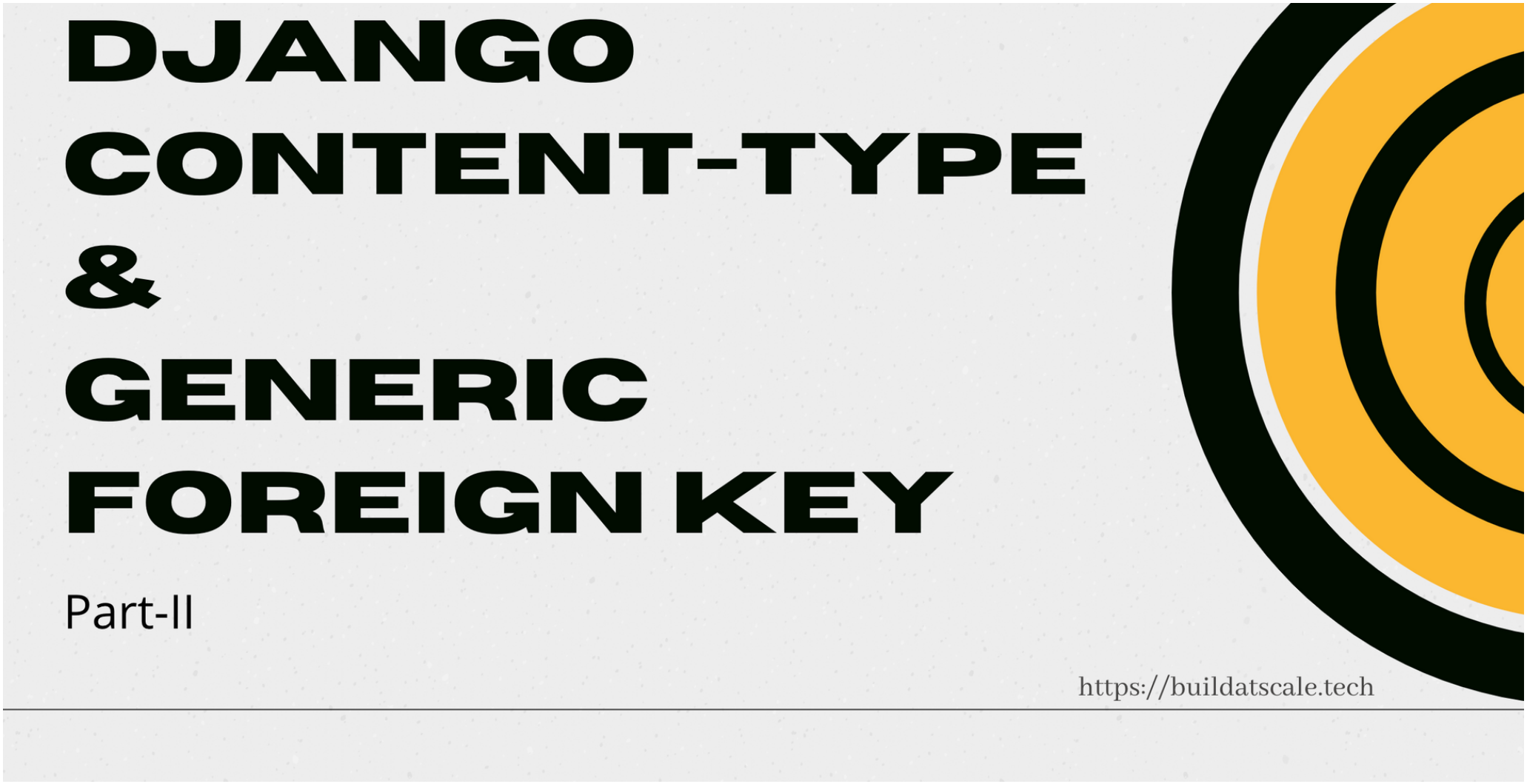
CONTENTTYPE

# How To Use GenericForeignKeys In Django

RAHUL MISHRA

21 AUG 2021 • 4 MIN READ




In the first part of the article, we learned about the `contenttypes` framework and the

Let's first understand when we can need a generic foreign key, consider a case, where we want to store all social media posts made by a user on different sites, such as Twitter, Facebook, etc. One way would be to keep 1 model each for each social media site. Let's see how that model looks.

```
from django.contrib.auth.models import User

from common.models import BaseModel


class FacebookPost(BaseModel):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.TextField()
    posted_on = models.DateTimeField()


class TwitterPost(BaseModel):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.TextField()
    posted_on = models.DateTimeField()


class InstagramPost(BaseModel):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    content = models.TextField()
    posted_on = models.DateTimeField()
```

some problems.

- Not DRY: One issue that we see in the above approach is code repetition. This looks like a code-smell.
- Another issue is; to find all the posts made by a user, we have to fire 3 database calls and aggregate them in the application logic. Also, if we add a new supported site, application logic now needs to fire query on that table as well.
- A more subtle issue is, there is no relation between these models. Domain-wise, they all represent a single entity which is a Post, however, there is no hierarchy/relationship we can see in the code.

