

# Django DB (1:N)

# 1:N 관계

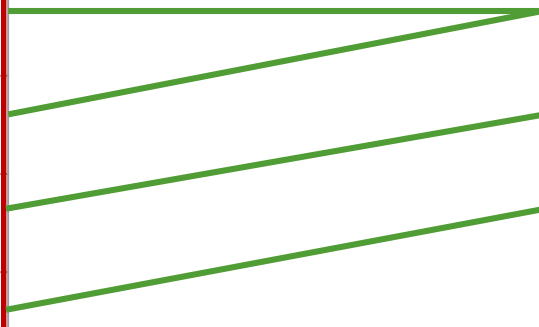
A 테이블의 하나의 레코드가 B 테이블의 하나 이상의 레코드와 연결  
하지만! B 테이블의 하나의 레코드는 A 테이블 하나의 레코드에만 연결!!

반납 핸드폰 테이블

ID	핸드폰 번호	반납 시간	학생 ID
1	010-0000-0000	10:00	1
2	010-1111-1111	10:00	1
3	010-2222-2222	11:00	2
4	010-3333-3333	11:30	3

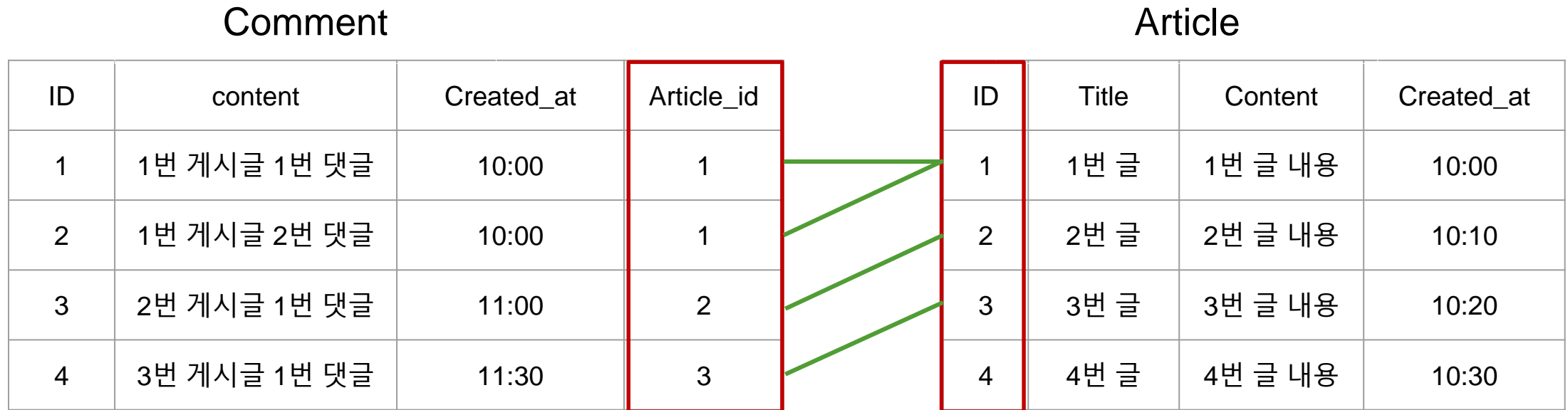
학생 테이블

ID	이름	주소	성별
1	김씨	서울	남
2	나씨	경기	여
3	박씨	부산	남
4	이씨	강원	여



# 1:N 관계

하나의 Article에는 여러 개의 댓글이 달릴 수 있음



# 댓글 모델 생성

---

## ForeignKey

### 1. 1:N 관계를 설정할 때 사용하는 모델 필드

```
# articles/models.py
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

# 댓글 모델 생성

---

## ForeignKey

### 2. 속성값은 단수형으로 작성하는 것을 권장

```
# articles/models.py
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

# 댓글 모델 생성

---

## ForeignKey

### 3. 첫 번째 인자에는 참조하려는 모델을 작성

```
# articles/models.py
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

# 댓글 모델 생성

---

## ForeignKey

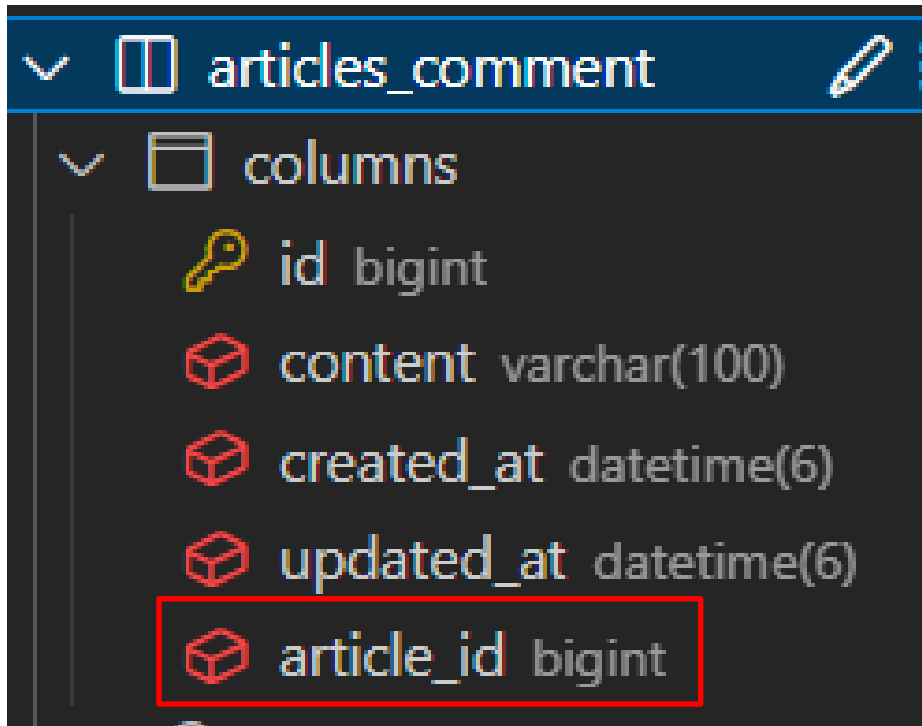
4. 두 번째 인자에는 참조 모델이 삭제될 때 어떻게 처리할 지를 정의
- CASCADE: 참조 모델이 삭제될 때, 참조하는 모델도 삭제

