
graphene-elastic Documentation

Release 0.1

Artur Barseghyan <artur.barseghyan@gmail.com>

Sep 07, 2019

Contents

1	Prerequisites	3
2	Main features and highlights	5
3	Documentation	7
4	Installation	9
5	Examples	11
5.1	Install requirements	11
5.2	Populate sample data	11
5.3	Sample document definition	11
5.4	Sample apps	12
5.4.1	Sample Flask app	12
5.4.2	Sample Django app	12
5.4.3	ConnectionField example	13
5.4.3.1	Filter	15
5.4.3.1.1	Sample queries	15
5.4.3.1.2	Implemented filter lookups	16
5.4.3.2	Search	17
5.4.3.3	Ordering	17
5.4.3.4	Pagination	18
6	Road-map	19
7	Testing	21
7.1	Running Elasticsearch	21
7.2	Running tests	21
8	Debugging	23
9	Writing documentation	25
10	License	27
11	Support	29
12	Author	31

13	Project documentation	33
13.1	Concepts	34
13.1.1	Sample document definition	34
13.1.2	Sample schema definition	35
13.1.2.1	filter_backends	37
13.1.2.2	filter_fields	38
13.1.2.3	search_fields	39
13.1.2.4	ordering_fields	39
13.1.2.5	ordering_defaults	39
13.2	Quick start	39
13.2.1	Clone the repository	39
13.2.2	Start Elasticsearch	39
13.2.3	Install requirements	40
13.2.4	Populate dummy data	40
13.2.5	Open the browser	40
13.2.6	Make some experiments	40
13.2.7	Run tests	40
13.3	Search	40
13.4	Filtering	41
13.4.1	Filter lookups	41
13.4.1.1	Filter lookup contains	42
13.4.1.2	Filter lookup ends_with	42
13.4.1.3	Filter lookup exclude	42
13.4.1.4	Filter lookup exists	43
13.4.1.5	Filter lookup gt	43
13.4.1.6	Filter lookup gte	43
13.4.1.7	Filter lookup in	44
13.4.1.8	Filter lookup lt	44
13.4.1.9	Filter lookup lte	44
13.4.1.10	Filter lookup prefix	45
13.4.1.11	Filter lookup range	45
13.4.1.12	Filter lookup starts_with	45
13.4.1.13	Filter lookup term	46
13.4.1.14	Filter lookup terms	46
13.4.1.15	Filter lookup wildcard	46
13.5	Ordering	46
13.6	Pagination	47
13.6.1	Limits	47
13.6.2	Enforce first or last	47
13.6.3	User controlled pagination	47
13.7	Settings	48
13.8	Running Elasticsearch	49
13.8.1	Docker	49
13.8.1.1	Project default	49
13.8.1.2	Run specific version	50
13.8.1.2.1	6.x	50
13.8.1.2.2	7.x	50
13.9	FAQ	50
13.9.1	Questions and answers	50
13.10	Debugging	50
13.11	Release history and notes	51
13.11.1	0.1	51
13.11.2	0.0.13	51
13.11.3	0.0.12	51

13.11.4	0.0.11	51
13.11.5	0.0.10	52
13.11.6	0.0.9	52
13.11.7	0.0.8	52
13.11.8	0.0.7	52
13.11.9	0.0.6	52
13.11.100	0.5	52
13.11.110	0.4	53
13.11.120	0.3	53
13.11.130	0.2	53
13.11.140	0.1	53
13.12	graphene_elastic package	53
13.12.1	Subpackages	53
13.12.1.1	graphene_elastic.filter_backends package	53
13.12.1.1.1	Subpackages	53
13.12.1.1.1.1	graphene_elastic.filter_backends.filtering package	53
13.12.1.1.1.2	Submodules	53
13.12.1.1.1.3	graphene_elastic.filter_backends.filtering.common module	53
13.12.1.1.1.4	Module contents	65
13.12.1.1.1.5	graphene_elastic.filter_backends.search package	65
13.12.1.1.1.6	Submodules	65
13.12.1.1.1.7	graphene_elastic.filter_backends.search.common module	65
13.12.1.1.1.8	Module contents	68
13.12.1.1.2	Submodules	68
13.12.1.1.3	graphene_elastic.filter_backends.base module	68
13.12.1.1.4	Module contents	70
13.12.1.2	graphene_elastic.types package	70
13.12.1.2.1	Submodules	70
13.12.1.2.2	graphene_elastic.types.elastic_types module	70
13.12.1.2.3	graphene_elastic.types.json_string module	71
13.12.1.2.4	Module contents	71
13.12.2	Submodules	71
13.12.3	graphene_elastic.advanced_types module	71
13.12.4	graphene_elastic.constants module	72
13.12.5	graphene_elastic.converter module	72
13.12.6	graphene_elastic.enums module	72
13.12.7	graphene_elastic.fields module	73
13.12.8	graphene_elastic.helpers module	74
13.12.9	graphene_elastic.registry module	74
13.12.10	graphene_elastic.settings module	74
13.12.11	graphene_elastic.utils module	74
13.12.12	Module contents	74
14	Indices and tables	77
	Python Module Index	79
	Index	81

Elasticsearch (DSL) integration for Graphene.

