Django modelių lygmuo

Albertas Gimbutas

[1]Matematikos ir informatikos institutas
Vilniaus universitetas

2014 pavasaris

Lietuvos Python bendruomenė

- PyConLT 2014 gegužės 10d. nuo 11:00 (MIF 303): registruotis
  - Renginys nemokamas tiek žiūrovams, tiek pranešėjams.
  - Pranešėjai gali registruotis iki gegužės 3 d.
- Laiškų konferencija: python@konferencijos.lt

Išankstinis egzaminas

- Išankstinis egzaminas **2014 gegužės 12 d.** paskaitos metu.
  - Atsinešti asmens dokumentą (ar LSP) ir rašymo priemonę.
  - Egzaminas truks **1** valandą.
  - **Gegužės 15 d.** bus galima peržiūrėti ištaisytus darbus. Tą pačią dieną pažymiai bus įrašyti į duomenų bazę.
  - Atsiskaityti laboratorinius ir seminarą galima iki **gegužės 12 d**.

Turinys

Modelių kūrimas ir naudojimas

- **ORM** (object-relational mapping) - technika susijenti objektinio programavimo duomenis su reliacine duomenų baze.
- **Modeliai** - duomenų struktūrą ir elgseną aprašančios klasės.

models.py:

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()
```

views.py:

```
    tomas = Person(name="Tomas", age="25")
    tomas.save()
```

Standartiniai laukų atributai

`null`	ar leisti tuščią reikšmę duomenų bazėje
`blank`	ar leisti tuščią reikšmę formoje
`unique`	ar duomenų bazėje reikšmė turi būti unikali
`default`	standartinė reikšmė
`choices`	leidžiamos reikšmės
`verbose_name`	įvedimo lauko pavadinimas
`help_text`	pagalbinis tekstas ties įvedimo lauku
`db_index`	ar naudoti duomenų bazės indeksą
`primary_key`	ar šis laukas yra pirminis raktas
`editable`	ar rodyti naudotojui ir leisti redaguoti

- Django: modelių laukai

Pagrindiniai laukų tipai

`BooleanField`	loginės reikšmės laukas
`CharField`	simbolių eilutės laukas (būtina **max_length**)
`TextField`	teksto laukas
`DateField`	datos laukas
`DateTimeField`	datos ir laiko laukas
`IntegerField`	sveikojo skaičiaus laukas
`FloatField`	sklankaus kablelio laukas
`DecimalField`	neriboto ilgio slankaus kablelio laukas, būtina nurodyti tikslumą (**max_digits**, **decimal_places**)
`EmailField`	el. pašto laukas
`FileField`	failo laukas, reikia nurodyti **upload_to**
`ImageField`	paveiksliuko laukas
`IPAdressField`	IP adreso laukas
`URLField`	URL adreso laukas

- Django: modelių laukų tipai

Papildytas modelio pavyzdys

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()
    email = models.EmailField()
    avatar = models.ImageField()
    homepage = models.URLField()
    last_IP = models.IPAddressField()
    description = TextField()
```

Ryšiai tarp modelių

`ForeignKey`	Išorinis raktas (ryšys vienas su daug).
`OneToOneField`	Ryšys vienas su vienu.
`ManyToManyField`	Ryšys daug su daug. Realizuojamas per pagalbinę lentelę. Modelį per kurį jungti galima nurodyti parametru **through**

- Automatiškai sukuriamas ir atgalinis ryšys *<model_name>_set*. Jo pavadinimą galima perrašyti nurodant **related_name**.

```
class User(models.Model):
    name = models.CharField(max_length=30)

class Word(models.Model):
    word = models.CharField(max_length=80)

class Translation(modls.Model):
    translation = models.TextField()
    word = models.ManyToMany('Word')
    author = models.ForeignKey('User')   # translation_set
```

Objektų parinkimas

```
>>> user = User(name='Tomas')
>>> trans = Translation(translation='netikslu', user=user)
>>> trans.user
<User: Tomas>
>>> user.translation_set
<django.db.models.fields.related.RelatedManager at 0x3f8385
>>> user.translation_set.all()  # Grąžina Queryset objektą
[<Translation: netikslu>]
>>> type(user.translation_set.all())
django.db.models.query.QuerySet
>>> user.translation_set.filter(name__startswith='ne')
[<Translation: netikslu>]
>>> qs = user.translation_set.order_by('user__name')
>>> qs[0]
<Translation: netikslu>
>>> qs[-1]
Traceback (most recent call last):
AssertionError: Negative indexing is not supported.
```

