■ 모델

- 모델은 app/models.py 내의 파이썬 클래스로 표현
- django.db.models.Model 클래스를 상속 받아서 구현
  - 코드 구성

```
from django.db import models

class 모델클래스(models.Model):
    속성1 = models.CharField(max_length=30)
    속성2 = models.IntegerField()
```

- 모델 클래스를 프로젝트에 반영하기 위해 app 추가
  - settings.py

```
INSTALLED_APPS = [
    '추가할 APP',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

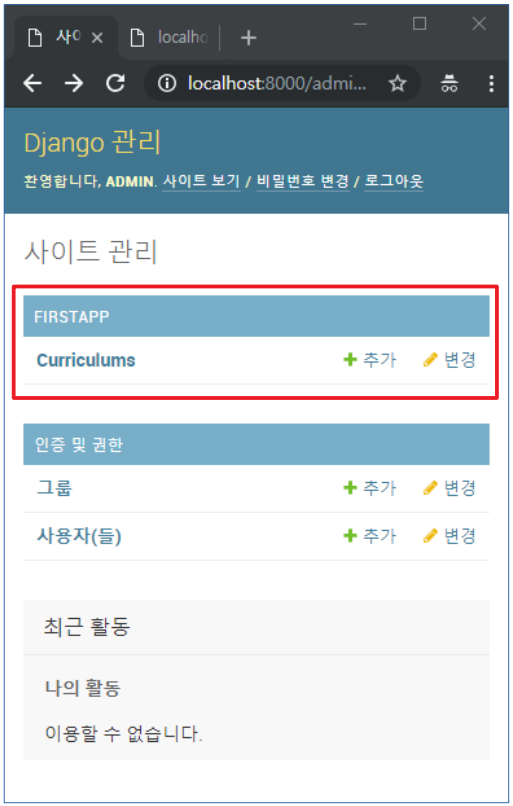
■ 모델

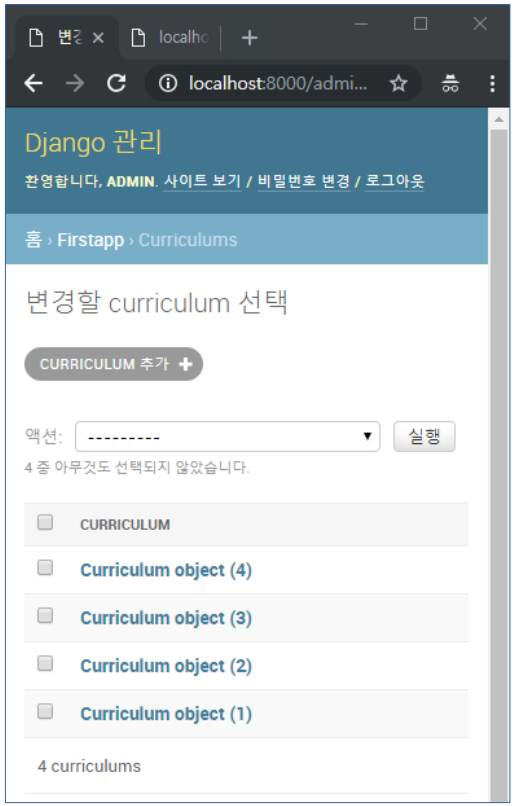
● 관리자 사이트에서 데이터 제어를 위해 모델 클래스 등록

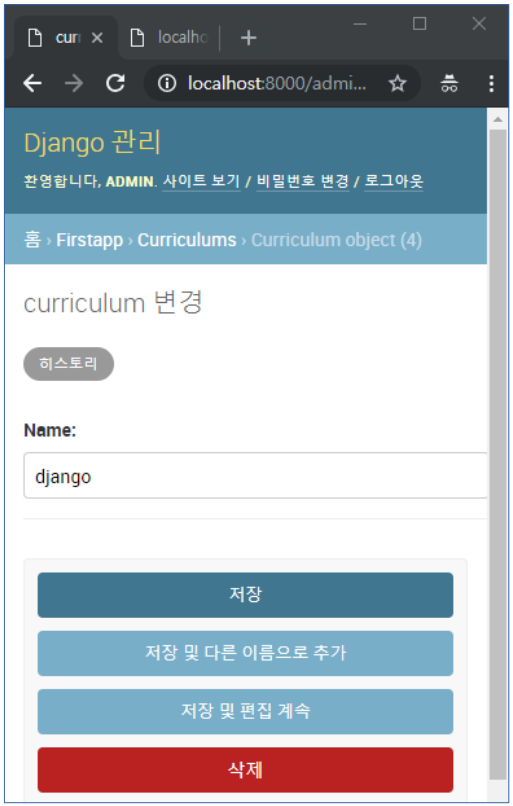
- app/admin.py

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

admin.site.register(모델클래스)
```







