By **Vitor Freitas**

I'm a passionate software developer and researcher from Brazil, currently living in Finland. I write about Python, Django and Web Development on a weekly basis. Read more.

TIPS

Django Tips #22 Designing Better Models

Feb 10, 2018 8 minutes read 24 comments 33,264 views

In this post, I will share some tips to help you improve the design of your Django Models. Many of those tips are related to naming conventions, which can improve a lot the readability of your code.

The PEP8 is widely used in the Python ecosystem (Django included). So it's a good idea to use it in your own projects.

Besides PEP8, I like to follow Django's Coding Style which is a guideline for people writing code for inclusion in the Django code base itself.

Below, an overview of the items we are going to explore:

Naming Your Models

The model definition is a class, so always use **CapWords** convention (no underscores). E.g. `User`, `Permission`, `ContentType`, etc.

For the model's attributes use **snake_case**. E.g. `first_name`, `last_name`, etc.

Example:

```
from django.db import models

class Company(models.Model):
    name = models.CharField(max_length=30)
    vat_identification_number = models.CharField(max_length=20)
```

Always name your models using **singular**. Call it `Company` instead of `Companies`. A model definition is the representation of a single object (the object in this example is a **company**), and not a collection of companies.

This usually cause confusion because we tend to think in terms of the database tables. A model will eventually be translated into a table. The table is correct to be named using its plural form because the table represents a collection of objects.

In a Django model, we can access this collection via `Company.objects`. We can renamed the `objects` attribute by defining a `models.Manager` attribute:

```
from django.db import models

class Company(models.Model):
    # ...
    companies = models.Manager()
```

So with that we would access the collection of companies as `Company.companies.filter(name='Google')`. But I usually don't go there. I prefer keeping the `objects` attribute there for consistency.

Model Style Ordering

The Django Coding Style suggests the following order of inner classes, methods and attributes:

- If **choices** is defined for a given model field, define each choice as a tuple of tuples, with an all-uppercase name as a class attribute on the model.
- All database fields
- Custom manager attributes
- `class Meta`
- `def __str__()`
- `def save()`
- `def get_absolute_url()`
- Any custom methods

Example:

```
from django.db import models
from django.urls import reverse

class Company(models.Model):
    # CHOICES
    PUBLIC_LIMITED_COMPANY = 'PLC'
    PRIVATE_COMPANY_LIMITED = 'LTD'
    LIMITED_LIABILITY_PARTNERSHIP = 'LLP'
    COMPANY_TYPE_CHOICES = (
        (PUBLIC_LIMITED_COMPANY, 'Public limited company'),
        (PRIVATE_COMPANY_LIMITED, 'Private company limited by shares'),
        (LIMITED_LIABILITY_PARTNERSHIP, 'Limited liability partnership'),
    )

    # DATABASE FIELDS
    name = models.CharField('name', max_length=30)
    vat_identification_number = models.CharField('VAT', max_length=20)
    company_type = models.CharField('type', max_length=3, choices=COMPANY_TYPE_CHOICES)

    # MANAGERS
    objects = models.Manager()
    limited_companies = LimitedCompanyManager()

    # META CLASS
```

```
    class Meta:
        verbose_name = 'company'
        verbose_name_plural = 'companies'

    # TO STRING METHOD
    def __str__(self):
        return self.name

    # SAVE METHOD
    def save(self, *args, **kwargs):
        do_something()
        super().save(*args, **kwargs)  # Call the "real" save() method.
        do_something_else()

    # ABSOLUTE URL METHOD
    def get_absolute_url(self):
        return reverse('company_details', kwargs={'pk': self.id})

    # OTHER METHODS
    def process_invoices(self):
        do_something()
```

Reverse Relationships

related_name

The `related_name` attribute in the `ForeignKey` fields is extremely useful. It let's us define a meaningful name for the reverse relationship.

