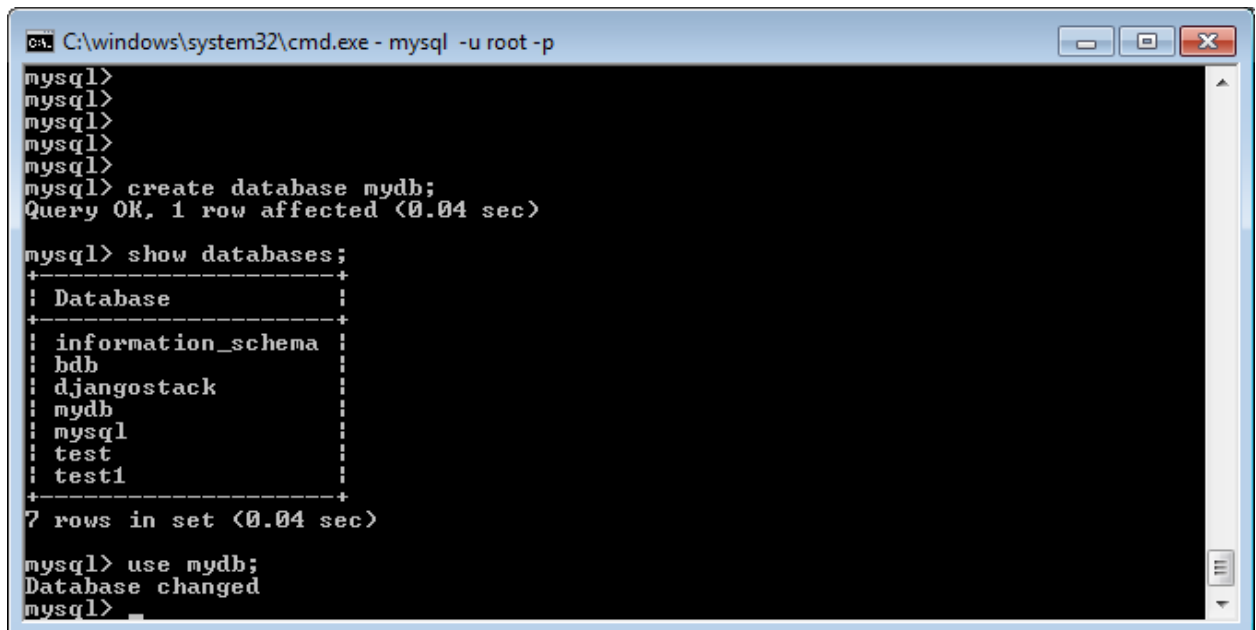
```
django-admin.py startproject booksdb
```

Для создания БД воспользуемся СУБД mysql:

- a. Соединимся с СУБД : `mysql -u %user_name% -p`

Соответственно, в текущем случае `mysql -u root -p`, имя пользователя и пароль должны соответствовать тем данным, которые вы указали при установке mysql

- b. Создадим БД: `create database mydb`:



```
cmd. C:\windows\system32\cmd.exe - mysql -u root -p
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> create database mydb;
Query OK, 1 row affected (0.04 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| bdb          |
| djangostack  |
| mydb        |
| mysql       |
| test        |
| test1       |
+-----+
7 rows in set (0.04 sec)

mysql> use mydb;
Database changed
mysql>
```

Рис. 1 Создание БД

Команда `show databases` выводит список всех баз данных.

Команда `use mydb` переключает на использование конкретной базы данных (mydb).

