190529

project : modelrelation

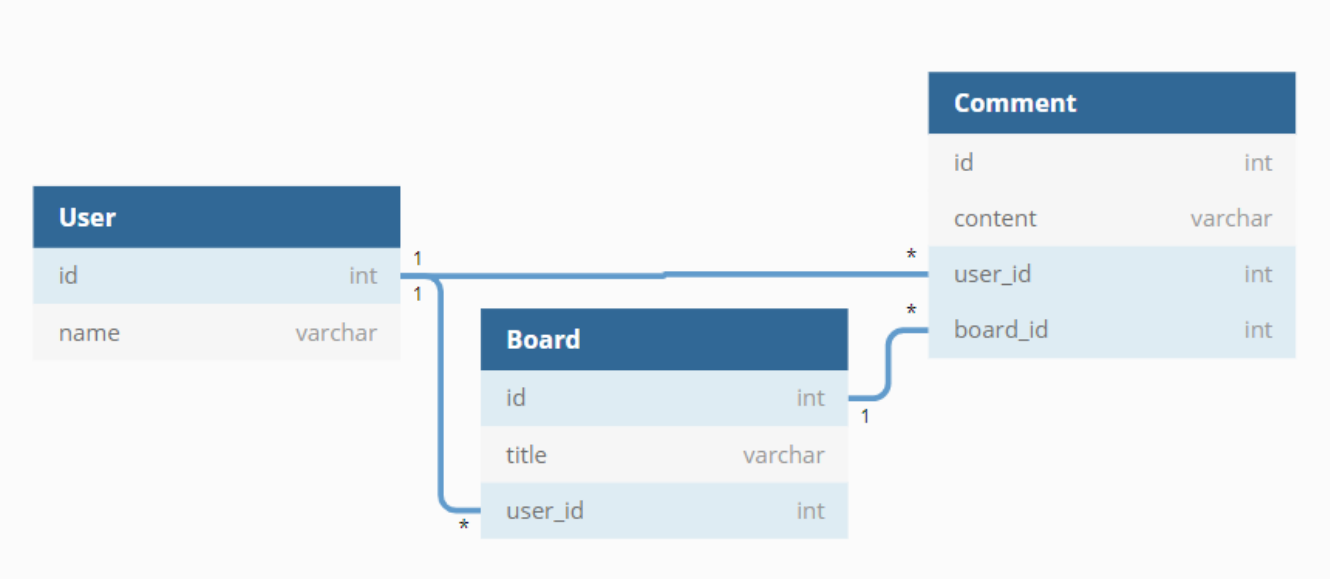
[1:n 관계]

app : onetomany

1. model 생성

[models.py]

```
from django.db import models

# Create your models here.
class User(models.Model):
    name = models.CharField(max_length=20)

    def __str__(self):
        return f'{self.name}'

class Board(models.Model):
    title = models.CharField(max_length=20)
    user = models.ForeignKey(User, on_delete=models.CASCADE) # 해당 유저가 지워지면, 모든 게시글들
이 삭제됨

    def __str__(self):
        return f'{self.title}'

class Comment(models.Model):
    content = models.CharField(max_length=20)
    board = models.ForeignKey(Board, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE) # 해당 유저가 지워지면, 모든 게시글들
이 삭제됨

    def __str__(self):
        return f'{self.content}'
```



퀴즈

특정 유저의 게시글에 달린 모든 댓글 확인

```
>>> for board in user1.board_set.all():
...     for comment in board.comment_set.all():
...             print(comment.content)
...
1글1댓글
1글2댓글
1글3댓글
1글4댓글
!1글5댓글
2글1댓글
!2글2댓글
```

2번 댓글을 쓴 사람의 모든 게시글 확인

```
>>> c2.user.board_set.all()
<QuerySet [<Board: 3글>]>
```

1번 게시글의 첫번째 댓글 쓴 사람의 이름 확인

```
>>> board1.comment_set.first().user.name
'Kim'
>>> board1.comment_set.all()[0].user.name
'Kim'
```

1번 게시글의 두번째 댓글 쓴 사람의 첫번째 게시글의 작성자 이름 확인

```
>>> board1.comment_set.all()[1].user.board_set.first().user.name
'Lee'
```

