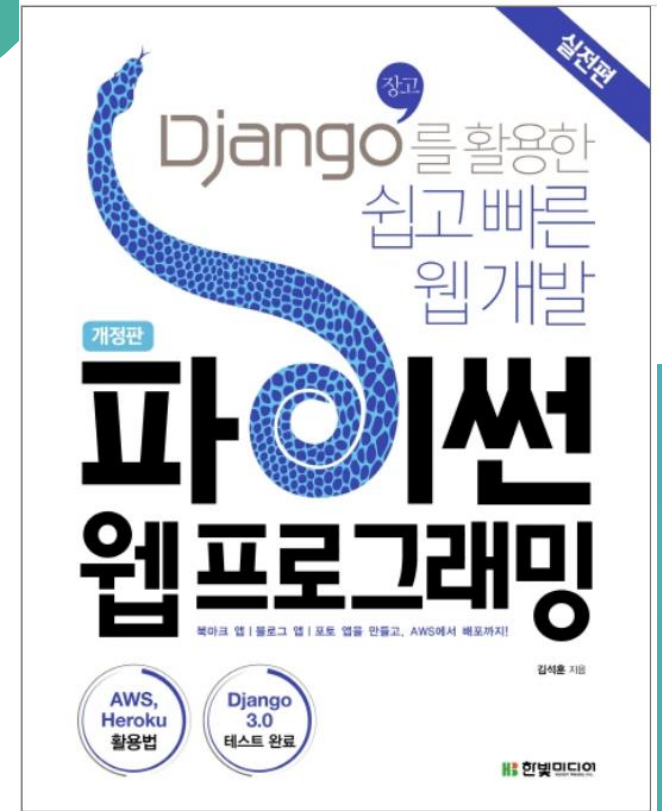


▶ CHAPTER 9 실전 프로그램 개발 - Photo 앱

파이썬 웹프로그래밍



가천대학교 컴퓨터공학과
왕보현



CHAPTER 9 실전 프로그램 개발 – Photo 앱







- Foreign Key로 연결된 두 테이블의 처리
- 사용자 정의 필드 만드는 방법








/photo/
/photo/album ➡ AlbumLV ➡ album_list.html

Django - Python Web Programming Home Bookmark Blog Photo Namecard Student Util ▾ Add ▾ Chang



Django Django related pictures



Nature 자연의 동물, 식물들



사람들 자연을 포함한 사람들의 표정들




```

graph LR
    A["/photo/photo/00"] --> B["PhotoDV"]
    B --> C["photo_detail.html"]

```

Django Book Tag Cloud



- Photo Description

장고 책의 주요 단어로 만든 태그 클라우드 그래픽.

- Date Uploaded

Oct. 17, 2020, 7:42 p.m.

- Album Name

Django



1. Photo App 생성 및 settings.py 수정

Photo App 생성

```
ch99>python manage.py startapp photo
```

settings.py 수정

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
  
    'bookmark.apps.BookmarkConfig',  
    'blog.apps.BlogConfig',  
    'namecard.apps.NamecardConfig',  
    'student.apps.StudentConfig',  
    'widget_tweaks',  
  
    'photo.apps.PhotoConfig', ##추가  
]
```

```
MEDIA_URL = '/media/'          ## 이 두 줄이 있나 확인  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

※ Media File : 사용자가 웹에서 upload 하는 정적 파일
Media File에 접근하는 URL이 MEDIA_URL
Media File이 위치하는 서버 상 경로는 MEDIA_ROOT에 할당



2. 테이블 설계

Album 테이블

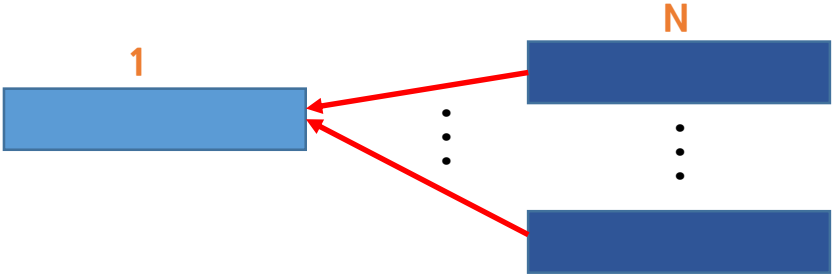
필드명	타입	제약조건, 디폴트	설명
id	integer	PK. Auto Increment	기본 키
name	CharField(50)		앨범 제목
description	CharField(100)	Blank	앨범에 대한 한 줄 설명

