

# Database

# INDEX

---

- A many-to-one relationship
- N:1 (Comment – Article)
- N:1 (Article – User)
- N:1 (Comment – User)

**A many-to-one  
relationship**

# A many-to-one relationship

## | 개요

- 관계형 데이터베이스에서의 외래 키 속성을 사용해 모델간 N:1 관계 설정하기

# Intro

## RDB(관계형 데이터베이스) 복습

- 데이터를 테이블, 행, 열 등으로 나누어 구조화하는 방식
- RDB의 모든 테이블에는 행에서 고유하게 식별 가능한 기본 키라는 속성이 있으며, 외래 키를 사용하여 각 행에서 서로 다른 테이블 간의 관계를 만드는 데 사용할 수 있음

## | [참고] 관계 (Relationship)

- 테이블 간의 상호작용을 기반으로 설정되는 여러 테이블 간의 논리적인 연결

## 테이블 간 관계 예시 (1/5)

주문 테이블

| 주문 id |
|-------|
| 제품명   |
| 주문일   |
| 배송일   |
| 주문상태  |

고객 테이블

| 고객 id (primary key) |
|---------------------|
| 이름                  |
| 주소지                 |
| 배송지                 |

- 다음과 같이 어떠한 서비스의 데이터베이스에 고객 테이블과 주문 테이블이 존재
- 고객 테이블에는 고객에 관한 데이터가,  
주문 테이블에는 주문에 관한 거래 정보가 포함됨



## 테이블 간 관계 예시 (2/5)

주문 데이터

| 주문 id | 제품명 | 주문일        | 배송일        | 주문상태   |
|-------|-----|------------|------------|--------|
| 1     | 생수  | 2000-01-01 | 2000-01-03 | 배송중    |
| 2     | 영양제 | 2000-01-02 | 2000-01-07 | 배송 준비중 |
| 3     | 음료수 | 2000-01-03 | 2000-01-05 | 배송중    |

- 만약 고객들이 특정 제품을 주문한다면 주문 테이블에 레코드가 생성됨
- 그런데 해당 주문이 올바르게 배송되기 위해서는 어떤 고객이 주문 했는지를 알아야 함
  - 즉, 배송지 주소를 가지고 있는 고객 테이블의 정보를 포함해야 함
- 주문 테이블에서 어떻게 고객 테이블 정보를 포함 할 수 있을까?

## 테이블 간 관계 예시 (3/5)

주문 데이터

| 주문 id | 제품명 | 주문일        | 배송일        | 주문상태   | 고객정보 |
|-------|-----|------------|------------|--------|------|
| 1     | 생수  | 2000-01-01 | 2000-01-03 | 배송중    | 김진수  |
| 2     | 영양제 | 2000-01-02 | 2000-01-07 | 배송 준비중 | 박영희  |
| 3     | 음료수 | 2000-01-03 | 2000-01-05 | 배송중    | 김진수  |

- 각 주문데이터에 고객 정보를 입력하는 방법이 있음
- 하지만 이렇게 이름으로 저장할 경우 이름이 같은 다른 사용자를 구분할 수 없음
- 그렇다면 고객 정보의 어떤 데이터를 사용하는 것이 적합할까?

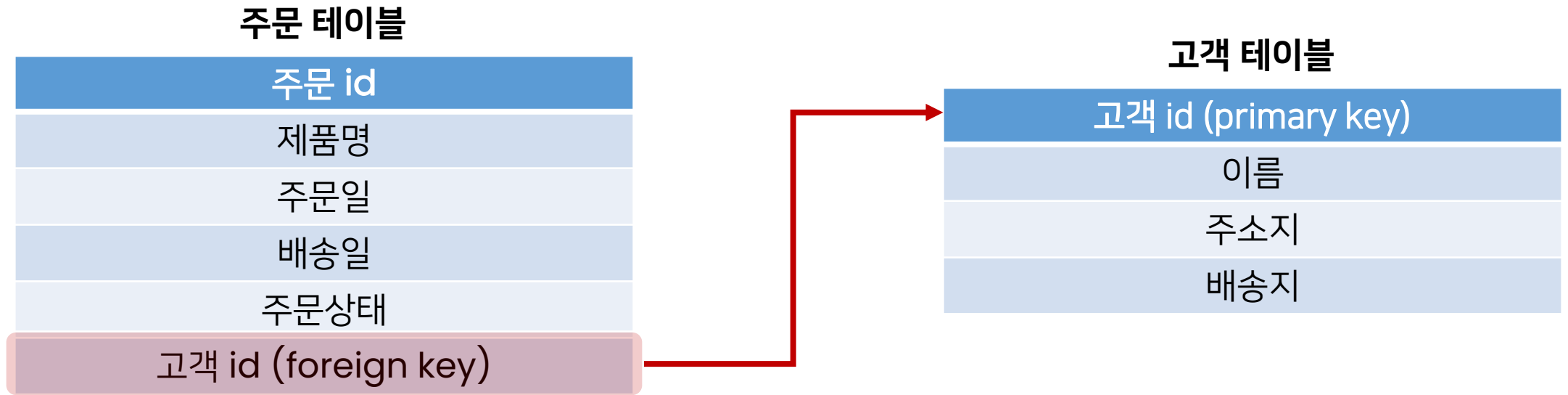
## 테이블 간 관계 예시 (4/5)

주문 데이터

| 주문 id | 제품명 | 주문일        | 배송일        | 주문상태   | 고객 id |
|-------|-----|------------|------------|--------|-------|
| 1     | 생수  | 2000-01-01 | 2000-01-03 | 배송중    | 2     |
| 2     | 영양제 | 2000-01-02 | 2000-01-07 | 배송 준비중 | 1     |
| 3     | 음료수 | 2000-01-03 | 2000-01-05 | 배송중    | 2     |

- 고객 정보의 기본 키인 고객 id 정보를 저장하는 방법이 있음
- 이처럼 관계형 데이터베이스에서 한 테이블의 필드 중 다른 테이블의 행을 식별할 수 있는 키를 외래 키(foreign key, FK)라 함

## 테이블 간 관계 예시 (5/5)



- 이렇게 되면 이 두 테이블은 공유된 고객 id를 기반으로 연결되며 다양한 명령 처리를 진행할 수 있음
  - 특정 날짜에 구매한 모든 고객 정보 확인하기
  - 지난 달에 배송이 지연된 주문을 받은 고객 처리하기
  - 특정 고객이 주문한 모든 주문 정보 조회하기... 등
- 실제 상황보다는 간단한 예시지만 이처럼 RDB는 데이터 간의 매우 복잡한 관계를 보여주고 처리하는 데 탁월한 방식

## RDB에서의 관계

### 1. 1:1

- One-to-one relationships
- 한 테이블의 레코드 하나가 다른 테이블의 레코드 단 한 개와 관련된 경우

### 2. N:1

- Many-to-one relationships
- 한 테이블의 0개 이상의 레코드가 다른 테이블의 레코드 한 개와 관련된 경우
- 기준 테이블에 따라(1:N, One-to-many relationships)이라고도 함

### 3. M:N

- Many-to-many relationships
- 한 테이블의 0개 이상의 레코드가 다른 테이블의 0개 이상의 레코드와 관련된 경우
- 양쪽 모두에서 N:1 관계를 가짐

