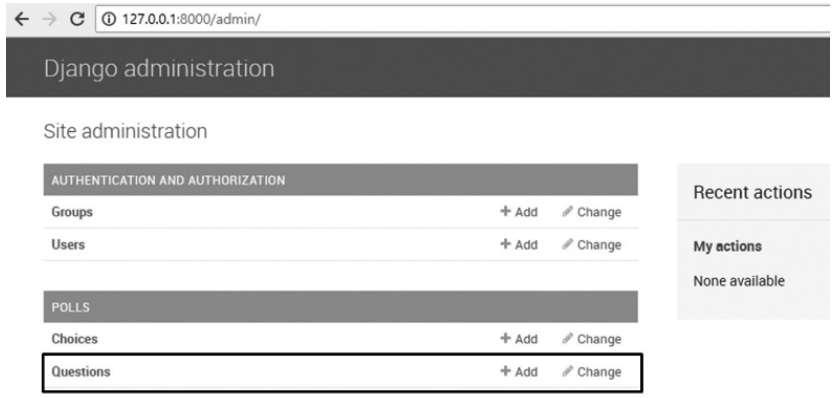
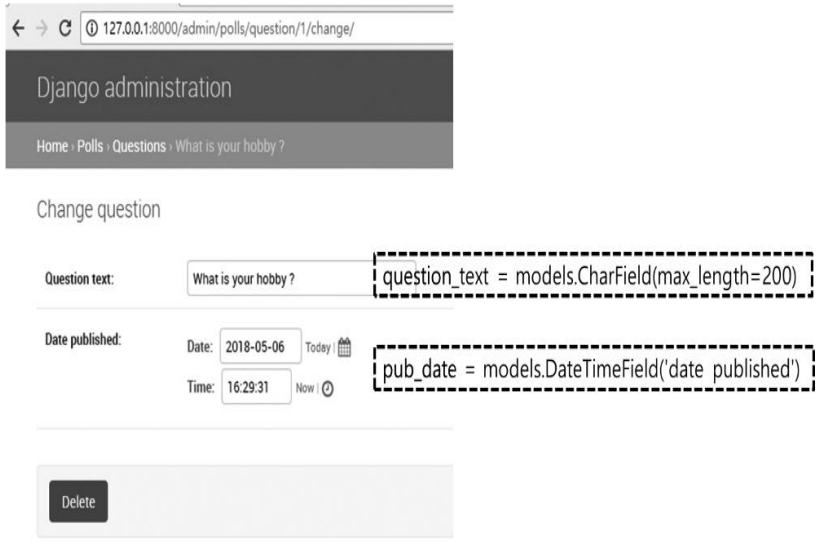
Django의 핵심 기능

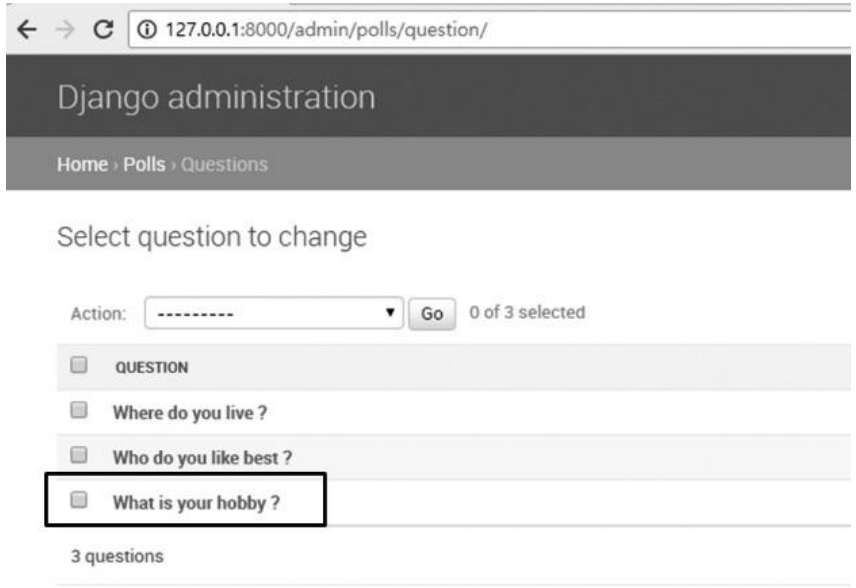
장고의 기능 중에서 실제 프로젝트 개발을 위해 꼭 알아야 되는 6개 기능

Admin 사이트 꾸미기

데이터 입력 및 수정







Admin 사이트 꾸미기

데이터 입력 및 수정

예제 4-1 모델 클래스와 Admin UI 간 위젯 매핑

```
# 위의 내용 동일
class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
# 아래 내용 동일
```

- Save : 변경사항을 저장하고 현 객체의 리스트를 보여주는 페이지로 돌아감
- Save and continue editing : 변경사항을 저장하고 현재의 페이지를 다시 보여줌
- Save and add another : 변경사항을 저장하고 새로운 레코드를 추가할 수 있는 폼 페이지를 보여줌
- Delete : 삭제 확인 페이지를 보여줌

