

Django 기반의 웹프로그래밍

이 소 영

yisy0703@naver.com

Web Frameworks for Python(<https://wiki.python.org/moin/WebFrameworks>)

장고 공식 사이트(<https://www.djangoproject.com/>)

장고 공식 소스 저장소(<http://github.com/django/django>)

장고 참조 문서 영어(<https://docs.djangoproject.com/en/3.2/>)

Agenda

Django(장고) 기반의 파이썬 웹 프로그래밍

Ch01. Django 시작하기

1. Django 란?
2. 개발 환경 구축
3. Django 구조

Ch02. Django App

1. Django Project
2. Model
3. View

Ch03. Model

1. Model 속성 및 옵션
2. Relationship
3. Migrations
4. Admin App

Ch04. Django SQL

1. Django shell
2. Manager & QuerySet
3. 조회 SQL
4. 생성/수정/삭제 SQL
5. Django-Debug-Toolbar

Ch05. Template

1. Template Loader
2. URL Dispatcher
3. Template 상속
4. Template Engines
5. Template Filter

Ch06. Django View

1. View 기본
2. View 활용

Ch07. Django Form

1. HTML form
2. CSRF
3. HttpRequest/HttpResponse
4. Django Form
5. Django Model Form
6. Form Validation

Ch08. File 관리

1. Static Files
2. Media Files
3. Image Thumbnail

Ch09. 사용자 인증

1. Auth App
2. 회원가입 구현
3. 로그인/아웃 구현
4. Oauth 라이브러리 활용

Ch 03 Model

1. Model 속성 및 옵션
2. Relationship
3. Migrations

Chapter 03. Model

Django Model

- 파이썬 ORM(<https://github.com/vinta/awesome-python#orm>)
- django.db.backends(<https://github.com/django/django/tree/master/django/db/backends>)
- models(<https://docs.djangoproject.com/en/3.2/topics/db/models/>)
- Field types(<https://docs.djangoproject.com/ko/3.2/ref/models/fields/#model-field-types>)
- Field options(<https://docs.djangoproject.com/ko/3.2/ref/models/fields/#common-model-field-options>)
- Model Meta options(<https://docs.djangoproject.com/en/3.2/ref/models/options/>)
- SQLite Browser(<http://sqlitebrowser.org/>)
- Many-to-one relationships(https://docs.djangoproject.com/ko/3.2/topics/db/examples/many_to_one/)
- Many-to-many relationships(https://docs.djangoproject.com/ko/3.2/topics/db/examples/many_to_many/)
- One-to-one relationships(https://docs.djangoproject.com/ko/3.2/topics/db/examples/one_to_one/)
- django.contrib.auth.models(<https://github.com/django/django/blob/master/django/contrib/auth/models.py>)
- Migrations(<https://docs.djangoproject.com/ko/3.2/topics/migrations/>)

Django admin app

- The Django admin site(<https://docs.djangoproject.com/en/3.2/ref/contrib/admin/>)
- ModelAdmin 소스(https://docs.djangoproject.com/en/2.2/_modules/django/contrib/admin/options/#ModelAdmin)
- ModelAdmin options(<https://docs.djangoproject.com/en/2.2/ref/contrib/admin/#modeladmin-options>)



1. Model 속성 및 옵션



DB 개요

1. Model 속성 및 옵션

- 데이터 저장 방법

- 데이터 베이스 : RDBMS, NoSQL
- 파일 : 로컬, 외부 정적 스토리지

- 데이터 베이스 종류

- RDBMS : PostgreSQL, MySQL, SQLite, MS-SQL, Oracle 등
- NoSQL : MongoDB, Cassandra, CouchDB, Google Big Table 등
- Django는 RDBMS만 지원

- SQL

- 관계형 데이터베이스 관리 시스템의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

ORM(Object-relational mapping)

1. Model 속성 및 옵션

- ORM(Object-relational mapping)이란?
 - DB와 객체 지향 프로그래밍 언어간의 호환되지 않는 데이터를 변환하는 프로그래밍 기법
 - 객체 지향 언어에서 사용할 수 있는 '가상' 객체 데이터 베이스를 구축하는 방법
 - SQL의 의존적인 코딩에서 벗어나 생산적인 코딩이 가능함
 - 유지보수가 편리함
- 다양한 파이썬 ORM
 - Relational Database
 - Django Models, SQLAlchemy, Orator, Preview, PonyORM
- SQL
 - 관계형 데이터베이스 관리 시스템의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

Django 기본 backends

1. Model 속성 및 옵션


Tag: 2.1.1 ▾

[django](#) / [django](#) / [db](#) / [backends](#) /

Create new file

Find file

History



[jdufresne](#) and [timgraham](#) [2.1.x] Refs #29015 -- Added database name to PostgreSQL database nam...

Latest commit cae8490 on 18 Aug

..

base	[2.1.x] Fixed #29496 -- Fixed crash on Oracle when converting a non-u...	4 months ago
dummy	Refs #27656 -- Updated django.db docstring verbs according to PEP 257.	2 years ago
mysql	[2.1.x] Fixed #29544 -- Fixed regex lookup on MariaDB.	3 months ago
oracle	[2.1.x] Fixed #29496 -- Fixed crash on Oracle when converting a non-u...	4 months ago
postgresql	[2.1.x] Refs #29015 -- Added database name to PostgreSQL database nam...	2 months ago
postgresql_psycopg2	Removed postgresql_psycopg2.version	a year ago
sqlite3	Refs #29350 -- Fixed 'invalid escape sequence' warning in SQLite intr...	5 months ago
__init__.py	Fixed #22603 -- Reorganized classes in django.db.backends.	4 years ago
ddl_references.py	Fixed #28465 -- Unified index SQL creation in DatabaseSchemaEditor	a year ago
signals.py	Fixed #13798 -- Added connection argument to the connection_created s...	8 years ago
utils.py	Used Decimal.scaleb() in backends.utils.format_number() and DecimalFi...	9 months ago

Django Model

1. Model 속성 및 옵션

- 모델(Model)과 폼(form)
 - 생산성 높은 웹 서비스 개발이 가능
- Model/Form 함께 다양한 ORM 라이브러리 사용 가능
- **SQL문 직접 실행 가능하나 ORM 사용**

```
D:\src\django\myproject>python manage.py shell
```

```
import sqlite3
```

```
con = sqlite3.connect("test.db")
```

```
cur = con.cursor()
```

```
cur.execute("create table phoneBook(Name text, PhoneNum text)")
```

```
cur.execute("insert into phoneBook values('Amy', '010-123-4569')")
```

```
cur.execute('select * from phonebook')
```

```
for row in cur:
```

```
    print(row)
```

Django Model

1. Model 속성 및 옵션

- 모델클래스는 DB 테이블과 1:1 매핑됨
 - 모델클래스명은 단수형으로 지정 Person(O), Persons(X)
 - 모델 클래스는 DB 테이블 필드와 일치해야 함
- 모델클래스는 django.db.models.Model을 상속해야 함
- 테이블이름은 " 앱이름_모델이름" 소문자로 자동 생성됨

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```



```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Django Model 처리 방법

1. Model 속성 및 옵션

- Django Model을 통해 DB 형상 관리하는 경우

- 1) 모델 클래스 작성
- 2) 모델 클래스로부터 마이그레이션 파일 생성
 - makemigrations 명령
- 3) 마이그레이션 파일을 DB에 적용
 - migrate 명령
- 4) 모델 활용

- Django 외부에서 DB 형상 관리하는 경우

- 1) 모델 활용

Model Fields Type

1. Model 속성 및 옵션

```
from django.db import models

class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

- 모델의 필드는 DB 테이블의 필드와 매핑
- 모델의 필드 타입에 따라 다음 사항이 결정
 - DB에 저장할 데이터 타입 (예: INTEGER, VARCHAR, TEXT)
 - HTML 폼에서 사용할 위젯 (예: <input type="text">, <select>)
 - Django admin 또는 자동 생성 폼에서 검사할 validator