```
# articles/models.py
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

# 댓글 모델 생성

---

migrations을 통한 테이블 생성 확인





# 댓글 작성 구현

댓글을 입력받기 위해 Comment ModelForm 정의 및 detail 페이지 수정

```
# articles/forms.py
from .models import Article, Comment

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = '__all__'
```

```
# articles/views.py
from .forms import ArticleForm, CommentForm

def detail(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm()
    context = {
        'article': article,
        'comment_form': comment_form
    }
    return render(request, 'articles/detail.html', context)
```

```
<!-- articles/detail.html -->
<form action="#" method="POST">
    {% csrf_token %}
    {{ comment_form }}
    <input type="submit" value="댓글 작성하기">
</form>
```



The screenshot shows a web browser at the URL localhost:8000/articles/detail/1/. The page title is "DETAIL". Below the title, it says "1 번째 글". The article details are: "작성자: admin", "제목: 기사1", and "내용: 기사2". Below this is a section titled "댓글 목록" (Comment List). It contains a form with a dropdown menu for "Article:" (currently showing "-----"), a text input for "Content:", and a button labeled "댓글 작성하기" (Write Comment). At the bottom of the form area is a link "[back]" in blue.

# 댓글 작성 구현

## 댓글 입력 시 글을 선택하는 필드를 제거

```
# articles/forms.py
from .models import Article, Comment

class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ('content', )
```

← → ↻ ⓘ localhost:8000/articles/detail/1/

### DETAIL

1 번째 글

---

작성자: admin  
제목: 기사1  
내용: 기사2

---

#### 댓글 목록

Article:  ▼

Content:

[\[back\]](#)



← → ↻ ⓘ localhost:8000/articles/detail/1/

### DETAIL

1 번째 글

---

작성자: admin  
제목: 기사1  
내용: 기사2

---

#### 댓글 목록

Content:

[\[back\]](#)

# 댓글 작성 로직 구현

---

detail 페이지에 댓글 작성 url로 연결 및 url 작성

```
<!-- articles/detail.html-->
<form action="{% url 'articles:create_comment' pk=article.pk %}" method="POST">
    {% csrf_token %}
    {{ comment_form }}
    <input type="submit" value="댓글 작성하기">
</form>
```

```
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('create_comment/<int:pk>', views.create_comment, name='create_comment'),
    ...
]
```

# 댓글 작성 로직 구현

---

## 댓글 작성 함수 생성

```
# articles/views.py
def create_comment(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm(request.POST)
    if comment_form.is_valid():
        comment_form.save()
        return redirect('articles:detail', pk)
    context = {
        'article': article,
        'comment_form': comment_form
    }
    return render(request, 'articles/detail.html', context)
```

# 댓글 작성 로직 구현

댓글 작성 시 아래와 같은 에러 발생

=> comment에 필요한 article\_id를 넣어주지 않았기 때문에

← → ↻ ⓘ localhost:8000/articles/create\_comment/1

IntegrityError at /articles/create\_comment/1  
(1048, "Column 'article\_id' cannot be null")

Request Method: POST  
Request URL: http://localhost:8000/articles/create\_comment/1  
Django Version: 5.0.4  
Exception Type: IntegrityError  
Exception Value: (1048, "Column 'article\_id' cannot be null")

**Request information**

**USER** iju9530

**GET** No GET data

**POST**

Variable	Value
csrfmiddlewaretoken	'iTcA9IcTkntS5Ct9bmSGIAoLGJQ3I51UHMqQHpcFE9YA8oP1I08mprXHlWMpuP'
content	'123'

# 댓글 작성 로직 구현

---

commit의 인스턴스를 생성한 후에 article 데이터를 삽입 후 생성

=> **save(commit=False)** 은 인스턴스만 생성하는 함수

```
# articles/views.py
def create_comment(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm(request.POST)
    if comment_form.is_valid():
        comment = comment_form.save(commit=False)
        comment.article = article
        comment.save()
        return redirect('articles:detail', pk)
    context = {
        'article': article,
        'comment_form': comment_form
    }
    return render(request, 'articles/detail.html', context)
```

# Django 1:N 참조와 역참조

---

Comment가 작성된 게시글을 조회하는 경우 ( 참조 )

- Comment Model에 article 속성이 존재하기 때문에 바로 참조 가능

```
# articles/models.py
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
In [2]: comment = Comment.objects.get(pk=5)
```

```
In [3]: comment.article
```

```
Out[3]: <Article: Article object (1)>
```

```
In [4]: comment.article.title
```

```
Out[4]: '기사1'
```

# Django 1:N 참조와 역참조

---

게시글에 작성된 댓글을 조회하는 경우 ( 역참조 )