[n:m 관계]

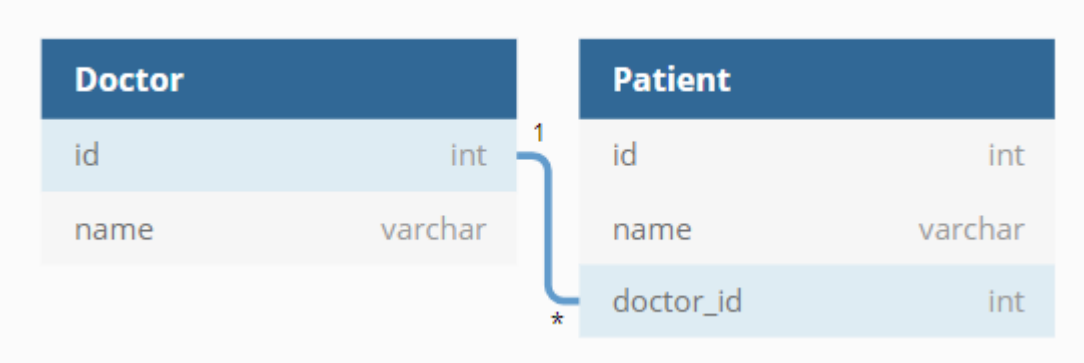
app : manytomany

1. model 생성

[models.py]

```python
from django.db import models

# Create your models here.
class Doctor(models.Model):
    name = models.CharField(max_length=20)

    def __str__(self):
        return f'{self.id}번 의사 {self.name}'

class Patient(models.Model):
    name = models.CharField(max_length=20)
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
```

```
    def __str__(self):
        return f'{self.id}번 환자 {self.name}'
```



의문점 : 의사-환자가 다대다 관계일 때, 어떻게 구성해야 하는가?

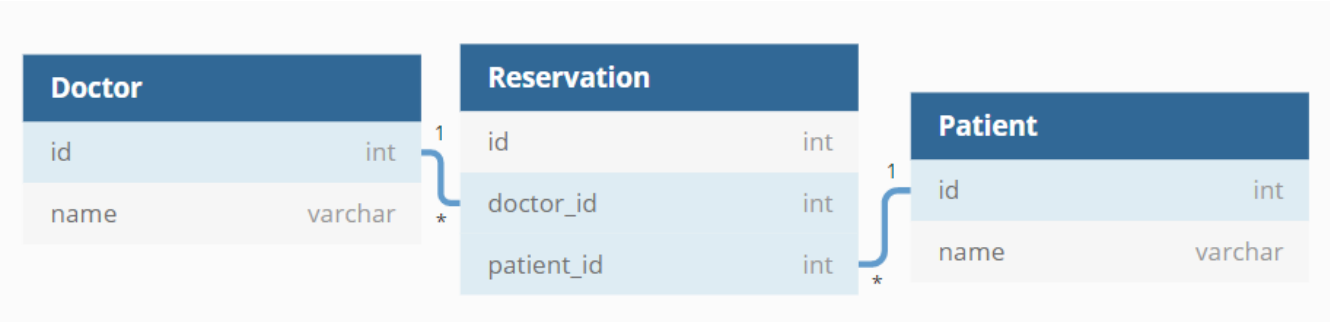
- 한 환자가 2명의 의사에게 진료를 받을 경우
	A	B	C	D	E	F
1	Doctor			Patient		
2	id	name		id	name	doctor_id
3	1	Kim		1	John	1
4	2	Lee		2	Sally	2
5				3	Sally	1
6						
  - 환자 Sally는 의사 Kim, Lee에게 진료를 받고 있다.
  - 문제점 : Patient 테이블에 Sally가 두 번 들어가서, Sally는 id를 2개를 갖게 된다. (id: 2, 3)
  - => 진료를 받는 **관계를 표현할 테이블** (중계모델)을 따로 만들어보자.
	A	B	C	D	E	F	G	H	I
1	Doctor			Reservation				Patient	
2	id	name		id	doctor_id	patient_id		id	name
3	1	Kim		1	1	1		1	John
4	2	Lee		2	1	2		2	Sally
5				3	2	2			

