

---

# **django-querysetsequence**

***Release 0.17***

**Jun 27, 2023**



---

## Contents:

---

<b>1</b>	<b>Getting Started</b>	<b>3</b>
<b>2</b>	<b>Overview</b>	<b>5</b>
<b>3</b>	<b>Getting Started</b>	<b>7</b>
<b>4</b>	<b>Basic Usage</b>	<b>9</b>
<b>5</b>	<b>Project Information</b>	<b>11</b>
<b>6</b>	<b>Full Table of Contents</b>	<b>13</b>
<b>7</b>	<b>Indices and tables</b>	<b>25</b>



Release v0.17 (*What's new?* <changelog>).



# CHAPTER 1

---

## Getting Started

---

`django-querysetsequence` adds helpers for treating multiple disparate `QuerySet` objects as a single `QuerySet`. This is useful for passing into APIs that only accepted a single `QuerySet`.

The `QuerySetSequence` wrapper is used to combine multiple `QuerySet` instances.





## CHAPTER 2

---

### Overview

---

`QuerySequence` aims to provide the same behavior as Django's `QuerySets`, but applied across multiple `QuerySet` instances.

Supported features:

- Methods that take a list of fields (e.g. `filter()`, `exclude()`, `get()`, `order_by()`) must use fields that are common across all sub-`QuerySets`.
- Relationships across related models work (e.g. `'foo__bar'`, `'foo'`, or `'foo_id'`). syntax).
- The sub-`QuerySets` are evaluated as late as possible (e.g. during iteration, slicing, pickling, `repr()/len()/list()/bool()` calls).
- Public `QuerySet` API methods that are untested/unimplemented raise `NotImplementedError`.



## CHAPTER 3

---

### Getting Started

---

Install the package using pip.

```
pip install --upgrade django-querysetsequence
```



## CHAPTER 4

---

### Basic Usage

---

```
# Import QuerySetSequence
from queryset_sequence import QuerySetSequence

# Create QuerySets you want to chain.
from .models import SomeModel, OtherModel

# Chain them together.
query = QuerySetSequence(SomeModel.objects.all(), OtherModel.objects.all())

# Use query as if it were a QuerySet! E.g. in a ListView.
```



## CHAPTER 5

---

### Project Information

---

django-querysetsequence is released under the ISC license, its documentation lives on [Read the Docs](#), the code on [GitHub](#), and the latest release on [PyPI](#). It supports Python 3.7+, Django 3.2/4.0/4.1/4.2, and is optionally compatible with [Django REST Framework 3.11+](#).

Some ways that you can contribute:

- Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug.
- Fork the repository on GitHub to start making your changes.
- Write a test which shows that the bug was fixed or that the feature works as expected.
- Send a pull request and bug the maintainer until it gets merged and published.





## 6.1 Example usage

Below is a fuller example of how to use a `QuerySetSequence`. Two similar, but not identical models exist (Article and Book):

```
class Author(models.Model):
    name = models.CharField(max_length=50)

    class Meta:
        ordering = ['name']

    def __str__(self):
        return self.name

class Article(models.Model):
    title = models.CharField(max_length=100)
    author = models.ForeignKey(Author)

    def __str__(self):
        return "%s by %s" % (self.title, self.author)

class Book(models.Model):
    title = models.CharField(max_length=50)
    author = models.ForeignKey(Author)
    release = models.DateField(auto_now_add=True)

    def __str__(self):
        return "%s by %s" % (self.title, self.author)
```

We'll also want some data to illustrate how `QuerySetSequence` works:

```
# Create some data.
alice = Author.objects.create(name='Alice')
article = Article.objects.create(title='Dancing with Django', author=alice)

bob = Author.objects.create(name='Bob')
article = Article.objects.create(title='Django-isms', author=bob)
article = Book.objects.create(title='Biography', author=bob)

# Create some QuerySets.
books = Book.objects.all()
articles = Article.objects.all()
```

By wrapping a `QuerySet` of each into a `QuerySetSequence` they can be treated as a single `QuerySet`, for example we can filter to a particular author's work, or alphabetize all all articles and books together.

```
# Combine them into a single iterable.
published_works = QuerySetSequence(books, articles)

# Find Bob's titles.
bob_works = published_works.filter(author=bob)
# Still an iterable.
print([w.title for w in bob_works]) # prints: ['Biography', 'Django-isms']

# Alphabetize the QuerySet.
published_works = published_works.order_by('title')
print([w.title for w in published_works]) # prints ['Biography', 'Dancing with Django
↪', 'Django-isms']
```

## 6.1.1 Django REST Framework integration

django-querysetsequence comes with a custom `CursorPagination` class that helps integration with Django REST Framework. It is optimized to iterate over a `QuerySetSequence` first by `QuerySet` and then by the normal ordering configuration. This uses the optimized code-path for iteration that avoids interleaving the individual `QuerySets`.

