

```
if article.like_users.filter(pk=request.user.pk).exists():
# if request.user in article.like_users.all():
# 좋아요 취소
```

위 아래는 같은 말임

exists: 쿼리셋에 결과가 포함되어있으면 true를 아니면 false를 반환, 검색에 유용, ~in보다 접근이 빠르다(평가에서 캐시를 사용하지 않음)

follow 기능 구현

article이 아닌 user의 profile을 만들어 진행하고 싶기 때문에 accounts앱에서

```
app_name = 'accounts'
urlpatterns = [
    path('login/', views.login, name='login'),
    path('logout/', views.logout, name='logout'),
    path('signup/', views.signup, name='signup'),
    path('delete/', views.delete, name='delete'),
    path('update/', views.update, name='update'),
    path('password/', views.change_password, name='change_password'),
    path('<username>/', views.profile, name='profile'),
]
```

< str:username > == < username >

```
def profile(request, username):
    person = get_object_or_404(get_user_model(), username=username)
    context = {
        'person': person,
    }
    return render(request, 'accounts/profile.html', context)
```

username을 보내서 만들었으므로 username을 전달

유저와 article: 1대 n

유저와 comment: 1대 n

유저와 좋아요 누른 게시글 : n대 m 이므로

```

<h2>{{ person.username }}'s 게시물</h2>
{% for article in person.article_set.all %}
    <div>{{ article.title }}</div>
{% endfor %}

<hr>

<h2>{{ person.username }}'s 댓글</h2>
{% for comment in person.comment_set.all %}
    <div>{{ comment.content }}</div>

<h2>{{ person.username }}'s likes</h2>
{% for article in person.like_articles.all %}
    <div>{{ article.title }}</div>
{% endfor %}

```

이런식으로 profile html 만들기

```

urls.py  views.py  profile.html  base.html  index.html
articles > templates > articles > index.html
1  {% extends 'base.html' %}
2
3  {% block content %}
4      <h1>Articles</h1>
5      {% if request.user.is_authenticated %}
6          <a href="{% url 'articles:create' %}">[CREATE]</a>
7      {% else %}
8          <a href="{% url 'accounts:login' %}">[새 글을 작성하려면 로그인하세요.]</a>
9      {% endif %}
10     <hr>
11     {% for article in articles %}
12         <p>
13             <a href="{% url 'accounts:profile' request.user.username %}">
14             </p>
15         <p>글 번호 : {{ article.pk }}</p>
16         <p>글 제목 : {{ article.title }}</p>
17         <p>글 내용 : {{ article.content }}</p>
18         <div>
19             <form action="{% url 'articles:likes' article.pk %}" method="POST">
20                 {% csrf_token %}
21                 {% if request.user in article.like_users.all %}
22                     <button>좋아요 취소</button>

```

이런식으로 작성하면 요청보낸 사람의 profile로 이동한다

```

1 {% extends 'base.html' %}
2
3 {% block content %}
4 <h1>Articles</h1>
5 {% if request.user.is_authenticated %}
6 <a href="{% url 'articles:create' %}">[CREATE]</a>
7 {% else %}
8 <a href="{% url 'accounts:login' %}">[새 글을 작성하려면 로그인하세요.]</a>
9 {% endif %}
10 <hr>
11 {% for article in articles %}
12 <p>
13 <a href="{% url 'accounts:profile' article.user.username %}">
14 </p>
15 <p>글 번호 : {{ article.pk }}</p>
16 <p>글 제목 : {{ article.title }}</p>
17 <p>글 내용 : {{ article.content }}</p>
18 <div>
19 <form action="{% url 'articles:likes' article.pk %}" method="POST">
20 {% csrf_token %}
21 {% if request.user in article.like_users.all %}
22 <button>좋아요 취소</button>