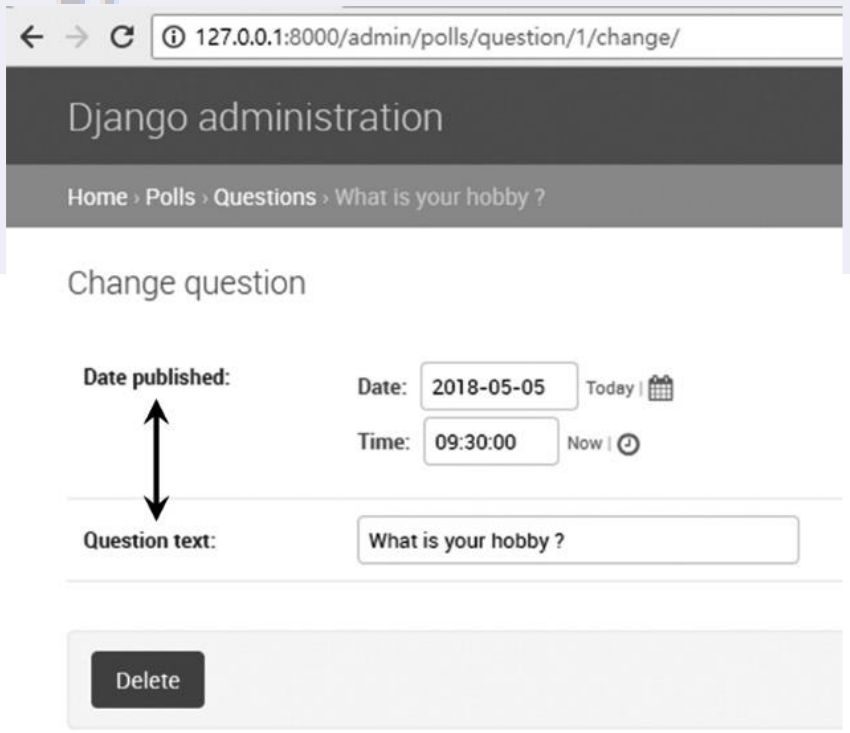
Admin 사이트 꾸미기

필드 순서 변경하기

예제 4-2 QuestionAdmin 클래스 정의 - admin.py

```
# 위의 내용 동일
class QuestionAdmin(admin.ModelAdmin):
    fields = ['pub_date', 'question_text']   # 필드 순서 변경

admin.site.register(Question, QuestionAdmin)
admin.site.register(Choice)
```

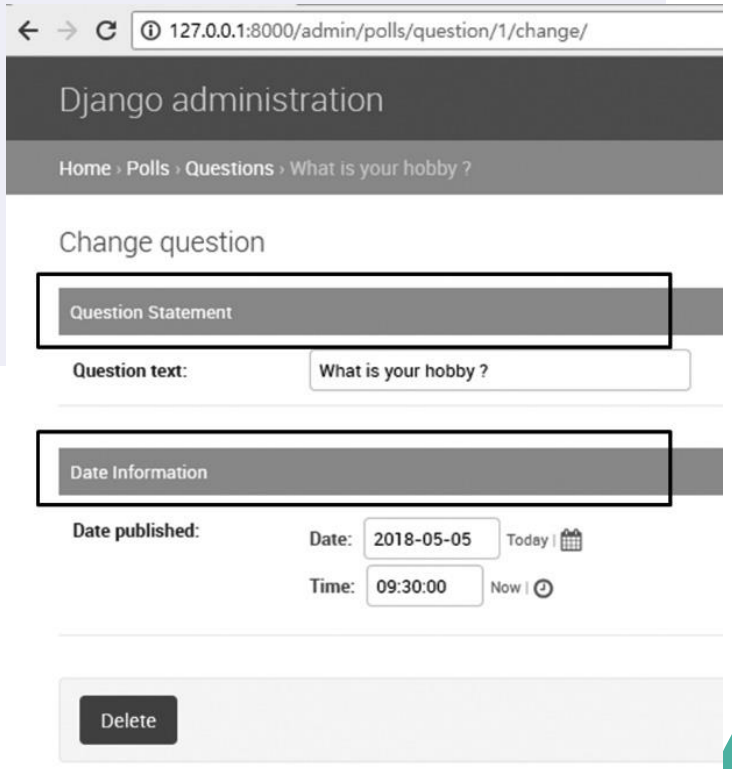


Admin 사이트 꾸미기

각 필드를 분리해서 보여주기

예제 4-3 필드를 분리해서 보여주기 - polls/admin.py 코딩

```
# 위의 내용 동일
class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        ('Question Statement', {'fields': ['question_text']}),
        ('Date Information', {'fields': ['pub_date']}),
    ]
# 아래 내용 동일
```



Admin 사이트 꾸미기

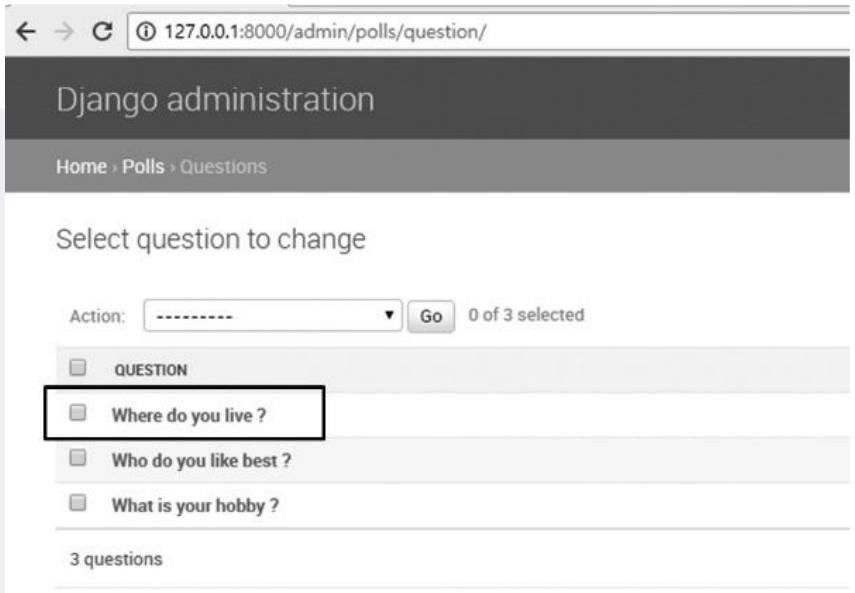
Question 및 Choice를 한 화면에서 변경하기

예제 4-5 Question과 Choice 같이 보기 – polls/admin.py 코딩

```
# 위의 내용 동일
class ChoiceInline(admin.StackedInline):
    model = Choice
    extra = 2

class QuestionAdmin(admin.ModelAdmin):
    fieldsets = [
        (None,               {'fields': ['question_text']}),
        ('Date information', {'fields': ['pub_date'], 'classes': ['collapse']}),
    ]
    inlines = [ChoiceInline]      # Choice 모델 클래스 같이 보기

admin.site.register(Question, QuestionAdmin)
admin.site.register(Choice)
```

