Django-maat Documentation

Release 1.0

Germano Guerrini

February 21, 2017

A Django application that optimizes ordered queryset retrieving **when using MySQL**.

This is expecially useful when you have to display a list of objects using one or more ordering criteria.

The high speed is due to an external table thoughtfully indexed that optimizes both the join and the ordering of the rows **without file sorting**.

Specifically, the index is built in a way that all the columns in the *WHERE* clause are constant.

Further documentation can be found here: http://dev.mysql.com/doc/refman/5.0/en/order-by-optimization.html

Requirements

Python 2	> 2.7
Python 3	>= 3.4
Django	>= 1.8

CHAPTER 2

Contents

Changelog

Version 1.6

- Support for Django 1.9 and 1.10
- Dropped support for Django < 1.8

Version 1.5

- populate_maat_ranking command accepts typologies

Version 1.4.2

- Django 1.8 compatibility

Version 1.4.1

- More idiomatic syntax

Version 1.4

- Added migrations files for Django 1.7+

Version 1.3

- Added support for Python >= 3.2

Version 1.2

- Django 1.7 compatibility

Version 1.1

- Django 1.6 compatibility
- Optimized memory usage

Version 1.0

- First version

Installation

Install the `djangomaat` package:

```
pip install djangomaat
```

Make sure `djangomaat` is listed among your `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    # [...]
    'djangomaat',
]
```

Run the `syncdb` command, or `migrate` if you are using Django 1.7:

```
./manage.py syncdb
```

CHAPTER 4

Usage

In order to use Maat, you need to create a subclass of `MaatHandler` for each Django model you want to use and attach it to it.

This class expects one or more methods whose name starts with `get_pk_list_for_`.

The suffix you will append to the name of the method will be used to retrieve the objects from the original model (more on this later).

Each of these methods must return a list of pks (or an iterator, which is actually preferable) in the same order they are expected to be returned. Note that nothing forces you to returns the whole list of instances of the model. You can returns a subset if that is what you need.

```
from djangomaat.register import maat
from djangomaat.handlers import MaatHandler

class ArticleMaatHanlder(MaatHandler):

    def get_pk_list_for_comment_count(self):
        return Article.objects.filter(
            thread__content_type=ContentType.objects.get_for_model(Article),
        ).order_by('-thread__comment_count').values_list('pk', flat=True)[:1000].iterator()

    def get_pk_list_for_popularity(self):
        return Article.objects.filter(
            popularity__content_type=ContentType.objects.get_for_model(Article),
        ).order_by('-popularity__score').values_list('pk', flat=True)[:1000].iterator()

maat.register(Article, ArticleMaatHanlder)
```

In the example above, we have two methods because we want to list the *most commented* and the *most popular* articles (only the first thousand of them).

After that, remember to register your handler as shown on the last line.

Note: In Django 1.6 the preferred module for this code to live in is the `models.py` of your application, as it gets imported automatically.

If you are using Django > 1.7 you might want to use AppConfig.ready() instead.

Now that you are done defining your subclasses, you need to tell Maat to built its index:

```
./manage.py populate_maat_ranking
```

This will, as a matter of fact, denormalize the *current* order of the objects.

Now you can finally retrieve your objects, for example in a Django view, using a `maat` handler that is automatically attached to the model class:

```
most_popular_articles = Article.maat.ordered_by('popularity')
most_commented_articles = Article.maat.ordered_by('comment_count')
```

You can also retrieve them in inverted order:

```
less_popular_article = Article.maat.ordered_by('-popularity')
less_commented_article = Article.maat.ordered_by('-comment_count')
```

Important: the order of the objects is *frozen* at the time you run `populate_maat_ranking`. Depending on your requirements, you should schedule the command to run at regular intervals.

If you need to have different intervals for different models, you can pass a list of `app_label.model_name`:

```
./manage.py populate_maat_ranking my_app.article
```

This will rebuild only the handler registered for that particular model.

Moreover, you can rebuild only given sorting function with the following syntax:

> ./manage.py populate_maat_ranking my_app.article:popularity,comment_count

That is, add a comma separated values after a colon with the name of your sorting criteria.