Học Django

1. Tạo project django

Cmd: django-admin startproject tên project

1. Start server:

Command trong vs code: python manage.py runserver

1. Tạo app

* Python manage.py start app tên app

Khi tạo xong 1 app thì ta phải đăng ký app đó trong file setting.py



Đường dẫn trang web sẽ dc định nghĩa trong file urls.py trong thư mục chính

Với mỗi app ta có thể định nghĩa các file urls.py riêng biệt nhưng sau đó trong file urls.py ở thư mục gốc chúng ta phải include vào. **Ví dụ poll là tên app, urls là file path**



Sau đó vào trình duyệt gõ 127.0.0.1:8000/path

Chức năng Model để tương tác với CSDL

Trước hết tạo cơ sở dl ta phải tạo cái migration trước để nó lưu lại những thay đổi trong csdl. 

Để tạo migration ta gõ lệnh. Python manage.py makemigrations

Để tạo database gõ lệnh: Python manage.py migrate

* Tạo user admin: python manage.py createsuperuser
* Khi muốn quản lý data thì ta vào phần admin.py
* Trong file admin.py import class trong phần models vào
* Comand: admin.site.register(tên class)
* Mặc định thì django chỉ hiển thị một phần của database ví dụ như title, để hiển thị thêm ta có thể bổ sung các list list\_display, list\_filter, search\_fieldsm,…
* 

Khi tương tác với csdl, khi muốn view ra nội dung trong database ta có thể sử dụng một số hàm tương tác để biến nội dung trong database thành các mã html để có thể hiển thị chính xác nội dung cần thể hiện.

Ví dụ như thẻ: safe: Thực thi nội dung trong một trường database có cú pháp là một mã HTML

Hoặc linrbreak: Tạo nội dung xuống dòng, nếu như trong nội dung của database muốn xuống dòng, còn nếu ko thì nó sẽ viết liền



**Loại bỏ hardcode url**

Trong đường dẫn path có thể có những đường dẫn tương đối giống nhau, nên khi gọi view sẽ dễ bị nhầm lẫn, vậy ta có cách là trong phần path của file url ta đặt tên cho path đó, rồi sau đó gọi tên nó trong file template





Khi ta bị lỗi 404,505 thì django sẽ trả về lỗi mặc định có chi tiết lỗi thì như thế sẽ bị lỗi bảo mật, và ta có thể tự định nghĩa lỗi trả về. Ví dụ:





Tham số Debug là khi ta phát triển web thì giá trị là True, còn khi deploy thì ta cho là false

**Mô hình MVC có nghĩa là Model = models**

View = Template

Controler = views

Vậy trong django còn gọi là mô hình MTV

* Khi chạy server thì ta dùng lệnh python manage.py runserver <port> ví dụ 8888
* Để vào dc trang admin thì ta phải tạo 1 tài khoản admin bằng câu lệnh: python manage.py createsuperuser
* File url của project sẽ trỏ đến file url của các app, sau đó trong file url của app mới dẫn đến các view

url(r'^(?P<question\_id>[0-9]+)/results/$', views.results, name='results'),

Chuỗi Regex (?P<question\_id>[0-9]+) cho Django biết có một chuỗi con trong đoạn URL có dạng một con số (có 1 hoặc nhiều chữ số) và biến được truyền vào phương thức detail() có tên là question\_id.

* Sử dụng module “loader” để get template
* from django.template import loader
* template = loader.get\_template('polls/index.html')

|  |
| --- |
| * return HttpResponse(template.render(context, request)) |

Sau đó chúng ta gọi đến phương thức template.render() để tạo nội dung HTML có sử dụng template. Tham số đầu tiên là nội dung HTML trả về, tham số thứ 2 là đối tượng request được gửi đến.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | from django.shortcuts import render    def index(request):      latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]      context = {          'latest\_question\_list': latest\_question\_list,      }      return render(request, 'polls/index.html', context) |

Ngoài ra bạn cũng có thể dùng hàm render() để trả về một đối tượng HttpResponse một cách trực tiếp luôn, với tham số thứ nhất là đối tượng request, tham số thứ 2 là đường dẫn đến file template, tham số thứ 3 là nội dung HTML trả về.

**Đặt namespace cho URL**

 Chúng ta có thể tham chiếu đến các đối tượng url đã định nghĩa trong file urls.py của ứng dụng. Khi chúng ta tạo các đối tượng url trong file urls.py, tham số thứ 3 là name, tham số này do chúng ta tự đặt, và Django cho phép chúng ta tham chiếu đến chúng trong các file template.

Ví dụ: url(r'^details/(?P<question\_id>[0-9]+)/$', views.detail, name='detail'),

Index.html

|  |  |
| --- | --- |
| 1  2  3 | <a href="{% url 'detail' question.id %}">      {{ question.question\_text }}  </a> |

Bằng cách này chúng ta có thể tham chiếu đến đối tượng url trong file urls.py, và khi nào cần thay đổi URL mới thì chúng ta chỉ cần thay đổi trong file urls.py là được:

Để đặt tên namespace cho các đối tượng url thì chúng ta chỉ cần đặt giá trị cho biến app\_nametrong file urls.py là được.

from django.conf.urls import url

from . import views

app\_name = "polls"

### STATIC\_URL

URL để sử dụng khi tham chiếu đến các tệp tĩnh nằm trong STATIC\_ROOT.

**STATICFILES\_DIRS**

Cài đặt này xác định các vị trí bổ sung mà ứng dụng staticfiles sẽ đi qua nếu công cụ tìm FileSystem Downloader được bật

### STATIC\_ROOT

Đường dẫn tuyệt đối đến thư mục nơi “collectstatic” sẽ thu thập các “static file” để triển khai

Đặt cài đặt STATIC\_ROOT vào thư mục mà từ đó bạn muốn sử dụng các tệp này

Chạy lệnh  [**collectstatic**](https://docs.djangoproject.com/en/2.1/ref/contrib/staticfiles/#django-admin-collectstatic) :

$ python manage.py collectstatic

Điều này sẽ sao chép tất cả các tệp từ các thư mục tĩnh của bạn vào thư mục STATIC\_ROOT.

Tức là nó sẽ sao chép tất cả các file khai báo trong **STATICFILES\_DIRS đến thư mục** STATIC\_ROOT

### Ví dụ:

STATICFILES\_DIRS = [

"/home/special.polls.com/polls/static",

"/home/polls.com/polls/static",

"/opt/webfiles/common",

]

**URL:**

**Mỗi yêu cầu url phải kết thúc bằng một dấu gạch chéo**

**VD: url(‘articles/2003/’** ,view**)**

Sử dụng biểu thức chính quy (Regular expressions)

Nếu cú ​​pháp đường dẫn và trình chuyển đổi là không đủ để xác định các mẫu URL của bạn, bạn cũng có thể sử dụng các biểu thức thông thường. Để làm như vậy, hãy sử dụng re\_path () thay vì path ().

Trong các biểu thức chính quy của Python, cú pháp cho các nhóm biểu thức chính quy được đặt tên là (? P <name> pattern), trong đó name là name của group và pattern là một số pattern phù hợp.