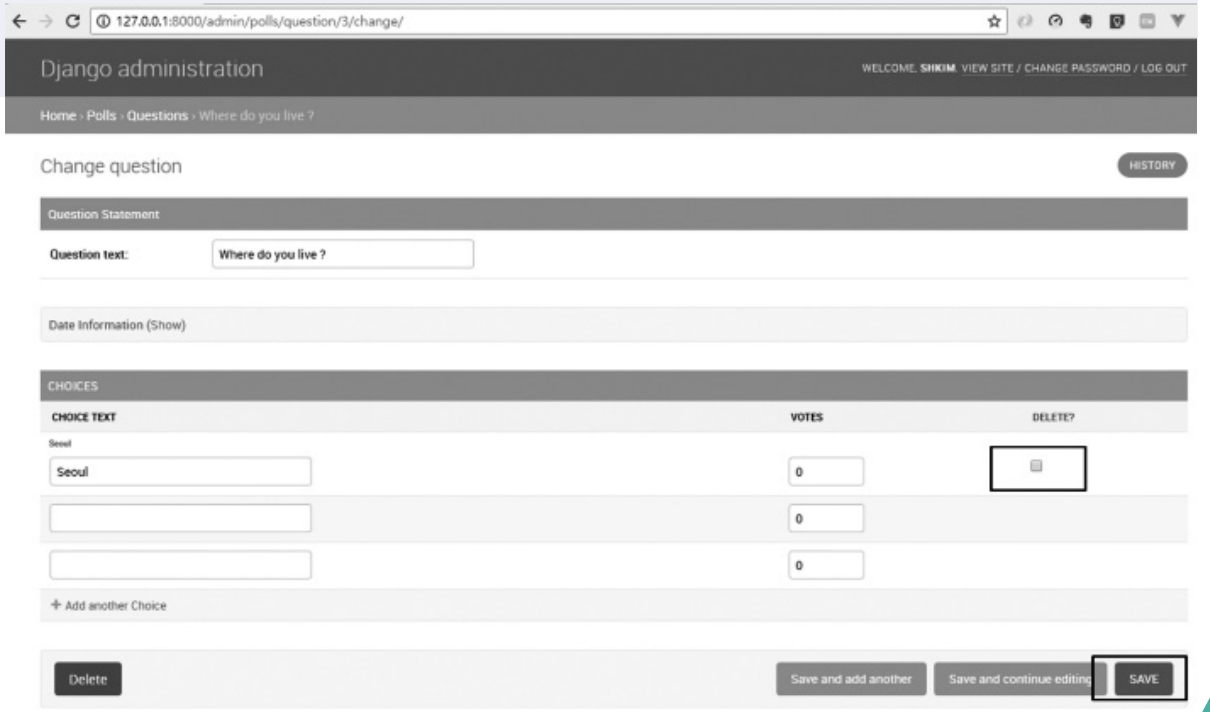


Admin 사이트 꾸미기

테이블 형식으로 보여주기

예제 4-6 테이블 형식으로 보여주기 - polls/admin.py 코딩

```
# 위의 내용 동일
class ChoiceInline(admin.TabularInline):
# 아래 내용 동일
```