Rule of thumb: **if you are not sure what would be the `related_name`, use the plural of the model holding the `ForeignKey`.**

```
class Company:
    name = models.CharField(max_length=30)

class Employee:
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    company = models.ForeignKey(Company, on_delete=models.CASCADE, related_name='employees')
```

That means the `Company` model will have a special attribute named `employees`, which will return a `QuerySet` with all employees instances related to the company.

```
google = Company.objects.get(name='Google')
google.employees.all()
```

You can also use the reverse relationship to modify the `company` field on the `Employee` instances:

```
vitor = Employee.objects.get(first_name='Vitor')
google = Company.objects.get(name='Google')
google.employees.add(vitor)
```

related_query_name

This kind of relationship also applies to query filters. For example, if I wanted to list all companies that employs people named 'Vitor', I could do the following:

```
companies = Company.objects.filter(employee__first_name='Vitor')
```

If you want to customize the name of this relationship, here is how we do it:

```
class Employee:
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    company = models.ForeignKey(
        Company,
        on_delete=models.CASCADE,
        related_name='employees',
        related_query_name='person'
    )
```

Then the usage would be:

```
companies = Company.objects.filter(person__first_name='Vitor')
```

To use it consistently, `related_name` goes as **plural** and `related_query_name` goes as **singular**.

Blank and Null Fields

I've written about the differences between Blank and Null fields in another post, but I will try to summarize it here:

- **Null**: It is database-related. Defines if a given database column will accept null values or not.
- **Blank**: It is validation-related. It will be used during forms validation, when calling `form.is_valid()`.

Do not use `null=True` for text-based fields that are optional. Otherwise, you will end up having two possible values for "no data," that is: **None** and an **empty string**. Having two possible values for "no data" is redundant. The Django convention is to use the empty string, not NULL.

Example:

```
# The default values of `null` and `blank` are `False`.
class Person(models.Model):
    name = models.CharField(max_length=255)  # Mandatory
    bio = models.TextField(max_length=500, blank=True)  # Optional (don't put null=True)
    birth_date = models.DateField(null=True, blank=True) # Optional (here you may add null=True)
```

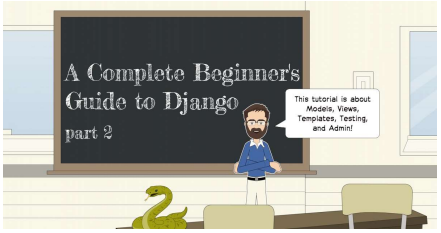
Further Reading

Models definition is one of the most important parts of your application. Something that makes all the difference is defining the field types properly. Make sure to review the Django models field types to know your options. You can also define custom field types.

If you are interested in code conventions, I suggest having a look on Django's Coding Style. I've also published an tutorial about the flake8 library which helps you check for PEP8 issues in your code.

That's it for today! You can also subscribe to my newsletter to receive updates from the blog.

Related Posts



A Complete Beginner's Guide to Django - Part 2

Django Tips #17 Using QuerySet Latest & Earliest Methods

Django Tips #16 Simple Database Access Optimizations

django models pep8

Share this post

24 Comments Simple is Better Than Complex Login

Recommend 13 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

Evgenii Fedulov • a year ago
Cool! Great example about related name, it has to be added to official django tutorials.
6 • Reply • Share ›

Foridur Kayes Shawon • a year ago
You are such a great writer. Thank you very much.
4 • Reply • Share ›

alzearafat • a year ago
Wow, thank you! God bless you Vitor
2 • Reply • Share ›

Ryan Manzer • 10 months ago
Nice example. I have one suggestion that has saved me quite a bit of time. I include by default a "get_delete_url(self):" method on all my models (I added it as part of my Django model snippet in Atom). In multiple projects using AJAX approaches, this has allowed me to use a single confirm_delete.html and a general set of Javascript functions across multiple models as well as multiple apps. Perhaps it's something others might find useful as well.
1 • Reply • Share ›

ocobacho • a year ago
Well done! Thank you. Are you planning on writting something about channels?
1 • Reply • Share ›

Kristof Van Coillie • a year ago
Great post as always, one question: what about using null=True and blank=True on a CharField when also using unique=True?
(https://docs.djangoproject....

The other solution is not to enforce the uniques on database level but handle it in the clean function, what is considered the better option?