Note: Project status is alpha.

CHAPTER 1

Prerequisites

- Graphene 2.x. *Support for Graphene 1.x is not planned, but might be considered.*
- Python 3.6, 3.7. *Support for Python 2 is not intended.*
- Elasticsearch 6.x, 7.x. *Support for Elasticsearch 5.x is not intended.*

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.
- Implemented search backend.
- Implemented filter backend.
- Implemented ordering backend.
- Implemented pagination.

See the [Road-map](#) for what's yet planned to implemented.

CHAPTER 3

Documentation

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

CHAPTER 4

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
```


5.1 Install requirements

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

5.2 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
```

5.3 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,
)
```

(continues on next page)

(continued from previous page)

```
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()
    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},
        }
```

5.4 Sample apps

5.4.1 Sample Flask app

Run the sample Flask app:

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

Open Flask graphiql client

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

5.4.2 Sample Django app

Run the sample Django app:

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

Open Django graphiql client

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

5.4.3 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,
    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,
            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`).
```

(continues on next page)

(continued from previous page)

```

'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},
    '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

```

(continues on next page)

(continued from previous page)

```

# 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)

```

5.4.3.1 Filter

5.4.3.1.1 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"]}}
  ) {

```

(continues on next page)

(continued from previous page)

```
edges {
  node {
    id
    title
    category
    content
    createdAt
    comments
  }
}
```

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

```
{
  allPostDocuments(filter:{
    category:{term:"Python"},
    numViews:{gt:"700"}
  }) {
    edges {
      node {
        category
        title
        comments
        numViews
      }
    }
  }
}
```

5.4.3.1.2 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.

5.4.3.2 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
      }
    }
  }
}
```

5.4.3.3 Ordering

Possible choices are ASC and DESC.

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

(continues on next page)

(continued from previous page)

```
        category
        title
        content
        numViews
        tags
      }
    }
  }
}
```

5.4.3.4 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
      }
    }
  }
}
```


Road-map and development plans.

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

- 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)
- Aggregations (faceted search) backend
- Post-filter backend
- Nested backend
- Highlight backend
- Suggester backend
- Global aggregations backend
- More-like-this backend
- Complex search backends, such as Simple query search
- Source filter backend

Stay tuned or reach out if you want to help.

Project is covered with tests.

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

7.1 Running Elasticsearch

Run Elasticsearch 7.x with Docker

```
docker-compose up elasticsearch
```

7.2 Running tests

Make sure you have the test requirements installed:

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

To test with all supported Python versions type:

```
tox
```

To test against specific environment, type:

```
tox -e py37
```

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
```

CHAPTER 8

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 10

License

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

CHAPTER 11

Support

For any issues contact me at the e-mail given in the *Author* section.

CHAPTER 12

Author

Artur Barseghyan <artur.barseghyan@gmail.com>

Contents:

Table of Contents

- *graphene-elastic*
 - *Prerequisites*
 - *Main features and highlights*
 - *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*
- *Road-map*
- *Testing*
 - * *Running Elasticsearch*
 - * *Running tests*
- *Debugging*
- *Writing documentation*
- *License*
- *Support*
- *Author*
- *Project documentation*
- *Indices and tables*

13.1 Concepts

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

13.1.1 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):
```

(continues on next page)

(continued from previous page)

```

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},
    }

```

13.1.2 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)

(continued from previous 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
        # 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`).
    }

```

(continues on next page)

(continued from previous page)

```

        '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)

```

13.1.2.1 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.

13.1.2.2 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.

13.1.2.3 search_fields

Used by `SearchFilterBackend` backend.

13.1.2.4 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.

13.1.2.5 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.

13.2 Quick start

13.2.1 Clone the repository

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

13.2.2 Start Elasticsearch

```
docker-compose up elasticsearch
```

13.2.3 Install requirements

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

13.2.4 Populate dummy data

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

13.2.5 Open the browser

<http://127.0.0.1:8000/graphql>

13.2.6 Make some experiments

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

13.2.7 Run tests

```
./runtests.py
```

13.3 Search

Search in all fields:

```
query {
  allPostDocuments(
    search:{query:"Release Box"}
  ) {
    edges {
```

(continues on next page)

(continued from previous page)

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

Search in specific fields:

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

13.4 Filtering

13.4.1 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

13.4.1.1 Filter lookup contains

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

13.4.1.2 Filter lookup ends_with

Note: endsWith for camelCase.