Photo 테이블

필드명	타입	제약조건, 디폴트	설명
id	integer	PK. Auto Increment	기본 키
album	ForeignKey	FK(Album.id)	Album에 대한 외래 키
title	CharField(50)		사진 제목
image	ThumbnailImageField		원본 및 썸네일 사진
description	TextField	Blank	사진에 대한 설명
upload_dt	DateTimeField	auto_now_add	사진을 업로드한 일시

참조
1. Album에 없는 기본 키를 참조할 수 없음
2. Album 테이블의 자료가 삭제되면 이를 참조하는 Photo 테이블의 자료들도 삭제되어야 함.

Album 테이블 : Photo 테이블 → 1 : N의 관계





2. 테이블 설계

ch99WphotoWmodels.py

작성

```
from django.db import models
from django.urls import reverse

from photo.fields import ThumbnailImageField ①

class Album(models.Model):
    name = models.CharField('NAME', max_length=30)
    description = models.CharField('One Line Description', max_length=100, blank=True)

    class Meta:
        ordering = ('name',)

    def __str__(self):
        return self.name

    def get_absolute_url(self): ②
        return reverse('photo:album_detail', args=(self.id,))

class Photo(models.Model):
    album = models.ForeignKey(Album, on_delete=models.CASCADE) ③
    title = models.CharField('TITLE', max_length=30)
    description = models.TextField('Photo Description', blank=True) ④
    image = ThumbnailImageField('IMAGE', upload_to='photo/%Y/%m') ⑤
    upload_dt = models.DateTimeField('UPLOAD DATE', auto_now_add=True) ⑥

    class Meta:
        ordering = ('title',) ⑦

    def __str__(self):
        return self.title

    def get_absolute_url(self): ⑧
        return reverse('photo:photo_detail', args=(self.id,))
```

2. 테이블 설계

ch99\photo\models.py 작성

```
from django.db import models
from django.urls import reverse

from photo.fields import ThumbnailImageField ①

class Album(models.Model):
    name = models.CharField('NAME', max_length=30)
    description = models.CharField('One Line Description', max_length=100, blank=True)

    class Meta:
        ordering = ('name',)

    def __str__(self):
        return self.name

    def get_absolute_url(self): ②
        return reverse('photo:album_detail', args=(self.id,))
```

- ① ThumbnailImageField 를 임포트. ThumbnailImageField 필드 클래스는 사진에 대한 원본 이미지와 썸네일 이미지를 모두 저장할 수 있는 필드로, 직접 만든 커스텀 필드임. 이 커스텀 필드는 fields.py 파일에 정의되어 있음.
- ② 이 메소드가 정의된 객체를 지칭하는 URL을 반환. 메소드 내에서는 장고의 내장 함수인 reverse()를 호출. 본 예제에서는 /photo/album/99 형식의 URL을 반환

2. 테이블 설계

ch99\photo\models.py 작성

```
class Photo(models.Model):
    album = models.ForeignKey(Album, on_delete=models.CASCADE) ③
    title = models.CharField('TITLE', max_length=30)
    description = models.TextField('Photo Description', blank=True) ④
    image = ThumbnailImageField('IMAGE', upload_to='photo/%Y/%m') ⑤
    upload_dt = models.DateTimeField('UPLOAD DATE', auto_now_add=True) ⑥
```

- ③ album 컬럼은 Album 테이블에 연결된 외래 키. 즉, 본 사진이 소속된 앨범 객체를 가리키는 reference 역할을 함.
- ④ description 컬럼은 TextField를 사용해 여러 줄의 입력이 가능. 컬럼에 대한 레이블은 Photo Description 이고 내용이 없어도 됨.
- ⑤ image 컬럼은 필드 타입이 ThumbnailImageField 임. ThumbnailImageField 필드는 사진에 대한 원본 이미지 및 썸네일 이미지를 둘 다 저장할 수 있는 필드인데, upload_to 옵션으로 저장할 위치를 지정함.
'photo/%Y/%m'의 의미는 MEDIA_ROOT로 정의된 디렉터리 하위에 ~/photo/2019/08/처럼 연도와 월을 포함해 디렉터리를 만들고 그 곳에 업로드하는 사진의 원본 및 썸네일 사진을 저장. 본 예제에서는 2019년 8월에 사진을 업로드 한다면 다음 디렉터리에 사진이 저장됨. 업로드하는 시점에 디렉터리가 없다면 자동으로 생성
예) /ch99/media/photo/2019/08/
- ⑥ upload_dt 컬럼은 날짜와 시간을 입력하는 DateTimeField이며 auto_now_add 속성은 객체가 생성될 때의 시각을 자동으로 기록하게 함. 즉 사진이 처음 업로드되는 시간을 자동으로 기록