1 • Reply • Share ›

Vishal Chitnis • 2 months ago

Great post. Spaghetti code becomes impossible to maintain later.

• Reply • Share ›

Sehan shah C • a year ago

Add this post with self referential model and it's purpose.
Eg,

class Category(models.Model):
category_name = models.CharField(max_length=255, null=True)
parent_id = models.ForeignKey('self', null=True, blank=True)

Where parent_id with NULL will be the parent category and parent_id with id of category_name will be its child.

• Reply • Share ›

Michael Brooks • a year ago

I would also recommend building ugettext_lazy strings into your models for easy translation later on:

from django.utils.translation import ugettext_lazy as _

class Employee:
name = models.CharField(_("Name"), max_length=30)
company = models.ForeignKey(Company, verbose_name=_("Company"))

• Reply • Share ›

Marcelo C. Lopes • a year ago

Incredible, your blog is already in my favorites !!!

• Reply • Share ›

Keshav Metkar • a year ago

Very Handy... i am becoming your fan Vitor Freitas

• Reply • Share ›

Pranay reddy • a year ago • edited

I used foreignkey in model and i want to multiselect different fields in selection form, In the form i used forms.MultipleChoiceField(widget=forms.CheckboxSelectMultiple),it is not showing any data in that field.
how to get done.

• Reply • Share ›

Eli Clement • a year ago

Thanks Vitor for again a very clear tutorial! Looking forward to read your Django REST Framework insights...

• Reply • Share ›

Can Elma • a year ago

I like your helpful articles. Thank you.

Reply • Share ›

Luca • a year ago

Great article :)
Just a few corrections (mostly spelling):
This usually cause confusion because we tend -> should be 'causes'
We can renamed the objects attribute -> should be 'rename'
When you use the related name for a query example, shouldn't it be 'employees' (plural) instead of 'employee'?
`companies = Company.objects.filter(employees__first_name='Vitor')`

Reply • Share ›

William Kpabitey Kwabla • a year ago

Hi Vitor, You very good with Python and Django. I am William from Ghana.
I would love if you could write articles on how to be good with Python and Django for we the beginners. Steps or path we need to take to learn them.

Reply • Share ›

Diego Emanuel Armoa • a year ago

muchas gracias por los consejos.....

Reply • Share ›

HC.conf • a year ago

Great Tip ! Vitor :D

Reply • Share ›

Chris Shyi • a year ago

Incredibly useful to Django users, good work!

Reply • Share ›

Kuldeep Pisda • a year ago

You are amazing :-)

Reply • Share ›

vikram iyer • a year ago

Thanks, Vitor :)

Reply • Share ›

Kavie • a year ago

Thank you Vitors! I will not realize how inconsistent my Django have until reading this!!

Reply • Share ›

Avatar This comment was deleted.

MaT ➔ Guest • a year ago

So it means you did not read the official Django documentation. :-) Because there it is

explicitly mentioned:
https://docs.djangoproject....
And I think it's also mentioned in lot of tutorials.

• Reply • Share ›

Angel Felipe González • a year ago

Again Vitor: great post, i enjoy reading your useful tips! thaks a lot

• Reply • Share ›

ALSO ON **SIMPLE IS BETTER THAN COMPLEX**

A Complete Beginner's Guide to Django - Part 4

156 comments • 2 years ago

Mohamed Abdallah — I think you should upgrade to django 2.0 to solve this problem

Launching our Community Forum

4 comments • 8 months ago

Trung Bát — Great guide, thanks for sharing.

A Complete Beginner's Guide to Django - Part 6

54 comments • 2 years ago

Rohit Kumar — Please someone help me to Delete a Post / Topic / Replies ????

A Complete Beginner's Guide to Django - Part 1

109 comments • 2 years ago

gamesbook — Hi Vitor - this is much needed. But *please*, do incorporate actual, real, comprehensive tests (along with a method to

Subscribe to our Mailing List

Receive updates from the Blog!

Your email address

SUBSCRIBE

Popular Posts

[How to Extend Django User Model]

[How to Setup a SSL Certificate on Nginx for a Django Application]

[How to Deploy a Django Application to Digital Ocean]