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

13.4.1.3 Filter lookup exclude

For a single term:

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

(continues on next page)

(continued from previous page)

```
}  
}
```

For multiple terms:

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

13.4.1.4 Filter lookup exists

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

13.4.1.5 Filter lookup gt

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

13.4.1.6 Filter lookup gte

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

13.4.1.7 Filter lookup in

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

13.4.1.8 Filter lookup lt

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

13.4.1.9 Filter lookup lte

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

```

(continues on next page)

(continued from previous page)

```

        content
        numViews
    }
}
}
}

```

13.4.1.10 Filter lookup prefix

```

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

```

13.4.1.11 Filter lookup range

```

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

```

13.4.1.12 Filter lookup starts_with

Note: startsWith for camelCase.

Alias for filter lookup “prefix”.

13.4.1.13 Filter lookup term

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

13.4.1.14 Filter lookup terms

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

13.4.1.15 Filter lookup wildcard

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

13.5 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
      }
    }
  }
}
```

13.6 Pagination

13.6.1 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.

13.6.2 Enforce first or last

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

13.6.3 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
      hasNextPage
      hasPreviousPage
    }
    edges {
      cursor
      node {
        category
        title
        content
        numViews
      }
    }
  }
}
```

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

13.7 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 = {
    "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)
)

```

13.8 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).

13.8.1 Docker

13.8.1.1 Project default

```
docker-compose up elasticsearch
```

13.8.1.2 Run specific version

13.8.1.2.1 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
```

13.8.1.2.2 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
```

13.9 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](#).

13.9.1 Questions and answers

13.10 Debugging

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",
```

(continues on next page)

(continued from previous page)

```
    json.dumps(DEFAULTS)
)
```

13.11 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.

13.11.1 0.1

2019-09-08

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

13.11.2 0.0.13

2019-09-07

- Documentation improvements and fixes.
- Clean up.

13.11.3 0.0.12

2019-09-06

Note: In memory of Erik Slim. RIP.

- More tests.

13.11.4 0.0.11

2019-09-05

- Fixes in search backend.

13.11.5 0.0.10

2019-09-04

- Fixes.
- Clean up.

13.11.6 0.0.9

2019-09-03

- Added pagination.
- Documentation improvements.

13.11.7 0.0.8

2019-09-02

- Tested default ordering backend.
- Documentation improvements.

13.11.8 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.

13.11.9 0.0.6

2019-08-30

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

13.11.10 0.0.5

2019-08-30

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

13.11.11 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.

13.11.12 0.0.3

2019-08-26

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

13.11.13 0.0.2

2019-08-25

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

13.11.14 0.0.1

2019-08-24

- Initial alpha release.

13.12 graphene_elastic package

13.12.1 Subpackages

13.12.1.1 graphene_elastic.filter_backends package

13.12.1.1.1 Subpackages

13.12.1.1.1.1 graphene_elastic.filter_backends.filtering package

13.12.1.1.1.2 Submodules

13.12.1.1.1.3 graphene_elastic.filter_backends.filtering.common module

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

Bases: *graphene_elastic.filter_backends.base.BaseBackend*

Filtering filter backend.

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:"100", upper:"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*

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:"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:"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(filter:{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:"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:"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`

field_belongs_to (*field_name*)

filter (*queryset*)

Filter.

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'
```

classmethod get_gte_lte_params (*value, lookup, options*)

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

Syntax:

```
TODO
```

Example:

```
{
    allPostDocuments(filter:{numViews:{gt:"100", lt:"200"}}) {
        edges {
```

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

Parameters

- **value** (*str*) –
- **lookup** (*str*) –
- **options** (*dict*) –

Returns Params to be used in *range* query.

Return type dict

classmethod **get_range_params** (*value*, *options*)

Get params for *range* query.

Syntax:

TODO

Example:

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

Parameters

- **value** (*str*) –
- **options** (*dict*) –

Returns Params to be used in *range* query.

Return type dict

has_fields = True

prefix = 'filter'

prepare_filter_fields()

Prepare filter fields.

Possible structures:

```
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:

```
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

13.12.1.1.1.4 Module contents

13.12.1.1.1.5 graphene_elastic.filter_backends.search package

13.12.1.1.1.6 Submodules

13.12.1.1.1.7 graphene_elastic.filter_backends.search.common module

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

Bases: *graphene_elastic.filter_backends.base.BaseBackend*

Search filter backend.

construct_search()

Construct search.