❖ M:N에 대한 자세한 내용은 N:1 이후 진행

## Many-to-one relationships 예시

주문 데이터

| 주문 id | 제품명 | 주문일        | 배송일        | 주문상태   | 고객 id |
|-------|-----|------------|------------|--------|-------|
| 1     | 생수  | 2000-01-01 | 2000-01-03 | 배송중    | 2     |
| 2     | 영양제 | 2000-01-02 | 2000-01-07 | 배송 준비중 | 1     |
| 3     | 음료수 | 2000-01-03 | 2000-01-05 | 배송중    | 2     |

고객 데이터

| 고객 id | 이름  | 주소지 | 배송지 |
|-------|-----|-----|-----|
| 1     | 김진수 | 경기  | 경기  |
| 2     | 박영희 | 부산  | 강원  |

- 여러 개의 주문 입장에서 각각 어떤 주문에 속해 있는지 표현해야 하므로  
고객 테이블의 PK를 주문 테이블에 FK로 집어 넣어 관계를 표현
  - 고객(1)은 여러 주문(N)을 진행할 수 있음
- ❖ 만약 고객이 단 한 개의 주문만 생성할 수 있다면 두 테이블은 1:1 관계라 할 수 있음

# Foreign Key

## | 개념

- 외래 키(외부 키)
- 관계형 데이터베이스에서 다른 테이블의 행을 식별할 수 있는 키
- 참조되는 테이블의 기본 키(Primary Key)를 가리킴
- 참조하는 테이블의 행 1개의 값은, 참조되는 측 테이블의 행 값에 대응됨
  - 이 때문에 참조하는 테이블의 행에는,  
참조되는 테이블에 나타나지 않는 값을 포함할 수 없음
- 참조하는 테이블 행 여러 개가, 참조되는 테이블의 동일한 행을 참조할 수 있음



## | 특징

- 키를 사용하여 부모 테이블의 유일한 값을 참조 (by 참조 무결성)
- 외래 키의 값이 반드시 부모 테이블의 기본 키 일 필요는 없지만 유일한 값이어야 함

## [참고] 참조 무결성

- 데이터베이스 관계 모델에서 관련된 2개의 테이블 간의 일관성을 말함
- 외래 키가 선언된 테이블의 외래 키 속성(열)의 값은 그 테이블의 부모가 되는 테이블의 기본 키 값으로 존재해야 함

# 이어서 ..

삼성 청년 SW 아카데미

# N:1 (Comment - Article)

# N:1 (Comment - Article)

## | 개요

- Comment(N) - Article(1)
- Comment 모델과 Article 모델 간 관계 설정
- “0개 이상의 댓글은 1개의 게시글에 작성 될 수 있음”

# 모델 관계 설정

## | 모델 관계 설정 (1/3)

- 게시판의 게시글과 N:1 관계를 나타낼 수 있는 댓글 구현
- N:1 관계에서 댓글을 담당할 Comment 모델은 N, Article 모델은 1이 될 것

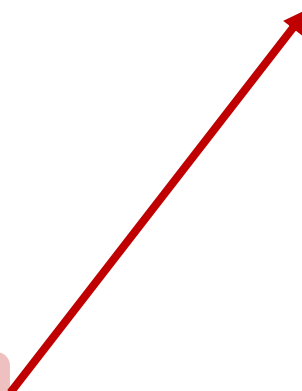
## | 모델 관계 설정 (2/3)

Comment

| id          |
|-------------|
| content     |
| created_at  |
| updated_at  |
| Article의 id |

Article

| id         |
|------------|
| title      |
| content    |
| created_at |
| updated_at |





## 모델 관계 설정 (3/3)

### Comment 데이터 예시

| id | content | created_at | updated_at | Article의 id |
|----|---------|------------|------------|-------------|
| 1  | 댓글 내용 1 | ..         | ..         | 1           |
| 2  | 댓글 내용 2 | ..         | ..         | 3           |
| 3  | 댓글 내용 3 | ..         | ..         | 3           |

- 만약 comment 테이블에 데이터가 다음과 같이 작성되었다면  
1번 게시글에는 1개의 댓글이, 3번 게시글에는 2개의 댓글이 작성되어 있다고  
볼 수 있음

# Django Relationship fields

# Django Relationship fields

## | Django Relationship fields 종류

### 1. OneToOneField()

- A one-to-one relationship

### 2. ForeignKey()

- A many-to-one relationship

### 3. ManyToManyField()

- A many-to-many relationship

# Django Relationship fields

## | ForeignKey(to, on\_delete, \*\*options)

- A many-to-one relationship을 담당하는 Django의 모델 필드 클래스
- Django 모델에서 관계형 데이터베이스의 외래 키 속성을 담당
- 2개의 필수 위치 인자가 필요
  1. 참조하는 model class
  2. on\_delete 옵션
- <https://docs.djangoproject.com/en/3.2/ref/models/fields/#foreignkey>

# Comment Model

## Comment 모델 정의

```
# articles/models.py

class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    content = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

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

- 외래 키 필드는 ForeignKey 클래스를 작성하는 위치와 관계없이 필드의 마지막에 작성됨
- ForeignKey() 클래스의 인스턴스 이름은 참조하는 모델 클래스 이름의 단수형(소문자)으로 작성하는 것을 권장 (이유는 이어지는 모델 참조에서 확인 예정)

## ForeignKey arguments - on\_delete

- 외래 키가 참조하는 객체가 사라졌을 때,  
외래 키를 가진 객체를 어떻게 처리할 지를 정의
- 데이터 무결성을 위해서 매우 중요한 설정
- on\_delete 옵션 값
  - **CASCADE**: 부모 객체(참조 된 객체)가 삭제 됐을 때 이를 참조하는 객체도 삭제
  - PROTECT, SET\_NULL, SET\_DEFAULT ... 등 여러 옵션 값들이 존재
  - 수업에서는 CASCADE 값만 사용할 예정

## [참고] 데이터 무결성 (Data Integrity)

- 데이터의 정확성과 일관성을 유지하고 보증하는 것
- 데이터베이스나 RDBMS의 중요한 기능
- 무결성 제한의 유형
  1. 개체 무결성 (Entity integrity)
  2. 참조 무결성 (Referential integrity)
  3. 범위 무결성 (Domain integrity)
- [https://en.wikipedia.org/wiki/Data\\_integrity](https://en.wikipedia.org/wiki/Data_integrity)



## Migration 과정 진행 (1/2)

1. models.py에서 모델에 대한 수정사항이 발생했기 때문에 migration 과정을 진행

```
$ python manage.py makemigrations
```

2. 마이그레이션 파일 **0002\_comment.py** 생성 확인

3. migrate 진행

```
$ python manage.py migrate
```

## Migration 과정 진행 (2/2)

- migrate 후 Comment 모델 클래스로 인해 생성된 테이블 확인



A screenshot of a database table structure for 'articles\_comment'. The table is expanded, showing its fields. The first field is 'id' of type 'integer' and is marked as the primary key with a key icon. The other fields are 'content' (varchar(200)), 'created\_at' (datetime), 'updated\_at' (datetime), and 'article\_id' (bigint), all marked with diamond icons.

| articles_comment       |
|------------------------|
| id : integer           |
| content : varchar(200) |
| created_at : datetime  |
| updated_at : datetime  |
| article_id : bigint    |

- ForeignKey 모델 필드로 인해 작성된 컬럼의 이름이 **article\_id**인 것을 확인
- 만약 ForeignKey 인스턴스를 article이 아닌 abcd로 생성 했다면 abcd\_id로 만들어짐
  - 이처럼 명시적인 모델 관계 파악을 위해 참조하는 클래스 이름의 소문자(단수형)로 작성하는 것이 권장 되었던 이유

## | 댓글 생성 연습하기 (1/7)

- shell\_plus 실행

