Django Property Filter

Release 1.0.0

Eric Ziethen

Mar 26, 2021

CONTENTS:

CHAPTER

ONE

OVERVIEW

1.1 Overview

Django-property-filter provides an extenstion to django-filter. It extend's django-filter's classes to provide additional support for filtering django models by properties.

The aim is to provide identical (where possible) functionality for properties as django-filter does for database fields. For this the new classes directly inherit their django-filter counterpart's features and the setup and configuration is aimed to be the same.

This means that the django-filter documentation can be applied to django-property-filter as well.

For example django-filter uses a class NumberFilter and django-property-filter extends it and creates PropertyNumberFilter supporting the same functionality and additional the possibility to filter properties as well.

Because property fields are not part of database tables they cannot be queried directly with sql and are therefore not natively supported by django and django-filter.

Django-property-filter also provides a filterset that can handle filters and property filters together.

1.2 How it works

Where django-filter directly applies the filtering to the queryset, django-property-filter can't do that because the properties are not database fields. To workaround this, all entries are compared in memory against all specified filters resulting in a list of matching primary keys. This list can then be used to filter the original queryset like this:

```
queryset.filter(pk__in=filtered_pk_list)
```

Because of this the actual filtering is happening in memory of the django application rather than in sql.

CHAPTER

TWO

INSTALLATION

2.1 Requirements

- **Python**: 3.6, 3.7, 3.8, 3.9
- **Django**: 2.2, 3.0, 3.1
- **Django-filter**: 2.3+

2.2 Installation

Install using pip:

```
pip install django-property-filter
```

Then add 'django_property_filter' to your INSTALLED_APPS.

```
INSTALLED_APPS = [
    ...
    'django_property_filter',
]
```

CHAPTER

THREE

GETTING STARTED

Django-property-filter provides extended functionality to django-filter to allow filtering by class properties by providing new Sub Classes to django-filter's Filter and Filterset classes.

All existing django-filter functionality is still working as before.

3.1 Example Model

Our Model:

```
from django.db import models

class BookSeries(models.Model):
    name = models.CharField(max_length=255)

    @property
    def book_count(self):
      return Book.objects.filter(series=self).count()

class Book(models.Model):
    title = models.CharField(max_length=255)
    price = models.DecimalField()
    discount_percentage = models.IntegerField()
    author = models.TextField()
    series = models.ForeignKey(BookSeries)

    @property
    def discounted_price(self):
      return self.price * self.discount_percentage / 100
```

3.2 Implicit Filter Creation

If we want to filter by discounted price as well as number of books in a series, which both are properties and not fields in the database, we would do the following.:

```
from django_property_filter import (
  PropertyFilterSet,
  PropertyNumberFilter
)
```

(continues on next page)

(continued from previous page)

```
class BookFilterSet(PropertyFilterSet):

  class Meta:
      model = Book
      exclude = ['price']
      property_fields = [
        ('discounted_price', PropertyNumberFilter, ['lt', 'exact']),
        ('series.book_count.', PropertyNumberFilter, ['gt', 'exact']),
      ]
```

This will create 4 Filters

1.) **A "less than" and an "exact" filter for the "discounted_price" property** of the Book Model

2.) **A "greater than" and an "exact" filter for the "book_count" property** of the related Model "series".

Since PropertyFilterSet is and extension to django-filter's Filterset which requires either the Meta attribute "fields" or "exclude" to be set we excluded the "price" field. If we had instead used:

```
fields = ['price']
```

It would also have created an "exact" filter for the book price.

The only difference to using a normal FilterSet from django-filter is the "property_fields" field.

The "property_fields" is a list of tuples with 3 values.

1.) **The property name.** If the property is on a related Model it should be separated by "__", and can span multiple levels e.g. fk__fk__fk__property

2.) **The specific Property Filter to use.** This is necessary since it can't be determined what the return type of the property will be in all cases

3.) The list of lookup expressions.

3.3 Explicit Filter Creation

It is also possible to create Filters explicitely:

```
from django_property_filter import PropertyNumberFilter, PropertyFilterSet

class BookFilterSet(PropertyFilterSet):
    prop_number = PropertyNumberFilter(field_name='discounted_price', lookup_expr='gte
↪')

    class Meta:
        model = NumberClass
        fields = ['prop_number']
```

This creates a "greater than or equel" filter for the discounted_price property

CHAPTER

FOUR

LIMITATIONS

4.1 Available filter expressions

Most but not all filter expressions that django-filter supports are supported. To see a list of available expressions for each property filter see *Filter Reference*

4.2 Property types

Because properties are evaluated at runtime the types cannot be predetermined beforehand like it is the case with database fields. Therefore there might be unexpected behaviour during filtering.

