
graphene-elastic Documentation

Release 0.8

Artur Barseghyan <artur.barseghyan@gmail.com>

Dec 25, 2022

CONTENTS

1 Prerequisites	3
2 Main features and highlights	5
3 Demo	7
4 Documentation	9
5 Installation	11
6 Examples	13
6.1 Install requirements	13
6.2 Populate sample data	13
6.3 Sample document definition	13
6.4 Sample apps	14
6.4.1 Sample Flask app	14
6.4.2 Sample Django app	15
6.4.3 ConnectionField example	15
6.4.3.1 Filter	18
6.4.3.1.1 Sample queries	18
6.4.3.1.2 Implemented filter lookups	19
6.4.3.2 Search	20
6.4.3.3 Ordering	20
6.4.3.4 Pagination	21
6.4.3.5 Highlighting	21
7 Road-map	23
8 Testing	25
8.1 Testing with Docker	25
8.2 Running tests with virtualenv or tox	25
9 Debugging	27
10 Writing documentation	29
11 License	31
12 Support	33
13 Author	35

14	Project documentation	37
14.1	Concepts	38
14.1.1	Sample document definition	38
14.1.2	Sample schema definition	39
14.1.2.1	filter_backends	42
14.1.2.2	filter_fields	42
14.1.2.3	search_fields	43
14.1.2.4	ordering_fields	43
14.1.2.5	ordering_defaults	44
14.2	Quick start	44
14.2.1	Clone the repository	44
14.2.2	Start Elasticsearch	44
14.2.3	Install requirements	44
14.2.4	Populate dummy data	44
14.2.5	Run the test server	44
14.2.6	Open the <code>graphiql</code> client the browser	44
14.2.7	Make some experiments	45
14.2.8	Run tests	45
14.3	Search	45
14.4	Filtering	46
14.4.1	Filter lookups	46
14.4.1.1	Filter lookup <code>contains</code>	47
14.4.1.2	Filter lookup <code>ends_with</code>	47
14.4.1.3	Filter lookup <code>exclude</code>	47
14.4.1.4	Filter lookup <code>exists</code>	48
14.4.1.5	Filter lookup <code>gt</code>	48
14.4.1.6	Filter lookup <code>gte</code>	49
14.4.1.7	Filter lookup <code>in</code>	49
14.4.1.8	Filter lookup <code>lt</code>	49
14.4.1.9	Filter lookup <code>lte</code>	50
14.4.1.10	Filter lookup <code>prefix</code>	50
14.4.1.11	Filter lookup <code>range</code>	50
14.4.1.12	Filter lookup <code>starts_with</code>	51
14.4.1.13	Filter lookup <code>term</code>	51
14.4.1.14	Filter lookup <code>terms</code>	51
14.4.1.15	Filter lookup <code>wildcard</code>	52
14.5	Post-filter backend	52
14.5.1	Filter lookups	52
14.5.1.1	Filter lookup <code>contains</code>	53
14.5.1.2	Filter lookup <code>ends_with</code>	53
14.5.1.3	Filter lookup <code>exclude</code>	53
14.5.1.4	Filter lookup <code>exists</code>	54
14.5.1.5	Filter lookup <code>gt</code>	54
14.5.1.6	Filter lookup <code>gte</code>	55
14.5.1.7	Filter lookup <code>in</code>	55
14.5.1.8	Filter lookup <code>lt</code>	55
14.5.1.9	Filter lookup <code>lte</code>	56
14.5.1.10	Filter lookup <code>prefix</code>	56
14.5.1.11	Filter lookup <code>range</code>	56
14.5.1.12	Filter lookup <code>starts_with</code>	57
14.5.1.13	Filter lookup <code>term</code>	57
14.5.1.14	Filter lookup <code>terms</code>	57
14.5.1.15	Filter lookup <code>wildcard</code>	57
14.6	Ordering	58

14.7	Highlight	59
14.8	Source	61
14.9	Score	62
14.10	Faceted search	64
14.11	Query string backend	67
14.12	Simple query string backend	69
14.13	Pagination	71
14.13.1	Limits	71
14.13.2	Enforce first or last	71
14.13.3	User controlled pagination	71
14.14	Custom filter backends	72
14.15	Settings	74
14.16	Running Elasticsearch	75
14.16.1	Docker	75
14.16.1.1	Project default	75
14.16.1.2	Run specific version	75
14.16.1.2.1	6.x	75
14.16.1.2.2	7.x	76
14.17	FAQ	76
14.17.1	Questions and answers	76
14.18	Debugging	76
14.18.1	Logging queries to console	76
14.19	Release history and notes	77
14.19.1	0.8.1	77
14.19.2	0.8	77
14.19.3	0.7	77
14.19.4	0.6.6	78
14.19.5	0.6.5	78
14.19.6	0.6.4	78
14.19.7	0.6.3	78
14.19.8	0.6.2	78
14.19.9	0.6.1	79
14.19.100.6		79
14.19.110.5		80
14.19.120.4		80
14.19.130.3		80
14.19.140.2		80
14.19.150.1		80
14.19.160.0.13		80
14.19.170.0.12		81
14.19.180.0.11		81
14.19.190.0.10		81
14.19.200.0.9		81
14.19.210.0.8		81
14.19.220.0.7		81
14.19.230.0.6		82
14.19.240.0.5		82
14.19.250.0.4		82
14.19.260.0.3		82
14.19.270.0.2		82
14.19.280.0.1		82
14.20	graphene_elastic package	83
14.20.1	Subpackages	83
14.20.1.1	graphene_elastic.filter_backends package	83

14.20.1.1.1	Subpackages	83
14.20.1.1.1.1	graphene_elastic.filter_backends.faceted_search package	83
14.20.1.1.1.2	Submodules	83
14.20.1.1.1.3	graphene_elastic.filter_backends.faceted_search.common module	83
14.20.1.1.1.4	Module contents	85
14.20.1.1.1.5	graphene_elastic.filter_backends.filtering package	85
14.20.1.1.1.6	Submodules	85
14.20.1.1.1.7	graphene_elastic.filter_backends.filtering.common module	85
14.20.1.1.1.8	graphene_elastic.filter_backends.filtering.mixins module	89
14.20.1.1.1.9	graphene_elastic.filter_backends.filtering.queries module	100
14.20.1.1.1.10	Module contents	100
14.20.1.1.1.11	graphene_elastic.filter_backends.highlight package	100
14.20.1.1.1.12	Submodules	100
14.20.1.1.1.13	graphene_elastic.filter_backends.highlight.common module	100
14.20.1.1.1.14	Module contents	102
14.20.1.1.1.15	graphene_elastic.filter_backends.ordering package	102
14.20.1.1.1.16	Submodules	102
14.20.1.1.1.17	graphene_elastic.filter_backends.ordering.common module	102
14.20.1.1.1.18	Module contents	107
14.20.1.1.1.19	graphene_elastic.filter_backends.post_filter package	107
14.20.1.1.1.20	Submodules	107
14.20.1.1.1.21	graphene_elastic.filter_backends.post_filter.common module	107
14.20.1.1.1.22	Module contents	110
14.20.1.1.1.23	graphene_elastic.filter_backends.score package	110
14.20.1.1.1.24	Submodules	110
14.20.1.1.1.25	graphene_elastic.filter_backends.score.common module	110
14.20.1.1.1.26	Module contents	111
14.20.1.1.1.27	graphene_elastic.filter_backends.search package	111
14.20.1.1.1.28	Submodules	111
14.20.1.1.1.29	graphene_elastic.filter_backends.search.common module	111
14.20.1.1.1.30	graphene_elastic.filter_backends.search.query_string module	117
14.20.1.1.1.31	graphene_elastic.filter_backends.search.simple_query_string module	118
14.20.1.1.1.32	Module contents	119
14.20.1.1.1.33	graphene_elastic.filter_backends.source package	119
14.20.1.1.1.34	Submodules	119
14.20.1.1.1.35	graphene_elastic.filter_backends.source.common module	119
14.20.1.1.1.36	Module contents	120
14.20.1.1.2	Submodules	120
14.20.1.1.3	graphene_elastic.filter_backends.base module	120
14.20.1.1.4	graphene_elastic.filter_backends.queries module	123
14.20.1.1.5	Module contents	127
14.20.1.2	graphene_elastic.relay package	127
14.20.1.2.1	Submodules	127
14.20.1.2.2	graphene_elastic.relay.connection module	127
14.20.1.2.3	graphene_elastic.relay.connectiontypes module	127
14.20.1.2.4	Module contents	127
14.20.1.3	graphene_elastic.tests package	127
14.20.1.3.1	Submodules	127
14.20.1.3.2	graphene_elastic.tests.base module	127
14.20.1.3.3	graphene_elastic.tests.test_faceted_search_backend module	128
14.20.1.3.4	graphene_elastic.tests.test_filter_backend module	128
14.20.1.3.5	graphene_elastic.tests.test_highlight_backend module	129
14.20.1.3.6	graphene_elastic.tests.test_ordering_backend module	129

14.20.1.3.7	graphene_elastic.tests.test_pagination module	129
14.20.1.3.8	graphene_elastic.tests.test_post_filter_backend module	130
14.20.1.3.9	graphene_elastic.tests.test_query_string_backend module	130
14.20.1.3.10	graphene_elastic.tests.test_score_backend module	130
14.20.1.3.11	graphene_elastic.tests.test_search_backend module	130
14.20.1.3.12	graphene_elastic.tests.test_simple_query_string_backend module	131
14.20.1.3.13	graphene_elastic.tests.test_source_backend module	131
14.20.1.3.14	graphene_elastic.tests.test_versions module	131
14.20.1.3.15	Module contents	132
14.20.1.4	graphene_elastic.types package	132
14.20.1.4.1	Submodules	132
14.20.1.4.2	graphene_elastic.types.elastic_types module	132
14.20.1.4.3	graphene_elastic.types.json_string module	132
14.20.1.4.4	Module contents	133
14.20.2	Submodules	133
14.20.3	graphene_elastic.advanced_types module	133
14.20.4	graphene_elastic.arrayconnection module	133
14.20.5	graphene_elastic.compat module	134
14.20.6	graphene_elastic.constants module	134
14.20.7	graphene_elastic.converter module	134
14.20.8	graphene_elastic.enums module	135
14.20.9	graphene_elastic.fields module	135
14.20.10	graphene_elastic.logging module	136
14.20.11	graphene_elastic.registry module	136
14.20.12	graphene_elastic.settings module	136
14.20.13	graphene_elastic.utils module	136
14.20.14	graphene_elastic.versions module	137
14.20.15	Module contents	137
15	Indices and tables	139
	Python Module Index	141
	Index	143

Elasticsearch (DSL)/ OpenSearch (DSL) integration for Graphene.

PREREQUISITES

- Graphene 2.x. *Support for Graphene 1.x is not intended.*
- Python 3.6, 3.7, 3.8, 3.9 and 3.10. *Support for Python 2 is not intended.*
- Elasticsearch 6.x, 7.x. *Support for Elasticsearch 5.x is not intended.*
- OpenSearch 1.x, 2.x.

MAIN FEATURES AND HIGHLIGHTS

- Implemented `ElasticsearchConnectionField` and `ElasticsearchObjectType` are the core classes to work with graphene.
- Pluggable backends for searching, filtering, ordering, etc. Don't like existing ones? Override, extend or write your own.
- Search backend.
- Filter backend.
- Ordering backend.
- Pagination.
- Highlighting backend.
- Source filter backend.
- Faceted search backend (including global aggregations).
- Post filter backend.
- Score filter backend.
- Query string backend.
- Simple query string backend.

See the *Road-map* for what's yet planned to implemented.

Do you need a similar tool for Django REST Framework? Check [django-elasticsearch-dsl-drf](#).

**CHAPTER
THREE**

DEMO

Check the [live demo app](#) (FastAPI + Graphene 2 + Elasticsearch 7) hosted on Heroku and bonsai.io.

DOCUMENTATION

Documentation is available on [Read the Docs](#).

INSTALLATION

Install latest stable version from PyPI:

```
pip install graphene-elastic
```

Or latest development version from GitHub:

```
pip install https://github.com/barseghyanartur/graphene-elastic/archive/master.zip
```

Note: Starting from version 0.8, the `elasticsearch` and `elasticsearch-dsl` packages are no longer installed by default. You must either install them explicitly in your requirements or install as optional dependencies as follows: `pip install graphene-elastic[elasticsearch]`. Alternatively, you can use `opensearch-py` and `opensearch-dsl`. You would then need to install the `opensearch-py` and `opensearch-dsl` packages explicitly in your requirements or install them as optional dependencies as follows: `pip install graphene-elastic[opensearch]`.

EXAMPLES

Note: In the examples, we use `elasticsearch_dsl` package for schema definition. You can however use `opensearch_dsl` or if you want to achieve portability between Elasticsearch and OpenSearch, use `anysearch` package. Read more [here](#).

6.1 Install requirements

```
pip install -r requirements.txt
```

6.2 Populate sample data

The following command will create indexes for User and Post documents and populate them with sample data:

```
./scripts/populate_elasticsearch_data.sh
```

6.3 Sample document definition

search_index/documents/post.py

See `examples/search_index/documents/post.py` for full example.

```
import datetime
from elasticsearch_dsl import (
    Boolean,
    Date,
    Document,
    InnerDoc,
    Keyword,
    Nested,
    Text,
    Integer,
)

class Comment(InnerDoc):
```

(continues on next page)

```
author = Text(fields={'raw': Keyword()})
content = Text(analyzer='snowball')
created_at = Date()

def age(self):
    return datetime.datetime.now() - self.created_at
```

```
class Post(Document):

    title = Text(
        fields={'raw': Keyword()}
    )
    content = Text()
    created_at = Date()
    published = Boolean()
    category = Text(
        fields={'raw': Keyword()}
    )
    comments = Nested(Comment)
    tags = Text(
        analyzer=html_strip,
        fields={'raw': Keyword(multi=True)},
        multi=True
    )
    num_views = Integer()

class Index:
    name = 'blog_post'
    settings = {
        'number_of_shards': 1,
        'number_of_replicas': 1,
        'blocks': {'read_only_allow_delete': None},
    }
```

6.4 Sample apps

6.4.1 Sample Flask app

Run the sample Flask app:

```
./scripts/run_flask.sh
```

Open Flask graphiql client

```
http://127.0.0.1:8001/graphql
```

6.4.2 Sample Django app

Run the sample Django app:

```
./scripts/run_django.sh runserver
```

Open Django graphiql client

```
http://127.0.0.1:8000/graphql
```

6.4.3 ConnectionField example

ConnectionField is the most flexible and feature rich solution you have. It uses filter backends which you can tie to your needs the way you want in a declarative manner.

Sample schema definition

```
import graphene
from graphene_elastic import (
    ElasticsearchObjectType,
    ElasticsearchConnectionField,
)
from graphene_elastic.filter_backends import (
    FilteringFilterBackend,
    SearchFilterBackend,
    HighlightFilterBackend,
    OrderingFilterBackend,
    DefaultOrderingFilterBackend,
)
from graphene_elastic.constants import (
    LOOKUP_FILTER_PREFIX,
    LOOKUP_FILTER_TERM,
    LOOKUP_FILTER_TERMS,
    LOOKUP_FILTER_WILDCARD,
    LOOKUP_QUERY_EXCLUDE,
    LOOKUP_QUERY_IN,
)

# Object type definition
class Post(ElasticsearchObjectType):

    class Meta(object):
        document = PostDocument
        interfaces = (Node,)
        filter_backends = [
            FilteringFilterBackend,
            SearchFilterBackend,
            HighlightFilterBackend,
            OrderingFilterBackend,
            DefaultOrderingFilterBackend,
        ]

    # For `FilteringFilterBackend` backend
```

(continues on next page)

(continued from previous page)

```
filter_fields = {
    # The dictionary key (in this case `title`) is the name of
    # the corresponding GraphQL query argument. The dictionary
    # value could be simple or complex structure (in this case
    # complex). The `field` key points to the `title.raw`, which
    # is the field name in the Elasticsearch document
    # (`PostDocument`). Since `lookups` key is provided, number
    # of lookups is limited to the given set, while term is the
    # default lookup (as specified in `default_lookup`).
    'title': {
        'field': 'title.raw',
        # Available lookups
        'lookups': [
            LOOKUP_FILTER_TERM,
            LOOKUP_FILTER_TERMS,
            LOOKUP_FILTER_PREFIX,
            LOOKUP_FILTER_WILDCARD,
            LOOKUP_QUERY_IN,
            LOOKUP_QUERY_EXCLUDE,
        ],
        # Default lookup
        'default_lookup': LOOKUP_FILTER_TERM,
    },

    # The dictionary key (in this case `category`) is the name of
    # the corresponding GraphQL query argument. Since no lookups
    # or default_lookup is provided, defaults are used (all lookups
    # available, term is the default lookup). The dictionary value
    # (in this case `category.raw`) is the field name in the
    # Elasticsearch document (`PostDocument`).
    'category': 'category.raw',

    # The dictionary key (in this case `tags`) is the name of
    # the corresponding GraphQL query argument. Since no lookups
    # or default_lookup is provided, defaults are used (all lookups
    # available, term is the default lookup). The dictionary value
    # (in this case `tags.raw`) is the field name in the
    # Elasticsearch document (`PostDocument`).
    'tags': 'tags.raw',

    # The dictionary key (in this case `num_views`) is the name of
    # the corresponding GraphQL query argument. Since no lookups
    # or default_lookup is provided, defaults are used (all lookups
    # available, term is the default lookup). The dictionary value
    # (in this case `num_views`) is the field name in the
    # Elasticsearch document (`PostDocument`).
    'num_views': 'num_views',
}

# For `SearchFilterBackend` backend
search_fields = {
    'title': {'boost': 4},
```

(continues on next page)

(continued from previous page)

```

        'content': {'boost': 2},
        'category': None,
    }

    # For `OrderingFilterBackend` backend
    ordering_fields = {
        # The dictionary key (in this case `tags`) is the name of
        # the corresponding GraphQL query argument. The dictionary
        # value (in this case `tags.raw`) is the field name in the
        # Elasticsearch document (`PostDocument`).
        'title': 'title.raw',

        # The dictionary key (in this case `created_at`) is the name of
        # the corresponding GraphQL query argument. The dictionary
        # value (in this case `created_at`) is the field name in the
        # Elasticsearch document (`PostDocument`).
        'created_at': 'created_at',

        # The dictionary key (in this case `num_views`) is the name of
        # the corresponding GraphQL query argument. The dictionary
        # value (in this case `num_views`) is the field name in the
        # Elasticsearch document (`PostDocument`).
        'num_views': 'num_views',
    }

    # For `DefaultOrderingFilterBackend` backend
    ordering_defaults = (
        '-num_views', # Field name in the Elasticsearch document
        'title.raw', # Field name in the Elasticsearch document
    )

    # For `HighlightFilterBackend` backend
    highlight_fields = {
        'title': {
            'enabled': True,
            'options': {
                'pre_tags': ["<b>"],
                'post_tags': ["</b>"],
            }
        },
        'content': {
            'options': {
                'fragment_size': 50,
                'number_of_fragments': 3
            }
        },
        'category': {},
    }

# Query definition
class Query(graphene.ObjectType):
    all_post_documents = ElasticsearchConnectionField(Post)

```

(continues on next page)

(continued from previous page)

```
# Schema definition
schema = graphene.Schema(query=Query)
```

6.4.3.1 Filter

6.4.3.1.1 Sample queries

Since we didn't specify any lookups on *category*, by default all lookups are available and the default lookup would be term. Note, that in the `{value:"Elastic"}` part, the `value` stands for default lookup, whatever it has been set to.

```
query PostsQuery {
  allPostDocuments(filter:{category:{value:"Elastic"}}) {
    edges {
      node {
        id
        title
        category
        content
        createdAt
        comments
      }
    }
  }
}
```

But, we could use another lookup (in example below - terms). Note, that in the `{terms:["Elastic", "Python"]}` part, the `terms` is the lookup name.

```
query PostsQuery {
  allPostDocuments(
    filter:{category:{terms:["Elastic", "Python"]}}
  ) {
    edges {
      node {
        id
        title
        category
        content
        createdAt
        comments
      }
    }
  }
}
```

Or apply a `gt` (range) query in addition to filtering:

```
{
  allPostDocuments(filter:{
    category:{term:"Python"},
```

(continues on next page)

(continued from previous page)

```
    numViews:{gt:"700"}
  }) {
    edges {
      node {
        category
        title
        comments
        numViews
      }
    }
  }
}
```

6.4.3.1.2 Implemented filter lookups

The following lookups are available:

- contains
- ends_with (or endsWith for camelCase)
- exclude
- exists
- gt
- gte
- in
- is_null (or isNull for camelCase)
- lt
- lte
- prefix
- range
- starts_with (or startsWith for camelCase)
- term
- terms
- wildcard

See [dedicated documentation on filter lookups](#) for more information.

