django-ordered-field Documentation

Release 1.0.0

Kim-Georg Aase

Aug 09, 2018

Contents

Contents:

CHAPTER 1

django-ordered-field

A django field to make it easy to order your model instances. If you have made an ordered list and you change the position of the list item instance then all the other list iteminstances belonging to that list has their position automatically updated to keep the list ordered without holes and list items with duplicate positions. `OrderedField field` is a global ordering field for the entire table. `OrderedCollectionField` order instances with respect to one or more other fields on the instance.

1.1 Requires

- python>=3.6
- django>=2.0

1.2 Documentation

The full documentation is at https://django-ordered-field.readthedocs.io.

1.3 Quickstart

Install django-ordered-field:

```
pip install git+https://github.com/kimgea/django-ordered-field.git
```

In your models.py add the field you want `OrderedField` or `OrderedCollectionField`:

```
from django_ordered_field import OrderedField

class YourModel(models.Model):
```

(continues on next page)

(continued from previous page)

```
    name = models.CharField(max_length=100)
    order = OrderedField()
```

And your ready to go.

1.4 Features

- `OrderedField` will keep correct ordering between all instances in the enire table
- `OrderedCollectionField` can seperate the table in different collection based on one or more columns and keep order in each collection
- `update_auto_now` will update all other fields containing auto_now=True with django.utils.timezone.now if it is set to True
- `extra_field_updates` can be used to update other fields when their order is changed
- `self_updates_on_collection_change` can be used to update self (current instance) when it changes collection. Setting `self_updates_on_collection_change_like_regular` to True will make it use the values from the extra_field_updates

1.5 Limitations

- Must user model.save(). queryset methods does not work
- Order field cant be unique or in an uniqu_togheter constraint
- After a position has been updated, other members of the collection are updated using a single SQL UPDATE statement, this means the save method of the other instances won't be called. As a partial work-around use the `update_auto_now`, `extra_field_updates` and the `self_updates_on_collection_change` functionalities.

1.6 TODO

- Finish setup.py
- Check project files
- Try to download from git and use in other project
- Register on pip
- Register on django
- Make example project - eh, probably skiping it

1.7 Credits

Based on django-positions (it did not work for django 2 at the time):

- django-positions

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-ordered-field
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-ordered-field
$ pip install django-ordered-field
```

CHAPTER 3

Usage

3.1 Ordered Field

The OrderedField class keeps all the records in the table in order from 0 to count - 1

```
from django_ordered_field import OrderedField

class Table(models.Model):
    name = models.CharField(max_length=100)
    order = OrderedField()
```

3.2 Ordered Collection Field

The OrderedCollectionField class is used to keep records in order related to another field. This can be used to make structures like ordered lists and so on.

```
from django_ordered_field import OrderedCollectionField

class Item(models.Model):
    name = models.OrderedCollectionField(max_length=100)
    item_type = models.IntegerField()
    order = OrderedCollectionField(collection='item_type')
```

A collection can consist of another signle field or it can be a combination of multiple fields

```
OrderedCollectionField(collection='item_type')
# Or
OrderedCollectionField(collection=['list', 'sub_list'])
```

3.3 Update table data

Inserting, updating and deletion of instances has to use methods that uses the model.save() and model.delete() methods. queryset.update(…), queryset.delete() and similar functions that omits model.save() and model.delete() will destroy the ordering of the instances.

```
Item.objects.create(name="A")
# [('A', 0)]
item = Item(name="B")
item.save()
# [('A', 0), ('B', 1)]
Item.objects.create(name="C", order=0)
# [('C', 0), ('A', 1), ('B', 2)]
item = Item.objects.filer(name='A').first()
item.order = -1
item.save()
# [('C', 0), ('B', 1), ('A', 2)]
item = Item.objects.filer(name='A').first()
item.order = 0
item.save()
# [('A', 0), ('C', 1), ('B', 2)]
item = Item.objects.filer(name='A').first()
item.delete()
# [('C', 0), ('B', 1)]
item = Item.objects.filer(name='B').first()
item.delete()
# [('C', 0)]
```

3.4 Other fields updated when order is changed

It is possible to specify other fields than the order field to be automatically updated when a field has its position changed by another field that was inserted/changed/deleted.

The update_auto_now setting will make sure that all date/datetime related fields that are taged to be automatically updated on change will be updated when the order is changed. This setting is default on, so remember to turn ot off if it is not wanted.

```
OrderedField(update_auto_now=True)
```

The extra_field_updates is a dictionary and it is used to specify other field to be updated when the order field is changed by anothers position change.