VD:

**from** **django.urls** **import** path, re\_path

**from** **.** **import** views

urlpatterns = [

path('articles/2003/', views.special\_case\_2003),

re\_path(r'^articles/(?P<year>[0-9]**{4}**)/$', views.year\_archive),

re\_path(r'^articles/(?P<year>[0-9]**{4}**)/(?P<month>[0-9]**{2}**)/$', views.month\_archive),

re\_path(r'^articles/(?P<year>[0-9]**{4}**)/(?P<month>[0-9]**{2}**)/(?P<slug>[\w-]+)/$', views.article\_detail),

]

Tại bất kỳ thời điểm nào, các mô hình url của bạn có thể “include” các mô-đun URL khác của URL. Bất cứ khi nào Django gặp phải include (), nó sẽ loại bỏ bất kỳ phần nào của URL khớp với điểm đó và gửi chuỗi còn lại đến URLconf kèm theo để xử lý thêm.

## **Path converters**

・str - Khớp với bất kỳ chuỗi không trống nào, ngoại trừ dấu phân cách đường dẫn, '/'. Đây là mặc định nếu một trình chuyển đổi không kèm theo trong biểu thức.

・int - Khớp 0 hoặc bất kỳ số nguyên dương nào. Trả lại một giá trị int.

VD: path('blog/page<int:num>/', views.page),

・slug - Ghép nối bất kỳ chuỗi sên nào bao gồm các chữ cái hoặc số ASCII, cộng với dấu gạch nối và ký tự gạch dưới.

VD: **building-your-1st-django-site**.

・UUID: Ghép một UUID được định dạng. Để ngăn nhiều URL ánh xạ vào cùng một trang, các dấu gạch ngang phải được bao gồm và các chữ cái phải là chữ thường.

Ví dụ: 075194d3-6885-417e-a8a8-6c931e272f00.

・path - Khớp với bất kỳ chuỗi không trống nào, kể cả dấu phân cách đường dẫn, '/'. Điều này cho phép bạn khớp với một đường dẫn URL hoàn chỉnh thay vì chỉ một đoạn của đường dẫn URL như với str.

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path('articles/2003/', views.special\_case\_2003),

path('articles/<int:year>/', views.year\_archive),

path('articles/<int:year>/<int:month>/', views.month\_archive),

path('articles/<int:year>/<int:month>/<slug:slug>/', views.article\_detail),

Ví dụ:

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path('<page\_slug>-<page\_id>/history/', views.history),

path('<page\_slug>-<page\_id>/edit/', views.edit),

path('<page\_slug>-<page\_id>/discuss/', views.discuss),

path('<page\_slug>-<page\_id>/permissions/', views.permissions),

]

Chúng ta có thể cải thiện điều này bằng cách chỉ nêu tiền tố đường dẫn chung một lần và nhóm các hậu tố khác nhau:

**from** **django.urls** **import** include, path

**from** **.** **import** views

urlpatterns = [

path('<page\_slug>-<page\_id>/', include([

path('history/', views.history),

path('edit/', views.edit),

path('discuss/', views.discuss),

path('permissions/', views.permissions),

])),

]

URLconfs có một hook cho phép bạn chuyển các đối số bổ sung cho các hàm xem của bạn, như một Dictionary Python.

Hàm path () có thể lấy một đối số thứ ba tùy chọn phải là một dictionary của các đối số từ khóa bổ sung để chuyển đến hàm xem.

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path('blog/<int:year>/', views.year\_archive, {'foo': 'bar'}),

]

Trong ví dụ này, đối với yêu cầu tới / blog / 2005 /, Django sẽ gọi view.year\_archive (request, year = 2005, foo = 'bar').

Kỹ thuật này được sử dụng trong khung cung cấp để chuyển siêu dữ liệu và các tùy chọn cho các view

-VIEWS

Phương thức trả về đối tượng HttpResponse từ thư viện

from django.http import HttpResponse

- Khi làm việc với Templates ta thường sử dụng một trong hai thư viện

from django.template import loader

from django.shortcuts import render

- Tham số được kèm trong lời gọi hàm HttpResponse bao gồm có tham số mặc định “request”,”Templates”, và content thường là một dictionnary.

Ví dụ:

Sử dụng thư viện “loader”

def index(request):

    latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

    template = loader.get\_template('polls/index.html')

    context = {

        'latest\_question\_list': latest\_question\_list,

    }

    return HttpResponse(template.render(context, request))

Sử dụng thư viện “render” trả về một đối tượng HttpResponse một cách trực tiếp

from django.shortcuts import render

def index(request):

    latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

    context = {

        'latest\_question\_list': latest\_question\_list,

    }

    return render(request, 'polls/index.html', context)

- [**HttpResponseRedirect**](https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpResponseRedirect): Trả về một URL thích hợp cho các đối số được truyền.

Có các tham số là:

- [get\_absolute\_url()](https://docs.djangoproject.com/en/2.2/ref/models/instances/#django.db.models.Model.get_absolute_url): hàm [get\_absolute\_url()](https://docs.djangoproject.com/en/2.2/ref/models/instances/#django.db.models.Model.get_absolute_url) của model sẽ được gọi

- một view có thể được gọi với tham số :reverse() sẽ được sử dụng để phan giải ngược tên

- Một URL tuyệt đối hoặc tương đối, sẽ được sử dụng như là dành cho vị trí chuyển hướng.

VD:

HttpResponseRedirect(reverse('polls:results', args=(question1.id,)))

HttpResponseRedirect nhận một đối tượng url thông qua hàm reverse(), hàm này sẽ trả về một đường dẫn có dạng như /polls/[ args ]/results/.

Hàm Reverse () có thể reverse rất nhiều mẫu biểu thức chính quy cho URL, nhưng không phải mọi mẫu có thể. Hạn chế chính tại thời điểm này là mẫu không thể chứa các lựa chọn thay thế bằng ký tự thanh dọc ("|"). khá vui khi sử dụng các mẫu như vậy để khớp với các URL đến và gửi chúng đến các chế độ xem, nhưng bạn không thể đảo ngược các mẫu đó.

- Templates

Sử dụng ngôn ngữ HTML xen thêm các câu lệnh Python với những ký hiệu đặc biệt gọi là jinja2

Ví dụ:

<h1>{{ question.question\_text }}</h1>

{% if error\_messsage %}

<strong>{{ error\_messsage }}</strong>

{% endif %}

<form action="{% url 'polls:vote' question.id %}" method="post">

    {% csrf\_token %}

    {% for choice in question.choice\_set.all %}

        <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />

        <label for="choice{{ forloop.counter }}">{{ choice.choice\_text }}</label>

    {% endfor %}

    <input type="submit" value="Vote" />

</form>

Phần bôi vàng là những thành phần “KEY” nằm trong dictionary, file html lấy ra sử dụng.

Bất cứ bảng nào có một bảng khác chứa khóa ngoại tham chiếu đến khóa chính của nó đều sẽ có một thuộc tính được tạo tự động là một tập hợp các đối tượng của bảng kia. Ở đây bảng Choice chứa khóa ngoại tham chiếu đến bảng Question, do đó bảng Question sẽ có một danh sách các đối tượng Choice, chúng ta có thể lấy danh sách này qua choice\_set.all() (tên danh sách được đặt theo <tên bảng>\_set).