Model Fields Type

1. Model 속성 및 옵션

- <https://docs.djangoproject.com/ko/3.2/ref/models/fields/#model-field-types>
- Primary Key
 - AutoField, BigAutoField
- 문자열
 - CharField, TextField, SlugField
- 날짜/시간
 - DateField, DateTimeField, TimeField, DurationField
- 참/거짓
 - BooleanField, NullBooleanField
- 숫자
 - IntegerField, DecimalField, FloatField, PositiveIntegerField, PositiveSmallIntegerField, SmallIntegerField, BigIntegerField

Model Fields Type

1. Model 속성 및 옵션

- 파일

- BinaryField, **FileField**, FilePathField, **ImageField**

- 이메일 : EmailField

- URL : URLField

- UUID : UUIDField

- IP : GenericIPAddressField

문자열 + 글자패턴(정규표현식 validator check포함)

- Relationship Types

- ForeignKey

- ManyToManyField

- OneToOneField

다른 테이블과 관계 설정

Model Fields Type

1. Model 속성 및 옵션

- Automatic primary key fields

- 기본적으로 Django는 각 모델에 **다음 필드를 자동 추가**

```
id = models.AutoField(primary_key=True)
```

- id 값은 자동으로 증가
- 모델 설계시 **primary_key=True 필드가 존재하면 자동 추가되지 않음**
- 모델은 반드시 하나의 primary_key = True 필드를 가져야 함

Model Fields Type

1. Model 속성 및 옵션

● 파이썬 타입 = 모델 타입

- int = AutoField
- bytes = BinaryField
- bool = BooleanField
- None, bool = NullBooleanField
- str = CharField/TextField/EmailField/GenericIPAddress/SlugField/URLField

● 테이블 타입 = 모델 타입

- varchar = CharField, SlugField, URLField, EmailField 등
- DB에 따라 다른 타입으로 처리될 수 있음

Model Fields Options

1. Model 속성 및 옵션

- <https://docs.djangoproject.com/ko/3.2/ref/models/fields/#common-model-field-options>
- **null**
 - DB와 관련. DB 컬럼의 null 값 여부 지정. 기본값 False
- **blank**
 - 유효성과 관련. form.is_valid() 호출시 폼 유효성 검사에 사용.
 - 기본값은 False

```
class Person(models.Model):  
    name = models.CharField(max_length=255) #필수  
    bio = models.TextField(max_length=500, blank=True) #선택  
    # bio = models.TextField(max_length=500, null=True) # wrong  
    birth_date = models.DateField(null=True, blank=True) #선택
```

Model Fields Options

1. Model 속성 및 옵션

● choices

- list 또는 tuple 항목으로 필드 값 지정

```
from django.db import models

class Student(models.Model):
    FRESHMAN = 'FR'
    SOPHOMORE = 'SO'
    JUNIOR = 'JR'
    SENIOR = 'SR'
    YEAR_IN_SCHOOL_CHOICES = (
        (FRESHMAN, 'Freshman'),
        (SOPHOMORE, 'Sophomore'),
        (JUNIOR, 'Junior'),
        (SENIOR, 'Senior'),
    )
    year_in_school = models.CharField(
        max_length=2,
        choices=YEAR_IN_SCHOOL_CHOICES,
        default=FRESHMAN,
    )

    def is_upperclass(self):
        return self.year_in_school in (self.JUNIOR, self.SENIOR)
```

student 추가

Year in school:

Freshman ▼

Freshman

Sophomore

Junior

Senior

Model Fields Options

1. Model 속성 및 옵션

- **db_column**
 - DB 컬럼이름 지정, 기본값은 필드이름
- **db_index**
 - true이면 필드의 DB 인덱스를 생성
- **db_tablespace**
 - 필드의 index가 설정된 경우 db tablespace의 이름 지정
- **default**
 - 필드의 기본값 지정

```
def contact_default():  
    return {"email": "to1@example.com"}  
  
contact_info = JSONField("ContactInfo", default=contact_default)
```

Model Fields Options

1. Model 속성 및 옵션

- **editable**
 - False이면 admin 또는 ModelForm에서 표시되지 않고 모델 유효성 검사도 생략함.
기본값은 True
- **error_message**
 - 필드의 기본 메시지값 변경. 사전 타입으로 지정
- **help_text**
 - 위젯과 함께 표시되는 도움말
- **primary_key**
 - 기본키 설정
- **unique**
 - 중복되지 않는 값을 가짐
- **unique_for_date, unique_for_month, unique_for_year**
 - 고유한 날짜 필드 값을 가짐

Model Fields Options

1. Model 속성 및 옵션

● verbose_name : 필드의 레이블

- verbose name 지정 → 첫번째 인자로 지정

```
first_name = models.CharField("person's first name", max_length=30)
```

- verbose name 미지정시 → first_name은 "first name"으로 지정
- ForeignKey, ManyToManyField, OneToOneField 는 verbose_name 옵션 사용

```
poll = models.ForeignKey(
    Poll,
    on_delete=models.CASCADE,
    verbose_name="the related poll",
)
sites = models.ManyToManyField(Site, verbose_name="list of sites")
place = models.OneToOneField(
    Place,
    on_delete=models.CASCADE,
    verbose_name="related place",
)
```

Model Fields Options

1. Model 속성 및 옵션

● validators

- 유효성 검사를 수행할 다수의 함수 지정

```
from django.core.exceptions import ValidationError
from django.utils.translation import gettext_lazy as _

def validate_even(value):
    if value % 2 != 0:
        raise ValidationError(
            _('%(value)s is not an even number'),
            params={'value': value},
        )
```

```
from django.db import models

class MyModel(models.Model):
    even_field = models.IntegerField(validators=[validate_even])
```

Model Method

1. Model 속성 및 옵션

● 클래스 메소드

- 테이블 레벨에서 동작하는 메소드. 장고에서는 지원하지 않음
- Manager 클래스를 통해 테이블 레벨의 메소드 지원

● 객체 메소드

- 레코드 레벨에서 지원하는 메소드

```
class Post(models.Model):  
    title = models.CharField(max_length=100)  
    content = models.TextField()  
    create_at = models.DateField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)  
  
    def __str__(self):  
        return self.title
```

Model 내부 클래스

1. Model 속성 및 옵션

- 모델에 대한 메타데이터 정의
- <https://docs.djangoproject.com/en/3.2/ref/models/options>
- 자주 사용하는 Meta 내부 클래스 속성
 - ordering : 리스트 출력시 정렬 기준
 - db_table : DB에 저장되는 테이블 이름 지정
 - verbose_name : 모델 객체의 별칭

```
class Post(models.Model):  
  
    class Meta:  
        ordering = ['-updated_at']
```


DB 확인 방법

1. Model 속성 및 옵션

● Shell 에서 확인

```
>> python manage.py dbshell  
>> .tables  
>> .schema blog_post
```

● SQLiteBrowser

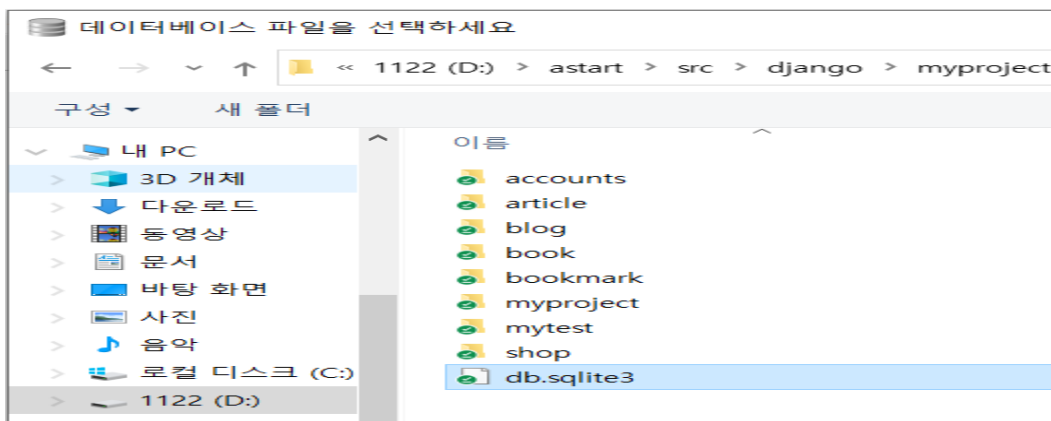
- <http://sqlitebrowser.org/>



SQLiteBrowser

1. Model 속성 및 옵션

- SQLiteBrowser 실행
- 파일 -> 데이터베이스 열기
- 파일 선택



- 데이터보기에서 blog_post 선택

The screenshot shows the SQLiteBrowser interface with the "데이터 보기" tab selected. The "테이블(T):" dropdown shows "blog_post". The table data is as follows:

	id	title	content	created_at	updated_at
1	1	농구	제목1이 본문...	2021-12-21	2021-12-21 10...
2	2	아이스하키	빙판에서 5명...	2021-12-21	2021-12-21 10...