6.4.3.2 Search

Search in all fields:

```
query {
  allPostDocuments(
    search:{query:"Release Box"}
  ) {
    edges {
      node {
        category
        title
        content
      }
    }
  }
}
```

Search in specific fields:

```
query {
  allPostDocuments(
    search:{
      title:{value:"Release", boost:2},
      content:{value:"Box"}
    }
  ) {
    edges {
      node {
        category
        title
        content
      }
    }
  }
}
```

6.4.3.3 Ordering

Possible choices are ASC and DESC.

```
query {
  allPostDocuments(
    filter:{category:{term:"Photography"}},
    ordering:{title:ASC}
  ) {
    edges {
      node {
        category
        title
        content
        numViews
        tags
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
    }  
  }  
}
```

6.4.3.4 Pagination

The `first`, `last`, `before` and `after` arguments are supported. By default number of results is limited to 100.

```
query {  
  allPostDocuments(first:12) {  
    pageInfo {  
      startCursor  
      endCursor  
      hasNextPage  
      hasPreviousPage  
    }  
    edges {  
      cursor  
      node {  
        category  
        title  
        content  
        numViews  
      }  
    }  
  }  
}
```

6.4.3.5 Highlighting

Simply, list the fields you want to highlight. This works only in combination with search.

```
query {  
  allPostDocuments(  
    search:{content:{value:"alice"}, title:{value:"alice"}},  
    highlight:[category, content]  
  ) {  
    edges {  
      node {  
        title  
        content  
        highlight  
      }  
      cursor  
    }  
  }  
}
```


ROAD-MAP

Road-map and development plans.

This package is designed after [django-elasticsearch-dsl-drf](#) and is intended to offer similar functionality.

Lots of features are planned to be released in the upcoming Beta releases:

- Suggester backend.
- Nested backend.
- Geo-spatial backend.
- Filter lookup `geo_bounding_box` (or `geoBoundingBox` for camelCase).
- Filter lookup `geo_distance` (or `geoDistance` for camelCase).
- Filter lookup `geo_polygon` (or `geoPolygon` for camelCase).
- More-like-this backend.

Stay tuned or reach out if you want to help.

Project is covered with tests.

8.1 Testing with Docker

```
make docker-test
```

8.2 Running tests with virtualenv or tox

By default tests are executed against the Elasticsearch 7.x.

Run Elasticsearch 7.x with Docker

```
docker-compose up elasticsearch
```

Install test requirements

```
pip install -r requirements/test.txt
```

To test with all supported Python versions type:

```
tox
```

To test against specific environment, type:

```
tox -e py38-elastic7
```

To test just your working environment type:

```
./runtests.py
```

To run a single test module in your working environment type:

```
./runtests.py src/graphene_elastic/tests/test_filter_backend.py
```

To run a single test class in a given test module in your working environment type:

```
./runtests.py src/graphene_elastic/tests/test_filter_backend.  
↪ py::FilterBackendElasticTestCase
```


DEBUGGING

For development purposes, you could use the flask app (easy to debug). Standard pdb works (`import pdb; pdb.set_trace()`). If `ipdb` does not work well for you, use `ptpdb`.

WRITING DOCUMENTATION

Keep the following hierarchy.

```
=====  
title  
=====  
  
header  
=====  
  
sub-header  
-----  
  
sub-sub-header  
~~~~~  
  
sub-sub-sub-header  
^^^^^^  
  
sub-sub-sub-sub-header  
+++++++  
  
sub-sub-sub-sub-sub-header  
*****
```

**CHAPTER
ELEVEN**

LICENSE

GPL-2.0-only OR LGPL-2.1-or-later

CHAPTER
TWELVE

SUPPORT

For any security issues contact me at the e-mail given in the *Author* section. For overall issues, go to [GitHub](#).

CHAPTER
THIRTEEN

AUTHOR

Artur Barseghyan <artur.barseghyan@gmail.com>

PROJECT DOCUMENTATION

Contents:

Table of Contents

- *graphene-elastic*
 - *Prerequisites*
 - *Main features and highlights*
 - *Demo*
 - *Documentation*
 - *Installation*
 - *Examples*
 - * *Install requirements*
 - * *Populate sample data*
 - * *Sample document definition*
 - * *Sample apps*
 - *Sample Flask app*
 - *Sample Django app*
 - *ConnectionField example*
 - *Filter*
 - *Sample queries*
 - *Implemented filter lookups*
 - *Search*
 - *Ordering*
 - *Pagination*
 - *Highlighting*
 - *Road-map*
 - *Testing*
 - * *Testing with Docker*

- * *Running tests with virtualenv or tox*
- *Debugging*
- *Writing documentation*
- *License*
- *Support*
- *Author*
- *Project documentation*
- *Indices and tables*

14.1 Concepts

In order to explain in details, we need an imaginary app.

14.1.1 Sample document definition

search_index/documents/post.py

See *examples/search_index/documents/post.py* for full example.

```
import datetime
from elasticsearch_dsl import (
    Boolean,
    Date,
    Document,
    InnerDoc,
    Keyword,
    Nested,
    Text,
    Integer,
)

class Comment(InnerDoc):

    author = Text(fields={'raw': Keyword()})
    content = Text(analyzer='snowball')
    created_at = Date()

    def age(self):
        return datetime.datetime.now() - self.created_at

class Post(Document):

    title = Text(
        fields={'raw': Keyword()})
    )
    content = Text()
```

(continues on next page)

(continued from previous page)

```

created_at = Date()
published = Boolean()
category = Text(
    fields={'raw': Keyword()}
)
comments = Nested(Comment)
tags = Text(
    analyzer=html_strip,
    fields={'raw': Keyword(multi=True)},
    multi=True
)
num_views = Integer()

class Index:
    name = 'blog_post'
    settings = {
        'number_of_shards': 1,
        'number_of_replicas': 1,
        'blocks': {'read_only_allow_delete': None},
    }

```

14.1.2 Sample schema definition

ConnectionField is the most flexible and feature rich solution you have. It uses filter backends which you can tie to your needs the way you want in a declarative manner.

schema.py

```

import graphene
from graphene_elastic import (
    ElasticsearchObjectType,
    ElasticsearchConnectionField,
)
from graphene_elastic.filter_backends import (
    FilteringFilterBackend,
    SearchFilterBackend,
    OrderingFilterBackend,
    DefaultOrderingFilterBackend,
)
from graphene_elastic.constants import (
    LOOKUP_FILTER_PREFIX,
    LOOKUP_FILTER_TERM,
    LOOKUP_FILTER_TERMS,
    LOOKUP_FILTER_WILDCARD,
    LOOKUP_QUERY_EXCLUDE,
    LOOKUP_QUERY_IN,
)

# Object type definition
class Post(ElasticsearchObjectType):

```

(continues on next page)

```

class Meta(object):
    document = PostDocument
    interfaces = (Node,)
    filter_backends = [
        FilteringFilterBackend,
        SearchFilterBackend,
        OrderingFilterBackend,
        DefaultOrderingFilterBackend,
    ]

    # For `FilteringFilterBackend` backend
    filter_fields = {
        # The dictionary key (in this case `title`) is the name of
        # the corresponding GraphQL query argument. The dictionary
        # value could be simple or complex structure (in this case
        # complex). The `field` key points to the `title.raw`, which
        # is the field name in the Elasticsearch document
        # (`PostDocument`). Since `lookups` key is provided, number
        # of lookups is limited to the given set, while term is the
        # default lookup (as specified in `default_lookup`).
        'title': {
            'field': 'title.raw', # Field name in the Elastic doc
            # Available lookups
            'lookups': [
                LOOKUP_FILTER_TERM,
                LOOKUP_FILTER_TERMS,
                LOOKUP_FILTER_PREFIX,
                LOOKUP_FILTER_WILDCARD,
                LOOKUP_QUERY_IN,
                LOOKUP_QUERY_EXCLUDE,
            ],
            # Default lookup
            'default_lookup': LOOKUP_FILTER_TERM,
        },

        # The dictionary key (in this case `category`) is the name of
        # the corresponding GraphQL query argument. Since no lookups
        # or default_lookup is provided, defaults are used (all lookups
        # available, term is the default lookup). The dictionary value
        # (in this case `category.raw`) is the field name in the
        # Elasticsearch document (`PostDocument`).
        'category': 'category.raw',

        # The dictionary key (in this case `tags`) is the name of
        # the corresponding GraphQL query argument. Since no lookups
        # or default_lookup is provided, defaults are used (all lookups
        # available, term is the default lookup). The dictionary value
        # (in this case `tags.raw`) is the field name in the
        # Elasticsearch document (`PostDocument`).
        'tags': 'tags.raw',

        # The dictionary key (in this case `num_views`) is the name of

```

(continues on next page)

(continued from previous page)

```

# the corresponding GraphQL query argument. Since no lookups
# or default_lookup is provided, defaults are used (all lookups
# available, term is the default lookup). The dictionary value
# (in this case `num_views`) is the field name in the
# Elasticsearch document (`PostDocument`).
'num_views': 'num_views',
}

# For `SearchFilterBackend` backend
search_fields = {
    'title': {'boost': 4},
    'content': {'boost': 2},
    'category': None,
}

# For `OrderingFilterBackend` backend
ordering_fields = {
    # The dictionary key (in this case `tags`) is the name of
    # the corresponding GraphQL query argument. The dictionary
    # value (in this case `tags.raw`) is the field name in the
    # Elasticsearch document (`PostDocument`).
    'title': 'title.raw',

    # The dictionary key (in this case `created_at`) is the name of
    # the corresponding GraphQL query argument. The dictionary
    # value (in this case `created_at`) is the field name in the
    # Elasticsearch document (`PostDocument`).
    'created_at': 'created_at',

    # The dictionary key (in this case `num_views`) is the name of
    # the corresponding GraphQL query argument. The dictionary
    # value (in this case `num_views`) is the field name in the
    # Elasticsearch document (`PostDocument`).
    'num_views': 'num_views',
}

# For `DefaultOrderingFilterBackend` backend
ordering_defaults = (
    '-num_views', # Field name in the Elasticsearch document
    'title.raw', # Field name in the Elasticsearch document
)

# Query definition
class Query(graphene.ObjectType):
    all_post_documents = ElasticsearchConnectionField(Post)

# Schema definition
schema = graphene.Schema(query=Query)

```

14.1.2.1 filter_backends

The list of filter backends you want to enable on your schema.

The following filter backends are available at the moment:

- FilteringFilterBackend,
- SearchFilterBackend
- OrderingFilterBackend
- DefaultOrderingFilterBackend

graphene-elastic would dynamically transform your definitions into fields and arguments to use for searching, filtering, ordering, etc.

14.1.2.2 filter_fields

Used by FilteringFilterBackend backend.

It's dict with keys representing names of the arguments that would become available to the GraphQL as input for querying. The values of the dict would be responsible for precise configuration of the queries.

Let's review the following example:

```
'title': {
  'field': 'title.raw',
  'lookups': [
    LOOKUP_FILTER_TERM,
    LOOKUP_FILTER_TERMS,
    LOOKUP_FILTER_PREFIX,
    LOOKUP_FILTER_WILDCARD,
    LOOKUP_QUERY_IN,
    LOOKUP_QUERY_EXCLUDE,
  ],
  'default_lookup': LOOKUP_FILTER_TERM,
}
```

field

The field is the corresponding field of the Elasticsearch Document. In the example below it's `title.raw`.

```
class Post(Document):
    title = Text(
        fields={'raw': Keyword()}
    )
```

lookups

In the given example, the available lookups for the `title.raw` would be limited to `term`, `terms`, `prefix`, `wildcard`, `in` and `exclude`. The latter two are functional queries, as you often see such lookups in ORMs (such as Django) while the others are Elasticsearch native lookups.

In our query we would then explicitly specify the lookup name (`term` in the example below):

```

query PostsQuery {
  allPostDocuments(filter:{title:{term:"Elasticsearch 7.1 released!"}}) {
    edges {
      node {
        id
        title
        category
        content
        createdAt
        comments
      }
    }
  }
}

```

default_lookup

But we could also fallback to the `default_lookup` (term in the example below).

Sample query using `default_lookup`:

```

query PostsQuery {
  allPostDocuments(filter:{title:{value:"Elasticsearch 7.1 released!"}}) {
    edges {
      node {
        id
        title
        category
        content
        createdAt
        comments
      }
    }
  }
}

```

In the block `{title:{value:"Elasticsearch 7.1 released!"}}` the `value` would stand for the `default_lookup` value.

14.1.2.3 search_fields

Used by `SearchFilterBackend` backend.

14.1.2.4 ordering_fields

Used by `OrderingFilterBackend` backend.

Similarly to *filter_fields*, keys of the dict represent argument names that would become available to the GraphQL for queries. The value would be the field name of the corresponding Elasticsearch document.

14.1.2.5 ordering_defaults

Used by `DefaultOrderingFilterBackend`.

If no explicit ordering is given (in the GraphQL query), this would be the fallback - the default ordering. It's expected to be a list or a tuple with field names to be used as default ordering. For descending ordering, add - (minus sign) as prefix to the field name.

14.2 Quick start

14.2.1 Clone the repository

```
git clone git@github.com:barseghyanartur/graphene-elastic.git && cd graphene-elastic
```

14.2.2 Start Elasticsearch

```
docker-compose up elasticsearch
```

14.2.3 Install requirements

```
pip install -r requirements.txt
```

14.2.4 Populate dummy data

```
./scripts/populate_elasticsearch_data.sh
```

14.2.5 Run the test server

```
./scripts/run_flask.sh
```

14.2.6 Open the graphql client the browser

```
http://127.0.0.1:8001/graphql
```

14.2.7 Make some experiments

```
{
  allPostDocuments {
    pageInfo {
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
    edges {
      cursor
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.2.8 Run tests

```
./runtests.py
```

14.3 Search

Search in all fields:

```
query {
  allPostDocuments(
    search:{query:"Release Box"}
  ) {
    edges {
      node {
        category
        title
        content
      }
    }
  }
}
```

Search in specific fields:

```
query {
  allPostDocuments(
    search:{
```

(continues on next page)

(continued from previous page)

```
        title:{value:"Release", boost:2},
        content:{value:"Box"}
    }
) {
  edges {
    node {
      category
      title
      content
    }
  }
}
}
```

14.4 Filtering

14.4.1 Filter lookups

The following lookups are available:

- contains
- ends_with (or endsWith for camelCase)
- exclude
- exists
- gt
- gte
- in
- is_null (or isNull for camelCase)
- lt
- lte
- prefix
- range
- starts_with (or startsWith for camelCase)
- term
- terms
- wildcard

14.4.1.1 Filter lookup contains

```

query {
  allPostDocuments(filter:{category:{contains:"tho"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

14.4.1.2 Filter lookup ends_with

Note: endsWith for camelCase.

```

query {
  allPostDocuments(filter:{category:{endsWith:"thon"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

14.4.1.3 Filter lookup exclude

For a single term:

```

query {
  allPostDocuments(filter:{category:{exclude:"Python"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

For multiple terms:

```
query {
  allPostDocuments(filter:{category:{exclude:["Python", "Django"]}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.4.1.4 Filter lookup exists

```
query {
  allPostDocuments(filter:{category:{exists:true}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.4.1.5 Filter lookup gt

Available value types are:

- decimal
- float
- int
- datetime
- date

```
query {
  allPostDocuments(filter:{numViews:{gt:{decimal:"100.05"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

14.4.1.6 Filter lookup gte

Same value types as in ``gt``

```
query {
  allPostDocuments(filter:{numViews:{gte:{decimal:"100.05"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.4.1.7 Filter lookup in

```
query {
  allPostDocuments(filter:{tags:{in:["photography", "models"]}}) {
    edges {
      node {
        category
        title
        content
        numViews
        tags
      }
    }
  }
}
```

14.4.1.8 Filter lookup lt

Same value types as in ``gt``

```
query {
  allPostDocuments(filter:{numViews:{lt:{date:"2019-10-09"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
    }
  }
}
```

14.4.1.9 Filter lookup lte

Same value types as in ``gt``

```
query {
  allPostDocuments(filter:{numViews:{lte:{date:"2009-10-09"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.4.1.10 Filter lookup prefix

```
query {
  allPostDocuments(filter:{category:{prefix:"Pyth"}}) {
    edges {
      node {
        category
        title
        content
        numViews
        comments
      }
    }
  }
}
```

14.4.1.11 Filter lookup range

Same value types as in ``gt``

```
query {
  allPostDocuments(filter:{numViews:{range:{
    lower:{decimal:"100.05"},
    upper:{decimal:"200.05"}
  }}}) {
    edges {
```

(continues on next page)

(continued from previous page)

```
node {
  category
  title
  content
  numViews
}
}
```

14.4.1.12 Filter lookup starts_with

Note: startsWith for camelCase.

Alias for filter lookup ``prefix``.

14.4.1.13 Filter lookup term

```
query {
  allPostDocuments(filter:{category:{term:"Python"}}) {
    edges {
      node {
        category
        title
        content
        numViews
        comments
      }
    }
  }
}
```

14.4.1.14 Filter lookup terms

```
query {
  allPostDocuments(filter:{category:{terms:["Python", "Django"]}}) {
    edges {
      node {
        category
        title
        content
        numViews
        comments
      }
    }
  }
}
```

14.4.1.15 Filter lookup wildcard

```
query {
  allPostDocuments(filter: {category: {wildcard: "*ytho*"}}) {
    edges {
      node {
        category
        title
        content
        numViews
        comments
      }
    }
  }
}
```

14.5 Post-filter backend

Works the same way as `FilteringFilterBackend`, but does not affect aggregations.

14.5.1 Filter lookups

The following lookups are available:

- `contains`
- `ends_with` (or `endsWith` for camelCase)
- `exclude`
- `exists`
- `gt`
- `gte`
- `in`
- `is_null` (or `isNull` for camelCase)
- `lt`
- `lte`
- `prefix`
- `range`
- `starts_with` (or `startsWith` for camelCase)
- `term`
- `terms`
- `wildcard`

14.5.1.1 Filter lookup contains

```

query {
  allPostDocuments(postFilter:{category:{contains:"tho"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

14.5.1.2 Filter lookup ends_with

Note: endsWith for camelCase.

```

query {
  allPostDocuments(postFilter:{category:{endsWith:"thon"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

14.5.1.3 Filter lookup exclude

For a single term:

```

query {
  allPostDocuments(postFilter:{category:{exclude:"Python"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

For multiple terms:

```
query {
  allPostDocuments(postFilter:{category:{exclude:["Python", "Django"]}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.5.1.4 Filter lookup exists

```
query {
  allPostDocuments(postFilter:{category:{exists:true}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.5.1.5 Filter lookup gt

```
query {
  allPostDocuments(postFilter:{numViews:{gt:{decimal:"100"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.5.1.6 Filter lookup gte

```
query {
  allPostDocuments(postFilter:{numViews:{gte:{decimal:"100"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.5.1.7 Filter lookup in

```
query {
  allPostDocuments(postFilter:{tags:{in:["photography", "models"]}}) {
    edges {
      node {
        category
        title
        content
        numViews
        tags
      }
    }
  }
}
```

14.5.1.8 Filter lookup lt

```
query {
  allPostDocuments(postFilter:{numViews:{lt:{decimal:"200"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.5.1.9 Filter lookup lte

```
query {
  allPostDocuments(postFilter:{numViews:{lte:{decimal:"200"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

14.5.1.10 Filter lookup prefix

```
query {
  allPostDocuments(postFilter:{category:{prefix:"Pyth"}}) {
    edges {
      node {
        category
        title
        content
        numViews
        comments
      }
    }
  }
}
```

14.5.1.11 Filter lookup range

```
query {
  allPostDocuments(postFilter:{numViews:{range:{
    lower:{decimal:"100"},
    upper:{decimal:"200"}
  }}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```


14.5.1.12 Filter lookup starts_with

Note: startsWith for camelCase.

Alias for filter lookup ``prefix``.