- Tức là từ bảng mà có một bảng khác chứa khóa ngoại tham chiếu đến khóa chính của nó thì đều có thể từ bảng đó mà thao tác với bảng kia.

Ví dụ:

q=bangkhoachinh.objects.get(pk=1)

q.bangkhoangoai\_set.all(): Danh sách các đối tượng bangkhoangoai

q.bangkhoangoai\_set.create(tencot =”giatri”,…): Tạo các đối tượng cho bangkhoangoai

q.bangkhoangoai\_set.count():Lấy số lượng các bản ghi

q.bangkhoangoai\_set.filter():Lọc lấy bản ghi để hiển thị

q.bangkhoangoai\_set.filter():Lọc lấy bản ghi và lưu vào một biến nào đó.

q.save():Lưu vào cơ sở dữ liệu

c.delete():Xóa một bản ghi

Nhưng chỉ áp dụng với những trường mà có q.id = bangkhoangoai.bangkhoachinh\_id

|  |
| --- |
| {% csrf\_token %} |

câu lệnh trên sẽ giúp website của chúng ta chống lại kiểu tấn công *CSRF*

*Các lớp View có sẵn trong django*

|  |
| --- |
| from django.views import generic |

Đầu tiên chúng ta import module generic.

class IndexView(generic.ListView):

class DetailView(generic.DetailView):

Chúng ta sẽ sử dụng 2 loại view là ListView và DetailView, ListView lưu trữ danh sách các đối tượng, DetailView lưu thông tin về một đối tượng cụ thể.

model = Question

Mỗi lớp View cần biết về mô hình dữ liệu mà nó sẽ lưu trữ thông qua thuộc tính model. Khi đã biết loại model mà mình sẽ dùng, các view này tự động “nghĩ” rằng khóa chính trong CSDL có tên là pk, do đó trong các đối tượng url chúng ta gửi lên tham số có tên là pk.

-Chúng ta phải sửa lại các đường dẫn url :

*url(r'^(?P<pk>[0-9]+)/results/$', views.ResultsView.as\_view(), name='results'),*

Mặc định các lớp View này cũng “nghĩ” rằng template sẽ được dùng có dạng <app\_name>/<model\_name>\_detail.html (hoặc <app\_name>/<model\_name>\_list.html), nếu không có file nào như vậy tồn tại thì Django sẽ tự động tạo các file này và sử dụng, chúng ta có thể “bảo” Django sử dụng template do chúng ta tự viết bằng cách gán vào thuộc tính template\_name, như thế Django sẽ không tạo template cho nó nữa.

template\_name = 'polls/detail.html'

Trong hàm detail() cũ, chúng ta tự khai báo các biến question và latest\_question\_list để dùng trong template, mặc định các biến này đã có sẵn trong các lớp DetailView. Tuy nhiên đối với hàm ListView thì biến mặc định lại có tên là question\_list, nên để Django dùng tên do chúng ta tự đặt thì chúng ta phải gắn tên đó vào thuộc tính context\_object\_name hoặc bạn phải dùng tên do Django đặt trong các file template.

class IndexView(generic.ListView):

    ...

    context\_object\_name = 'latest\_question\_list'

*-* ***Static file***

Ngoài nội dung HTML được sinh ra bởi server thì một ứng dụng web còn cần đến các file bổ sung khác, chẳng hạn như các file hình ảnh, Javascript, CSS… Trong Django thì các file này được gọi là file tĩnh.

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" />

Dòng {% load staticfiles %} sẽ tự động gán đường dẫn đến thư mục mysite/polls/static/ vào một biến có tên static do Django tự đặt, từ đó bạn chỉ cần lấy biến static là có thể lấy được đường dẫn tuyệt đối đến thư mục này.

Sau đó chúng ta chỉ cần lấy biến static trong cặp thẻ {% %} ra và gắn thêm đường dẫn đến các file css, js… của bạn.

<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" />

## Tùy chỉnh form

Khi chúng ta đăng ký các lớp model Question và Choice bằng hàm admin.site.register(), Django sẽ dựa vào các thuộc tính mà chúng ta khai báo trong các lớp đó để hiển thị form trên trang admin. Chúng ta có thể quy định Django chỉ chọn một số thuộc tính được phép hiện ra để chỉnh sửa, thay vì hiện ra hết, bởi vì khi làm một ứng dụng có sử dụng CSDL, có những thuộc tính mà bạn muốn máy tính tự động sinh ra chứ không phải do con người nhập vào, chẳng hạn như ID, ngày giờ…

Chúng ta sẽ thay đổi bằng cách sửa lại code trong file polls/admin.py.

from django.contrib import admin

# Register your models here.

from .models import Question, Choice

class QuestionAdmin(admin.ModelAdmin):

    fields = ['pub\_date', 'question\_text']

admin.site.register(Question, QuestionAdmin)

admin.site.register(Choice)

Việc này làm rất đơn giản, chúng ta chỉ cần viết một lớp kế thừa từ lớp admin.ModelAdmin, ở đây mình đặt tên là QuestionAdmin, sau đó khai báo list có tên là fields có các item là tên các thuộc tính trong model mà chúng ta muốn hiện ra trong trang Admin, sau đó truyền lớp này vào hàm admin.site.register().

*MODEL*

## Sử dụng model

Sau khi đã định nghĩa xong các model thì chúng ta phải thiết lập để Django sử dụng các model đó, bằng cách chỉnh sửa biến INSTALLED\_APPS trong file settings.py.

## Các trường

Thành phần quan trọng và cũng là thành phần bắt buộc phải có trong một model là danh sách các trường hay các thuộc tính của một lớp. Chú ý khi đặt tên bạn nhớ tránh đặt tên trùng với các từ khóa có sẵn trong Django.

## Tham số trong Field

Ngoài các tham số bắt buộc thì còn có một số tham số tùy chọn nữa, ở đây mình chỉ giới thiệu một số:

* null: có giá trị True hoặc False. Tham số này cho Django biết sẽ dùng giá trị NULL thay cho các bản ghi có giá trị rỗng, tức là trong CSDL mà kiểu chuỗi mà có giá trị "" thì sẽ thay bằng NULL.
* blank: có giá trị True hoặc False. Tham số này cho Django biết cột tương ứng trong bảng CSDL được phép NULL hay NOT NULL.
* default: thiết lập giá trị mặc định cho trường.
* help\_text: đây một giá trị chuỗi dùng để hiển thị một đoạn text ngắn mô tả về trường đó, vd như khi di chuột vào cột trên form HTML thì hiện ra đoạn text.
* primary\_key: True hoặc False, chỉ định trường đó có phải là khóa chính hay không. Như bạn đã biết, nếu bạn không chỉ định trường nào làm khóa chính thì Django sẽ tự động tạo một trường AutoField để làm khóa chính luôn. Có một điều nữa là khi bạn thay đổi giá trị khóa chính của một model trong Python thì Django không báo lỗi mà thay vào đó là tạo một đối tượng model mới với khóa chính mới.