Užklausos objekto metodai

`all`	Paima visus objektus
`get`	Pritaiko sąlygą ir paima vieną objektą arba sukelia klaidą
`filter`	Grąžina kelis objektus atitinkančius sąlygą
`exclude`	Atmeta objketus, atitinkančius sąlygą
`exists`	Grąžina **True**, jeigu yra sąlygą atitinkančių objektų
`order_by`	Surikiuoja objektus nurodyta tvarka
`annotate`	Kiekvienam užklausos objektui suskaičiuoja išvestinę reik
`aggregate`	Grąžina visos užklausos išvestines reikšmes

```
>>> # 3 ekvivalenčios užklausos
>>> Log.objects.filter(user=u).filter(title__contains='ne')
>>> Log.objects.filter(user=u, title__contains='ne')
>>> q = {'user': u, 'title_contains': 'ne'}
>>> Log.objects.filter(**q)
```

- Django: užklausų formavimas

Laukų patikrinimo galimybės

```
>>> Person.objects.filter(job__address__contains="MIF")
# Sąlygoje "__" nurodo gilesnę ryšio reikšmę ar operaciją
```

Filtro sąlygų operacijos:

`exact, iexact`	Reikšmė lygiai kokia nurodyta*
`contains, icontains`	Talpina simbolių eilutę*
`startswith, endswith`	Prasideda/baigiasi nurodytaja eilute*
`in`	Reikšmė yra nurodytame sąraše
`gt, lt, gte, lte`	Reikšmė >, <, >=, <= už nurodytąją
`range`	Reikšmė yra tarp nurodytųjų
`isnull`	Reikšmė nenustatyta (lygi None)
`regex`	Reikšmė atitinka reguliariąją išraišką

- * - jeigu operacija prasideda **i** tai ji nejautri didžiosioms raidėms
- Django: užklausų filtravimas

Anotavimo ir agregavimo funkcijos

- Funkcijos: `Sum`, `Count`, `Max`, `Min`, `Avg`, `Variance`, `StdDev`

```
from django.db import models
from math import factorial

class Factorial(models.Model):
    value = models.IntegerField()

for i in range(5):
    Factorial.objects.create('value'=factorial(i))
```

```
>>> from django.db.models import Count, Max, Sum
>>> Factorial.objects.aggregate(Count('value'),
                                Sum('value'),
                                Max('value'))
{'value__count': 5, 'value__sum': 153, 'value__max': 120}
```

Užklausų objektas Q

```
>>> from django.db.models import Q
>>> Q(title__icontains='Kivy')
>>>
>>> # Pavadinimas talpina 'Kivy' arba 'App'
>>> Q(title__icontains='Kivy') | Q(title__icontans='App')
>>>
>>> # Pavadinimas prasideda su 'Pillow' ir talpina 'fork'
>>> Q(title__startswith='Pillow', title__icontains='fork')
>>>
>>> # ~ žymi priešingą užklausą
>>> ~Q(created__year='2013')    # sukurtas ne 2013 metais
>>>
>>> Seminar.objects.filter(~Q(created__year='2013'))
```

- Django: užklausos su Q objektu

Migracijos

- **Migracija** - tai duomenų arba duomenų bazės schemos pakeitimas, neprarandant duomenų.
- Iki Django 1.6 versijos pagrindinis migracijų modulis buvo **South**, nuo 1.7 versijos Django turi savo migracijas.

Įprastas migracijos kūrimas su South:

```
$ bin/django startmigration projectname
$ bin/django schemamigration projectname --auto
Created 0006_auto__add_field_topic_tmp.py.
$ bin/django migrate projectname
Running migrations for projectname:
 > hack4lt:0006_auto__add_field_person_email
  - Loading initial data for projectname.
  Installed 0 object(s) from 0 fixture(s)
```

- Django: migracijos

Migracijos failo turinys

migrations/0006_auto__add_field_person_email.py

```
from south.v2 import SchemaMigration

class Migration(SchemaMigration):
    def forwards(self, orm):
        db.add_column(u'projectname_person', 'email',
                                        keep_default=False)
    def backwards(self, orm):
        db.delete_column(u'projectname_person', 'email')
```