## 2. Настройка соединения с БД

Прежде, чем приступать к работе с базой данных, в файле настроек `settings.py` следует указать основные параметры соединения с БД:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # Add 'postgresql_psycopg2', 'postgresql',
        'mysql', 'sqlite3' or 'oracle'.
        'NAME': 'mydb',
        'USER': 'root', # Not used with sqlite3.
        'PASSWORD': '1q2w3e', # Not used with sqlite3.
        'HOST': '', # Set to empty string for localhost. Not used with sqlite3.
        'PORT': '', # Set to empty string for default. Not used with sqlite3.
    }
}
```

}

Наиболее важными параметрами являются:

- ENGINE – название используемого движка СУБД (в данном случае mysql);
- NAME – название БД;
- USER – имя пользователя для соединения с БД;
- PASSWORD – пароль для соединения с БД.

### 3. Проверка соединения с БД

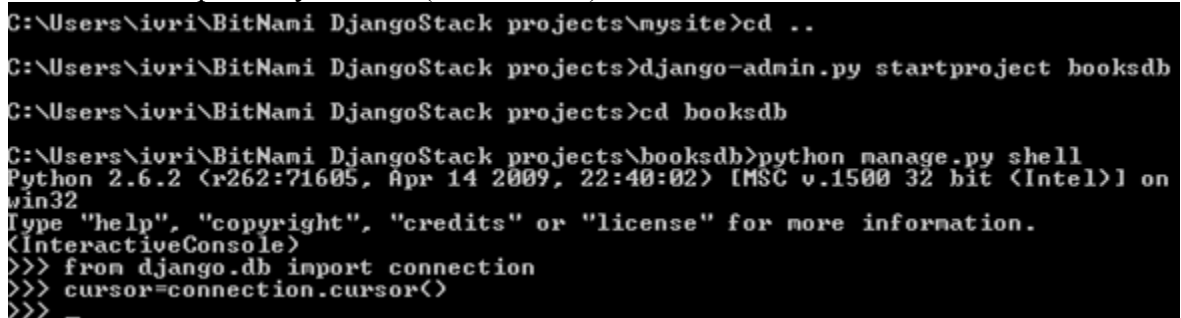
Для проверки соединения зайдём в консольную оболочку:

```
python manage.py shell
```

В ней попробуем соединиться с базой :

```
from django.db import connection  
cursor = connection.cursor()
```

Соединение прошло успешно (без ошибок):



```
C:\Users\ivri\BitNami DjangoStack projects>cd ..  
C:\Users\ivri\BitNami DjangoStack projects>django-admin.py startproject booksdb  
C:\Users\ivri\BitNami DjangoStack projects>cd booksdb  
C:\Users\ivri\BitNami DjangoStack projects\booksdb>python manage.py shell  
Python 2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)] on  
win32  
Type "help", "copyright", "credits" or "license" for more information.  
<InteractiveConsole>  
>>> from django.db import connection  
>>> cursor=connection.cursor()  
>>>
```

Рис. 2 Соединение с БД

### 4. Первое приложение

Для создания первого приложения (модели) вызовем скрипт (в основной директории нашего проекта):

```
python manage.py startapp books
```

В результате будет создана поддиректория с именем приложения (books):

```
books/  
  __init__.py  
  models.py  
  views.py
```

Как видно, появился новый файл, *models.py*, в котором происходит описание самих моделей.

### 5. Описание моделей

Создадим модель, описывающую книги и включающую в себя 3 сущности: Publisher (Издательство), Author (Автор), Book (Книга).

Сначала импортируем необходимый модуль

```
from django.db import models
```

```

class Publisher(models.Model): #наследуем от Model
    name = models.CharField(max_length=30) # имя, строка (макс. Длина=30)
    address = models.CharField(max_length=50) # адрес, строка (макс.
Длина=50)
    city = models.CharField(max_length=60) # город, строка (макс. Длина=60)
    state_province = models.CharField(max_length=30) # область, строка (макс.
Длина=30)
    country = models.CharField(max_length=50) # страна, строка (макс.
Длина=50)
    website = models.URLField() # сайт, URL

```

```

class Author(models.Model):
    first_name = models.CharField(max_length=30) #имя, строка (макс.
Длина=30)
    last_name = models.CharField(max_length=40) #фамилия, строка (макс.
Длина=40)
    email = models.EmailField() # эл. почта, Email

```

```

class Book(models.Model):
    title = models.CharField(max_length=100) # название (макс. Длина=100)
    authors = models.ManyToManyField(Author) # поле многие-ко-многим
    publisher = models.ForeignKey(Publisher) # FK на Publisher
    publication_date = models.DateField() #дата публикации

```

Более подробное описание полей, их значений, задания типов отношений смотрите в приложении.

## 6. Установка модели и создание БД

### Установка модели

Для того, чтобы установить модель, необходимо в файле settings.py в секцию INSTALLED\_APPS добавить на нее ссылку в формате *имя\_проекта.имя\_приложения*:

```

INSTALLED_APPS = (
    #'django.contrib.auth',
    #'django.contrib.contenttypes',
    #'django.contrib.sessions',
    #'django.contrib.sites',
    #'django.contrib.messages',
    #'django.contrib.staticfiles',
    'bookdb.books',
)