## Khóa chính tự tạo

Khóa chính do Django tự tạo ra có dạng sau:

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

## Quan hệ

Sức mạnh của các cơ sở dữ liệu quan hệ nằm ở tính năng mối quan hệ giữa các bảng. Django hỗ trợ 3 loại quan hệ phổ biến của CSDL quan hệ là: **many-to-one** (**một-nhiều**), **many-to-many**(**nhiều-nhiều**), **one-to-one** (**một-một**).

## Quan hệ Many-to-one

Chúng ta đã làm việc với kiểu quan hệ này trong các bài trước bằng cách dùng lớp django.db.models.ForeignKey. Tham số bắt buộc phải có của ForeignKey là model cha của nó.

## Quan hệ Many-to-many

Để sử dụng mối quan hệ này thì chúng ta sử dụng lớp ManyToManyField với tham số bắt buộc là tên model mà nó sẽ trỏ đến.

Cũng giống như với ForeignKey, bạn cũng có thể dùng quan hệ đệ quy với ManyToManyField.

## Thuộc tính của một mối quan hệ many-to-many

Một mối quan hệ không đơn giản chỉ là quan hệ giữa bảng này với bảng kia, mà chúng còn có các thông tin riêng nữa.

Ví dụ chúng ta có model Person và Group có quan hệ many-to-many, ý nghĩa là một người có thể tham gia nhiều group và một group có thể có nhiều người, ý nghĩa của quan hệ này không chỉ có bao nhiêu đó mà còn có các thông tin khác như người đó tham gia vào nhóm vào ngày nào, lý do tham gia chẳng hạn…

Để thêm các trường vào một mối quan hệ thì chúng ta sẽ dùng đến một tham số tùy chọn trong hàm khởi tạo ManyToManyField là through.

from django.db import models

class Person(models.Model):

    name = models.CharField(max\_length=128)

class Group(models.Model):

    name = models.CharField(max\_length=128)

    members = models.ManyToManyField(Person, through='Membership')

class Membership(models.Model):

    person = models.ForeignKey(Person, on\_delete=models.CASCADE)

    group = models.ForeignKey(Group, on\_delete=models.CASCADE)

    date\_joined = models.DateField()

    invite\_reason = models.CharField(max\_length=64)

Chúng ta phải định nghĩa một model khác lưu trữ các thông tin của một mối quan hệ, tham số through sẽ nhận tên của model này.

Trong Django thì các model kiểu này (Membership) được gọi mà model trung gian (Intermediate model), trong đó bạn phải khai báo rõ ràng các model tham gia vào mối quan hệ này, ở đây là 2 đối tượng ForeignKey: person và group.

Chúng ta phải định nghĩa một model khác lưu trữ các thông tin của một mối quan hệ, tham số through sẽ nhận tên của model này.

Trong Django thì các model kiểu này (Membership) được gọi mà model trung gian (Intermediate model), trong đó bạn phải khai báo rõ ràng các model tham gia vào mối quan hệ này, ở đây là 2 đối tượng ForeignKey: person và group.

bạn phải khai báo rõ ràng các thông tin của mối quan hệ đó ra (bằng cách dùng hàm khởi tạo

ví dụ: m1 = Membership(person=ringo, group=beatles,

... date\_joined=date(1962, 8, 16),

... invite\_reason="Needed a new drummer.")

## Mối quan hệ one-to-one

Để định nghĩa một mối quan hệ one-to-one thì chúng ta sử dụng lớp OneToOneField, lớp này cũng nhận tham số là tên của model mà nó sẽ trỏ tới.

## Tham chiếu model thông qua file

Bạn có thể định nghĩa các lớp model trong các file khác nhau và khi cần tham chiếu đến chúng thì chỉ cần import vô là được.

## Metadata

Các lớp trong Django có thể chứa các metadata (siêu dữ liệu), đây là các thông tin lưu trữ về các thiết lập của model, các thông tin này không được dùng để tạo bảng.

Ví dụ:

from django.db import models

class Ox(models.Model):

    horn\_length = models.IntegerField()

    class Meta:

        ordering = ["horn\_length"]

        db\_table = "oxen"

## Thừa kế model

Model trong Django cũng có thể thừa kế lẫn nhau. Chúng ta có thể viết các lớp model cơ sở (hay model cha) rồi sau đó viết các model khác thừa kế từ lớp model cơ sở này. Các lớp model cơ sở phải được thừa kế từ lớp django.db.models.Model, rồi từ đó các lớp con sẽ kế thừa từ lớp cha này. Model cha có thể có bảng riêng hoặc không có.

Có 3 cách để một model được thừa kế trong Django:

* Thừa kế từ lớp trừu tượng.
* Thừa kế đa bảng.
* Dùng lớp Proxy.

### Thừa kế từ lớp trừu tượng

Chúng ta sử dụng lớp trừu tượng khi muốn model cha chỉ có nhiệm vụ là lưu trữ các thông tin dùng chung trong các model con chứ không phải nhằm mục đích lưu các thông tin thật trong CSDL. Để một model là “trừu tượng” thì chúng ta gán thuộc tính abstract = True của lớp nội Meta.

Ví du:

from django.db import models

class CommonInfo(models.Model):

    name = models.CharField(max\_length=100)

    age = models.PositiveIntegerField()

    class Meta:

        abstract = True

class Student(CommonInfo):

    home\_group = models.CharField(max\_length=5)

Model Student thừa kế từ model trừu tượng là CommonInfo, Student sẽ bao gồm 3 trường là name, age và home\_group.

Khi chúng ta dùng lệnh manage.py migrate, chỉ có model Student mới được tạo bảng cùng với 3 trường của nó. Khi một model con thừa kế model cha mà không khai báo lớp nội Meta thì sẽ tự động thừa kế lớp nội đó từ model cha. Nhưng nếu muốn các model vẫn có thể có lớp nội riêng hoặc kế thừa và thêm các thuộc tính mới từ model cha.

## Thừa kế đa bảng

Thừa kế đa bảng chỉ khác kiểu thừa kế lớp trừu tượng ở chỗ là lớp cha ở đây là lớp “thật” chứ không phải lớp trừu tượng. Tức là các model cha bây giờ sẽ có cả các bảng riêng của chúng trong CSDL.

Khác với kiểu kế thừa từ lớp trừu tượng, ở lớp trừu tượng thì các trường của lớp cha sẽ nằm cùng bảng với các lớp con vì lớp cha không được tạo bảng riêng, với thừa kế đa bảng thì các lớp con cũng sẽ kế thừa các trường của lớp cha nhưng các trường của lớp cha sẽ nằm trong bảng riêng của lớp cha.

Ví dụ:

>>> Place.objects.filter(name="Bob's Cafe")

>>> Restaurant.objects.filter(name="Bob's Cafe")

Chúng ta vẫn có thể truy xuất các thông tin từ cả 2 lớp con và lớp cha như thường.

## Lớp Proxy – lớp ủy quyền

Django cho phép chúng ta tạo ra các lớp Proxy (lớp ủy quyền), các lớp này kế thừa từ lớp cha nhưng không được tạo bảng mới trong CSDL, những lớp con này sẽ chỉ dùng để lưu các phương thức mới chứ không lưu các thuộc tính mới. Bạn có thể thực hiện thêm, sửa, xóa dữ liệu được kế thừa từ lớp cha, những dữ liệu này vẫn lưu ở bảng cha.