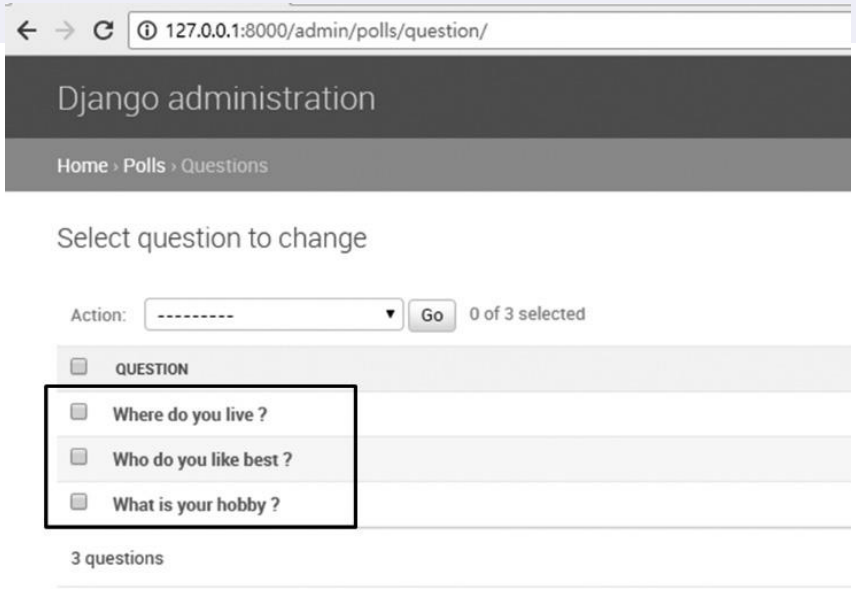


Admin 사이트 꾸미기

레코드 리스트 컬럼 지정하기

예제 4-7 레코드 리스트 컬럼 지정하기 - polls/admin.py 코딩

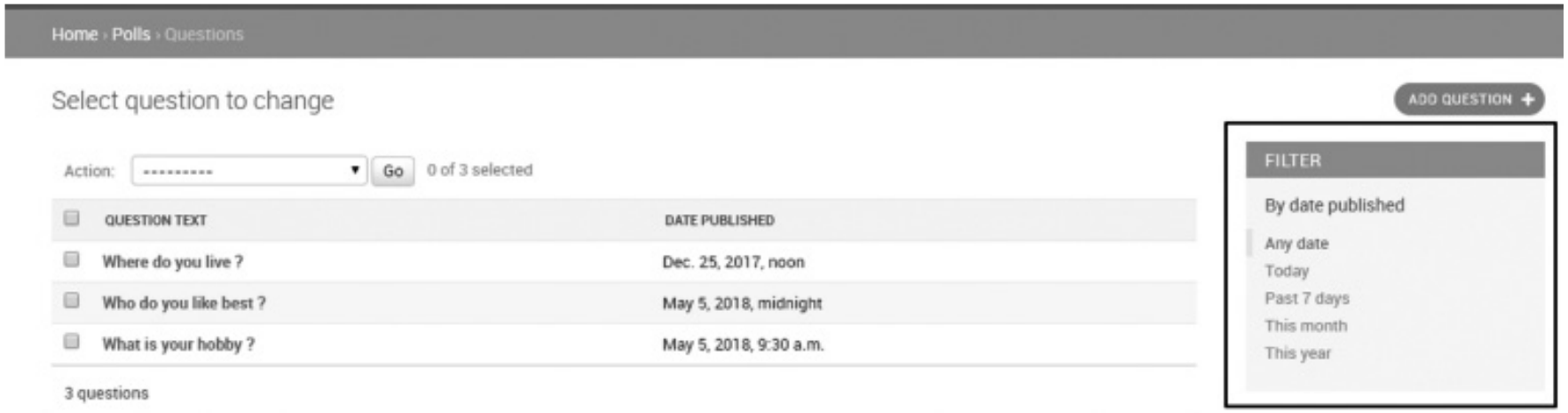
```
# 위의 내용 동일
inlines = [ChoiceInline] # Choice 모델 클래스 같이 보기
list_display = ('question_text', 'pub_date')          # 레코드 리스트 컬럼 지정
# 아래 내용 동일
```



Admin 사이트 꾸미기

list_filter 필터

```
# 위의 내용 동일
inlines = [ChoiceInline] # Choice 모델 클래스 같이 보기
list_display = ('question_text', 'pub_date')          # 레코드 리스트 컬럼 항목 지정
list_filter = ['pub_date']          # 필터 사이드 바 추가
# 아래 내용 동일
```

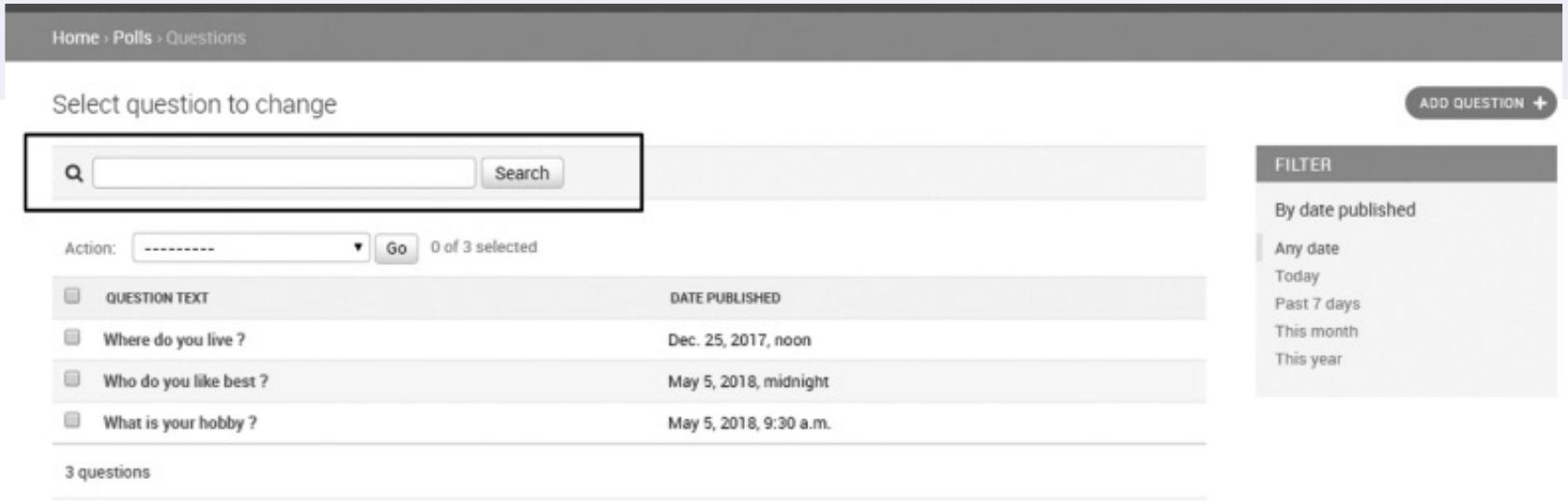


Admin 사이트 꾸미기

search_fields

예제 4-9 search_fields 검색 박스 추가 – polls/admin.py 코딩

```
# 위의 내용 동일
inlines = [ChoiceInline] # Choice 모델 클래스 같이 보기
list_display = ('question_text', 'pub_date')          # 레코드 리스트 컬럼 항목 지정
list_filter = ['pub_date']              # 필터 사이드바 추가
search_fields = ['question_text']  # 검색 박스 추가
# 아래 내용 동일
```



Admin 사이트 꾸미기

polls/admin.py 변경 내역 정리

예제 4-10 polls/admin.py 최종 모습 — 소스제공: ch4₩polls₩admin.py

```
from django.contrib import admin

# Register your models here.
from polls.models import Question, Choice

#class ChoiceInline(admin.StackedInline):  ①
class ChoiceInline(admin.TabularInline):  ②
    model = Choice
    extra = 2

class QuestionAdmin(admin.ModelAdmin):
    #fields = ['pub_date', 'question_text']    # 필드 순서 변경  ③
    fieldsets = [  ④
        ('Question Statement', {'fields': ['question_text']}),
        #('Date Information', {'fields': ['pub_date']}),
        ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),  ⑤
    ]
    inlines = [ChoiceInline]    # Choice 모델 클래스 같이 보기
    list_display = ('question_text', 'pub_date')    # 레코드 리스트 컬럼 항목 지정  ⑥
    list_filter = ['pub_date']    # 필터 사이드 바 추가  ⑦
    search_fields = ['question_text']    # 검색 박스 추가  ⑧

admin.site.register(Question, QuestionAdmin)
admin.site.register(Choice)
```

