Содержание

Сериализация в Django

Django поддерживает механизм сериализации.

Попробуем сериализовать объекты одной из наших моделей.
Сначала перейдем в директорию проекта третьей лабораторной работы и запустим оболочку *python manage.py shell:*

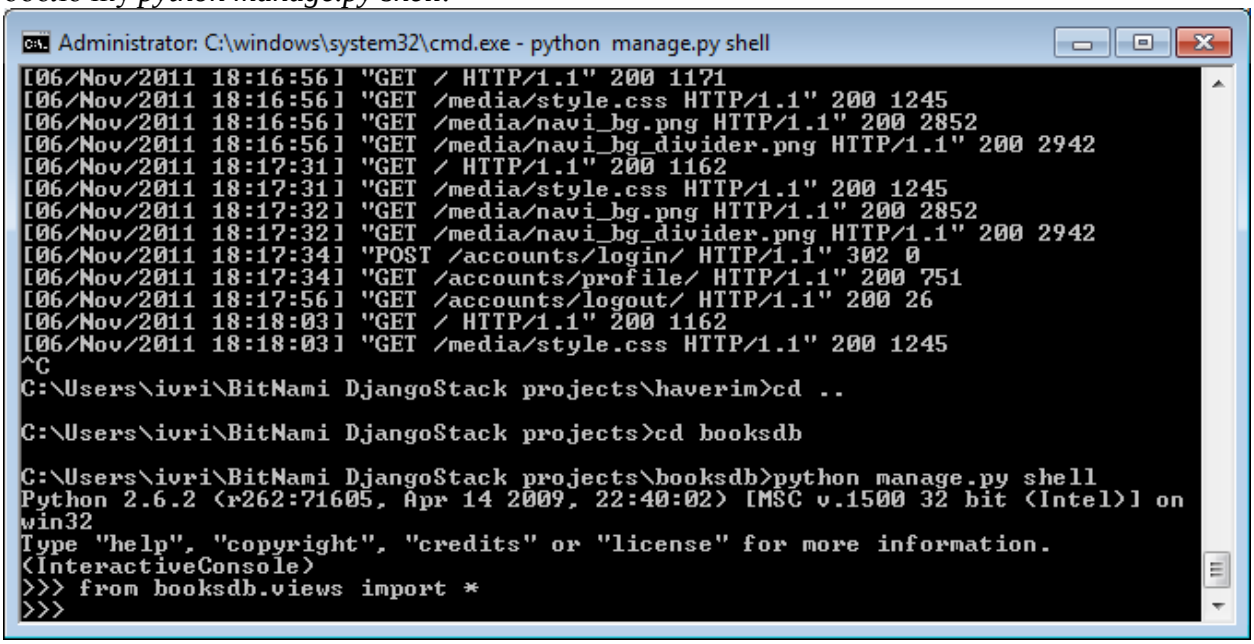

Рис. 1 Оболочка Python

После чего в ней вызовем следующую последовательность команд:
from django.core import serializers # загружаем сериализаторы
*from booksdb.views import * # загружаем модели*

XMLSerializer = serializers.get_serializer("xml") # выбираем формат сериализации
xml_serializer = XMLSerializer() # создаем сериализатор
out = open("file.xml", "w") # открываем файл на запись
xml_serializer.serialize(Book.objects.all(), stream=out) # запрашиваем все объекты типа Book и сериализуем их
out.close() # закрываем файл

Вот как выглядит это в консоли:

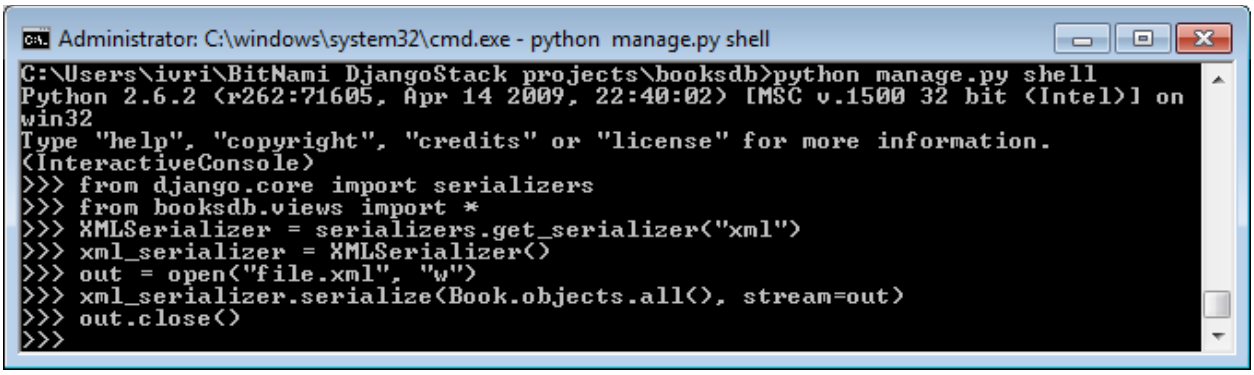

Рис. 2 Сериализация

Получаем следующий Xml:

```
<?xml version="1.0" encoding="utf-8" ?>
 <django-objects version="1.0">
 <object pk="1" model="books.book">
  <field type="CharField" name="title">On the road</field>
  <field to="books.publisher" name="publisher" rel="ManyToOneRel">2</field>
  <field type="DateField" name="publication_date">2011-10-14</field>
 <field to="books.author" name="authors" rel="ManyToManyRel">
  <object pk="3" />
     </field>
     </object>
 <object pk="2" model="books.book">
  <field type="CharField" name="title">The martian way</field>
  <field to="books.publisher" name="publisher" rel="ManyToOneRel">3</field>
  <field type="DateField" name="publication_date">2011-10-14</field>
 <field to="books.author" name="authors" rel="ManyToManyRel">
  <object pk="1" />
     </field>
     </object>
 <object pk="3" model="books.book">
  <field type="CharField" name="title">Seeing voices</field>
  <field to="books.publisher" name="publisher" rel="ManyToOneRel">4</field>
  <field type="DateField" name="publication_date">2011-10-14</field>
 <field to="books.author" name="authors" rel="ManyToManyRel">
  <object pk="2" />
     </field>
    </object>
  </django-objects>
```

Десериализация

Для десериализации вызываем метод deserialize:

```
for obj in serializers.deserialize("xml", data):
    do_something_with(obj)
```

Вызов *obj.save()* сохраняет объект в БД.

Возможные форматы сериализации

Django поддерживает следующие форматы:

- Xml
- Json
- Yaml

Сериализация ссылок (ключей)

Заметим, что строка `<field to="books.publisher" name="publisher" rel="ManyToOneRel">4</field>` , а также `<field to="books.author" name="authors" rel="ManyToManyRel">` выглядит весьма нечитабельной .

Исправим данный недостаток.
Добавим класс SomethingManager (models.Manager) и в нем определим метод для представления первичного ключа *get_by_natural_key*.
Помимо этого, в самой модели определим метод *natural_key*. И класс *class Meta:* с полем *unique_together* , в котором опишем значения, являющиеся вместе уникальными.

```
from django.db import models

class PublisherManager(models.Manager):
    def get_by_natural_key(self, name, city):
        return self.get(name=name, city=city)


class Publisher(models.Model):
        name = models.CharField(max_length=5)
        address = models.CharField(max_length=50)
        city = models.CharField(max_length=60)
        state_province = models.CharField(max_length=30)
        country = models.CharField(max_length=50)
        website = models.URLField()

        def __unicode__(self):
                return self.name

        def natural_key(self):
                return (self.name, self.city)

        class Meta:
```

```
        unique_together = (('name', 'city'),)


class AuthorManager(models.Manager):
    def get_by_natural_key(self, first_name, last_name):
        return self.get(first_name=first_name, last_name=last_name)

class Author(models.Model):
        first_name = models.CharField(max_length=30)
        last_name = models.CharField(max_length=40)
        email = models.EmailField()

        def __unicode__(self):
                return '%s %s' % (self.first_name, self.last_name)

        def natural_key(self):
                return (self.first_name, self.last_name)

        class Meta:
                unique_together = (('first_name', 'last_name'),)

class Book(models.Model):
        title = models.CharField(max_length=100)
        authors = models.ManyToManyField(Author)
        publisher = models.ForeignKey(Publisher)
        publication_date = models.DateField()

        def __unicode__(self):
                return self.title
```