- Article Model에 comment 속성이 존재하기 때문에 참조 불가능

```
class Article(models.Model):  
    title = models.CharField(max_length=20)  
    content = models.TextField()  
    created_at = models.DateTimeField(auto_now_add=True)  
    updated_at = models.DateTimeField(auto_now=True)
```



# Django 1:N 참조와 역참조

---

게시글에 작성된 댓글을 조회하는 경우 ( 역참조 )

```
article.comment_set.all()
```

- **article** : model instance
- **comment\_set**
  - related manager, 역참조를 도와주는 매니저
  - 역참조에 사용되며 기본적으로 모델명\_set으로 생성됨
- **all** : 전체를 조회하는 QuerySet API

# Django 1:N 참조와 역참조

---

게시글에 작성된 댓글을 조회하는 경우 ( 역참조 )

```
article.comment_set.all()
```

- **article** : model instance
- **comment\_set**
  - related manager, 역참조를 도와주는 매니저
  - 역참조에 사용되며 기본적으로 모델명\_set으로 생성됨
- **all** : 전체를 조회하는 QuerySet API

# 댓글 목록 구현

```
# articles/views.py

def detail(request, pk):
    article = Article.objects.get(pk=pk)
    comments = article.comment_set.all()
    comment_form = CommentForm()
    context = {
        'article': article,
        'comments': comments,
        'comment_form': comment_form
    }
    return render(request, 'articles/detail.html', context)
```

```
<!-- articles/detail.html -->
<h3>댓글 목록</h3>
<ol>
    {% for comment in comments %}
        <li>
            {{ comment.content }}
        </li>
    {% endfor %}
</ol>
```

← → ↻ ⓘ localhost:8000/articles/detail/1/

## DETAIL

### 1 번째 글

---

작성자: admin

제목: 기사1

내용: 기사2

---

### 댓글 목록

1. 123

Content:

---

[\[back\]](#)

# 역참조 쉽게 하기

역참조 매니저의 이름을 설정하기

=> 이름을 변경하면 기존 매니저는 사용할 수 없음

```
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE, related_name="comments")
    content = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
def detail(request, pk):
    article = Article.objects.get(pk=pk)
    comments = article.comments.all()
    comment_form = CommentForm()
    context = {
        'article': article,
        'comments': comments,
        'comment_form': comment_form
    }
    return render(request, 'articles/detail.html', context)
```

# 댓글 삭제 구현

```
<!-- articles/detail.html -->
<h3>댓글 목록</h3>
<ol>
    {% for comment in comments %}
        <li>
            {{ comment.content }}
            <form action="{% url 'articles:delete_comment' article_id=article.pk comment_id=comment.id %}" method="POST">
                {% csrf_token %}
                <input type="submit" value="댓글 삭제하기">
            </form>
        </li>
    {% endfor %}
</ol>
```

```
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('<int:article_id>/comments/<int:comment_id>/delete/',
        views.delete_comment, name="delete_comment"),
    ...
]
```

```
# articles/views.py
def delete_comment(request, article_id, comment_id):
    comment = Comment.objects.get(pk=comment_id)
    comment.delete()
    return redirect('articles:detail', article_id)
```

### 댓글 목록

1. 34

댓글 삭제하기

Content:

댓글 작성하기

[\[back\]](#)

# 댓글 기능 활용하기

## 댓글이 없는 경우 처리하기

```
<!-- articles/detail.html -->
<h3>댓글 목록</h3>
<ol>
  {% if comments %}
    {% for comment in comments %}
      <li>
        {{ comment.content }}
        <form action="{% url 'articles:delete_comment' article_id=article.pk comment_id=comment.id %}"
              method="POST">
          {% csrf_token %}
          <input type="submit" value="댓글 삭제하기">
        </form>
      </li>
    {% endfor %}
  {% else %}
    <div>댓글이 없습니다.</div>
  {% endif %}
</ol>
```

← → ↺ ⓘ localhost:8000/articles/detail/1/

### DETAIL

#### 1 번째 글

---

작성자: admin  
제목: 기사1  
내용: 기사2

---

#### 댓글 목록

댓글이 없습니다.

Content:

---

[\[back\]](#)

# 댓글 기능 활용하기

## 댓글 목록 옆에 댓글 개수 출력하기

```
<!-- articles/detail.html -->
<h3>댓글 목록 ( {{ comments|length }} )</h3>
<ol>
  {% if comments %}
    {% for comment in comments %}
      <li>
        {{ comment.content }}
        <form action="{% url 'articles:delete_comment' article_id=article.pk comment_id=comment.id %}"
              mehtod="POST">
          {% csrf_token %}
          <input type="submit" value="댓글 삭제하기">
        </form>
      </li>
    {% endfor %}
  {% else %}
    <div>댓글이 없습니다.</div>
  {% endif %}
</ol>
```

← → ↻ ⓘ localhost:8000/articles/detail/1/

---

### DETAIL

#### 1 번째 글

---

작성자: admin  
제목: 기사1  
내용: 기사2

---

#### 댓글 목록 ( 2 )

- 123
- 45

Content:

---

[\[back\]](#)