To handle exceptions and filtering correctly, a model must be specified when creating the `QuerySetSequence`. Note that an abstract model may be used.

For example:

```
from queryset_sequence.pagination import SequenceCursorPagination

class PublicationPagination(SequenceCursorPagination):
    ordering = ['author', 'title']

class PublicationViewSet(viewsets.ModelViewSet):
    pagination_class = PublicationPagination

    def get_queryset(self):
        # This will return all Books first, then all Articles. Each of those
        # is individually ordered by ``author``, then ``title``.
        return QuerySetSequence(Book.objects.all(), Article.objects.all(), model=Book)
```

## 6.2 API Reference

Much of the `QuerySet` API is implemented by `QuerySetSequence`, but it is not fully compatible.

### 6.2.1 Summary of Supported APIs

Table 1: Methods that return new `QuerySets`

Method	Implemented?	Notes
<code>filter()</code>	✓	See <sup>1</sup> for information on the <code>QuerySet</code> lookup: <code>'#'</code> .
<code>exclude()</code>	✓	See <sup>1</sup> for information on the <code>QuerySet</code> lookup: <code>'#'</code> .
<code>annotate()</code>	✓	
<code>alias()</code>		
<code>order_by()</code>	✓	Does not support random ordering (e.g. <code>order_by('?')</code> ). See <sup>1</sup> for information on the <code>QuerySet</code> lookup: <code>'#'</code> .
<code>reverse()</code>	✓	
<code>distinct()</code>	✓	Does not support calling <code>distinct()</code> if there are multiple underlying <code>QuerySet</code> instances of the same model.
<code>values()</code>	✓	See <sup>1</sup> for information on including the <code>QuerySet</code> index: <code>'#'</code> .
<code>values_list()</code>	✓	See <sup>1</sup> for information on including the <code>QuerySet</code> index: <code>'#'</code> .
<code>dates()</code>		
<code>datetimes()</code>		
<code>none()</code>	✓	
<code>all()</code>	✓	
<code>union()</code>		
<code>intersection()</code>		
<code>difference()</code>		
<code>select_related()</code>	✓	
<code>prefetch_related()</code>	✓	
<code>extra()</code>	✓	
<code>defer()</code>	✓	
<code>only()</code>	✓	
<code>using()</code>	✓	
<code>select_for_update()</code>		
<code>raw()</code>		

<sup>1</sup> `QuerySetSequence` supports a special field lookup that looks up the index of the `QuerySet`, this is represented by `'#'`. This can be used in any of the operations that normally take field lookups (i.e. `filter()`, `exclude()`, and `get()`), as well as `order_by()` and `values()`. A few examples are below:

```
# Order first by QuerySet, then by the value of the 'title' field.
QuerySetSequence(...).order_by('#', 'title')

# Filter out the first QuerySet.
QuerySetSequence(...).filter(**{'#__gt': 0})
```

**Note:** Ordering first by `QuerySet` allows for a more optimized code path when iterating over the entries.

Table 2: Operators that return new QuerySets

Operator	Implemented?	Notes
<code>AND (&amp;)</code>	✓	A <code>QuerySetSequence</code> can be combined with a <code>QuerySet</code> . The <code>QuerySets</code> in the <code>QuerySetSequence</code> are filtered to ones matching the same Model. Each of those is ANDed with the other <code>QuerySet</code> .
<code>OR ( )</code>	✓	A <code>QuerySetSequence</code> can be combined with a <code>QuerySet</code> or <code>QuerySetSequence</code> . When combining with a <code>QuerySet</code> , it is added to the <code>QuerySetSequence</code> . Combining with another <code>QuerySetSequence</code> adds together the two underlying sets of <code>QuerySets</code> .