2. 테이블 설계

ch99WphotoWmodels.py 작성

```
class Meta:
    ordering = ('title',) ⑦

    def __str__(self):
        return self.title

    def get_absolute_url(self): ⑧
        return reverse('photo:photo_detail', args=(self.id,))
```

- ⑦ Meta 내부 클래스로, 객체 리스트를 출력할 때의 정렬 기준을 정의함. title 컬럼을 기준으로 오름차순으로 정렬
- ⑧ get_absolute_url() 메소드는 이 메소드가 정의된 객체를 지칭하는 URL을 반환. 메소드 내에서는 장고의 내장 함수인 reverse()를 호출. 본 예제에서는 /photo/photo/99/ 형식의 URL을 반환



2. 테이블 설계

ch99WphotoWadmin.py 작성

```
from django.contrib import admin

from photo.models import Album, Photo

class PhotoInline(admin.StackedInline): ①
    model = Photo                        ②
    extra = 2                           ③

@admin.register(Album)
class AlbumAdmin(admin.ModelAdmin):      ④
    inlines = (PhotoInline,)
    list_display = ('id', 'name', 'description')

@admin.register(Photo)                  ⑤
class PhotoAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'upload_dt')
```

- ① 외래 키로 연결된 Album, Photo 테이블 간에는 1:N 관계가 성립되므로, 앨범(Album) 객체를 보여줄때 객체에 연결된 사진(Photo) 객체들을 같이 보여줄 수 있음. 같이 보여주는 형식은 StackedInline과 TabularInline 두 가지가 있음.

Home » Photo » Albums » Django

Change album

NAME:

One Line Description:

PHOTOS

Photo: Django Book Tag Cloud View on site

TITLE:

Photo Description:

장고 책의 주요 단어로 만든 태그 클라우드 그래픽.

IMAGE:

Currently: photo/2019/08/django-cloud.jpg

Change: 선택된 파일 없음

StackedInline

Change album HISTORY VIEW ON SITE

NAME:

One Line Description:

PHOTOS

TITLE	PHOTO DESCRIPTION	IMAGE	DELETE?
Django Book Tag Cloud View on site			
<input type="text" value="Django Book Tag Cloud"/>	<div>장고 책의 주요 단어로 만든 태그 클라우드 그래픽.</div>	<div>Currently: photo/2019/08/django-cloud.jpg</div> <div>Change: <input type="button" value="파일 선택"/> 선택된 파일 없음</div>	<input type="checkbox"/>

TabularInline

2. 테이블 설계

ch99WphotoWadmin.py 작성

```
from django.contrib import admin

from photo.models import Album, Photo

class PhotoInline(admin.StackedInline): ①
    model = Photo                        ②
    extra = 2                            ③

@admin.register(Album)
class AlbumAdmin(admin.ModelAdmin):      ④
    inlines = (PhotoInline,)
    list_display = ('id', 'name', 'description')

@admin.register(Photo)                  ⑤
class PhotoAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'upload_dt')
```

- ② 추가로 보여주는 테이블이 Photo 테이블 임.
- ③ 이미 존재하는 객체 외에 추가로 입력할 수 있는 Photo 테이블 객체의 수는 2개를 의미
- ④ Album 객체(레코드) 수정 화면을 보여줄 때 PhotoInline 클래스에 정의한 사항을 같이 보여줌.
- Album 객체 수정 시 Photo 객체를 Inline으로 수정 화면에 추가함.
하나의 Album에 연결된 Photo들을 수정할 수 있음.
- ⑤ Photo와 PhotoAdmin 클래스를 등록할 때 admin.site.regiter()함수를 사용해도 되지만
데코레이터(@)를 사용하면 좀 더 간단
- ※ 데코레이터 – 함수를 받아 명령을 추가한 뒤 이를 다시 함수의 형태로 반환하는 함수

