graphene-elastic Documentation

Release 0.5

Artur Barseghyan <artur.barseghyan@gmail.com>

Sep 28, 2019

Contents

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.*

CHAPTER 2

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.

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 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
```

CHAPTER 5

Examples

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,
    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
        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
```

(continues on next page)

(continued from previous page)

```
        # 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},
        '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',
```

(continues on next page)

(continued from previous page)

```
            # 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)

# 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"}}) {
```

(continues on next page)

(continued from previous page)

```
    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"},
        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
```

(continues on next page)

(continued from previous page)

```
        title
        content
      }
    }
  }
}
```

5.4.3.3 Ordering

Possible choices are `ASC` and `DESC`.

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

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
      }
    }
  }
}
```

5.4.3.5 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
    }
  }
}
```

CHAPTER 6

Road-map

Road-map and development plans.

This package was designed after django-elasticsearch-dsl-drf. It's intended to offer similar functionality in `graphene-elastic` (this package).

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.
- Complex search backends, such as Simple query search.

Stay tuned or reach out if you want to help.

CHAPTER 7

Testing

Project is covered with tests.

7.1 Running tests

By defaults 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 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
```

7.2 Testing with Docker

```
docker-compose -f docker-compose.yml -f docker-compose-test.yml up --build test
```

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`.

CHAPTER 9

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>

CHAPTER 13

Project documentation

Contents:

Table of Contents

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
```

(continues on next page)

(continued from previous page)

```
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},
        }
```

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
```

(continues on next page)

(continued from previous page)

```
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`).
            '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
```

(continues on next page)

(continued from previous page)

```
            # (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)
```

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 Run the test server

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

13.2.6 Open the `graphiql` client the browser

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

13.2.7 Make some experiments

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

13.2.8 Run tests

```
./runtests.py
```

13.3 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
      }
    }
  }
}
```

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
      }
    }
  }
}
```

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
```

(continues on next page)

(continued from previous page)

```
        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
      }
    }
```

(continues on next page)

(continued from previous page)

```
  }
}
```

13.4.1.9 Filter lookup `lte`

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

(continues on next page)

(continued from previous page)

```
  }
}
```

13.5 Post-filter backend

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

13.5.1 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`

13.5.1.1 Filter lookup `contains`

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

13.5.1.2 Filter lookup `ends_with`

Note: `endsWith` for camelCase.

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

13.5.1.3 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
      }
    }
  }
}
```

13.5.1.4 Filter lookup `exists`

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

13.5.1.5 Filter lookup `gt`

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

13.5.1.6 Filter lookup `gte`

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

13.5.1.7 Filter lookup `in`

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

(continues on next page)

(continued from previous page)

```
        numViews
        tags
      }
    }
  }
}
```

13.5.1.8 Filter lookup `lt`

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

13.5.1.9 Filter lookup `lte`

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

13.5.1.10 Filter lookup `prefix`

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

13.5.1.11 Filter lookup `range`

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

13.5.1.12 Filter lookup `starts_with`

Note: `startsWith` for camelCase.

Alias for filter lookup ''prefix''.

13.5.1.13 Filter lookup `term`

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

13.5.1.14 Filter lookup `terms`

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

(continues on next page)

(continued from previous page)

```
      }
    }
  }
}
```

13.5.1.15 Filter lookup `wildcard`

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

13.6 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
```

(continues on next page)

(continued from previous page)

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

13.7 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,
                    '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 remember.",
            "highlight": {
              "title": [
                "PM <b>decide</b>."
              ],
              "content": [
                "Mind large similar probably lawyer <em>since</em>."
              ]
            }
          },
          "cursor": "YXJyYXljb25uZWN0aW9uOjA="
        },
        {
          "node": {
            "title": "Many add.",
            "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="
        }
    }
  }
}
```

13.8 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
        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,
```

(continues on next page)

(continued from previous page)

```
            "category": null,
            "comments": []
          },
          "cursor": "YXJyYXljb25uZWN0aW9uOjA="
        },
        {
          "node": {
            "id": "UG9zdDpvZkhuUlcwQlhfYXJjd2RMc0w1Nw==",
            "title": "prevent Alice citizen",
            "content": null,
            "category": null,
            "comments": []
          },
          "cursor": "YXJyYXljb25uZWN0aW9uOjE="
        }
      ]
    }
  }
}
```

13.9 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,
            },
```

(continues on next page)

(continued from previous page)

```
            'tags': {
                '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,
          "aggs": {
            "doc_count_error_upper_bound": 0,
            "sum_other_doc_count": 0,
            "buckets": [
```

(continues on next page)

(continued from previous page)


```
            {
              "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
            },
            {
              "key": "Elastic",
              "doc_count": 1
            },
            {
              "key": "Machine Learning",
              "doc_count": 1
            },
```


(continues on next page)

(continued from previous page)

```
                {
                  "key": "MongoDB",
                  "doc_count": 1
                }
              ]
            }
          }
        },
        "edges": [
          {
            "node": {
              "id": "UG9zdDpBVWNwVm0wQklwZ2dXbVlJTndOVA==",
              "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`).