■ 모델

● 모델 클래스 속성

– Field Type

```
from django.db import models

class 모델클래스(models.Model):
    속성1 = models.CharField(max_length=30)
    속성2 = models.IntegerField()
```

타입	설명
CharField	제한된 문자열 타입, max_length 옵션으로 최대 입력 길이 지정
IntegerField	정수 타입
FloatField	실수 타입
DateTimeField	날짜 / 시간 타입 (파이썬의 datetime.datetime)
BooleanField	True / False 타입
Text, FilePath, Email, Image, URL, …	

■ 모델

● 모델 클래스 속성

– Field Option

```
class 모델클래스(models.Model):
    속성1 = models.CharField(max_length=30, null=True)
    속성2 = models.IntegerField(default=0)
```

옵션	설명
null (Field.null)	null 허용
blank (Field.blank)	빈 값 허용
primary_key (Field.primary_key)	기본 키
unique (Field.unique)	unique index 생성
default (Field.default)	기본 값 지정
db_column (Field.db_column)	컬럼명 임의 지정
db_index, db_tablespace, help_text, verbose_name, validators, …	

■ 모델

● Manager 속성

- 모든 모델은 매니저 속성을 가져야 하며,

  명시적으로 매니저 속성을 정의하지 않았다면 objects 라는 기본값을 가짐

  ex) 모든 데이터 조회 → 모델클래스.objects.all()

- 매니저 속성 정의

```
from django.db import models

# 매니저 정의
class SecondManager(models.Manager):
    def get_queryset(self):
        return super(SecondManager, self).get_queryset().filter(name__contains='kim')

class Curriculum(models.Model):
    name = models.CharField(max_length=255)

    objects = models.Manager()
    second_objects = SecondManager()

    def __str__(self):
        return self.name
```

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

– python manage.py shell

```
C:₩Users₩GGoReb₩work_django₩tutorial>python manage.py shell
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

– 데이터 전체 조회 : 모델클래스.objects.all()

```
In [1]: from firstapp.models import Curriculum

In [2]: Curriculum.objects.all()
Out[2]: <QuerySet [<Curriculum: Curriculum object (1)>, <Curriculum: Curriculum object (2)>,
<Curriculum: Curriculum object (3)>, <Curriculum: Curriculum object (4)>, <Curriculum: Curric
ulum object (5)>]>

In [3]:
```

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

- 데이터 조회 : 모델클래스.objects.get(속성=검색어)

```
In [3]: Curriculum.objects.get(name='python')
Out[3]: <Curriculum: Curriculum object (2)>

In [4]: Curriculum.objects.get(id=3)
Out[4]: <Curriculum: Curriculum object (3)>

In [5]:
```

- 데이터 조회 : 모델클래스.objects.filter(속성=검색어)

```
In [11]: Curriculum.objects.filter(name__contains='chain')
Out[11]: <QuerySet [<Curriculum: Curriculum object (5)>]>

In [12]:
```

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

- 데이터 제외 : 모델클래스.objects.exclude(속성=검색어)

```
In [9]: Curriculum.objects.exclude(name='python')
Out[9]: <QuerySet [<Curriculum: Curriculum object (1)>, <Curriculum: Curriculum object (3)>,
<Curriculum: Curriculum object (4)>, <Curriculum: Curriculum object (5)>]>

In [10]: Curriculum.objects.exclude(pk=1)
Out[10]: <QuerySet [<Curriculum: Curriculum object (2)>, <Curriculum: Curriculum object (3)>,
 <Curriculum: Curriculum object (4)>, <Curriculum: Curriculum object (5)>]>
```

	id	name
	필터	필터
1	1	linux
2	2	python
3	3	html/css/js
4	4	django
5	5	block chain

- 데이터 개수 : 모델클래스.objects.count()