# Article – User 관계 설정

---

유저를 연결할 때에는 직접적으로 연결하지 않고, 간접 참조 권장  
=> 동작에는 차이가 없지만, 차후 유지보수의 간편함을 위해서

```
from django.conf import settings

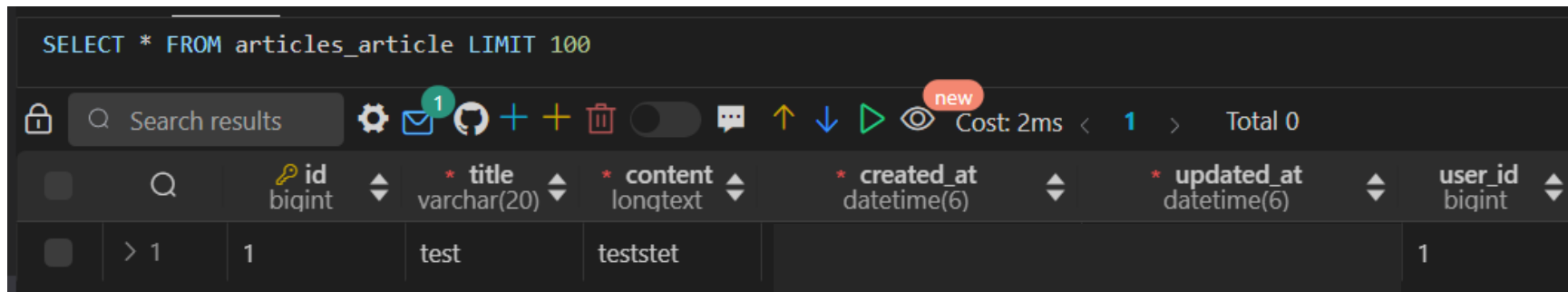
class Article(models.Model):
    # user = models.ForeignKey(User, on_delete=models.CASCADE, related_name="articles")
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="articles")
    title = models.CharField(max_length=20)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```



# Article – User 관계 설정

마이그레이션 진행 시 NOT NULL 조건인 user 모델을 채워줘야 함  
⇒ 설정창에서 1을 입력하여, Article 작성자를 관리자(id=1)로 설정

```
$ python manage.py makemigrations
It is impossible to add a non-nullable field 'user' to article without specifying a default. This is because the database needs something to populate e
xisting 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 manually define a default value in models.py.
Select an option: 1
Please enter the default value as valid Python.
The datetime and django.utils.timezone modules are available, so it is possible to provide e.g. timezone.now as a value.
Type 'exit' to exit this prompt
>>> 1
Migrations for 'articles':
  articles\migrations\0002_article_user.py
    - Add field user to article
```



The screenshot shows a database query interface. At the top, the SQL query is `SELECT * FROM articles_article LIMIT 100`. Below the query, there is a toolbar with various icons for search, settings, and execution. The results are displayed in a table with columns: `id` (bigint), `* title` (varchar(20)), `* content` (longtext), `* created_at` (datetime(6)), `* updated_at` (datetime(6)), and `user_id` (bigint). The first row of results shows `id` as 1, `title` as 'test', `content` as 'testtet', and `user_id` as 1. The `user_id` column is highlighted with a red box.

	<code>id</code> bigint	<code>* title</code> varchar(20)	<code>* content</code> longtext	<code>* created_at</code> datetime(6)	<code>* updated_at</code> datetime(6)	<code>user_id</code> bigint
> 1	1	test	testtet			1

# 게시글 작성 구현

게시글 작성 시 유저 선택창을 없애기 위해서 forms.py 수정

```
# articles/forms.py
class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = ['title', 'content']
        widgets = {
            'title': forms.TextInput(attrs={"placeholder": "제목 입력창"}),
            'content': forms.Textarea(attrs={"class": "my-class"}),
        }
        error_messages = {
            'title': {
                'max_length': '입력 길이를 초과했습니다.',
            }
        }
```

```
# articles/views.py
@login_required
def create(request):
    if request.method == 'POST':
        form = ArticleForm(request.POST)
        if form.is_valid():
            article = form.save(commit=False)
            article.user = request.user
            article.save()
            article = form.save()
            return redirect('articles:detail', article.pk)
    else:
        form = ArticleForm()
        context = {
            'form': form
        }
        return render(request, 'articles/create.html', context)
```

# 게시글 작성자 화면 구현

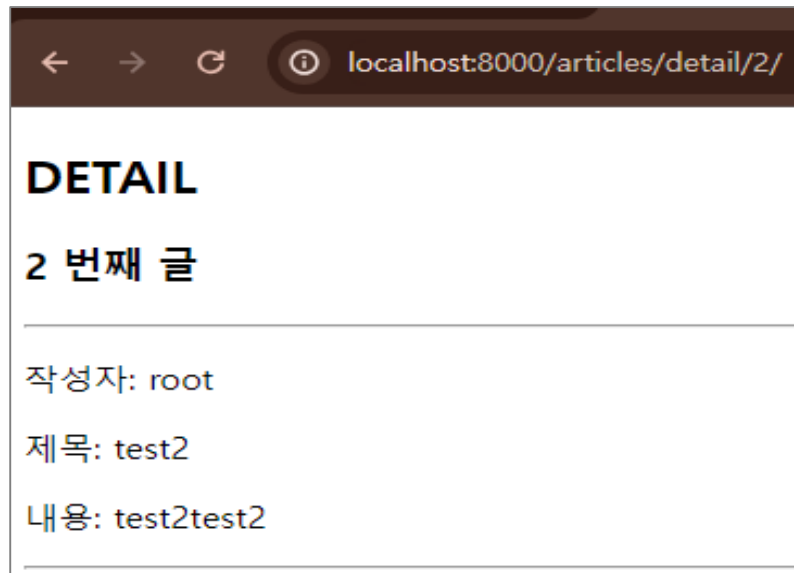
```
<!-- templates/articles/index.html-->
{% extends "articles/base.html" %}