We have to deal with two types of structures:

Type 1:

```
>>> search_fields = (  
>>>     'title',  
>>>     'description',  
>>>     'summary',  
>>> )
```

Type 2:

```
>>> search_fields = {  
>>>     'title': {'field': 'title', 'boost': 2},  
>>>     'description': None,  
>>>     'summary': None,  
>>> }
```

In GraphQL shall be:

```
query {  
  allPostDocuments(search:{  
    query:"Another", title:{value:"Another"} summary:{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*)

filter (*queryset*)

Filter.

Parameters *queryset* –

Returns

get_all_query_params ()

get_backend_default_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

has_fields = True

prefix = 'search'

prepare_search_fields ()

Prepare search fields.

Possible structures:

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

13.12.1.1.1.8 Module contents

13.12.1.1.1.2 Submodules

13.12.1.1.1.3 graphene_elastic.filter_backends.base module

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

Bases: object

add_arg_prefix (*arg_name*)

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

arg_belongs_to (*arg_name*)

doc_type

field_belongs_to (*field_name*)

filter (*queryset*)

filter_fields

classmethod generic_fields ()

Generic backend specific fields.

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

Returns

Rtype dict

get_backend_default_fields_params ()

Backend default filter params.

Return type dict

Returns

get_backend_fields (*items, is_filterable_func, get_type_func*)

Construct backend fields.

Parameters

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

Returns

get_field_name (*arg_name*)

get_field_type (*field_name, field_value, base_field_type*)

Get field type.

Returns

has_fields = False

prefix = None

search_fields

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

13.12.1.1.4 Module contents

13.12.1.2 graphene_elastic.types package

13.12.1.2.1 Submodules

13.12.1.2.2 graphene_elastic.types.elastic_types module

`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*,
reg-
istry)

class `graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions` (*class_type*)

Bases: `graphene.types.objecttype.ObjectTypeOptions`

connection = None

document = None

filter_fields = ()

registry = None

search_fields = ()

search_nested_fields = ()

```
class graphene_elastic.types.elastic_types.ElasticsearchObjectType(*args,
                                                                    **kwargs)
    Bases: graphene.types.objecttype.ObjectType
    classmethod get_node(info, id)
    id
    classmethod is_type_of(root, info)
    classmethod rescan_fields()
        Attempts to rescan fields and will insert any not converted initially.
    resolve_id(info)
```

13.12.1.2.3 graphene_elastic.types.json_string module

```
class graphene_elastic.types.json_string.JSONString(*args, **kwargs)
    Bases: graphene.types.json.JSONString
    static serialize(dt)
graphene_elastic.types.json_string.to_serializable(o)
```

13.12.1.2.4 Module contents

13.12.2 Submodules

13.12.3 graphene_elastic.advanced_types module

```
class graphene_elastic.advanced_types.FileFieldType(*args, **kwargs)
    Bases: graphene.types.objecttype.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: graphene_elastic.advanced_types._CoordinatesTypeField
    coordinates = <graphene.types.structures.List object>

class graphene_elastic.advanced_types.PointFieldType(*args, **kwargs)
    Bases: graphene_elastic.advanced_types._CoordinatesTypeField
    coordinates = <graphene.types.structures.List object>
```

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

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

13.12.4 graphene_elastic.constants module

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

13.12.5 graphene_elastic.converter module

```
exception graphene_elastic.converter.ElasticsearchConversionError
    Bases: Exception

graphene_elastic.converter.convert_elasticsearch_field(field, registry=None)
graphene_elastic.converter.convert_field_to_boolean(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)
```

13.12.6 graphene_elastic.enums module

```
class graphene_elastic.enums.NoValue
    Bases: enum.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')
```

Prepare list values for creating an Enum.

Example:

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

13.12.7 graphene_elastic.fields module

```
class graphene_elastic.fields.ElasticsearchConnectionField(type, *args,
                                                         **kwargs)
    Bases: graphene.relay.connection.IterableConnectionField

    args
    chained_resolver (resolver, is_partial, root, info, **args)
    classmethod connection_resolver (resolver, connection_type, root, info, **args)
    default_filter_backends
    default_resolver (_root, info, **args)
    doc_type
    document
    field_args
    fields
    filter_args_mapping
    filter_backends
    filter_fields
    get_manager ()
    get_queryset (document, info, **args)
    get_resolver (parent_resolver)
    node_type
    ordering_args_mapping
    ordering_defaults
    ordering_fields
    reference_args
    registry
    classmethod resolve_connection (connection_type, args, resolved)
    search_args_mapping
    search_fields
    type
```

13.12.8 graphene_elastic.helpers module

13.12.9 graphene_elastic.registry module

```
class graphene_elastic.registry.Registry
    Bases: object

    get_converted_field (field)

    get_type_for_document (document)

    register (cls)

    register_converted_field (field, converted)

graphene_elastic.registry.get_global_registry()
graphene_elastic.registry.reset_global_registry()
```

13.12.10 graphene_elastic.settings module

13.12.11 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.mongengine.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)
```

13.12.12 Module contents

```
class graphene_elastic.ElasticsearchConnectionField (type, *args, **kwargs)
    Bases: graphene.relay.connection.IterableConnectionField

    args

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

    classmethod connection_resolver (resolver, connection_type, root, info, **args)

    default_filter_backends

    default_resolver (_root, info, **args)

    doc_type

    document

    field_args

    fields

    filter_args_mapping
```



```
filter_backends
filter_fields
get_manager()
get_queryset(document, info, **args)
get_resolver(parent_resolver)
node_type
ordering_args_mapping
ordering_defaults
ordering_fields
reference_args
registry
classmethod resolve_connection(connection_type, args, resolved)
search_args_mapping
search_fields
type
class graphene_elastic.ElasticsearchObjectType(*args, **kwargs)
    Bases: graphene.types.objecttype.ObjectType
    classmethod get_node(info, id)
    id
    classmethod is_type_of(root, info)
    classmethod rescan_fields()
        Attempts to rescan fields and will insert any not converted initially.
    resolve_id(info)
```