После смены модели перезапустим *python manage.py syncdb*.
Запустим сериализацию с дополнительным параметром *use_natural_keys=True*:

```
from django.core import serializers
from booksdb.views import *

XMLSerializer = serializers.get_serializer("xml") # определяем формат сериализации
xml_serializer = XMLSerializer()
out = open("file.xml", "w") # открываем внешний файл на запись
xml_serializer.serialize(Book.objects.all(), stream=out, use_natural_keys=True)
out.close() # закрываем файл
```

В результате получим:

```
<?xml version="1.0" encoding="utf-8" ?>
 <django-objects version="1.0">
```

```
    <object pk="1" model="books.book">
  <field type="CharField" name="title">On the road</field>
<field to="books.publisher" name="publisher" rel="ManyToOneRel">
  <natural>Addison-Wesley</natural>
  <natural>Boston</natural>
    </field>
  <field type="DateField" name="publication_date">2011-10-14</field>
 <field to="books.author" name="authors" rel="ManyToManyRel">
<object>
  <natural>Jack</natural>
  <natural>Kerouac</natural>
    </object>
    </field>
    </object>
<object pk="2" model="books.book">
  <field type="CharField" name="title">The martian way</field>
 <field to="books.publisher" name="publisher" rel="ManyToOneRel">
  <natural>O'Reilly</natural>
  <natural>Cambridge</natural>
    </field>
  <field type="DateField" name="publication_date">2011-10-14</field>
<field to="books.author" name="authors" rel="ManyToManyRel">
<object>
  <natural>Isaac</natural>
  <natural>Azimov</natural>
    </object>
    </field>
    </object>
<object pk="3" model="books.book">
  <field type="CharField" name="title">Seeing voices</field>
<field to="books.publisher" name="publisher" rel="ManyToOneRel">
  <natural>Apress Publishing</natural>
  <natural>Berkeley</natural>
    </field>
  <field type="DateField" name="publication_date">2011-10-14</field>
<field to="books.author" name="authors" rel="ManyToManyRel">
 <object>
  <natural>Oliver</natural>
  <natural>Sacks</natural>
    </object>
    </field>
   </object>
</django-objects>
```

Язык XQuery

Устанавливаем http://exist.sourceforge.net/.

При установке вам может понадобиться установить также JDK (JSE). В принципе, вы можете использовать любой редактор XQuery.

Перед запуском оболочки eXist, нужно запустить eXist Database Startup. Он расположен в основном меню.

Затем в стороке браузера следует набрать:

http://localhost:8080/exist/index.xml

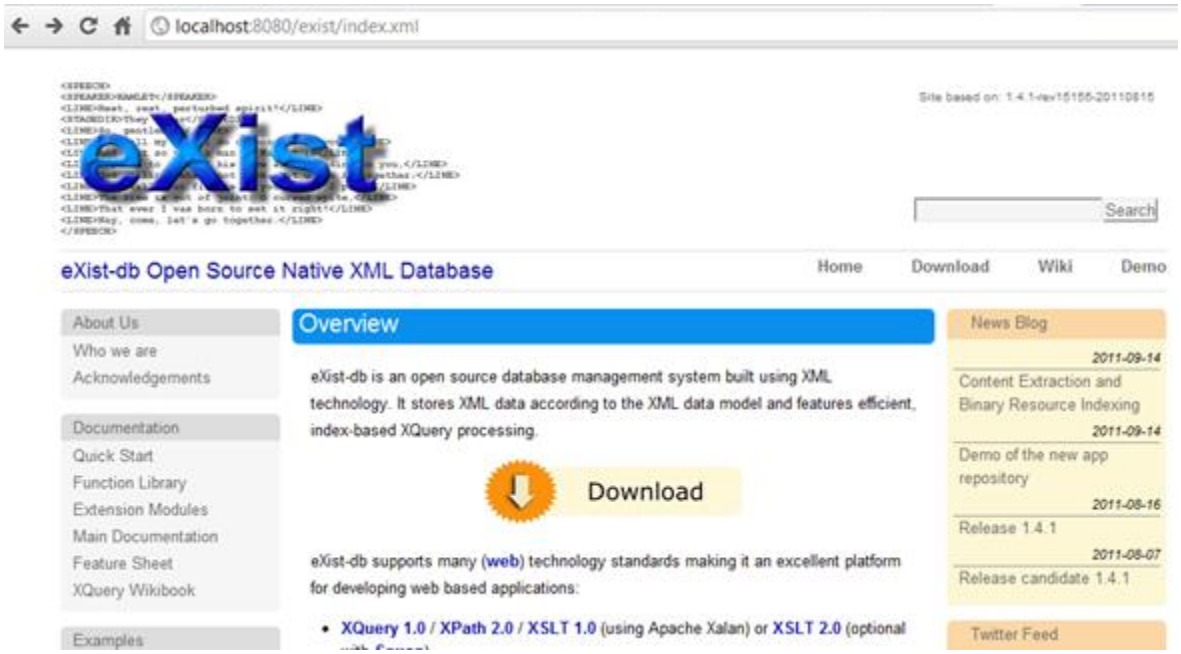

Рис. 3 Главное окно СУБД eXist

В меню слева находим XQuery Sandbox и переходим по ссылке.

http://localhost:8080/exist/sandbox/sandbox.xql

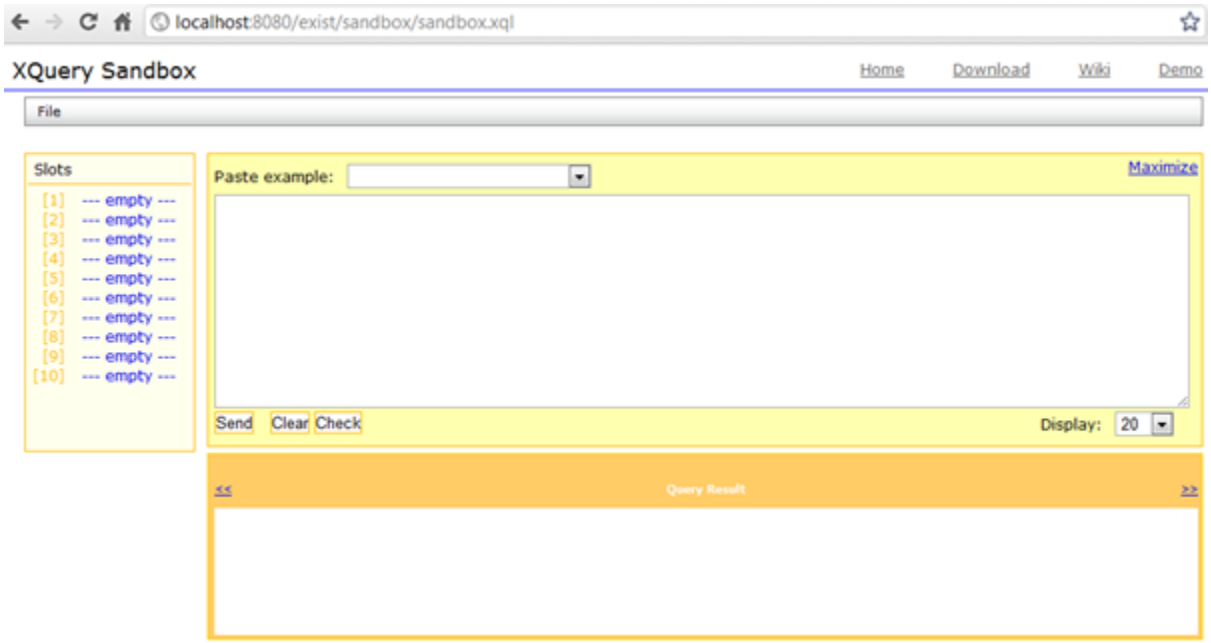

Рис. 4 Редактор XQuery

Разберем простые примеры.