Blog 앱 수정(1)

1. Model 속성 및 옵션

```
# blog/models.py - 필드, 옵션 추가
```

```
from django.db import models
```

```
class Post(models.Model):
    title = models.CharField(max_length=100, verbose_name='제목',
                             help_text='포스팅 제목을 입력해주세요. 최대 100자 내외')
    content = models.TextField(verbose_name='내용')
    tags = models.CharField(max_length=100, blank=True)
    lnglat = models.CharField(max_length=50, blank=True,
                              help_text='경도, 위도 포맷으로 입력')
    create_at = models.DateField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
c:\dev\myproject>python manage.py makemigrations blog
c:\dev\myproject>python manage.py migrate blog
```

blog 앱 수정(2)

1. Model 속성 및 옵션

`blog/models.py` - 필드추가시 기본값 지정

```
class Post(models.Model):
    author = models.CharField(max_length=20)
    title = models.CharField(max_length=100, ~ )
    ~ 생략 ~
```

```
c:\dev\myproject>python manage.py makemigrations blog
```

You are trying to add a non-nullable field 'author' to post without a default; we can't do that (the database needs something to populate existing rows).

Please select a fix:

- 1) Provide a one-off default now (will be set on all existing rows with a null value for this column)
- 2) Quit, and let me add a default in models.py

Select an option: **1**

Please enter the default value now, as valid Python

The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now

Type 'exit' to exit this prompt

```
>>> 'anonymous'
```

Migrations for 'blog':

```
blog\migrations\0004_post_author.py
- Add field author to post
```

```
c:\dev\myproject>python manage.py migrate blog
```

blog 앱 수정(3)

1. Model 속성 및 옵션

blog/models.py - choices 옵션

```
class Post(models.Model):
    REGION_CHOICE = (
        ('Africa', '아프리카'),
        ('Europe', '유럽'),
        ('Oceania', '오세아니아'),
        ('Asia', '아시아'),
        ('North America', '북아메리카'),
        ('South America', '남아메리카'),
    )
    ~ 생략 ~

    region = models.CharField(max_length=20,
                              choices=REGION_CHOICE, default='Asia')
```

blog 앱 수정(4)

1. Model 속성 및 옵션

blog/models.py - 유효성 체크

```
import re
from django.db import models
from django.forms import ValidationError

def lnglat_validator(value):
    if not re.match(r'(\d+\.\?\d*),(\d+\.\?\d*)$', value):
        raise ValidationError('Invalid LngLat Type')

class Post(models.Model):
    title = models.CharField(max_length=100, verbose_name='제목',
                             help_text='포스팅 제목을 입력해주세요. 최대 100자 내외')

    content = models.TextField(verbose_name='내용')

    tags = models.CharField(max_length=100, blank=True)
    lnglat = models.CharField(max_length=50, blank=True, validators=[lnglat_validator],
                              help_text='경도, 위도 포맷으로 입력')

    create_at = models.DateField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

bookmark 앱 작성(1)

1. Model 속성 및 옵션

```
c:\dev\myproject> python manage.py startapp bookmark
```

```
# myproject\settings.py 추가
```

```
INSTALLED_APPS = [  
    'bookmark',  
]
```

```
# bookmark/models.py 생성
```

```
class Bookmark(models.Model):  
    title = models.CharField(max_length=100, blank=True, null=True)  
    url = models.URLField(unique=True)  
  
    def __str__(self):  
        return self.title
```

bookmark 앱 작성(2)

1. Model 속성 및 옵션

```
# bookmark/admin.py
```

```
from django.contrib import admin
from .models import Bookmark
```

```
admin.site.register(Bookmark)
```

```
c:\dev\myproject>python manage.py makemigrations bookmark
c:\dev\myproject>python manage.py migrate bookmark
```

admin 페이지에서 데이터 입력하기

```
Naver   : http://www.naver.com
Daum    : http://www.daum.net
Google  : http://www.google.com
```


bookmark 앱 작성(3)

1. Model 속성 및 옵션

myproject/urls.py 추가

```
urlpatterns = [  
    path('bookmark/', include('bookmark.urls')),  
]
```

bookmark/urls.py 생성

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

```
app_name = 'bookmark'  
urlpatterns = [  
    path('', views.BookmarkLV.as_view(), name='index'),  
    path('<pk>/', views.BookmarkDV.as_view(), name='detail')  
]
```

bookmark 앱 작성(4)

1. Model 속성 및 옵션

```
# bookmark/views.py

from django.shortcuts import render
from django.views.generic import ListView, DetailView
from .models import Bookmark

class BookmarkLV(ListView):
    model = Bookmark

class BookmarkDV(DetailView):
    model = Bookmark
```

● ListView

- 테이블의 레코드 리스트를 보여주기 위한 뷰
- 기본값 - 컨텍스트 변수 : object_list/템플릿 파일 : 모델명소문자_list.html

● DetailView

- 테이블의 특정 레코드에 대한 상세 정보를 보여주기 위한 뷰
- 기본값 - 컨텍스트 변수 : object/템플릿 파일 : 모델명소문자_detail.html

bookmark 앱 작성(5)

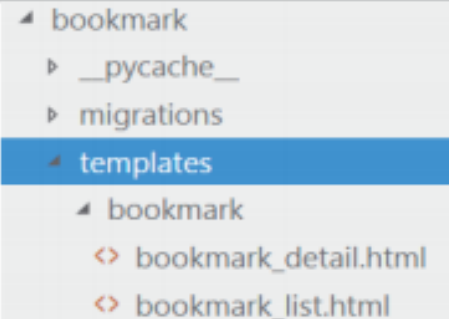
1. Model 속성 및 옵션

bookmark/templates/bookmark/bookmark_list.html 작성