CHAPTER 14

Indices and tables

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

g

- `graphene_elastic`, 74
- `graphene_elastic.advanced_types`, 71
- `graphene_elastic.constants`, 72
- `graphene_elastic.converter`, 72
- `graphene_elastic.enums`, 72
- `graphene_elastic.fields`, 73
- `graphene_elastic.filter_backends`, 70
 - `graphene_elastic.filter_backends.base`, 68
 - `graphene_elastic.filter_backends.filtering`, 65
 - `graphene_elastic.filter_backends.filtering.common`, 53
 - `graphene_elastic.filter_backends.search`, 68
 - `graphene_elastic.filter_backends.search.common`, 65
- `graphene_elastic.helpers`, 74
- `graphene_elastic.registry`, 74
- `graphene_elastic.settings`, 74
- `graphene_elastic.types`, 71
 - `graphene_elastic.types.elastic_types`, 70
 - `graphene_elastic.types.json_string`, 71
- `graphene_elastic.utils`, 74

A

add_arg_prefix() (graphene_elastic.filter_backends.base.BaseBackend class method), 60
 apply_filter() (graphene_elastic.filter_backends.base.BaseBackend class method), 60
 apply_filter_prefix() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 53
 apply_filter_range() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 54
 apply_filter_term() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 55
 apply_filter_terms() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 55
 apply_query() (graphene_elastic.filter_backends.base.BaseBackend class method), 68
 apply_query_contains() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 56
 apply_query_endswith() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 56
 apply_query_exclude() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 57
 apply_query_exists() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 58
 apply_query_gt() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 58
 apply_query_gte() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 59
 apply_query_in() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 59
 apply_query_isnull() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 60
 apply_query_lt() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 60
 apply_query_lte() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 61
 apply_query_wildcard() (graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method), 62
 arg_belongs_to() (graphene_elastic.filter_backends.base.BaseBackend class method), 68
 args (graphene_elastic.ElasticsearchConnectionField attribute), 74
 args (graphene_elastic.fields.ElasticsearchConnectionField attribute), 73

B

BaseBackend (class in graphene_elastic.filter_backends.base), 68

C

chained_resolver() (graphene_elastic.ElasticsearchConnectionField method), 74
 chained_resolver() (graphene_elastic.fields.ElasticsearchConnectionField method), 73
 chunk_size (graphene_elastic.advanced_types.FileFieldType attribute), 71
 connection (graphene_elastic.types.elastic_types.ElasticsearchObjectType attribute), 70
 connection_resolver() (graphene_elastic.ElasticsearchConnectionField class method), 74
 connection_resolver() (graphene_elastic.fields.ElasticsearchConnectionField class method), 73
 construct_fields() (in module graphene_elastic.types.elastic_types), 70