```
$ python manage.py shell_plus
```

## 댓글 생성 연습하기 (2/7)

### 1. 댓글 생성

```
# Comment 클래스의 인스턴스 comment 생성
comment = Comment()

# 인스턴스 변수 저장
comment.content = 'first comment'

# DB에 댓글 저장
comment.save()

# 에러 발생
django.db.utils.IntegrityError: NOT NULL constraint failed: articles_comment.article_id
# articles_comment 테이블의 ForeignKeyField, article_id 값이 저장시 누락되었기 때문
```

## 댓글 생성 연습하기 (3/7)

### 1. 댓글 생성

```
# 게시글 생성 및 확인
article = Article.objects.create(title='title', content='content')
article
=> <Article: title>

# 외래 키 데이터 입력
# 다음과 같이 article 객체 자체를 넣을 수 있음
comment.article = article
# 또는 comment.article_id = article.pk 처럼 pk 값을 직접 외래 키 컬럼에
# 넣어 줄 수도 있지만 권장하지 않음

# DB에 댓글 저장 및 확인
comment.save()
comment
=> <Comment: first comment>
```

## 댓글 생성 연습하기 (4/7)

### 2. 댓글 속성 값 확인

```
comment.pk
=> 1

comment.content
=> 'first comment'

# 클래스 변수명인 article로 조회 시 해당 참조하는 게시물 객체를 조회할 수 있음
comment.article
=> <Article: title>

# article_pk는 존재하지 않는 필드이기 때문에 사용 불가
comment.article_id
=> 1
```

## | 댓글 생성 연습하기 (5/7)

### 3. comment 인스턴스를 통한 article 값 접근하기

```
# 1번 댓글이 작성된 게시물의 pk 조회  
comment.article.pk  
=> 1
```

```
# 1번 댓글이 작성된 게시물의 content 조회  
comment.article.content  
=> 'content'
```

## | 댓글 생성 연습하기 (6/7)

### 4. 두번째 댓글 작성해보기

```
comment = Comment(content='second comment', article=article)
comment.save()

comment.pk
=> 2

comment
=> <Comment: second comment>

comment.article_id
=> 1
```



## 댓글 생성 연습하기 (7/7)

- 작성된 댓글 확인 해보기

| id | content        | created_at                 | updated_at                 | article_id |
|----|----------------|----------------------------|----------------------------|------------|
| 1  | first comment  | 2022-09-30 01:39:19.005005 | 2022-09-30 01:39:19.005040 | 1          |
| 2  | second comment | 2022-09-30 01:42:25.177322 | 2022-09-30 01:42:25.177356 | 1          |

# 관계 모델 참조

## Related manager

- Related manager는 N:1 혹은 M:N 관계에서 사용 가능한 문맥(context)
- Django는 모델 간 N:1 혹은 M:N 관계가 설정되면 역참조할 때에 사용할 수 있는 manager를 생성
  - 우리가 이전에 모델 생성 시 **objects**라는 매니저를 통해 queryset api를 사용했던 것처럼 related manager를 통해 queryset api를 사용할 수 있게 됨
- 지금은 N:1 관계에서의 related manager 만을 학습할 것
- <https://docs.djangoproject.com/en/3.2/ref/models/relations/>

## | 역참조

- 나를 참조하는 테이블(나를 외래 키로 지정한)을 참조하는 것
- 즉, 본인을 외래 키로 참조 중인 다른 테이블에 접근하는 것
- N:1 관계에서는 1이 N을 참조하는 상황
  - 외래 키를 가지지 않은 1이 외래 키를 가진 N을 참조

## `article.comment_set.method()`

- Article 모델이 Comment 모델을 참조(역참조)할 때 사용하는 매니저
- `article.comment` 형식으로는 댓글 객체를 참조 할 수 없음
  - 실제로 Article 클래스에는 Comment와의 어떠한 관계도 작성되어 있지 않음
- 대신 Django가 역참조 할 수 있는 `comment_set` manager를 자동으로 생성해 `article.comment_set` 형태로 댓글 객체를 참조할 수 있음
  - ❖ N:1 관계에서 생성되는 Related manager의 이름은 참조하는 “모델명\_set” 이름 규칙으로 만들어짐
- 반면 참조 상황(`Comment → Article`)에서는 실제 ForeignKey 클래스로 작성한 인스턴스가 Comment 클래스의 클래스 변수이기 때문에 `comment.article` 형태로 작성 가능

## | Related manager 연습하기 (1/5)

- shell\_plus 실행

```
$ python manage.py shell_plus
```

## | Related manager 연습하기 (2/5)

### 1. 1번 게시글 조회하기

```
article = Article.objects.get(pk=1)
```

## Related manager 연습하기 (3/5)

2. dir() 함수를 사용해 클래스 객체가 사용할 수 있는 메서드를 확인하기

```
dir(article)

[
    ...중략...,
    'comment_set',
    'content',
    'created_at',
    'date_error_message',
    'delete',
    'from_db',
    'full_clean',
    ...중략...
]
```



## | Related manager 연습하기 (4/5)

3. 1번 게시글에 작성된 모든 댓글 조회하기 (역참조)

```
article.comment_set.all()  
=> <QuerySet [<Comment: first comment>, <Comment: second comment>]>
```

## | Related manager 연습하기 (5/5)

### 4. 1번 게시글에 작성된 모든 댓글 출력하기

```
comments = article.comment_set.all()

for comment in comments:
    print(comment.content)
```

## ForeignKey arguments – `related_name`

```
# articles/models.py

class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE, related_name='comments')
    ...
```

- ForeignKey 클래스의 선택 옵션
- 역참조 시 사용하는 매니저 이름(model\_set manager)을 변경할 수 있음
- 작성 후, migration 과정이 필요
- 선택 옵션이지만 상황에 따라 반드시 작성해야 하는 경우가 생기기도 하는데  
이는 추후 자연스럽게 만나볼 예정
- 작성 후 다시 원래 코드로 복구
  - ❖ 위와 같이 변경 하면 기존 `article.comment_set`은 더 이상 사용할 수 없고, `article.comments`로 대체됨

## | admin site 등록

- 새로 작성한 Comment 모델을 admin site에 등록하기

```
# articles/admin.py  
  
from .models import Article, Comment  
  
admin.site.register(Article)  
admin.site.register(Comment)
```

# Comment 구현

## | CREATE (1/9)

- 사용자로부터 댓글 데이터를 입력 받기 위한 CommentForm 작성

```
# articles/forms.py

from .models import Article, Comment

class CommentForm(forms.ModelForm):

    class Meta:
        model = Comment
        fields = '__all__'
```

## CREATE (2/9)

- detail 페이지에서 CommentForm 출력 (view 함수)