Admin 사이트 꾸미기

Admin 사이트 템플릿 수정

예제 4-11 base_site.html 복사 및 settings.py 수정

```
C:₩Users₩shkim>cd C:₩RedBook₩ch4
C:₩RedBook₩ch4>mkdir templates
C:₩RedBook₩ch4>mkdir templates₩admin
C:₩RedBook₩ch4>copy ₩Develop₩gm₩Python36₩Lib₩site-packages₩django₩contrib₩
admin₩templates₩admin₩base_site.html templates₩admin₩
C:₩RedBook₩ch4>notepad mysite₩settings.py

# 위의 내용 동일
TEMPLATES = [
    {
        . . .
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        . . .
    }
]
# 아래 내용 동일
```

장고 파이썬 쉘로 데이터 조작하기

```
C:\Users\shkim>cd C:\RedBook\ch4
C:\RedBook\ch4>python manage.py shell
```

- 앞의 장고 파이썬 쉘 명령도 역시 파이썬 쉘을 기동하는 것입니다. 다만 파이썬 쉘과 비교했을 때 장고 파이썬 쉘의 다른 점은 manage.py 모듈에서 정의한 **DJANGO_SETTINGS_MODULE** 속성을 이용하여 미리 mysite/settings.py 모듈을 임포트한다는 것

- 테이블은 레코드의 모음이라고 할 수 있는데, 장고의 ORM은 이러한 테이블의 구조를 클래스로 표현.
- 클래스의 객체를 생성하는 것은 테이블의 레코드를 생성하는 것을 의미.

장고 파이썬 쉘로 데이터 조작하기

Create - 데이터 생성/입력

```
>>> from polls.models import Question, Choice
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())
>>> q.save()
```

- 데이터 입력, 즉 테이블에 레코드를 생성하기 위해서는 필드값을 지정하여 객체를 생성한 후에 save() 메소드를 호출하면 됨.
- save() 명령이 실행되기 전에는 메모리에서만 변경된 것이므로, 변경사항을 데이터베이스에 반영하기 위해서는 save() 명령을 실행

장고 파이썬 쉘로 데이터 조작하기

Read - 데이터 조회

- 데이터베이스로부터 데이터를 조회하기 위해서는 **QuerySet** 객체를 사용.
- QuerySet은 데이터베이스 테이블로부터 꺼내 온 객체들의 콜렉션.
- QuerySet은 필터를 가질 수 있으며, 필터를 사용하여 QuerySet 내의 항목 중에서 조건에 맞는 레코드만 다시 추출.
- SQL 용어로 QuerySet은 **SELECT** 문장에 해당하며, 필터는 **WHERE** 절에 해당

```
>>> Question.objects.all()        # 'Question테이블.레코드들.모두'라고 해석하면 됨
```

```
>>> Question.objects.filter(
...     question_text__startswith='What'
... ).exclude(
...     pub_date__gte=datetime.date.today()
... ).filter(
...     pub_date__gte=datetime(2005, 1, 30)
... )
```

```
>>> one_entry = Question.objects.get(pk=1)
```

장고 파이썬 쉘로 데이터 조작하기

Update - 데이터 수정

이미 존재하는 객체에 대한 필드값을 수정하는 경우에도 필드 속성값을 수정한 후 save() 메소드를 호출 하면됨

```
>>> q.question_text = 'What is your favorite hobby ?'
>>> q.save()
```

```
>>> Question.objects.filter(pub_date__year=2007).update(question_text='Everything is
the same')
```

장고 파이썬 쉘로 데이터 조작하기

Delete - 데이터 삭제

객체를 삭제하기 위해서는 delete() 메소드를 사용.
아래 문장은 pub_date 필드의 연도가 2005년인 모든 객체를 삭제하는 명령

```
>>> Question.objects.filter(pub_date__year=2005).delete()
```

이 문장이 문법적으로는 맞게 보이지 만, 장고에서 허용하지 않는 이유는 **all** ()이란 문구를 사용하게 함으로써 의도치 않게 모든 레코드 를 삭제하는 사용자 실수를 줄이기 위함

```
>>> Question.objects.delete()
```

장고 파이썬 쉘로 데이터 조작하기

polls 애플리케이션의 데이터 실습

예제 4-13 장고 파이썬 쉘 – 실습 1

```
C:₩Users₩shkim>cd C:₩RedBook₩ch4
C:₩RedBook₩ch4>python manage.py shell

# 우리가 정의한 모델을 사용하기 위해 임포트합니다.
>>> from polls.models import Question, Choice

# 현재 각 테이블에 들어있는 레코드를 확인합니다.
# 3장의 예제를 실습하면서 입력한 Question 레코드 3개가 보입니다.
>>> Question.objects.all()
[<Question: What is your hobby ?>, <Question: What do you like best ?>, <Question:
Where do you live ?>]

# Admin 사이트를 실행하면서 4.1.6 절에서 입력한 Choice 레코드 6개가 보입니다.
>>> Choice.objects.all()
[<Choice: Reading>, <Choice: Soccer>, <Choice: Climbing>, <Choice: Seoul>, <Choice:
Daejeon>, <Choice: Jeju>]

# 추가로 레코드 하나를 생성하겠습니다.
# 날짜를 입력하기 위해 timezone 모듈을 임포트합니다.
# settings.py 모듈에 TZ 세팅이 바르게 되어 있어야 합니다.
# datetime.datetime.now()보다는 timezone.now() 사용을 추천합니다.
>>> from django.utils import timezone
>>> q = Question(question_text="What's up?", pub_date=timezone.now())

# 데이터베이스에 저장을 위해 save() 함수를 호출합니다.
>>> q.save()

# id 속성이 자동으로 생성된 것을 확인합니다.
>>> q.id
4

# 속성에 접근할 때는 파이썬 문법 그대로 '.'을 사용합니다.
>>> q.question_text
"What's up ?"
>>> q.pub_date
datetime.datetime(2018, 5, 10, 13, 0, 0, 775217, tzinfo=<UTC>)

# 기존의 속성값을 변경하고 데이터베이스에 저장합니다.
>>> q.question_text = "What's new ?"
>>> q.save()

# 테이블의 모든 레코드를 조회합니다.
>>> Question.objects.all()
[<Question: What is your hobby ?>, <Question: What do you like best ?>, <Question:
Where do you live ?>, <Question: What's new ?>]
# 만일 레코드 제목이 [<Question: Question object>]처럼 나온다면,
# models.py 모듈에 __str__() 메소드를 확인하기 바랍니다.

# 파이썬 쉘을 빠져 나오기 위해서는, exit() 또는 Ctrl-Z(리눅스에서는 Ctrl-D)를 입력합니다
>>> exit()
```