14.5.1.13 Filter lookup term

```
query {
  allPostDocuments(postFilter:{category:{term:"Python"}}) {
    edges {
      node {
        category
        title
        content
        numViews
        comments
      }
    }
  }
}
```

14.5.1.14 Filter lookup terms

```
query {
  allPostDocuments(postFilter:{category:{terms:["Python", "Django"]}}) {
    edges {
      node {
        category
        title
        content
        numViews
        comments
      }
    }
  }
}
```

14.5.1.15 Filter lookup wildcard

```
query {
  allPostDocuments(postFilter:{category:{wildcard:"*ytho*"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
        comments
      }
    }
  }
}
```

14.6 Ordering

Possible choices are ASC and DESC.

```
query {
  allPostDocuments(
    filter:{category:{term:"Photography"}},
    ordering:{title:ASC}
  ) {
    edges {
      node {
        category
        title
        content
        numViews
        tags
      }
    }
  }
}
```

Multiple values are allowed:

```
query {
  allPostDocuments(
    filter:{category:{term:"Photography"}},
    ordering:{numViews:DESC, createdAt:ASC}
  ) {
    edges {
      node {
        category
        title
        content
        numViews
        tags
      }
    }
  }
}
```

Ordering by score is implemented as well:

```
query {
  allPostDocuments(
```

(continues on next page)

(continued from previous page)

```

        search:{query:"Alice"},
        ordering:{score:DESC}
    ) {
    edges {
        node {
            id
            title
            content
            category
            createdAt
            score
        }
    }
}

```

14.7 Highlight

Sample type definition:

```

from graphene import Node
from graphene_elastic import ElasticsearchObjectType
from graphene_elastic.filter_backends import HighlightFilterBackend

class Post(ElasticsearchObjectType):

    class Meta:

        document = PostDocument
        interfaces = (Node,)
        filter_backends = [
            # ...
            HighlightFilterBackend, # Important
            # ...
        ]

        # ...

        # For `HighlightFilterBackend` backend
        highlight_fields = {
            'title': {
                'enabled': True,
                'options': {
                    'pre_tags': ["<b>"],
                    'post_tags': ["</b>"],
                }
            },
            'content': {
                'options': {
                    'fragment_size': 50,

```

(continues on next page)

(continued from previous page)

```

        'number_of_fragments': 3
      }
    },
    'category': {},
  }
# ...

```

Sample query:

```

query {
  allPostDocuments(
    search:{content:{value:"since"}, title:{value:"decide"}},
    highlight:[category, content]
  ) {
    edges {
      node {
        title
        content
        highlight
      }
      cursor
    }
  }
}

```

Sample response:

```

{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
          "node": {
            "title": "PM decide.",
            "content": "Cut dog young only. Whole natural state Republican year.\nFinancial oil current sea. Mind large similar probably lawyer since. Son control fire.\nremember.",
            "highlight": {
              "title": [
                "PM <b>decide</b>."
              ],
              "content": [
                "Mind large similar probably lawyer <em>since</em>."
              ]
            }
          },
          "cursor": "YXJyYXljb25uZWN0aW9uOjA="
        },
        {
          "node": {
            "title": "Many add.",

```

(continues on next page)

(continued from previous page)

```

        "content": "Read almost consumer perform water. Really protect push send_
↪body wind. Training point since involve public last let new.",
        "highlight": {
            "content": [
                "Training point <em>since</em> involve public last let new."
            ]
        },
        "cursor": "YXJyYXljb25uZWN0aW9uOjE="
    }
}
}
}

```

14.8 Source

Source query is meant to lighten the Elasticsearch load by reducing amount of data sent around. Although GraphQL seems to have solved the issue between frontend and backend, Elasticsearch would still send all the data to the backend. That's where we might use the source backend.

Sample type definition:

```

from graphene import Node
from graphene_elastic import ElasticsearchObjectType
from graphene_elastic.filter_backends import SourceFilterBackend

class Post(ElasticsearchObjectType):

    class Meta:

        document = PostDocument
        interfaces = (Node,)
        filter_backends = [
            # ...
            SourceFilterBackend, # Important
            # ...
        ]

```

Sample query:

```

query {
  allPostDocuments(
    search:{content:{value:"alice"}, title:{value:"alice"}},
    source:[title, id]
  ) {
    edges {
      node {
        id
        title
        content
        category
      }
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

        comments
      }
      cursor
    }
  }
}

```

Sample response:

As you could see, although we do ask for more fields in the node `{...}` part, the requested fields are empty. We only get data in the fields we have specified in source (they are title and id).

```

{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
          "node": {
            "id": "UG9zdDpvX0huUlcwQlhFYXJjd2RMc0w2aQ==",
            "title": "only Alice miss",
            "content": null,
            "category": null,
            "comments": []
          },
          "cursor": "YXJyYXljb25uZWN0aW9uOjA="
        },
        {
          "node": {
            "id": "UG9zdDpvZkhuUlcwQlhFYXJjd2RMc0w1Nw==",
            "title": "prevent Alice citizen",
            "content": null,
            "category": null,
            "comments": []
          },
          "cursor": "YXJyYXljb25uZWN0aW9uOjE="
        }
      ]
    }
  }
}

```

14.9 Score

Score is `relevance` in Elasticsearch.

Sample type definition:

```

from graphene import Node
from graphene_elastic import ElasticsearchObjectType
from graphene_elastic.filter_backends import ScoreFilterBackend, OrderingFilterBackend

```

(continues on next page)

(continued from previous page)

```

class Post(ElasticsearchObjectType):

    class Meta:

        document = PostDocument
        interfaces = (Node,)
        filter_backends = [
            # ...
            ScoreFilterBackend, # Important
            OrderingFilterBackend, # Important
            # ...
        ]

        # For `OrderingFilterBackend` backend
        ordering_fields = {
            # Score
            'score': '_score',
        }

```

Sample query:

Note, that we want to order by relevance (most relevant on top).

```

query {
  allPostDocuments(
    search:{query:"Alice"},
    ordering:{score:DESC}
  ) {
    edges {
      node {
        id
        title
        content
        category
        createdAt
        score
      }
    }
  }
}

```

Sample response:

As you could see, score is calculated.

```

{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
          "node": {
            "id": "UG9zdDpMNnBiV1hNQjhYRzdJclZ2X20waA==",

```

(continues on next page)

(continued from previous page)

```

        "title": "Budget.",
        "category": "Elastic",
        "content": "Bed television public teacher behind human up.\nMind anyone.\n
↳ politics ball cost wife try adult. College work for.\nPlay five ten not sort energy.\n
↳ Common word behind spring. All behind voice policy.",
        "createdAt": "1973-03-12T00:00:00",
        "score": 20.420774
    }
},
...
]
}
}
}

```

14.10 Faceted search

Sample type definition (note the usage of `FacetedSearchFilterBackend` and `faceted_search_fields`).

```

from elasticsearch_dsl import DateHistogramFacet, RangeFacet
from graphene import Node
from graphene_elastic import ElasticsearchObjectType
from graphene_elastic.filter_backends import (
    FacetedSearchFilterBackend,
    # ...
)

from search_index.documents import Post as PostDocument

class Post(ElasticsearchObjectType):

    class Meta:

        document = PostDocument
        interfaces = (Node,)
        filter_backends = [
            # ...
            FacetedSearchFilterBackend,
            # ...
        ]

        # For `FacetedSearchFilterBackend` backend
        faceted_search_fields = {
            'category': 'category.raw',
            'category_global': {
                'field': 'category.raw',
                # Setting `global` to True, makes the facet global
                'global': True,
            },
            'tags': {

```

(continues on next page)

(continued from previous page)

```

        'field': 'tags.raw',
        'enabled': True, # Will appear in the list by default
        'global': True,
    },
    'created_at': {
        'field': 'created_at',
        'facet': DateHistogramFacet,
        'options': {
            'interval': 'year',
        }
    },
    'num_views_count': {
        'field': 'num_views',
        'facet': RangeFacet,
        'options': {
            'ranges': [
                (<10", (None, 10)),
                ("11-20", (11, 20)),
                ("20-50", (20, 50)),
                (">50", (50, None)),
            ]
        }
    },
},
}

```

Sample GraphQL query:

```

query {
  allPostDocuments(
    search:{title:{value:"alice"}}
    facets:[category]
  ) {
    facets
    edges {
      node {
        id
        title
        highlight
      }
    }
  }
}

```

Sample response:

```

{
  "data": {
    "allPostDocuments": {
      "facets": {
        "tags": {
          "doc_count": 9,

```

(continues on next page)

(continued from previous page)

```
"aggs": {
  "doc_count_error_upper_bound": 0,
  "sum_other_doc_count": 0,
  "buckets": [
    {
      "key": "photography",
      "doc_count": 7
    },
    {
      "key": "art",
      "doc_count": 6
    },
    {
      "key": "article",
      "doc_count": 5
    },
    {
      "key": "black and white",
      "doc_count": 5
    },
    {
      "key": "package",
      "doc_count": 5
    },
    {
      "key": "models",
      "doc_count": 4
    },
    {
      "key": "programming",
      "doc_count": 4
    }
  ]
},
"category": {
  "doc_count": 9,
  "aggs": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 0,
    "buckets": [
      {
        "key": "Python",
        "doc_count": 3
      },
      {
        "key": "Model Photography",
        "doc_count": 2
      },
      {
        "key": "Django",
        "doc_count": 1
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

    },
    {
      "key": "Elastic",
      "doc_count": 1
    },
    {
      "key": "Machine Learning",
      "doc_count": 1
    },
    {
      "key": "MongoDB",
      "doc_count": 1
    }
  ]
}
},
"edges": [
  {
    "node": {
      "id": "UG9zdDpBVWNwVm0wQklwZ2dXbVlJTnd0VA==",
      "title": "better Alice must",
      "highlight": {
        "title": [
          "better <b>Alice</b> must"
        ]
      }
    }
  },
  ...
]
}
}

```

Note, that category appeared in the result because we explicitly requested so (in `facets: [category]`) and the tags are there because they have been enabled by default (in `faceted_search_fields`).

14.11 Query string backend

Implementation of Query string query.

Sample type definition

```

from graphene import Node
from graphene_elastic import ElasticsearchObjectType
from graphene_elastic.filter_backends import QueryStringBackend

class Post(ElasticsearchObjectType):

    class Meta:

```

(continues on next page)

(continued from previous page)

```

document = PostDocument
interfaces = (Node,)
filter_backends = [
    # ...
    QueryStringBackend, # Important
    # ...
]

query_string_options = {
    "fields": ["title^2", "content", "category"],
    "boost": 2,
}

```

Sample query

```

query PostsQuery {
  allPostDocuments(
    queryString:"(White Rabbit) AND Alice"
  ) {
    edges {
      node {
        id
        title
        category
        content
        createdAt
        comments
      }
    }
  }
}

```

Sample response

```

{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
          "node": {
            "id": "UG9zdDppLVozZDNZQmNDeUtjYnh5WTU5Vg==",
            "title": "Alice",
            "category": "MongoDB",
            "content": "Personal green well method day report. White Rabbit is dead. ↵
↪Take stuff newspaper soldier up.",
            "createdAt": "1994-01-01T00:00:00",
            "comments": [
              {
                "author": "Matthew Jones",
                "content": "Despite consumer safe since range opportunity.",
                "created_at": "1970-05-05T00:00:00"
              }
            ]
          }
        }
      ]
    }
  }
}

```

(continues on next page)

(continued from previous page)

```
    category
    content
    createdAt
    comments
  }
}
```

Sample response

```
{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
          "node": {
            "id": "UG9zdDppLVozZDNZQmNDeUtjYnh5WTU5Vg==",
            "title": "Alice",
            "category": "MongoDB",
            "content": "Personal green well method day report. White Rabbit is dead.↵
↵Take stuff newspaper soldier up.",
            "createdAt": "1994-01-01T00:00:00",
            "comments": [
              {
                "author": "Matthew Jones",
                "content": "Despite consumer safe since range opportunity.",
                "created_at": "1970-05-05T00:00:00"
              },
              {
                "author": "Larry Brown",
                "content": "Environment drug artist. Pattern source sound hope trip.",
                "created_at": "2005-07-24T00:00:00"
              }
            ]
          }
        ]
      ]
    }
  }
}
```

14.13 Pagination

14.13.1 Limits

By default, max number of fetched items is limited to 100. It's configurable. Set the `RELAY_CONNECTION_MAX_LIMIT` setting to the desired value.

14.13.2 Enforce first or last

You could force users to provide `first` or `last`. Set `RELAY_CONNECTION_ENFORCE_FIRST_OR_LAST` to `True` for that.

14.13.3 User controlled pagination

The following (standard) arguments are available:

- `first`
- `last`
- `before`
- `after`

Sample query to return all results (limited by `RELAY_CONNECTION_MAX_LIMIT` setting only):

```
{
  allPostDocuments {
    pageInfo {
      startCursor
      endCursor
      hasNextPage
      hasPreviousPage
    }
    edges {
      cursor
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

Sample query to return first 12 results:

```
{
  allPostDocuments(first:12) {
    pageInfo {
      startCursor
      endCursor
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

    hasNextPage
    hasPreviousPage
  }
  edges {
    cursor
    node {
      category
      title
      content
      numViews
    }
  }
}
}

```

Sample query to return first 12 results after the given offset:

14.14 Custom filter backends

Filter backends can:

- Add new graphene input types to the (query) schema.
- Allow you to request additional information by adding new graphene fields to the schema.
- Alter current queryset.
- Alter slice, add additional information next to pageInfo and edges, such as facets, for example.

Let's learn by example on the SourceFilterBackend which allows us to apply source query to the current search queryset.

```

import enum
import graphene

from graphen_elastic.filter_backends.base import BaseBackend
from graphen_elastic.constants import DYNAMIC_CLASS_NAME_PREFIX

class SourceFilterBackend(BaseBackend):
    """Source filter backend."""

    prefix = 'source' # Name of the GraphQL query filter
    has_query_fields = True # Indicates whether backend has own filtering fields

    # The ``source_fields`` is the config options that we set on the
    # ``Post`` object type. In this case - absolutely optional.
    @property
    def source_fields(self):
        """Source filter fields."""
        return getattr(
            self.connection_field.type._meta.node._meta,
            'filter_backend_options',
            {}
        )

```

(continues on next page)

(continued from previous page)

```

    ).get('source_fields', {})

# This is where we dynamically create GraphQL filter fields for this
# backend.
def get_backend_filtering_fields(self, items, is_filterable_func, get_type_func):
    """Construct backend fields.

    :param items:
    :param is_filterable_func:
    :param get_type_func:
    :return:
    """
    _keys = list(
        self.connection_field.type._meta.node._meta.fields.keys()
    )
    _keys.remove('_id')
    params = zip(_keys, _keys)
    return {
        self.prefix: graphene.Argument(
            graphene.List(
                graphene.Enum.from_enum(
                    enum.Enum(
                        "{}{}BackendEnum".format(
                            DYNAMIC_CLASS_NAME_PREFIX,
                            self.prefix.title(),
                            self.connection_field.type.__name__
                        ),
                        params
                    )
                )
            )
        )
    }

# Some data normalisation.
def prepare_source_fields(self):
    """Prepare source fields.

    Possible structures:

        source_fields = ["title"]

    Or:

        search_fields = ["title", "author.*"]

    Or:

        source = {
            "includes": ["title", "author.*"],
            "excludes": [ "*.description" ]
        }

```

(continues on next page)

(continued from previous page)

```

        :return: Filtering options.
        :rtype: dict
        """
        source_args = dict(self.args).get(self.prefix, [])

        source_fields = dict(self.source_fields)

        if source_args:
            return source_args
        return source_fields

    # This is where the queryset is being altered.
    def filter(self, queryset):
        """Filter.

        :param queryset:
        :return:
        """
        source_fields = self.prepare_source_fields()

        if source_fields:
            queryset = queryset.source(source_fields)

        return queryset

```

14.15 Settings

Defaults are:

```

DEFAULTS = {
    "SCHEMA": None,
    "SCHEMA_OUTPUT": "schema.json",
    "SCHEMA_INDENT": 2,
    # "MIDDLEWARE": (),
    # Set to True if the connection fields must have
    # either the first or last argument
    "RELAY_CONNECTION_ENFORCE_FIRST_OR_LAST": False,
    # Max items returned in ConnectionFields / FilterConnectionFields
    "RELAY_CONNECTION_MAX_LIMIT": 100,
    "LOGGING_LEVEL": logging.ERROR,
}

```

See the example below to get a grasp on how to override:

```

import json
import logging
import os

DEFAULTS = {

```

(continues on next page)

(continued from previous page)

```

"SCHEMA": None,
"SCHEMA_OUTPUT": "schema.json",
"SCHEMA_INDENT": 2,
# Set to True if the connection fields must have
# either the first or last argument
"RELAY_CONNECTION_ENFORCE_FIRST_OR_LAST": False,
# Max items returned in ConnectionFields / FilterConnectionFields
"RELAY_CONNECTION_MAX_LIMIT": 100,
"LOGGING_LEVEL": logging.DEBUG,
}

os.environ.setdefault(
    "GRAPHENE_ELASTIC",
    json.dumps(DEFAULTS)
)

```

14.16 Running Elasticsearch

For development and testing purposes, it's often handy to be able to quickly switch between different Elasticsearch versions. You could make use of the following boxes/containers for development and testing.

For all containers/boxes mentioned below, no authentication is required (for Elasticsearch).

14.16.1 Docker

14.16.1.1 Project default

```
docker-compose up elasticsearch
```

14.16.1.2 Run specific version

14.16.1.2.1 6.x

6.3.2

```

docker pull docker.elastic.co/elasticsearch/elasticsearch:6.3.2
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.enabled=false"
↳ docker.elastic.co/elasticsearch/elasticsearch:6.3.2

```

6.4.0

```

docker pull docker.elastic.co/elasticsearch/elasticsearch:6.4.0
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.enabled=false"
↳ docker.elastic.co/elasticsearch/elasticsearch:6.4.0

```

14.16.1.2.2 7.x

7.1.1

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:7.1.1
docker run -p 9200:9200 -e "discovery.type=single-node" -e "xpack.security.enabled=false"
↪" docker.elastic.co/elasticsearch/elasticsearch:7.1.1
```

14.17 FAQ

You will find a lot of useful information in the [documentation](#).

Additionally, all raised issues that were questions have been marked as *question*, so you could take a look at the [closed \(question\) issues](#).

14.17.1 Questions and answers

Question

What about authentication and/or filtering the results based on authenticated user.

Answer

Check [this doc](#)

14.18 Debugging

14.18.1 Logging queries to console

The `LOGGING_LEVEL` key represents the logging level (defaults to `logging.ERROR`). Override if needed.

Typical development setup would be:

```
import json
import logging
import os

DEFAULTS = {
    # ...
    "LOGGING_LEVEL": logging.DEBUG,
    # ...
}

os.environ.setdefault(
    "GRAPHENE_ELASTIC",
    json.dumps(DEFAULTS)
)
```

14.19 Release history and notes

Sequence based identifiers are used for versioning (schema follows below):

```
major.minor[.revision]
```

- It's always safe to upgrade within the same minor version (for example, from 0.3 to 0.3.4).
- Minor version changes might be backwards incompatible. Read the release notes carefully before upgrading (for example, when upgrading from 0.3.4 to 0.4).
- All backwards incompatible changes are mentioned in this document.

14.19.1 0.8.1

Unreleased

- Drop support for Python 3.6. No specific code changes. However, since it's no longer possible to pull Python 3.6 directly from GitHub CI, to simplify things we simply drop it.

14.19.2 0.8

2022-07-31

Note: The `elasticsearch` and `elasticsearch-dsl` packages are no longer installed by default. You must either install them explicitly in your requirements or install alongside this package as optional dependencies as follows: `pip install graphene-elastic[elasticsearch]`. Alternatively, you can use `opensearch-py` and `opensearch-dsl`. You would then need to install the `opensearch-py` and `opensearch-dsl` packages explicitly in your requirements or install alongside this package as optional dependencies as follows: `pip install graphene-elastic[opensearch]`.

- Added support for OpenSearch (1.x and 2.x).
- Belated migration to GitHub Actions.

14.19.3 0.7

2021-03-08

Note: First beta release.

Note: Release dedicated to defenders of Armenia and Artsakh (Nagorno Karabakh) and all the victims of Turkish and Azerbaijani aggression.

- `ObjectField/NestedField` to `ObjectType` and added nested filter to `FilteringFilterBackend`.

14.19.4 0.6.6

2020-12-26

- Enable middleware.
- Tested against Python 3.9 and 3.10.

14.19.5 0.6.5

2020-12-19

Note: Release dedicated to defenders of Armenia and Artsakh (Nagorno Karabakh) and all the victims of Turkish and Azerbaijani aggression.