Ví dụ:

from django.db import models

class Person(models.Model):

    first\_name = models.CharField(max\_length=30)

    last\_name = models.CharField(max\_length=30)

class MyPerson(Person):

    class Meta:

        proxy = True

    def do\_something(self):

        # ...

        pass

Bất cứ thao tác cập nhật dữ liệu nào từ lớp MyPerson cũng sẽ là thao tác trực tiếp với dữ liệu từ lớp Person.

## Đa thừa kế

Vì Django được phát triển từ Python nên các model trong Django có thể thừa kế từ nhiều model cha. Lớp Meta của model con chỉ có thể thừa kế từ lớp Meta của lớp cha đầu tiên. Ngoài ra khi sử dụng đa thừa kế mà các model cha có cùng chung tên thuộc tính khóa chính thì Django sẽ báo lỗi. Do đó khi sử dụng đa thừa kế, chúng ta nên khai báo các trường làm khóa chính một cách rõ ràng.

Ví dụ:

class Article(models.Model):

    article\_id = models.AutoField(primary\_key=True)

    ...

class Book(models.Model):

    book\_id = models.AutoField(primary\_key=True)

    ...

class BookReview(Book, Article):

    pass

## Tạo đối tượng

Mỗi bản ghi hay mỗi dòng trong từng bảng CSDL sẽ tương ứng với một đối tượng cụ thể. Để tạo một bản ghi trong CSDL thì chúng ta chỉ cần tạo một đối tượng tương ứng với trong Python rồi gọi phương thức save() là xong. Vd:

>>> from blog.models import Blog

>>> b = Blog(name='J.R.R Blog', tagline='All the latest J.R.R news.')

>>> b.save()

Đoạn code trên sẽ thực hiện câu lệnh INSERT, bạn phải gọi phương thức save() thì dữ liệu mới được cập nhật lên CSDL.

Ngoài ra nếu bạn muốn vừa tạo đối tượng vừa cập nhật thẳng lên CSDL luôn thì dùng phương thức create() trong đối tượng objects, đối tượng này là một đối tượng tĩnh do Django tạo ra cho chúng ta để đơn giản hóa việc thao tác với CSDL.

>>> Blog.objects.create(name='J.R.R Blog', tagline='All the latest J.R.R news.')

## Cập nhật đối tượng

Bạn có thể thay đổi giá trị của các đối tượng trong Python rồi chỉ cần gọi phương thức save() là Django sẽ cập nhật mới trong CSDL.

>>> b.name = 'New name'

>>> b.save()

### Cập nhật các thuộc tính khóa ngoại

Việc gán giá trị cho các thuộc tính khóa ngoại **many-to-one** (ForeignKey) rất đơn giản, bạn chỉ cần gán thuộc tính khóa ngoại với đối tượng cần trỏ đến là xong.

Ví dụ:

>>> from django.utils import timezone

>>> e = Entry(blog=b,

... headline='The lord of the rings',

... body\_text='Chapter 0',

... pub\_date=timezone.now(),

... mod\_date=timezone.now(),

... n\_comments=0,

... n\_pingbacks=0,

... rating=0)

>>> e.save()

Đối với các thuộc tính khóa ngoại **many-to-many** thì chúng ta không được phép gán trực tiếp như đối với ForeignKey mà phải dùng phương thức riêng của thuộc tính khóa ngoại đó, bởi vì khóa ngoại ForeignKey chỉ lưu trữ các đối tượng đơn lẻ trong khi khóa ngoại ManyToManyField thì lại lưu trữ một danh sách các đối tượng khác nhau:

>>> e.authors.add(a)

>>> e.save()

## Truy xuất dữ liệu

Ngoài cách thao tác với dữ liệu thông qua từng bản ghi như trên thì Django còn cung cấp cho chúng ta một đối tượng thuộc lớp django.db.models.manager.Manager để chúng ta thao tác với chính bảng được tạo ra đó, đối tượng này mặc định được đặt tên là objects.

## Lấy toàn bộ dữ liệu

Khi chúng ta lấy dữ liệu từ CSDL về thì objects sẽ trả về một đối tượng QuerySet.

>>> Blog.objects.all()

>>> [<Blog: New name>]

## Lọc dữ liệu

Có 2 phương thức hỗ trợ lọc dữ liệu là filter() và exclude(). Tham số của 2 phương thức là một biểu thức tìm kiếm, trong đó phương thức filter() sẽ trả về dữ liệu khớp với biểu thức tìm kiếm, còn exclude() sẽ trả về dữ liệu không khớp.

>>> Entry.objects.filter(pub\_date\_\_year==2016)

## Lấy một dòng dữ liệu

Django cung cấp phương thức get() giúp bạn lấy một bản ghi duy nhất, tham số của phương thức này cũng là một biểu thức tìm kiếm như 2 phương thức filter() và exclude(). Bạn cũng chỉ dùng get() khi biết dữ liệu trả về chỉ có 1 bản ghi thôi, nếu có nhiều bản ghi khớp với biểu thức tìm kiếm thì get() sẽ báo lỗi exception MultipleObjectsReturned.

>>> one\_entry = Entry.objects.get(pk=1)

Một sự khác nhau nữa giữa get() và filter()/exclude() là nếu get() không tìm thấy bản ghi nào thì sẽ trả về một lỗi exception là DoesNotExist còn 2 phương thức kia sẽ trả về một đối tượng QuerySet rỗng.

Ngoài ra:

Entry.objects.filter(pub\_date\_\_year=2005).order\_by('-pub\_date', 'headline')

Author.objects.distinct()

Blog.objects.filter(name\_\_startswith='Beatles').**values**()

[{'id': 1, 'name': 'Beatles Blog', 'tagline': 'All the latest Beatles news.'}]

Author.objects.values\_list('name', 'entry\_\_headline')

[('Noam Chomsky', 'Impressions of Gaza'),

('George Orwell', 'Why Socialists Do Not Believe in Fun'),

('George Orwell', 'In Defence of English Cooking'),

('Don Quixote', None)]

**>>>** Entry.objects.values\_list('headline', flat=**True**).get(pk=1)

'First entry'

Entry.objects.dates('pub\_date', 'year')

[datetime.date(2005, 1, 1)]

**>>>** Entry.objects.dates('pub\_date', 'month')

[datetime.date(2005, 2, 1), datetime.date(2005, 3, 1)]

**>>>** Entry.objects.dates('pub\_date', 'day')

[datetime.date(2005, 2, 20), datetime.date(2005, 3, 20)]

## Lọc số lượng bản ghi

Bạn có thể lọc số lượng bản ghi cần lấy theo cú pháp của Python.

Ví dụ:

>>> Entry.objects.all()[:5]     # lấy 5 phần tử đầu tiên

>>> Entry.objects.all()[5:10]   # lấy các phần tử từ vị trí 5 đến 10

## Biểu thức tìm kiếm

Các biểu thức tìm kiếm sẽ thực hiện câu truy vấn SQL WHERE trong CSDL. Biểu thức tìm kiếm là tham số cho các phương thức lọc dữ liệu filter(), exclude() và get().

