Django Model Revisioning Documentation

Release 0.0.1

Vidir Valberg Gudmundsson

Dec 04, 2019

Contents

django-model-revisioning adds history to your models - migration compatible!

Contents:

CHAPTER 1

Options

Doc Brown uses a class similar to the `Meta` class in django models. Listed below are all the options available.

1.1 fields

Which fields should be revisioned. Will take all fields if not defined or set to `'__all__'`.

1.2 soft_deletion

Controls whether instances actually get deleted or not when `delete()` is called. If set to `True` a `is_deleted` boolean field will be added to the model and this set instead of deleting the instance.

CHAPTER 2

Admin integration

Getting a interface for viewing revision, and even changing the current head, is quite easy. Simply use `RevisionedModelAdmin` as such:

```
from django.contrib import admin
from model_revisioning.admin import RevisionedModelAdmin
from .models import Bar

admin.site.register(Bar, RevisionedModelAdmin)
```

Since `RevisionedModelAdmin` inherits from `ModelAdmin`, it is possible to extend the admin as usual:

```
from django.contrib import admin
from model_revisioning.admin import RevisionModelAdmin
from .models import Bar

class BarAdmin(RevisionModelAdmin):
    list_display = ('char', 'current_revision', 'revisions_count')

admin.site.register(Bar, BarAdmin)
```

CHAPTER 3

Signals

django-model-revisioning emits the following signals when dealing with revisions:

3.1 pre_revision

model_revisioning.signals.**pre_save**

Sent before creating a revision.

Arguments:

sender The model class.

instance The instance for which a revision is about to be created.

3.2 post_revision

model_revisioning.signals.**post_save**

Sent a revision has been created.

Arguments:

sender The model class.

instance The instance for which a revision has been created.

revision The revision instance itself.

3.3 pre_change_head

model_revisioning.signals.**pre_change_head**

Sent before head gets changed on an object.

Arguments:

sender The model class.

instance The instance for which the head is about to change

current_head The current head.

future_head The head which is about to become the current.

3.4 post_change_head

model_revisioning.signals.**post_change_head**

Sent after head gets changed on an object.

Arguments:

sender The model class.

instance The instance for which the head is about to change

old_head The head which used to be current.

new_head The head which is now current.

CHAPTER 4

Management commands

4.1 graph_revision

`./manage.py graph_revision <model_path:label> <pk> <output>`

Create a graphviz directed graph of revisions. Useful for getting visual overview of branches.

Two files will be produced. A `.gv` with the raw graphviz markup, and a `.gv.png` which is a rendered image.

Requirements:

Both graphviz itself, and the python package called graphviz are required.

Arguments:

`model_path` Dotted path to model, skipping `models`. Thus a model named `Bar` in the app `foo` would be `foo.Bar`.

By default the `pk` of the revision is used as a label for the corresponding node. If another field should be used, append it prefixed with a `:`. Thus to show the field `name` use: `foo.Bar:name`.

`pk` Which instance of the given model to graph.

`output` Name of the output file.

Example

```
./manage.py graph_revision foo.Bar:name 42 graph
```

CHAPTER 5

What does django-model-revisioning provide?

django-model-revisioning makes copies of your models so that the django migration framework actual tables in your database.

Say you have a model called *Movie*, django-model-revisioning will create a model called *MovieRevision*. Every time you save an instance of *Movie* a *MovieRevision* instance will be created as well.

If you then add new fields to *Movie*, django-model-revisioning will pick up on it and add the same fields to *MovieRevision*.

CHAPTER 6

Installation

You can install the pre-release version using the following command:

```
pip install django-model-revisioning
```

Note that this is an alpha version and is not recommended for production use!

CHAPTER 7

Usage

To install a flux capacitor in your model inherit from `RevisionModel` and define a `Revisions` class in your model, like this:

```
from django.db import models
from model_revisioning.models import RevisionModel

class Movie(RevisionModel):
    name = models.CharField(max_length=200)
    year = models.IntegerField()

    class Revisions:
        fields = ["name", "year"]
```

See *Options* for which options are available.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

M