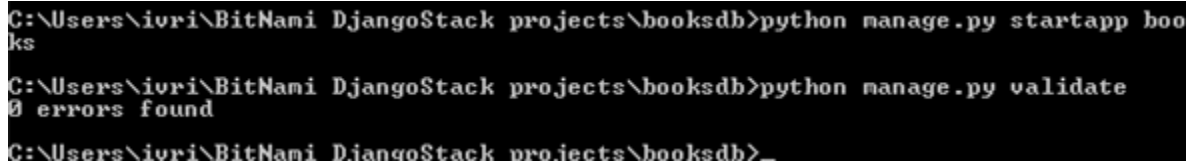
```

Остальные строки можно смело закомментировать в силу того, что пока что они нам не понадобятся.

### Создание схемы базы данных

Прежде, чем создавать схему базы данных, необходимо проверить синтаксическую корректность описанной модели. Для этого запустим скрипт:

```
python manage.py validate
```



```
C:\Users\ivri\BitNami DjangoStack projects\booksdb>python manage.py startapp books
C:\Users\ivri\BitNami DjangoStack projects\booksdb>python manage.py validate
0 errors found
C:\Users\ivri\BitNami DjangoStack projects\booksdb>
```

Рис. 3 Проверка правильности модели

Если ошибок не обнаружено, то переходим к следующему шагу.

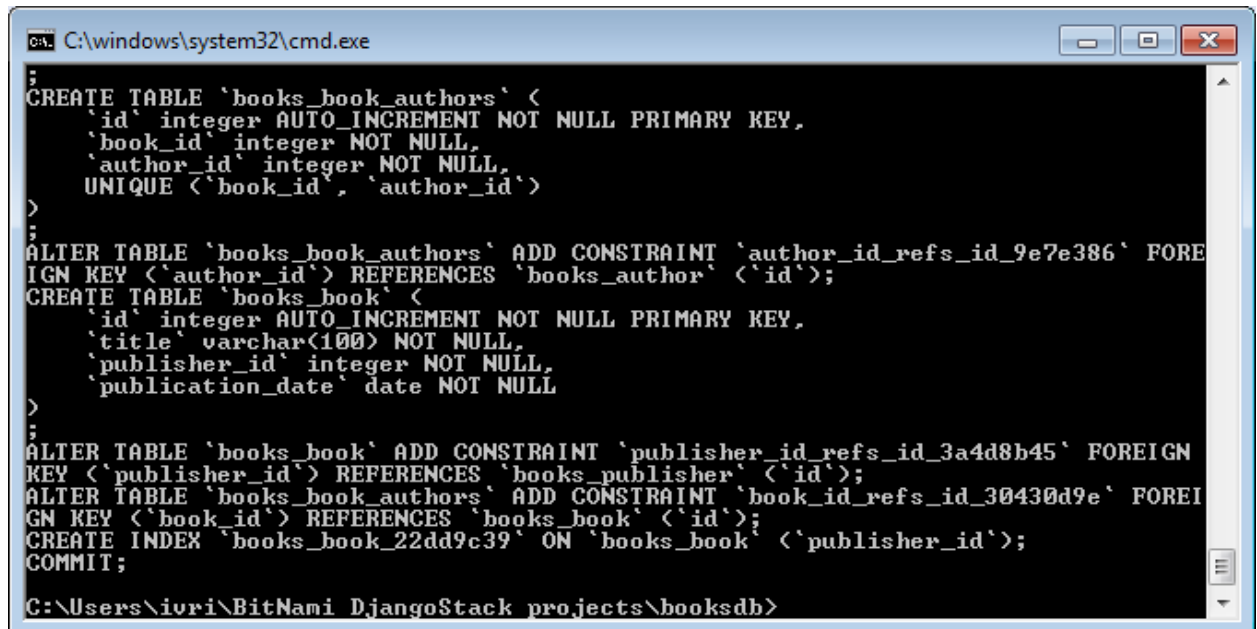
Сгенерируем схему базы данных по нашей модели. Отметим, что сама база данных пока что не изменяется и никаких скриптов не запускается.

Для создания схемы запустим команду:

```
python manage.py sqlall books
```

*books* – имя нашего приложения

На выходе мы получаем готовую схему БД, которую можно сохранить во внешний файл:



```
C:\windows\system32\cmd.exe
;
CREATE TABLE `books_book_authors` (
  `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `book_id` integer NOT NULL,
  `author_id` integer NOT NULL,
  UNIQUE (`book_id`, `author_id`)
)
;
ALTER TABLE `books_book_authors` ADD CONSTRAINT `author_id_refs_id_9e7e386` FOREIGN KEY (`author_id`) REFERENCES `books_author` (`id`);
CREATE TABLE `books_book` (
  `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `title` varchar(100) NOT NULL,
  `publisher_id` integer NOT NULL,
  `publication_date` date NOT NULL
)
;
ALTER TABLE `books_book` ADD CONSTRAINT `publisher_id_refs_id_3a4d8b45` FOREIGN KEY (`publisher_id`) REFERENCES `books_publisher` (`id`);
ALTER TABLE `books_book_authors` ADD CONSTRAINT `book_id_refs_id_30430d9e` FOREIGN KEY (`book_id`) REFERENCES `books_book` (`id`);
CREATE INDEX `books_book_22dd9c39` ON `books_book` (`publisher_id`);
COMMIT;
C:\Users\ivri\BitNami DjangoStack projects\booksdb>
```

Рис. 4 Генерация схемы БД

Имена таблицам назначаются автоматически по принципу имя-приложения\_имя-класса.

Для запуска осталось только запустить команду:

```
python manage.py syncdb
```

Все запросы будут выполнены и в базе появятся соответствующие таблицы:

```
ca. Select C:\windows\system32\cmd.exe
CREATE TABLE `books_book` (
  `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
  `title` varchar(100) NOT NULL,
  `publisher_id` integer NOT NULL,
  `publication_date` date NOT NULL
)
;
ALTER TABLE `books_book` ADD CONSTRAINT `publisher_id_refs_id_3a4d8b45` FOREIGN
KEY (`publisher_id`) REFERENCES `books_publisher` (`id`);
ALTER TABLE `books_book_authors` ADD CONSTRAINT `book_id_refs_id_30430d9e` FOREI
GN KEY (`book_id`) REFERENCES `books_book` (`id`);
CREATE INDEX `books_book_22dd9c39` ON `books_book` (`publisher_id`);
COMMIT;

C:\Users\ivri\BitNami DjangoStack projects\booksdb>python manage.py syncdb
Creating tables ...
Creating table books_publisher
Creating table books_author
Creating table books_book_authors
Creating table books_book
Installing custom SQL ...
Installing indexes ...
No fixtures found.

C:\Users\ivri\BitNami DjangoStack projects\booksdb>
```

Рис. 5 Запуск запросов и создание БД

```
ca. C:\windows\system32\cmd.exe - mysql -u root -p
+-----+
| information_schema |
| bdb                 |
| djangostack         |
| mydb                |
| mysql               |
| test                |
| test1               |
+-----+
7 rows in set (0.04 sec)

mysql> use mydb;
Database changed
mysql> show tables;
+-----+
| Tables_in_mydb     |
+-----+
| books_author        |
| books_book          |
| books_book_authors |
| books_publisher     |
+-----+
4 rows in set (0.25 sec)

mysql>
```

Рис. 6 Созданные таблицы в MySQL

## 7. Доступ к данным

Зайдем в консольную оболочку  
*python manage.py shell*

Попробуем импортировать модель Publisher:  
*from books.models import Publisher*

Если не возникло ошибок, значит, все прошло успешно.

Теперь создадим объект

```
p1 = Publisher(name='Addison-Wesley', address='75 Arlington Street',  
city='Boston', state_province='MA', country='U.S.A.',  
website='http://www.apress.com/')
```

Пока что в самой базе ничего не поменялось.

Чтобы записать данные в базу, объект следует сохранить:

```
p1.save()
```

Запись об объекте автоматически появилась в таблице:

```
mysql>  
mysql> select * from books_publisher;  
+-----+-----+-----+-----+-----+-----+  
| id | name | address | city | state_province | country |  
| website | | | | | |  
+-----+-----+-----+-----+-----+-----+  
| 1 | Addison-Wesley | 75 Arlington Street | Boston | MA | U.S.A. |  
| http://www.apress.com/ | | | | | |  
+-----+-----+-----+-----+-----+-----+  
1 row in set (0.00 sec)
```

Рис. 7 Сохраненный объект добавлен в таблицу

Добавим еще один объект:

```
p2 = Publisher(name="O'Reilly", address='10 Fawcett St.', city='Cambridge',  
state_province='MA', country='U.S.A.', website='http://www.oreilly.com/')  
p2.save()
```

Запросим все имеющиеся в базе объекты типа Publisher:

```
publisher_list = Publisher.objects.all()  
publisher_list
```

```
>>> p2 = Publisher(name="O'Reilly", address='10 Fawcett St.', city='Cambridge',  
state_province='MA', country='U.S.A.', website='http://www.oreilly.com/')  
>>> p2.save()  
>>> publisher_list = Publisher.objects.all()  
>>> publisher_list  
[<Publisher: Publisher object>, <Publisher: Publisher object>]
```

Рис. 8 Список объектов

Появились два объекта, но без указания имен. Предположим, мы бы хотели вместо Publisher object получать название издательства.

## 8. Строковое представление модели

Итак, внесем небольшие изменения в модель (models.py) и укажем, какие именно поля нужно выводить в случае возврата объекта в строковом виде.

Для этого в каждый класс добавим специальный метод `__unicode__`:

```

class Publisher:
    def __unicode__(self):
        return self.name # название издательства
class Author:
    def __unicode__(self):
        return '%s %s' % (self.first_name, self.last_name) # ФИИ автора
class Book:
    def __unicode__(self):
        return self.title # название книги

```

Не забудем сохранить модель и вызвать `python manage.py syncdb`.

Теперь опять запросим список всех Издателей:

```

>>> from books.models import Publisher
>>> publisher_list = Publisher.objects.all()
>>> publisher_list
<Publisher: Addison-Wesley>, <Publisher: O'Reilly>]

```

**Рис. 9** Строковое представление объектов

В строковом представлении Издателя теперь стоит его название (name).

## 9. Изменение данных

Создадим новый объект :

```

p = Publisher(name='Apress', address='2855 Telegraph Ave.', city='Berkeley',
state_province='CA', country='U.S.A.', website='http://www.apress.com/')
p.save()

```

У данного объекта можно получить основные атрибуты (id (назначается автоматически после сохранения объекта), name, address и другие).

```

>>> p = Publisher(name='Apress', address='2855 Telegraph Ave.', city='Berkeley',
state_province='CA', country='U.S.A.', website='http://www.apress.com/')
>>> p.save()
>>> p.id
1
>>> print p.id
1
>>> p.name
Apress
>>> p.address
2855 Telegraph Ave.

```

**Рис. 10** Запрос атрибутов объекта

Предположим, что мы решили переименовать Apress в Apress Publishing. Для этого, нужно просто присвоить значение соответствующему атрибуту:

```

p.name='Apress Publishing'

```

и сохранить

```

p.save()

```



```

C:\Users\ivori\BitNami DjangoStack projects\booksdb>python manage.py shell
Python 2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
<InteractiveConsole>
>>> from books.models import Publisher
>>> publisher_list = Publisher.objects.all()
>>> publisher_list
[<Publisher: Addison-Wesley>, <Publisher: O'Reilly>]
>>> p = Publisher(name='Apress', address='2855 Telegraph Ave.', city='Berkeley',
state_province='CA', country='U.S.A.', website='http://www.apress.com/')
>>> p.save()
>>> p.id
3L
>>> print p.id
3
>>> p.name
'Apress'
>>> p.address
'2855 Telegraph Ave.'
>>> p.name='Apress Publishing'
>>> p.save()
>>> p
<Publisher: Apress Publishing>
>>>

```

Рис. 11 Изменение объекта

Информация в базе, соответственно, также была изменена:

```

C:\windows\system32\cmd.exe - mysql -u root -p
1 row in set (0.00 sec)

mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql>
mysql> select * from books_publisher where id=3;
+-----+-----+-----+-----+-----+-----+
| id | name           | address           | city       | state_province | country |
+-----+-----+-----+-----+-----+-----+
| 3  | Apress Publishing | 2855 Telegraph Ave. | Berkeley  | CA             | U.S.A. |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>

```

Рис. 12 Изменение информации в БД

## 10. Селекция/Изменение данных

Выборка всех объектов:

*Publisher.objects.all()*

Аналог: SELECT id, name, address, city, state\_province, country, website FROM books\_publisher;

Селекция по определенному критерию:

*Publisher.objects.filter(name='Apress Publishing') :*

Аналог: SELECT id, name, address, city, state\_province, country, website FROM books\_publisher WHERE name = 'Apress';

Селекция с указанием нечеткого значения (строки):

*Publisher.objects.filter(name\_\_contains="press")*

Аналог: SELECT id, name, address, city, state\_province, country, website FROM books\_publisher WHERE name LIKE '%press%';

Выборка одного объекта: *Publisher.objects.get(name="Apress Publishing")*

Сортировка объектов по определенному атрибуту:  
*Publisher.objects.order\_by("name")*

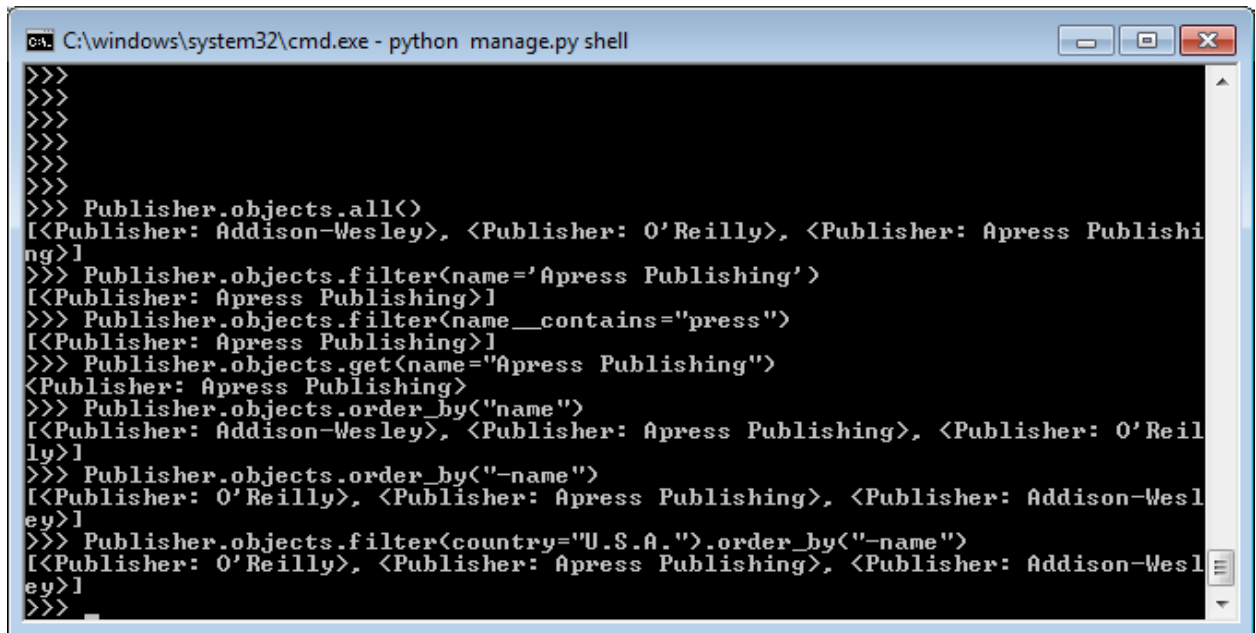
Цепочка запросов: *Publisher.objects.filter(country="U.S.A.").order\_by("-name")*

Изменение мн-ва объектов одним запросом:

*p = Publisher.objects.get(name='Apress Publishing')*

*p.name="Apress"*

*p.save()*



```
C:\windows\system32\cmd.exe - python manage.py shell
>>>
>>>
>>>
>>>
>>>
>>>
>>> Publisher.objects.all()
[<Publisher: Addison-Wesley>, <Publisher: O'Reilly>, <Publisher: Apress Publishing>]
>>> Publisher.objects.filter(name='Apress Publishing')
[<Publisher: Apress Publishing>]
>>> Publisher.objects.filter(name__contains="press")
[<Publisher: Apress Publishing>]
>>> Publisher.objects.get(name="Apress Publishing")
<Publisher: Apress Publishing>
>>> Publisher.objects.order_by("name")
[<Publisher: Addison-Wesley>, <Publisher: Apress Publishing>, <Publisher: O'Reilly>]
>>> Publisher.objects.order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress Publishing>, <Publisher: Addison-Wesley>]
>>> Publisher.objects.filter(country="U.S.A.").order_by("-name")
[<Publisher: O'Reilly>, <Publisher: Apress Publishing>, <Publisher: Addison-Wesley>]
>>>
```

Рис. 13 Различные методы селекции данных

## 11. Удаление данных

Удаление единичного объекта:

*p=Publisher.objects.get(name="O'Reilly")*

*p.delete()*

```

C:\windows\system32\cmd.exe - mysql -u root -p
mysql> select * from books_publisher;
+-----+-----+-----+-----+-----+
| id | name | address | city | state_province | country |
+-----+-----+-----+-----+-----+
| 1 | Addison-Wesley | 75 Arlington Street | Boston | MA | U.S. |
| 3 | Apress Publishing | 2855 Telegraph Ave. | Berkeley | CA | U.S. |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from books_publisher;
+-----+-----+-----+-----+-----+
| id | name | address | city | state_province | country |
+-----+-----+-----+-----+-----+
| 1 | Addison-Wesley | 75 Arlington Street | Boston | MA | U.S. |
| 3 | Apress Publishing | 2855 Telegraph Ave. | Berkeley | CA | U.S. |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