```
# 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)
```

❖ 기존에 ArticleForm 클래스의 인스턴스명을 form으로 작성했기 때문에 헷갈리지 않도록 comment\_form으로 작성

## CREATE (3/9)

- detail 페이지에서 CommentForm 출력 (템플릿)

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
...
<a href="{% url 'articles:index' %}">back</a>
<hr>
<form action="#" method="POST">
  {% csrf_token %}
  {{ comment_form }}
  <input type="submit">
</form>
{% endblock content %}
```



## CREATE (4/9)

Article:  Content:

- detail 페이지에 출력된 CommentForm을 살펴보면 다음과 같이 출력됨
- 실 서비스에서는 댓글을 작성할 때 댓글을 어떤 게시글에 작성하는지 직접 게시글 번호를 선택하지 않음
- 실제로는 해당 게시글에 댓글을 작성하면 자연스럽게 그 게시글에 댓글이 작성되어야 함
- 다음과 같이 출력되는 이유는 Comment 클래스의 외래 키 필드 article 또한 데이터 입력이 필요하기 때문에 출력 되고 있는 것
- 하지만, 외래 키 필드는 **사용자의 입력으로 받는 것이 아니라 view 함수 내에서 받아 별도로 처리되어 저장**되어야 함

## CREATE (5/9)

- 외래 키 필드를 출력에서 제외 후 확인

```
# articles/forms.py

class CommentForm(forms.ModelForm):

    class Meta:
        model = Comment
        exclude = ('article',)
```

Content:

Submit

## CREATE (6/9)

- 출력에서 제외된 외래 키 데이터는 어디서 받아와야 할까?
- detail 페이지의 url을 살펴보면 `path('<int:pk>/', views.detail, name='detail')`  
url에 해당 게시글의 pk 값이 사용 되고 있음
- 댓글의 외래 키 데이터에 필요한 정보가 바로 게시글의 pk 값
- 이전에 학습했던 url을 통해 변수를 넘기는 `variable routing`을 사용

### CREATE (7/9)

```
# articles/urls.py

urlpatterns = [
    ...,
    path('<int:pk>/comments/', views.comments_create,
        name='comments_create'),
]
```

```
<!-- articles/detail.html -->

<form action="{% url 'articles:comments_create'
article.pk %}" method="POST">
    {% csrf_token %}
    {{ comment_form }}
    <input type="submit">
</form>
```

```
# articles/views.py

def comments_create(request, pk):
    article = Article.objects.get(pk=pk)
    comment_form = CommentForm(request.POST)
    if comment_form.is_valid():
        # article 객체는 언제 저장할 수 있을까?
        comment_form.save()
    return redirect('articles:detail', article.pk)
```

- 작성을 마치고 보면 article 객체 저장이 이루어질 타이밍이 보이지 않음
- 그래서 save() 메서드는 데이터베이스에 저장하기 전에 객체에 대한 추가적인 작업을 진행할 수 있도록 인스턴스만을 반환해주는 옵션 값을 제공

## | The `save()` method

- `save(commit=False)`
  - “Create, but don't save the new instance.”
  - 아직 데이터베이스에 저장되지 않은 인스턴스를 반환
  - 저장하기 전에 객체에 대한 사용자 지정 처리를 수행할 때 유용하게 사용
- <https://docs.djangoproject.com/en/3.2/topics/forms/modelforms/#the-save-method>

## CREATE (8/9)

- save 메서드의 commit 옵션을 사용해 DB에 저장되기 전 article 객체 저장하기

```
# articles/views.py

def comments_create(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', article.pk)
```

## CREATE (9/9)

- 댓글 작성 후 테이블 확인

| id | content        | created_at                 | updated_at                 | article_id |
|----|----------------|----------------------------|----------------------------|------------|
| 1  | first comment  | 2022-09-30 01:39:19.005005 | 2022-09-30 01:39:19.005040 | 1          |
| 2  | second comment | 2022-09-30 01:42:25.177322 | 2022-09-30 01:42:25.177356 | 1          |
| 3  | 댓글 작성          | 2022-09-30 02:46:34.279533 | 2022-09-30 02:46:34.279587 | 1          |

## READ (1/3)

- 작성한 댓글 목록 출력하기
- 특정 article에 있는 모든 댓글을 가져온 후 context에 추가

```
# articles/views.py

from .models import Article, Comment

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



## READ (2/3)

- detail 템플릿에서 댓글 목록 출력하기

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
...
<a href="{% url 'articles:index' %}">back</a>
<hr>
<h4>댓글 목록</h4>
<ul>
  {% for comment in comments %}
    <li>{{ comment.content }}</li>
  {% endfor %}
</ul>
<hr>
...
{% endblock content %}
```

## READ (3/3)

- detail 템플릿에서 댓글 목록 출력 확인하기

### 댓글 목록

- first comment
- second comment
- 댓글 작성

---

Content:

## DELETE (1/3)

- 댓글 삭제 구현하기 (url, view)

```
# articles/urls.py

urlpatterns = [
    ...,
    path('<int:article_pk>/comments/<int:comment_pk>/delete/', views.comments_delete, name='comments_delete'),
]
```

```
# articles/views.py

def comments_delete(request, article_pk, comment_pk):
    comment = Comment.objects.get(pk=comment_pk)
    comment.delete()
    return redirect('articles:detail', article_pk)
```

## DELETE (2/3)

- 댓글을 삭제할 수 있는 버튼을 각각의 댓글 옆에 출력 될 수 있도록 함

```
<!-- articles/detail.html -->

{% block content %}
...
<h4>댓글 목록</h4>
<ul>
  {% for comment in comments %}
    <li>
      {{ comment.content }}
      <form action="{% url 'articles:comments_delete' article.pk comment.pk %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="DELETE">
      </form>
    </li>
  {% endfor %}
</ul>
<hr>
...
{% endblock content %}
```

## DELETE (3/3)

- 댓글 삭제 버튼 출력 확인 및 삭제 시도해보기

### 댓글 목록

- first comment
- second comment
- 댓글 작성

---

Content:

## | 댓글 수정을 지금 구현하지 않는 이유

- 댓글 수정도 게시글 수정과 마찬가지로 구현이 가능
  - 게시글 수정 페이지가 필요했던 것처럼 댓글 수정 페이지가 필요하게 됨
- 하지만 일반적으로 댓글 수정은 수정 페이지로 이동 없이 현재 페이지가 유지된 상태로 댓글 작성 Form 부분만 변경되어 수정 할 수 있도록 함
- 이처럼 페이지의 일부 내용만 업데이트 하는 것은 JavaScript의 영역이기 때문에 JavaScript를 학습한 후 별도로 진행하도록 함

# Comment 추가 사항

## | 개요

- 댓글에 관련된 아래 2가지 사항을 작성하면서 마무리하기
  1. 댓글 개수 출력하기
    1. DTL filter - `length` 사용
    2. Queryset API - `count()` 사용
  2. 댓글이 없는 경우 대체 콘텐츠 출력하기



## 댓글 개수 출력하기 (1/3)

1. DTL filter - `length` 사용

```
{{ comments|length }}  
  
{{ article.comment_set.all|length }}
```

2. Queryset API - `count()` 사용

```
{{ comments.count }}  
  
{{ article.comment_set.count }}
```

## 댓글 개수 출력하기 (2/3)

- detail 템플릿에 작성하기

```
<!-- articles/detail.html -->

<h4>댓글 목록</h4>
{% if comments %}
    <p><b>{{ comments|length }}개의 댓글이 있습니다.</b></p>
{% endif %}
```

## 댓글 개수 출력하기 (3/3)

- 작성 후 출력 확인

### 댓글 목록

2개의 댓글이 있습니다.