- Added *QueryStringBackend*.

14.19.6 0.6.4

2020-12-12

Note: Release dedicated to defenders of Armenia and Artsakh (Nagorno Karabakh) and all the victims of Turkish and Azerbaijani aggression.

- Added *SimpleQueryStringBackend*.

14.19.7 0.6.3

2020-07-19

Note: Release dedicated to defenders of Armenia and Artsakh (Nagorno Karabakh) and all the victims of Turkish and Azerbaijani aggression.

- Added *ScoreFilterBackend* and ordering by score.

14.19.8 0.6.2

2020-07-07

- Renamed `JSONString` to `ElasticJSONString` and changed references in related files. The reason for this change is to support `grapehene_federation build_schema` which eventually helps build federated schemas

14.19.9 0.6.1

2020-02-13

- Tested against Python 3.8.
- Tested against Elasticsearch 6.x and 7.x on Travis.
- Replace some custom code parts (casing related) with *stringcase* package functionality.

14.19.10 0.6

2019-10-11

Note: Release dedicated to John Lennon. Happy birthday, dear John!

Note: This release introduces minor backwards incompatibility for `range`, `gt`, `gte`, `lt` and `lte` filters. You should update your code.

- The `range`, `gt`, `gte`, `lt` and `lte` filters are now complex input types. This makes it possible to use the following types in comparison: `decimal.Decimal`, `float`, `int`, `datetime.datetime` and `datetime.date`.

Sample new GraphQL query:

```

query {
  allPostDocuments(postFilter:{numViews:{gt:{decimal:"100.00"}}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

Sample old GraphQL query:

```

query {
  allPostDocuments(postFilter:{numViews:{gt:"100.00"}}) {
    edges {
      node {
        category
        title
        content
        numViews
      }
    }
  }
}

```

14.19.11 0.5

2019-09-29

- PostFilter backend.
- Documentation improvements.

14.19.12 0.4

2019-09-23

- Added faceted search backend (with global aggregations support).
- Some refactoring which makes possible for the backends to alter the connection. A lot of minor changes. If you have written custom filter backend, you most likely need to modify some parts.

14.19.13 0.3

2019-09-20

- Minor refactoring allowing third-party independent backends do a lot more without touching the core.
- Source filter backend.
- More tests.

14.19.14 0.2

2019-09-18

- Highlight filter backend.

14.19.15 0.1

2019-09-08

- Documentation fixes.
- Speed up tests.
- Clean up requirements.

14.19.16 0.0.13

2019-09-07

- Documentation improvements and fixes.
- Clean up.

14.19.17 0.0.12

2019-09-06

Note: In memory of Erik Slim. RIP.

- More tests.

14.19.18 0.0.11

2019-09-05

- Fixes in search backend.

14.19.19 0.0.10

2019-09-04

- Fixes.
- Clean up.

14.19.20 0.0.9

2019-09-03

- Added pagination.
- Documentation improvements.

14.19.21 0.0.8

2019-09-02

- Tested default ordering backend.
- Documentation improvements.

14.19.22 0.0.7

2019-09-01

- Ordering backend.
- Added more filter lookups.
- Minor fixes in existing filter lookups.
- Improved test coverage for the filtering backend.
- Documentation improvements.

14.19.23 0.0.6

2019-08-30

- Added more filter lookups.
- Fixes in filtering backend.
- Improved test coverage for the filtering backend.
- Documentation improvements.

14.19.24 0.0.5

2019-08-30

- Implemented custom lookups in favour of a single lookup attribute.
- Updated tests.

14.19.25 0.0.4

2019-08-28

- Fixed travis config (moved to elasticsearch 6.x on travis, since 7.x was causing problems).
- Fixes in setup.py.

14.19.26 0.0.3

2019-08-26

- Documentation fixes.
- Add test suite and initial tests for filter backend and search backend.

14.19.27 0.0.2

2019-08-25

- Added dynamic lookup generation for the filter backend.
- Working lookup param argument handling on the schema (filter backend).

14.19.28 0.0.1

2019-08-24

- Initial alpha release.

14.20 graphene_elastic package

14.20.1 Subpackages

14.20.1.1 graphene_elastic.filter_backends package

14.20.1.1.1 Subpackages

14.20.1.1.1.1 graphene_elastic.filter_backends.faceted_search package

14.20.1.1.1.2 Submodules

14.20.1.1.1.3 graphene_elastic.filter_backends.faceted_search.common module

class graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend(*connection_field*, *args=None*)

Bases: *BaseBackend*

Faceted search filter backend.

aggregate(*queryset*, *agg_field_name_getter*=<function default_agg_field_name_getter>, *agg_bucket_name_getter*=<function default_agg_bucket_name_getter>)

Aggregate.

Parameters

- **queryset** –
- **agg_field_name_getter** – callable.
- **agg_bucket_name_getter** –

Returns

alter_connection(*connection*, *slice*)

Alter connection object.

You can add various properties here, returning the altered object. Typical use-case would be adding facets to the connection.

Parameters

- **connection** –
- **slice** –

Returns

construct_facets()

Construct facets.

Turns the following structure:

```
>>> {
>>>     'publisher': {
>>>         'field': 'publisher.raw',
>>>         'facet': TermsFacet,
```

(continues on next page)

(continued from previous page)

```

>>>     'enabled': False,
>>>     }
>>>     'date_published': {
>>>         'field': 'date_published',
>>>         'facet': DateHistogramFacet,
>>>         'options': {
>>>             'interval': 'month',
>>>         },
>>>         'enabled': True,
>>>     },
>>> }

```

Into the following structure:

```

>>> {
>>>     'publisher': TermsFacet(field='publisher.raw'),
>>>     'publishing_frequency': DateHistogramFacet(
>>>         field='date_published.raw',
>>>         interval='month'
>>>     ),
>>> }

```

property `faceted_search_fields`

Faceted search filter fields.

`field_belongs_to`(*field_name*)

Check if given filter field belongs to the backend.

Parameters

field_name –

Returns

`filter`(*queryset*)

Filter the queryset.

Parameters

queryset (*elasticsearch_dsl.search.Search*) – Base queryset.

Returns

Updated queryset.

Return type

`elasticsearch_dsl.search.Search`

`get_backend_query_fields`(*items, is_filterable_func, get_type_func*)

Construct backend filtering fields.

Parameters

- **items** –
- **is_filterable_func** –
- **get_type_func** –

Returns

get_faceted_search_query_params()

Get highlight query params.

Returns

List of search query params.

Return type

list

has_connection_fields = True

has_query_fields = True

prefix = 'facets'

prepare_faceted_search_fields()

Prepare faceted search fields.

Prepares the following structure:

```

>>> {
>>>   'publisher': {
>>>     'field': 'publisher.raw',
>>>     'facet': TermsFacet,
>>>     'enabled': False,
>>>   }
>>>   'date_published': {
>>>     'field': 'date_published.raw',
>>>     'facet': DateHistogramFacet,
>>>     'options': {
>>>       'interval': 'month',
>>>     },
>>>     'enabled': True,
>>>   },
>>> }

```

Returns

Faceted search fields options.

Return type

dict

14.20.1.1.1.4 Module contents**14.20.1.1.1.5 graphene_elastic.filter_backends.filtering package****14.20.1.1.1.6 Submodules****14.20.1.1.1.7 graphene_elastic.filter_backends.filtering.common module**

class graphene_elastic.filter_backends.filtering.common.**FilteringFilterBackend**(*connection_field*,
args=None)

Bases: *BaseBackend*, *FilteringFilterMixin*

Filtering filter backend.

field_belongs_to(*field_name*)

Check if given filter field belongs to the backend.

Parameters

field_name –

Returns

filter(*queryset*)

Filter.

property filter_args_mapping

property filter_fields

Filtering filter fields.

get_backend_query_fields(*items, is_filterable_func, get_type_func*)

Fail proof override.

Parameters

- **items** –
- **is_filterable_func** –
- **get_type_func** –

Returns

get_field_lookup_param(*field_name*)

Get field lookup param.

Parameters

field_name –

Returns

get_field_options(*field_name, filter_fields=None*)

Nodedocument

: 1. author 2. comments.author 3. comments.author.name

get_field_type(*field_name, field_value, base_field_type*)

Get field type.

Returns

get_filter_query_params()

Get query params to be filtered on.

We can either specify it like this:

```
query_params = {
    'category': {
        'value': 'Elastic',
    }
}
```

Or using specific lookup:

```
query_params = {
```

```

    'category': {
      'term': 'Elastic', 'range': {
        'lower': Decimal('3.0')
      }
    }
  }
}

```

Note, that *value* would only work on simple types (string, integer, decimal). For complex types you would have to use complex param anyway. Therefore, it should be forbidden to set *default_lookup* to a complex field type.

Sample values:

```

query_params = {
  'category': {
    'value': 'Elastic',
  }, 'comments': {
    'author': {
      'name': {
        'value': 'Elastic'
      }
    }
  }
}

filter_fields = {
  'category': {
    'field': 'category.raw', 'type': 'normal', 'default_lookup': 'term', 'lookups': (
      'term', 'terms', 'range', 'exists', 'prefix', 'wildcard', 'contains', 'in', 'gt', 'gte', 'lt',
      'lte', 'starts_with', 'ends_with', 'is_null', 'exclude'
    )
  }, 'comments': {
    'field': 'comments', 'type': 'nested', 'properties': {
      'author': {
        'type': 'object', 'path': 'comments.author', 'properties': {
          'name': {
            'type': 'normal', 'field': 'author.name', 'default_lookups': 'term',
            'lookups': (
              ...
            )
          }
        }
      }
    }
  }
}

```

```

        }
    }
}
field_name = 'category' {
    'inventory_type': {'value': '1'}, 'spu': {
        'supplier_entity_id': {
            'contains': 'Elastic'
        }, 'brand': {
            'code': {
                'term': 'Elastic'
            }
        }
    }
}
}

```

get_nested_field_type(*field_name*, *field_value*, *base_field_type*, *field_options*)

has_query_fields = True

prefix = 'filter'

prepare_filter_fields()

Prepare filter fields

Assume that we have a document like this.

```

"""python class Comment(InnerDoc):
    author = Text(fields={'raw': Keyword()}) content = Text(analyzer='snowball') created_at =
    Date()
    def age(self):
        return datetime.now() - self.created_at

class Post(Document):
    title = Text() title_suggest = Completion() created_at = Date() published = Boolean() category =
    Text(
        analyzer=html_strip, fields={'raw': Keyword()})
    )
    comments = Nested(Comment)
"""

```

Possible structures:

```

filter_fields = {
    'title': {
        'type': 'normal|object|nested', 'field': 'title' # custom field name 'lookups': [
            ... # custom lookup list
        ], 'default_lookup': ... # custom default lookup
    }, 'created_at': {
        'type': 'normal', 'field': 'created_at', 'lookups': [

```



```

        LOOKUP_FILTER_RANGE # only range
    ]
    }, 'published': LOOKUP_FILTER_TERM # treated as default lookup ...
}

```

We shall finally have:

```

filter_fields = {
    'title': {
        'type': 'normal', 'field': 'title.raw', 'lookups': [
            ...
        ], 'default_lookup': ...
    }, ... # any else fields indexed of this document 'comments': {
        'type': 'nested', 'properties': {
            'author': {
                'type': 'normal', 'field': 'comments.author', ...
            }, 'content': {
                'type': 'normal', 'field': 'comments.content', ...
            }
        }
    }
}

```

prepare_query_params()

Prepare query params.

Returns

14.20.1.1.1.8 graphene_elastic.filter_backends.filtering.mixins module

class graphene_elastic.filter_backends.filtering.mixins.**FilteringFilterMixin**

Bases: object

Filtering filter mixin.

apply_filter: Callable

classmethod **apply_filter_prefix**(*queryset, options, value*)

Apply *prefix* filter.

Syntax:

TODO

Example:

```

query {
    allPostDocuments(filter:{category:{prefix:"Pyth"}}) {
        edges {
            node {
                category title content numViews comments
            }
        }
    }
}

```

```

        }
    }
}

```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod `apply_filter_range(queryset, options, value)`

Apply *range* filter.

Syntax:

TODO

Example:

```

{
    allPostDocuments(filter:{numViews:{range:{
        lower:{decimal:"100"}, upper:{decimal:"200"}
    }}}) {
        edges {
            node {
                category title content numViews
            }
        }
    }
}

```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod `apply_filter_term(queryset, options, value)`

Apply *term* filter.

Syntax:

TODO

Example:

```

query {
  allPostDocuments(filter:{category:{term:"Python"}}) {
    edges {
      node {
        category title content numViews comments
      }
    }
  }
}

```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod **apply_filter_terms**(*queryset, options, value*)

Apply *terms* filter.

Syntax:

TODO

Note, that number of values is not limited.

Example:

```

query {
  allPostDocuments(filter:{category:{
    terms:["Python", "Django"]
  }}) {
    edges {
      node {
        category title content numViews comments
      }
    }
  }
}

```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*mixed: either str or iterable (list, tuple).*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

apply_query: Callable**classmethod apply_query_contains**(*queryset, options, value*)Apply *contains* filter.

Syntax:

TODO

Example:

```
query {
    allPostDocuments(filter:{category:{contains:"tho"}}) {
        edges {
            node {
                category title content numViews
            }
        }
    }
}
```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod apply_query_endswith(*queryset, options, value*)Apply *endswith* filter.

Syntax:

TODO

Example:

```
query {
    allPostDocuments(filter:{category:{endsWith:"thon"}}) {
        edges {
            node {
                category title content numViews
            }
        }
    }
}
```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.

- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

`elasticsearch_dsl.search.Search`

classmethod `apply_query_exclude`(*queryset, options, value*)

Apply *exclude* functional query.

Syntax:

TODO

Note, that number of values is not limited.

Example:

```
query {
  allPostDocuments(filter:{category:{exclude:"Python"}}) {
    edges {
      node {
        category title content numViews
      }
    }
  }
}
```

Or exclude multiple terms at once:

```
query {
  allPostDocuments(filter:{category:{exclude:["Ruby", "Java"]}}) {
    edges {
      node {
        category title content numViews
      }
    }
  }
}
```

Parameters

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

`elasticsearch_dsl.search.Search`

classmethod `apply_query_exists(queryset, options, value)`

Apply *exists* filter.

Syntax:

TODO

Example:

```
{
  allPostDocuments(filter:{category:{exists:true}}) {
    edges {
      node {
        category title content numViews
      }
    }
  }
}
```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod `apply_query_gt(queryset, options, value)`

Apply *gt* functional query.

Syntax:

TODO

Example:

```
query {
  allPostDocuments(filter:{numViews:{
    gt:{decimal:"100"}
  }}) { edges {
    node {
      category title content numViews
    }
  }
}
```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.

- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

`elasticsearch_dsl.search.Search`

classmethod `apply_query_gte`(*queryset, options, value*)

Apply *gte* functional query.

Syntax:

TODO

Example:

```
query {
  allPostDocuments(filter:{numViews:{
    gte:{decimal:"100"}
  }}) { edges {
    node {
      category title content numViews
    }
  }
}
```

Parameters

- **queryset** (`elasticsearch_dsl.search.Search`) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

`elasticsearch_dsl.search.Search`

classmethod `apply_query_in`(*queryset, options, value*)

Apply *in* functional query.

Syntax:

TODO

Note, that number of values is not limited.

Example:

```
query {
  allPostDocuments(postFilter:{tags:{
    in:["photography", "models"]
  }}) { edges {
    node {
      category title content numViews tags
    }
  }
}
```

```
    }
  }
```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod `apply_query_isnull(queryset, options, value)`

Apply *isnull* functional query.

Syntax:

TODO

Example:

```
query {
  allPostDocuments(filter:{category:{isNull:true}}) {
    edges {
      node {
        category title content numViews comments
      }
    }
  }
}
```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod `apply_query_lt(queryset, options, value)`

Apply *lt* functional query.

Syntax:

TODO

Example:

```
query {
  allPostDocuments(filter:{numViews:{
    lt:{decimal:"200"}
  }}) { edges {
```



```

        node {
            category title content numViews
        }
    }
}

```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod apply_query_lte(*queryset, options, value*)

Apply *lte* functional query.

Syntax:

TODO

Example:

```

query {
    allPostDocuments(filter:{numViews:{
        lte:{decimal:"200"}
    }}) { edges {
        node {
            category title content numViews
        }
    }
}

```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod apply_query_wildcard(*queryset, options, value*)

Apply *wildcard* filter.

Syntax:

TODO

Example:

```

query {
  allPostDocuments(filter:{category:{wildcard:"ytho"}}) {
    edges {
      node {
        category title content numViews comments
      }
    }
  }
}

```

Parameters

- **queryset** (*elasticsearch_dsl.search.Search*) – Original queryset.
- **options** (*dict*) – Filter options.
- **value** (*str*) – value to filter on.

Returns

Modified queryset.

Return type

elasticsearch_dsl.search.Search

classmethod `get_gte_lte_params`(*value, lookup, options*)

Get params for *gte*, *gt*, *lte* and *lt* query.

Syntax:

TODO

Example:

```

{
  allPostDocuments(filter:{numViews:{
    gt:{decimal:"100"}, lt:{decimal:"200"}
  }}) { edges {
    node {
      category title content numViews
    }
  }
}

```

Parameters

- **value** (*graphene_elastic.filter_backends.filtering.queries.InputObjectType*) –
- **lookup** (*str*) –
- **options** (*dict*) –

Returns

Params to be used in *range* query.

Return type

dict

classmethod `get_range_param_value(value)`

Get range param value.

Parameters**value** (*graphene_elastic.filter_backends.filtering.queries.InputObjectType*) –**Returns****classmethod** `get_range_params(value, options)`Get params for *range* query.

Syntax:

TODO

Example:

```

{
  allPostDocuments(filter:{numViews:{range:{
    lower:{decimal:"100"}, upper:{decimal: "200"},
    boost:"2.0"
  }}}) {
    edges {
      node {
        category title content numViews
      }
    }
  }
}

```

Parameters

- **value** (*graphene_elastic.filter_backends.filtering.queries.InputObjectType*) –
- **options** (*dict*) –

ReturnsParams to be used in *range* query.**Return type**

dict

split_lookup_complex_value: Callable

14.20.1.1.1.9 graphene_elastic.filter_backends.filtering.queries module

14.20.1.1.1.10 Module contents

14.20.1.1.1.11 graphene_elastic.filter_backends.highlight package

14.20.1.1.1.12 Submodules

14.20.1.1.1.13 graphene_elastic.filter_backends.highlight.common module

`class graphene_elastic.filter_backends.highlight.common.HighlightFilterBackend(connection_field, args=None)`

Bases: *BaseBackend*

Highlight filter backend.

field_belongs_to(*field_name*)

Check if given filter field belongs to the backend.

Parameters

field_name –

Returns

filter(*queryset*)

Filter.

Parameters

queryset –

Returns

get_backend_document_fields()

Get additional document fields for the backend.

For instance, the Highlight backend add additional field named `highlight` to the list of fields.

Sample query:

```
query {
  allPostDocuments(search:{title:{value:"alice"}}) {
    edges {
      node {
        id title highlight
      }
    }
  }
}
```

Sample response:

```
{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
```



```

        }, 'description': {}},
    }
}
Sample query would be:
query {
  allPostDocuments(
    search: {content: {value:"since"}, title: {value:"decide"}}, highlight:[category, content]
  ) { edges {
    node {
      title content highlight
    } cursor
  }
}
}
Returns
  Filtering options.
Return type
  dict

```

14.20.1.1.1.14 Module contents

14.20.1.1.1.15 graphene_elastic.filter_backends.ordering package

14.20.1.1.1.16 Submodules

14.20.1.1.1.17 graphene_elastic.filter_backends.ordering.common module

Ordering backend.