{% block content %}
<h1>Articles</h1>
<a href="{% url 'articles:create' %}">게시글 작성하기</a>
<hr>
{% for article in articles %}
<p>글 번호: {{ article.pk }}</p>
<p>글 작성자: {{ article.user }}</p>
<a href="{% url 'articles:detail' pk=article.pk %}">
  <p>글 제목: {{ article.title }}</p>
</a>
<p>글 내용: {{ article.content }}</p>
<hr>
{% endfor %}
{% endblock content %}
```



```
<!-- templates/articles/detail.html-->
{% extends "articles/base.html" %}

{% block content %}
<h2>DETAIL</h2>
<h3>{{ article.pk }} 번째 글</h3>
<hr>
<p>작성자: {{ article.user }}</p>
<p>제목: {{ article.title }}</p>
<p>내용: {{ article.content }}</p>
<hr>
```



# 게시글 수정 권한 확인

## 작성자만 수정할 수 있도록 수정하기

```
<!-- templates/articles/detail.html-->
{% if request.user == article.user %}
    <a href="{% url 'articles:update' pk=article.pk %}">[수정하기]</a>
    <hr>
    <a href="{% url 'articles:delete' pk=article.pk %}">[삭제하기]</a>
    <hr>
{% endif %}
```

```
# articles/views.py
def update(request, pk):
    article = Article.objects.get(pk=pk)
    if request.user != article.user:
        return redirect('articles:index')

    if request.method == 'POST':
        form = ArticleForm(request.POST, instance=article)
        if form.is_valid():
            form.save()
            return redirect('articles:detail', article.pk)
    else:
        form = ArticleForm(instance=article)
    context = {
        'form': form,
        'article': article
    }
    return render(request, 'articles/edit.html', context)
```

# 게시글 삭제 권한 확인

## 작성자만 삭제할 수 있도록 수정하기

```
<!-- templates/articles/detail.html-->
{% if request.user == article.user %}
    <a href="{% url 'articles:update' pk=article.pk %}">[수정하기]</a>
    <hr>
    <a href="{% url 'articles:delete' pk=article.pk %}">[삭제하기]</a>
    <hr>
{% endif %}
```

```
# articles/views.py
def delete(request, pk):
    article = Article.objects.get(pk=pk)
    if request.user == article.user:
        article.delete()
    return redirect('articles:index')
```

# Comment – User 관계 설정

---

유저는 여러 댓글을 작성할 수 있고, 댓글은 유저 한 명이 작성

```
# articles/models.py
class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE, related_name="comments")
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="comments")
    content = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

# Comment – User 관계 설정

유저는 여러 댓글을 작성할 수 있고, 댓글은 유저 한 명이 작성

```
$ python manage.py makemigrations
It is impossible to add a non-nullable field 'user' to comment without specifying a default. This is because 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 manually define a default value in models.py.
Select an option: 1
Please enter the default value as valid Python.
The datetime and django.utils.timezone modules are available, so it is possible to provide e.g. timezone.now as a value.
Type 'exit' to exit this prompt
>>> 1
Migrations for 'articles':
  articles\migrations\0003_comment_user.py
    - Add field user to comment
```

SELECT \* FROM articles\_comment LIMIT 100

	id bigint	* content varchar(100)	* created_a datetime(6)	* updated_ datetime(6)	* article_id bigint	user_id bigint
> 1	1	test	(NULL)	(NULL)	1	1

# 댓글 작성 로직 수정

---

```
# articles/views.py
def create_comment(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm(request.POST)
    if comment_form.is_valid():
        comment = comment_form.save(commit=False)
        comment.article = article
        comment.user = request.user
        comment.save()
        return redirect('articles:detail', pk)
    context = {
        'article': article,
        'comment_form': comment_form
    }
    return render(request, 'articles/detail.html', context)
```



# 댓글 삭제 권한 추가

```
# aritcles/views.py
def delete_comment(request, article_id, comment_id):
    comment = Comment.objects.get(pk=comment_id)
    if request.user == comment.user:
        comment.delete()
    return redirect('articles:detail', article_id)
```

```
{% if comments %}
    {% for comment in comments %}
        <li>
            {{ comment.content }}
            {% if request.user == comment.user %}
                <form action="{% url 'articles:delete_comment' article_id=article.pk comment_id=comment.id %}"
                    method="POST">
                    {% csrf_token %}
                    <input type="submit" value="댓글 삭제하기">
                </form>
            {% endif %}
        </li>
    {% endfor %}
{% else %}
    <div>댓글이 없습니다.</div>
{% endif %}
```

