django-custom-query Documentation

Release 0.3.0

Luis Fagundes

Jan 17, 2019

Contents

django-custom-query is a python module to write user provided search queries, using AND, OR and parenthesis grouping. This module will translate those to Django ORM Q objects.

CHAPTER 1

Install

```
$ pip3 install django-custom-query
```

django-custom-query was developed and tested on Python 3.5. Is based on sqlparse.

CHAPTER 2

Source

Source can be downloaded as a tar.gz file from http://pypi.python.org/pypi/django-custom-query

Using git you can clone the source from http://github.com/lfagundes/django-custom-query.git

django-custom-query is free and open for usage under the MIT license.

CHAPTER 3

Documentation

Contents:

3.1 Using Django Custom Query

```
>>> from customquery import Parser
>>> from myapp import MyModel
>>>
>>> parser = Parser(MyModel)
>>> query = parser.parse("numberfield = 10 or (numberfield > 20 and numberfield < 30)
↪")
>>> items = MyModel.objects.filter(query)
```

Parser.parse() will create Django Q objects based on SQL-like condition statements.

The model is used for field validation and proper interpretation of field input:

```
>>> class MyModel(models.Model):
>>>     numberfield = models.IntegerField()
```

3.1.1 Foreign Keys

```
>>> class RelatedModel(models.Model):
>>>     name = models.CharField(max_length=16)
>>>
>>> class MyModel(models.Model):
>>>     related = models.ForeignKey(RelatedModel, on_delete=models.CASCADE)
>>>
>>> parser = Parser(MyModel)
>>> query = parser.parse('related__name="foo bar"')
>>> query = parser.parse('related.name="foo bar"') # dots can be used instead of __
```

3.1.2 Annotations

```
>>> from django.db.models import Value
>>> from django.db.models.functions import Concat
>>>
>>> qs = MyModel.objects.annotate(full_name=Concat('first_name', Value(' '), 'last_
↪name')).all()
>>> parser = Parser(qs)
>>> query = parser.parse('full_name="foo bar"')
```

3.1.3 Date formatting

```
>>> class MyModel(models.Model):
>>>     birthday = models.DateField()
>>>
>>> parser = Parser(MyModel, date_format='%d/%m/%Y')
>>> parser.parse('birthday=13/12/2018')
```

3.1.4 Operators

```
>>> parser.parse("numfield=1")      # Q(numfield=1))
>>> parser.parse("numfield = 1")    # Q(numfield=1))
>>> parser.parse("numfield  = 1")   # Q(numfield=1))
>>> parser.parse("numfield > 1")    # Q(numfield__gt=1))
>>> parser.parse("numfield >= 1")   # Q(numfield__gte=1))
>>> parser.parse("numfield < 1")    # Q(numfield__lt=1))
>>> parser.parse("numfield <= 1")   # Q(numfield__lte=1))
>>> parser.parse("numfield > 1")    # Q(numfield__gt=1))
>>> parser.parse("numfield >= 1")   # Q(numfield__gte=1))
>>> parser.parse("numfield < 1")    # Q(numfield__lt=1))
>>> parser.parse("numfield <= 1")   # Q(numfield__lte=1))
>>> parser.parse("numfield <> 1")   # ~Q(numfield=1))
>>> parser.parse("numfield != 1")   # ~Q(numfield=1))
>>> parser.parse("numfield NOT 1") # ~Q(numfield=1))
```

CHAPTER 4

API Reference

4.1 Parser

CHAPTER 5

Contributing

Please submit bugs and patches, preferably with tests. All contributors will be acknowledged. Thanks!

CHAPTER 6

Credits

django-custom-query was created by Luis Fagundes and was sponsored by Spatial Datalyst.

CHAPTER 7

Changelog

- 0.3.0
  - Support for annotations
  - Support for date formatting
  - NOT operator
  - IN operator
- 0.2.0
  - Support for filters on models.ForeignKey fields
- 0.1.0
  - Initial release, with support for integer and char fields, =, >, <, <=, <=, <>, !=, AND, OR, BETWEEN and parenthesis.