```
<h1>Bookmark List</h1>
<ul>
    {% for bookmark in object_list %}
        <li>
            <a href="/bookmark/{{bookmark.id}}">{{bookmark}}</a>
        </li>
    {% endfor %}
</ul>
```

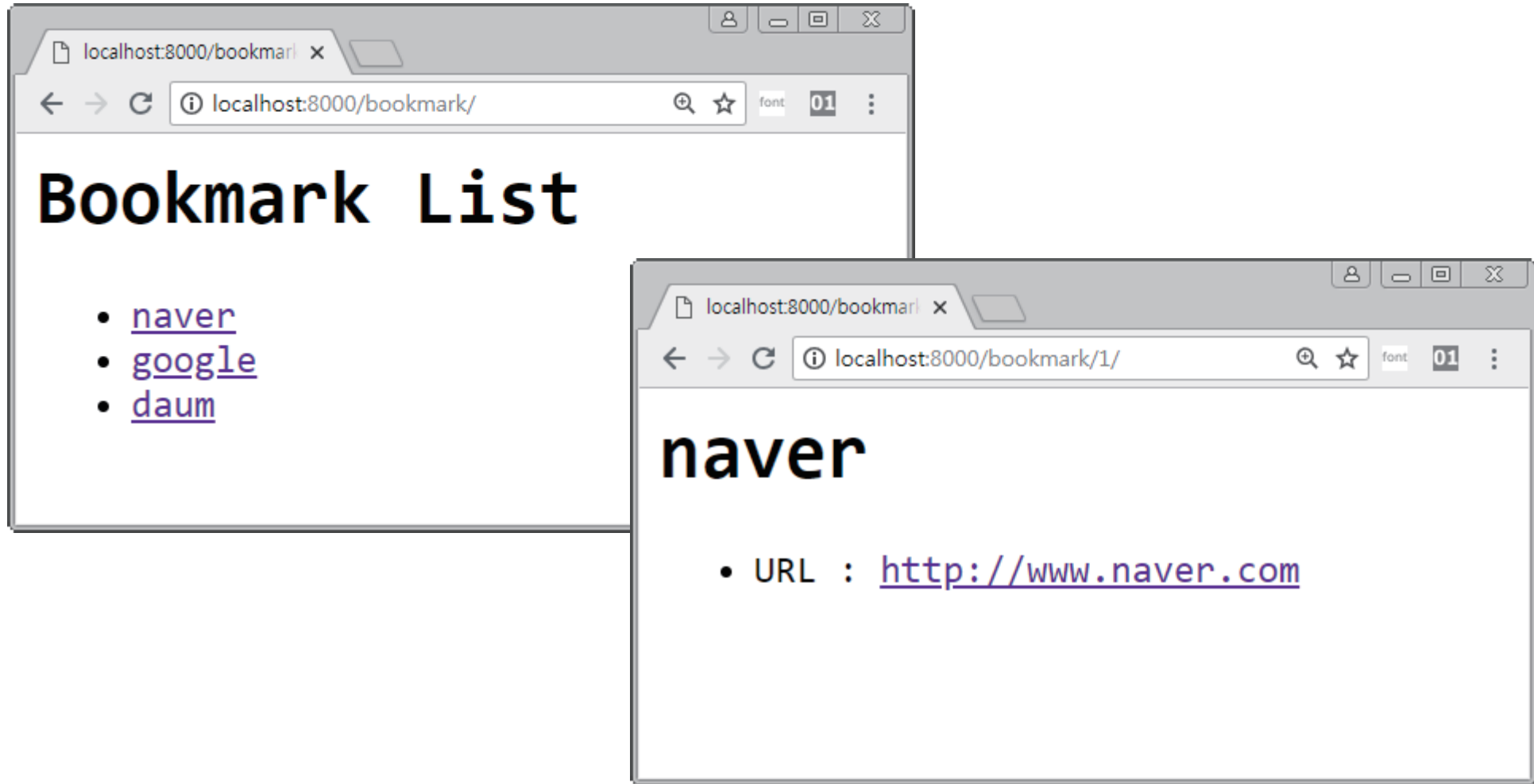
bookmark/templates/bookmark/bookmark_detail.html 작성