Объявим последовательность:

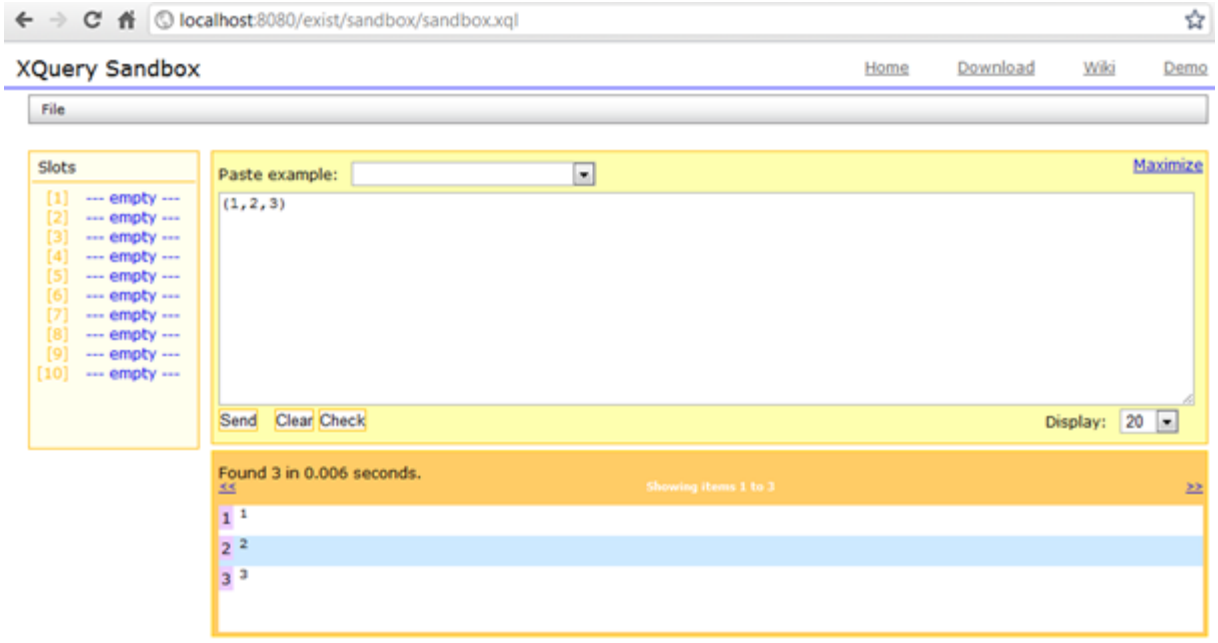

Рис. 5 Последовательности в XQuery

Динамическое создание XML :

Рис. 6 Создание XML в XQuery

Цикл For:

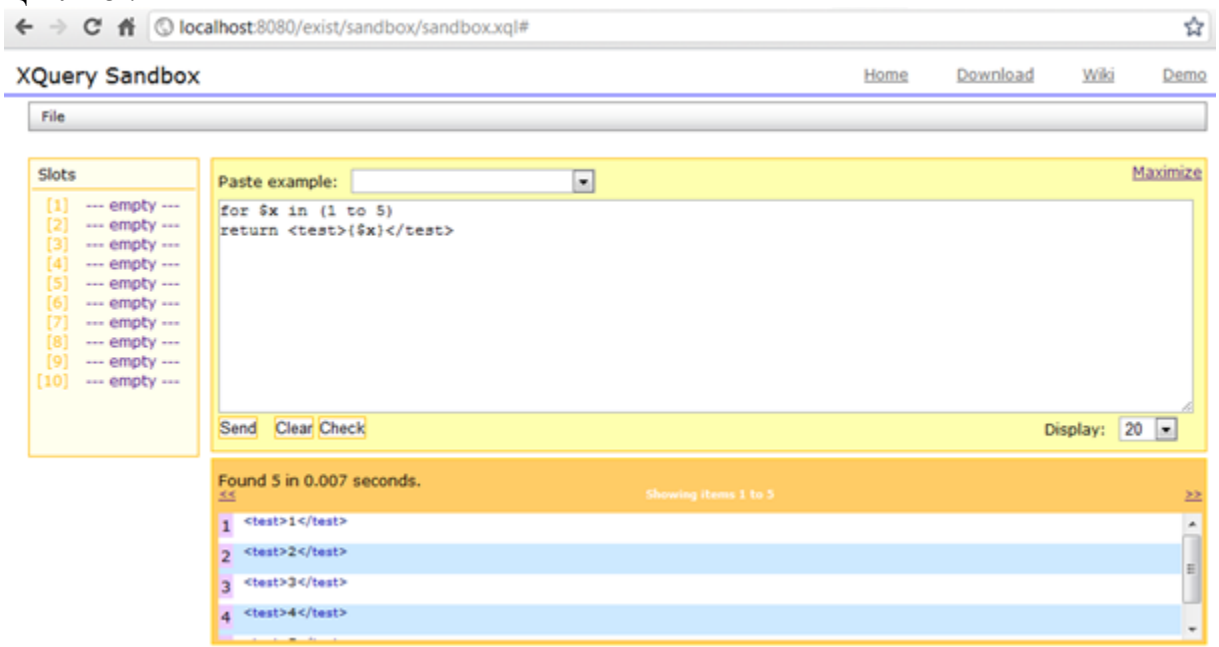

Рис. 7 Цикл For в XQuery

Теперь попробуем взять часть полученного в предыдущем пункте XML и преобразовать его в более удобочитаемый формат.

let $q:=<django-objects version="1.0">
<object pk="1" model="books.book">
<field type="CharField" name="title">On the road</field>

```
<field to="books.publisher" name="publisher" rel="ManyToOneRel">
  <natural>Addison-Wesley</natural>
  <natural>Boston</natural>
  </field>
  <field type="DateField" name="publication_date">2011-10-14</field>
 <field to="books.author" name="authors" rel="ManyToManyRel">
<object>
  <natural>Jack</natural>
  <natural>Kerouac</natural>
  </object>
  </field>
  </object>
</django-objects>

for $i in $q//object//field
return
if ($i/@name="title")
then <name>{data($i)}</name>
else if ($i/@name="publisher")
then (<publisher>
<name>{data($i//natural[1])}</name>
<location>{data($i//natural[2])}</location>
</publisher>)
else if ($i/@name="authors") then
(<author>
<first-name>{data($i//object//natural[1])}</first-name>
<second-name>{data($i//object//natural[2])}</second-name>
</author>
)
else if ($i/@name="publication_date")
then <publication-date>{data($i)}</publication-date>
else false
```

В результате получим следующую структуру:

```
<name>On the road</name>
<publisher>
      <name>Addison-Wesley</name>
      <location>Boston</location>
</publisher>
<publication-date>2011-10-14</publication-date>
<author>
      <first-name>Jack</first-name>
      <second-name>Kerouac</second-name>
</author>
```

Разберем построчно:

Сначала мы объявляем итератор по полям <field…>: *for $i in $q//object//field ($i/@name="title")* – у текущего *<field…>* берем атрибут name *(@name*) и сравниваем его с соответствующим значением. Если значения совпали, то в возвращаемое значение добавляем *<name>{data($i)}</name>*, где *data (…)* берет значение value, находящееся внутри тега (<field….> value </field>). Данное значение мы вставляем между тегами *<name>…</name>*.
Составные и сложные конструкции заключаются в круглые скобки.
Констукция IF…THEN…ELSE всегда должна присутствовать целиком.

Более подробно о языке XQuery:

- http://en.wikibooks.org/wiki/XQuery
- http://www.w3schools.com/xquery/xquery_select.asp
- Методичка Григорьева Ю.А., Гапанюка Ю.Е.

Задание

Для модели, описанной в двух предыдущих лабораторных работах:

1. Выбрать класс для сериализации
2. Сериализовать объекты данного класса
3. Произвести преобразование полученного xml в более «читабельный» формат по вышеописанному примеру.
   В отчет: описание класса, полученный после сериализации XML, код XQuery для преобразования, полученный в результате преобразования XML.