```
In [12]: Curriculum.objects.count()
Out[12]: 5
```

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

– 오름차순 정렬 : 모델클래스.objects.order_by(속성)

```
In [5]: Curriculum.objects.order_by('id')
Out[5]: <QuerySet [<Curriculum: Curriculum object (1)>, <Curriculum: Curriculum object (2)>,
<Curriculum: Curriculum object (3)>, <Curriculum: Curriculum object (4)>, <Curriculum: Curric
ulum object (5)>]>

In [6]: Curriculum.objects.order_by('-id')
Out[6]: <QuerySet [<Curriculum: Curriculum object (5)>, <Curriculum: Curriculum object (4)>,
<Curriculum: Curriculum object (3)>, <Curriculum: Curriculum object (2)>, <Curriculum: Curric
ulum object (1)>]>
```

	id	name
	필터	필터
1	1	linux
2	2	python
3	3	html/css/js
4	4	django
5	5	block chain

– 내림차순 정렬 : 모델클래스.objects.order_by(-속성)

```
In [7]: Curriculum.objects.order_by('name')
Out[7]: <QuerySet [<Curriculum: Curriculum object (5)>, <Curriculum: Curriculum object (4)>,
<Curriculum: Curriculum object (3)>, <Curriculum: Curriculum object (1)>, <Curriculum: Curric
ulum object (2)>]>

In [8]: Curriculum.objects.order_by('-name')
Out[8]: <QuerySet [<Curriculum: Curriculum object (2)>, <Curriculum: Curriculum object (1)>,
<Curriculum: Curriculum object (3)>, <Curriculum: Curriculum object (4)>, <Curriculum: Curric
ulum object (5)>]>
```

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

- 처음 데이터 조회 : 모델클래스.objects.order_by(속성).first()

```
In [18]: Curriculum.objects.order_by('-name').first()
Out[18]: <Curriculum: Curriculum object (2)>
```

- 마지막 데이터 조회 : 모델클래스.objects.order_by(속성).last()

```
In [19]: Curriculum.objects.order_by('id').last()
Out[19]: <Curriculum: Curriculum object (5)>
```

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

– 데이터 추가 : 모델클래스.objects.create(속성=값)

```
In [9]: Curriculum.objects.create(name='java')
Out[9]: <Curriculum: Curriculum object (6)>

In [10]: Curriculum.objects.create(name='sqlite')
Out[10]: <Curriculum: Curriculum object (7)>

In [11]:
```

	id	name
	필터	필터
1	8	pandas
2	7	sqlite
3	6	java
4	5	block chain
5	4	django
6	3	html/css/js
7	2	python
8	1	linux

– 데이터 추가 : 모델클래스 생성 후 save()

```
In [12]: c = Curriculum(name='pandas')

In [13]: c.save()

In [14]:
```

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

- 데이터 수정 : 데이터 조회 후 속성 값 변경

```
In [14]: data = Curriculum.objects.get(id=8)

In [15]: data.name = 'numpy'

In [16]: data.save()

In [17]:
```

데이터베이스 구조 | 데이터 보기 | Pra

테이블: firstapp_curriculu

	id	name
	필터	필터
1	8	numpy
2	7	sqlite
3	6	java
4	5	block chain
5	4	django
6	3	html/css/js
7	2	python
8	1	linux

■ 모델

● 장고 쉘을 이용한 모델 함수 테스트

- 데이터 삭제 : 데이터 조회 후 삭제

```
In [28]: data.delete()
Out[28]: (1, {'firstapp.Curriculum': 1})