# Django DB (N:N)

# N:N 관계

A 테이블의 여러 레코드가 B 테이블의 여러 레코드와 연결  
서로 복잡한 상호 작용을 모델링할 때 자주 사용

수강과목 테이블

ID	강의 제목
1	수학
2	국어
3	영어
4	컴퓨터

수강신청 테이블

ID	강의 ID	시간	학생 ID
1	1	10:00	1
2	2	10:00	1
3	3	11:00	2
4	1	11:30	3

학생 테이블

ID	이름	주소	성별
1	김씨	서울	남
2	나씨	경기	여
3	박씨	부산	남
4	이씨	강원	여

# N:N 관계

- 하나의 Article에는 여러 User로부터 좋아요를 받을 수 있음
- 하나의 User는 여러 Article에 좋아요를 누를 수 있음

User Table

ID	Title
1	Admin
2	User_1
3	User_2
4	User_3

User-Article 중계 테이블

ID	강의 ID	학생 ID
1	1	1
2	2	1
3	3	2
4	1	3

Article Table

ID	Title	Content
1	글1	내용1
2	글2	내용2
3	글3	내용3
4	글4	내용4

# ManyToManyField

- N:N 관계를 설정할 때 사용하는 모델 필드 (중계 테이블을 생성)
- 반대편 N은 관계를 설정할 필요가 없음

```
# articles/models.py
class Article(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE, related_name="articles")
    like_users = models.ManyToManyField(settings.AUTH_USER_MODEL, related_name="like_articles", through="ArticleLikeUsers")
    title = models.CharField(max_length=20)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

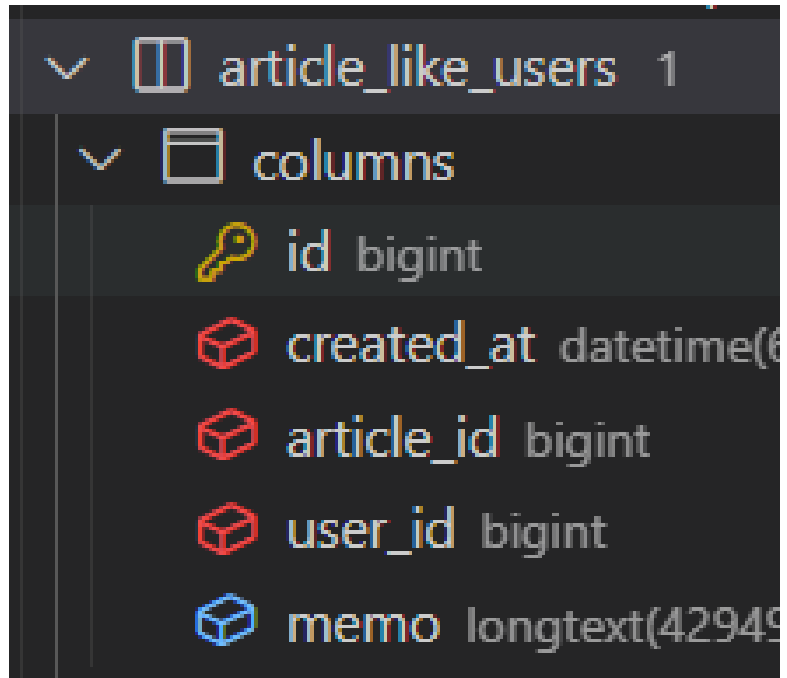
```
# articles/models.py
class ArticleLikeUsers(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    memo = models.TextField(null=True)

    class Meta:
        db_table = 'article_like_users'
```

# 중계 테이블 생성

---

migrations 진행 시 중계 테이블 생성됨



# 게시글에 좋아요 기능 구현

---

게시글에 좋아요/좋아요 취소 함수

1. 게시글에 좋아요한 유저를 참조해서 좋아요를 한 경우에는 취소

```
# articles/views.py
def article_like(request, article_id):
    article = Article.objects.get(pk=article_id)
    # 이미 좋아요를 한 사람의 경우에는 좋아요를 취소
    if request.user in article.like_users.all():
        article.like_users.remove(request.user)
    else:
        article.like_users.add(request.user)
    return redirect('articles:index')
```

# 게시글에 좋아요 기능 구현

---

게시글에 좋아요/좋아요 취소 함수

2. 게시글에 좋아요한 유저 객체 중에 요청한 유저 객체를 제거

```
# articles/views.py
def article_like(request, article_id):
    article = Article.objects.get(pk=article_id)
    # 이미 좋아요를 한 사람의 경우에는 좋아요를 취소
    if request.user in article.like_users.all():
        article.like_users.remove(request.user)
    else:
        article.like_users.add(request.user)
    return redirect('articles:index')
```



# 게시글에 좋아요 기능 구현

---

게시글에 좋아요/좋아요 취소 함수

## 3. 중계 테이블에 게시글-좋아요 유저 데이터 추가

```
# articles/views.py
def article_like(request, article_id):
    article = Article.objects.get(pk=article_id)
    # 이미 좋아요를 한 사람의 경우에는 좋아요를 취소
    if request.user in article.like_users.all():
        article.like_users.remove(request.user)
    else:
        article.like_users.add(request.user)
    return redirect('articles:index')
```

# 게시글에 좋아요 기능 구현

---

게시글에 좋아요/좋아요 취소 함수

## 4. 중계 테이블의 추가 컬럼에 데이터 집어넣기

```
# articles/views.py
def article_like(request, article_id):
    article = Article.objects.get(pk=article_id)
    # 이미 좋아요를 한 사람의 경우에는 좋아요를 취소
    if request.user in article.like_users.all():
        article.like_users.remove(request.user)
    else:
        article.like_users.add(request.user, through_defaults={'memo': '메모'})
    return redirect('articles:index')
```

# 게시글에 좋아요 구현

```
# articles/urls.py
app_name = 'articles'
urlpatterns = [
    path('like/<int:article_id>/', views.article_like, name='article_like'),
    ...
]
```

```
<!-- articles/index.html -->
<form action="{% url 'articles:article_like' article_id=article.pk %}" method="POST">
    {% csrf_token %}
    {% if request.user in article.like_users.all %}
        <input type="submit" value = "좋아요 취소하기">
    {% else %}
        <input type="submit" value = "좋아요">
    {% endif %}
</form>
```

← → ↻ ⓘ localhost:8000/articles/index/

## Articles

[게시글 작성하기](#)

---

글 번호: 1

글 작성자: root

[글 제목: test](#)

글 내용: tset

# 유저 프로필 페이지 구현

---

## 팔로우 기능 구현을 위한 유저 프로필 페이지 구현