4.3 Performance

Because all the filtering is done in memory it performs slower than django-filter where the filtering happens directly in sql. This will be impacted by the numbers of filters used at the same time and the size of the data in the table.

4.4 Database limitations

In theory there is no limit for most databases how many results can be returned from a filter query unless the database implements a limit which will impact how many results django-property-filter can return.

sqlite

> **Warning:** Sqlite3 defines SQLITE_MAX_VARIABLE_NUMBER which is a limit for parameters passed to a query.
>
> See "Maximum Number Of Host Parameters In A Single SQL Statement" at https://www.sqlite.org/limits.html for further details.
>
> Depending on the version this limit might differ. By default from version 3.32.0 onwards, sqlite should have a default of 32766 while versions before this the limit was 999. A different limit can also be set at compile time and python is compiling their own sqlite version.
>
> For example Python 3.9.1 comes with sqlite version 3.33.0 and the 999 max parameter limitation still exists

Because of the way django-property-filter queries the database (i.e. with a prefilterd list of primary keys), the number of sql parameters needed might exceed the set limit.

Django-property-filter will try to return all values if possible, but if not possible it will try to return as many as possible limiting the sql parameters to not more than 999 and log a warning message similar to:

```
WARNING:root:Only returning the first 3746 items because of max parameter␣
↪limitations of Database "sqlite"
```

It is possible to set a custom limit via the environment variable "USER_DB_MAX_PARAMS". For example the user uses a custom compiled sqlite version with a different than the default value for SQLITE_MAX_VARIABLE_NUMBER the setting "USER_DB_MAX_PARAMS" to that value will use this value as a fallback rather than default values.

CHAPTER

FIVE

ADDITIONAL FILTERSET OPTIONS

This document provides a guide on using additional PropertyFilterSet features in addition to FilterSet.

5.1 Meta options

- *property_fields*

5.1.1 Automatic filter generation with `property_fields`

The `PropertyFilterSet` is capable of automatically generating filters for a given clas Properties accessible by the `model` or it's related models.

```
class BookFilterSet(PropertyFilterSet):

  class Meta:
      model = Book
      exclude = ['price']
      property_fields = [
        ('discounted_price', PropertyNumberFilter, ['lt', 'exact']),
        ('series__book_count.', PropertyNumberFilter, ['gt', 'exact']),
      ]
```

The "property_fields" is a list of tuples with 3 values.

1.) The property name. If the property is on a related Model it should be separated by "__", and can span multiple levels e.g. fk__fk__fk__property

2.) The specific Property Filter to use. This is necessary since it can't be determined what the return type of the property will be in all cases

3.) The list of lookup expressions.

CHAPTER

SIX

FILTER REFERENCE

This is a reference document with a list of the filters and their property specific arguments specific for property filters.

6.1 Filter to Property Filter Mapping

The following tables shows the corresponding Property Filters for Filters from django-filters.

Filter	Property Filter
AllValuesFilter	PropertyAllValuesFilter
AllValuesMultipleFilter	PropertyAllValuesMultipleFilter
BaseCSVFilter	PropertyBaseCSVFilter
BaseInFilter	PropertyBaseInFilter
BaseRangeFilter	PropertyBaseRangeFilter
BooleanFilter	PropertyBooleanFilter
CharFilter	PropertyCharFilter
ChoiceFilter	PropertyChoiceFilter
DateFilter	PropertyDateFilter
DateFromToRangeFilter	PropertyDateFromToRangeFilter
DateRangeFilter	PropertyDateRangeFilter
DateTimeFilter	PropertyDateTimeFilter
DateTimeFromToRangeFilter	PropertyDateTimeFromToRangeFilter
DurationFilter	PropertyDurationFilter
Filter	Property Filter
IsoDateTimeFilter	PropertyIsoDateTimeFilter
IsoDateTimeFromToRangeFilter	PropertyIsoDateTimeFromToRangeFilter
LookupChoiceFilter	PropertyLookupChoiceFilter
ModelChoiceFilter	N/A (Not needed because filtering foreign key
ModelMultipleChoiceFilter	N/A (Not needed because filtering foreign key
MultipleChoiceFilter	PropertyMultipleChoiceFilter
NumberFilter	PropertyNumberFilter
NumericRangeFilter	PropertyNumericRangeFilter
OrderingFilter	PropertyOrderingFilter
RangeFilter	PropertyRangeFilter
TimeFilter	PropertyTimeFilter
TimeRangeFilter	PropertyTimeRangeFilter
TypedChoiceFilter	PropertyTypedChoiceFilter
TypedMultipleChoiceFilter	PropertyTypedMultipleChoiceFilter
UUIDFilter	PropertyUUIDFilter

6.2 Supported Property Filter Expressions

The following tables shows the supported lookup expressions and hightlights the default one if none is specified.