Cú pháp lọc có dạng **<tên thuộc tính>\_\_<kiểu tìm kiếm>=<giá trị>** (lưu ý ở đây có 2 dấu gạch dưới).

Ví dụ:

>>> Entry.objects.filter(pub\_date\_\_lte='2016-01-01')

Trong biểu thức tìm kiếm thì **<tên thuộc tính>** bao giờ cũng là tên do chúng ta đặt khi định nghĩa model, ngoại trừ thuộc tính khóa ngoại phải thêm vào \_id ở cuối tên. Ví dụ:

>>> Entry.objects.filter(blog\_id=4)

- Một số cú pháp tìm kiếm thông dụng:

* exact: dữ liệu tìm được phải giống chính xác với giá trị cần tìm. Vd Entry.objects.get(headline\_\_exact='Cat bites dog') chỉ trả về những bản ghi có headline là *“Cat bites dog*“.
* iexact: dữ liệu tìm được chỉ cần giống kí tự với giá trị cần tìm, không phân biệt chữ HOA-thường. Vd Blog.objects.get(name\_\_iexact='beatles blog'), những bản ghi có headline là *“Beatles Blog”*, *“beatles blog”* hoặc *“BeAtlES blOG”*đều đ*ược nhận*.
* contains: dữ liệu chỉ cần chứa chuỗi giá trị là được. Vd Entry.objects.get(headlines\_\_contains='Lennon'). Giống LIKE tròn SQL Ngoài ra còn có icontains cũng có chức năng như contains nhưng không phân biệt chữ HOA-thường (giống ILIKE trong SQL).
* **startswith, endswith**:  dữ liệu có chuỗi bắt đầu hoặc kết thúc giống với giá trị.
* **istartswith, iendswith**:  có chức năng tương tự như startswith nhưng không phân biệt chữ hoa thường.
* in:  dữ liệu tồn tại trong khoảng. Ví dụ: Entry.objects.filter(id\_\_in=[1, 3, 4])
* gt: lớn hơn
* gte: lớn hơn hoặc bằng
* lt: nhỏ hơn
* lte: nhỏ hơn hoặc bằng

Ví dụ:

Entry.objects.filter(id\_\_gt=4) ⬄ SELECT ... WHERE id > 4;

* range: kiểm tra phạm vi ví dụ:

Entry.objects.filter(pub\_date\_\_range=(start\_date, end\_date))

SQL: SELECT ... WHERE pub\_date BETWEEN '2005-01-01' **and** '2005-03-31'

* date: kiểm tra các giá trị dạng ngày tháng năm

Entry.objects.filter(pub\_date\_\_date=datetime.date(2005, 1, 1))

Entry.objects.filter(pub\_date\_\_date\_\_gt=datetime.date(2005, 1, 1))

* year: kiểm tra các giá trị năm

ví dụ:

Entry.objects.filter(pub\_date\_\_year=2005)

Entry.objects.filter(pub\_date\_\_year\_\_gte=2005)

SQL tương đương

SELECT ... WHERE pub\_date BETWEEN '2005-01-01' AND '2005-12-31';

SELECT ... WHERE pub\_date >= '2005-01-01';

* iso\_year: kiểm tra các giá trị năm (Giá trị là một năm nguyên)
* month: kiểm tra các giá trị tháng

ví dụ:

Entry.objects.filter(pub\_date\_\_month=12)

Entry.objects.filter(pub\_date\_\_month\_\_gte=6)

Sql tương đương:

SELECT ... WHERE EXTRACT('month' FROM pub\_date) = '12';

SELECT ... WHERE EXTRACT('month' FROM pub\_date) >= '6';

* day: kiểm tra các giá trị ngày
* week: Giá trị tuần

ví dụ:

Entry.objects.filter(pub\_date\_\_week=52)

Entry.objects.filter(pub\_date\_\_week\_\_gte=32, pub\_date\_\_week\_\_lte=38)

* week\_day: Giá trị ngày trong tuần từ 1 (Sunday) đến 7 (Saturday).

Ví du:

Entry.objects.filter(pub\_date\_\_week\_day=2)

Entry.objects.filter(pub\_date\_\_week\_day\_\_gte=2)

* quarter: Giá trị một phần tư của năm
* time: Giá trị thời gian

ví dụ:

Entry.objects.filter(pub\_date\_\_time=datetime.time(14, 30))

Entry.objects.filter(pub\_date\_\_time\_\_range=(datetime.time(8), datetime.time(17)))

* hour: Giá trị thời gian (giờ) có 24 giờ từ 0-23
* minute: Giá trị thời gian (phút) có 60 phút từ 0-59
* second: Giá trị thời gian (giây) có 60 giây từ 0-59
* isnull: Trả về giá trị True hoặc False
* regex: Biểu thức chính quy

ví dụ:

Entry.objects.get(title\_\_regex=r'^(An?|The) +')

SQL tương đương:

SELECT ... WHERE title REGEXP BINARY '^(An?|The) +'; -- MySQL

SELECT ... WHERE REGEXP\_LIKE(title, '^(An?|The) +', 'c'); -- Oracle

SELECT ... WHERE title ~ '^(An?|The) +'; -- PostgreSQL

SELECT ... WHERE title REGEXP '^(An?|The) +'; -- SQLite

* regex: Biểu thức chính quy không phân biệt chữ hoa thường

## Tìm kiếm đa bảng

Các bảng trong CSDL có quan hệ với nhau nhờ vào các khóa ngoại, khi truy vấn dữ liệu, SQL cho phép bạn truy vấn các bản ghi có liên quan với nhau bằng cách nối các bảng lại bằng câu lệnh JOIN, Django cũng cho phép bạn làm điều đó, bạn chỉ cần đưa vào biểu thức tìm kiếm cú pháp **<tên khóa ngoại>\_\_<tên thuộc tính của bảng khác>=<giá trị>**. Ví dụ:

|  |  |
| --- | --- |
| 1 | >>> Entry.objects.filter(blog\_\_name='Beatles Blog') |

Câu lệnh trên sẽ tìm các bản ghi Entry có quan hệ với bảng Blog với name là “Beatles Blog”.

# **Django – Form có sẵn của Django**

## Tạo Form

Trong thư mục user-auth chúng ta tạo một file mới với tên là forms.py để lưu các lớp form.

Ví dụ:

from django import forms

class RegisterForm(forms.Form):

    username = forms.CharField(label='Username', max\_length=100)

    password = forms.CharField(widget=forms.PasswordInput)

    email = forms.EmailField(label='Email')

Các lớp dùng để tạo form được kế thừa từ lớp django.forms.Form.

Bên trong lớp này chúng ta cũng khai báo các trường là các đối tượng Field, nhưng các đối tượng Field này không giống như Field khi tạo model, Field ở đây là để tạo form HTML còn field bên model là để tạo bảng CSDL

Vì HTML có nhiều thẻ element có công dụng chung nhưng lại hiển thị dữ liệu khác nhau, chẳng hạn như một thẻ <input> có thể dùng để nhập tên, số điện thoại, password… do đó Django cung cấp cho mỗi đối tượng Field một đối tượng Widget để chúng ta có thể chỉ định loại text nào hiển thị cái gì.  label của CharField sẽ được dùng cho thuộc tính name trong thẻ <label>...