```
<h1>{{object.title}}</h1>
<ul>
    <li>URL : <a href="{{object.url}}">{{object.url}}</a></li>
</ul>
```



bookmark 앱 작성(6)

1. Model 속성 및 옵션





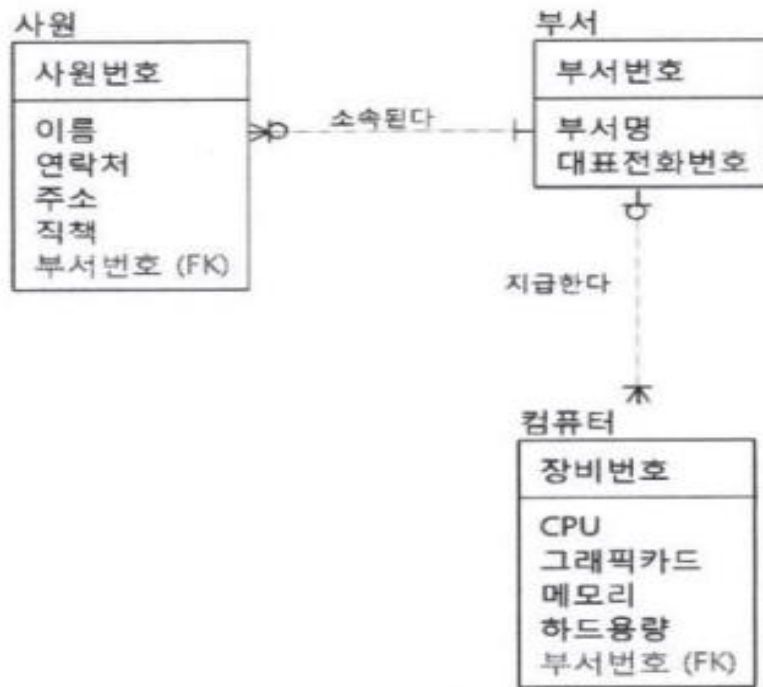
2. Relationship



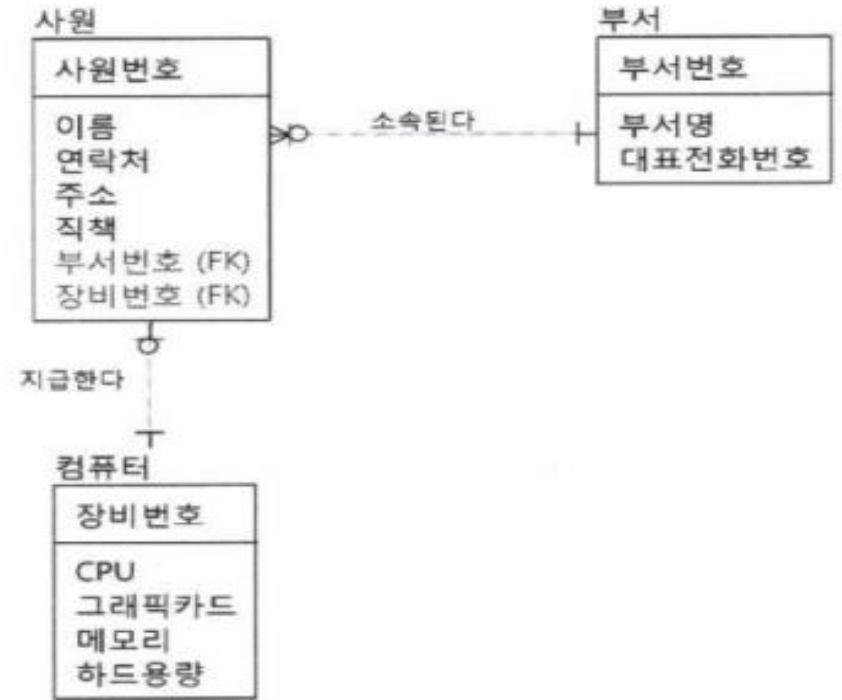
관계(Relationship)

2. Relationship

- 관계란 두 모델 간에 존재하는 업무적인 연관성을 의미
- 업무 규칙이 변경됨에 따라 관계는 변할 수 있음



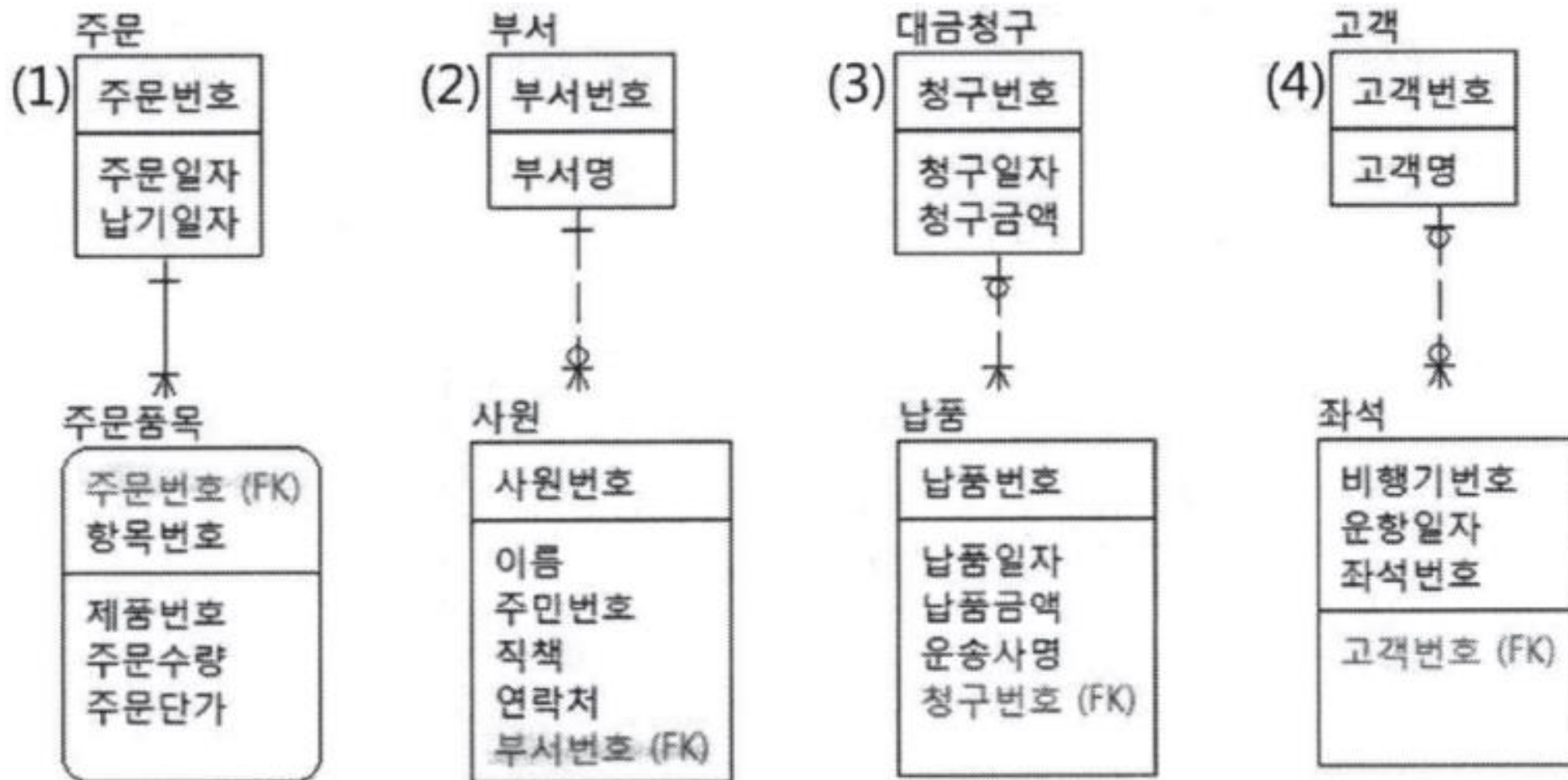
부서 단위로 사원수만큼
여러대 컴퓨터 지급



사원별 한대씩 컴퓨터 지급

1:N 관계

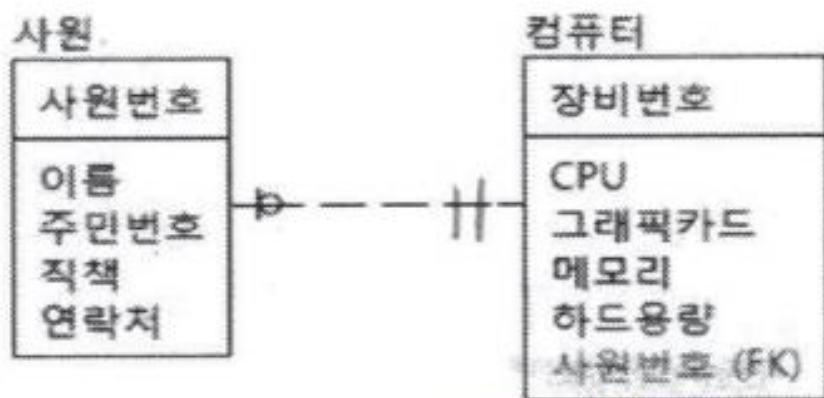
2. Relationship



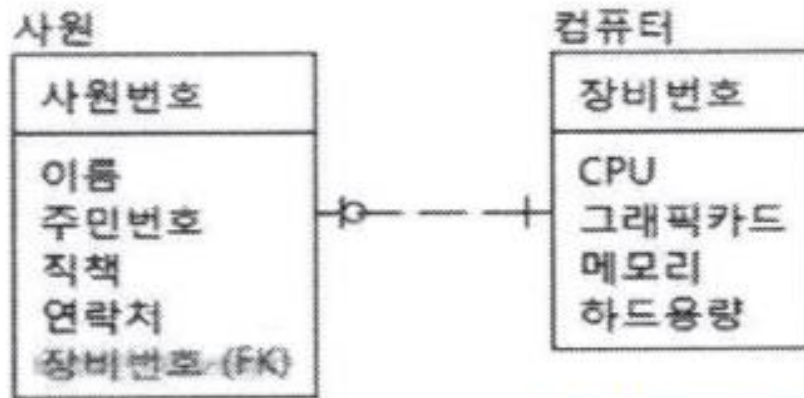
1:1 관계

2. Relationship

- 사원 200명 컴퓨터 250대
- 사원은 반드시 컴퓨터를 한대씩 지급받고, 컴퓨터는 한 명의 사원에게 지급이 되거나 안될 수도 있는 경우
- “1:1의 관계는 어느 한쪽의 식별자를 다른 쪽의 관계속성을 둔다”가 규칙



(1) 50개 null 발생



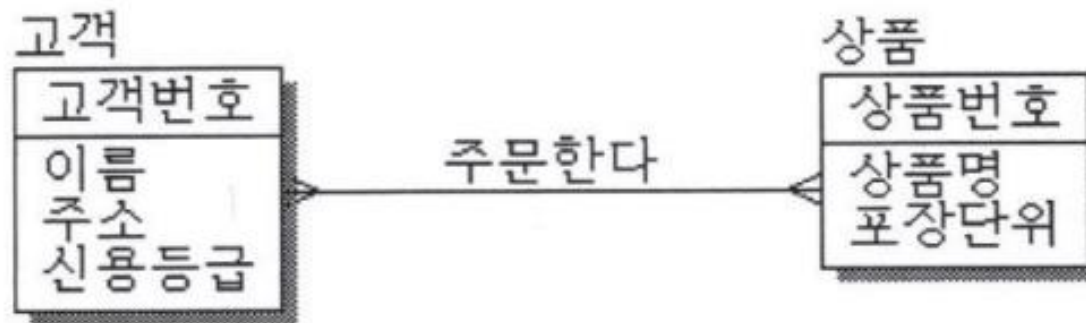
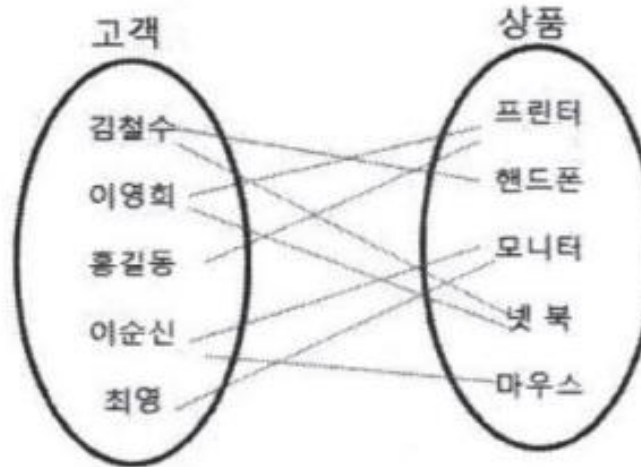
(2) null 발생하지 않음

< FK가 어느 쪽이든 상관없음 >

M:M 관계

2. Relationship

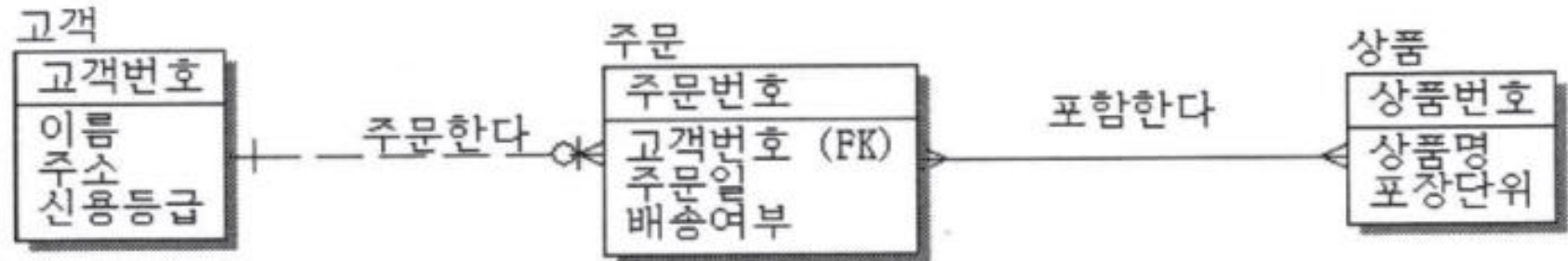
1. 고객과 상품 사이의 M:M의 관계



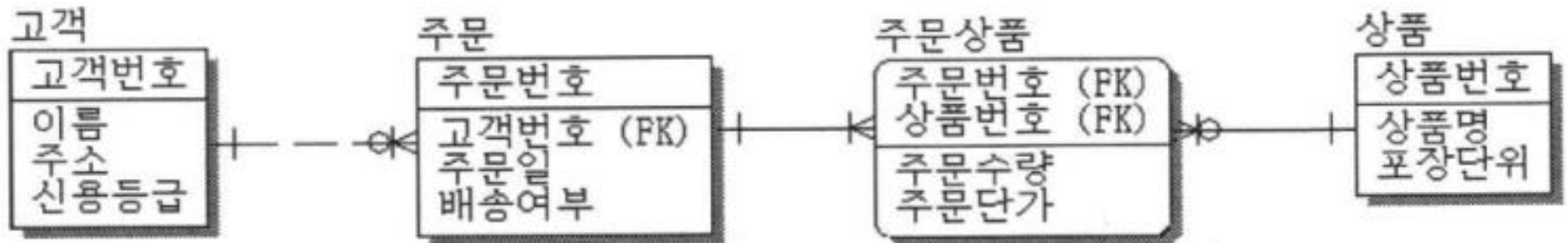
M:M 관계

2. Relationship

2. 1차 M:M의 관계 해소



3. 주문과 상품 사이의 M:M의 관계 해소



ForeignKey

2. Relationship

● Many-to-one 관계

- `django.db.models.ForeignKey` 사용

