Django: Check in template if queryset is empty

Asked 2 years, 8 months ago Active 1 year, 4 months ago Viewed 2k times

0

I have found the solution for this but only by checking in views so far. I need to check in templates.

My view:

```
brands = Brand.objects.all()
for brand in brands:
    brand.products = Product.objects.filter(brand=brand.id)
```

And so in my template I want to show all my brands but not those which do not have any product.

```
{% for brand in brands %}
    {% if brand.product is not None %}
        <!-- Displays brand -->
    {% endif %}
{% endfor %}
```

Something like that, but the code `is not None` doesn't work for empty querysets and it is still displaying brands with no objects in them. What can I do?

EDIT: Sharing model as requested.

```
class Brand(models.Model):
    name = models.CharField(max_length=100, null=False)

    def __str__(self):
        return self.name


class Product(models.Model):
    name = models.CharField(max_length=100, null=False)
    brand = models.IntegerField(null=True)
    price = models.DecimalField(max_digits=12, decimal_places=2, null=False)

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

django templates view django-queryset

Share Improve this question Follow

edited May 4 '19 at 15:01

asked May 4 '19 at 14:54

Gonzalo Dambra
660 11 24

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with GitHub Sign up with Facebook

Can you share your models please? – Willem Van Onsem May 4 '19 at 14:56

@WillemVanOnsem for sure. Done! If you can tell me any solution without making a one-to-one relationship I would really appreciate it! – Gonzalo Dambra May 4 '19 at 15:01

1 Hold on, you did not use a `ForeignKey` here? Please change the `IntegerField` to a `ForeignKey` : docs.djangoproject.com/en/2.2/topics/db/examples/many_to_one it will add extra database constraints, and make the Django ORM more expressive, it will also create a better modeling layer. – Willem Van Onsem May 4 '19 at 15:02

That's a great solution and I appreciate your help, but the problem is that I am migrating a desktop Windows application, and I must keep the DB tables as they currently are. – Gonzalo Dambra May 4 '19 at 15:04

except for renaming `brand` to `brand_id` , not much will change at the "database" layer, but it will make the model more consistent. – Willem Van Onsem May 4 '19 at 15:05

3 Answers

Active Oldest Votes

2

Your template should be like:

```
{% for brand in brands %}
    {% if brand.product %}
        <!-- Displays brand -->
    {% endif %}
{% endfor %}
```

Share Improve this answer Follow edited May 4 '19 at 15:17 answered May 4 '19 at 15:04

JBoy
134 1 15

2 Note that the second example will not work, since Django templates do *not* allow function calls (well not explicit ones at least). – Willem Van Onsem May 4 '19 at 15:15

2

It is usually *not* a good idea to use `IntegerField` s, etc. to represent relations. Django uses **`ForeignKey`** [Django-doc] to implement such relations. There are multiple advantages for this: (1) it will add extra constraints to the database, such that the key can only refer to a real `Brand`; and (2) you will have extra ways to retrieve the related `Product` s.

The `Product` model thus should have a `ForeignKey` to the `Brand` model, like:

```
class Product(models.Model):
    name = models.CharField(max_length=100, null=False)
    brand = models.ForeignKey(Brand, null=True, db_column='brand')
    price = models.DecimalField(max_digits=12, decimal_places=2, null=False)
```

Here we leave the database structure itself intact, but it will add a `ForeignKey` on it, as well as an *index*, making retrieving all products for a given `Brand` much faster.

It is usually a bad idea to write (business)logic in templates. In fact one of the reasons why the Django template language does not allow to make calls with parameters, etc. is to avoid that people write business logic in the template. So I strongly advice you to shift the logic to the view.

You can retrieve the `Brand` s that at least have *one* related `Product` in the *view* with a one-liner:

```
Brand.objects.filter(product__isnull=False).distinct()
```

So it is *not* necessary to check each brand individually. Here we also check the existance of a `Product` in a *single* query for all `Brand` s, not an extra query per `Brand` to check if there are related `Product` s.

This will result in a query that looks like:

```
SELECT DISTINCT brand.*
FROM brand
INNER JOIN product ON brand.id = product.brand_id
WHERE product.id IS NOT NULL
```

Share Improve this answer Follow

answered May 4 '19 at 15:15
Willem Van Onsem
368k 28 332 450

0

If you want to display something different when a collection is empty, check this out:

```
{% for i in list %}
    // Do this in non - empty condition
{% empty %}
    // Do this in empty condition
{% endfor %}
```

credit: GeeksForGeeks

Share Improve this answer Follow

answered Aug 29 '20 at 19:21

orangecaterpillar
389 4 10