In [29]:
```

데이터베이스 구조 | 데이터 보기 | Pra

테이블: firstapp_curriculu

	id	name
	필터	필터
1	8	numpy
2	7	sqlite
3	5	block chain
4	4	django
5	3	html/css/js
6	2	python
7	1	linux

■ 모델 관계

● 테이블 간에는 관계를 맺을 수 있으며, 3가지로 분류해 제공

- 1:N (one-to-many) / N:N (many-to-many) / 1:1 (one-to-one)

● 관계는 양쪽 모델에서 정의가 필요한게 원칙이지만

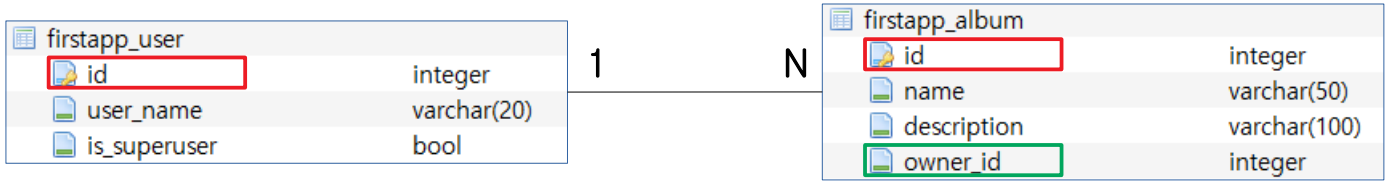
한쪽 모델(클래스)에서만 관계를 정의해도 상대쪽 정의는 자동으로 처리

- 상대쪽 정의가 명시적으로 보이지 않아도 이해 할 수 있도록 학습

■ 모델 관계

● 1:N (one-to-many)

- ForeignKey 속성 이용
- 1:N 중 N의 방향 모델에 적용




```
class User(models.Model):
    user_name = models.CharField(max_length=20)
    is_superuser = models.BooleanField(default=False)

    def __str__(self):
        return self.user_name
```

```
class Album(models.Model):
    name = models.CharField(max_length=50)
    description = models.CharField(
        'One Line Description',
        max_length=100, null=True)
    owner = models.ForeignKey(User,
                              null=True, on_delete=models.CASCADE)

    def __str__(self):
        return self.name + '[' + self.owner.user_name + ']'
```

■ 모델 관계

● 1:N (one-to-many)

- 모델 import

```
from firstapp.models import User, Album
```

- User 데이터 입력

```
User.objects.create(user_name='a')
User.objects.create(user_name='b')
User.objects.create(user_name='c')
```

- Album 데이터 입력

```
user_a = User.objects.get(user_name='a')
Album.objects.create(name='a_Album1', description='앨범', owner=user_a)
Album.objects.create(name='a_Album2', description='앨범', owner=user_a)

user_b = User.objects.get(user_name='b')
Album.objects.create(name='b_Album1', description='앨범', owner=user_b)
Album.objects.create(name='b_Album2', description='앨범', owner=user_b)

user_c = User.objects.get(user_name='c')
Album.objects.create(name='c_Album1', description='앨범', owner=user_c)
Album.objects.create(name='c_Album2', description='앨범', owner=user_c)
```

■ 모델 관계

● 1:N (one-to-many)

- User 및 Album 데이터 입력 결과

id	user_name	is_superuser
필터	필터	필터
3	c	0
2	b	0
1	a	0

id	name	description	owner_id
필터	필터	필터	필터
6	c_Album2	앨범	3
5	c_Album1	앨범	3
4	b_Album2	앨범	2
3	b_Album1	앨범	2
2	a_Album2	앨범	1
1	a_Album1	앨범	1

- User의 모든 레코드 확인

```
User.objects.all()
→ <QuerySet [<User: a>, <User: b>, <User: c>]>
```

■ 모델 관계

● 1:N (one-to-many)

- Album의 모든 레코드 확인

```
Album.objects.all()
→ <QuerySet [<Album: a_Album1[a]>, <Album: a_Album2[a]>, <Album: b_Album1[b]>, <Album: b_Album2[b]>, <Album: c_Album1[c]>, <Album: c_Album2[c]>]>
```

- Album의 레코드 1개 조회 후 소유자(User) 확인

```
a1 = Album.objects.all()[1]
a1
→ <Album: a_Album2[a]>
a1.owner
→ <User: a>
```

- Album 객체를 이용하여 User 객체 접근 후 이름 확인

```
a1.owner.user_name
→ 'a'
```

■ 모델 관계

● 1:N (one-to-many)

- Album 객체 생성 : 앨범 생성 후 소유자 지정

```
album_c3 = Album(name='c_Album3', description='추가앨범')
user_c = User.objects.get(user_name='c')
album_c3.owner = user_c
album_c3.save()
album_c3.owner.user_name
→ 'c'
```

id	name	description	owner_id
필터	필터	필터	필터
7	c_Album3	추가앨범	3
6	c_Album2	앨범	3
5	c_Album1	앨범	3
4	b_Album2	앨범	2
3	b_Album1	앨범	2
2	a_Album2	앨범	1
1	a_Album1	앨범	1

■ 모델 관계

● 1:N (one-to-many)

- Album 객체 생성 : 소유자 지정 후 앨범 추가

