Theory: Admin interface

⏱ 25 minutes   0 / 5 problems solved

[Skip this topic]   [Start practicing]

Often developers do not have interactive access to the database on the production server, even with the Python console. Then how can you manipulate data in the database? And what can your colleagues do with business objects if they don't know Python and Django at all?

You're probably dreading having to write yet another web interface for that. Well, it's not all that bad: you'll need to write some code, but not as much as you think. Django provides a helpful admin interface to work with objects in it.

§1. Registering Models

Initially, you don't have any models in the admin interface. In order to add them, you should register them in the *admin.py* module. This module exists in each application by default.

Let's look at a hypothetical project. We are creating an online auction, but do not have much time to prepare the first release. To avoid using external dependencies, we change the content of the site in the Django admin interface. The models look like this:

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


class Lot(models.Model):
    description = models.CharField(max_length=256)
    initial_price = models.FloatField()



class Bet(models.Model):

    lot = models.ForeignKey(Lot, on_delete=models.CASCADE)

    price = models.FloatField()

    user = models.ForeignKey(User, on_delete=models.CASCADE)
```

To see models in the admin interface, add admin classes to each model and match them with the `register` function. The content of admin classes can be the `pass` keyword; it means that Django can configure it by itself. Let's see what we should add to the *admin.py* module.

```python
from django.contrib import admin
from .models import Bet, Lot


class BetAdmin(admin.ModelAdmin):
    pass


class LotAdmin(admin.ModelAdmin):

    pass




admin.site.register(Bet, BetAdmin)

admin.site.register(Lot, LotAdmin)
```

We've registered our models. Now let's see what you can get from it.

> You cannot register the same model twice with different ModelAdmin classes.

§2. Admin Interface

Admins are not ordinary users; they are *superusers* in Django terminology. To use the interface, create a superuser account if you don't have one already. Also, do not remove `path('admin/', admin.site.urls)` from the default configuration of the main *urls.py* module of your project.

If you're done with all preparations, run `python manage.py runserver` from the console and go to '*http://localhost:8000/admin*' URL in your browser. You should log in with a superuser account.

Now you're able to add, change or delete *Users*, *Groups*, *Bets* and *Lots*. Try to check out every link and change the objects you want: it will help you get familiar with the interface.

All objects in the database have a page, and you can see and change the object's content on it.

You already know how to fill the content of the service through the web interface. Let's now learn how we can upgrade it and add restrictions and modifications.

§3. Customize Fields

Not all data should be available for making changes: for example, the clients won't be happy if we adjust their bets. No matter how honest developers and business users are, it's better to forbid them to change any data in `Bet` objects.

The instance of the `ModelAdmin` class has a lot of customizable attributes. The `fields` attribute is the list of model's fields you want to display on a page. The `readonly_fields` is the list of fields that all admins can see but not modify. Both attributes can be `tuple` or `list` instances.

```
1    class BetAdmin(admin.ModelAdmin):
2        fields = ('lot', 'price', 'user')
3        readonly_fields = ('lot', 'price', 'user')
```

Now you can check that you see all the data but cannot modify any bets. You can choose by yourself what to include on a page removing some elements from the list.

> The *readonly_fields* attribute should be a subset of *fields* attribute, otherwise not all elements from read-only fields will be present on a page.

§4. Additional Fields

It's great to get the plain data from the tables with several lines of code, but how do we add more information from the database?

We want to see which lots are trending. In our case, it's the lots with 10 bets or more. There is no such field in a model, but we can prepare the `ModelAdmin`'s field with this value. We choose any name we want (unless it intersects with the existing attribute of the class). Look through the attributes in the source code. We add a `trending` method and add a string '*trending*' to the `fields` list.

```
1    class LotAdmin(admin.ModelAdmin):
2        fields = ('description', 'initial_price', 'trending')
3        readonly_fields = ('trending',)
4
5        def trending(self, obj):
6            return '🔥' if obj.bet_set.count() >= 10 else '❄'
```

After you've added the custom field, you can see if a lot is *trending* or not. Besides strings, a method may return any other Python objects converting them to strings implicitly.

> Do not forget to add a custom field to *readonly_fields* list, otherwise, it will be an error: custom fields are not modifiable.

§5. Inline Admin Field

We can add any fields we like if they are custom values or parts of a model. But can we display all the bets for a lot on the page? We can use the inline fields for that.

This time, we need to create an additional class for displaying each inline object. We want to add bets to a page with the lot, so we need to create the `TabularInline` class for the `Bet` model.

```
1     from django.contrib import admin
2     from .models import Bet, Lot
3
4
5     class BetInline(admin.TabularInline):
6         can_delete = False
7         model = Bet
8         readonly_fields = ('price', 'user')
9
1
0
1
1     class LotAdmin(admin.ModelAdmin):
1
2         inlines = (BetInline,)
1
3
1
4
1
5     admin.site.register(Lot, LotAdmin)
```

We don't register the `BetInline` class because we cannot use it independently, but we define the `model` attribute for it. We restrict actions for this class with read-only fields and set `can_delete` attribute to `False` to prevent deleting bets by the admin user.

In the `LotAdmin` class we define the `inlines` attribute and add the `BetInline` to it.

That's it, now all bets with the lot item are displayed on the same page.

> You can create multiple different inline classes for the same model.

There are a lot of options available for you to modify default classes, and you can read more about it if you like.

▤ Report a typo

22 users liked this theory. **6** didn't like it. **What about you?**

😍   🙂   😐   🙁   😡

Start practicing

Comments (4)   Hints (0)   Useful links (0)   Show discussion