Рис. 14 Результат удаления объекта

Удаление выборки объектов:

```
Publisher.objects.filter(country='USA').delete()
```

Удаление всех объектов:

```
Publisher.objects.all().delete()
```

## 12. Привязка к проекту

Следующим шагом попробуем соединить отобразить данные на странице. Для начала добавим немного информации в БД.

Добавим Author и Book аналогичным способом как мы делали в случае с Publisher.

Для вставки Foreign Key и заполнения поля Many-to-Many достаточно в них передать экземпляр, а не сам ключ или id.

Допустим, у нас уже имеется Publisher p1 и автор a1 (мы можем сохранить это объекты после создания, либо просто получить их запросом *a1=Author.objects.get(first\_name='Isaac')*).

Теперь добавим книгу:

```
b1=Book(title='The Martian Way', publisher=p1, publication_date=datetime.datetime.now())
```

Заметим, что авторов мы пока не указали, мы их сможем добавить после сохранения этого объекта.

```
b1.save()
```

```
b1.authors.add(a1)
```

```
b1.save()
```

Можно добавить необходимое количество данных, а затем переходить к реализации их обработки и представления.

### Создание привязки URL:

В файл `urls.py` добавим шаблон, который при попадании на страницу `/books/` будет вызывать функцию `books`:

```
('^books/$', books),
```

### Опишем функцию `books` в `views.py`:

```
#загрузим модели
```

```
from books.models import *
```

```
def books(request):
```

```
    books=Book.objects.all() # получение списка книг
```

```
    return render_to_response("books.html", locals())
```

В шаблон `books.html` мы передаем список всех книг.

### Опишем шаблон `books.html`:

```
{% extends "base.html" %}
```

```
{% block title %}Books list{% endblock %}
```

```
{% block content %}
```

```
<table border=0>
```

```
<tr><th>ID</th><th>Title</th><th>Author</th><th>Publisher</th><th>Publication date</th></tr>
```

```
{% for book in books %}
```

```
<tr align=center><td>{{ book.id }}</td><td>{{ book.title }}</td>
```

```
<td>{% for author in book.authors.all %} {{ author }}<br/> {% endfor %}</td>
```

```
<td>{{ book.publisher }}</td>
```

```
<td>{{ book.publication_date }}</td>
```

```
</tr>
```

```
{% endfor %}
```

```
</table>
```

```
{% endblock %}
```

Заметим, что при обращении `book.publisher` загружается не `id`, а именно строковое представление, которое мы определили в самом начале. Аналогично с `author`.



ID	Title	Author	Publisher	Publication date
1	Seeing voices	Oliver Sacks	Apress Publishing	Oct. 10, 2011
2	On the road	Jack Kerouac	Addison-Wesley	Oct. 10, 2011
3	The Martian Way	Isaac Azimov	Apress Publishing	Oct. 11, 2011

Thanks for visiting my site.

**Рис. 15** Отображение данных из базы

### 13.Задание

Основываясь на схеме БД из первой лабораторной работы\*, создать и описать модель данных в Django.

Сделать привязку к проекту из Лабораторной работы №2, то есть:

1. Используя старый проект, создать в нем приложение с вашей моделью данных
2. На основе модели автоматически создать БД в выбранной СУБД
3. Наполнить БД небольшим количеством данных (используя объекты, а не SQL запросы)
4. Добавить страницу для отображения данных вашей базы (по вышеуказанному примеру).
5. В отчет: схема БД изначальная, модель, основные скриншоты, код программы.

\* -можно сделать новую схему, но включающую как минимум 3 сущности.

## Краткий справочник

### Определения модели

**AutoField** – IntegerField с автоматическим инкрементом. Обычно не используется, поскольку поле ID создается автоматически.

**BooleanField** – булево поле

**CharField** – строка с указанием максимальной длины(maxlength)

**CommaSeparatedIntegerField** – поле для значений типа Integer, разделенных запятой. Обязательный параметр maxlength.

**DateField** – поле для даты. Опциональные аргументы:

*auto\_now* – автоматически сохраняет время при сохранении объекта.

Полезен как параметр “last modified”.

