Django Closuretree Documentation

Release 1.2.0

Mike Bryant

Nov 06, 2017

Contents

1 Requirements **3**

2 Basic Usage **5**

3 Adding to existing models **7**

4 Indirect relations **9**

5 Using the `update` QuerySet method **11**

6 API Documentation **13**
6.1 API Documentation 13

7 Indices and tables **15**

django-closuretree is an implementation of a closure tree for tree-based Django models. It aims to reduce the number of database hits required when traversing complex tree-based relationships between models in your Django app.

CHAPTER 1

Requirements

- Django 1.4+
- Sphinx (for documentation)

CHAPTER 2

Basic Usage

Inherit your models from `closuretree.models.ClosureModel` instead of `django.db.models.Model`:

```
from django.db import models
from closuretree.models import ClosureModel

class MyModel(ClosureModel):
    parent = models.ForeignKey('self', related_name='children')
    name = models.CharField(max_length=32)

    def __unicode__(self):
        return '%s: %s" % (self.id, self.name)
```

django-closuretree will automatically use the field named `parent` as the relationship. This can be manually overriden:

```
from django.db import models
from closuretree.models import ClosureModel

class MyModel(ClosureModel):
    parent_rel = models.ForeignKey('self', related_name='children')
    name = models.CharField(max_length=32)

    class ClosureMeta(object):
        parent_attr = 'parent_rel'

    def __unicode__(self):
        return '%s: %s' % (self.id, self.name)
```

Perhaps the most useful methods provided by `closuretree.models.ClosureModel` are the following:

```
>> my_model = MyModel.objects.get(pk=10)
>> my_model.get_ancestors()
[<MyModel: 1: Foo>, <MyModel: 2: Bar>, <MyModel: 3: Fish>]
>> my_model.get_descendants()
[<MyModel: 11: Bob>, <MyModel: 12: Alice>]
```

```
>> my_model.get_descendants(depth=1)
[<MyModel: 11: Bob>]
>> my_model.get_root()
<MyModel: 1: Foo>
>> my_model.is_ancestor_of(MyModel.objects.get(name='Alice'))
True
>> my_model.is_descendant_of(MyModel.objects.get(name='Bar'))
True
```

Read the *closuretree Package* model documentation for more methods.

CHAPTER 3

Adding to existing models

If you add **django-closuretree** to existing models, you'll need to build the closure table for the pre-existing data:

```
MyModel.rebuildtable()
```

CHAPTER 4

Indirect relations

If your model is linked to itself via an indirect relationship (for example, ModelA -> ModelB -> ModelC -> ModelA), then you'll need to define a parent property that traverses this relationship, and set a sentinel attribute as the foriegn key to ModelB:

```
class ModelA(ClosureModel):
    model_b = models.ForeignKey(ModelB)

    @property
    def parent(self):
        return self.model_b.model_c.model_a

    class ClosureMeta:
        sentinel_attr = 'model_b'
```

Closuretree will watch the sentinel attribute for changes, and use the value of the parent property when rebuilding the tree.

CHAPTER 5

Using the `update` QuerySet method

If you change the parent field of a model (or number of models) using the QuerySet `update` method (i.e. `MyModel.objects.filter(...).update(parent=...)`) you'll need to rebuild the closure table for that model manually, as the pre- and post-save signal handlers are not called:

```
MyModel.rebuildtable()
```

CHAPTER 6

API Documentation

6.1 API Documentation

6.1.1 closuretree Package

`models` Module

CHAPTER 7

Indices and tables

- genindex
- modindex
- search