템플릿 시스템

- MVT 방식에서 사용자에게 보여주는 화면, 즉 UIUser Interface를 담당하고 있는 기능이 템플릿 시스템.
- 템플릿 코드를 작성 시에 HTML 코드와 장고의 템플릿 코드가 섞이지만, 중요한 점은 템플릿에서는 로직을 표현하는 것이 아니라 사용자에게 어떻게 보여줄지에 대한 룩앤필을 표현한 다는 것임

템플릿 시스템

템플릿 변수

```
{{ variable }}
```

- 템플릿 시스템은 변수를 평가해서 변수값으로 출력.
- 변수명은 일반 프로그래밍의 변수명처럼 문자, 숫자, 밑줄(_)을 사용하여 이름을 정의

템플릿 시스템

템플릿 필터

- 아래의 예시처럼 파이프(¦) 문자를 사용.
- name 변수값의 모든 문자를 소문자로 바꿔주는 필터

```
{{ name|lower }}
```

- 필터를 체인으로 연결할 수도 있음

```
{{ text|escape|linebreaks }}
```

- 다음은 bio 변수값 중에서 앞에 30개의 단어만 보여 주고,
- 줄 바꿈 문자는 모두 없애줌.

```
{{ bio|truncatewords:30 }}
```

템플릿 시스템

템플릿 필터

- 필터의 인자에 빈칸이 있는 경우는 따옴표로 묶어줌

```
{{ list|join:" // " }}
```

- value 변수값이 False이거나 없는 경우, “nothing”으로 보여줌

```
{{ value|default:"nothing" }}
```

- value 변수값의 길이를 반환

```
{{ value|length }}
```

- value 변수값에서 HTML 태그를 모두 제거

```
{{ value|striptags }}
```

템플릿 시스템

템플릿 태그

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```

for 태그에 사용되는 변수들

변수명	설명
forloop.counter	현재까지 루프를 실행한 루프 카운트(1부터 카운트함)
forloop.counter0	현재까지 루프를 실행한 루프 카운트(0부터 카운트함)
forloop.revcounter	루프 끝에서 현재가 몇 번째인지 카운트한 숫자(1부터 카운트함)
forloop.revcounter0	루프 끝에서 현재가 몇 번째인지 카운트한 숫자(0부터 카운트함)
forloop.first	루프에서 첫 번째 실행이면 True 값을 가짐
forloop.last	루프에서 마지막 실행이면 True 값을 가짐
forloop.parentloop	중첩된 루프에서 현재의 루프 바로 상위의 루프를 의미함

템플릿 시스템

템플릿 태그

{% if %} 태그

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
    No athletes.
{% endif %}
```

| {% csrf_token %} 태그 |

```
<form action="." method="post">{% csrf_token %}
```

| {% url %} 태그 |

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

템플릿 시스템

템플릿 태그

| {% with %} 태그 |

```
{% with total=business.employees.count %}
    {{ total }} people works at business
{% endwith %}
```

| {% load %} 태그 |

```
{% load somelibrary package.otherlibrary %}
```

템플릿 시스템

템플릿 주석

첫번째는 한 줄 주석문으로, 아래처럼 {# #} 형식을 따름. 한 문장의 전부 또는 일부를 주석 처 리하는 방법

```
{# greeting #}hello
```

{# #} 주석문 내에 템플릿 코드가 들어있어도 정상적으로 주석 처리 됨

```
{# {% if foo %}bar{% else %} #}
```

여러 줄의 주석문으로, 아래처럼 {% comment %} 태그를 사용합니다

```
{% comment "Optional note" %}
<p>Commented out text here</p>
{% endcomment %}
```

템플릿 시스템

HTML 이스케이프

- 사용자가 입력한 데이터를 그대로 렌더링하는 것은 위험
- 장고는 이를 방지하기 위해서 자동 이스케이프 기능을 제공.
- 즉 장고는 디폴트 로 HTML에 사용되는 예약 문자들을 아래처럼 예약 의미를 제거한 문자로 변경해주는 기능을 제공

```
< (less than) 문자는 &lt; 로 변경함
> (greater than) 문자는 &gt; 로 변경함
' (single quote) 문자는 &#39; 로 변경함
" (double quote) 문자는 &quot; 로 변경함
& (ampersand) 문자는 &amp; 로 변경함
```

템플릿 시스템

템플릿 상속

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <link rel="stylesheet" href="{% static 'admin/css/base.css' %}" />
    <title>{% block title %}My Amazing Site{% endblock %}</title>
</head>

<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Project_Home</a></li>
            <li><a href="/admin/">Admin</a></li>
        </ul>
        {% endblock %}
        <br>
    </div>

    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

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

{% block content %}
    <h2>Books Management System</h2>
    <ul>
        {% for modelname in model_list %}
{% with "books:"|add:modelname|lower|add:"_list" as urlvar %}
            <li><a href="{% url urlvar %}">{{ modelname }}</a></li>
{% endwith %}
        {% endfor %}
    </ul>
{% endblock content %}
```