※ 이후 makemigrations 및 migrate 수행



2. 테이블 설계

ch99WphotoWfields.py 작성

```
import os
from PIL import Image ①
from django.db.models import ImageField ②
from django.db.models.fields.files import ImageFieldFile

class ThumbnailImageFieldFile(ImageFieldFile): ③
    def _add_thumb(self, s): ④
        parts = s.split('.')
        parts.insert(-1, 'thumb')
        if parts[-1].lower() not in ('jpeg', 'jpg'): ⑤
            parts[-1] = 'jpg'
        return '.'.join(parts)

    @property ⑥
    def thumb_path(self):
        return self._add_thumb(self.path)

    @property ⑦
    def thumb_url(self):
        return self._add_thumb(self.url)

    def save(self, name, content, save=True): ⑧
        super().save(name, content, save) ⑨

        img = Image.open(self.path)
        size = (self.field.thumb_width, self.field.thumb_height) ⑩
        img.thumbnail(size)
        background = Image.new('RGB', size, (255, 255, 255)) ⑪
        box = (int((size[0]-img.size[0])/2), int((size[1]-img.size[1])/2)) ⑫
        background.paste(img, box)
        background.save(self.thumb_path, 'JPEG') ⑬

    def delete(self, save=True): ⑭
        if os.path.exists(self.thumb_path):
            os.remove(self.thumb_path)
        super().delete(save)

class ThumbnailImageField(ImageField): ⑮
    attr_class = ThumbnailImageFieldFile ⑯

    def __init__(self, verbose_name=None, thumb_width=128, thumb_height=128, **kwargs): ⑰
        self.thumb_width, self.thumb_height = thumb_width, thumb_height
        super().__init__(verbose_name, **kwargs) ⑱
```

2. 테이블 설계 ch99WphotoWfields.py 작성

```
import os
from PIL import Image          ①
from django.db.models import ImageField ②
from django.db.models.fields.files import ImageFieldFile

class ThumbnailImageFieldFile(ImageFieldFile): ③
    def _add_thumb(self, s):          ④
        parts = s.split('.')
        parts.insert(-1, 'thumb')
        if parts[-1].lower() not in ('jpeg', 'jpg'): ⑤ 이미지의 확장자가 jpeg 또는 jpg가 아니면 jpg로 변경
            parts[-1] = 'jpg'
        return ''.join(parts) ⑤-1 join 함수 : 리스트를 특정 구분자를 포함해 문자열로 변화해 주는 함수
```

※ 커스텀 필드를 작성할 때 파일명은 보통 fields.py라고 지정하지만 개발자가 임의로 원하는 파일명을 사용 가능

※ 커스텀 필드를 작성할 때는 기존의 비슷한 필드를 상속받아 작성하는 것이 보통. 이미지 커스텀 필드는 ImageField 클래스를 상속받아 작성

※ ImageField 필드는 이미지 파일을 파일 시스템에 쓰고 삭제하는 작업이 필요하므로 추가적으로 ImageFieldFile 클래스가 필요하고 두 개의 클래스를 연계시켜 주는 코드도 필요

※ 커스텀 필드를 작성하는 방법 : <https://docs.djangoproject.com/en/3.1/howto/custom-model-fields/> 참고

① 1장에서 설치한 파이썬의 이미지 처리 라이브러리 PIL.image 를 import함

② 장고의 기본 필드인 ImageField, ImageFieldFile 클래스를 import 함

③ ThumbnailImageFieldFile 클래스는 ImageFieldFile을 상속받음. 이 클래스는 파일 시스템에 직접 파일을 쓰고 지우는 작업 수행

④ 이 메소드는 기존 이미지 파일명을 기준으로 썸네일 이미지 파일명을 만들어 줌. 예를 들어 이미지 파일이 abc.jpg이면 썸네일 이미지 파일명은 abc.thumb.jpg가 됨. 여기서는 썸네일 이미지의 경로나 URL을 만들 때 사용함.