`construct_search()`
 (`graphene_elastic.filter_backends.search.common.SearchFilterBackend`), 66
`construct_self_referenced_fields()` (in module `graphene_elastic.types.elastic_types`), 70
`content_type` (`graphene_elastic.advanced_types.FileFieldType` attribute), 71
`convert_elasticsearch_field()` (in module `graphene_elastic.converter`), 72
`convert_field_to_boolean()` (in module `graphene_elastic.converter`), 72
`convert_field_to_datetime()` (in module `graphene_elastic.converter`), 72
`convert_field_to_float()` (in module `graphene_elastic.converter`), 72
`convert_field_to_int()` (in module `graphene_elastic.converter`), 72
`convert_field_to_jsonstring()` (in module `graphene_elastic.converter`), 72
`convert_field_to_string()` (in module `graphene_elastic.converter`), 72
`convert_list_to_enum()` (in module `graphene_elastic.enums`), 72
`coordinates` (`graphene_elastic.advanced_types.MultiPolygonFieldType` attribute), 71
`coordinates` (`graphene_elastic.advanced_types.PointFieldType` attribute), 71
`coordinates` (`graphene_elastic.advanced_types.PolygonFieldType` attribute), 72
D
`data` (`graphene_elastic.advanced_types.FileFieldType` attribute), 71
`default_filter_backends`
 (`graphene_elastic.ElasticsearchConnectionField` attribute), 74
`default_filter_backends`
 (`graphene_elastic.fields.ElasticsearchConnectionField` attribute), 73
`default_resolver()`
 (`graphene_elastic.ElasticsearchConnectionField` method), 74
`default_resolver()`
 (`graphene_elastic.fields.ElasticsearchConnectionField` method), 73
`doc_type` (`graphene_elastic.ElasticsearchConnectionField` attribute), 74
`doc_type` (`graphene_elastic.fields.ElasticsearchConnectionField` attribute), 73
`doc_type` (`graphene_elastic.filter_backends.base.BaseBackend` attribute), 68
`document` (`graphene_elastic.ElasticsearchConnectionField` attribute), 74
`document` (`graphene_elastic.fields.ElasticsearchConnectionField` attribute), 73
`document` (`graphene_elastic.types.elastic_types.ElasticsearchObjectType` attribute), 70
E
`ElasticsearchConnectionField` (class in `graphene_elastic`), 74
`ElasticsearchConnectionField` (class in `graphene_elastic.fields`), 73
`ElasticsearchConversionError`, 72
`ElasticsearchObjectType` (class in `graphene_elastic`), 75
`ElasticsearchObjectType` (class in `graphene_elastic.types.elastic_types`), 70
`ElasticsearchObjectTypeOptions` (class in `graphene_elastic.types.elastic_types`), 70
F
`field_args` (`graphene_elastic.ElasticsearchConnectionField` attribute), 74
`field_args` (`graphene_elastic.fields.ElasticsearchConnectionField` attribute), 73
`field_belongs_to()`
 (`graphene_elastic.filter_backends.base.BaseBackend` method), 68
`field_belongs_to()`
 (`graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend` method), 62
`field_belongs_to()`
 (`graphene_elastic.filter_backends.search.common.SearchFilterBackend` method), 67
`fields` (`graphene_elastic.ElasticsearchConnectionField` attribute), 74
`fields` (`graphene_elastic.fields.ElasticsearchConnectionField` attribute), 73
`FileFieldType` (class in `graphene_elastic.advanced_types`), 71
`filter()` (`graphene_elastic.filter_backends.base.BaseBackend` method), 69
`filter()` (`graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend` method), 62
`filter()` (`graphene_elastic.filter_backends.search.common.SearchFilterBackend` method), 67
`filter_args_mapping`
 (`graphene_elastic.ElasticsearchConnectionField` attribute), 74
`filter_args_mapping`
 (`graphene_elastic.fields.ElasticsearchConnectionField` attribute), 73
`filter_backends` (`graphene_elastic.ElasticsearchConnectionField` attribute), 74
`filter_backends` (`graphene_elastic.fields.ElasticsearchConnectionField` attribute), 73

[filter_fields\(graphene_elastic.ElasticsearchConnectionField class method\)](#), 63
[attribute](#)), 75
[filter_fields\(graphene_elastic.fields.ElasticsearchConnectionField class method\)](#), 75
[attribute](#)), 73
[filter_fields\(graphene_elastic.filter_backends.base.BaseBackend class method\)](#), 73
[attribute](#)), 69
[filter_fields\(graphene_elastic.types.elastic_types.ElasticsearchObjectType class method\)](#), 75
[attribute](#)), 70
[FilteringFilterBackend \(class in graphene_elastic.filter_backends.filtering.common\)](#), 71
[53](#)
G
[generic_fields\(\)\(graphene_elastic.filter_backends.base.BaseBackend class method\)](#), 69
[get_all_query_params\(\)\(graphene_elastic.filter_backends.search.common.SearchFilterBackend class method\)](#), 67
[get_backend_default_fields_params\(\)\(graphene_elastic.filter_backends.base.BaseBackend class method\)](#), 69
[get_backend_default_fields_params\(\)\(graphene_elastic.filter_backends.search.common.SearchFilterBackend class method\)](#), 67
[get_backend_fields\(\)\(graphene_elastic.filter_backends.base.BaseBackend class method\)](#), 69
[get_converted_field\(\)\(graphene_elastic.registry.Registry method\)](#), 74
[get_document_fields\(\)\(in graphene_elastic.utils\)](#), 74
[get_field_description\(\)\(in graphene_elastic.utils\)](#), 74
[get_field_lookup_param\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 62
[get_field_name\(\)\(graphene_elastic.filter_backends.base.BaseBackend class method\)](#), 69
[get_field_options\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 62
[get_field_type\(\)\(graphene_elastic.filter_backends.base.BaseBackend class method\)](#), 69
[get_field_type\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 62
[get_field_type\(\)\(graphene_elastic.filter_backends.search.common.SearchFilterBackend class method\)](#), 67
[get_filter_query_params\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 62
[get_global_registry\(\)\(in graphene_elastic.registry\)](#), 74
[get_gte_lte_params\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 62
[get_manager\(\)\(graphene_elastic.ElasticsearchConnectionField class method\)](#), 75
[get_manager\(\)\(graphene_elastic.fields.ElasticsearchConnectionField class method\)](#), 73
[get_node\(\)\(graphene_elastic.ElasticsearchObjectType class method\)](#), 75
[get_node\(\)\(graphene_elastic.types.elastic_types.ElasticsearchObjectType class method\)](#), 75
[get_node_from_global_id\(\)\(in graphene_elastic.utils\)](#), 74
[get_queryset\(\)\(graphene_elastic.ElasticsearchConnectionField class method\)](#), 75
[get_range_params\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 64
[get_resolver\(\)\(graphene_elastic.ElasticsearchConnectionField class method\)](#), 75
[get_resolver\(\)\(graphene_elastic.fields.ElasticsearchConnectionField class method\)](#), 73
[get_resolver\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 64
[get_resolver\(\)\(graphene_elastic.ElasticsearchConnectionField class method\)](#), 75
[get_resolver\(\)\(graphene_elastic.fields.ElasticsearchConnectionField class method\)](#), 73
[get_resolver\(\)\(graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend class method\)](#), 64
[get_type_for_document\(\)\(in graphene_elastic.utils\)](#), 74
[graphene_elastic \(module\)](#), 74
[graphene_elastic.advanced_types \(module\)](#), 71
[graphene_elastic.constants \(module\)](#), 72
[graphene_elastic.converter \(module\)](#), 72
[graphene_elastic.enums \(module\)](#), 72
[graphene_elastic.fields \(module\)](#), 73
[graphene_elastic.filter_backends \(module\)](#), 73
[graphene_elastic.filter_backends.base \(module\)](#), 68
[graphene_elastic.filter_backends.filtering \(module\)](#), 65
[graphene_elastic.filter_backends.filtering.common \(module\)](#), 53
[graphene_elastic.filter_backends.search \(module\)](#), 68
[graphene_elastic.filter_backends.filtering.common.FilteringFilterBackend \(module\)](#), 65
[graphene_elastic.filter_backends.search.common.SearchFilterBackend \(module\)](#), 74
[graphene_elastic.registry \(module\)](#), 74
[graphene_elastic.settings \(module\)](#), 74
[graphene_elastic.types.elastic_types \(module\)](#), 71
[graphene_elastic.types.json_string \(module\)](#), 71
[graphene_elastic.utils \(module\)](#), 74