One solution we can use here to use `GenericForeignKeys`. We will create 1 single entity `Post`, which can be either a Facebook post, a tweet, or an Instagram post. Let's see how we can re-model the above case.

```
from django.contrib.auth.models import User
from django.contrib.contenttypes.fields import GenericForeignKey
from django.contrib.contenttypes.models import ContentType

from common.models import BaseModel
```

```
    posted_on = models.DateTimeField()

    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey()


class FacebookPost(BaseModel):
    pass


class TwitterPost(BaseModel):
    pass


class InstagramPost(BaseModel):
    pass
```

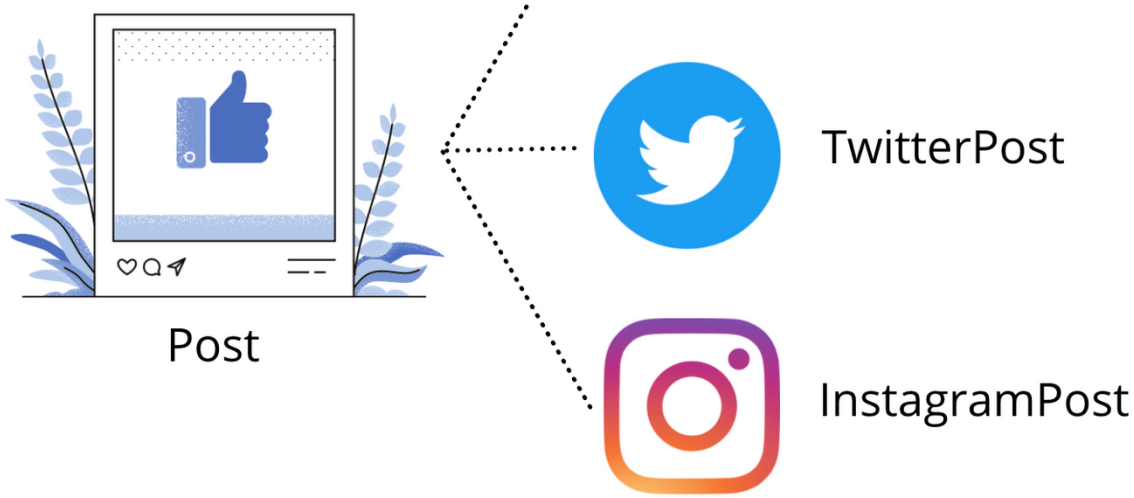
Before digging deep into the GenericForeignKeys constructs, let's see did we solve the problems we had the last time?

- We can clearly see, nothing is repeated, so DRY is achieved. All the common fields are moved to the Post table and any un-common fields can go into the corresponding tables. This solves our first problem.
- Now, we can do 1 query to the Post table to get all the posts made by a user. Even

Post, but by looking at the code, we can still not say how these models are related.

Now, let's see what are those 3 weird-looking fields, that we have added to our model definition.

1. content_type: This is a **`ForeignKey`** to **`ContentType`**. In the above example, if the post is of Facebook, content_type will point to the ContentType of FacebookPost table. The usual name for this field is “content_type”.
2. object_id: This field is used for storing primary key values from the models we are relating to. For most models, this means a **`PositiveIntegerField`**. In our case, for the same Facebook post, this will be the `id` of the FacebookPost row. The usual name for this field is “object_id”.
3. content_object: We need to add a **`GenericForeignKey`**, and pass it the names of the two fields described above. In our case, the value of this field will be the FacebookPost object, whose `id` is mentioned in the object_id.




Post Relationship

```
>>> from django.contrib.contenttypes.models import ContentType
>>> from django.utils import timezone
>>> fb_post = FacebookPost.objects.create()
>>> user = User.objects.last()
>>> post = Post.objects.create(content_object=fb_post, user=user, conte

>>> post.object_id == fb_post.id
True
```

If we see `Post` table in the database, we can see that it does not have content_object, which means `content_object` is a Django/ORM construct and hence, if we try to filter/exclude on content_object in Post table,

```
mysql> desc common_post;
+-----------------+------------------+------+-----+---------+----------------+
| Field           | Type             | Null | Key | Default | Extra          |
+-----------------+------------------+------+-----+---------+----------------+
| id              | bigint(20)       | NO   | PRI | NULL    | auto_increment |
| created_at      | datetime(6)      | NO   |     | NULL    |                |
| updated_at      | datetime(6)      | NO   |     | NULL    |                |
| content         | longtext         | NO   |     | NULL    |                |
| posted_on       | datetime(6)      | NO   |     | NULL    |                |
| object_id       | int(10) unsigned | NO   |     | NULL    |                |
| content_type_id | int(11)          | NO   | MUL | NULL    |                |
| user_id         | int(11)          | NO   | MUL | NULL    |                |
+-----------------+------------------+------+-----+---------+----------------+
8 rows in set (0.00 sec)
```

Post Table

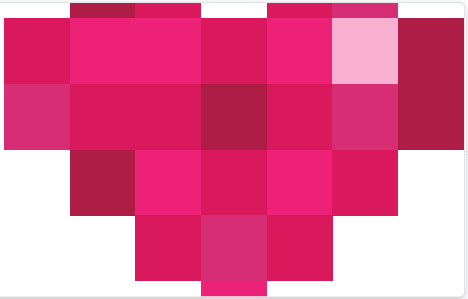
```
>>> Post.objects.filter(content_object=fb_post)
FieldError: Field 'content_object' does not generate an automatic reverse relat
```

only posts from 1 site, we can `filter/exclude` on content_type.

---

**The contenttypes framework | Django documentation | Django**

Django •



**Model Inheritance In Python Django**

There are 3 styles of model inheritance possible in Python Django. Abstract base classes, Mutlitable inheritance, and proxy method.

Build at scale • • Rahul Mishra


## Fixing N+1 query problem in Django

## Django Custom Middlewa

Django middleware is a way to globally

every result of a previous query. If the definition looks cryptic, let's understand this problem

2 MAR 2022 • 4 MIN READ

that will be applied to all requests and responses. Consider an example

13 FEB 2022 • 3 MIN READ