```
from django.db import models

1 class Manufacturer(models.Model):
    # ...
    : pass

N class Car(models.Model):
    manufacturer = models.ForeignKey(Manufacturer, on_delete=models.CASCADE)
    # ...
```

- 1:N 관계에서 N측에 명시
- 참조문서
 - https://docs.djangoproject.com/ko/3.2/topics/db/examples/many_to_one

ForeignKey

2. Relationship

● ForeignKey(to, on_delete)

- to : 대상 모델
 - 클래스명 지정
 - 클래스명을 문자열로 지정.
- on_delete : 1측의 Row가 삭제될 경우, N측의 Row의 처리에 대한 동작을 지정
 - CASCADE : 연결된 Row 를 일괄 삭제 (디폴트 동작)
 - PROTECT : ProtectedError 예외를 발생시키며, 삭제 방지
 - SET_NULL : null=True 설정이 되어있을 때, 삭제되면 해당 필드를 null 설정
 - SET_DEFAULT : 필드에 지정된 디폴트값으로 설정
 - SET : 값이나 함수를 지정. 함수의 경우 호출결과값을 지정
 - DO_NOTHING : 어떤 액션도 하지 않음 . DB에 따라 오류발생 가능

ForeignKey

2. Relationship

● Many-to-one 관계 예제

1

:

M

```
from django.db import models

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

    def __str__(self):
        return "%s %s" % (self.first_name, self.last_name)

class Article(models.Model):
    headline = models.CharField(max_length=100)
    pub_date = models.DateField()
    reporter = models.ForeignKey(Reporter, on_delete=models.CASCADE)

    def __str__(self):
        return self.headline

class Meta:
    ordering = ('headline',)
```

ManyToManyField

2. Relationship

● Many-to-many 관계

- django.db.models.ManyToManyField 사용

M
:
N

```
from django.db import models

class Topping(models.Model):
    # ...
    pass

class Pizza(models.Model):
    # ...
    toppings = models.ManyToManyField(Topping)
```

- M:N 관계에서 어느 쪽이라도 가능
- 참조 문서

- https://docs.djangoproject.com/ko/3.2/topics/db/examples/many_to_many

ManyToManyField

2. Relationship

● Many-to-many 관계 예제

```
from django.db import models

class Publication(models.Model):
    title = models.CharField(max_length=30)

    def __str__(self):
        return self.title

    class Meta:
        ordering = ('title',)

class Article(models.Model):
    headline = models.CharField(max_length=100)
    publications = models.ManyToManyField(Publication)

    def __str__(self):
        return self.headline

    class Meta:
        ordering = ('headline',)
```


OneToOneField

2. Relationship

● One-to-one 관계

- `django.db.models.OneToOneField` 사용
- 1:1 관계에서 어느 쪽이라도 가능
- 참조문서

- https://docs.djangoproject.com/ko/3.2/topics/db/examples/one_to_one

OneToOneField

2. Relationship

● One-to-One 관계 예제

```
from django.db import models

class Place(models.Model):
    name = models.CharField(max_length=50)
    address = models.CharField(max_length=80)

    def __str__(self):
        return "%s the place" % self.name

class Restaurant(models.Model):
    place = models.OneToOneField(
        Place,
        on_delete=models.CASCADE,
        primary_key=True,
    )
    serves_hot_dogs = models.BooleanField(default=False)
    serves_pizza = models.BooleanField(default=False)

    def __str__(self):
        return "%s the restaurant" % self.place.name

class Waiter(models.Model):
    restaurant = models.ForeignKey(Restaurant, on_delete=models.CASCADE)
    name = models.CharField(max_length=50)

    def __str__(self):
        return "%s the waiter at %s" % (self.name, self.restaurant)
```

1:N 관계

2. Relationship

● blog/models.py 추가

```
class Comment(models.Model):  
    post = models.ForeignKey(Post, on_delete=models.CASCADE)  
    author = models.CharField(max_length=20)  
    message = models.TextField()  
    created_at = models.DateField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)
```

● blog/admin.py 추가

```
admin.site.register(Comment)
```

M:M 관계

2. Relationship

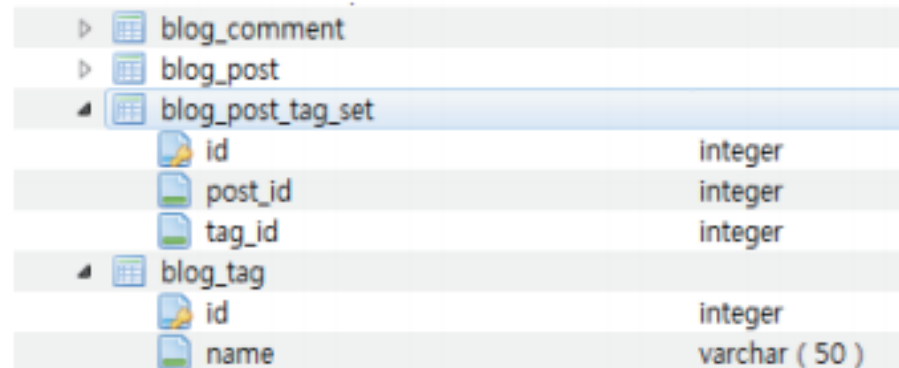
- Blog/models.py 추가

```
class Tag(models.Model):  
    name = models.CharField(max_length=50, unique=True)  
    def __str__(self):  
        return self.name
```

- blog/models.py의 Post 모델 필드 추가

```
tags = models.ManyToManyField('Tag')
```

- migrations 작업



The screenshot shows a Django ORM migration table structure. It lists three models: `blog_comment`, `blog_post`, and `blog_post_tag_set`. The `blog_post_tag_set` model is expanded, showing its fields: `id` (integer), `post_id` (integer), and `tag_id` (integer). Below it, the `blog_tag` model is also expanded, showing its fields: `id` (integer) and `name` (varchar (50)).

▶	blog_comment	
▶	blog_post	
▶	blog_post_tag_set	
	id	integer
	post_id	integer
	tag_id	integer
▶	blog_tag	
	id	integer
	name	varchar (50)

M:M 관계

2. Relationship

- blog/admin.py