13.10 Pagination

13.10.1 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.

13.10.2 Enforce `first` or `last`

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

13.10.3 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:

13.11 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',
            {}
        ).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):
```

(continues on next page)

(continued from previous page)

```
        """Prepare source fields.

        Possible structures:

            source_fields = ["title"]

        Or:

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

        Or:

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

        :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
```

13.12 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,
```

(continues on next page)

(continued from previous page)

```
    # 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)
)
```

13.13 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).

13.13.1 Docker

13.13.1.1 Project default

```
docker-compose up elasticsearch
```

13.13.1.2 Run specific version

13.13.1.2.1 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
```

13.13.1.2.2 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
```

13.14 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.

13.14.1 Questions and answers

13.15 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",
    json.dumps(DEFAULTS)
)
```

13.16 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.

13.16.1 0.5

2019-09-29

- PostFilter backend.
- Documentation improvements.

13.16.2 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.

13.16.3 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.

13.16.4 0.2

2019-09-18

- Highlight filter backend.

13.16.5 0.1

2019-09-08

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

13.16.6 0.0.13

2019-09-07

- Documentation improvements and fixes.
- Clean up.

13.16.7 0.0.12

2019-09-06

Note: In memory of Erik Slim. RIP.

- More tests.

13.16.8 0.0.11

2019-09-05

- Fixes in search backend.

13.16.9 0.0.10

2019-09-04

- Fixes.
- Clean up.

13.16.10 0.0.9

2019-09-03

- Added pagination.
- Documentation improvements.

13.16.11 0.0.8

2019-09-02

- Tested default ordering backend.
- Documentation improvements.

13.16.12 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.

13.16.13 0.0.6

2019-08-30

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

13.16.14 0.0.5

2019-08-30

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

13.16.15 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.

13.16.16 0.0.3

2019-08-26

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

13.16.17 0.0.2

2019-08-25

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

13.16.18 0.0.1

2019-08-24

- Initial alpha release.

13.17 graphene_elastic package

13.17.1 Subpackages

13.17.1.1 graphene_elastic.filter_backends package

13.17.1.1.1 Subpackages

13.17.1.1.1.1 graphene_elastic.filter_backends.filtering package

13.17.1.1.2 Submodules

13.17.1.1.3 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*, graphene_elastic.filter_backends.filtering.mixins.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.

filter_args_mapping

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 = '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

13.17.1.1.1.4 Module contents

13.17.1.1.1.5 graphene_elastic.filter_backends.search package

13.17.1.1.1.6 Submodules

13.17.1.1.1.7 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*)
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

has_query_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

search_args_mapping

search_fields
Search filter fields.

13.17.1.1.1.8 Module contents

13.17.1.1.2 Submodules

13.17.1.1.3 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

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 Alice 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

13.17.1.1.4 Module contents

13.17.1.2 graphene_elastic.types package

13.17.1.2.1 Submodules

13.17.1.2.2 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*, *registry*)

class graphene_elastic.types.elastic_types.**ElasticsearchObjectTypeOptions**(*class_type*)
Bases: graphene.types.objecttype.ObjectTypeOptions

connection = None

document = None

registry = None

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*)

13.17.1.2.3 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*)

13.17.1.2.4 Module contents

13.17.2 Submodules

13.17.3 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>

13.17.4 graphene_elastic.constants module

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

13.17.5 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*)

13.17.6 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'*, *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'}
```

13.17.7 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*, *connection_field=None*, ***args*)

default_filter_backends

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

doc_type

document

field_args

fields

filter_backends

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

get_resolver(*parent_resolver*)

node_type

reference_args

registry

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

type

13.17.8 graphene_elastic.helpers module

graphene_elastic.helpers.**to_camel_case**(*snake_str: str*) → str

Convert snake_case to camelCase.

Capitalize the first letter of each part except the first one with the *capitalize* method and join all the parts together.

Adapted from this response in StackOverflow http://stackoverflow.com/a/19053800/1072990

Parameters **snake_str** –

Returns

graphene_elastic.helpers.**to_pascal_case**(*snake_str: str*) → str

Convert snake_case to CapWords.

Parameters **snake_str** –

Returns

13.17.9 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**()

13.17.10 graphene_elastic.settings module

13.17.11 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.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*)

13.17.12 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*, *connection_field=None*, ***args*)

default_filter_backends

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

doc_type

document

field_args

fields

filter_backends

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

get_resolver(*parent_resolver*)

node_type

reference_args

registry

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

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

Python Module Index

g

Index

E

F

G

R

S

T