H

<code>has_fields(graphene_elastic.filter_backends.base.BaseBackend</code>	<code>(graphene_elastic.fields.ElasticsearchConnectionField</code>
<code>attribute), 69</code>	<code>attribute), 73</code>
<code>has_fields(graphene_elastic.filter_backends.filtering.ordering_fields(graphene_elastic.ElasticsearchConnectionField</code>	<code>attribute), 75</code>
<code>attribute), 64</code>	
<code>has_fields(graphene_elastic.filter_backends.search.common.SearchFilterBackend</code>	<code>(graphene_elastic.fields.ElasticsearchConnectionField</code>
<code>attribute), 67</code>	<code>attribute), 73</code>

1

id (graphene_elastic.ElasticsearchObjectType attribute), 75	PointFieldType (class graphene_elastic.advanced_types), 71	in
id (graphene_elastic.types.elastic_types.ElasticsearchObjectType attribute), 71	PolygonFieldType (class graphene_elastic.advanced_types), 71	in
import_single_dispatch() (in module graphene_elastic.utils), 74	prefix (graphene_elastic.filter_backends.base.BaseBackend attribute), 69	
is_type_of() (graphene_elastic.ElasticsearchObjectType class method), 75	prefix (graphene_elastic.filter_backends.filtering.common.FilteringFilter attribute), 64	
is_type_of() (graphene_elastic.types.elastic_types.ElasticsearchObjectType class method), 71	prefix (graphene_elastic.filter_backends.search.common.SearchFilterBa attribute), 67	
is_valid_elasticsearch_document() (in module graphene_elastic.utils), 74	prepare_filter_fields() (graphene_elastic.filter_backends.filtering.common.FilteringFilter attribute), 64	

J

`JSONString` (class `graphene_elastic.types.json_string`), 71 in (method), 65

L

length(*graphene_elastic.advanced_types.FileFieldType*
attribute), 71

M

md5 (*graphene_elastic.advanced_types.FileFieldType attribute*), 71

MultiPolygonFieldType (class in *graphene_elastic.advanced_types*), 71

reference_args (*graphene_elastic.fields.ElasticsearchConnectionField attribute*), 73

register () (*graphene_elastic.registry.Registry method*), 74

N

<code>node_type(graphene_elastic.ElasticsearchConnectionFactory</code>	<code>registry(class in graphene_elastic.registry), 74</code>
<code>attribute), 75</code>	<code>registry(graphene_elastic.ElasticsearchConnectionFactory</code>
<code>node_type(graphene_elastic.fields.ElasticsearchConnectionFactory</code>	<code>attribute), 75</code>
<code>attribute), 73</code>	<code>registry(graphene_elastic.fields.ElasticsearchConnectionFactory</code>
<code>NoValue(class in graphene_elastic.enums), 72</code>	<code>attribute) 73</code>