폼처리하기

HTML에서의 폼

- 우리는 웹 사이트를 개발할 때 사용자로부터 입력을 받기 위해서 폼을 사용.
- HTML로 표현 하면 폼은 <form>...</form> 사이에 있는 엘리먼트들의 집합.
- 웹 사이트 사용자는 폼을 통 해 텍스트를 입력할 수도 있고, 항목을 선택할 수도 있음
- form 태그에 action 속성으로 전송 대상을 지정
- form 태그에 method 속성으로 데이터 전송방법을 지정
  - post : 메시지 본문에 데이터를 기록하고 전송
  - get : 주소 뒤에 ?이름=값&이름=값2의 형식으로 데이터 전송

폼처리하기

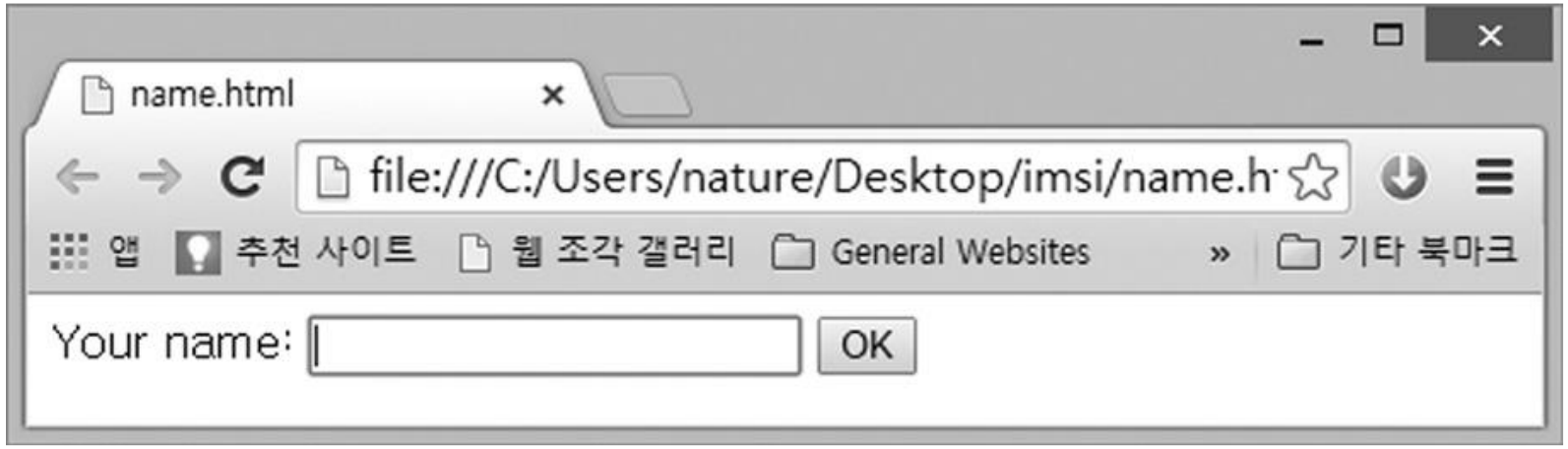
장고의 폼 기능

- 여러 가지 타입의 많은 위젯이 폼 화면 출력용으로 준비되어야 하고, HTML로 렌더링되며, 적절한 인터페이스를 사용하여 입력 및 수정되고, 서버로 보내져서 데이터가 유효한지 검증을 거친 후에 적절한 처리를 위해 저장되거나 전달.
- 장고는 이러한 폼 기능들을 단순화하고 자동화해서 개발자가 직접 코딩하는 것보다 훨씬 안전하게 처리.
- 장고는 폼 처리를 위하여 다음의 3가지 기능을 제공
  - 폼 생성에 필요한 데이터를 폼 클래스로 구조화하기
  - 폼 클래스의 데이터를 렌더링하여 HTML 폼 만들기
  - 사용자로부터 제출된 폼과 데이터를 수신하고 처리하기
- 폼도 결국은 템플릿의 일부이므로 템플릿 코드에 포함되어서 렌더링 절차
  - 렌더링할 객체를 뷰로 가져오기(예를 들어, 데이터베이스로부터 객체를 추출하기)
  - 그 객체를 템플릿 시스템으로 넘겨주기
  - 템플릿 문법을 처리해서 HTML 마크업 언어로 변환하기

폼처리하기

폼 클래스로 폼 생성

```
<form action="/your-name/" method="post">
    <label for="your_name">Your name: </label>
    <input id="your_name" type="text" name="your_name" value="{{ current_name }}">
    <input type="submit" value="OK">
```



예제 4-18 폼 클래스 정의

```
from django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label='Your name', max_length=100)
```

폼처리하기

뷰에서 폼 클래스 처리

예제 4-21 뷰에서 폼 클래스를 처리하는 방식

```
from django.shortcuts import render
from django.http import HttpResponseRedirect

def get_name(request):                                                    ①
    # POST 방식이면, 데이터가 담긴 제출된 폼으로 간주합니다.
    if request.method == 'POST':                                          ②, ④
        # request에 담긴 데이터로, 클래스 폼을 생성합니다.
        form = NameForm(request.POST)                                     ⑤
        # 폼에 담긴 데이터가 유효한지 체크합니다.
        if form.is_valid():                                               ⑥
            # 폼 데이터가 유효하면, 데이터는 cleaned_data로 복사됩니다.
            new_name = form.cleaned_data['name']
            # 로직에 따라 추가적인 처리를 합니다.                          ⑧

            # 새로운 URL로 리다이렉션시킵니다.
            return HttpResponseRedirect('/thanks/')

    # POST 방식이 아니면(GET 요청임),
    # 빈 폼을 사용자에게 보여줍니다.
    else:                                                                 ②
        form = NameForm()                                                 ③

    return render(request, 'name.html', {'form': form})                   ⑦, ⑨
```

폼처리하기

폼 클래스를 템플릿으로 변환

- {{ form.as_table }} : <tr> 태그로 감싸서 테이블 셀로 렌더링됩니다. {{form}}과 동일함
- {{ form.as_p }} : <p> 태그로 감싸도록 렌더링됩니다.
- {{ form.as_ul }} : <li> 태그로 감싸도록 렌더링됩니다.

