A Django Model Manager for Soft Deleting Records and How to Customize the Django Admin

Posted on July 1st, 2009 by Greg Allard in Django | 8 Comments

Sometimes it's good to hide things instead of deleting them. Users may accidentally delete something and this way there will be an extra backup. The way I've been doing this is I set a flag in the database, **deleted = 1**. I wrote this code to automatically hide records from django if they are flagged.

Django allows developers to create model managers that can change how the models work. The code below was written to return only the undeleted records by default. I added two new methods in case I need to get some of the deleted records.

```
from django.db import models

class SoftDeleteManager(models.Manager):
    ''' Use this manager to get objects that have a deleted field '''
    def get_query_set(self):
        return super(SoftDeleteManager, self).get_query_set().filter(deleted=False)
    def all_with_deleted(self):
        return super(SoftDeleteManager, self).get_query_set()
    def deleted_set(self):
        return super(SoftDeleteManager, self).get_query_set().filter(deleted=True)
```

This is usable by many models by adding this line to the model (it needs a deleted field) **objects = SoftDeleteManager()**

This will hide deleted records from django completely, even the django admin and even if you specify the id directly. The only way to find it is through the database itself or an app like phpMyAdmin. This might be good for some cases, but I went a step further to make it possible to undelete things in the django admin.

Django has a lot of customization options for the admin interface (this article has some more info on customizing the django admin). I wanted the queryset to be different in the admin, so I created a ModelAdmin to customize what is displayed. First I set it up to show a few more columns than just __unicode__ on the list of items and added a filter to help easily separate the deleted from the active.

```
from django.contrib import admin

class SoftDeleteAdmin(admin.ModelAdmin):
    list_display = ('id', '__unicode__', 'deleted',)
    list_filter = ('deleted',)
# this requires __unicode__ to be defined in your model
```

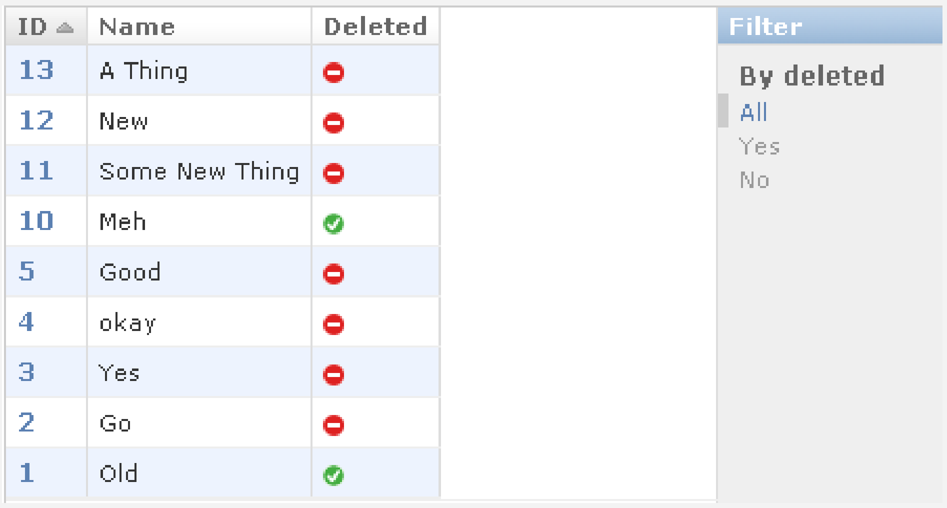
This can also be used by many models by adding this at the bottom of the models.py file:

```
from django.contrib import admin
from wherever import SoftDeleteAdmin
admin.site.register(MyModel, SoftDeleteAdmin)
```

The next thing to do was override the queryset method in the default ModelAdmin. I copied the code from the django source and changed it from using get_query_set to make it use **all_with_deleted()** which was a method added to the ModelManager. The following code was **added to SoftDeleteAdmin**.

```
    def queryset(self, request):
        """ Returns a QuerySet of all model instances that can be edited by the
        admin site. This is used by changelist_view. """
        # Default: qs = self.model._default_manager.get_query_set()
        qs = self.model._default_manager.all_with_deleted()
        # TODO: this should be handled by some parameter to the ChangeList.
        ordering = self.ordering or () # otherwise we might try to *None, which is b
        if ordering:
            qs = qs.order_by(*ordering)
        return qs
```

The list of objects in the admin will start to look like this.



ID	Name	Deleted
13	A Thing	⊖
12	New	⊖
11	Some New Thing	⊖
10	Meh	✓
5	Good	⊖
4	okay	⊖
3	Yes	⊖
2	Go	⊖
1	Old	✓

A screenshot of the django admin interface

They are showing up there now, but won't be editable yet because django is using get_query_set to find them. There are two methods I **added to SoftDeleteManager** so that django can find the deleted records.

```
    def get(self, *args, **kwargs):
        ''' if a specific record was requested, return it even if it's deleted '''
        return self.all_with_deleted().get(*args, **kwargs)

    def filter(self, *args, **kwargs):
        ''' if pk was specified as a kwarg, return even if it's deleted '''
        if 'pk' in kwargs:
            return self.all_with_deleted().filter(*args, **kwargs)
        return self.get_query_set().filter(*args, **kwargs)
```

With those updated methods, django will be able to find records if the primary key is specified, not only in the admin section, but everywhere in the project. Lists of objects will only return deleted records in the admin section still.

This code can be applied to a bunch of models and easily allow soft deletes of records to prevent loss of accidentally deleted objects.

I use webfaction to host a lot of my django projects. It has an easy setup that will get you developing quickly and a great community of talented programmers. There is also a quick setup for rails, wordpress, and a lot more.

Related posts:

1. How to Display Realtime Traffic Analytics Users of Presskit'n have been asking for traffic statistics on their press releases so I decided I would get them...
2. How to Write Django Template Tags Template tags can be useful for making your applications more reusable by other projects. For this example I will be...
3. How to Write Reusable Apps for Pinax and Django Pinax is a collection of reusable django apps that brings together features that are common to many websites. It...