- first comment
- second comment

---

Content:

## 댓글이 없는 경우 대체 콘텐츠 출력하기 (1/2)

- DTL **for empty** 활용하기

```
<!-- articles/detail.html -->

{% for comment in comments %}
  <li>
    {{ comment.content }}
    <form action="{% url 'articles:comments_delete' article.pk comment.pk %}" method="POST">
      {% csrf_token %}
      <input type="submit" value="DELETE">
    </form>
  </li>
  {% empty %}
    <p>댓글이 없어요..</p>
  {% endfor %}
```

## | 댓글이 없는 경우 대체 콘텐츠 출력하기 (2/2)

- 새로운 게시글을 작성하거나 모든 댓글을 삭제 후 확인

### 댓글 목록

댓글이 없어요..

Content:

Submit

# 이어서 ..

삼성 청년 SW 아카데미

# N:1 (Article - User)

# N:1 (Article - User)

## | 개요

- Article(N) - User(1)
- Article 모델과 User 모델 간 관계 설정
- “0개 이상의 게시글은 1개의 회원에 의해 작성 될 수 있음”



## Referencing the User model

# Referencing the User model

## | Django에서 User 모델을 참조하는 방법 (1/3)

1. `settings.AUTH_USER_MODEL`
2. `get_user_model()`

## | Django에서 User 모델을 참조하는 방법 (2/3)

### 1. `settings.AUTH_USER_MODEL`

- 반환 값 : 'accounts.User' (문자열)
- User 모델에 대한 외래 키 또는 M:N 관계를 정의 할 때 사용
- `models.py`의 모델 필드에서 User 모델을 참조할 때 사용

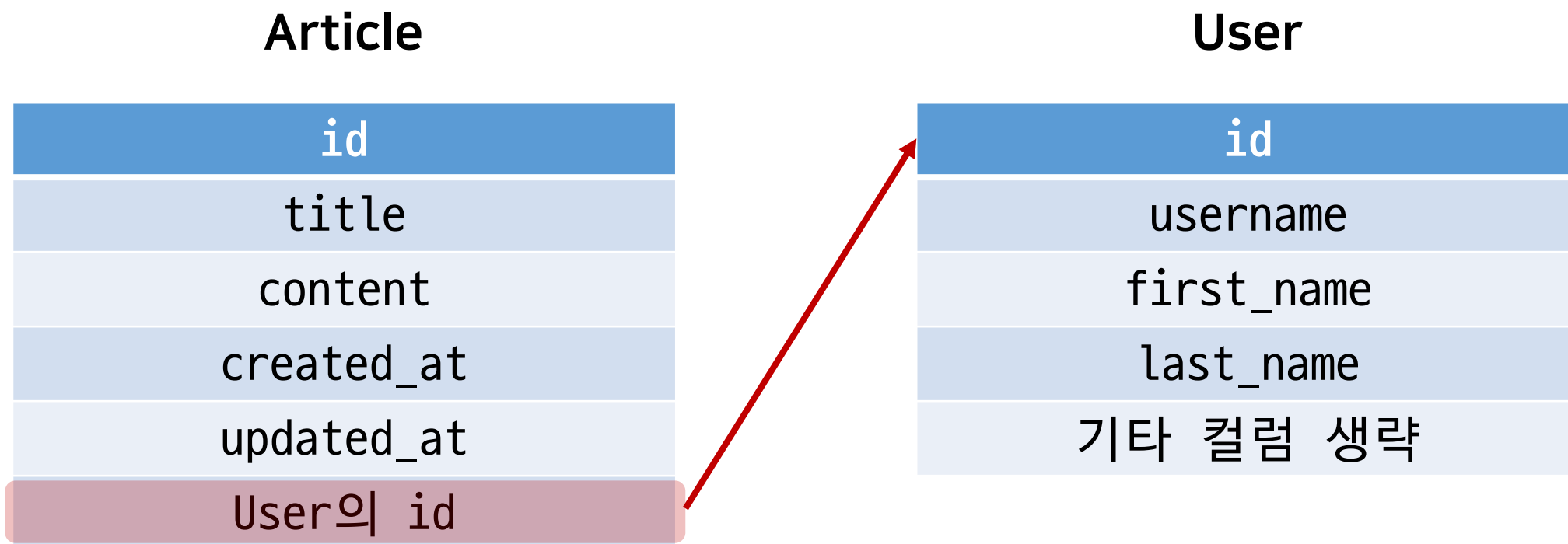
## | Django에서 User 모델을 참조하는 방법 (3/3)

### 2. `get_user_model()`

- 반환 값 : User Object (객체)
- 현재 활성화(active)된 User 모델을 반환
- 커스터마이징한 User 모델이 있을 경우는 Custom User 모델, 그렇지 않으면 User를 반환
- `models.py`가 아닌 다른 모든 곳에서 유저 모델을 참조할 때 사용

# 모델 관계 설정

## Article과 User간 모델 관계 설정 (1/2)



## | Article과 User간 모델 관계 설정 (2/2)

- Article 모델에 User 모델을 참조하는 외래 키 작성

```
# articles/models.py

from django.conf import settings

class Article(models.Model)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    ...
```

## | Migration 진행 (1/5)

- 기존에 존재하던 테이블에 새로운 컬럼이 추가되어야 하는 상황이기 때문에 migrations 파일이 곧바로 만들어지지 않고 일련의 과정이 필요

```
$ python manage.py makemigrations
```



## Migration 진행 (2/5)

```
You are trying to add a non-nullable field 'user' to article without a default; we can't do that (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 let me add a default in models.py
```

```
Select an option:
```

- 첫번째 화면
  - 기본적으로 모든 컬럼은 NOT NULL 제약조건이 있기 때문에 데이터가 없는 새로 추가되는 외래 키 필드 user\_id가 생성되지 않음
  - 그래서 기본값을 어떻게 작성할 것인지 선택해야 함
  - 1을 입력하고 Enter 진행 (다음 화면에서 직접 기본 값 입력)

## Migration 진행 (3/5)

```
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g.
timezone.now
Type 'exit' to exit this prompt
>>>
```

- 두번째 화면
  - article의 user\_id에 어떤 데이터를 넣을 것인지 직접 입력해야 함
  - 마찬가지로 1 입력하고 Enter 진행
  - 그러면 기존에 작성된 게시글이 있다면 모두 1번 회원이 작성한 것으로 처리됨


## | Migration 진행 (4/5)

- migrations 파일 생성 후 migrate 진행

```
$ python manage.py migrate
```

## Migration 진행 (5/5)

- article 테이블 스키마 변경 및 확인

▼  articles\_article

- 🔑 id : integer
- ◆ title : varchar(10)
- ◆ content : text
- ◆ created\_at : datetime
- ◆ updated\_at : datetime
- ◆ user\_id : bigint

| id | title | content | created_at                 | updated_at                 | user_id |
|----|-------|---------|----------------------------|----------------------------|---------|
| 1  | title | content | 2022-09-30 01:39:02.150657 | 2022-09-30 01:39:02.150707 | 1       |
| 2  | 제목    | 내용      | 2022-09-30 05:54:05.241149 | 2022-09-30 05:54:05.241353 | 1       |