```
class graphene_elastic.filter_backends.ordering.common.DefaultOrderingFilterBackend(connection_field,
                                                                                   args=None)
```

Bases: *BaseBackend*, *OrderingMixin*

Default ordering filter backend for Elasticsearch.

Make sure this is your last ordering backend.

Example:

```

>>> import graphene
>>> from graphene import Node
>>> from graphene_elastic import (
>>>     ElasticsearchObjectType,
>>>     ElasticsearchConnectionField,
>>> )
>>> from graphene_elastic.filter_backends import (
>>>     FilteringFilterBackend,
>>>     SearchFilterBackend,
>>>     OrderingFilterBackend,

```

(continues on next page)

(continued from previous page)

```

>>>     DefaultOrderingFilterBackend,
>>> )
>>> from graphene_elastic.constants import (
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_TERM,
>>>     LOOKUP_FILTER_TERMS,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_IN,
>>> )
>>>
>>> from search_index.documents import Post as PostDocument
>>>
>>> class Post(ElasticsearchObjectType):
>>>
>>>     class Meta(object):
>>>         document = PostDocument
>>>         interfaces = (Node,)
>>>         filter_backends = [
>>>             FilteringFilterBackend,
>>>             SearchFilterBackend,
>>>             OrderingFilterBackend,
>>>             DefaultOrderingFilterBackend
>>>         ]
>>>         filter_fields = {
>>>             'id': '_id',
>>>             'title': {
>>>                 'field': 'title.raw',
>>>                 'lookups': [
>>>                     LOOKUP_FILTER_TERM,
>>>                     LOOKUP_FILTER_TERMS,
>>>                     LOOKUP_FILTER_PREFIX,
>>>                     LOOKUP_FILTER_WILDCARD,
>>>                     LOOKUP_QUERY_IN,
>>>                     LOOKUP_QUERY_EXCLUDE,
>>>                 ],
>>>                 'default_lookup': LOOKUP_FILTER_TERM,
>>>             },
>>>             'category': 'category.raw',
>>>             'tags': 'tags.raw',
>>>             'num_views': 'num_views',
>>>         }
>>>         search_fields = {
>>>             'title': {'boost': 4},
>>>             'content': {'boost': 2},
>>>             'category': None,
>>>         }
>>>         ordering_fields = {
>>>             'id': None,
>>>             'title': 'title.raw',
>>>             'created_at': 'created_at',
>>>             'num_views': 'num_views',

```

(continues on next page)

(continued from previous page)

```

>>>     }
>>>
>>>     ordering_defaults = ('id', 'title',)

```

filter(*queryset*)

Filter the queryset.

Parameters**queryset** (*elasticsearch_dsl.search.Search*) – Base queryset.**Returns**

Updated queryset.

Return type*elasticsearch_dsl.search.Search***get_default_ordering_params**()

Get the default ordering params for the view.

Returns

Ordering params to be used for ordering.

Return type

list

get_ordering_query_params()

Get ordering query params.

Returns

Ordering params to be used for ordering.

Return type

list

has_query_fields = False**property ordering_defaults**

Ordering filter fields.

property ordering_fields

Ordering filter fields.

prefix = 'ordering'

```
class graphene_elastic.filter_backends.ordering.common.OrderingFilterBackend(connection_field,
                                                                              args=None)
```

Bases: *BaseBackend*, *OrderingMixin*

Ordering filter backend for Elasticsearch.

Example:

```

>>> import graphene
>>> from graphene import Node
>>> from graphene_elastic import (
>>>     ElasticsearchObjectType,
>>>     ElasticsearchConnectionField,
>>> )
>>> from graphene_elastic.filter_backends import (
>>>     FilteringFilterBackend,
>>>     SearchFilterBackend,
>>>     OrderingFilterBackend,

```

(continues on next page)

(continued from previous page)

```

>>>     DefaultOrderingFilterBackend,
>>> )
>>> from graphene_elastic.constants import (
>>>     LOOKUP_FILTER_PREFIX,
>>>     LOOKUP_FILTER_TERM,
>>>     LOOKUP_FILTER_TERMS,
>>>     LOOKUP_FILTER_WILDCARD,
>>>     LOOKUP_QUERY_EXCLUDE,
>>>     LOOKUP_QUERY_IN,
>>> )
>>>
>>> from search_index.documents import Post as PostDocument
>>>
>>> class Post(ElasticsearchObjectType):
>>>
>>>     class Meta(object):
>>>         document = PostDocument
>>>         interfaces = (Node,)
>>>         filter_backends = [
>>>             FilteringFilterBackend,
>>>             SearchFilterBackend,
>>>             OrderingFilterBackend,
>>>             DefaultOrderingFilterBackend
>>>         ]
>>>         filter_fields = {
>>>             'id': '_id',
>>>             'title': {
>>>                 'field': 'title.raw',
>>>                 'lookups': [
>>>                     LOOKUP_FILTER_TERM,
>>>                     LOOKUP_FILTER_TERMS,
>>>                     LOOKUP_FILTER_PREFIX,
>>>                     LOOKUP_FILTER_WILDCARD,
>>>                     LOOKUP_QUERY_IN,
>>>                     LOOKUP_QUERY_EXCLUDE,
>>>                 ],
>>>                 'default_lookup': LOOKUP_FILTER_TERM,
>>>             },
>>>             'category': 'category.raw',
>>>             'tags': 'tags.raw',
>>>             'num_views': 'num_views',
>>>         }
>>>         search_fields = {
>>>             'title': {'boost': 4},
>>>             'content': {'boost': 2},
>>>             'category': None,
>>>         }
>>>         ordering_fields = {
>>>             'id': None,
>>>             'title': 'title.raw',
>>>             'created_at': 'created_at',
>>>             'num_views': 'num_views',

```

(continues on next page)

```

>>>     }
>>>
>>>     ordering_defaults = ('id', 'title',)

```

The basic usage would be:

```

query {
  allPostDocuments(ordering:{title:ASC}) {
    edges {
      node {
        title content category numViews createdAt
      }
    }
  }
}

```

field_belongs_to(*field_name*)

Check if given filter field belongs to the backend.

Parameters

field_name –

Returns

filter(*queryset*)

Filter the queryset.

Parameters

queryset (*elasticsearch_dsl.search.Search*) – Base queryset.

Returns

Updated queryset.

Return type

elasticsearch_dsl.search.Search

get_backend_default_query_fields_params()

Get default query fields params for the backend.

Return type

dict

Returns

get_field_type(*field_name, field_value, base_field_type*)

Get field type.

Returns

get_ordering_query_params()

Get ordering query params.

Returns

Ordering params to be used for ordering.

Return type

list

has_query_fields = True

property ordering_args_mapping

property ordering_fields

Ordering filter fields.

```
prefix = 'ordering'
score_field_name = 'score'
```

14.20.1.1.1.18 Module contents

14.20.1.1.1.19 graphene_elastic.filter_backends.post_filter package

14.20.1.1.1.20 Submodules

14.20.1.1.1.21 graphene_elastic.filter_backends.post_filter.common module

```
class graphene_elastic.filter_backends.post_filter.common.PostFilterFilteringBackend(connection_field,
                                                                                   args=None)
```

Bases: *BaseBackend, FilteringFilterMixin*

Post filter filtering backend.

```
classmethod apply_filter(queryset, options=None, args=None, kwargs=None)
```

Apply filter.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

```
classmethod apply_query(queryset, options=None, args=None, kwargs=None)
```

Apply query.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

```
field_belongs_to(field_name)
```

Check if given filter field belongs to the backend.

Parameters

- **field_name** –

Returns

```
filter(queryset)
```

Filter.

```
property filter_args_mapping
```

```
property filter_fields
```

Filtering filter fields.

```
get_backend_query_fields(items, is_filterable_func, get_type_func)
```

Fail proof override.

Parameters

- **items** –

- `is_filterable_func` –
- `get_type_func` –

Returns

`get_field_lookup_param(field_name)`

Get field lookup param.

Parameters

`field_name` –

Returns

`get_field_options(field_name)`

Get field option params.

Parameters

`field_name` –

Returns

`get_field_type(field_name, field_value, base_field_type)`

Get field type.

Returns

`get_filter_query_params()`

Get query params to be filtered on.

We can either specify it like this:

```
query_params = {
    'category': {
        'value': 'Elastic',
    }
}
```

Or using specific lookup:

```
query_params = {
    'category': {
        'term': 'Elastic', 'range': {
            'lower': Decimal('3.0')
        }
    }
}
```

Note, that *value* would only work on simple types (string, integer, decimal). For complex types you would have to use complex param anyway. Therefore, it should be forbidden to set *default_lookup* to a complex field type.

Sample values:

```
query_params = {
    'category': {
        'value': 'Elastic',
    }
}

filter_fields = {
    'category': {
        'field': 'category.raw', 'default_lookup': 'term', 'lookups': (
            'term', 'terms', 'range', 'exists', 'prefix', 'wildcard', 'contains', 'in',
            'gt', 'gte', 'lt', 'lte', 'starts_with', 'ends_with', 'is_null', 'exclude'
        )
    }
}
```

```

    }
  }
  field_name = 'category'

```

has_query_fields = True

prefix = 'postFilter'

prepare_filter_fields()

Prepare filter fields.

Possible structures:

```

post_filter_fields = {
  'title': {
    'field': 'title.raw', 'lookups': [
      LOOKUP_FILTER_TERM,          LOOKUP_FILTER_TERMS,
      LOOKUP_FILTER_PREFIX,       LOOKUP_FILTER_WILDCARD,
      LOOKUP_QUERY_IN,            LOOKUP_QUERY_EXCLUDE,
    ], 'default_lookup': LOOKUP_FILTER_TERM,
  }, 'category': 'category.raw',
}

```

We shall finally have:

```

post_filter_fields = {
  'title': {
    'field': 'title.raw', 'lookups': [
      LOOKUP_FILTER_TERM,          LOOKUP_FILTER_TERMS,
      LOOKUP_FILTER_PREFIX,       LOOKUP_FILTER_WILDCARD,
      LOOKUP_QUERY_IN,            LOOKUP_QUERY_EXCLUDE,
    ], 'default_lookup': LOOKUP_FILTER_TERM,
  }, 'category': {
    'field': 'category.raw', 'lookups': [
      LOOKUP_FILTER_TERM,         LOOKUP_FILTER_TERMS,
      LOOKUP_FILTER_PREFIX,
      LOOKUP_FILTER_WILDCARD,     LOOKUP_QUERY_IN,
      LOOKUP_QUERY_EXCLUDE, ... # All other lookups
    ], 'default_lookup': LOOKUP_FILTER_TERM,
  }
}

```

prepare_query_params()

Prepare query params.

Returns

14.20.1.1.1.22 Module contents

14.20.1.1.1.23 graphene_elastic.filter_backends.score package

14.20.1.1.1.24 Submodules

14.20.1.1.1.25 graphene_elastic.filter_backends.score.common module

class graphene_elastic.filter_backends.score.common.**ScoreFilterBackend**(*connection_field*,
args=None)

Bases: *BaseBackend*

Score filter backend.

Sample query would be:

```

query {
  allPostDocuments(
    search:{content:{value:"since"}, title:{value:"decide"}}
  ) { edges {
    node {
      title content score
    }
  } cursor
}

```

field_belongs_to(*field_name*)

Check if given filter field belongs to the backend.

Parameters

field_name –

Returns

filter(*queryset*)

Filter.

Parameters

queryset –

Returns

get_backend_document_fields()

Get additional document fields for the backend.

For instance, the Highlight backend add additional field named `highlight` to the list of fields.

Sample query:

```

query {
  allPostDocuments(search:{title:{value:"alice"}}) {
    edges {
      node {
        id title highlight
      }
    }
  }
}

```

```

    }
  }
}
Sample response:
{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
          "node": {
            "id": "UG9zdDp5a1ppVlcwQklwZ2dXbVIJQV91Rw==",
            "title": "thus Alice style", "highlight": {
              "title": [
                "thus <b>Alice</b> style"
              ]
            }
          }
        }
      ]
    }
  }
}

```

That `highlight` part on both sample query and sample response isn't initially available on the connection level, but added with help of the filter backend. `:return:`

```

has_query_fields = False

prefix = 'score'

score_field_name = 'score'

property score_fields
    Score filter fields.

```

14.20.1.1.1.26 Module contents

14.20.1.1.1.27 `graphene_elastic.filter_backends.search` package

14.20.1.1.1.28 Submodules

14.20.1.1.1.29 `graphene_elastic.filter_backends.search.common` module

```

class graphene_elastic.filter_backends.search.common.SearchFilterBackend(connection_field,
                                                                           args=None)

```

Bases: `BaseBackend`

Search filter backend.

clean_all_query_params()

Get cleaned query params. Remove query lookup.

construct_nested_search()

Construct nested search.

Type 1:

```
>>> search_nested_fields = {
>>>     "comments": {
>>>         "path": "comments",
>>>         "fields": [
>>>             "tag",
>>>             "content"
>>>         ]
>>> }
```

Type 2:

```
>>> search_nested_fields = {
>>>     "comments": {
>>>         "path": "comments",
>>>         "fields": [
>>>             {
>>>                 "tag": {
>>>                     "field": "tag.raw",
>>>                     "boost": 4
>>>                 }
>>>             },
>>>             {
>>>                 "content": {
>>>                     "field": "content",
>>>                     "boost": 2
>>>                 }
>>>             }
>>>         ]
>>> }
```

In GraphQL shall be:

```
query {
  allPostDocuments(search:{
    comments: {
      tag: {
        value: "Python", boost: 2
      }
    }
  }) { pageInfo {
    startCursor endCursor hasNextPage hasPreviousPage
  } edges {
    cursor node {
```



```

query {
  allPostDocuments(search:{
    query:"Another",          title:{value:"Another"}    sum-
    mary:{value:"Another"}
  }) { pageInfo {
    startCursor endCursor hasNextPage hasPreviousPage
  } edges {
    cursor node {
      category title content numViews
    }
  }
}

```

Or simply:

```

query {
  allPostDocuments(search:{query:"education technology"}) {
    pageInfo {
      startCursor endCursor hasNextPage hasPreviousPage
    } edges {
      cursor node {
        category title content numViews
      }
    }
  }
}

```

Returns

Updated queryset.

Return type

elasticsearch_dsl.search.Search

field_belongs_to(*field_name*)

Check if given filter field belongs to the backend.

Parameters

field_name –

Returns

filter(*queryset*)

Filter.

Parameters

queryset –

Returns

get_all_query_params()

get_backend_default_query_fields_params()

Get backend default filter params.

Return type

dict

Returns**get_field_type(field_name, field_value, base_field_type)**

Get field type.

Returns

TODO: required

get_search_nested_fields_tree(start=None, value=None)

We got a prepared nested fields ,

```
{
  'country': {
    'path': 'country', 'fields': [
      {
        'name': {'boost': 2}
      }
    ]
  }, 'city': {
    'path': 'country.city', 'fields': [
      {
        'name': {'boost': 2}
      }
    ]
  }
}
```

Then we should turn it to

```
{
  'country': {
    'name': {}, # {} or None represents no more leaves. 'city': {
      'name': {}
    }
  }
}
```

has_query_fields = True

property nested_search_args_mapping

prefix = 'search'

prepare_search_fields()

Prepare search fields.

Possible structures:

```
search_fields = {
  'title': {'boost': 4, 'field': 'title.raw'}, 'content': {'boost': 2}, 'category': None, 'com-
ments': None
}
```

We shall finally have:

```
search_fields = {
```

```

        'title': {
            'field': 'title.raw', 'boost': 4
        }, 'content': {
            'field': 'content', 'boost': 2
        }, 'category': {
            'field': 'category'
        }
    }
}

```

Sample query would be:

```

{
    allPostDocuments(search:{query:"Another"}) {
        pageInfo {
            startCursor endCursor hasNextPage hasPreviousPage
        } edges {
            cursor node {
                category title content numViews
            }
        }
    }
}

```

Returns

Filtering options.

Return type

dict

prepare_search_nested_fields()

Prepare search fields.

Possible structures:

Type1

```

search_nested_fields = {
    'comments': {
        'path': 'comments', 'fields': [
            {'author': {'boost': 4}}, {'tag': {'boost': 2}},
        ]
    }
}

```

Type2

```

search_nested_fields = {
    'comments': {
        'path': 'comments', 'fields': ['author', 'tag']
    }
}

```

We shall finally have:

```

search_nested_fields = {
    'comments': {
        'path': 'comments', 'fields': {
            {'author': {'boost': 4}}, {'tag': {'boost': 2}}
        }
    }
}

```

```

    }
  }
}
Sample query would be:
{
  allPostDocuments(search:{query:"Another"}) {
    pageInfo {
      startCursor endCursor hasNextPage hasPreviousPage
    } edges {
      cursor node {
        category title content numViews
      }
    }
  }
}

```

Returns
Filtering options.

Return type
dict

property search_args_mapping

property search_fields

Search filter fields.

property search_nested_fields

Search nested filter fields.

14.20.1.1.1.30 graphene_elastic.filter_backends.search.query_string module

class graphene_elastic.filter_backends.search.query_string.QueryStringBackend(*connection_field*,
args=None)

Bases: *BaseBackend*

filter(*queryset*)

Filter.

Parameters

queryset –

Returns

get_all_query_params()

get_backend_query_fields(*items*, *is_filterable_func*, *get_type_func*)

Construct backend query fields.

Parameters

- **items** –
- **is_filterable_func** –
- **get_type_func** –

Returns

```
has_query_fields = True
```

```
prefix = 'query_string'
```

```
property query_string_options
```

Query string options.

Possible options:

```
    query_string_options = {
        'fields': ['title', 'category', 'tag'], 'default_operator': "and", 'lenient': true, 'minimum_should_match': 3
    }
```

For list of all options: <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-simple-query-string-query.html>

14.20.1.1.1.31 graphene_elastic.filter_backends.search.simple_query_string module

```
class graphene_elastic.filter_backends.search.simple_query_string.SimpleQueryStringBackend(connection_field, args=None)
```

Bases: *BaseBackend*

```
filter(queryset)
```

Filter.

Parameters

queryset –

Returns

```
get_all_query_params()
```

```
get_backend_query_fields(items, is_filterable_func, get_type_func)
```

Construct backend query fields.

Parameters

- **items** –
- **is_filterable_func** –
- **get_type_func** –

Returns

```
has_query_fields = True
```

```
prefix = 'simple_query_string'
```

```
property simple_query_string_options
```

Simple query string options.

Possible options:

```
    simple_query_string_options = {
        'fields': ['title', 'category', 'tag'], 'default_operator': "and", 'lenient': true, 'minimum_should_match': 3
    }
```

For list of all options: <https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-simple-query-string-query.html>

14.20.1.1.1.32 Module contents

14.20.1.1.1.33 graphene_elastic.filter_backends.source package

14.20.1.1.1.34 Submodules

14.20.1.1.1.35 graphene_elastic.filter_backends.source.common module

class graphene_elastic.filter_backends.source.common.**SourceFilterBackend**(*connection_field*,
args=None)

Bases: *BaseBackend*

Source filter backend.

filter(*queryset*)

Filter.

Parameters

queryset –

Returns

get_backend_query_fields(*items*, *is_filterable_func*, *get_type_func*)

Construct backend filtering fields.

Parameters

- **items** –
- **is_filterable_func** –
- **get_type_func** –

Returns

has_query_fields = True

prefix = 'source'

prepare_source_fields()

Prepare source fields.

Possible structures:

source_fields = ["title"]

Or:

search_fields = ["title", "author.*"]

Or:

```
source = {
    "includes": ["title", "author.*"], "excludes": [ "*.*.description" ]
}
```

Returns

Filtering options.

Return type

dict

property source_fields

Source filter fields.

14.20.1.1.1.36 Module contents

14.20.1.1.2 Submodules

14.20.1.1.3 graphene_elastic.filter_backends.base module

class graphene_elastic.filter_backends.base.**BaseBackend**(*connection_field*, *args=None*)

Bases: object

Base backend.

alter_connection(*connection*, *slice*)

Alter connection object.

You can add various properties here, returning the altered object. Typical use-case would be adding facets to the connection.

Parameters

- **connection** –
- **slice** –

Returns

classmethod **apply_filter**(*queryset*, *options=None*, *args=None*, *kwargs=None*)

Apply filter.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

classmethod **apply_query**(*queryset*, *options=None*, *args=None*, *kwargs=None*)

Apply query.

Parameters

- **queryset** –
- **options** –
- **args** –
- **kwargs** –

Returns

property **doc_type**

Shortcut to the Elasticsearch document type.

Returns

field_belongs_to(*field_name*)

Check if given filter field belongs to the backend.

Parameters

- **field_name** –

Returns

filter(*queryset*)

Filter. This method alters current queryset.

Parameters

- **queryset** –

Returns

classmethod generic_query_fields()

Generic backend specific query fields.

For instance, for search filter backend it would be `{'search': String()}`.

Returns

Rtype dict

get_backend_connection_fields()

Get backend connection fields.

Typical use-case - a backend that alters the Connection object and adds additional fields next to *edges* and *pageInfo* (see the *graphene_elastic.relay.connection.Connection* for more information).

Rtype dict

Returns

get_backend_connection_fields_type()

Get backend connection fields type.

Typical use-case - a backend that alters the Connection object and adds additional fields next to *edges* and *pageInfo* (see the *graphene_elastic.relay.connection.Connection* for more information).

Returns

get_backend_default_query_fields_params()

Get default query fields params for the backend.

Return type

dict

Returns

get_backend_document_fields()

Get additional document fields for the backend.

For instance, the Highlight backend add additional field named `highlight` to the list of fields.

Sample query:

```
query {
  allPostDocuments(search:{title:{value:"alice"}}) {
    edges {
      node {
        id title highlight
      }
    }
  }
}
```

Sample response:

```
{
  "data": {
    "allPostDocuments": {
      "edges": [
        {
          "node": {
            "id": "UG9zdDp5a1ppVlcwQklwZ2dXbVlJQV91Rw==",
            "title": "thus Alice style", "highlight": {
```

```
        "title": [
            "thus <b>Alice</b> style"
        ]
    }
}
]
```

That `highlight` part on both sample query and sample response isn't initially available on the connection level, but added with help of the filter backend. `:return:`

get_backend_query_fields(*items*, *is_filterable_func*, *get_type_func*)

Construct backend query fields.

Parameters

- **items** –
- **is_filterable_func** –
- **get_type_func** –

Returns

get_field_type(*field_name*, *field_value*, *base_field_type*)

Get field type.

Returns

has_connection_fields = False

has_query_fields = False

prefix = None

classmethod split_lookup_complex_multiple_value(*value*, *maxsplit=-1*)

Split lookup complex multiple value.

Parameters

- **value** (*str*) – Value to split.
- **maxsplit** (*int*) – The *maxsplit* option of *string.split*.

Returns

Lookup filter split into a list.

Return type

list

classmethod split_lookup_complex_value(*value*, *maxsplit=-1*)

Split lookup complex value.

Parameters

- **value** (*str*) – Value to split.
- **maxsplit** (*int*) – The *maxsplit* option of *string.split*.

Returns

Lookup filter split into a list.

Return type

list

classmethod split_lookup_filter(*value*, *maxsplit=-1*)

Split lookup filter.

Parameters

- **value** (*str*) – Value to split.
- **maxsplit** (*int*) – The *maxsplit* option of *string.split*.

Returns

Lookup filter split into a list.

Return type

list

classmethod `split_lookup_name`(*value*, *maxsplit=-1*)

Split lookup value.

Parameters

- **value** (*str*) – Value to split.
- **maxsplit** (*int*) – The *maxsplit* option of *string.split*.

Returns

Lookup value split into a list.

Return type

list

14.20.1.1.4 graphene_elastic.filter_backends.queries module

class `graphene_elastic.filter_backends.queries.Contains`(*args, **kwargs)

Bases: `String`

Wildcard.

filter:[

 {category: {contains: “lish”}}}]

]

required = `True`

class `graphene_elastic.filter_backends.queries.Direction`(*args, **kwargs)

Bases: `Enum`

ASC = `'asc'`

DESC = `'desc'`

class `graphene_elastic.filter_backends.queries.EndsWith`(*args, **kwargs)

Bases: `String`

Ends with.

filter:[

 {category: {endsWith: “dren”}}}]

]

required = `True`

class `graphene_elastic.filter_backends.queries.Exclude`(*args, **kwargs)

Bases: `_ListOfTypeString`

Exclude.

filter:[

 {category: {exclude: [“Python”, “Django”}}}]

]

```

class graphene_elastic.filter_backends.queries.Exists(*args, **kwargs)
    Bases: Boolean
    Exists.
    filter:[
        {category: {exist: true}}}]
    ]
    required = True

class graphene_elastic.filter_backends.queries.GeoBoundingBox(*args, **kwargs)
    Bases: InputObjectType
    Geo polygon.
    filter:[
        {place: {geoBoundingBox: {
            topLeft: {lat: "40.0", lon="70.0"}, bottomRight: {lat: "80.0", lon: "90.0"}
        }}}
    ]
    bottom_right = <graphene.types.field.Field object>
    top_left = <graphene.types.field.Field object>

class graphene_elastic.filter_backends.queries.GeoDistance(*args, **kwargs)
    Bases: InputObjectType
    Geo distance.
    filter:[
        {place: {geoDistance: {distance: "9km", lat: "40.0", lon="70.0"}}}]
    ]
    distance = <graphene.types.scalars.String object>
    lat = <graphene.types.decimal.Decimal object>
    lon = <graphene.types.decimal.Decimal object>

class graphene_elastic.filter_backends.queries.GeoPolygon(*args, **kwargs)
    Bases: InputObjectType
    Geo polygon.
    filter:[
        {place: {geoPolygon: {points: [{lat: "40.0", lon="70.0"}]}}}]
    ]
    points = <graphene.types.structures.List object>

class graphene_elastic.filter_backends.queries.Gt(*args, **kwargs)
    Bases: _NumberOrDate
    Gt.
    filter:[
        {category: {gt: "1"}}}]
    ]
    required = True

```

```

class graphene_elastic.filter_backends.queries.Gte(*args, **kwargs)
    Bases: _NumberOrDate
    Gte.
    filter:[
        {category: {gte: "1"}}}
    ]
    required = True

class graphene_elastic.filter_backends.queries.In(*args, **kwargs)
    Bases: _ListOfTypeString
    In.
    filter:[
        {category: {in: ["Python", "Django"]}}
    ]

class graphene_elastic.filter_backends.queries.IsNull(*args, **kwargs)
    Bases: Boolean
    Is null.
    filter:[
        {category: {isNull: true}}}]
    ]
    required = True

class graphene_elastic.filter_backends.queries.Lt(*args, **kwargs)
    Bases: _NumberOrDate
    Lt.
    filter:[
        {category: {lt: "1"}}}
    ]
    required = True

class graphene_elastic.filter_backends.queries.Lte(*args, **kwargs)
    Bases: _NumberOrDate
    Lte.
    filter:[
        {category: {lte: "1"}}}
    ]
    required = True

class graphene_elastic.filter_backends.queries.Point(*args, **kwargs)
    Bases: InputObjectType
    Point. Helper for many geo lookups.
    lat = <graphene.types.decimal.Decimal object>
    lon = <graphene.types.decimal.Decimal object>

```

```

class graphene_elastic.filter_backends.queries.Prefix(*args, **kwargs)
    Bases: String
    Prefix.
    filter:[
        {category: {prefix: "bio"}}]
    ]
    required = True

class graphene_elastic.filter_backends.queries.Range(*args, **kwargs)
    Bases: InputObjectType
    Range.
    filter:[
        {range: {lower: "1000", upper: "2000", boost: "2.0"}}]
    ]
    boost = <graphene.types.decimal.Decimal object>
    lower = <graphene.types.field.Field object>
    upper = <graphene.types.field.Field object>

class graphene_elastic.filter_backends.queries.StartsWith(*args, **kwargs)
    Bases: Prefix
    Starts with (alias of prefix).

class graphene_elastic.filter_backends.queries.Term(*args, **kwargs)
    Bases: String
    Terms.
    filter:[
        {category: {term: "Python"}}]
    ]
    required = True

class graphene_elastic.filter_backends.queries.Terms(*args, **kwargs)
    Bases: _ListOfTypeString
    Terms.
    filter:[
        {category: {terms: ["Python", "Django"]}}]
    ]

class graphene_elastic.filter_backends.queries.Wildcard(*args, **kwargs)
    Bases: String
    Wildcard.
    filter:[
        {category: {wildcard: "bio"}}]
    ]
    required = True

```

14.20.1.1.5 Module contents

14.20.1.2 graphene_elastic.relay package

14.20.1.2.1 Submodules

14.20.1.2.2 graphene_elastic.relay.connection module

Some overrides of the original `graphql_relay.connection.connection` module. For sanity and ease of updates/sync with modifications from upstream, this module isn't formatted in accordance with the rest of the package. Pull requests code-style changes wouldn't be accepted.

```
class graphene_elastic.relay.connection.Connection(*args, **kwargs)
```

Bases: `ObjectType`

```
class graphene_elastic.relay.connection.ConnectionOptions(class_type)
```

Bases: `ObjectTypeOptions`

```
node = None
```

14.20.1.2.3 graphene_elastic.relay.connectiontypes module

Some overrides of the original `graphql_relay.connection.connectiontypes` module. For sanity and ease of updates/sync with modifications from upstream, this module isn't formatted in accordance with the rest of the package. Pull requests code-style changes wouldn't be accepted.

```
class graphene_elastic.relay.connectiontypes.Connection(edges, page_info, facets=None)
```

Bases: `object`

```
to_dict()
```

```
class graphene_elastic.relay.connectiontypes.Facets(facets)
```

Bases: `object`

```
to_dict()
```

14.20.1.2.4 Module contents

14.20.1.3 graphene_elastic.tests package

14.20.1.3.1 Submodules

14.20.1.3.2 graphene_elastic.tests.base module

```
class graphene_elastic.tests.base.BaseGrapheneElasticTestCase(methodName='runTest')
```

Bases: `TestCase`

Base graphene-elastic test case.

```
create_elasticsearch_indexes()
```

Create ES indexes.

remove_elasticsearch_index(*index_name*, *retry=0*)

remove_elasticsearch_indexes()

Remove all ES indexes.

setUp()

Hook method for setting up the test fixture before exercising it.

classmethod setUpClass()

Hook method for setting up class fixture before running tests in the class.

classmethod sleep(*value=3*)

tearDown()

Hook method for deconstructing the test fixture after testing it.

14.20.1.3.3 graphene_elastic.tests.test_faceted_search_backend module

class graphene_elastic.tests.test_faceted_search_backend.FacetedSearchBackendElasticTestCase(*methodName*)

Bases: *BaseGrapheneElasticTestCase*

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.4 graphene_elastic.tests.test_filter_backend module

class graphene_elastic.tests.test_filter_backend.FilterBackendElasticTestCase(*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

endpoint = 'allPostDocuments'

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.5 graphene_elastic.tests.test_highlight_backend module

class graphene_elastic.tests.test_highlight_backend.**HighlightBackendElasticTestCase**(*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

query_name = 'allPostDocuments'

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

class graphene_elastic.tests.test_highlight_backend.**HighlightCompoundBackendElasticTestCase**(*methodName='runTest'*)

Bases: *HighlightBackendElasticTestCase*

query_name = 'allReadOnlyPostDocuments'

14.20.1.3.6 graphene_elastic.tests.test_ordering_backend module

class graphene_elastic.tests.test_ordering_backend.**OrderingBackendElasticTestCase**(*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.7 graphene_elastic.tests.test_pagination module

class graphene_elastic.tests.test_pagination.**PaginationTestCase**(*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

query_name = 'allPostDocuments'

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.8 graphene_elastic.tests.test_post_filter_backend module

`class graphene_elastic.tests.test_post_filter_backend.PostFilterBackendElasticTestCase` (*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

`setUp()`

Hook method for setting up the test fixture before exercising it.

`test_all()`

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.9 graphene_elastic.tests.test_query_string_backend module

`class graphene_elastic.tests.test_query_string_backend.QueryStringBackendElasticTestCase` (*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

`setUp()`

Hook method for setting up the test fixture before exercising it.

`test_all()`

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.10 graphene_elastic.tests.test_score_backend module

`class graphene_elastic.tests.test_score_backend.ScoreBackendElasticTestCase` (*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

`setUp()`

Hook method for setting up the test fixture before exercising it.

`test_all()`

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.11 graphene_elastic.tests.test_search_backend module

`class graphene_elastic.tests.test_search_backend.CompoundSearchBackendElasticTestCase` (*methodName='runTest'*)

Bases: *SearchBackendElasticTestCase*

`query_name = 'allReadOnlyPostDocuments'`

`class graphene_elastic.tests.test_search_backend.SearchBackendElasticTestCase` (*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

`query_name = 'allPostDocuments'`

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.12 graphene_elastic.tests.test_simple_query_string_backend module

class `graphene_elastic.tests.test_simple_query_string_backend.SimpleQueryStringBackendElasticTestCase`(*m*

Bases: *BaseGrapheneElasticTestCase*

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.13 graphene_elastic.tests.test_source_backend module

class `graphene_elastic.tests.test_source_backend.HighlightBackendElasticTestCase`(*methodName='runTest'*)

Bases: *BaseGrapheneElasticTestCase*

setUp()

Hook method for setting up the test fixture before exercising it.

test_all()

Test all.

Since we don't write in specific tests, it's more efficient to run them all from a single method in order to save on speed ups between tests.

14.20.1.3.14 graphene_elastic.tests.test_versions module

class `graphene_elastic.tests.test_versions.VersionsTest`(*methodName='runTest'*)

Bases: *TestCase*

Tests of `graphene_elastic.versions` module.

setUp()

Hook method for setting up the test fixture before exercising it.

test_elasticsearch_dsl_6_3_0()

Tests as if we were using `elasticsearch_dsl==6.3.0`.

test_elasticsearch_dsl_7_0_0()

Tests as if we were using `elasticsearch_dsl==7.0.0`.

14.20.1.3.15 Module contents

14.20.1.4 graphene_elastic.types package

14.20.1.4.1 Submodules

14.20.1.4.2 graphene_elastic.types.elastic_types module

class graphene_elastic.types.elastic_types.ElasticsearchObjectType(*args, **kwargs)

Bases: ObjectType

classmethod get_node(info, id)

property id

classmethod is_type_of(root, info)

classmethod rescan_fields()

Attempts to rescan fields and will insert any not converted initially.

resolve_id(info)

class graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions(class_type)

Bases: ObjectTypeOptions

connection = None

document = None

registry = None

graphene_elastic.types.elastic_types.**construct_fields**(document, registry, only_fields, exclude_fields)

Args:

document (elasticsearch_dsl.Document): registry (graphene_elastic.registry.Registry): only_fields ([str]): exclude_fields ([str]):

Returns:

(OrderedDict, OrderedDict): converted fields and self reference fields.

graphene_elastic.types.elastic_types.**construct_self_referenced_fields**(self_referenced, registry)

14.20.1.4.3 graphene_elastic.types.json_string module

class graphene_elastic.types.json_string.ElasticJSONString(*args, **kwargs)

Bases: JSONString

static serialize(dt)

graphene_elastic.types.json_string.**to_serializable**(o)

14.20.1.4.4 Module contents

14.20.2 Submodules

14.20.3 graphene_elastic.advanced_types module

```
class graphene_elastic.advanced_types.FileFieldType(*args, **kwargs)
```

```
    Bases: ObjectType
```

```
    chunk_size = <graphene.types.scalars.Int object>
```

```
    content_type = <graphene.types.scalars.String object>
```

```
    data = <graphene.types.scalars.String object>
```

```
    length = <graphene.types.scalars.Int object>
```

```
    md5 = <graphene.types.scalars.String object>
```

```
    resolve_chunk_size(info)
```

```
    resolve_content_type(info)
```

```
    resolve_data(info)
```

```
    resolve_length(info)
```

```
    resolve_md5(info)
```

```
class graphene_elastic.advanced_types.MultiPolygonFieldType(*args, **kwargs)
```

```
    Bases: _CoordinatesTypeField
```

```
    coordinates = <graphene.types.structures.List object>
```

```
class graphene_elastic.advanced_types.PointFieldType(*args, **kwargs)
```

```
    Bases: _CoordinatesTypeField
```

```
    coordinates = <graphene.types.structures.List object>
```

```
class graphene_elastic.advanced_types.PolygonFieldType(*args, **kwargs)
```

```
    Bases: _CoordinatesTypeField
```

```
    coordinates = <graphene.types.structures.List object>
```

14.20.4 graphene_elastic.arrayconnection module

```
graphene_elastic.arrayconnection.connection_from_list_slice(list_slice, args=None,
                                                            connection_type=None,
                                                            edge_type=None,
                                                            pageinfo_type=None, slice_start=0,
                                                            list_length=0, list_slice_length=None,
                                                            connection_field=None)
```

Given a slice (subset) of an array, returns a connection object for use in GraphQL. This function is similar to `connectionFromArray`, but is intended for use cases where you know the cardinality of the connection, consider it too large to materialize the entire array, and instead wish pass in a slice of the total result large enough to cover the range specified in `args`.

14.20.5 graphene_elastic.compat module

14.20.6 graphene_elastic.constants module

Constants module. Contains Elasticsearch constants, lookup constants, functional constants, suggesters, etc.

14.20.7 graphene_elastic.converter module

exception graphene_elastic.converter.ElasticsearchConversionError

Bases: Exception

graphene_elastic.converter.convert_elasticsearch_field(*field*, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Keyword, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Text, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Short, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Long, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Integer, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Byte, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Boolean, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: ScaledFloat, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Float, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: HalfFloat, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Double, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Date, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Nested, registry=None)

graphene_elastic.converter.convert_elasticsearch_field(*field*: Object, registry=None)

graphene_elastic.converter.convert_field_to_boolean(*field*, registry=None)

graphene_elastic.converter.convert_field_to_complex_object(*field*, registry=None)

graphene_elastic.converter.convert_field_to_datetime(*field*, registry=None)

graphene_elastic.converter.convert_field_to_float(*field*, registry=None)

graphene_elastic.converter.convert_field_to_int(*field*, registry=None)

graphene_elastic.converter.convert_field_to_jsonstring(*field*, registry=None)

graphene_elastic.converter.convert_field_to_string(*field*, registry=None)

graphene_elastic.converter.singledispatch(*func*)

Single-dispatch generic function decorator.

Transforms a function into a generic function, which can have different behaviours depending upon the type of its first argument. The decorated function acts as the default implementation, and additional implementations can be registered using the register() attribute of the generic function.

14.20.8 graphene_elastic.enums module

class graphene_elastic.enums.NoValue(*value*)

Bases: Enum

String values in enum.

Example:

```
>>> class Color(NoValue):
>>>     RED = 'stop'
>>>     GREEN = 'go'
>>>     BLUE = 'too fast!'
```

Graphene example:

```
>>> @graphene.Enum.from_enum
>>> class ColorOptions(NoValue):
>>>
>>>     RED = 'stop'
>>>     GREEN = 'go'
>>>     BLUE = 'too fast!'
```

graphene_elastic.enums.convert_list_to_enum(*values*, *enum_name*='DynamicEnum', *upper*=True)

Prepare list values for creating an Enum.

Example:

```
>>> values = ['red', 'green', 'blue']
>>> print(prepare_list_for_enum(values))
{'RED': 'red', 'GREEN': 'green', 'BLUE': 'blue'}
```

14.20.9 graphene_elastic.fields module

class graphene_elastic.fields.ElasticsearchConnectionField(*type*, **args*, ***kwargs*)

Bases: IterableConnectionField

property args

chained_resolver(*resolver*, *is_partial*, *root*, *info*, ***args*)

classmethod connection_resolver(*resolver*, *connection_type*, *root*, *info*, *connection_field*=None, ***args*)

property default_filter_backends

default_resolver(*_root*, *info*, ***args*)

property doc_type

property document

property field_args

property fields

property `filter_backends`

property `get_queryset(document, info, **args)`

property `get_resolver(parent_resolver)`

property `node_type`

property `reference_args`

property `registry`

classmethod `resolve_connection(connection_type, args, resolved, connection_field=None)`

property `type`

14.20.10 graphene_elastic.logging module

14.20.11 graphene_elastic.registry module

class `graphene_elastic.registry.Registry`

Bases: `object`

property `get_converted_field(field)`

property `get_type_for_document(document)`

property `register(cls)`

property `register_converted_field(field, converted)`

`graphene_elastic.registry.get_global_registry()`

`graphene_elastic.registry.reset_global_registry()`

14.20.12 graphene_elastic.settings module

14.20.13 graphene_elastic.utils module

`graphene_elastic.utils.get_document_fields(document, excluding=None)`

`graphene_elastic.utils.get_field_description(field, registry=None)`

Common metadata includes `verbose_name` and `help_text`.

<http://docs.mongoengine.org/apireference.html#fields>

`graphene_elastic.utils.get_node_from_global_id(node, info, global_id)`

`graphene_elastic.utils.get_type_for_document(schema, document)`

`graphene_elastic.utils.import_single_dispatch()`

`graphene_elastic.utils.is_valid_elasticsearch_document(document)`

14.20.14 graphene_elastic.versions module

Contains information about the current Elasticsearch version in use, including (LTE and GTE).

`graphene_elastic.versions.get_elasticsearch_version(default=(2, 0, 0))`

Get Elasticsearch version.

Parameters

default (*tuple*) – Default value. Mainly added for building the docs when Elasticsearch is not running.

Returns

Return type

list

14.20.15 Module contents

class `graphene_elastic.ElasticsearchConnectionField`(*type, *args, **kwargs*)

Bases: `IterableConnectionField`

property `args`

chained_resolver(*resolver, is_partial, root, info, **args*)

classmethod `connection_resolver`(*resolver, connection_type, root, info, connection_field=None, **args*)

property `default_filter_backends`

default_resolver(*_root, info, **args*)

property `doc_type`

property `document`

property `field_args`

property `fields`

property `filter_backends`

get_queryset(*document, info, **args*)

get_resolver(*parent_resolver*)

property `node_type`

property `reference_args`

property `registry`

classmethod `resolve_connection`(*connection_type, args, resolved, connection_field=None*)

property `type`

class `graphene_elastic.ElasticsearchObjectType`(**args, **kwargs*)

Bases: `ObjectType`

classmethod `get_node`(*info, id*)

property `id`

classmethod `is_type_of(root, info)`

classmethod `rescan_fields()`

Attempts to rescan fields and will insert any not converted initially.

resolve_id(*info*)

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

graphene_elastic, 137
graphene_elastic.advanced_types, 133
graphene_elastic.arrayconnection, 133
graphene_elastic.compat, 134
graphene_elastic.constants, 134
graphene_elastic.converter, 134
graphene_elastic.enums, 135
graphene_elastic.fields, 135
graphene_elastic.filter_backends, 127
graphene_elastic.filter_backends.base, 120
graphene_elastic.filter_backends.faceted_search, 85
graphene_elastic.filter_backends.faceted_search.common, 83
graphene_elastic.filter_backends.filtering, 100
graphene_elastic.filter_backends.filtering.common, 85
graphene_elastic.filter_backends.filtering.mixins, 89
graphene_elastic.filter_backends.filtering.queries, 100
graphene_elastic.filter_backends.highlight, 102
graphene_elastic.filter_backends.highlight.common, 100
graphene_elastic.filter_backends.ordering, 107
graphene_elastic.filter_backends.ordering.common, 102
graphene_elastic.filter_backends.post_filter, 110
graphene_elastic.filter_backends.post_filter.common, 107
graphene_elastic.filter_backends.queries, 123
graphene_elastic.filter_backends.score, 111
graphene_elastic.filter_backends.score.common, 110
graphene_elastic.filter_backends.search, 119
graphene_elastic.filter_backends.search.common, 111
graphene_elastic.filter_backends.search.query_string, 117
graphene_elastic.filter_backends.search.simple_query_string, 118
graphene_elastic.filter_backends.source, 120
graphene_elastic.filter_backends.source.common, 119
graphene_elastic.logging, 136
graphene_elastic.registry, 136
graphene_elastic.relay, 127
graphene_elastic.relay.connection, 127
graphene_elastic.relay.connectiontypes, 127
graphene_elastic.settings, 136
graphene_elastic.tests, 132
graphene_elastic.tests.base, 127
graphene_elastic.tests.test_faceted_search_backend, 128
graphene_elastic.tests.test_filter_backend, 128
graphene_elastic.tests.test_highlight_backend, 129
graphene_elastic.tests.test_ordering_backend, 129
graphene_elastic.tests.test_pagination, 129
graphene_elastic.tests.test_post_filter_backend, 130
graphene_elastic.tests.test_query_string_backend, 130
graphene_elastic.tests.test_score_backend, 130
graphene_elastic.tests.test_search_backend, 130
graphene_elastic.tests.test_simple_query_string_backend, 131
graphene_elastic.tests.test_source_backend, 131
graphene_elastic.tests.test_versions, 131
graphene_elastic.types, 133
graphene_elastic.types.elastic_types, 132
graphene_elastic.types.json_string, 132
graphene_elastic.utils, 136
graphene_elastic.versions, 137

A

- `aggregate()` (`graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend` class method), 93
- `aggregate()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterBackend` class method), 94
- `alter_connection()` (`graphene_elastic.filter_backends.base.BaseBackend` class method), 120
- `alter_connection()` (`graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend` class method), 95
- `alter_connection()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterBackend` class method), 95
- `apply_filter` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` attribute), 89
- `apply_filter()` (`graphene_elastic.filter_backends.base.BaseBackend` class method), 120
- `apply_filter()` (`graphene_elastic.filter_backends.post_filter.common.PostFilterFilteringBackend` class method), 107
- `apply_filter_prefix()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 89
- `apply_filter_range()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 90
- `apply_filter_term()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 90
- `apply_filter_terms()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 91
- `apply_query` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` attribute), 92
- `apply_query()` (`graphene_elastic.filter_backends.base.BaseBackend` class method), 120
- `apply_query()` (`graphene_elastic.filter_backends.post_filter.common.PostFilterFilteringBackend` class method), 107
- `apply_query_contains()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 92
- `apply_query_endswith()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 92
- `apply_query_exclude()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 93
- `apply_query_exists()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 93
- `apply_query_gt()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 94
- `apply_query_gte()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 95
- `apply_query_in()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 95
- `apply_query_isnull()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 96
- `apply_query_lt()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 96
- `apply_query_lte()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 97
- `apply_query_wildcard()` (`graphene_elastic.filter_backends.filtering.mixins.FilteringFilterMixin` class method), 97
- `args` (`graphene_elastic.ElasticsearchConnectionField` property), 137
- `args` (`graphene_elastic.fields.ElasticsearchConnectionField` property), 135
- ASC (`graphene_elastic.filter_backends.queries.Direction` attribute), 123

B

- `BaseBackend` (`graphene_elastic.filter_backends.base.BaseBackend` class), 120
- `BaseGrapheneElasticTestCase` (`graphene_elastic.tests.base` class), 127
- `boost` (`graphene_elastic.filter_backends.queries.Range` attribute), 126
- `bottom_right` (`graphene_elastic.filter_backends.queries.GeoBoundingBox` attribute), 124

C

- `ChainedFilterMixin` (`graphene_elastic.ElasticsearchConnectionField` method), 137
- `chained_resolver()` (`graphene_elastic.fields.ElasticsearchConnectionField` method), 135
- `chunk_size` (`graphene_elastic.advanced_types.FileFieldType` attribute), 133

- `clean_all_query_params()` (`graphene_elastic.filter_backends.search.common.SearchFilterBackend` method), 111
 - `CompoundSearchBackendElasticTestCase` (class in `graphene_elastic.tests.test_search_backend`), 130
 - `Connection` (class in `graphene_elastic.relay.connection`), 127
 - `Connection` (class in `graphene_elastic.relay.connectiontypes`), 127
 - `connection` (`graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions` attribute), 132
 - `connection_from_list_slice()` (in module `graphene_elastic.arrayconnection`), 133
 - `connection_resolver()` (`graphene_elastic.ElasticsearchConnectionFactory` class method), 137
 - `connection_resolver()` (`graphene_elastic.fields.ElasticsearchConnectionFactory` class method), 135
 - `ConnectionOptions` (class in `graphene_elastic.relay.connection`), 127
 - `construct_facets()` (`graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend` method), 83
 - `construct_fields()` (in module `graphene_elastic.types.elastic_types`), 132
 - `construct_nested_search()` (`graphene_elastic.filter_backends.search.common.SearchFilterBackend` method), 112
 - `construct_search()` (`graphene_elastic.filter_backends.search.common.SearchFilterBackend` method), 113
 - `construct_self_referenced_fields()` (in module `graphene_elastic.types.elastic_types`), 132
 - `Contains` (class in `graphene_elastic.filter_backends.queries`), 123
 - `content_type` (`graphene_elastic.advanced_types.FileFieldType` attribute), 133
 - `convert_elasticsearch_field()` (in module `graphene_elastic.converter`), 134
 - `convert_field_to_boolean()` (in module `graphene_elastic.converter`), 134
 - `convert_field_to_complex_object()` (in module `graphene_elastic.converter`), 134
 - `convert_field_to_datetime()` (in module `graphene_elastic.converter`), 134
 - `convert_field_to_float()` (in module `graphene_elastic.converter`), 134
 - `convert_field_to_int()` (in module `graphene_elastic.converter`), 134
 - `convert_field_to_jsonstring()` (in module `graphene_elastic.converter`), 134
 - `convert_field_to_string()` (in module `graphene_elastic.converter`), 134
 - `convert_list_to_enum()` (in module `graphene_elastic.enums`), 135
 - `coordinates` (`graphene_elastic.advanced_types.PointFieldType` attribute), 133
 - `coordinates` (`graphene_elastic.advanced_types.PolygonFieldType` attribute), 133
 - `create_elasticsearch_indexes()` (`graphene_elastic.tests.base.BaseGrapheneElasticTestCase` method), 127
- ## D
- `data` (`graphene_elastic.advanced_types.FileFieldType` attribute), 133
 - `default_filter_backends` (`graphene_elastic.ElasticsearchConnectionFactory` property), 137
 - `default_filter_backends` (`graphene_elastic.fields.ElasticsearchConnectionFactory` property), 135
 - `default_resolver()` (`graphene_elastic.ElasticsearchConnectionFactory` method), 137
 - `default_resolver()` (`graphene_elastic.fields.ElasticsearchConnectionFactory` method), 135
 - `DefaultOrderingFilterBackend` (class in `graphene_elastic.filter_backends.ordering.common`), 102
 - `direction` (`graphene_elastic.filter_backends.queries.Direction` attribute), 123
 - `direction` (`graphene_elastic.filter_backends.queries.Direction` attribute), 123
 - `distance` (`graphene_elastic.filter_backends.queries.GeoDistance` attribute), 124
 - `doc_type` (`graphene_elastic.ElasticsearchConnectionFactory` property), 137
 - `doc_type` (`graphene_elastic.fields.ElasticsearchConnectionFactory` property), 135
 - `doc_type` (`graphene_elastic.filter_backends.base.BaseBackend` property), 120
 - `document` (`graphene_elastic.ElasticsearchConnectionFactory` property), 137
 - `document` (`graphene_elastic.fields.ElasticsearchConnectionFactory` property), 135
 - `document` (`graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions` attribute), 132
- ## E
- `ElasticJSONString` (class in `graphene_elastic.types.json_string`), 132
 - `ElasticsearchConnectionFactory` (class in `graphene_elastic`), 137
 - `ElasticsearchConnectionFactory` (class in `graphene_elastic.fields`), 135
 - `ElasticsearchConversionError`, 134

ElasticsearchObjectType	(class in graphene_elastic), 137	filter()	(graphene_elastic.filter_backends.base.BaseBackend method), 120
ElasticsearchObjectType	(class in graphene_elastic.types.elastic_types), 132	filter()	(graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend method), 84
ElasticsearchObjectTypeOptions	(class in graphene_elastic.types.elastic_types), 132	filter()	(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend method), 86
endpoint	(graphene_elastic.tests.test_filter_backend.FilterBackendElasticTestCase attribute), 128	filter()	(graphene_elastic.filter_backends.highlight.common.HighlightFilterBackend method), 100
EndsWith	(class in graphene_elastic.filter_backends.queries), 123	filter()	(graphene_elastic.filter_backends.ordering.common.DefaultOrderingFilterBackend method), 104
Exclude	(class in graphene_elastic.filter_backends.queries), 123	filter()	(graphene_elastic.filter_backends.ordering.common.OrderingFilterBackend method), 106
Exists	(class in graphene_elastic.filter_backends.queries), 123	filter()	(graphene_elastic.filter_backends.post_filter.common.PostFilterBackend method), 107
F		filter()	(graphene_elastic.filter_backends.score.common.ScoreFilterBackend method), 110
faceted_search_fields	(graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend property), 84	filter()	(graphene_elastic.filter_backends.search.common.SearchFilterBackend method), 114
FacetedSearchBackendElasticTestCase	(class in graphene_elastic.tests.test_faceted_search_backend), 128	filter()	(graphene_elastic.filter_backends.search.simple_query_string.SimpleQueryStringFilterBackend method), 118
FacetedSearchFilterBackend	(class in graphene_elastic.filter_backends.faceted_search.common), 83	filter()	(graphene_elastic.filter_backends.source.common.SourceFilterBackend method), 119
Facets	(class in graphene_elastic.relay.connectiontypes), 127	filter_args_mapping	(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend property), 86
field_args	(graphene_elastic.ElasticsearchConnectionField property), 137	filter_args_mapping	(graphene_elastic.filter_backends.post_filter.common.PostFilterBackend property), 107
field_args	(graphene_elastic.fields.ElasticsearchConnectionField property), 135	filter_backends	(graphene_elastic.ElasticsearchConnectionField property), 137
field_belongs_to()	(graphene_elastic.filter_backends.base.BaseBackend method), 120	filter_backends	(graphene_elastic.fields.ElasticsearchConnectionField property), 135
field_belongs_to()	(graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend method), 84	filter_fields	(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend property), 86
field_belongs_to()	(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend method), 85	filter_fields	(graphene_elastic.filter_backends.post_filter.common.PostFilterBackend property), 107
field_belongs_to()	(graphene_elastic.filter_backends.highlight.common.HighlightFilterBackend method), 100	FilterBackendElasticTestCase	(class in graphene_elastic.tests.test_filter_backend), 128
field_belongs_to()	(graphene_elastic.filter_backends.ordering.common.OrderingFilterBackend method), 106	FilteringFilterBackend	(class in graphene_elastic.filter_backends.filtering.common), 85
field_belongs_to()	(graphene_elastic.filter_backends.post_filter.common.PostFilterBackend method), 107	FilteringFilterMixin	(class in graphene_elastic.filter_backends.filtering.mixins), 89
field_belongs_to()	(graphene_elastic.filter_backends.score.common.ScoreFilterBackend method), 110	GeoBoundingBox	(class in graphene_elastic.filter_backends.queries),
field_belongs_to()	(graphene_elastic.filter_backends.search.common.SearchFilterBackend method), 114		
fields	(graphene_elastic.ElasticsearchConnectionField property), 137	G	
fields	(graphene_elastic.fields.ElasticsearchConnectionField property), 135	generic_query_fields()	(graphene_elastic.filter_backends.base.BaseBackend class method), 120
FileFieldType	(class in graphene_elastic.advanced_types), 133		

<code>get_gte_lte_params()</code> (<code>graphene_elastic.filter_backends.filtering.mixins.FilterMixin</code> class method), 98	<code>graphene_elastic.enums</code> module, 115
<code>get_highlight_query_params()</code> (<code>graphene_elastic.filter_backends.highlight.common.HighlightBackend</code> method), 101	<code>graphene_elastic.fields</code> module, 135
<code>get_nested_field_type()</code> (<code>graphene_elastic.filter_backends.filtering.common.FilterMixin</code> method), 88	<code>graphene_elastic.filter_backends</code> module, 127
<code>get_node()</code> (<code>graphene_elastic.ElasticsearchObjectType</code> class method), 137	<code>graphene_elastic.filter_backends.base</code> module, 123
<code>get_node()</code> (<code>graphene_elastic.types.elastic_types.ElasticsearchObjectType</code> class method), 132	<code>graphene_elastic.filter_backends.faceted_search</code> module, 85
<code>get_node_from_global_id()</code> (in <code>module</code> <code>graphene_elastic.utils</code>), 136	<code>graphene_elastic.filter_backends.faceted_search.common</code> module, 82
<code>get_ordering_query_params()</code> (<code>graphene_elastic.filter_backends.ordering.common.OrderingBackend</code> method), 104	<code>graphene_elastic.filter_backends.filtering</code> module, 100
<code>get_ordering_query_params()</code> (<code>graphene_elastic.filter_backends.ordering.common.OrderingBackend</code> method), 106	<code>graphene_elastic.filter_backends.filtering.common</code> module, 85
<code>get_queryset()</code> (<code>graphene_elastic.ElasticsearchConnectionFactory</code> method), 137	<code>graphene_elastic.filter_backends.filtering.mixins</code> module, 89
<code>get_queryset()</code> (<code>graphene_elastic.fields.ElasticsearchConnectionFactory</code> method), 136	<code>graphene_elastic.filter_backends.filtering.queries</code> module, 100
<code>get_range_param_value()</code> (<code>graphene_elastic.filter_backends.filtering.mixins.FilterMixin</code> class method), 99	<code>graphene_elastic.filter_backends.highlight</code> module, 102
<code>get_range_params()</code> (<code>graphene_elastic.filter_backends.filtering.mixins.FilterMixin</code> class method), 99	<code>graphene_elastic.filter_backends.highlight.common</code> module, 100
<code>get_resolver()</code> (<code>graphene_elastic.ElasticsearchConnectionFactory</code> method), 137	<code>graphene_elastic.filter_backends.ordering</code> module, 107
<code>get_resolver()</code> (<code>graphene_elastic.fields.ElasticsearchConnectionFactory</code> method), 136	<code>graphene_elastic.filter_backends.ordering.common</code> module, 102
<code>get_search_nested_fields_tree()</code> (<code>graphene_elastic.filter_backends.search.common.SearchBackend</code> method), 115	<code>graphene_elastic.filter_backends.post_filter</code> module, 110
<code>get_type_for_document()</code> (<code>graphene_elastic.registry.Registry</code> method), 136	<code>graphene_elastic.filter_backends.post_filter.common</code> module, 107
<code>get_type_for_document()</code> (in <code>module</code> <code>graphene_elastic.utils</code>), 136	<code>graphene_elastic.filter_backends.queries</code> module, 123
<code>graphene_elastic</code> module, 137	<code>graphene_elastic.filter_backends.score</code> module, 110
<code>graphene_elastic.advanced_types</code> module, 133	<code>graphene_elastic.filter_backends.score.common</code> module, 110
<code>graphene_elastic.arrayconnection</code> module, 133	<code>graphene_elastic.filter_backends.search</code> module, 119
<code>graphene_elastic.compat</code> module, 134	<code>graphene_elastic.filter_backends.search.common</code> module, 111
<code>graphene_elastic.constants</code> module, 134	<code>graphene_elastic.filter_backends.search.query_string</code> module, 117
<code>graphene_elastic.converter</code> module, 134	<code>graphene_elastic.filter_backends.search.simple_query_string</code> module, 118
	<code>graphene_elastic.filter_backends.source</code> module, 120
	<code>graphene_elastic.filter_backends.source.common</code> module, 119
	<code>graphene_elastic.logging</code> module, 136
	<code>graphene_elastic.registry</code> module, 136

graphene_elastic.relay	attribute), 122
module, 127	
graphene_elastic.relay.connection	has_connection_fields