```
user_c = User.objects.get(user_name='c')
album_c4 = user_c.album_set.create(name='c_Album4', description='추가앨범')
album_c4
→ <Album: c_Album4[c]>
```

id	name	description	owner_id
필터	필터	필터	필터
8	c_Album4	추가앨범	3
7	c_Album3	추가앨범	3
6	c_Album2	앨범	3
5	c_Album1	앨범	3
4	b_Album2	앨범	2
3	b_Album1	앨범	2
2	a_Album2	앨범	1
1	a_Album1	앨범	1

■ 모델 관계

● 1:N (one-to-many)

- username이 'c'인 User의 모든 Album 조회

```
user_c = User.objects.get(user_name='c')
user_c.album_set.all()
→ <QuerySet [<Album: c_Album1[c]>, <Album: c_Album2[c]>, <Album: c_Album3[c]>, <Album: c_Album4[c]>]>
user_c.album_set.count()
→ 4
```

- 모델 간 검색 (User → Album / Album → User)

```
user_c.album_set.filter(name__startswith='c_')
→ <QuerySet [<Album: c_Album1[c]>, <Album: c_Album2[c]>, <Album: c_Album3[c]>, <Album: c_Album4[c]>]>
user_c.album_set.filter(name__endswith='4')
→ 4
Album.objects.filter(owner__user_name='a')
→ <QuerySet [<Album: a_Album1[a]>, <Album: a_Album2[a]>]>
Album.objects.filter(owner__user_name='a', owner__is_superuser=False)
→ <QuerySet [<Album: a_Album1[a]>, <Album: a_Album2[a]>]>
```

■ 모델 관계

● 1:N (one-to-many)

- Album의 name에 3 또는 4가 포함되어 있는 레코드 조회

```
from django.db.models import Q
Album.objects.filter(Q(name__contains='3') | Q(name__contains='4'))
→ <QuerySet [<Album: c_Album3[c]>, <Album: c_Album4[c]>]>
```

- Album의 name에 3 또는 4가 포함되어 있고 owner가 3인 레코드 조회

```
Album.objects.filter(Q(name__contains='3') | Q(name__contains='4'), owner=3)
→ <QuerySet [<Album: c_Album3[c]>, <Album: c_Album4[c]>]>
```

- Album의 name에 3 또는 4가 포함되어 있고 owner가 3인 레코드 삭제

```
Album.objects.filter(
    Q(name__contains='3') | Q(name__contains='4'
), owner=3).delete()
→ <QuerySet [<Album: c_Album3[c]>, <Album: c_Album4[c]>]>
```

■ 모델 관계

● 1:N (one-to-many)

- 1:N 관계 중 1 방향의 객체(User) 삭제 → N 방향 객체 같이 삭제 (cascade)

```
user_b = User.objects.get(user_name='b')
user_b.delete()

User.objects.all()
→ <QuerySet [<User: a>, <User: c>]>

Album.objects.all()
→ <QuerySet [<Album: a_Album1[a]>, <Album: a_Album2[a]>, <Album:
c_Album1[c]>, <Album: c_Album2[c]>]>
```

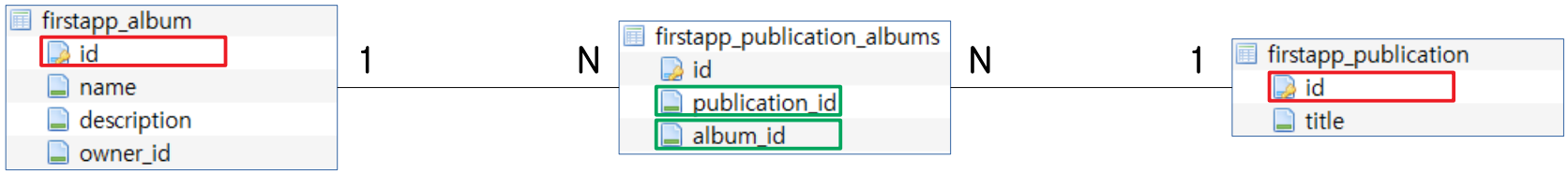
id	user_name	is_superuser
필터	필터	필터
1	a	0
3	c	0

id	name	description	owner_id
필터	필터	필터	필터
6	c_Album2	앨범	3
5	c_Album1	앨범	3
2	a_Album2	앨범	1
1	a_Album1	앨범	1

■ 모델 관계

● N:N (many-to-many)

- ManyToManyField 속성 이용
- N:N 중 어느 방향이든 한 방향 모델에 적용