예제 4-22 ContactForm 폼 클래스 정의

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

예제 4-23 ContactForm 폼 클래스의 {{ form.as_p }} 렌더링 결과

```
<p><label for="id_subject">Subject:</label>
    <input id="id_subject" type="text" name="subject" maxlength="100" /></p>
<p><label for="id_message">Message:</label>
    <input type="text" name="message" id="id_message" /></p>
<p><label for="id_sender">Sender:</label>
    <input type="email" name="sender" id="id_sender" /></p>
<p><label for="id_cc_myself">Cc myself:</label>
    <input type="checkbox" name="cc_myself" id="id_cc_myself" /></p>
```

클래스형 뷰

클래스형 뷰의 시작점

예제 4-24 클래스형 뷰의 진입 메소드 as_view()

```
# urls.py
from django.urls import path
from myapp.views import MyView

urlpatterns = [
    path('about/', MyView.as_view()),
]
```

클래스형 뷰

클래스형 뷰의 장점 – 효율적인 메소드 구분

- GET, POST 등의 HTTP 메소드에 따른 처리 기능을 코딩할 때, IF 함수를 사용하지 않고 메소드명으로 구분할 수 있으므로 코드의 구조가 깔끔해짐
- 다중 상속과 같은 객체 지향 기술이 가능하므로, 클래스형 제네릭 뷰 및 믹스인 클래스 등을 사용할 수 있고, 이는 코드의 재사용성이나 개발 생산성을 획기적으로 높여줌.

예제 4-26 함수형 뷰로 HTTP GET 메소드 코딩

```
from django.http import HttpResponse

def my_view(request):
    if request.method == 'GET':
        # 뷰 로직 작성
        return HttpResponse('result')
```

클래스형 뷰

클래스형 뷰의 장점 – 상속 기능 가능

예제 4-29 클래스형 뷰 작성 – TemplateView 상속

```
# some_app/urls.py
from django.urls import path
from some_app.views import AboutView

urlpatterns = [
    path('about/', AboutView.as_view()),
]
```

```
# some_app/views.py
from django.views.generic import TemplateView

class AboutView(TemplateView):
    template_name = "about.html"
```

예제 4-30 클래스형 뷰 작성 – URLconf에 TemplateView 지정

```
# some_app/urls.py
from django.urls import path
from django.views.generic import TemplateView

urlpatterns = [
    path('about/', TemplateView.as_view(template_name="about.html")),
]
```

클래스형 뷰

클래스형 제네릭 뷰

장고의 제네릭 뷰 리스트(일부)

Base View	View	가장 기본이 되는 최상위 제네릭 뷰입니다. 다른 모든 제네릭 뷰는 View의 하위 클래스입니다.
	TemplateView	템플릿이 주어지면 해당 템플릿을 렌더링해줍니다.
	RedirectView	URL이 주어지면 해당 URL로 리다이렉트시켜줍니다.
Generic Display View	ListView	조건에 맞는 여러 개의 객체를 보여줍니다.
	DetailView	객체 하나에 대한 상세한 정보를 보여줍니다.
Generic Edit View	FormView	폼이 주어지면 해당 폼을 보여줍니다.
	CreateView	객체를 생성하는 폼을 보여줍니다.
	UpdateView	기존 객체를 수정하는 폼을 보여줍니다.
	DeleteView	기존 객체를 삭제하는 폼을 보여줍니다.
Generic Date View	ArchiveIndexView	조건에 맞는 여러 개의 객체 및 그 객체들에 대한 날짜 정보를 보여줍니다.
	YearArchiveView	연도가 주어지면 그 연도에 해당하는 객체들을 보여줍니다.
	MonthArchiveView	연, 월이 주어지면 그에 해당하는 객체들을 보여줍니다.
	WeekArchiveView	연도와 주차(week)가 주어지면 그에 해당하는 객체들을 보여줍니다.
	DayArchiveView	연, 월, 일이 주어지면 그 날짜에 해당하는 객체들을 보여줍니다.
	TodayArchiveView	오늘 날짜에 해당하는 객체들을 보여줍니다.
	DateDetailView	연, 월, 일, 기본키(또는 슬러그)가 주어지면 그에 해당하는 특정 객체 하나에 대한 상세한 정보를 보여줍니다.

클래스형 뷰

클래스형 뷰에서 폼 처리

```
from django.http import HttpResponseRedirect
from django.shortcuts import render

from .forms import MyForm

def myview(request):
    if request.method == "POST":
        form = MyForm(request.POST)
        if form.is_valid():
            # cleaned_data로 관련 로직 처리
            return HttpResponseRedirect('/success/')
    else:
        form = MyForm(initial={'key': 'value'})

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

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from django.views.generic import View

from .forms import MyForm

class MyFormView(View):
    form_class = MyForm
    initial = {'key': 'value'}
    template_name = 'form_template.html'

    def get(self, request, *args, **kwargs):
        form = self.form_class(initial=self.initial)
        return render(request, self.template_name, {'form': form})

    def post(self, request, *args, **kwargs):
        form = self.form_class(request.POST)
        if form.is_valid():
            # cleaned_data로 관련 로직 처리
            return HttpResponseRedirect('/success/')

        return render(request, self.template_name, {'form': form})
```

최초의 GET

유효한 데이터를 가진 POST

유효하지 않은 데이터를 가진 POST