module, 127	(<i>graphene_elastic.filter_backends.faceted_search.common.FacetedSearchBackend</i> attribute), 85
graphene_elastic.relay.connectiontypes	has_query_fields(<i>graphene_elastic.filter_backends.base.BaseBackend</i> attribute), 122
module, 127	
graphene_elastic.settings	has_query_fields(<i>graphene_elastic.filter_backends.faceted_search.common.FacetedSearchBackend</i> attribute), 85
module, 136	
graphene_elastic.tests	has_query_fields(<i>graphene_elastic.filter_backends.filtering.common.FilteringBackend</i> attribute), 88
module, 132	
graphene_elastic.tests.base	has_query_fields(<i>graphene_elastic.filter_backends.highlight.common.HighlightBackend</i> attribute), 101
module, 127	
graphene_elastic.tests.test_faceted_search_backend	has_query_fields(<i>graphene_elastic.filter_backends.ordering.common.OrderingBackend</i> attribute), 104
module, 128	
graphene_elastic.tests.test_filter_backend	has_query_fields(<i>graphene_elastic.filter_backends.ordering.common.OrderingBackend</i> attribute), 106
module, 128	
graphene_elastic.tests.test_highlight_backend	has_query_fields(<i>graphene_elastic.filter_backends.post_filter.common.PostFilterBackend</i> attribute), 109
module, 129	
graphene_elastic.tests.test_ordering_backend	has_query_fields(<i>graphene_elastic.filter_backends.score.common.ScoreBackend</i> attribute), 111
module, 129	
graphene_elastic.tests.test_pagination	has_query_fields(<i>graphene_elastic.filter_backends.search.common.SearchBackend</i> attribute), 115
module, 129	
graphene_elastic.tests.test_post_filter_backend	has_query_fields(<i>graphene_elastic.filter_backends.search.query_string.QueryStringBackend</i> attribute), 117
module, 130	
graphene_elastic.tests.test_query_string_backend	has_query_fields(<i>graphene_elastic.filter_backends.search.simple_query_string.SimpleQueryStringBackend</i> attribute), 118
module, 130	
graphene_elastic.tests.test_score_backend	has_query_fields(<i>graphene_elastic.filter_backends.source.common.SourceBackend</i> attribute), 119
module, 130	
graphene_elastic.tests.test_search_backend	highlight_fields(<i>graphene_elastic.filter_backends.highlight.common.HighlightBackend</i> property), 101
module, 130	
graphene_elastic.tests.test_simple_query_string_backend	<i>HighlightBackendElasticTestCase</i> (class in <i>graphene_elastic.tests.test_highlight_backend</i>), 129
module, 131	
graphene_elastic.tests.test_source_backend	<i>HighlightBackendElasticTestCase</i> (class in <i>graphene_elastic.tests.test_source_backend</i>), 131
module, 131	
graphene_elastic.tests.test_versions	<i>HighlightCompoundBackendElasticTestCase</i> (class in <i>graphene_elastic.tests.test_highlight_backend</i>), 129
module, 131	
graphene_elastic.types	<i>HighlightFilterBackend</i> (class in <i>graphene_elastic.filter_backends.highlight.common</i>), 100
module, 133	
graphene_elastic.types.elastic_types	
module, 132	
graphene_elastic.types.json_string	
module, 132	
graphene_elastic.utils	
module, 136	
graphene_elastic.versions	<i>id</i> (<i>graphene_elastic.ElasticsearchObjectType</i> property), 137
module, 137	
Gt (class in <i>graphene_elastic.filter_backends.queries</i>), 124	<i>id</i> (<i>graphene_elastic.types.elastic_types.ElasticsearchObjectType</i> property), 132
Gte (class in <i>graphene_elastic.filter_backends.queries</i>), 124	<i>import_single_dispatch</i> () (in module <i>graphene_elastic.utils</i>), 136
	<i>In</i> (class in <i>graphene_elastic.filter_backends.queries</i>), 125
H	<i>is_type_of</i> () (<i>graphene_elastic.ElasticsearchObjectType</i> class method), 138
has_connection_fields	
(<i>graphene_elastic.filter_backends.base.BaseBackend</i>	

- `is_type_of()` (*graphene_elastic.types.elastic_types.ElasticsearchObjectType* class method), 132
 - `is_valid_elasticsearch_document()` (in module *graphene_elastic.utils*), 136
 - `IsNull` (class in *graphene_elastic.filter_backends.queries*), 125
- L**
- `lat` (*graphene_elastic.filter_backends.queries.GeoDistance* attribute), 124
 - `lat` (*graphene_elastic.filter_backends.queries.Point* attribute), 125
 - `length` (*graphene_elastic.advanced_types.FileFieldType* attribute), 133
 - `lon` (*graphene_elastic.filter_backends.queries.GeoDistance* attribute), 124
 - `lon` (*graphene_elastic.filter_backends.queries.Point* attribute), 125
 - `lower` (*graphene_elastic.filter_backends.queries.Range* attribute), 126
 - `Lt` (class in *graphene_elastic.filter_backends.queries*), 125
 - `Lte` (class in *graphene_elastic.filter_backends.queries*), 125
- M**
- `md5` (*graphene_elastic.advanced_types.FileFieldType* attribute), 133
 - module
 - `graphene_elastic`, 137
 - `graphene_elastic.advanced_types`, 133
 - `graphene_elastic.arrayconnection`, 133
 - `graphene_elastic.compat`, 134
 - `graphene_elastic.constants`, 134
 - `graphene_elastic.converter`, 134
 - `graphene_elastic.enums`, 135
 - `graphene_elastic.fields`, 135
 - `graphene_elastic.filter_backends`, 127
 - `graphene_elastic.filter_backends.base`, 120
 - `graphene_elastic.filter_backends.faceted_search`, 85
 - `graphene_elastic.filter_backends.faceted_search.common`, 83
 - `graphene_elastic.filter_backends.filtering`, 100
 - `graphene_elastic.filter_backends.filtering.common`, 85
 - `graphene_elastic.filter_backends.filtering.mixins`, 89
 - `graphene_elastic.filter_backends.filtering.queries`, 100
 - `graphene_elastic.filter_backends.highlight`, 102
 - `graphene_elastic.filter_backends.highlight.common`, 100
 - `graphene_elastic.filter_backends.ordering`, 107
 - `graphene_elastic.filter_backends.ordering.common`, 102
 - `graphene_elastic.filter_backends.post_filter`, 110
 - `graphene_elastic.filter_backends.post_filter.common`, 107
 - `graphene_elastic.filter_backends.queries`, 123
 - `graphene_elastic.filter_backends.score`, 111
 - `graphene_elastic.filter_backends.score.common`, 110
 - `graphene_elastic.filter_backends.search`, 119
 - `graphene_elastic.filter_backends.search.common`, 111
 - `graphene_elastic.filter_backends.search.query_string`, 117
 - `graphene_elastic.filter_backends.search.simple_query_string`, 118
 - `graphene_elastic.filter_backends.source`, 120
 - `graphene_elastic.filter_backends.source.common`, 119
 - `graphene_elastic.logging`, 136
 - `graphene_elastic.registry`, 136
 - `graphene_elastic.relay`, 127
 - `graphene_elastic.relay.connection`, 127
 - `graphene_elastic.relay.connectiontypes`, 127
 - `graphene_elastic.settings`, 136
 - `graphene_elastic.tests`, 132
 - `graphene_elastic.tests.base`, 127
 - `graphene_elastic.tests.test_faceted_search_backend`, 128
 - `graphene_elastic.tests.test_filter_backend`, 128
 - `graphene_elastic.tests.test_highlight_backend`, 128
 - `graphene_elastic.tests.test_ordering_backend`, 129
 - `graphene_elastic.tests.test_pagination`, 129
 - `graphene_elastic.tests.test_post_filter_backend`, 130
 - `graphene_elastic.tests.test_query_string_backend`, 130
 - `graphene_elastic.tests.test_score_backend`, 130
 - `graphene_elastic.tests.test_search_backend`, 130

130
graphene_elastic.tests.test_simple_query_string_backend (class in
graphene_elastic.tests.test_post_filter_backend), 130
131
graphene_elastic.tests.test_source_backendPostFilterFilteringBackend (class in
131 graphene_elastic.filter_backends.post_filter.common),
107
graphene_elastic.tests.test_versions, 131
graphene_elastic.types, 133
graphene_elastic.types.elastic_types, 132
graphene_elastic.types.json_string, 132
graphene_elastic.utils, 136
graphene_elastic.versions, 137
MultiPolygonFieldType (class in
graphene_elastic.advanced_types), 133

N

nested_search_args_mapping
(graphene_elastic.filter_backends.search.common.SearchFilterBackend
property), 115
node (graphene_elastic.relay.connection.ConnectionOptions
attribute), 127
node_type (graphene_elastic.ElasticsearchConnectionFields
property), 137
node_type (graphene_elastic.fields.ElasticsearchConnectionFields
property), 136
NoValue (class in graphene_elastic.enums), 135

O

ordering_args_mapping
(graphene_elastic.filter_backends.ordering.common.OrderingFilterBackend
property), 106
ordering_defaults (graphene_elastic.filter_backends.ordering.common.DefaultOrderingFilterBackend
property), 104
ordering_fields (graphene_elastic.filter_backends.ordering.common.DefaultOrderingFilterBackend
property), 104
ordering_fields (graphene_elastic.filter_backends.ordering.common.OrderingFilterBackend
property), 106
OrderingBackendElasticTestCase (class in
graphene_elastic.tests.test_ordering_backend),
129
OrderingFilterBackend (class in
graphene_elastic.filter_backends.ordering.common),
104

P

PaginationTestCase (class in
graphene_elastic.tests.test_pagination), 129
Point (class in graphene_elastic.filter_backends.queries),
125
PointFieldType (class in
graphene_elastic.advanced_types), 133
points (graphene_elastic.filter_backends.queries.GeoPolygon
attribute), 124
PolygonFieldType (class in
graphene_elastic.advanced_types), 133

PostFilterBackendElasticTestCase (class in
graphene_elastic.tests.test_post_filter_backend),
130
Prefix (class in graphene_elastic.filter_backends.queries),
125
prefix (graphene_elastic.filter_backends.base.BaseBackend
attribute), 122
prefix (graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend
attribute), 85
prefix (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend
attribute), 88
prefix (graphene_elastic.filter_backends.highlight.common.HighlightFilterBackend
attribute), 101
prefix (graphene_elastic.filter_backends.ordering.common.DefaultOrderingFilterBackend
attribute), 104
prefix (graphene_elastic.filter_backends.ordering.common.OrderingFilterBackend
attribute), 106
prefix (graphene_elastic.filter_backends.post_filter.common.PostFilterFilterBackend
attribute), 109
prefix (graphene_elastic.filter_backends.score.common.ScoreFilterBackend
attribute), 111
prefix (graphene_elastic.filter_backends.search.common.SearchFilterBackend
attribute), 115
prefix (graphene_elastic.filter_backends.search.query_string.QueryStringFilterBackend
attribute), 118
prefix (graphene_elastic.filter_backends.search.simple_query_string.SimpleQueryStringFilterBackend
attribute), 118
prefix (graphene_elastic.filter_backends.source.common.SourceFilterBackend
attribute), 119
prepare_faceted_search_fields()
(graphene_elastic.filter_backends.faceted_search.common.FacetedSearchFilterBackend
method), 85
prepare_filter_fields()
(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend
method), 88
prepare_filter_fields()
(graphene_elastic.filter_backends.post_filter.common.PostFilterFilterBackend
method), 109
prepare_highlight_fields()
(graphene_elastic.filter_backends.highlight.common.HighlightFilterBackend
method), 101
prepare_query_params()
(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend
method), 89
prepare_query_params()
(graphene_elastic.filter_backends.post_filter.common.PostFilterFilterBackend
method), 109
prepare_search_fields()
(graphene_elastic.filter_backends.search.common.SearchFilterBackend
method), 115
prepare_search_nested_fields()

(`graphene_elastic.filter_backends.search.common.SearchFieldBackend` method), 116

`prepare_source_fields()` (`graphene_elastic.filter_backends.source.common.SourceFileBackend` method), 119

Q

`query_name` (`graphene_elastic.tests.test_highlight_backend.HighlightBackendElasticTestCase` attribute), 129

`query_name` (`graphene_elastic.tests.test_highlight_backend.HighlightCompoundBackendElasticTestCase` attribute), 129

`query_name` (`graphene_elastic.tests.test_pagination.PaginationTestCase` attribute), 129

`query_name` (`graphene_elastic.tests.test_search_backend.CompoundSearchBackendElasticTestCase` attribute), 130

`query_name` (`graphene_elastic.tests.test_search_backend.SearchBackendElasticTestCase` attribute), 130

`query_string_options` (`graphene_elastic.filter_backends.search.query_string_options.QueryStringBackend` property), 118

`QueryStringBackend` (class in `graphene_elastic.filter_backends.search.query_string_options`), 117

`QueryStringBackendElasticTestCase` (class in `graphene_elastic.tests.test_query_string_backend`), 130

R

`Range` (class in `graphene_elastic.filter_backends.queries`), 126

`reference_args` (`graphene_elastic.ElasticsearchConnectionField` property), 137

`reference_args` (`graphene_elastic.fields.ElasticsearchConnectionField` property), 136

`register()` (`graphene_elastic.registry.Registry` method), 136

`register_converted_field()` (`graphene_elastic.registry.Registry` method), 136

`Registry` (class in `graphene_elastic.registry`), 136

`registry` (`graphene_elastic.ElasticsearchConnectionField` property), 137

`registry` (`graphene_elastic.fields.ElasticsearchConnectionField` property), 136

`registry` (`graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions` attribute), 132

`remove_elasticsearch_index()` (`graphene_elastic.tests.base.BaseGrapheneElasticTestCase` method), 127

`remove_elasticsearch_indexes()` (`graphene_elastic.tests.base.BaseGrapheneElasticTestCase` method), 128

`required` (`graphene_elastic.filter_backends.queries.Contains` attribute), 123

`required` (`graphene_elastic.filter_backends.queries.EndsWith` attribute), 123

`required` (`graphene_elastic.filter_backends.queries.Exists` attribute), 124

`required` (`graphene_elastic.filter_backends.queries.Gt` attribute), 124

`required` (`graphene_elastic.filter_backends.queries.Gte` attribute), 125

`required` (`graphene_elastic.filter_backends.queries.IsNull` attribute), 125

`required` (`graphene_elastic.filter_backends.queries.Lt` attribute), 125

`required` (`graphene_elastic.filter_backends.queries.Lte` attribute), 125

`required` (`graphene_elastic.filter_backends.queries.Prefix` attribute), 126

`required` (`graphene_elastic.filter_backends.queries.Term` attribute), 126

`required` (`graphene_elastic.filter_backends.queries.Wildcard` attribute), 126

`rescan_fields()` (`graphene_elastic.ElasticsearchObjectType` class method), 138

`rescan_fields()` (`graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions` class method), 132

`reset_global_registry()` (in `graphene_elastic.registry` module), 136

`resolve_chunk_size()` (`graphene_elastic.advanced_types.FileFieldType` method), 133

`resolve_connection()` (`graphene_elastic.ElasticsearchConnectionField` class method), 137

`resolve_connection()` (`graphene_elastic.fields.ElasticsearchConnectionField` class method), 136

`resolve_content_type()` (`graphene_elastic.advanced_types.FileFieldType` method), 133

`resolve_data()` (`graphene_elastic.advanced_types.FileFieldType` method), 133

`resolve_id()` (`graphene_elastic.ElasticsearchObjectType` method), 138

`resolve_id()` (`graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions` method), 132

`resolve_length()` (`graphene_elastic.advanced_types.FileFieldType` method), 133

`resolve_md5()` (`graphene_elastic.advanced_types.FileFieldType` method), 133

S

`score_field_name` (`graphene_elastic.filter_backends.ordering.common.Score` attribute), 107

`score_field_name` (`graphene_elastic.filter_backends.score.common.Score` attribute), 111

score_fields (graphene_elastic.filter_backends.score.common.ScoreFilterBackend property), 111

ScoreBackendElasticTestCase (class in graphene_elastic.filter_backends.search.simple_query_string), 130

ScoreFilterBackend (class in graphene_elastic.filter_backends.score.common), 110

search_args_mapping (graphene_elastic.filter_backends.search.common.SearchFilterBackend property), 117

search_fields (graphene_elastic.filter_backends.search.common.SourceFilterBackend property), 117

search_nested_fields (graphene_elastic.filter_backends.search.common.SearchFilterBackend property), 117

SearchBackendElasticTestCase (class in graphene_elastic.tests.test_search_backend), 130

SearchFilterBackend (class in graphene_elastic.filter_backends.search.common), 111

serialize() (graphene_elastic.types.json_string.ElasticJSONString static method), 132

setUp() (graphene_elastic.tests.base.BaseGrapheneElasticTestCase method), 128

setUp() (graphene_elastic.tests.test_faceted_search_backend.FacetedSearchBackendElasticTestCase method), 128

setUp() (graphene_elastic.tests.test_filter_backend.FilterBackendElasticTestCase method), 128

setUp() (graphene_elastic.tests.test_highlight_backend.HighlightBackendElasticTestCase method), 129

setUp() (graphene_elastic.tests.test_ordering_backend.OrderingBackendElasticTestCase method), 129

setUp() (graphene_elastic.tests.test_pagination.PaginationTestCase method), 129

setUp() (graphene_elastic.tests.test_post_filter_backend.PostFilterBackendElasticTestCase method), 130

setUp() (graphene_elastic.tests.test_query_string_backend.QueryStringBackendElasticTestCase method), 130

setUp() (graphene_elastic.tests.test_score_backend.ScoreBackendElasticTestCase method), 130

setUp() (graphene_elastic.tests.test_search_backend.SearchBackendElasticTestCase method), 130

setUp() (graphene_elastic.tests.test_simple_query_string_backend.SimpleQueryStringBackendElasticTestCase method), 131

setUp() (graphene_elastic.tests.test_source_backend.HighlightBackendElasticTestCase method), 131

setUp() (graphene_elastic.tests.test_versions.VersionsTestCase method), 131

setUpClass() (graphene_elastic.tests.base.BaseGrapheneElasticTestCase class method), 128

simple_query_string_options (graphene_elastic.filter_backends.search.simple_query_string.SimpleQueryStringBackend

SimpleQueryStringBackend (class in graphene_elastic.filter_backends.search.simple_query_string), 118

SimpleQueryStringBackendElasticTestCase (class in graphene_elastic.tests.test_simple_query_string_backend), 131

singledispatch() (in module graphene_elastic.converter), 134

sleep() (graphene_elastic.tests.base.BaseGrapheneElasticTestCase class method), 128

source_fields (graphene_elastic.filter_backends.source.common.SourceFilterBackend property), 119

SourceFilterBackend (class in graphene_elastic.filter_backends.source.common), 119

split_lookup_complex_multiple_value() (graphene_elastic.filter_backends.base.BaseBackend class method), 122

split_lookup_complex_value (graphene_elastic.filter_backends.filtering.mixins.FilteringFilterBackend attribute), 99

split_lookup_complex_value() (graphene_elastic.filter_backends.base.BaseBackend class method), 122

split_lookup_filter() (graphene_elastic.filter_backends.base.BaseBackend class method), 122

split_lookup_test_case() (graphene_elastic.filter_backends.base.BaseBackend class method), 122

StartsWith (class in graphene_elastic.filter_backends.queries), 126

tearDown() (graphene_elastic.tests.base.BaseGrapheneElasticTestCase method), 128

Term (class in graphene_elastic.filter_backends.queries), 126

Terms (class in graphene_elastic.filter_backends.queries), 126

test_all() (graphene_elastic.tests.test_faceted_search_backend.FacetedSearchBackendElasticTestCase method), 128

test_all() (graphene_elastic.tests.test_filter_backend.FilterBackendElasticTestCase method), 128

test_all() (graphene_elastic.tests.test_highlight_backend.HighlightBackendElasticTestCase method), 129

test_all() (graphene_elastic.tests.test_ordering_backend.OrderingBackendElasticTestCase method), 129

test_all() (graphene_elastic.tests.test_pagination.PaginationTestCase method), 129

test_all() (graphene_elastic.tests.test_post_filter_backend.PostFilterBackendElasticTestCase method), 130

test_all() (graphene_elastic.tests.test_query_string_backend.QueryStringBackendElasticTestCase method), 130

test_all() (graphene_elastic.tests.test_score_backend.ScoreBackendElasticTestCase method), 130

test_all() (graphene_elastic.tests.test_search_backend.SearchBackendElasticTestCase method), 130

test_all() (graphene_elastic.tests.test_simple_query_string_backend.SimpleQueryStringBackendElasticTestCase method), 131

test_all() (graphene_elastic.tests.test_source_backend.HighlightBackendElasticTestCase method), 131

test_all() (graphene_elastic.tests.test_versions.VersionsTestCase method), 131

[test_all\(\)](#) (*graphene_elastic.tests.test_query_string_backend.QueryStringBackendElasticTestCase* method), 130
[test_all\(\)](#) (*graphene_elastic.tests.test_score_backend.ScoreBackendElasticTestCase* method), 130
[test_all\(\)](#) (*graphene_elastic.tests.test_search_backend.SearchBackendElasticTestCase* method), 131
[test_all\(\)](#) (*graphene_elastic.tests.test_simple_query_string_backend.SimpleQueryStringBackendElasticTestCase* method), 131
[test_all\(\)](#) (*graphene_elastic.tests.test_source_backend.HighlightBackendElasticTestCase* method), 131
[test_elasticsearch_dsl_6_3_0\(\)](#) (*graphene_elastic.tests.test_versions.VersionsTest* method), 131
[test_elasticsearch_dsl_7_0_0\(\)](#) (*graphene_elastic.tests.test_versions.VersionsTest* method), 131
[to_dict\(\)](#) (*graphene_elastic.relay.connectiontypes.Connection* method), 127
[to_dict\(\)](#) (*graphene_elastic.relay.connectiontypes.Facets* method), 127
[to_serializable\(\)](#) (in *module graphene_elastic.types.json_string*), 132
[top_left](#) (*graphene_elastic.filter_backends.queries.GeoBoundingBox* attribute), 124
[type](#) (*graphene_elastic.ElasticsearchConnectionField* property), 137
[type](#) (*graphene_elastic.fields.ElasticsearchConnectionField* property), 136

U

[upper](#) (*graphene_elastic.filter_backends.queries.Range* attribute), 126

V

[VersionsTest](#) (class in *graphene_elastic.tests.test_versions*), 131

W

[Wildcard](#) (class in *graphene_elastic.filter_backends.queries*), 126