გაკვეთილი 20

Generic Relationship, Custom Manager Object

რა არის Generic Relationship?

Django-ში Generic Relation (ზოგადი კავშირი) გვჭირდება იმიტომ, რომ მოქნილი გავხადოთ მოდელებს შორის კავშირები. ეს საშუალებას გვაძლევს შევქმნათ ისეთი მოდელი, რომელიც შეიძლება დაუკავშირდეს სხვადასხვა სახის მოდელებს, მაგალითად:

1) კომენტარები, რომლებიც შეიძლება ეკუთვნოდეს სხვადასხვა ტიპის მოდელებს (პოსტი, პროდუქტი, მომხმარებელი)
2) ტეგები, რომლებიც შეიძლება მიმაგრებული იყოს სხვადასხვა ტიპის ობიექტებზე (პოსტზე, პროდუქტზე, კატეგორიაზე)
3) სურათი, რომელიც შეიძლება ეკუთვნოდეს სხვადასხვა ტიპის მოდელებს, მაგალითად პროდუქტზე, კატეგორიაზე, მომხმარებლის პროფილზე, პოსტზე და ა.შ

რა უპირატესობაა ჩვეულებრივი FK სთან შედარებით

1) **მრავალმხრივი კავშირი:**
- ForeignKey მხოლოდ ერთ კონკრეტულ მოდელს უკავშირდება
- Generic Relation რამდენიმე სხვადასხვა მოდელთან კავშირის შექმნის საშუალებას იძლევა

1) **მოქნილობა:**
- შეგიძლიათ შექმნათ ისეთი მოდელი, რომელიც დინამიურად დაუკავშირდება სხვადასხვა ტიპის მოდელებს
- არ გჭირდებათ წინასწარ განსაზღვრული კავშირები

1) **კოდის სისუფთავე:**
- არ გჭირდებათ ცალკეული ცხრილების შექმნა თითოეული კავშირისთვის
- ერთი მოდელი შეუძლია გაუმკლავდეს სხვადასხვა ტიპის ობიექტებთან კავშირს

ჩამოვკლონოთ რეპოზიტორი: https://github.com/fantozy/joins-in-django-lesson-1

შევხედოთ models.py ში არსებულ Image მოდელს

```python
class Image(models.Model):
    url = models.URLField()
    content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey('content_type', 'object_id')
```

როგორ მუშაობს Generic Relation?

`GenericRelation` და `GenericForeignKey` გამოიყენება მაშინ, როდესაც ერთი მოდელი (`Image`) უნდა დაუკავშირდეს მრავალ სხვადასხვა მოდელს (`Category`, `Item` ან სხვა). ასეთი მოქნილობა ვერ მიიღწევა ჩვეულებრივი `ForeignKey`-ით, რადგან:

- `ForeignKey` დაკავშირებულია მხოლოდ **ერთ** კონკრეტულ მოდელთან.
- `Image` უნდა იყოს დაკავშირებული **ყველა იმ მოდელთან**, რომელთაც სურთ თავისი სურათების ქონა.

ამიტომ საჭიროა **მექანიზმი, რომელიც გაიგებს:**

- რომელ მოდელთან (მაგ., `Category` ან `Item`) გვაქვს კავშირი.
- კონკრეტულად რომელი ჩანაწერისთვის (მაგ., `Category.id=5` ან `Item.id=10`).

რატომაა საჭირო content_type?

`content_type` არის `ForeignKey` Django-ს სისტემურ `ContentType` მოდელზე. ეს მოდელი ინახავს ინფორმაციას აპლიკაციაში რეგისტრირებული **ყველა მოდელის შესახებ**.

როგორ მუშაობს?

- `content_type` ასოცირებულია იმ მოდელთან, რომლის ობიექტიც უნდა იყოს დაკავშირებული სურათთან.
- იგი გვაძლევს საშუალებას დავადგინოთ, **რომელ მოდელთან** (მაგ., `Category` ან `Item`) გვაქვს საქმე.

content_type-ის მუშაობის სქემა

როდესაც სურათი უკავშირდება კონკრეტულ მოდელს, `Image` მოდელში ინახება:

- **`content_type`**: დაკავშირებული მოდელის სახელი (`Category`, `Item` და ა.შ.).
- **`object_id`**: ამ მოდელის კონკრეტული ობიექტის ID.

მაგალითად:

ID	URL	ContentType	Object_ID
1	img1.jpg	Category (ID=7)	5
2	img2.jpg	Item (ID=15)	10

პირველი ჩანაწერი (`img1.jpg`) ეკუთვნის `Category` მოდელს, რომლის ID-ც 5-ია.

მეორე ჩანაწერი (`img2.jpg`) ეკუთვნის `Item` მოდელს, რომლის ID-ც 10-ია.

- ამ შემთხვევაში ContentType არის ContentType Table ობიექტების ID, ხოლო Object_Id ინახავს უშუალოდ ობიექტის ID'ს

Image.content_object

`content_object` არის **GenericForeignKey**-ის მახასიათებელი, რომელიც Django-ს საშუალებას აძლევს, **კონკრეტული ობიექტი** მოიძიოს `ContentType` და `object_id` ველების საფუძველზე.

გამოყენება

1. შევქმნათ Management Command’ი
2. წამოვიღოთ 1 კატეგორიის ობიექტი და 1 პროდუქტის ობიექტი
3. შევქმნათ ორი სხვადასხვა სურათი და დავაკავშიროთ
4. წამოვიღოთ კატეგორიასთან დაკავშირებული სურათები
5. წამოვიღოთ პროდუქტთან დაკავშირებული სურათები
6. გავფილტროთ Image მოდელი
7. Image ობიექტით წამოვიღოთ მასთან დაკავშირებული კატეგორია/პროდუქტი

დავუმატოთ სურათი კატეგორიას/პროდუქტს

```
item = Item.objects.last()
category = Category.objects.last()

Image.objects.create(url="http://ihateblacks.com", content_object=item)
Image.objects.create(url="http://ihateblacks.com", content_object=category)

```

გამოვიტანოთ item/category დაკავშირებული სურათები

```python
class Command(BaseCommand):
    def handle(self, *args, **kwargs):
        item = Item.objects.last()
        category = Category.objects.last()

        print(item.images.all())
        print(category.images.all())

```

გავფილტროთ Image მოდელი

- წამოვიღოთ ყველა სურათი, რომელიც არის დაკავშირებული “Item” მოდელთან

```
item_content_type = ContentType.objects.get_for_model(Item)
item_images = Image.objects.filter(content_type=item_content_type)
```

რა მოხდა თქვენი აზრით?

დავბეჭდოთ წინა ქმედების Query

SELECT "shop_image"."id", "shop_image"."url", "shop_image"."content_type_id", "shop_image"."object_id" FROM "shop_image" WHERE "shop_image"."content_type_id" = 9

შედეგად უნდა მივიღოთ მსგავსი Query (განსხვავებული “id”)

წამოვიღოთ სურათის მეშვეობით კატეგორია/პროდუქტი

```
1
2  item_content_type = ContentType.objects.get_for_model(Item)
3  item_images = Image.objects.filter(content_type=item_content_type)
4
5  image = item_images.first()
6  print(image.content_object)
7
8  category_content_type = ContentType.objects.get_for_model(Category)
9  category_images = Image.objects.filter(content_type=category_content_type)
10
11 image = category_images.first()
12 print(image.content_object)
```