```

그러므로 article.user.username이라고 해야 작성자를 볼 수 있다

follow는 user과 user간 일이므로 accounts에 작성

팔로잉과 팔로워가 항상 같진 않으므로 symmetrical은 false

팔로잉 유저와 팔로워 유저를 다르게 하기 위해서

역참조를 followers로 준다.

self이므로 from ... to... 로 생긴다

```

4 # Create your models here.
5 class User(AbstractUser):
6     followings = models.ManyToManyField('self', symmetrical=False, related_name='followers')
7
8

```

```

path('<username>/', views.profile, name='profile'),
path('<int:user_pk>/follow/', )
]

```

get이 아니라 filter를 사용해 조회를 하는 이유?

get은 값이 없으면 does not exist 를 반환해 깨져버림(코드 동작 x)

filter는 값이 없으면 빈 쿼리셋을 리턴하기 때문에 ㄱㄷ다.

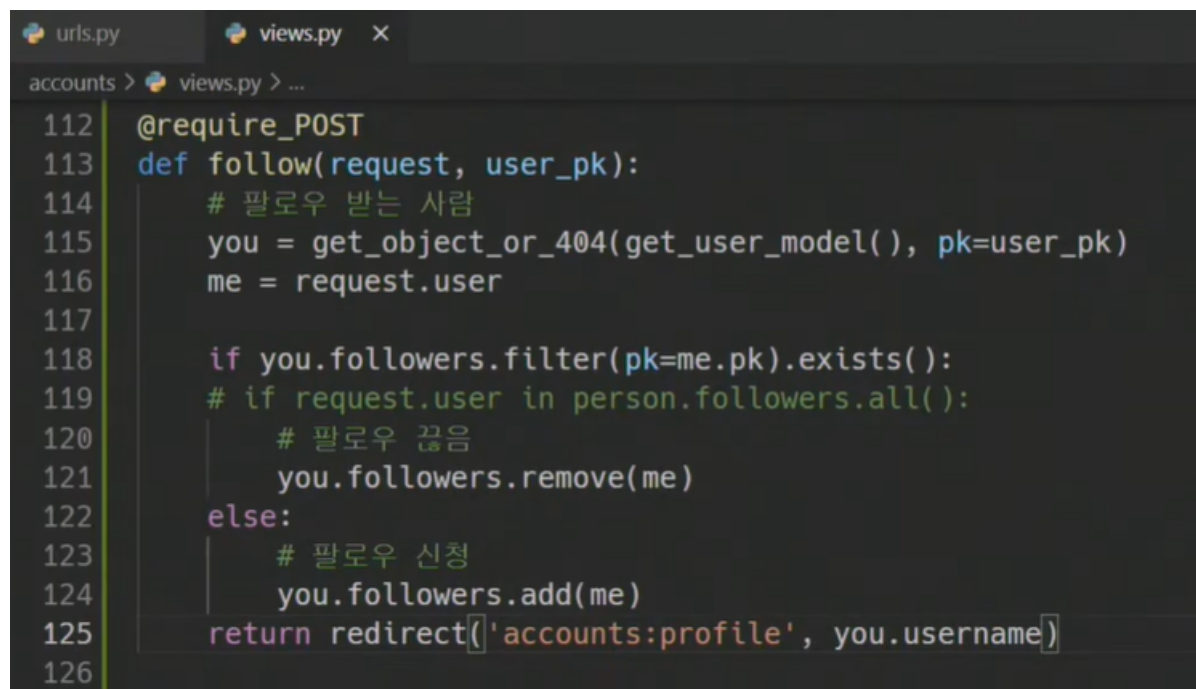
```

@require_POST
def follow(request, user_pk):
    # 팔로우 받는 사람
    person = get_object_or_404(get_user_model(), p

    if person.followers.filter(pk=request.user.pk)
    # if request.user in person.followers.all():
        # 팔로우 끊음
    else:
        # 팔로우 신청