## Tạo Template

<form action="" method="post">

    {% csrf\_token %}

    {{ form }}

    <input type="submit" value="Submit" />

</form>

Khi chúng ta viết các lớp form, Django sẽ nhìn các trường mà chúng ta khai báo rồi sinh ra các thẻ <input> tương ứng, ngoại trừ cặp thẻ <form>...</form> và thẻ <input type="submit"/>

<form action="" method="post">

Thuộc tính action sẽ chuyển dữ liệu tới đường dẫn tương ứng, ở đây mình để trống tức là gửi lại cho chính trang đó. Phương thức POST sẽ mã hóa dữ liệu được gửi đi.

{{ form }}

Chúng ta có thể tham chiếu đến phần tử form trong list context mà chúng ta sẽ khai báo trong các hàm view

## Tạo View

from django.shortcuts import render

# Create your views here.

from django.http import HttpResponse

from .forms import RegisterForm

def register(request):

    if request.method == 'POST':

        response = HttpResponse()

        response.write("<h1>Thanks for registering</h1></br>")

        response.write("Your username: " + request.POST['username'] + "</br>")

        response.write("Your email: " + request.POST['email'] + "</br>")

        return response

    registerForm = RegisterForm()

    return render(request, 'user\_auth/register.html', {'form':registerForm})

# **Django – Upload file**

## Tạo form upload file

Chúng ta tạo một app mới với tên là file\_uploader:

C:\Project\mysite>python manage.py startapp file\_uploader

Chúng ta khai báo app cho project.

INSTALLED\_APPS = [

    #...

    'file\_uploader',

    #...

]

Kế tiếp chúng ta tạo file forms.py bên trong thư mục file\_uploader:

file\_uploader/forms.py

from django import forms

class UploadFileForm(forms.Form):

    title = forms.CharField(max\_length=50)

    file = forms.FileField()

Để hiển thị thẻ element chọn file thì chúng ta dùng lớp FileField().

Tiếp theo chúng ta tạo thư mục templates và file template:

file\_uploader/templates/fileUploaderTemplate.html

<form action="" method="POST" enctype="multipart/form-data">

    {% csrf\_token %}

    <table>

        {{ form.as\_table }}

    </table>

    <input type="submit" value="Submit" />

</form>

Để gửi file lên thì trong thẻ <form> chúng ta phải khai báo thuộc tính enctype="multipart/form-data".

Sau khi đã có template và form để hiển thị thì chúng ta tiến hành tạo view để kết nối chúng với nhau.

from django.db import models

# Create your models here.

from django.http import HttpResponse

from .forms import UploadFileForm

def fileUploaderView(request):

    if request.method == 'POST':

        form = UploadFileForm(request.POST, request.FILES)

        if form.is\_valid():

            upload(request.FILES['file'])

            return HttpResponse("<h2>File uploaded successful!</h2>")

        else:

            return HttpResponse("<h2>File uploaded not successful!</h2>")

    form = UploadFileForm()

    return render(request, 'fileUploaderTemplate.html', {'form':form})

def upload(f):

    file = open(f.name, 'wb+')

    for chunk in f.chunks():

        file.write(chunk)

Phương thức is\_valid() có trong lớp django.forms.Form sẽ kiểm tra tính hợp lệ của dữ liệu được gửi lên. Mục đích chính của lớp Form trong Django chính là hỗ trợ chúng ta kiểm tra sự đúng đắn của dữ liệu được gửi lên. Chẳng hạn như khi bạn xây dựng chức năng up ảnh avatar thì bạn chỉ cho phép user gửi các file ảnh lên thôi, nhưng kẻ tấn công có thể gửi các file ảnh “giả” lên mà trình duyệt không biết, do đó khi dữ liệu được gửi tới server, chúng ta nên kiểm tra trước các dữ liệu này trước khi lưu vào hệ thống.

Ở đây phương thức is\_valid() chỉ có chức năng đơn giản là kiểm tra xem ô text title có dữ liệu hay không vì mặc định các field của Form trong Django bắt buộc phải có dữ liệu mới được nhận.

upload(request.FILES['files'])

return HttpResponse("<h2>File uploaded successful!</h2>")

Nếu dữ liệu là hợp lệ thì chúng ta tiến hành lưu file vào thư mục gốc của server và trả lời thông báo gửi thành công về cho người dùng. Ở đây ta lấy thuộc tính “files” trong tham số FILES của đối tượng request

def upload(f):

    file = open(f.name, 'wb+')

    for chunk in f.chunks():

        file.write(chunk)

Hàm upload() sẽ thực hiện copy file vào thư mục gốc của server (thư mục có file manage.py).

Hàm sẽ tạo một file có tên và định dạng giống file trong request.FILES['file']) sau đó mở file rồi ghi dữ liệu chunk vào file vừa tạo theo kiểu dữ liệu của file đó.

Tóm tắt quy trình tạo file upload

- Tạo form của django trong đó có 1 trường thuộc kiểu

forms. fileField

- Tạo tempaltes để hiển thị với form action=”” method=”post”

- lấy đối tượng request(FILES[“files”]) ra để sử dụng

- Hàm upload với tham số là đối tượng request(FILES[“files”]) ban đầu sẽ tạo một file có kiểu và tên giống với thuộc tính name trong request(FILES[“files”].name)

files = open(f.name,'wb+')

Sau đó sẽ thực hiện vòng lặp với giá trị chunk và ghi vào file

for chunk in f.chunks():

        files.write(chunk)

# **Tạo form từ model**

## Kiểu dữ liệu

Mỗi lớp Field của Model sẽ có một lớp Field của Form tương ứng. Dưới đây là bảng tên các lớp Field tương ứng của Model và Form.

| **MODEL FIELD** | **FORM FIELD** |
| --- | --- |
| [AutoField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.AutoField) | Không có |
| [BigIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.BigIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) với min\_value là -9223372036854775808 và  max\_value là 9223372036854775807. |
| [BooleanField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.BooleanField) | [BooleanField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.BooleanField) |
| [CharField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.CharField) | [CharField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.CharField) với max\_length được thiết lập trong thuộc tính max\_length của model |
| [CommaSeparatedIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.CommaSeparatedIntegerField) | [CharField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.CharField) |
| [DateField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.DateField) | [DateField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.DateField) |
| [DateTimeField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.DateTimeField) | [DateTimeField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.DateTimeField) |
| [DecimalField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.DecimalField) | [DecimalField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.DecimalField) |
| [EmailField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.EmailField) | [EmailField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.EmailField) |
| [FileField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.FileField) | [FileField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.FileField) |
| [FilePathField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.FilePathField) | [FilePathField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.FilePathField) |
| [FloatField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.FloatField) | [FloatField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.FloatField) |
| [ForeignKey](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.ForeignKey) | [ModelChoiceField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.ModelChoiceField) |
| ImageField | [ImageField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.ImageField) |
| [IntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.IntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| IPAddressField | IPAddressField |
| [GenericIPAddressField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.GenericIPAddressField) | [GenericIPAddressField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.GenericIPAddressField) |
| [ManyToManyField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.ManyToManyField) | [ModelMultipleChoiceField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.ModelMultipleChoiceField) |
| [NullBooleanField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.NullBooleanField) | [NullBooleanField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.NullBooleanField) |
| [PositiveIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.PositiveIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| [PositiveSmallIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.PositiveSmallIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| [SlugField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.SlugField) | [SlugField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.SlugField) |
| [SmallIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.SmallIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| [TextField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.TextField) | [CharField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.CharField) với widget=forms.Textarea |
| [TimeField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.TimeField) | [TimeField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.TimeField) |
| [URLField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.URLField) | [URLField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.URLField) |

