
Django Computed Property Documentation

Release 0.1

Jason Brechin

May 09, 2019

Contents

1	Prerequisites	3
2	Installation	5
3	Usage	7
3.1	Field types	8
4	Contributing	9

Computed Property fields for Django models, inspired by [Google Cloud NDB](#)

CHAPTER 1

Prerequisites

`django-computed-property` supports (i.e. is tested on) [Django](#) 1.8 - 2.2 and trunk on Python 2.7, 3.4, 3.5, 3.6, 3.7, pypy, and pypy3.

SQLite and Postgres are currently tested, but any Django database backend should work.

CHAPTER 2

Installation

`django-computed-property` is available on [PyPI](#). Install it with:

```
pip install django-computed-property
```


CHAPTER 3

Usage

Add `computed_property` to your list of `INSTALLED_APPS` in `settings.py`:

```
INSTALLED_APPS = [  
    ...  
    'computed_property'  
]
```

Then, simply import and use the included field classes in your models:

```
from django.db import models  
from computed_property import ComputedTextField  
  
class MyModel(models.Model):  
    name = ComputedTextField(compute_from='calculation')  
  
    @property  
    def calculation(self):  
        return 'some complicated stuff'
```

You can read values from the `name` field as usual, but you may not set the field's value. When the field is accessed and when a model instance is saved, it will compute the field's value using the provided callable (function/lambda), property name, or attribute name.

`compute_from` can be a reference to a function that takes a single argument (an instance of the model), or a string referring to a field, property, or other attribute on the instance.

Note: It is important to note that your computed field data will not immediately be written to the database. You must (re-)save all instances of your data to have the computed fields populated in the database. Until you do so, you will be able to access those fields when you load an instance of the model, but you will not benefit from their queryability.

One way you could do this is in a data migration, using something like:

```
for instance in MyModel.objects.all().iterator():
    instance.save()
```

3.1 Field types

Several other field classes are included: `ComputedCharField`, `ComputedEmailField`, `ComputedIntegerField`, `ComputedDateField`, `ComputedDateTimeField`, and others. All field classes accept the same arguments as their non-Computed versions.

To create an Computed version of some other field class, inherit from both `ComputedField` and the other field class:

```
from computed_property import ComputedField
from somewhere import MyField

class MyComputedField(ComputedField, MyField):
    pass
```

CHAPTER 4

Contributing

See the [contributing docs](#).