```

헛갈리면 you, me 사용하기



```

urls.py  views.py  X
accounts > views.py > ...
112 @require_POST
113 def follow(request, user_pk):
114     # 팔로우 받는 사람
115     you = get_object_or_404(get_user_model(), pk=user_pk)
116     me = request.user
117
118     if you.followers.filter(pk=me.pk).exists():
119         # if request.user in person.followers.all():
120             # 팔로우 끊음
121             you.followers.remove(me)
122     else:
123         # 팔로우 신청
124         you.followers.add(me)
125     return redirect('accounts:profile', you.username)
126

```

you의 페이지로 이동할 것이므로 you.username 같이 보내기

나 자신은 팔로워 할 수 없으므로

if you != me:

```

@require_POST
def follow(request, user_pk):
    # 팔로우 받는 사람
    you = get_object_or_404(get_user_model(), pk=user_pk)
    me = request.user

    # 나 자신은 팔로우 할 수 없다.
    if you != me:
        if you.followers.filter(pk=me.pk).exists():
            # if request.user in person.followers.all():
            # 팔로우 끊음
            you.followers.remove(me)
        else:
            # 팔로우 신청
            you.followers.add(me)
    return redirect('accounts:profile', you.username)

```

request.user과 person이 같으면 내 페이지라는 뜻이니까

팔로우랑 언팔로우 버튼 필요 없음

팔로워면 > 언팔버튼 / 언팔로워면 > 팔로우버튼

```

<div>
  <div>
    팔로잉 : {{}} / 팔로워 : {{}}
  </div>
  {% if request.user != person %}
    <div>
      <form action="{% url 'accounts:follow' person.pk %}" method="POST">
        {% csrf_token %}
        {% if request.user in person.followers.all %}
          <button>언팔로우</button>
        {% else %}
          <button>팔로우</button>
        {% endif %}
      </form>
    </div>
  {% endif %}
</div>

```

```

팔로잉 : {{ person.followings.all|length }} / 팔로워 : {{ person.followers.all|length }}
</div>

```



```

@require_POST
def follow(request, user_pk):
    if request.user.is_authenticated:
        # 팔로우 받는 사람
        you = get_object_or_404(get_user_model(), pk=user_pk)
        me = request.user

        # 나 자신은 팔로우 할 수 없다.
        if you != me:
            if you.followers.filter(pk=me.pk).exists():
                # if request.user in person.followers.all():
                # 팔로우 끊음
                you.followers.remove(me)
            else:
                # 팔로우 신청
                you.followers.add(me)
        return redirect('accounts:profile', you.username)
    return redirect('accounts:login')

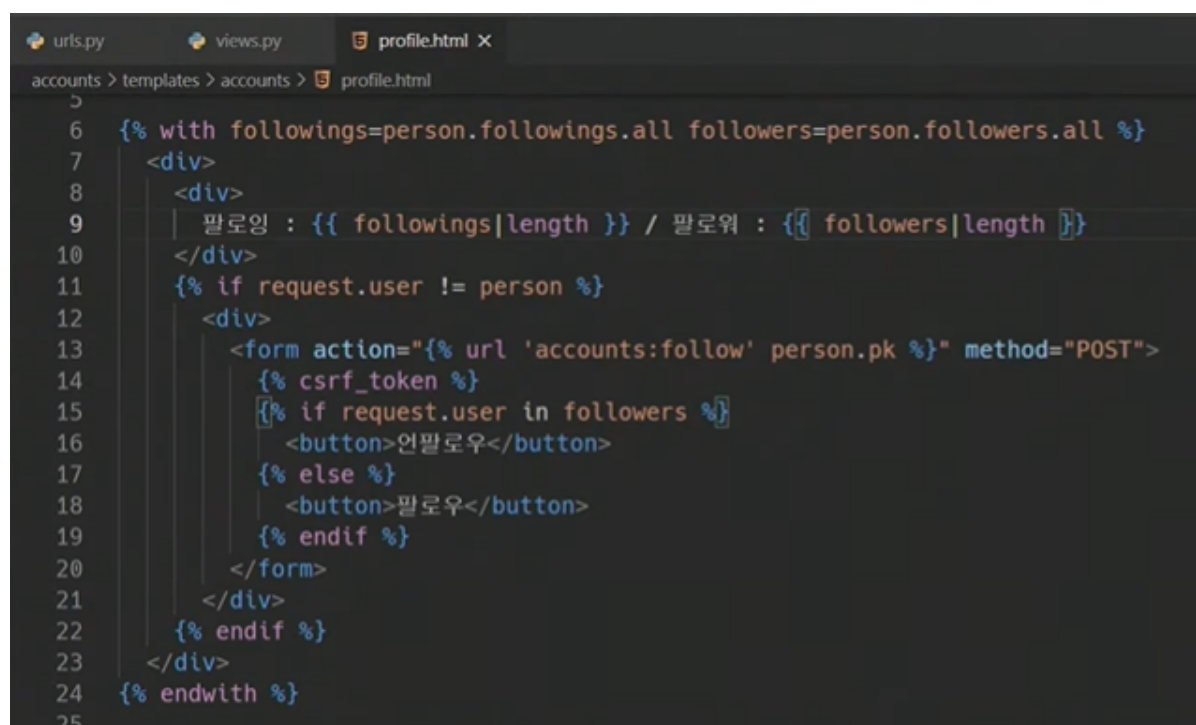
```