Table 3: Methods that do not return QuerySets

Method	Implemented?	Notes
<code>get()</code>	✓	See <sup>1</sup> for information on the <code>QuerySet</code> lookup: '# '.
<code>aget()</code>	✓	
<code>create()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>acreate()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>get_or_create()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>aget_or_create()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>update_or_create()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>aupdate_or_create()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>bulk_create()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>abulk_create()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>bulk_update()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>abulk_update()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>count()</code>	✓	
<code>acount()</code>	✓	
<code>in_bulk()</code>		Cannot be implemented in <code>QuerySetSequence</code> .
<code>ain_bulk()</code>		Cannot be implemented in <code>QuerySetSequence</code> .

Continued on next page

**Warning:** Not all lookups are supported when using '# ' (some lookups simply don't make sense; others are just not supported). The following are allowed:

- `exact`
- `ieexact`
- `contains`
- `icontains`
- `in`
- `gt`
- `gte`
- `lt`
- `lte`
- `startswith`
- `istartswith`
- `endswith`
- `iendswith`
- `range`

Table 3 – continued from previous page

Method	Implemented?	Notes
<code>iterator()</code>	✓	
<code>aiterator()</code>		
<code>latest()</code>	✓	If no fields are given, <code>get_latest_by</code> on each model is required to be identical.
<code>alatest()</code>		
<code>earliest()</code>	✓	See the documentation for <code>latest()</code> .
<code>aearliest()</code>		
<code>first()</code>	✓	If no ordering is set this is essentially the same as calling <code>first()</code> on the first <code>QuerySet</code> , if there is an ordering, the result of <code>first()</code> for each <code>QuerySet</code> is compared and the “first” value is returned.
<code>afirst()</code>		
<code>last()</code>	✓	See the documentation for <code>first()</code> .
<code>alast()</code>		
<code>aggregate()</code>		
<code>aaggregate()</code>		
<code>exists()</code>	✓	
<code>aexists()</code>	✓	
<code>contains()</code>	✓	
<code>acontains()</code>		
<code>update()</code>	✓	
<code>aupdate()</code>		
<code>delete()</code>	✓	
<code>adelete()</code>		
<code>as_manager()</code>	✓	
<code>explain()</code>	✓	
<code>aexplain()</code>		

Table 4: Additional methods specific to `QuerySetSequence`

Method	Notes
<code>get_querysets()</code>	Returns the list of <code>QuerySet</code> objects that comprise the sequence. Note, if any methods have been called which modify the <code>QuerySetSequence</code> , the <code>QuerySet</code> objects returned by this method will be similarly modified. The order of the <code>QuerySet</code> objects within the list is not guaranteed.

## 6.3 Changelog

### 6.3.1 0.17 (2023-06-27)

#### Bugfixes

- None values are now appropriately sorted first or last (depending on database support. Contributed by @vuongdv-spinshell. (#97)

#### Improvements

- Initial support for asynchronous queries. (#99, #103)

## Maintenance

- Support Python 3.11. (#101)
- Support Django 4.0, 4.1, and 4.2. (#83, #102)
- Support Django REST Framework 3.14. (#101)
- Drop support for Python 3.7. (#102)
- Drop support for Django 2.2 and 3.1. (#98)
- Drop support for Django REST Framework < 3.11. (#98)

## 6.3.2 0.16 (2022-04-01)

### Improvements

- Fix `QuerySetSequence`'s support with Django REST Framework's `DjangoFilterBackend` by accepting a `model` parameter. If one is not provided, a dummy model is used to provide a reasonable `DoesNotExist` error. Contributed by @j0nm1. (#88)

## Maintenance

- Support Python 3.10. (#86)
- Support Django REST Framework 3.13. (#86)
- Drop support for Python 3.6. (3fc1d0f)
- Improve package metadata. (#89)
- Run `black`, `isort`, `flake8`, and `pyupgrade`. (#90, #91)

## 6.3.3 0.15 (2021-12-10)

### Improvements

- Support the `contains()` method. (#85)

## Maintenance

- Support Django 4.0. (#83)
- Drop support for Django 3.0. (#83)
- Changed packaging to use `setuptools` declarative config in `setup.cfg`. (#84)

## 6.3.4 0.14 (2021-02-26)

### Improvements

- Support the `values()` and `values_list()` methods. (#73, #74)

- Support the `distinct()` method when each `QuerySet` instance is from a unique model. Contributed by @jpic. (#77, #80)
- Add [Sphinx documentation](#) which is available at Read the Docs.