```
admin.site.register(Tag)
```

- D:\src\ django\myproject> python manage.py shell

Post에서 Tag로 접근

```
>> from blog.models import Post, Comment
```

```
>> Post.objects.filter(tags__name='동해')
```

```
>> Post.objects.filter(tags__name__in=['동해', '방탄'])
```

```
>> Post.objects.filter(title='제목')
```

```
>> post = Post.objects.first()
```

```
>> post.tags.all() # M:M관계
```

```
>> Tag.objects.filter(post=post) #위와 같음
```

1:N 관계에서 N측 필드 접근

2. Relationship

● related_name

- 1:N 관계에서 N측 필드 접근 방법
 - 모델인스턴스.모델명소문자_set
- M:M관계에서 접근방법
 - 모델인스턴스.필드명
- 1:1 관계에서 접근방법
 - 모델인스턴스.모델명소문자

Post에서 Comment로 접근

```
>> from blog.models import Comment, Post, Tag
```

```
>> post = Post.objects.first()
```

```
>> post.comment_set.all() # 1:N관계
```

```
>> Comment.objects.filter(post=post) # 위와 동일
```

```
>> post.user # 1:1 관계
```

1:N 관계

2. Relationship

```
<h3>Comments</h3>
<ul>
{% for comment in post.comment_set.all %}
  <li>
    {{comment.message}}
    <small>by {{comment.author}}</small>
    <small>at {{comment.updated_at}}</small>
  </li>
{% endfor %}
</ul>
<h3>Tags</h3>
<ul>
{% if post.tags.all %}
  {% for tag in post.tags.all %}
    #{{tag.name}}
  {% endfor %}
{% else %}
  <b>등록된 태그가 없습니다</b>
{% endif %}
</ul>
```

1:1 관계

2. Relationship

- 'django.contrib.auth.User', 모델 기본 제공
- User에 대한 수정은 Profile 모델과 1:1 관계맺음
 - c:\dev\myproject>python manage.py startapp accounts
- myproject/settings.py

```
INSTALLED_APPS = [  
    'accounts',  
]
```

- myproject/urls.py

```
urlpatterns = [  
    path('accounts/',include('accounts.urls')),  
]
```

1:1 관계

2. Relationship

- accounts/models.py

```
from django.db import models
from django.contrib.auth.models import User

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    phone_number = models.CharField(max_length=20)
    address = models.CharField(max_length=50)
```

- C:\dev\myproject>python manage.py makemigrations accounts
- C:\dev\myproject>python manage.py migrate accounts

1:1 관계

2. Relationship

● accounts/admin.py

```
from django.contrib import admin
from .models import Profile

admin.site.register(Profile)
```

accounts_profile		CREATE TABLE "accounts_profile" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "phone_number" varchar (20) NOT NULL, "address" varchar (50) NOT NULL, "user_id" integer NOT NULL UNIQUE)
id	integer	`id` integer NOT NULL PRIMARY KEY AUTOINCREMENT
phone_number	varchar (20)	`phone_number` varchar (20) NOT NULL
address	varchar (50)	`address` varchar (50) NOT NULL
user_id	integer	`user_id` integer NOT NULL UNIQUE

ForeignKey와 OneToOneField

2. Relationship

- 생성되는 필드명은 같으나 유일성이 다름
- `user = models.ForeignKey(User)`
"user_id" integer NOT NULL REFERENCES "auth_user" ("id")
- `user = models.OneToOneField(User)`
"user_id" integer NOT NULL UNIQUE REFERENCES "auth_user" ("id")

ForeignKey와 OneToOneField

2. Relationship

ForeignKey → shell에서 작업

```
>> from blog.models import Comment
>> comment = Comment.objects.first()
>> from django.db import connection
>> connection.queries[-1]

{'sql': 'SELECT "blog_comment"."id", "blog_comment"."post_id", "blog_comment"."author",
        "blog_comment"."message", "blog_comment"."created_at", "blog_comment"."updated_at"
        FROM "blog_comment" ORDER BY "blog_comment"."id" ASC LIMIT 1',
'time': '0.001'}

>> comment.post_id
>> comment.post
>> connection.queries[-1]

{'sql': 'SELECT "blog_post"."id", "blog_post"."title", "blog_post"."author",
        "blog_post"."content", "blog_post"."tags", "blog_post"."region",
        "blog_post"."lnglat", "blog_post"."created_at", "blog_post"."updated_at"
        FROM "blog_post" WHERE "blog_post"."id" = 2',
'time': '0.001'}
```

ForeignKey와 OneToOneField

2. Relationship

OneToOneField → shell에서 작업

```
>> from accounts.models import Profile
>> profile = Profile.objects.first()
>> connection.queries[-1]
```

```
{'sql': 'SELECT "accounts_profile"."id", "accounts_profile"."user_id",
        "accounts_profile"."phone_number", "accounts_profile"."address"
        FROM "accounts_profile" ORDER BY "accounts_profile"."id" ASC LIMIT 1',
'time': '0.000'}
```

```
>> profile.user_id
>> profile.user
>> connection.queries[-1]
```

```
{'sql': 'SELECT "auth_user"."id", "auth_user"."password", "auth_user"."last_login",
        "auth_user"."is_superuser", "auth_user"."username", "auth_user"."first_name",
        "auth_user"."last_name", "auth_user"."email", "auth_user"."is_staff",
        "auth_user"."is_active",
        "auth_user"."date_joined" FROM "auth_user" WHERE "auth_user"."id" = 1',
'time': '0.000'}
```

auth.User 모델

2. Relationship

```
from django.conf import settings
from django.contrib.auth.models import User
```

- 방법1) 비추천

- `user = models.OneToOneField(User)`

- 방법2) 비추천

- `user = models.OneToOneField('auth.User')`

- 방법3) 추천

- `user = models.OneToOneField(settings.AUTH_USER_MODEL)`



auth.User 모델

2. Relationship

- accounts/models.py 수정

```
from django.conf import settings

class Profile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL,
                                on_delete=models.CASCADE)
```

1:N 관계에서 N측 필드 접근

2. Relationship

- related_name
 - 1:1 관계에서 접근방법
 - 모델인스턴스.필드명(보통 모델소문자와 동일하게 한다)

```
class Profile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL,
                                on_delete=models.CASCADE)
    phone_number = models.CharField(max_length=20)
    address = models.CharField(max_length=50)
```

```
>>profile = Profile.objects.first()
>>profile.user

>>user = User.objects.first()
>>user.profile
```

1:N 관계에서 N측 필드 접근

2. Relationship

● related_name

- 1:N 관계에서 N측 필드 접근 방법
 - 모델인스턴스.모델명소문자_set
- M:M관계에서 접근방법
 - 모델인스턴스.필드명(보통 모델명 소문자와 동일하게 한다)
- 1:1 관계에서 접근방법
 - 모델인스턴스.필드명(보통 모델명소문자와 동일하게 한다)

Profile에서 User로 접근

```
>> from account.models import Profile
```

```
>> from django.contrib.auth.models import User
```

```
>> pf = Profile.objects.first()
```

```
>> pf.user # 1:1 관계 User인스턴스 객체 출력됨
```

```
>> pf.user.username
```


1:N 관계에서 N측 필드 접근

2. Relationship

`blog/models.py` 수정

```
from django.conf import settings
```

```
class Post(models.Model):
```

```
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
```

```
    # author = models.CharField(max_length=20, default='anonymous')
```

```
C:\dev\myproject>python manage.py makemigrations blog
```

```
user_id값 1 입력
```

```
C:\dev\myproject>python manage.py migrate blog
```

```
>>from django.contrib.auth.models import User
```

```
>>user = User.objects.first()
```

```
>>user.post_set.all()
```

1:N 관계에서 N측 필드 접근

2. Relationship

● shop 앱 생성

```
c:\dev\myproject>python manage.py startapp shop
```

```
# myproject/settings.py 앱 등록
```

```
INSTALLED_APPS = [  
    'shop',  
]
```

```
# shop/urls.py 작성
```

```
from django.urls import path  
app_name = 'shop'  
urlpatterns = [ ]
```

```
# myproject/urls.py
```

```
urlpatterns = [  
    path('shop/',include('shop.urls')),  
]
```

1:N 관계에서 N측 필드 접근

2. Relationship

● shop 앱 모델 작성 및 마이그레이션