O

ordering_args_mapping	rescan_fields()	(<i>graphene_elastic.ElasticsearchObjectType</i>
(<i>graphene_elastic.ElasticsearchConnectionField</i>	<i>class method</i>), 75	
attribute), 75	rescan_fields()	(<i>graphene_elastic.types.elastic_types.ElasticsearchC</i>
ordering_args_mapping	<i>class method</i>), 71	
(<i>graphene_elastic.fields.ElasticsearchConnectionField</i>	reset_global_registry()	(in module
attribute), 73	<i>graphene_elastic.registry</i>), 74	
ordering_defaults	resolve_chunk_size()	
(<i>graphene_elastic.ElasticsearchConnectionField</i>	(<i>graphene_elastic.advanced_types.FileFieldType</i>	
attribute), 75	<i>method</i>), 71	

P

```

PointFieldType                (class          in
    graphene_elastic.advanced_types), 71
PolygonFieldType              (class          in
    graphene_elastic.advanced_types), 71
prefix(graphene_elastic.filter_backends.base.BaseBackend
    attribute), 69
prefix(graphene_elastic.filter_backends.filtering.common.FilteringFilter
    attribute), 64
prefix(graphene_elastic.filter_backends.search.common.SearchFilterBa
    attribute), 67
prepare_filter_fields()
    (graphene_elastic.filter_backends.filtering.common.FilteringFilter
    method), 64
prepare_query_params()
    (graphene_elastic.filter_backends.filtering.common.FilteringFilter
    method), 65
prepare_search_fields()
    (graphene_elastic.filter_backends.search.common.SearchFilterBa
    method), 67

```

R

```

reference_args(graphene_elastic.ElasticsearchConnectionField
               attribute), 75
reference_args(graphene_elastic.fields.ElasticsearchConnectionField
               attribute), 73
register() (graphene_elastic.registry.Registry
            method), 74
register_converted_field() (graphene_elastic.registry.Registry method), 74
registry(class in graphene_elastic.registry), 74
registry(graphene_elastic.ElasticsearchConnectionField
         attribute), 75
registry(graphene_elastic.fields.ElasticsearchConnectionField
         attribute), 73
registry(graphene_elastic.types.elastic_types.ElasticsearchObjectType
         attribute), 70
rescan_fields() (graphene_elastic.ElasticsearchObjectType
                 class method), 75
rescan_fields() (graphene_elastic.types.elastic_types.ElasticsearchO
                 class method), 71
register_global_registry() (in module
                             graphene_elastic.registry), 74
resolve_chunk_size() (graphene_elastic.advanced_types.FileFieldType
                      method), 71

```

```

resolve_connection() class method), 70
    (graphene_elastic.ElasticsearchConnectionField
    class method), 75
T
resolve_connection() to_serializable() (in module
    (graphene_elastic.fields.ElasticsearchConnectionField
    class method), 73 graphene_elastic.types.json_string), 71
    type (graphene_elastic.ElasticsearchConnectionField
    attribute), 75
resolve_content_type()
    (graphene_elastic.advanced_types.FileFieldType type (graphene_elastic.fields.ElasticsearchConnectionField
    method), 71 attribute), 73
resolve_data() (graphene_elastic.advanced_types.FileFieldType
    method), 71
resolve_id() (graphene_elastic.ElasticsearchObjectType
    method), 75
resolve_id() (graphene_elastic.types.elastic_types.ElasticsearchObjectType
    method), 71
resolve_length() (graphene_elastic.advanced_types.FileFieldType
    method), 71
resolve_md5() (graphene_elastic.advanced_types.FileFieldType
    method), 71

```

S

```

search_args_mapping
    (graphene_elastic.ElasticsearchConnectionField
    attribute), 75
search_args_mapping
    (graphene_elastic.fields.ElasticsearchConnectionField
    attribute), 73
search_fields(graphene_elastic.ElasticsearchConnectionField
    attribute), 75
search_fields(graphene_elastic.fields.ElasticsearchConnectionField
    attribute), 73
search_fields(graphene_elastic.filter_backends.base.BaseBackend
    attribute), 69
search_fields(graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions
    attribute), 70
search_nested_fields
    (graphene_elastic.types.elastic_types.ElasticsearchObjectTypeOptions
    attribute), 70
SearchFilterBackend (class in
    graphene_elastic.filter_backends.search.common),
    65
serialize() (graphene_elastic.types.json_string.JSONString
    static method), 71
split_lookup_complex_multiple_value()
    (graphene_elastic.filter_backends.base.BaseBackend
    class method), 69
split_lookup_complex_value()
    (graphene_elastic.filter_backends.base.BaseBackend
    class method), 69
split_lookup_filter()
    (graphene_elastic.filter_backends.base.BaseBackend
    class method), 70
split_lookup_name()
    (graphene_elastic.filter_backends.base.BaseBackend

```