```
class Album(models.Model):
    name = models.CharField(max_length=50)
    description = models.CharField(
        'One Line Description',
        max_length=100, null=True)
    owner = models.ForeignKey(User,
                              null=True, on_delete=models.CASCADE)

    def __str__(self):
        return self.name + '[' + self.owner.user_name + ']'
```

```
class Publication(models.Model):
    title = models.CharField(max_length=30)
    albums = models.ManyToManyField(Album)

    def __str__(self):
        return self.title
```

■ 모델 관계

● N:N (many-to-many)

- 모델 import

```
from firstapp.models import Album, Publication
```

- Publication 데이터 입력

```
p1 = Publication(title='The Python Journal')
p2 = Publication(title='Science News')
p3 = Publication(title='Science Weekly')
p1.save()
p2.save()
p3.save()

Publication.objects.all()
→ <QuerySet [<Publication: The Python Journal>, <Publication: Science
News>, <Publication: Science Weekly>]>
```

Publication

id	title
필터	필터
1	The Python Journal
2	Science News
3	Science Weekly

Album

id	name	description	owner_id
필터	필터	필터	필터
6	c_Album2	앨범	3
5	c_Album1	앨범	3
2	a_Album2	앨범	1
1	a_Album1	앨범	1

■ 모델 관계

● N:N (many-to-many)

Publication

id	title
필터	필터
1	The Python Journal
2	Science News
3	Science Weekly

Publication_Album

id	publication_id	album_id
필터	필터	필터
1	1	1

Album

id	name	description	owner_id
필터	필터	필터	필터
6	c_Album2	앨범	3
5	c_Album1	앨범	3
2	a_Album2	앨범	1
1	a_Album1	앨범	1

- Album 객체와 Publication 객체 조회 후 연결

```
album1 = Album.objects.get(id=1)
pub1 = Publication.objects.get(id=1)
pub1.albums.add(album1)
```

■ 모델 관계

● N:N (many-to-many)

Publication

id	title
필터	필터
1	The Python Journal
2	Science News
3	Science Weekly

Publication_Album

id	publication_id	album_id
필터	필터	필터
2	1	2
1	1	1

Album

id	name	description	owner_id
필터	필터	필터	필터
6	c_Album2	앨범	3
5	c_Album1	앨범	3
2	a_Album2	앨범	1
1	a_Album1	앨범	1

- Album 객체와 Publication 객체 조회 후 연결

```
album2 = Album.objects.get(id=2)
pub1 = Publication.objects.get(id=1)
album2.publication_set.add(pub1)
```

■ 모델 관계

● N:N (many-to-many)

Publication

id	title
필터	필터
1	The Python Journal
2	Science News
3	Science Weekly

Publication_Album

id	publication_id	album_id
필터	필터	필터
2	1	2
1	1	1

Album

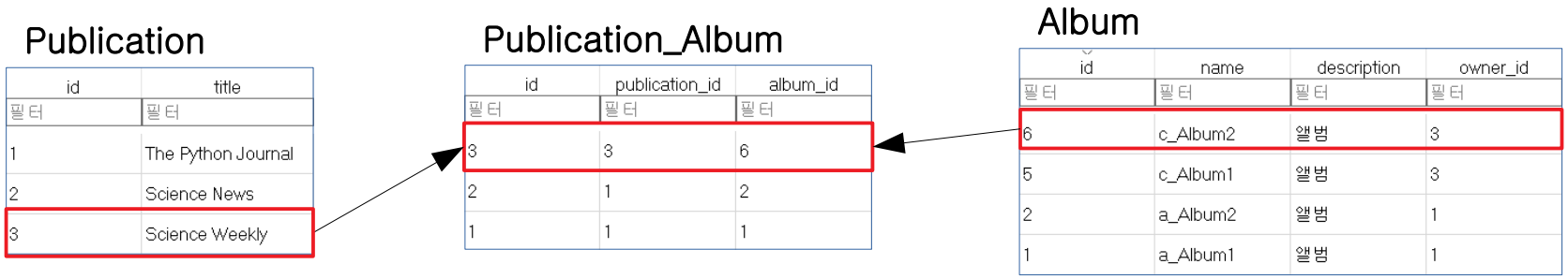
id	name	description	owner_id
필터	필터	필터	필터
6	c_Album2	앨범	3
5	c_Album1	앨범	3
2	a_Album2	앨범	1
1	a_Album1	앨범	1

- 모델 간 검색 (Publication → Album / Album → Publication)