```
# shop/models.py
from django.conf import settings
from django.db import models

class Post(models.Model):
    user=models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
C:\dev\myproject>python manage.py makemigrations shop
```

오류 발생!

1:N 관계에서 N측 필드 접근

2. Relationship

- `related_name` 속성 지정 (이 경우 반드시 지정 필요)

`blog/models.py` 수정

```
user = models.ForeignKey(settings.AUTH_USER_MODEL,  
                        on_delete=models.CASCADE,  
                        related_name='blog_post_set')
```

`shop/models.py` 수정

```
user = models.ForeignKey(settings.AUTH_USER_MODEL,  
                        on_delete=models.CASCADE,  
                        related_name='shop_post_set')
```

```
C:\dev\myproject>python manage.py makemigrations shop  
Migrations for 'shop':  
  shop\migrations\0001_initial.py  
    - Create model Post
```

1:N 관계에서 N측 필드 접근

2. Relationship

shell에서 작업

```
>> from django.contrib.auth.models import User
```

```
>> user = User.objects.first()
```

```
>> user.post_set.all() ➔ 오류 발생
```

```
>> user.blog_post_set.all()
```

```
>> user.shop_post_set.all()
```

1:N 관계에서 N측 필드 접근

2. Relationship

- `related_name = '+'`

- `Related_name`을 사용하지 않겠다는 의미
- `Post.objects.filter(user=user)` 형태로 접근해서 사용

```
# shop/models.py
user = models.ForeignKey(settings.AUTH_USER_MODEL,
                          on_delete=models.CASCADE,
                          related_name='+')
```

```
>> from django.contrib.auth.models import User
>> user = User.objects.first()
>> user.post_set.all() -> 오류 발생
>> from shop.models import Post
>> Post.objects.filter(user=user)
```



3. Migrations



Migrations

3. Migrations

- <https://docs.djangoproject.com/ko/2.1/topics/migrations/>
- model의 변경 내용을 데이터베이스 스키마로 반영
- model의 변경 내역에 대한 히스토리 관리
- 마이그레이션 파일 생성
 - `python manage.py makemigrations <앱이름>`
- 마이그레이션 DB 적용
 - `python manage.py migrate <앱이름>`
- 마이그레이션 적용 현황
 - `python manage.py showmigrations <앱이름>`
- 지정 마이그레이션의 SQL문 확인
 - `python manage.py sqlmigrate 앱이름 마이그레이션이름`


```
D:\src\myproject>python manage.py showmigrations
```

```
D:\src\myproject>python manage.py showmigrations shop
```

shop/models.py 내용 추가

```
name = models.CharField(max_length=100)
```

```
D:\src\myproject>python manage.py makemigrations shop
```

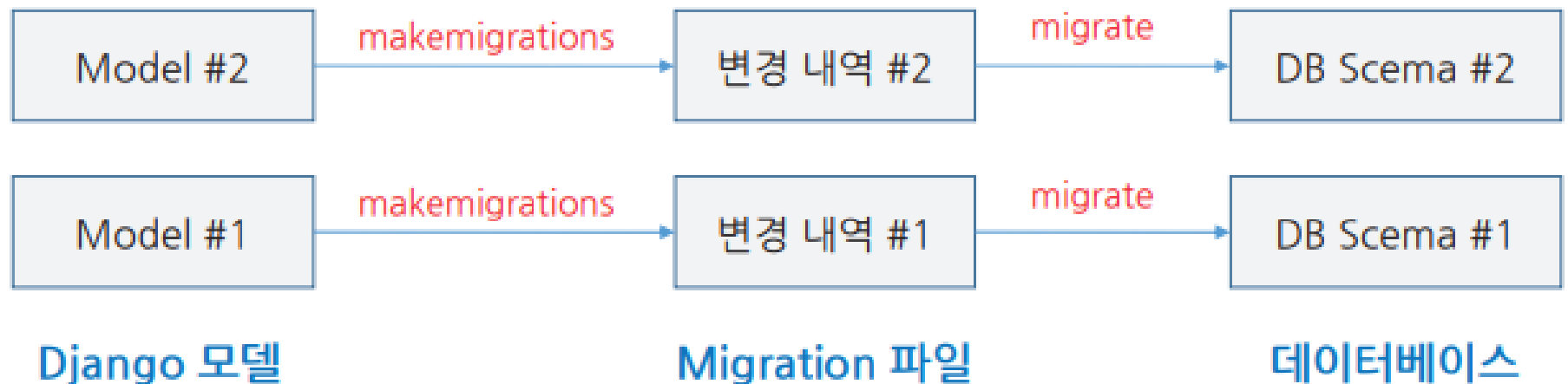
```
D:\src\myproject>python manage.py showmigrations shop # 반영확인
```

```
D:\src\myproject>python manage.py sqlmigrate shop 0003
```

Migrations

3. Migrations

- `makemigrations` : 마이그레이션 파일 생성
- `migrate` : 마이그레이션 파일 내용을 DB에 적용하기



```
# shop/models.py의 name 필드 이름을 title로 수정
title = models.CharField(max_length=100)
```

```
D:\src\myproject>python manage.py makemigrations shop
Did you rename post.name to post.title (a CharField)? [y/N] y
Migrations for 'shop':
  shop\migrations\0004_auto_20180831_1118.py
    -Rename field name on post to title
```

migrate 파일 삭제

```
D:\src\myproject>python manage.py makemigrations shop
Did you rename post.name to post.title (a CharField)? [y/N] n
~ 생략 ~
Migrations for 'shop':  shop\migrations\0004_auto_20180
831_1123.py
    -Remove field name from post
    -Add field title to post
```

Migrate

3. Migrations

- `python manage.py migrate <app-name>`
 - 순차적으로 모든 마이그레이션 파일 수행
- `python manage.py migrate <app-name> <migrate-file>`
 - 정방향 : <migrate-file>이 현재 마이그레이션보다 이후이면 순차적으로 진행
 - 역방향 : <migrate-file>이 현재 마이그레이션보다 이전이면 롤백 진행
- 마이그레이션 파일명 지정
 - 파일명 전체 또는 일부만 지정 가능

실습(1)

3. Migrations

```
D:\src\myproject>python manage.py showmigrations shop
shop
```

```
[X] 0001_initial
[X] 0002_auto_20180830_2114
[ ] 0003_post_name
[ ] 0004_auto_20180831_1123
```

```
D:\src\myproject>python manage.py migrate shop
```

```
Operations to perform:
```

```
Apply all migrations: shop Running
```

```
migrations:
```

```
Applying shop.0003_post_name... OK Applyi
ng shop.0004_auto_20180831_1136...
```

```
D:\src\myproject>python manage.py showmigrations shop
```

```
[X] 0001_initial
[X] 0002_auto_20180830_2114
[X] 0003_post_name
[X] 0004_auto_20180831_1136
```

실습(2)

3. Migrations

```
D:\src\myproject>python manage.py migrate shop 0003
```

```
Operations to perform:
```

```
Target specific migration: 0003_post_nameRunning migrations: Rendering model states... DONE
```

```
Unapplying shop.0004_auto_20180831_1136.
```

```
D:\src\myproject>python manage.py showmigrations shop
```

```
shop
```

```
[X] 0001_initial
```

```
[X] 0002_auto_20180830_2114
```

```
[X] 0003_post_name
```

```
[ ] 0004_auto_20180831_1136
```

```
D:\src\myproject>python manage.py migrate shop zero
```

```
Operations to perform:
```

```
Unapply all migrations: shop Running migrations:
```

```
Rendering model states... DONE Unapplying shop.0003_post_name... OK
```

```
Unapplying shop.0002_auto_20180830_2114... OK Unapplying shop.0001_initial... OK
```

실습(3)

3. Migrations

```
D:\src\myproject>python manage.py showmigrations shop
shop
[ ] 0001_initial
[ ] 0002_auto_20180830_2114
[ ] 0003_post_name
[ ] 0004_auto_20180831_1136
```