### Bugfixes

- Support calling `filter()` with `Q()` objects. Contributed by @jpic. (#76)

### Maintenance

- Support Python 3.9. (#78)
- Support Django 3.2. (#78, #81)
- Support Django REST Framework 3.12. (#75)
- Drop support for Python 3.5. (#82)
- Add an additional test for the interaction of `order_by()` and `only()`. (#72)
- Switch continuous integration to GitHub Actions. (#79)

## 6.3.5 0.13 (2020-07-27)

### Bugfixes

- `explain()` now passes through parameters to the underlying `QuerySet` instances. (#69)
- Fixes compatibility issue with `ModelChoiceField`. Contributed by @jpic. (#68)

### Maintenance

- Support Django 3.1. (#69)
- Drop support for Django < 2.2. (#70)

## 6.3.6 0.12 (2019-12-20)

### Bugfixes

- Do not use `is not` to compare to an integer literal. (#61)

### Maintenance

- Support Python 3.8. (#59)
- Support Django 3.0. (#59)
- Support Django REST Framework 3.10 and 3.11. (#59, #64)
- Drop support for Python 2.7. (#59)
- Drop support for Django 2.0 and 2.1. (#59)

### 6.3.7 0.11 (2019-04-25)

#### Improvements

- Add a `QuerySetSequence` specific method: `get_querysets()`. Contributed by @optiz0r. (#53)

#### Maintenance

- Support Python 3.7. Contributed by @michael-k. (#51)
- Support Django 2.2. Contributed by @michael-k. (#51)
- Support Django REST Framework 3.9. Contributed by @michael-k. (#51)
- Drop support for Python 3.4. Contributed by @michael-k. (#51)
- Drop support for Django REST Framework < 3.6.3. Contributed by @michael-k. (#51)

### 6.3.8 0.10 (2018-10-09)

#### Improvements

- Support `first()`, `last()`, `latest()`, and `earliest()` methods. (#40, #49)
- Support the `&` and `|` operators. (#41)
- Support `defer()` and `only()` methods to control which fields are returned. (#44)
- Support calling `using()` to switch databases for an entire `QuerySetSequence`. (#44)
- Support calling `extra()`, `update()`, and `annotate()` which get applied to each `QuerySet`. (#46, #47)
- Support calling `explain()` on Django `>= 2.1`. (#48)

#### Bugfixes

- Raise `NotImplementedError` on unimplemented methods. This fixes a regression introduced in 0.9. (#42)
- Expand tests for empty `QuerySet` instances. (#43)

### 6.3.9 0.9 (2018-09-20)

#### Bugfixes

- Stop using the internals of `QuerySet` for better forward compatibility. This change means that `QuerySetSequence` is no longer a sub-class of `QuerySet` and should improve interactions with other packages which modify `QuerySet`. (#38)



## Maintenance

- Support Django 2.0 and 2.1. Contributed by @michael-k. (#35, #39)
- Support Django REST Framework 3.7 and 3.8. (#33, #39)
- Drop support for Django < 1.11. (#36)
- Drop support for Django REST Framework < 3.4. (#36)

### 6.3.10 0.8 (2017-09-05)

## Improvements

- Optimize iteration when *not* slicing a `QuerySetSequence`. Contributed by @EvgeneOskin. (#29)

## Maintenance

- Support Django 1.11. Contributed by @michael-k. (#26, #32)
- Support Django REST Framework 3.5 and 3.6. (#26)

### 6.3.11 0.7.2 (2017-04-04)

## Bugfixes

- Calling an unimplemented method with parameters on `QuerySetSequence` raised a non-sensical error. (#28)

### 6.3.12 0.7.1 (2017-03-31)

## Bugfixes

- Slicing a `QuerySetSequence` did not work properly when the slice reduced the `QuerySetSequence` to a single `QuerySet`. (#23, #24)
- Typo fixes. (#19)

## Maintenance

- Support Django REST Framework 3.5. (#20)

### 6.3.13 0.7 (2016-10-20)

## Improvements

- Allow filtering / querying / ordering by the order of the `QuerySets` in the `QuerySetSequence` by using '#'. This allows for additional optimizations when using third-party applications, e.g. Django REST Framework. (#10, #14, #15, #16)