Lưu ý là các lớp ModelField hỗ trợ 3 kiểu khóa ngoại là **1-n**, **n-n** và **1-1** nhưng bên FormField chỉ có 2 kiểu duy nhất được hỗ trợ là ModelChoiceField tương ứng với ForeignKey và ModelMultipleChoiceField tương ứng với ManyToManyField. Cả 2 lớp trên đều nhận một tham số khi khởi tạo là một đối tượng QuerySet.

Mỗi lớp *Form* *Field* có chung một số thuộc tính như sau:

* required: nếu thuộc tính blank của Model là True thì required = False và ngược lại.
* label: thuộc tính này được gán tự động bằng thuộc tính verbose\_name của Model và kí tự đầu được viết hoa.
* help\_text: thuộc tính này được gán tự động bằng thuộc tính help\_text của model.
* required: nếu thuộc tính blank của Model là True thì required = False và ngược lại.
* label: thuộc tính này được gán tự động bằng thuộc tính verbose\_name của Model và kí tự đầu được viết hoa.
* help\_text: thuộc tính này được gán tự động bằng thuộc tính help\_text của model.

## Ví dụ

from django.db import models

from django.forms import ModelForm

TITLE\_CHOICES = (

    ('MR', 'Mr.'),

    ('MRS', 'Mrs.'),

    ('MS', 'Ms.'),

)

class Author(models.Model):

    name = models.CharField(max\_length=100)

    title = models.CharField(max\_length=3, choices=TITLE\_CHOICES)

    birth\_date = models.DateField(blank=True, null=True)

class Book(models.Model):

    name = models.CharField(max\_length=100)

    authors = models.ManyToManyField(Author)

class AuthorForm(ModelForm):

    class Meta:

        model = Author

        fields = ['name', 'title', 'birth\_date']

class BookForm(ModelForm):

    class Meta:

        model = Book

        fields = ['name', 'authors']

Chúng ta định nghĩa 2 lớp Model là Author và Book, sau đó định nghĩa 2 lớp Form có các Fieldđược tạo tự động từ 2 lớp model là AuthorForm và BookForm. Các lớp này được kế thừa từ lớp django.forms.ModelForm thay thì lớp django.forms.Form như trước.

class Meta:

    model = Author

    fields = ['name', 'title', 'birth\_date']

Chúng ta định nghĩa model được dùng trong lớp nội Meta, thuộc tính model sẽ tạo các field tương ứng từ lớp Model tương ứng, thuộc tính fields sẽ chọn các trường nào được dùng, nếu chúng ta không khai báo thuộc tính fields thì mặc định Django sẽ dùng tất cả các thuộc tính có trong lớp Model, ngoại trừ thuộc tính id nếu bạn không khai báo thuộc tính khóa chính. Ngoài ra lớp Meta còn có thuộc tính exclude, thuộc tính này trái ngược với thuộc tính fields, tức là thuộc tính này sẽ quy định các trường nào không được phép sử dụng.

## Lưu dữ liệu

Lớp ModelForm có một phương thức tên là save(), phương thức này tạo mới hoặc lưu một đối tượng Model vào cơ sở dữ liệu giống như phương thức save() bên các lớp *Model* vậy.

Khi khởi tạo các đối tượng ModelForm thì chúng ta có thể truyền các đối tượng model có sẵn vào hàm khởi tạo, nếu chúng ta chỉ truyền vào không thôi thì Django sẽ tạo mới một đối tượng trên CSDL, nếu chúng ta truyền vào và ghi rõ truyền với tham số là instance thì Django sẽ cập nhật dữ liệu trong đối tượng đó. Ví dụ:

>>> from myapp.models import Article

>>> from myapp.forms import ArticleForm

>>> f = ArticleForm(request.POST)

>>> new\_article = f.save()

Đoạn code trên tạo đối tượng mới từ đối tượng POST được gửi lên và lưu vào CSDL.

>>> a = Article.objects.get(pk=1)

>>> f = ArticleForm(request.POST, instance=a)

>>> f.save()

Lưu ý là nếu chúng ta không kiểm tra sự đúng đắn của dữ liệu thì khi gọi phương thức save(),Django cũng sẽ tự động làm việc đó và sẽ giải phóng lỗi exception ValueError nếu dữ liệu có lỗi, trong [bài trước](http://phocode.com/django/django-upload-file/) chúng ta đã dùng một phương thức để kiểm tra dữ liệu là is\_valid(), chúng ta sẽ tìm hiểu thêm về cách kiểm tra dữ liệu trong các bài sau.

# **Kiểm tra dữ liệu gửi lên form**

## Validator

# **Phân trang**

Django cung cấp lớp django.core.paginator hỗ trợ phân trang rất tốt.

Views:

from django.shortcuts import render

# Create your views here.

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

from .models import Customer

def listing(request):

    customer\_list = Customer.objects.all()

    paginator = Paginator(customer\_list, 5)

    pageNumber = request.GET.get('page')

    try:

        customers = paginator.page(pageNumber)

    except PageNotAnInteger:

        customers = paginator.page(1)

    except EmptyPage:

        customers = paginator.page(paginator.num\_pages)

    return render(request, 'list.html', {'customers':customers})

Chúng ta sử dụng lớp Pagination để thực hiện phân trang.

paginator = Paginator(customer\_list, 5)

Hàm khởi tạo Paginator() nhận vào 2 tham số, tham số đầu tiên là một đối tượng QuerySet,tham số thứ 2 là số tượng item trên mỗi “trang”. Trong ví dụ trên chúng ta đưa đối tượng customer\_list vào với số lượng 5 item mỗi trang.

pageNumber = request.GET.get('page')

URL của chúng ta có thêm tham số page là số thứ tự của trang muốn xem.

try:

    customers = paginator.page(pageNumber)

except PageNotAnInteger:

    customers = paginator.page(1)

except EmptyPage:

    customers = paginator.page(paginator.num\_pages)

Nếu tham số page không hợp lệ, chẳng hạn như page=abc thì Paginator sẽ giải phóng lỗi PageNotAnInterger, trong trường hợp này chúng ta trả về trang đầu tiên với phương thức Paginator.page(), hoặc nếu page nằm ngoài phạm vi trang cho phép, chẳng hạn như chúng ta chỉ có 4 trang nhưng tham số page=1000 thì Paginator sẽ giải phóng lỗi EmptyPage, ở đây chúng ta xử lý bằng cách trả về trang cuối cùng bằng thuộc tính num\_pages.