2. 테이블 설계

ch99WphotoWfields.py 작성

```
@property ⑥
def thumb_path(self):
    return self._add_thumb(self.path)

@property ⑦
def thumb_url(self):
    return self._add_thumb(self.url)

def save(self, name, content, save=True): ⑧
    super().save(name, content, save) ⑨
```

- ⑥ 이미지를 처리하는 필드는 파일의 경로(path)와 URL(url) 속성을 제공해야 함. 이 메소드는 원본 파일의 경로인 path 속성에 추가해, 썸네일의 경로인 thumb_path 속성을 만듦. @property 데코레이터를 사용하면, 메소드를 멤버 변수처럼 사용가능
- ⑦ 이 메소드는 원본 파일의 URL인 url 속성에 추가해, 썸네일의 URL인 thumb_url 속성을 만듦. 역시 @property 데코레이터를 사용
- ⑧ 파일 시스템에 파일을 저장하고 생성하는 메소드
- ⑨ 부모 ImageFieldFile 클래스의 save() 메소드를 호출해 원본 이미지를 저장

2. 테이블 설계

ch99WphotoWfields.py 작성

```
def save(self, name, content, save=True):
    super().save(name, content, save)

    img = Image.open(self.path)
    size = (self.field.thumb_width, self.field.thumb_height) ⑩
    img.thumbnail(size)
    background = Image.new('RGB', size, (255, 255, 255)) ⑪
    box = (int((size[0]-img.size[0])/2), int((size[1]-img.size[1])/2)) ⑫
    background.paste(img, box)
    background.save(self.thumb_path, 'JPEG') ⑬

def delete(self, save=True): ⑭
    if os.path.exists(self.thumb_path):
        os.remove(self.thumb_path)
    super().delete(save)
```

- ⑩ 원본 파일로부터 디폴트 128x128px 크기의 썸네일 이미지를 만듦. 썸네일 크기의 최대값을 필드 옵션으로 지정 가능. 썸네일 이미지를 만드는 함수는 PIL 라이브러리의 Image.thumbnail() 함수임. 이 함수는 썸네일을 만들 때 원본 이미지의 가로 x 세로 비율을 유지
- ⑪ RGB 모드인 동일한 크기의 백그라운드 이미지를 만듦. 이미지 색상은 흰색
- ⑫ 썸네일과 백그라운드 이미지를 합쳐서 정사각형(디폴트 크기인 경우) 모양의 썸네일 이미지를 만듦. 정사각형의 빈 공간은 백그라운드 이미지에 의해서 하얀색이 됨
- ⑬ 합쳐진 최종 이미지를 JPEG 형식으로 파일 시스템의 thumb_path 경로에 저장함
- ⑭ delete() 메소드 호출 시 원본 이미지뿐만 아니라 썸네일 이미지도 같이 삭제되도록 함

2. 테이블 설계

ch99WphotoWfields.py 작성

```
class ThumbnailImageField(ImageField): ⑮
    attr_class = ThumbnailImageFieldFile ⑮

    def __init__(self, verbose_name=None, thumb_width=128, thumb_height=128, **kwargs): ⑰
        self.thumb_width, self.thumb_height = thumb_width, thumb_height
        super().__init__(verbose_name, **kwargs) ⑱
```

- ⑮ ThumbnailImageField 클래스는 ImageField 클래스를 상속. 이 클래스가 장고 모델 정의에 사용하는 필드 역할
- ⑮ ThumbnailImageField와 같은 새로운 FileField 클래스를 정의할 때는 그에 상응하는 File 처리 클래스를 attr_class 속성에 지정하는 것이 필수. ThumbnailImageField에 상응하는 File 클래스로, 3에서 정의한 ThumbnailImageFieldFile을 지정
- ⑰ 모델의 필드 정의 시 이미지의 최대 크기로 thumb_width, thumb_height 옵션을 지정할 수 있으며, 지정하지 않으면 디폴트로 128px 을 사용. 필드에 대한 별칭을 줄 수도 있음.
- ⑱ 부모 ImageField 클래스의 생성자를 호출해 관련 속성들을 초기화 함.