2. Reservation 모델 추가

[models.py]

```
...
class Patient(models.Model):
    name = models.CharField(max_length=20)
    # doctor 외래키 삭제

    def __str__(self):
        return f'{self.id}번 환자 {self.name}'

class Reservation(models.Model):
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE)
```

```
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE)

    def __str__(self):
        return f'{self.doctor.id}번 의사의 {self.patient.id} 번 환자'
```



의사와 환자를 등록하고, 그 관계도 등록해보기

```
>>> doctor = Doctor.objects.create(name="Kim")
>>> patient = Patient.objects.create(name="John")
>>>
>>> Reservation.objects.create(doctor=doctor, patient=patient)
<Reservation: 1번 의사의 1 번 환자>
```

퀴즈

1번 의사가 담당하는 환자들 찾기

```
>>> doctor.reservation_set.all()
<QuerySet [<Reservation: 1번 의사의 1 번 환자>]>
```

1번 환자의 담당 의사 목록 찾기

```
>>> patient.reservation_set.all()
<QuerySet [<Reservation: 1번 의사의 1 번 환자>]>
```

특정 의사가 담당하는 환자들 이름 출력

```
>>> for reservation in doctor.reservation_set.all():
...     print(reservation.patient.name)
...
John
Tom
```

3. 직관적으로 관계 가져오기

현재는 reservation을 통해야만 하니까, 가져오기 좀 불편하다.

모델을 변경해서 좀 더 직관적으로 가져와보자.

[models.py]

```
...
class Patient(models.Model):
    name = models.CharField(max_length=20)
    doctors = models.ManyToManyField(Doctor, through='Reservation') # Reservation을 통해서 담당
의사 목록 받아옴

    def __str__(self):
        return f'{self.id}번 환자 {self.name}'
...
```


```
>>> patient.doctors.all() # 중계모델을 거치지 않음!
<QuerySet [<Doctor: 1번 의사 Kim>, <Doctor: 2번 의사 Hwnag>]>
```

지금, Doctor 모델 내부에는 manyTomany로 지정된 Patient 모델이 없음

의사가 담당하는 환자목록 가져오기

```
>>> doctor.patient_set.all()
<QuerySet [<Patient: 1번 환자 John>, <Patient: 2번 환자 Tom>]>
```

불편하니까, **역참조**를 활용해서, 의사도 바로 환자를 가져올 수 있도록 하자!

4. 역참조 설정

[models.py]

```
...
class Patient(models.Model):
    name = models.CharField(max_length=20)
    doctors = models.ManyToManyField(Doctor, through='Reservation', related_name='patients')
# 역으로 참조 당할 때, 나는 patients라는 이름의 필드일거야!

    def __str__(self):
        return f'{self.id}번 환자 {self.name}'
 ...
```

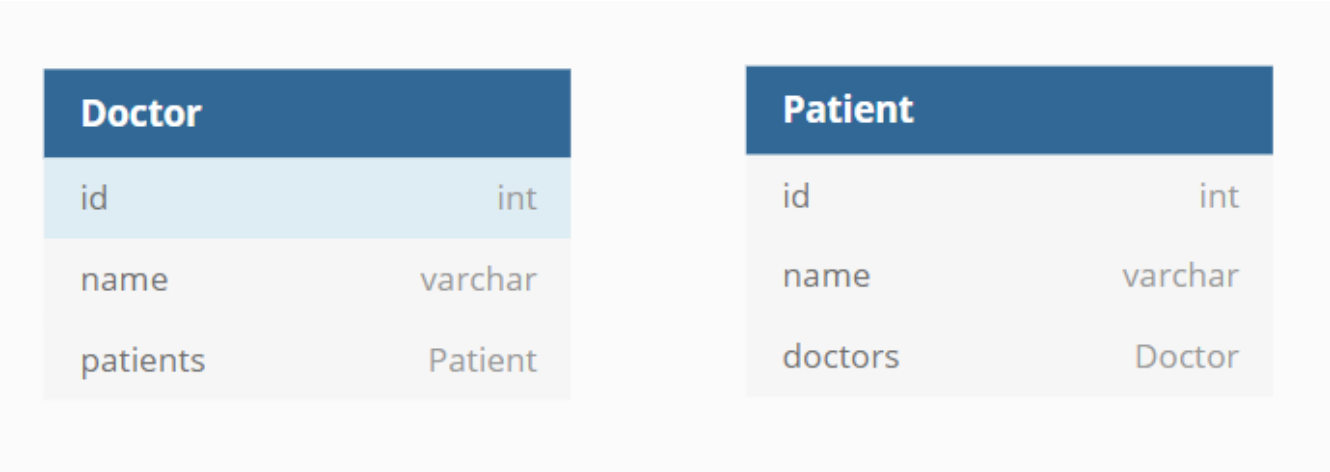
```
>>> doctor.patients.all() # doctor가 patient를 참조함. (역참조)
<QuerySet [<Patient: 1번 환자 John>, <Patient: 2번 환자 Tom>]>
```