- [Django REST Framework](#) integration: includes a subclass of the `CursorPagination` from Django REST Framework under `queryset_sequence.pagination.SequenceCursorPagination` which is designed to work efficiently with a `QuerySetSequence` by first ordering by internal `QuerySet`, then by the ordering attribute. (#17)

### Bugfixes

- `PartialInheritanceMeta` must be provided `INHERITED_ATTRS` and `NOT_IMPLEMENTED_ATTRS`. (#12)

### Maintenance

- Move `queryset_sequence` to an actual module in order to hide some implementation details. (#11)

## 6.3.14 0.6.1 (2016-08-03)

### Maintenance

- Support Django 1.10. (#9)

## 6.3.15 0.6 (2016-06-07)

### Improvements

- Allow specifying the `Model` to use when instantiating a `QuerySetSequence`. This is required for compatibility with some third-party applications that check the `model` field for equality, e.g. when using the `DjangoFilterBackend` with Django REST Framework. Contributed by [@CountZachula](#). (#6)
- Support `prefetch_related`. (#7)

### Bugfixes

- Fixes an issue when using Django Debug Toolbar. (#8)

## 6.3.16 0.5 (2016-02-21)

### Improvements

- Significant performance improvements when ordering the `QuerySetSequence`. (#5)
- Support `delete()` to remove items. (1bb1716)

## 6.3.17 0.4 (2016-02-03)

### Maintenance

- Support Python 3.4 and 3.5. Contributed by [@jpic](#). (#3)

### 6.3.18 0.3 (2016-01-29)

#### Improvements

- Raises `NotImplementedError` for `QuerySet` methods that `QuerySetSequence` does not implement. (e2c67c5, b376b87)
- Support `reverse()` to reverse the item ordering. (f27b2c7)
- Support `none()` to return an `EmptyQuerySet`. (6171c11)
- Support `exists()` to check if a `QuerySetSequence` has any results. (1aa705b)
- Support `select_related` to follow foreign-key relationships when generating results. (ad54d5e)

#### Bugfixes

- Do not evaluate any `QuerySets` when calling `filter()` or `exclude()` like a Django `QuerySet`. Contributed by @jpic. (#1, baaf448)
- Do not cache the results when calling `iterator()`. (6566a91)

### 6.3.19 0.2.4 (2016-01-21)

#### Improvements

- Support `order_by()` that references a related model (e.g. a `ForeignKey` relationship using `foo` or `foo_id` syntaxes). (94274d6)
- Support `order_by()` that references a field on a related model (e.g. `foo__bar`) (a97d940)

#### Maintenance

- Support Django 1.9.1. (9497e09)

### 6.3.20 0.2.3 (2016-01-11)

#### Bugfixes

- Fixed calling `order_by()` with a single field. (5c8521c)

### 6.3.21 0.2.2 (2016-01-08)

#### Improvements

- Support the `get()` method on `QuerySetSequence`. (957a650)

### 6.3.22 0.2.1 (2016-01-08)

#### Bugfixes

- Fixed a bug when there's no data to iterate. (02aafac)

### 6.3.23 0.2 (2016-01-08)

#### Bugfixes

- Do not try to instantiate `EmptyQuerySet`. (99dba06)

#### Maintenance

- Fixed packaging. (9b1ae74)

### 6.3.24 0.1 (2016-01-07)

- Support Django 1.8.0.
- Various bug fixes and tests.

The initial commits on based on DjangoSnippets and other code:

- DjangoSnippet 1103 by mattdw. foo\_7a081bfcfc0eff2aba4d550632d9733786c65ac8
- DjangoSnippet 1253 by joonas.  
foo\_8d989bcc36140573a0f4d5f1e0e1e99e9a90a9f4
  - Updated per comment 1553 by nosa\_manuel. foo\_ff258ca20f2a5c8e536a744fb9b64fba87046ef5
  - Updated per comment 4642 by esquevin. foo\_04b5fe14a5e8803c2b11259ff60c095fb9da8ce3
- DjangoSnippet 1933 by t\_rybik. foo\_93f5575b3661bd2334960767eadf4a1ba03bfb8f
- django-ko-demo from The Atlantic by @fdintino. foo\_0b875aeb8aaca20ba47fc2fbc285d078aee42240

## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`