## Django에서 User 모델을 참조하는 방법 정리

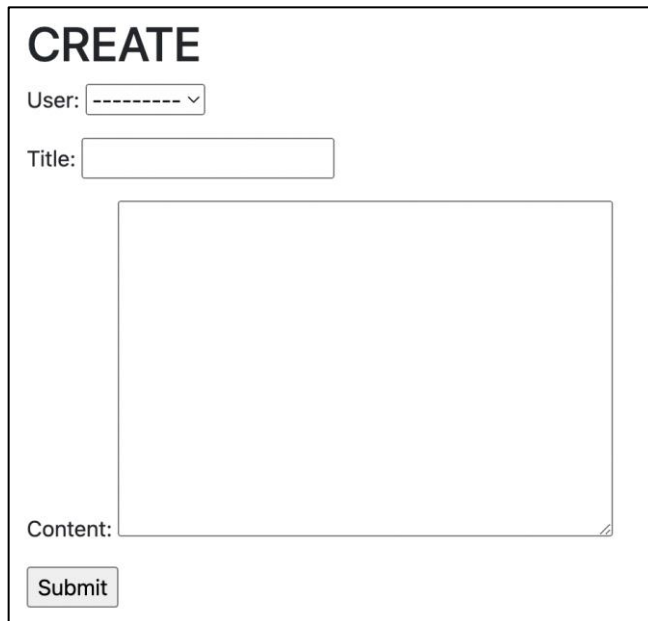
- 문자열과 객체를 반환하는 특징과 Django의 내부적인 실행 원리에 관련된 것이므로 이렇게만 외우도록 한다.
- User 모델을 참조할 때
  - `models.py`에서는 `settings.AUTH_USER_MODEL`
  - 다른 모든 곳에서는 `get_user_model()`

**CREATE**

## | 개요

- 인증된 회원의 게시글 작성 구현하기
- 작성하기 전 로그인을 먼저 진행한 상태로 진행

## ArticleForm (1/3)



CREATE

User:

Title:

Content:

Submit

- ArticleForm 출력을 확인해보면 create 템플릿에서 불필요한 필드(user)가 출력됨
- 이전에 CommentForm에서 외래 키 필드 article이 출력되는 상황과 동일한 상황
- user 필드에 작성해야 하는 user 객체는 view 함수의 request 객체를 활용해야 함



## | ArticleForm (2/3)

- ArticleForm의 출력 필드 수정

```
# articles/forms.py

class ArticleForm(forms.ModelForm):

    class Meta:
        model = Article
        fields = ('title', 'content',)
```

## ArticleForm (3/3)

- 수정 확인 후 게시글 작성하기

### CREATE

Title:

Content:

## | 외래 키 데이터 누락 (1/3)

- 게시글 작성 시 NOT NULL constraint failed: articles\_article.user\_id 에러 발생

IntegrityError at /articles/create/

NOT NULL constraint failed: articles\_article.user\_id

Request Method: POST

Request URL: http://127.0.0.1:8000/articles/create/

Django Version: 3.2.7

Exception Type: IntegrityError

Exception Value: NOT NULL constraint failed: articles\_article.user\_id

Exception Location: C:\Users\edujunho\Desktop\05\_django\_model\_relationship\_l\venv\lib\site-packages\django\db\backends\sqlite3\base.py, line 423, in execute

Python Executable: C:\Users\edujunho\Desktop\05\_django\_model\_relationship\_l\venv\Scripts\python.exe

Python Version: 3.9.6

Python Path: ['C:\\Users\\edujunho\\Desktop\\05\_django\_model\_relationship\_l',  
'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39\\python39.zip',  
'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39\\DLLs',  
'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39\\lib',  
'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39',  
'C:\\Users\\edujunho\\Desktop\\05\_django\_model\_relationship\_l\\venv',  
'C:\\Users\\edujunho\\Desktop\\05\_django\_model\_relationship\_l\\venv\\lib\\site-packages']

Server time: Sun, 17 Oct 2021 17:50:59 +0900

- “NOT NULL 제약 조건이 실패했다. articles\_article 테이블의 user\_id 컬럼에서”
- 게시글 작성 시 외래 키에 저장되어야 할 작성자 정보가 누락 되었기 때문

## | 외래 키 데이터 누락 (2/3)

- 게시물 작성 시 작성자 정보가 함께 저장될 수 있도록 save의 commit 옵션을 활용

```
# articles/views.py

@login_required
@require_http_methods(['GET', 'POST'])
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()
            return redirect('articles:detail', article.pk)
```

## | 외래 키 데이터 누락 (3/3)

- 수정 후 게시글이 잘 작성 되는지 확인

| id | title | content | created_at                 | updated_at                 | user_id |
|----|-------|---------|----------------------------|----------------------------|---------|
| 1  | title | content | 2022-09-30 01:39:02.150657 | 2022-09-30 01:39:02.150707 | 1       |
| 2  | 제목    | 내용      | 2022-09-30 05:54:05.241149 | 2022-09-30 05:54:05.241353 | 1       |
| 3  | 게시글   | 작성하기    | 2022-09-30 06:26:39.538131 | 2022-09-30 06:26:39.538177 | 1       |

**DELETE**

## | 게시물 삭제 시 작성자 확인

- 이제 게시물에는 작성자 정보가 함께 들어있기 때문에 현재 삭제를 요청하려는 사람과 게시물을 작성한 사람을 비교하여 본인의 게시물만 삭제 할 수 있도록 함

```
# articles/views.py

@require_POST
def delete(request, pk):
    article = Article.objects.get(pk=pk)
    if request.user.is_authenticated:
        if request.user == article.user:
            article.delete()
            return redirect('articles:index')
        return redirect('articles:detail', article.pk)
```

**UPDATE**



## | 게시물 수정 시 작성자 확인 (1/3)

- 수정도 마찬가지로 수정을 요청하려는 사람과 게시글을 작성한 사람을 비교하여 본인의 게시글만 수정 할 수 있도록 함

```
# articles/views.py

@login_required
@require_http_methods(['GET', 'POST'])
def update(request, pk):
    article = Article.objects.get(pk=pk)
    if request.user == article.user:
        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)
    else:
        return redirect('articles:index')
    ...
```

## | 게시물 수정 시 작성자 확인 (2/3)

- 추가로 해당 게시글의 작성자가 아니라면, 수정/삭제 버튼을 출력하지 않도록 함

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

{% block content %}

...
{% if request.user == article.user %}
    <a href="{% url 'articles:update' article.pk %}">UPDATE</a>
    <form action="{% url 'articles:delete' article.pk %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="DELETE">
    </form>
{% endif %}
...
```

## | 게시물 수정 시 작성자 확인 (3/3)

- 다른 계정으로 접속하여 detail 템플릿에서 다른 회원이 작성한 게시글을 확인

### DETAIL

#### 1 번째 글

제목: title

내용: content

작성 시각: Sept. 30, 2022, 1:39 a.m.

수정 시각: Sept. 30, 2022, 1:39 a.m.

[back](#)

**READ**

## 게시글 작성자 출력 (1/2)

- index 템플릿과 detail 템플릿에서 각 게시글의 작성자 출력