```
Publication.objects.filter(albums=album1)
→ <QuerySet [<Publication: The Python Journal>]>
Publication.objects.filter(albums__pk=1)
→ <QuerySet [<Publication: The Python Journal>]>
Publication.objects.filter(albums__id=2)
→ <QuerySet [<Publication: The Python Journal>]>
Album.objects.filter(publication=pub1)
→ <QuerySet [<Album: a_Album1[a]>, <Album: a_Album2[a]>]>
Album.objects.filter(publication=1)
→ <QuerySet [<Album: a_Album1[a]>, <Album: a_Album2[a]>]>
```

■ 모델 관계

● N:N (many-to-many)



- Album C_2 ↔ Publication 3 연결

```
album_c2 = Album.objects.get(id=6)

pub3 = Publication.objects.get(id=3)

pub3.albums.add(album_c2)

pub3.albums.all()
→ <QuerySet [<Album: c_Album2[c]>]>
```

■ 모델 관계

● N:N (many-to-many)

Publication

id	title
필터	필터
1	The Python Journal
2	Science News
3	Science Weekly

Publication_Album

id	publication_id	album_id
필터	필터	필터
2	1	2
1	1	1

Album

id	name	description	owner_id
필터	필터	필터	필터
5	c_Album1	앨범	3
2	a_Album2	앨범	1
1	a_Album1	앨범	1

- Album C_2 삭제

```
album_c2 = Album.objects.get(id=6)
album_c2.delete()
→ (2, {'firstapp.Publication_albums': 1, 'firstapp.Album': 1})
pub3.albums.all()
→ <QuerySet []>
```

■ 모델 관계

● 1:1 (one-to-one)

- OneToOneField 속성 이용
- 1:1 중 어느 방향이든 한 방향 모델에 적용


```
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)
    name = models.CharField(max_length=50, default='DefRestName')
    serves_pizza = models.BooleanField(default=False)

    def __str__(self):
        return '%s the restaurant' % self.name
```

■ 모델 관계

● 1:1 (one-to-one)

- 모델 import

```
from firstapp.models import Place, Restaurant
```

- Place 데이터 입력

```
Place.objects.create(name='Place1', address='Seoul')
Place.objects.create(name='Place2', address='Jeju')
Place.objects.create(name-'Place3', address='Busan')
```

- Restaurant 데이터 입력

```
p1 = Place.objects.get(address='Seoul')
Restaurant.objects.create(place=p1, name='서울식당')

p2 = Place.objects.get(address='Jeju')
Restaurant.objects.create(place=p2, name='제주식당')

p3 = Place.objects.get(address='Busan')
Restaurant.objects.create(place=p3, name='부산식당')
```

■ 모델 관계

● 1:1 (one-to-one)

- Place 및 Restaurant 데이터 입력 결과

id	name	address
필터	필터	필터
1	Place1	Seoul
2	Place2	Jeju
3	Place3	Busan

id	name	serves_pizza	place_id
필터	필터	필터	필터
1	서울식당	0	1
2	제주식당	0	2
3	부산식당	0	3

- 주소가 'Seoul'인 Place 조회

```
p1 = Place.objects.get(address='Seoul')
p1
→ <Place: Place1 the place>
```

- id가 1인 Restaurant 조회

```
r1 = Restaurant.objects.get(id=1)
r1
→ <Restaurant: 서울식당 the restaurant>
```

■ 모델 관계

● 1:1 (one-to-one)

- 모델 간 검색 (Restaurant → Place / Place → Restaurant)

```
r1.place
→ <Place: Place1 the place>
p1.restaurant
→ <Restaurant: 서울식당 the restaurant>
Restaurant.objects.get(place__name__contains='Place1')
→ <Restaurant: 서울식당 the restaurant>
Restaurant.objects.get(place__address__startswith='Seo')
→ <Restaurant: 서울식당 the restaurant>
```