2. URL 설계

URL 설계

URL 패턴	뷰 이름	템플릿 파일명
/photo/	AlbumLV(ListView)	album_list.html
/photo/album	AlbumLV(ListView)	album_list.html
/photo/album/99/	AlbumDV(DetailView)	album_detail.html
/photo/photo/99/	AlbumDV(DetailView)	album_detail.html

2. URL 설계

ch99\mysite\urls.py

```
from django.conf import settings  ##추가 ①
from django.conf.urls.static import static  ##추가 ②
from django.contrib import admin
from django.urls import path, include

from mysite.views import HomeView

urlpatterns = [
    path('admin/', admin.site.urls),
    # shkim
    path("", HomeView.as_view(), name='home'),
    path('bookmark/', include('bookmark.urls')),
    path('blog/', include('blog.urls')),
    path('photo/', include('photo.urls')),  ##추가 ③

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT) ## 추가 ④
```

- ① settings 변수를 임포트. settings 변수는 settings.py 모듈에서 정의한 항목들을 담고 있는 객체를 가리키는 reference 임
- ② static() 함수를 임포트. static() 함수는 정적 파일을 처리하기 위해 그에 맞는 URL 패턴을 반환하는 함수
- ③ /photo/ URL 요청이 오면, 토포 앱의 APP_URLCONF에 처리를 위임함



2. URL 설계

ch99\mysite\urls.py

```
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT) ## 추가 ④
```

④ 기존 URL 패턴에 static() 함수가 반환하는 URL 패턴을 추가함. static() 함수 형식은 다음과 같음

static(prefix, view=django.views.static.server, **kwargs)

즉, settings.MEDIA_URL로 정의된 /media/ URL 요청이 오면 django.views.static.server() 뷰 함수가 처리하고, 이 뷰 함수에 document_root=settings.MEDIA_ROOT 키워드 인자가 전달됩니다. static.server() 함수는 개발용이고 상용에는 httpd, nginx 등의 웹 서버 프로그램을 사용합니다.

※ Media File : 사용자가 웹에서 upload 하는 정적 파일

Media File에 접근하는 URL이 MEDIA_URL

Media File이 위치하는 서버 상 경로는 MEDIA_ROOT에 할당



2. URL 설계

ch99WphotoWurls.py 작성

```
from django.urls import path

from photo import views

app_name = 'photo'
urlpatterns = [

    # Example: /photo/
    path("", views.AlbumLV.as_view(), name='index'),

    # Example: /photo/album/, same as /photo/
    path('album', views.AlbumLV.as_view(), name='album_list'),

    # Example: /photo/album/99/
    path('album/<int:pk>', views.AlbumDV.as_view(), name='album_detail'),

    # Example: /photo/photo/99/
    path('photo/<int:pk>', views.PhotoDV.as_view(), name='photo_detail'),

]
```



3. View 코딩

ch99₩photo₩views.py 작성

```
from django.views.generic import ListView,
DetailView
from photo.models import Album, Photo
```

```
class AlbumLV(ListView):
    model = Album
```

```
class AlbumDV(DetailView):
    model = Album
```

```
class PhotoDV(DetailView):
    model = Photo
```



views.py 작성 대신 urls.py작성 → ch99₩photo₩urls.py 작성

```
from django.urls import path
```

```
from photo.views.generic import ListView, DetailView
```

```
app_name = 'photo'
urlpatterns = [
```

```
    # Example: /photo/
    path("", ListView.as_view(model=Album), name='index'),
```

```
    # Example: /photo/album/, same as /photo/
    path('album', ListView.as_view(model=Album), name='album_list'),
```

```
    # Example: /photo/album/99/
    path('album/<int:pk>', DetailView.as_view(model=Album), name='album_detail'),
```

```
    # Example: /photo/photo/99/
    path('photo/<int:pk>', DetailView.as_view(model=Photo), name='photo_detail'),
```

```
]
```