5. 중계모델 삭제

역참조가 설정 되었으니, 중계모델이 굳이 필요하지 않음. 굳이 중계모델을 통하지 않으니까! -> 지워버리자

[models.py]

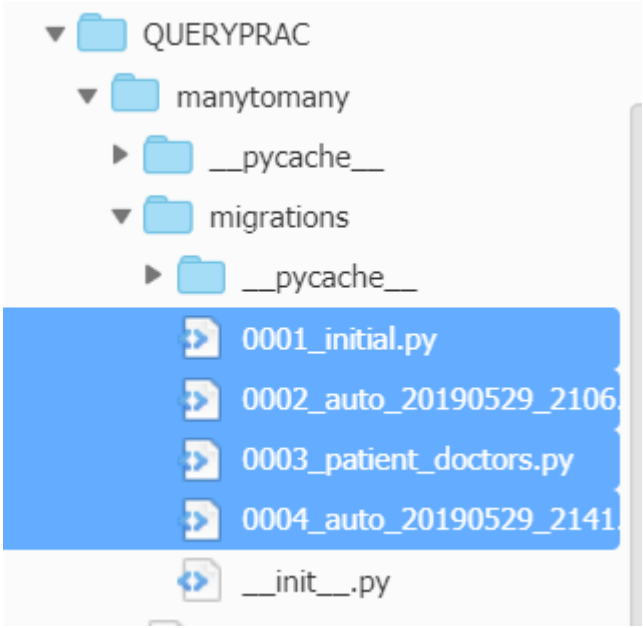
```
class Patient(models.Model):
    name = models.CharField(max_length=20)
    doctors = models.ManyToManyField(Doctor, related_name='patients') # through 삭제

    def __str__(self):
        return f'{self.id}번 환자 {self.name}'

# class Reservation 삭제
```



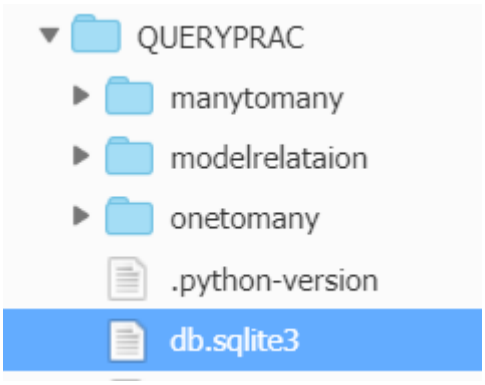
Database 날리기

모델이 많이 변해서 기존 db와 연동이 잘 안되나봄? 그래서 DB 날려줌

- migrations에서, init 빼고 전부 삭제



- db.sqlite3 삭제



역참조 활용하기

```
>>> doctor = Doctor.objects.create(name='Kim')
>>> patient = Patient.objects.create(name='John')
>>>
>>> doctor.patients.add(patient) # 중계모델 없이 접근함!
>>> doctor.patients.all()
<QuerySet [<Patient: 1번 환자 John>]>
```

[게시글 좋아요 기능 만들기]

project : PROJECT03

app : boards

Board 모델 변경

[models.py]

```python
class Board(models.Model):
    title = models.CharField(max_length=100)
    content = models.CharField(max_length=300)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    like_users = models.ManyToManyFields(settings.AUTH_USER_MODEL,
related_name='like_boards', blank=True) # board - user 간 다대다 관계

    def __str__(self):
        return f'[{self.pk}] {self.title} : {self.content}'
```