```
def get_loged_in_user():
    return "KGA"

OrderedField(extra_field_updates={
                                'order_changed_count': models.F("order_changed_count")␣
↪+ 1,
                                'updated_by': get_loged_in_user
                            })
```

The self_updates_on_collection_change parameter is used to specify fields to be updated when an instance changes collection. Unlike the extra_field_updates which is triggered when a records osition is changed when another field has its position changed the self_updates_on_collection_change works on the active instance and only when it changes collection.

```
def get_loged_in_user():
    return "KGA"

OrderedField(self_updates_on_collection_change={
                               'order_changed_count': models.F("order_changed_count")␣
↪+ 1,
                               'updated_by': get_loged_in_user
                           })
```

If self_updates_on_collection_change is the same as extra_field_updates like above then it is also possible to set the self_updates_on_collection_change_like_regular to True to avoid duplicating the settings.

```
def get_loged_in_user():
    return "KGA"

OrderedField(self_updates_on_collection_change_like_regular=True)
```

3.5 Model inheritance

NB: Remember to manually register the signals by using the add_signals method when using inheritance.

There are two ways to do regular inheritance. The first one is just to add inheritance without doing anything else. By doing this each model that inherit from it has its order related to its own table.

```
from django_ordered_field import (OrderedField, add_signals_for_inheritance)

class Unit(models.Model):
    name = models.CharField(max_length=100)
    position = OrderedField()


class Video(Unit):
    pass


add_signals_for_inheritance(Unit, Video, "position")


class Audio(Unit):
    pass


add_signals_for_inheritance(Unit, Audio, "position")


Video.objects.create(name="Video")
Quiz.objects.create(name="Audio")
print(list(Unit.objects.all().order_by("position").
                  values_list( "name", "position")))
# [("Video", 0), ("Audio", 0)]
```

The other method is to use the parent_link_name parameter. This will make the order field use the parrent model for its ordering.

```
from django_ordered_field import (OrderedField, add_signals_for_inheritance)

class Unit(models.Model):
    name = models.CharField(max_length=100)
    position = OrderedField(parent_link_name='unittwo_ptr')


class Video(Unit):
    pass


add_signals_for_inheritance(Unit, Video, "position")


class Audio(Unit):
    pass


add_signals_for_inheritance(Unit, Audio, "position")


Video.objects.create(name="Video")
Quiz.objects.create(name="Audio")
print(list(Unit.objects.all().order_by("position").
                  values_list( "name", "position")))
# [("Video", 0), ("Audio", 1)]
```

3.6 Abstract model

```
from django_ordered_field import OrderedField

class CommonInfo(models.Model):
    name = models.CharField(max_length=100)
    position = OrderedField()

    class Meta:
        abstract = True


class Person(CommonInfoTwo):
    description = models.CharField(max_length=100)
```

3.7 Proxy model

NB: Remember to manually register the signals by using the add_signals_for_proxy method when using inheritance.

```
from django_ordered_field import (OrderedField, add_signals_for_proxy)

class Person(models.Model):
    name = models.CharField(max_length=100)
    position = OrderedField()

```

(continues on next page)

(continued from previous page)

```
class PersonProxy(Person):

    class Meta:
        proxy = True


add_signals_for_proxy(Person, PersonProxy, "position")
```

3.8 Add signals

Current version has a limitation in a few circumstances than one has to mannually register some of the signals. If you use Proxy models or inherit from a model containing a order field then you have to manually register the signals.

Feel free to add a git pull request if you find a way to automatically register thise signals.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/kimgea/django-ordered-field/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

4.1.4 Write Documentation

django-ordered-field could always use more documentation, whether as part of the official django-ordered-field docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/kimgea/django-ordered-field/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *django-ordered-field* for local development.

1. Fork the *django-ordered-field* repo on GitHub.
2. Clone your fork locally:

```
$ git clone https://github.com/kimgea/django-ordered-field.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ virtualenv django-ordered-field-env
$ cd django-ordered-field
$ pip install -r requirements_dev.txt
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ pip install -r requirements_test.txt
$ flake8 django_ordered_field tests
$ python setup.py test
$ tox
```

The tests are not following pep8, but feel fre to clean them up.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6 and Django above version 2. Check https://travis-ci.org/kimgea/django-ordered-field/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python runtests.py tests.lists.tests.ChangeCollectionTest
```

CHAPTER 5

Credits

5.1 Development Lead

- Kim-Georg Aase <kim.georg.aase@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2018-04-28)

- First release on PyPI.