date: 0331

m:n LIKE

1. m: n model

- m: n 관계를 형성할 곳 중 편한 곳에 ManyToManyField를 작성한다.

```
class Article(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE) #유저들 (1:n)
    like_users = models.ManyToManyField(settings.AUTH_USER_MODEL,
related_name='like_articles') #좋아요 누른 유저들(m:n)
    #.......
```

  - related_name: 역참조시 이름을 재정의 해준다
    - why?????
    - User-Article 모델에서 1: n 관계에서 참조는 다음과 같이 지어진다. `article.user.all()`, `user.article_set.all()` 그런데 m:n 관계에서도 `article.like_users.all()`, `user.article_set.all()`로 역참조 매니저의 이름이 같아서 충돌오류가 발생한다
    - related_name은 사용하면 `user_like_articles`로 이름을 바꾸어 구분도 가능하고, 오류도 없애줄 수 있다.
  - ManyToManyField Table은 `app이름_model이름_필드이름`으로 작성된다.
    - ex) `articles_article_like_users`

2. URL

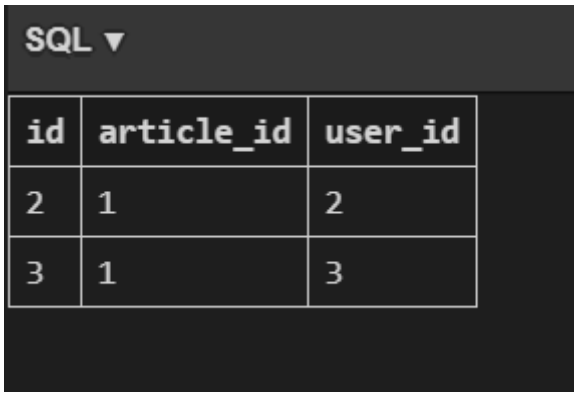
```
urlpatterns = [
    path('<int:article_pk>/likes/', views.likes, name='likes'),
]
```

3. views.py

```
# 좋아요는 POST로만!
@require_POST
def likes(request, article_pk):
    # 로그인을 안하면, 로그인페이지로 간다!
    if request.user.is_authenticated:
        # 좋아요 할 게시글의 정보를 가져옴
        article = get_object_or_404(Article, pk=article_pk)
        # 내가 좋아요 모음 테이블(article.like_user.all())에 있다면
        # if request.user in article.like_users.all():
        if article.like_users.filter(pk=request.user.pk).exists():
```

```
            # 좋아요 취소: article의 like_users테이블에 요청유저를 추가함
            article.like_users.remove(request.user)
        else:
            # 좋아요 누름: article의 like_users테이블에 요청유저를 추가함
            article.like_users.add(request.user)
        return redirect('articles:index')
    return redirect('accounts:login')
```

- 좋아요를 누른 후 `articles_article.like_users` 상태



  - 1번 게시글의 2번 유저와 3번 유저가 좋아요를 누른 것이 기록됨
- if article.like_users.filter(pk=request.user.pk).exists():
  - 의미는 완벽하게 `if request.user in article.like_users.all()`와 동일하다.
  - filter : pk가 요청자의 유저 pk와 동일한지 확인한다.
  - exists(): 적어도 하나라도 존재하는 지 확인, 전체조회가 아니다.
  - `exists()` 가 `in`보다 더욱 빠르다
    - why?
    - if문 평가(db로 쿼리셋을 넘길때)시 캐쉬로 쿼리셋을 넘겨둔다.(메모리를 아끼기 위해서)
    - in은 이 캐쉬를 저장해 두지만, .exists()는 캐쉬를 만들지 않아서 빠르다.
    - 단, 매우 큰 쿼리셋이 아니면 성능차이는 인식하기 힘들다.

4. index.html

```
<div>
     <form action="{% url 'articles:likes' article.pk %}" method="POST">
       {% csrf_token %}
       {% if request.user in article.like_users.all %}
         <button>좋아요 취소</button>
       {% else %}
         <button>좋아요</button>
       {% endif %}
     </form>
   </div>
   <p>{{ article.like_users.all|length }}명이 이 글을 좋아합니다.</p>
```

- 좋아요를 눌렀다면, 취소버튼이 보이게, 안눌렀다면 좋아요 버튼이 보이게 if문으로 구분
- length필터를 이용하여 총 몇 명이 누른 지도 표시 가능

cf). queryset 특강, 시험내용은 아닌듯

- queryset is lazy
    - queryset 만들기는 db가 전혀 관여하지 않음(filter등은 관여 x)
    - 평가할때(쿼리를 db로 보낸다, 쿼리셋 캐쉬를 생성한다)만 db가 관여
    - 평가 = 반복(for), 슬라이싱, repr(print), bool(if) 등
    - 평가시 캐쉬가 만들어지고 쿼리가 저장되어서 재사용 시 그 값을 반복 사용

- 최적화는 적은 쿼리를 사용하는 것

```
# 우리의 LIKE 코드를 예시로 사용해보자
like_set = article.like_users.filter(pk=request.user.pk)
if like_set: # 평가
    # 쿼리셋의 전체 결과가 필요하지 않은 상황임에도
    # ORM은 전체 결과를 가져옴
    article.like_users.remove(request.user)

# 개선 1
# exists() 쿼리셋 캐시를 만들지 않으면서 특정 레코드가 있는지 검사
if like_set.exists():
    # exists는 쿼리를 만들지 않는다.
    # 따라서, DB에서 가져온 레코드가 없다면
    # 메모리를 절약할 수 있다
    article.like_users.remove(request.user)
```

- 안일한 최적화
    - 최적화 하다가 망가지니까 지금은 신경쓰지 말것!!!!!!!!!!

```
# [참고] 쿼리셋 효과적으로 사용하기
# https://docs.djangoproject.com/en/3.1/topics/db/queries/#querysets-are-lazy
# https://docs.djangoproject.com/en/3.1/ref/models/querysets/#when-querysets-are-evaluated
# https://docs.djangoproject.com/en/3.1/topics/db/queries/#caching-and-querysets
```