팔로우 view 최종

TMI

person.followings.all이 너무 자주 쓰임

with를 사용하기(endwith까지가 범위임, with의 변수를 사용할 수 있는 공간)

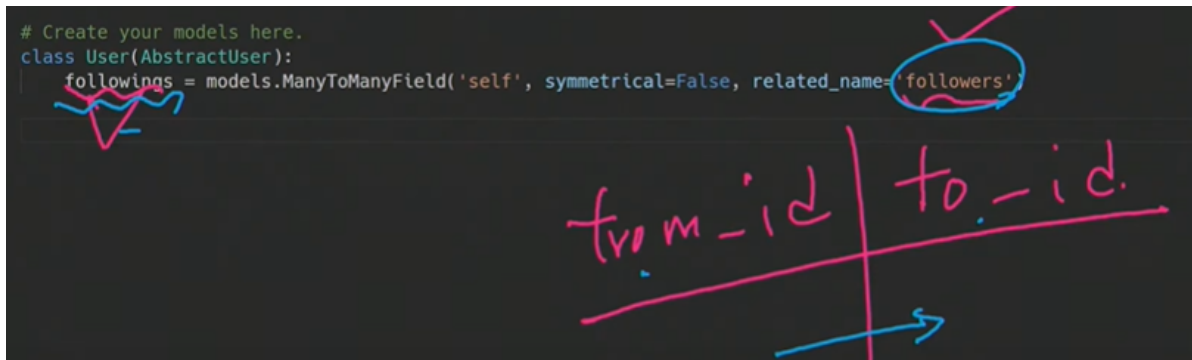


```

accounts > templates > accounts > profile.html
6  {% with followings=person.followings.all followers=person.followers.all %}
7  <div>
8      <div>
9          팔로잉 : {{ followings|length }} / 팔로워 : {{ followers|length }}
10     </div>
11     {% if request.user != person %}
12         <div>
13             <form action="{% url 'accounts:follow' person.pk %}" method="POST">
14                 {% csrf_token %}
15                 {% if request.user in followers %}
16                     <button>언팔로우</button>
17                 {% else %}
18                     <button>팔로우</button>
19                 {% endif %}
20             </form>
21         </div>
22     {% endif %}
23 </div>
24 {% endwith %}
25

```

최적화와는 관련이 없고 그냥 반복해서 길게 쓰는걸 줄일 수 있음



following과 follower를 너무 헷갈려할 필요는 없다.