*auto\_now\_add* – автоматически сохраняет время при создании объекта.

**DateTimeField** – поле для даты и времени. Аргументы аналогичны DateField.

**EmailField** – CharField, проверяющий, что значение является валидным email адресом.

**FileField** – директория для загрузки файлов. Обязательный параметр:

*upload\_to* – путь для загрузки, который будет добавлен к MEDIA\_ROOT

**FilePathField** – поле, значения которого ограничены именами файлов в заданной директории.

Обязательный параметр:

*path* – абсолютный путь до директории, файлы которой используются

Необязательные параметры:

*match* – паттерн, регулярное выражение, для фильтрации имен файлов

*recursive* – флаг, отвечающий за включение поддиректорий в path.

Пример:

*FilePathField(path="/home/images", match="foo.\*", recursive=True)*

Соответствует /home/images/foo.gif, но не /home/images/foo/bar.gif

**FloatField** – значения с плавающей точкой. Обязательные параметры:

*max\_digits* – максимальное количество цифр в числе

*decimal\_places* – количество позиций после запятой

**ImageField** – то же, что и FileField, но с проверкой, что файл является изображением.

**IntegerField** – целочисленное значение

**IPAddressField** – IP адрес в строковом значении

**NullBooleanField** – то же самое, что и BooleanField, но с дополнительным значением None/NULL

**PhoneNumberField** – CharField с проверкой валидности телефонного номера (в формате США)

**PositiveIntegerField** – IntegerField, строго больший нуля

**PositiveSmallIntegerField** – PositiveIntegerField, ограниченное сверху значением 65,535 (зависит от СУБД)

**SlugField** – короткая метка, содержащая буквы, цифры, нижнее подчеркивание. Обычно используется для хранения URL

**SmallIntegerField** – IntegerField, ограниченный диапазоном значений (обычно от -32,768 до +32,767)

**TextField** – неограниченное по длине текстовое поле

**TimeField** – время, имеет те же параметры, что и DateField и DateTimeField

**URLField** – поле для URL. Необязательный параметр:

*verify\_exists* – проверка существования/работы ссылки

**USStateField** – двухбуквенное обозначение штата (для США)

**XMLField** – TextField с проверкой валидности XML (соответствие его схеме).

Обязательный параметр:

*schema\_path* – путь до схемы

Требуется наличие `jing` (<http://thaiopensource.com/relaxng/jing.html>) для валидации.

## Универсальные опции полей

В данном разделе перечислены не все опции, а только наиболее употребляемые.

Полностью - <http://djangobook.com/en/1.0/appendixB/>

**null** – если True, то Django запишет пустое значение в БД как NULL. По умолчанию False.

**blank** – если True, то поле можно оставить пустым.

**choices** – возможные принимаемые значения

Пример:

```
YEAR_IN_SCHOOL_CHOICES = (  
    ('FR', 'Freshman'),  
    ('SO', 'Sophomore'),  
    ('JR', 'Junior'),  
    ('SR', 'Senior'),  
    ('GR', 'Graduate'),  
)
```

Первое значение – то, что сохраняется в базе

Второе значение – то, что показывается пользователю

```
GENDER_CHOICES = (  
    ('M', 'Male'),  
    ('F', 'Female'),  
)  
class Foo(models.Model):  
    gender = models.CharField(maxlength=1, choices=GENDER_CHOICES)
```

**help\_text** – строка подсказки

**primary\_key** – если True, то поле является первичным ключом

**unique** – если True, то поле должно быть уникальным

## Отношения

### Многие–к-одному (Many-to-one)

У машины есть один производитель(*Manufacturer*), производитель выпускает многие машины(*Car*)

```
class Manufacturer(models.Model):  
    ...  
  
class Car(models.Model):  
    manufacturer = models.ForeignKey(Manufacturer)  
    ...
```

Рекурсивное отношение – Многие-к-одному с самим собой – достигается через использование self:

```
class Employee(models.Model):  
    manager = models.ForeignKey('self')
```

### **Многие-ко-многим (Many-to-Many)**

Пицца(*Pizza*) может иметь различные заправки(*Topping*), заправка может быть использована в различных пиццах.

```
class Topping(models.Model):  
    ...  
  
class Pizza(models.Model):  
    toppings = models.ManyToManyField(Topping)  
    ...
```

В какой именно модели определено поле Многие-ко-Многим, не имеет значения, но оно должно быть определено как минимум в одной из моделей.

Более подробно - <http://djangobook.com/en/1.0/appendixB/>