Property Filter	Supported Expressions
PropertyAllValuesFilter	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyAllValuesMultipleFilter	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyBaseCSVFilter	**in**, range
PropertyBaseInFilter	**in**
PropertyBaseRangeFilter	**range**
PropertyBooleanFilter	**exact**, isnull
PropertyCharFilter	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyChoiceFilter[2]	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyDateFilter	**exact**, gt, gte, lt, lte
PropertyDateFromToRangeFilter	**range**
PropertyDateRangeFilter	**exact**
PropertyDateTimeFilter	**exact**, gt, gte, lt, lte
PropertyDateTimeFromToRangeFilter	**range**
PropertyDurationFilter	**exact**, gt, gte, lt, lte
PropertyIsoDateTimeFilter	**exact**, gt, gte, lt, lte
PropertyIsoDateTimeFromToRangeFilter	**range**
PropertyLookupChoiceFilter[2]	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyMultipleChoiceFilter[2]	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyNumberFilter	**exact**, contains, gt, gte, lt, lte, startswith, endswith
PropertyNumericRangeFilter[1]	**exact**, contains, contained_by, overlap
PropertyOrderingFilter[3]	**exact**
PropertyRangeFilter	**range**
PropertyTimeFilter	**exact**, gt, gte, lt, lte
PropertyTimeRangeFilter	**range**
PropertyTypedChoiceFilter[?]	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyTypedMultipleChoiceFilter[?]	**exact**, iexact, contains, icontains, gt, gte, lt, lte, startswith, istartswith, endswith, iendswith
PropertyUUIDFilter	**exact**

[2] Explicit Creation only, choices need to be passed (see *Explicit Filter Creation*)
[1] Postgres only
[3] see *PropertyOrderingFilter*

6.3 Supported Base Lookup Expressions

This is a list lookup expressions supported by all Property Filters unless excludes specifically.

Filter Expression	Purpose
contained_by	Subset of the given value
contains	Contains value (case sensitive)
endswith	Ends with value (case sensitive)
exact	Matches value exact (case sensitive)
gt	Greater than
gte	Greater than or equal
icontains	Contains value (case insensitive)
iendswith	Ends with value (case sensitive)
iexact	Matches value exact (case insensitive)
in	Matches specified list of values or range
isnull	Is null
istartswith	Starts with value (case sensitive)
lt	Less than
lte	Less than or equal
overlap	Overlapping with the given value
range	Part of the given range
startswith	Starts with value (case sensitive)

Warning: Sqlite by default uses case insensitive text comparison, so e.g. 'exact' and 'iexact' will give the same result. Even if turning on case sensitivity with PRAGMA case_sensitive_like, both still result in the same result.

Django-property-filter will behave as normally expected in this case and will correctly check for case sensitivity.

6.4 Invalid Type Comparison

When the selected Filter Type and comparison is incompatible with the type the the property returns that queryset entry will not be a match and an error is logged similar to

> Error during comparing property value "15" with filter value "text" with error: "'<' not supported between instances of 'int' and 'str'"

6.5 Core Arguments

6.5.1 `field_name`

The name of the property to lookup.

This can be

1.) Property directly on the model e.g. "field_name='my_property'"

2.) **A Related field property e.g. "field_name='related__my_property'"** which can span as many models as are related

6.5.2 lookup_expr

The lookup expression to filter against. The default lookup expression when not specified will be 'exact' if the filter supports it. Some filters only support 'range' and this will be the default.

6.6 Appendix

6.6.1 PropertyOrderingFilter

Because the field parameters are passed as arguments this filter can only be created explicitely. For example:

```
prop_age = PropertyOrderingFilter(fields=('prop_age', 'prop_age'))
```

Warning: Sorting is all happening in memory rather than sql. Since this filter depends on sorted querysets, the sorting loads the values into memory first and therefore can make it an expensive operator. Carefull with larger data sets.

Because of the in memory sorting, sorting is only supported by a single property

CHAPTER

SEVEN

DEVELOPMENT AND TESTING

7.1 Run the Test Suite locally

For running tests using sqlite use either

```
Windows: $ dev\run_tests.bat sqlite
Linux:   $ dev/run_tests.sh sqlite
```

or for postgresql (needs local postgres setup first)

```
Windows: $ dev\run_tests.bat postgres-local
Linux:   $ dev/run_tests.sh postgres-local
```

7.2 Run the Linters

```
Windows: $ dev\run_linters.bat
Linux:   $ dev/run_linters.sh
```

7.3 Run the Django Test Project

Change to the test project directory setup and run the django project

```
$ cd tests\django_test_proj
$ python manage.py migrate
$ python manage.py setup_data
$ python manage.py runserver
```

By default sqlite is used, but postgresql is also supported. For this set the environment variable to the local postgres settings

DJANGO_SETTINGS_MODULE=django_test_proj.settings_postgres_local