```
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('profile/<str:username>', views.profile, name='profile'),
    ...
]
```

```
# accounts/views.py
from django.contrib.auth import get_user_model

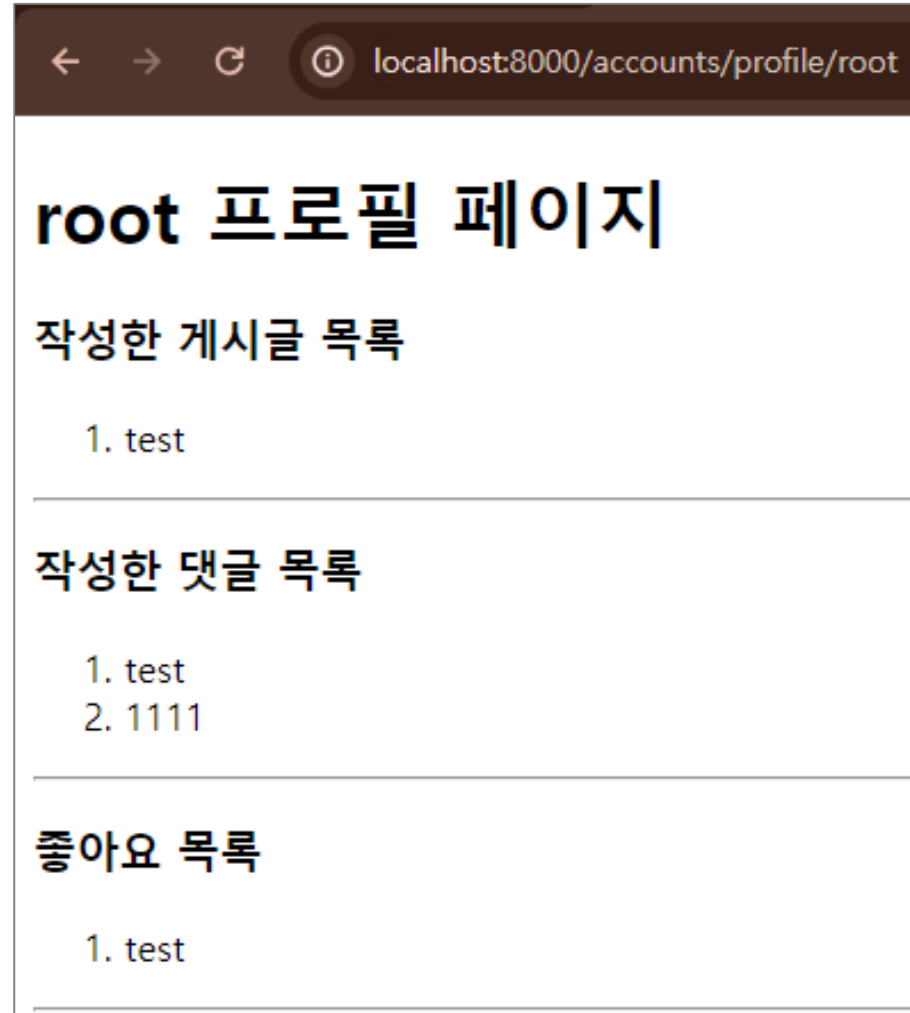
def profile(request, username):
    user = get_user_model().objects.get(username=username)
    context = {
        'user': user
    }
    return render(request, 'accounts/profile.html', context)
```

# 유저 프로필 페이지 구현

```
<!-- accounts/profile.html -->
{% extends "base.html" %}

{% block content %}
  <h1>{{ user.username }} 프로필 페이지 </h1>

  <h3> 작성한 게시글 목록 </h3>
  <ol>
    {% for article in user.articles.all %}
      <li> {{ article.title }}</li>
    {% endfor %}
  </ol><hr>
  <h3> 작성한 댓글 목록 </h3>
  <ol>
    {% for comment in user.comments.all %}
      <li> {{ comment.content }}</li>
    {% endfor %}
  </ol><hr>
  <h3> 좋아요 목록 </h3>
  <ol>
    {% for like_article in user.like_articles.all %}
      <li> {{ like_article.title }}</li>
    {% endfor %}
  </ol><hr>
{% endblock content %}
```

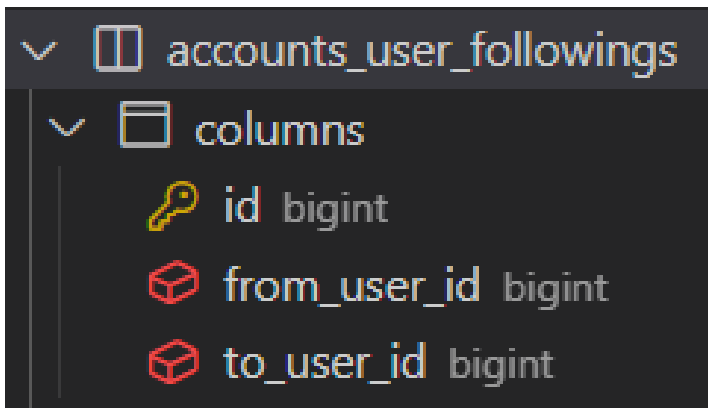


# 팔로우 기능 구현

팔로우는 서로 다른 회원이 서로 다른 회원을 팔로우할 수 있음 (N:N)

- 셀프 조인을 위해서 ManyToManyField 에서 '**self**' 활용
- symmetrical 속성을 이용해서 일방적인 팔로우 구현