```
<!-- articles/index.html -->

{% extends 'base.html' %}

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

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

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

## | 게시물 작성자 출력 (2/2)

- 출력 확인하기

| Articles  |
|---|
| <a href="#">CREATE</a>  |
| <p>작성자 : admin</p> <p>글 번호: 1</p> <p>글 제목: title</p> <p>글 내용: content</p> |
| <a href="#">DETAIL</a>  |
| <p>작성자 : admin</p> <p>글 번호: 2</p> <p>글 제목: 제목</p> <p>글 내용: 내용</p>         |
| <a href="#">DETAIL</a>  |

| DETAIL   |
|--|
| 1 번째 글   |
| <p>작성자 : admin</p> <p>제목: title</p> <p>내용: content</p> <p>작성 시각: Sept. 30, 2022, 1:39 a.m.</p> <p>수정 시각: Sept. 30, 2022, 1:39 a.m.</p> |
| <a href="#">back</a>   |

# 이어서 ..

삼성 청년 SW 아카데미

**N:1  
(Comment - User)**



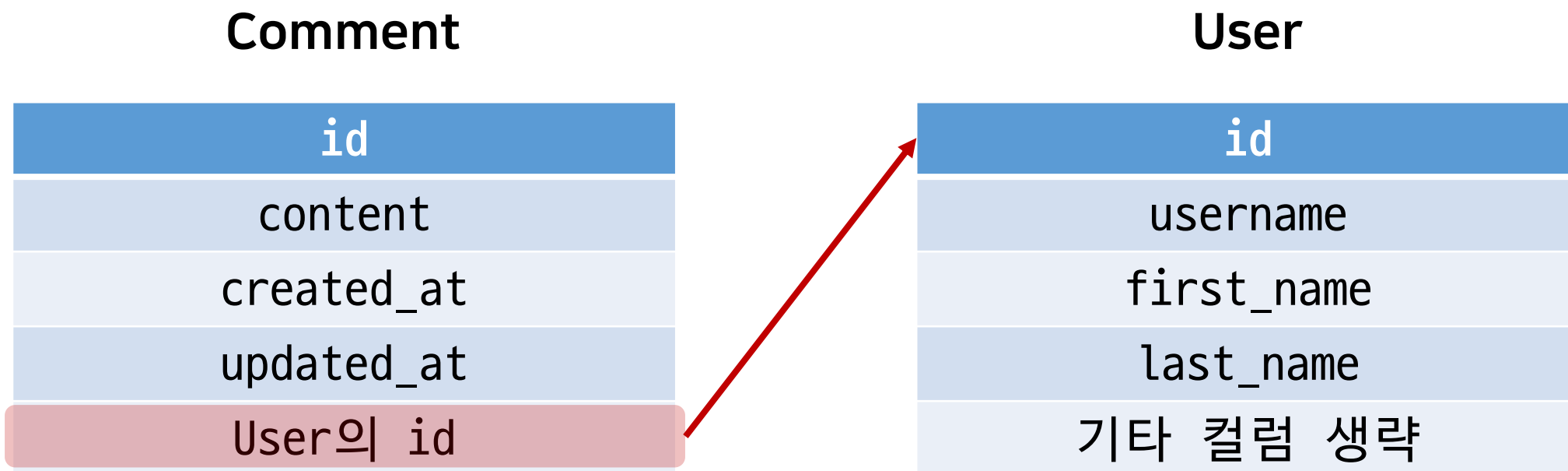
# N:1 (Comment - User)

## | 개요

- Comment(N) - User(1)
- Comment 모델과 User 모델 간 관계 설정
- “0개 이상의 댓글은 1개의 회원에 의해 작성 될 수 있음”

# 모델 관계 설정

## Comment와 User간 모델 관계 설정 (1/2)



## | Comment와 User간 모델 관계 설정 (2/2)

- Comment 모델에 User 모델을 참조하는 외래 키 작성

```
# articles/models.py

class Comment(models.Model):
    article = models.ForeignKey(Article, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    ...
```

## | Migration 진행 (1/5)

- 이전에 User와 Article 모델 관계 설정 때와 마찬가지로 기존에 존재하던 테이블에 새로운 컬럼이 추가되어야 하는 상황이기 때문에 migrations 파일이 곧바로 만들어지지 않고 일련의 과정이 필요

```
$ python manage.py makemigrations
```

## Migration 진행 (2/5)

```
You are trying to add a non-nullable field 'user' to article without a default; we can't do that (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 let me add a default in models.py
```

```
Select an option:
```

- 첫번째 화면
  - 기본적으로 모든 컬럼은 NOT NULL 제약조건이 있기 때문에 데이터가 없는 새로 추가되는 외래 키 필드 user\_id가 생성되지 않음
  - 그래서 기본값을 어떻게 작성할 것인지 선택해야 함
  - 1을 입력하고 Enter 진행 (다음 화면에서 직접 기본 값 입력)

## Migration 진행 (3/5)

```
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g.
timezone.now
Type 'exit' to exit this prompt
>>>
```

- 두번째 화면
  - comment의 user\_id에 어떤 데이터를 넣을 것인지 직접 입력해야 함
  - 마찬가지로 1 입력하고 Enter 진행
  - 그러면 기존에 작성된 댓글이 있다면 모두 1번 회원이 작성한 것으로 처리됨

## | Migration 진행 (4/5)

- migrations 파일 생성 후 migrate 진행

```
$ python manage.py migrate
```



## Migration 진행 (5/5)

- comment 테이블 스키마 변경 및 확인

articles\_comment

- id : integer
- content : varchar(200)
- created\_at : datetime
- updated\_at : datetime
- article\_id : bigint
- user\_id : bigint

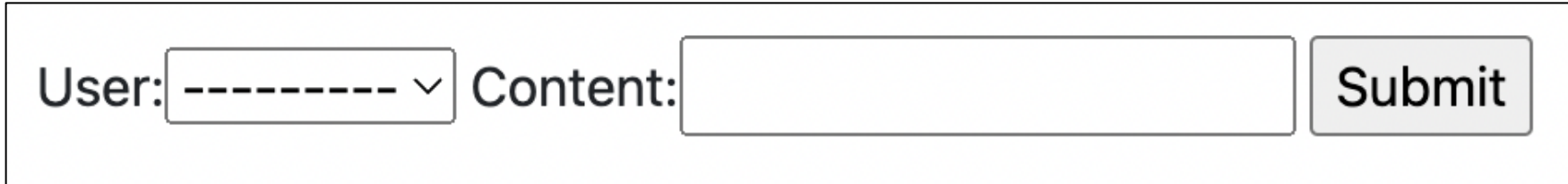
| id | content        | created_at                 | updated_at                 | article_id | user_id |
|----|----------------|----------------------------|----------------------------|------------|---------|
| 1  | first comment  | 2022-09-30 01:39:19.005005 | 2022-09-30 01:39:19.005040 | 1          | 1       |
| 2  | second comment | 2022-09-30 01:42:25.177322 | 2022-09-30 01:42:25.177356 | 1          | 1       |

**CREATE**

## | 개요

- 인증된 회원의 댓글 작성 구현하기
- 작성하기 전 로그인을 먼저 진행한 상태로 진행

## CommentForm (1/3)



User: ----- ▾ Content:

- CommentForm 출력을 확인해보면 create 템플릿에서 불필요한 필드(user)가 출력됨
- user 필드에 작성해야 하는 user 객체는 view 함수의 request 객체를 활용해야 함

## CommentForm (2/3)

- CommentForm의 출력 필드 수정

```
# articles/forms.py

class CommentForm(forms.ModelForm):

    class Meta:
        model = Comment
        exclude = ('article', 'user',)
```

## | CommentForm (3/3)

- 수정 확인 후 댓글 작성하기

Content:

Submit

## 외래 키 데이터 누락 (1/3)

- 댓글 작성 시 NOT NULL constraint failed: articles\_comment.user\_id 에러 발생

```
IntegrityError at /articles/1/comments/  
NOT NULL constraint failed: articles_comment.user_id  
  
Request Method: POST  
Request URL: http://127.0.0.1:8000/articles/1/comments/  
Django Version: 3.2.7  
Exception Type: IntegrityError  
Exception Value: NOT NULL constraint failed: articles_comment.user_id  
Exception Location: C:\Users\edujunho\Desktop\05_django_model_relationship_l\venv\lib\site-packages\django\db\backends\sqlite3\base.py, line 423, in execute  
Python Executable: C:\Users\edujunho\Desktop\05_django_model_relationship_l\venv\Scripts\python.exe  
Python Version: 3.9.6  
Python Path: ['C:\\Users\\edujunho\\Desktop\\05_django_model_relationship_l',  
              'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39\\python39.zip',  
              'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39\\DLLs',  
              'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39\\lib',  
              'C:\\Users\\edujunho\\AppData\\Local\\Programs\\Python\\Python39',  
              'C:\\Users\\edujunho\\Desktop\\05_django_model_relationship_l\\venv',  
              'C:\\Users\\edujunho\\Desktop\\05_django_model_relationship_l\\venv\\lib\\site-packages']  
  
Server time: Sun, 17 Oct 2021 18:07:24 +0900
```

- “NOT NULL 제약 조건이 실패했다. articles\_comment 테이블의 user\_id 컬럼에서”
- 댓글 작성 시 외래 키에 저장되어야 할 작성자 정보가 누락 되었기 때문

## | 외래 키 데이터 누락 (2/3)

- 댓글 작성 시 작성자 정보가 함께 저장될 수 있도록 save의 commit 옵션을 활용

```
# articles/views.py

def comments_create(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', article.pk)
```



## | 외래 키 데이터 누락 (3/3)

- 수정 후 댓글이 잘 작성 되는지 확인

| id | content        | created_at                 | updated_at                 | article_id | user_id |
|----|----------------|----------------------------|----------------------------|------------|---------|
| 1  | first comment  | 2022-09-30 01:39:19.005005 | 2022-09-30 01:39:19.005040 | 1          | 1       |
| 2  | second comment | 2022-09-30 01:42:25.177322 | 2022-09-30 01:42:25.177356 | 1          | 1       |
| 3  | 댓글을 작성해보자      | 2022-09-30 06:41:11.294281 | 2022-09-30 06:41:11.294312 | 1          | 2       |

**READ**

## 댓글 작성자 출력 (1/2)

- detail 템플릿에서 각 게시글의 작성자 출력

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
...
<h4>댓글 목록</h4>
...
<ul>
  {% for comment in comments %}
    <li>
      {{ comment.user }} - {{ comment.content }}
      <form action="{% url 'articles:comments_delete' article.pk comment.pk %}" method="POST">
        {% csrf_token %}
        <input type="submit" value="DELETE">
      </form>
    </li>
  {% endfor %}
...
</ul>
</block>
```

## 댓글 작성자 출력 (2/2)

- 출력 확인하기

### 댓글 목록

3개의 댓글이 있습니다.

- admin - first comment

DELETE

- admin - second comment

DELETE

**DELETE**

## | 댓글 삭제 시 작성자 확인 (1/3)

- 이제 댓글에는 작성자 정보가 함께 들어있기 때문에 현재 삭제를 요청하려는 사람과 댓글을 작성한 사람을 비교하여 본인의 댓글만 삭제 할 수 있도록 함

```
# articles/views.py

def comments_delete(request, article_pk, comment_pk):
    comment = Comment.objects.get(pk=comment_pk)
    if request.user == comment.user:
        comment.delete()
    return redirect('articles:detail', article_pk)
```

## 댓글 삭제 시 작성자 확인 (2/3)

- 추가로 해당 댓글의 작성자가 아니라면, 삭제 버튼을 출력하지 않도록 함

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
...
<ul>
  {% for comment in comments %}
    <li>
      {{ comment.user }} - {{ comment.content }}
      {% if request.user == comment.user %}
        <form action="{% url 'articles:comments_delete' article.pk comment.pk %}" method="POST">
          {% csrf_token %}
          <input type="submit" value="DELETE">
        </form>
      {% endif %}
    </li>
  {% endfor %}
...

```

## | 댓글 삭제 시 작성자 확인 (3/3)

- 다른 계정으로 접속하여 detail 템플릿에서 다른 회원이 작성한 댓글을 확인

### 댓글 목록

3개의 댓글이 있습니다.

- admin - first comment

DELETE

- admin - second comment

DELETE

- test1 - 댓글을 작성해보자



# 인증된 사용자에게 대한 접근 제한하기

# 인증된 사용자에게 대한 접근 제한하기

## | 개요

- is\_authenticated 와 View decorator를 활용하여 코드 정리하기

# 인증된 사용자에게 대한 접근 제한하기

## | 인증된 사용자인 경우만 댓글 작성 및 삭제하기 (1/2)

```
# articles/views.py

@require_POST
def comments_create(request, pk):
    if request.user.is_authenticated:
        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', article.pk)
    return redirect('accounts:login')
```

# 인증된 사용자에게 대한 접근 제한하기

## | 인증된 사용자인 경우만 댓글 작성 및 삭제하기 (2/2)

```
# articles/views.py

@require_POST
def comments_delete(request, article_pk, comment_pk):
    if request.user.is_authenticated:
        comment = Comment.objects.get(pk=comment_pk)
        if request.user == comment.user:
            comment.delete()
    return redirect('articles:detail', article_pk)
```

# 인증된 사용자에게 대한 접근 제한하기

## 비인증 사용자는 CommentForm을 볼 수 없도록 하기

```
<!-- articles/detail.html -->

{% extends 'base.html' %}

{% block content %}
    ...
<hr>
    {% if request.user.is_authenticated %}
        <form action="{% url 'articles:comments_create' article.pk %}" method="POST">
            {% csrf_token %}
            {{ comment_form }}
            <input type="submit">
        </form>
    {% else %}
        <a href="{% url 'accounts:login' %}">[댓글을 작성하려면 로그인하세요.]</a>
    {% endif %}
{% endblock content %}
```

# 이어서 ..

삼성 청년 SW 아카데미

# 마무리

## | 마무리 INDEX

- A many-to-one relationship
  - Foreign Key
  - Django Relationship fields
  - Related manager
- N:1 모델 관계 설정
  1. Comment – Article
  2. Article – User
    - Referencing the User model
  3. Comment – User



# 다음 방송에서 만나요!

삼성 청년 SW 아카데미