어차피 종속 관계가 아니기 때문이다

둘 중 하나는 참조, 둘 중 하나는 역참조일 뿐이다.

lazy queryset

```
# 실제 쿼리셋을 만드는 작업에는 DB 작업이 포함되지 않음
q = Entry.objects.filter(title__startswith="What")
q = q.filter(created_at__lte=datetime.date.today())
q = q.exclude(context__icontains="food")
print(q)
```

쿼리셋을 만드는 동안은 db에 반영되지 않음

print가 실행될 때 평가가 실행됨

평가

1. 쿼리를 db로 날린다(웹을 느려지게 하는 요인 중 하나)
2. 쿼리셋 캐시에 저장

filter는 평가가 되지 않으므로 길게 써도 db와는 관련이 없다

쿼리셋이 평가되는 경우(시점)

1. 반복될 때
2. 슬라이싱될 때
3. repr : 객체 표현할때(print 될때)
4. 길이 잴때
5. 리스트 호출할때
6. 불리언일때(if 문일때)

평가 후 쿼리셋의 내장 캐시에 저장

장점은 캐시에 저장해두기 때문에 재사용할 수 있음

but,,,

```
# 나쁜 예
print([e.headline for e in Entry.objects.all()]) # 평가
print([e.pub_date for e in Entry.objects.all()]) # 평가
```

평가가 두번 일어남

```
# 좋은 예
queryset = Entry.objects.all()
print([p.headline for p in queryset]) # Evaluate the query set. (평가)
print([p.pub_date for p in queryset]) # Re-use the cache from the evaluation. (캐시에서 재사용)
```

최적화에는 이 구조가 더 좋다

평가했던걸 다시 사용 가능

캐시되지 않는 경우도 있다

집약된 인덱스의 경우

최적화: 적은 쿼리로 원하는 데이터를 얻는것

```
168 # 우리의 LIKE 코드에서
169 like_set = article.like_users.filter(pk=request.user.pk)
170 if like_set: # 평가
171     # 쿼리셋의 전체 결과가 필요하지 않은 상황임에도
172     # ORM은 전체 결과를 가져옴
173     article.like_users.remove(request.user)
174
```

169번째줄 db 놓고있음

170번째줄에서 평가 일어남

but 전체 결과를 가져오게 됨(값이 없어도 전체 평가)

<개선1>

exists 사용 : 쿼리셋 캐시 만들지 않으면서 검사 가능

db에서 가져온 레코드가 없기 때문에 메모리 절약

```
175 # 개선 1
176 # exists() 쿼리셋 캐시를 만들지 않으면서 특정 레코드가 있는지 검사
177 if like_set.exists():
178     # DB에서 가져온 레코드가 없다면
179     # 메모리를 절약할 수 있다
180     article.like_users.remove(request.user)
181
```

<개선2>


```
# if에서 평가 후 캐싱 I
if like_set:
    # 순회할때는 위에서 캐싱된 쿼리셋을 사용
    for user in like_set:
        print(user.username)
```

이럴 땐 위에서 평가한걸 그대로 가져와서 사용하기 때문에
어느정도 최적화되었다.

but 쿼리셋이 너무너무 크다면..?

이터레이터는 잘게잘게 잘라서 올리고 다시 내리므로 이터레이터를 사용해준다

```
# 만약 쿼리셋 자체가 너무너무 크다면??
# iterator()
# 데이터를 작은 덩어리로 쪼개서 가져오고, 이미 사용한 레코드는 메모리에서 지움
# 전체 레코드의 일부씩만 DB에서 가져오므로 메모리를 절약
if like_set:
    for user in like_set.iterator():
```

여기서 if도 안쓰고 exists까지 쓰면 더 좋다

```
# 그런데 쿼리셋이 너무너무 크다면 if 평가에서도 버거움
if like_set.exists():
    for user in like_set.iterator():
```

하지만 여기서는 캐시가 생기지 않으므로

안일한 최적화가 일어날 가능성이 있음(아래에 어떻게 쓰일지 모르니까..!)