3. 템플릿 코딩

ch99\photo\templates\photo\album_list.html

- ① item.photo_set.all[slice:":5"] 표현식은 특정 앨범 객체에 들어 있는 사진 객체 리스트에서 앞에서부터 5개 객체를 추출함. slice 템플릿 필터는 파이썬 리스트의 슬라이싱 동작을 수행하는 [m:n] 표현식과 유사
- ② 각 썸네일 사진에 URL 링크를 연결함. 링크는 객체의 get_absolute_url() 메소드를 호출해 구하는데, /photo/photo/99/와 같은 형식
- ③ photo.image 컬럼은 ThumbnailImageField 필드로서 사진 원본 및 썸네일 정보를 저장. photo.image.thumb_url은 썸네일 사진에 대한 URL이고 photo.image.url은 원본 사진에 대한 URL

```
{% extends "base.html" %}
{% block title %}album_list.html{% endblock %}
{% block extra-style %}
<style>
.thumbnail {
    border: 3px solid #ccc;
}
</style>
{% endblock %}

{% block content %}

    {% for item in object_list %}

        <div class="mt-5">
            <a class="h2" href="{% url 'photo:album_detail' item.id %}">
                {{ item.name }}</a>&emsp;
            <span class="font-italic h5">{{ item.description }}</span>
        </div>

        <hr style="margin: 0 0 20px 0;">

        <div class="row">
            {% for photo in item.photo_set.all[slice:":5"] %} ①
                <div class="ml-5">
                    <div class="thumbnail">
                        <a href="{{ photo.get_absolute_url }}"> ②
                             ③
                        </a>
                    </div>
                </div>
            {% endfor %}
        </div>

    {% endfor %}
{% endblock %}
```

```
class Photo(models.Model):
    album = models.ForeignKey(Album, on_delete=models.CASCADE)
    title = models.CharField("TITLE", max_length=100)
    description = models.TextField("Photo Description")
    image = ThumbnailImageField("IMAGE", upload_to='photos')
    upload_dt = models.DateTimeField("UPLOAD DATE")
```

```
@property
def thumb_url(self): ⑦
    return self._add_thumb(self.url)
```



3. 템플릿 코딩

ch99\photo\templates\photo\album_list.html

```
<div class="row">
  {% for photo in item.photo_set.all|slice:"5" %}
    <div class="ml-5">
      <div class="thumbnail">
```

① item은 Album 의 객체(레코드)

Photo 가 Album의 id를 Foreign Key로 사용하고 있음.

Album의 객체가 자신을 참조하는 Photo 테이블의 객체들을 검색할 때
모델명소문자_set을 사용함.

A : foreign key로 참조되는 모델(테이블)

B : A를 foreign key로 참조하는 모델(테이블)

a : A의 객체

a 객체가 자신을 참조하는 B의 객체들을 저장하는 곳

a.b_set



3. 템플릿 코딩

ch99WphotoWtemplatesWph
album_detail.html

```
{% extends "base.html" %}
{% block title %}album_detail.html{% endblock %}
{% block extra-style %}
<style>
.thumbnail {
    border: 5px solid #ccc;
}
</style>
{% endblock %}
{% block content %}

    <div class="mt-5">
        <span class="h2">{{ object.name }}&nbsp;</span>
        <span class="h5 font-italic">{{ object.description }}</span>
    </div>

    <hr style="margin: 0 0 20px 0;">

    <div class="row">

        {% for photo in object.photo_set.all %}
        <div class="col-md-3 mb-5">
            <div class="thumbnail">
                <a href="{{ photo.get_absolute_url }}">
                    
                </a>
            </div>
            <ul>
                <li class="font-italic">{{ photo.title }}</li>
                <li class="font-italic">{{ photo.upload_dt|date:"Y-m-d" }}</li>
            </ul>
        </div>
        {% endfor %}

    </div>
{% endblock %}
```

① DetailView를 상속받은 뷰는 default로 object 속성에 검색한 객체(레코드)를 저장하여 html에 전달

② Album의 객체를 참조하는 photo 테이블의 객체들

③ 16page 7번 참조



3. 템플릿 코딩

ch99\photo\templates\photo\photo_detail.html

- ① photo 객체 object의 album 컬럼을 통해 album 객체의 name 속성을 읽어옴

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

{% block title %}photo_detail.html{% endblock %}

{% block content %}

    <h2 class="mt-5">{{ object.title }}</h2>

    <div class="row">
        <div class="col-md-9">
            <a href="{{ object.image.url }}">
                
            </a>
        </div>

        <ul class="col-md-3 mt-3">
            <li class="h5">Photo Description</li>
            {% if object.description %}<p>{{ object.description|linebreaks }}</p>
            {% else %}<p>(blank)</p>{% endif %}
            <li class="h5">Date Uploaded</li>
            <p class="font-italic">{{ object.upload_dt }}</p>
            <li class="h5">Album Name</li>
            <p class="font-italic">
                <a href="{% url 'photo:album_detail' object.album.id %}"> {{ object.album.name }}</a> ①
            </p>
        </ul>
    </div>

{% endblock %}
```