```
# accounts/models.py
class User(AbstractUser):
    followings = models.ManyToManyField('self', symmetrical=False, related_name="followers")
```



# 팔로우 기능 구현

---

```
# accounts/urls.py
app_name = 'accounts'
urlpatterns = [
    path('follow/<int:user_id>/', views.follow, name='follow'),
    ...
]
```

```
# accounts/views.py
def follow(request, user_id):
    user = get_user_model().objects.get(pk=user_id)
    # 본인을 팔로우할 수 없음
    if request.user == user:
        return redirect('accounts:profile', user.username)

    # 팔로우한 사람이라면, 팔로우 취소
    if request.user in user.followers.all():
        user.followers.remove(request.user)
    else:
        user.followers.add(request.user)

    return redirect('accounts:profile', user.username)
```

# 팔로우 기능 구현

```
<h1>{{ user.username }} 프로필 페이지 </h1>
<h4>팔로워: {{ user.followers.all|length }}</h3>
<h4>팔로잉: {{ user.follow.all|length }}</h3>
{% if request.user != user%}
    <form action="{% url 'accounts:follow' user_id=user.id %}" method="POST">
        {% csrf_token %}
        {% if request.user in user.followers.all %}
            <input type="submit" value="UnFollow">
        {% else %}
            <input type="submit" value="Follow">
        {% endif %}
    </form>
{% endif %}
```

```
SELECT * FROM accounts_user_followings LIMIT 100
```

Search results				
	id bigint	* from_user_id bigint	* to_user_id bigint	
> 1	1	2	1	





# exists 활용

```
# accounts/views.py
def follow(request, user_id):
    user = get_user_model().objects.get(pk=user_id)
    # 본인을 팔로우할 수 없음
    if request.user == user:
        return redirect('accounts:profile', user.username)

    # 팔로우한 사람이라면, 팔로우 취소
    if request.user in user.followers.all():
        user.followers.remove(request.user)
    else:
        user.followers.add(request.user)

    return redirect('accounts:profile', user.username)
```



```
# accounts/views.py
def follow(request, user_id):
    user = get_user_model().objects.get(pk=user_id)
    # 본인을 팔로우할 수 없음
    if request.user == user:
        return redirect('accounts:profile', user.username)

    # 팔로우한 사람이라면, 팔로우 취소
    if user.followers.filter(pk=request.user.id).exists():
        user.followers.remove(request.user)
    else:
        user.followers.add(request.user)

    return redirect('accounts:profile', user.username)
```

# Django Fixtures

# DB Fixtures

---

미리 준비된 데이터를 활용하는 방법

## 1. dumpdata

- 현재 데이터베이스의 데이터를 추출

## 2. loaddata

- 데이터를 현재 데이터베이스에 삽입

# dumpdata

---

현재 데이터베이스에서 데이터를 추출하는 명령어

```
$ python manage.py dumpdata --indent 4 articles.article > articles.json
```

- dumpdata : 데이터를 추출하는 명령어
- --indent 4 : 데이터를 json 형태로 추출할 때 들여쓰기 간격을 4로 설정
- articles.article: [앱이름].[모델이름] 에 해당하는 데이터를 타겟
- articles.json: 추출한 데이터를 저장할 파일 이름

# dumpdata

## 추출된 json 형태의 데이터 확인

```
{ } articles.json ×
Django 강의자료 > sample_projects > sample_proj_1 > { } articles.json > ...
1  [
2  {
3      "model": "articles.article",
4      "pk": 1,
5      "fields": {
6          "user": 1,
7          "title": "test",
8          "content": "tset"
9      }
10 },
11 {
12     "model": "articles.article",
13     "pk": 2,
14     "fields": {
15         "user": 1,
16         "title": "testtest",
17         "content": "testset"
18     }
19 }
20 ]
```

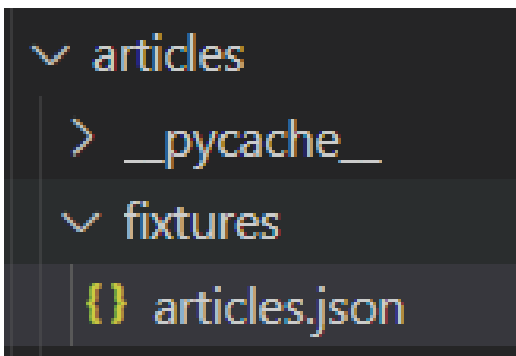
# loaddata

---

Fixtures 데이터를 데이터베이스로 삽입

```
$ python manage.py loaddata articles.json  
Installed 2 object(s) from 1 fixture(s)
```

- loaddata: App/fixtures 경로에 있는 Fixtures 데이터를 DB로 삽입
- 삽입 시에는 꼭 순서를 지켜야 함 ( 모델 관계를 유지하기 위해서 )
- Fixtures 데이터는 아래와 같은 경로에 위치해야 함



# Fixtures 추가 내용

---

## 1. 인코딩 에러가 발생하는 경우

=> -Xutf8 옵션을 추가로 작성

```
$ python -Xutf8 manage.py dumpdata --indent 4 articles.article > articles.json  
$ python -Xutf8 manage.py dumpdata --indent 4 articles.article > articles.json
```

## 2. Fixtures 파일은 가급적 직접 만들지 말 것 (dumpdata로 생성할 것)

⇒ 변환하는 과정에서 버그 발생 확률이 높음

⇒ 다른 형식의 데이터를 삽입할 때는 직접 ORM을 이용해 삽입할 것