Executing Search Requests Using Elasticsearch Query DSL

Janani Ravi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Introduce the Query DSL that Elasticsearch uses for search queries

Understand the contexts in which search queries run

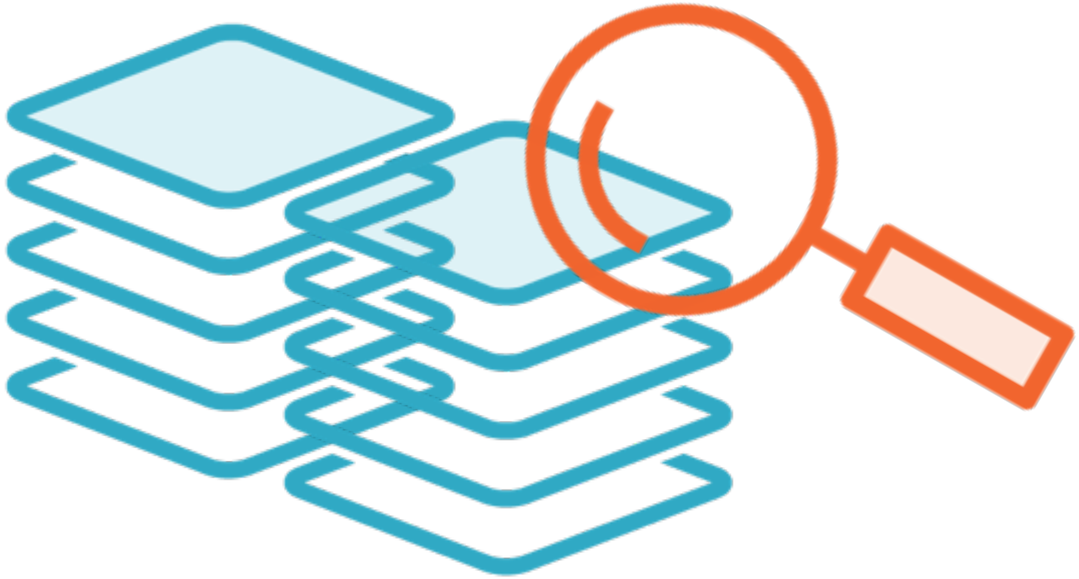
Work with full text searches, term searches, compound searches, filters

Understand how relevance is calculated

Implement all queries using the Elasticsearch REST API

Recap: How Does Search Work?

What Is the Objective of Search?



Find the most relevant documents with your search terms

Most Relevant Document for Search Terms

Know of the document's existence

Index the document for lookup

Know how relevant the document is

Retrieve ranked by relevance

Most Relevant Document for Search Terms

Web crawler

Index the document for lookup

Know how relevant the document is

Retrieve ranked by relevance

Most Relevant Document for Search Terms

Web crawler

Inverted index

Know how relevant the document is

Retrieve ranked by relevance

Most Relevant Document for Search Terms

Web crawler

Inverted index

Scoring

Retrieve ranked by relevance

Most Relevant Document for Search Terms

Web crawler

Inverted index

Scoring

Search

Most Relevant Document for Search Terms

Web crawler

Inverted index

Scoring

Search

Elasticsearch

Specify documents to index

Elasticsearch creates the inverted index behind the scenes

Applies a scoring algorithm for each document for each term

Retrieves documents using a Query DSL

The Query DSL

Query DSL

A flexible, expressive search language that Elasticsearch uses to expose most of the power of Lucene through a simple JSON interface. It is what you should be using to write your queries in production. It makes your queries more flexible, more precise, easier to read, and easier to debug.

https://www.elastic.co/guide/en/elasticsearch/guide/current/query-dsl-intro.html

Query DSL

A flexible, expressive search language that Elasticsearch uses to expose most of the power of Lucene through a simple JSON interface. It is what you should be using to write your queries in production. It makes your queries more flexible, more precise, easier to read, and easier to debug.

https://www.elastic.co/guide/en/elasticsearch/guide/current/query-dsl-intro.html

Query DSL

A flexible, expressive search language that Elasticsearch uses to expose most of the power of Lucene through a simple JSON interface. It is what you should be using to write your queries in production. It makes your queries more flexible, more precise, easier to read, and easier to debug.

https://www.elastic.co/guide/en/elasticsearch/guide/current/query-dsl-intro.html

Query DSL

A flexible, expressive search language that Elasticsearch uses to expose most of the power of Lucene through a simple JSON interface. It is what you should be using to write your queries in production. It makes your queries more flexible, more precise, easier to read, and easier to debug.

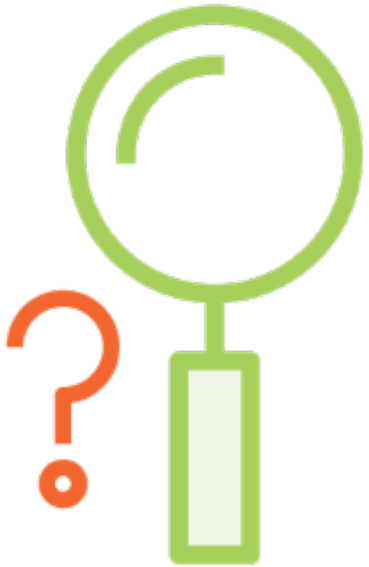
https://www.elastic.co/guide/en/elasticsearch/guide/current/query-dsl-intro.html

Query DSL

A flexible, expressive search language that Elasticsearch uses to expose most of the power of Lucene through a simple JSON interface. It is what you should be using to write your queries in production. It makes your queries more flexible, more precise, easier to read, and easier to debug.

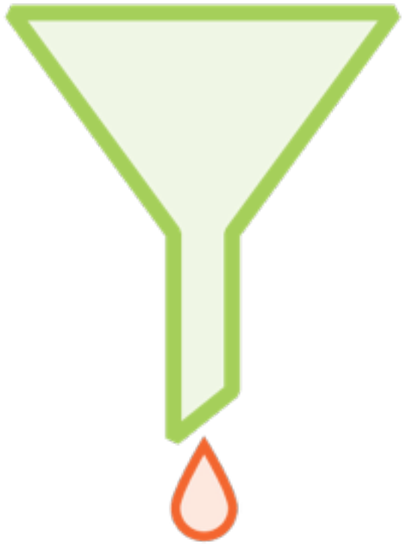
https://www.elastic.co/guide/en/elasticsearch/guide/current/query-dsl-intro.html

Two Contexts of Search



How well does this document match this query?

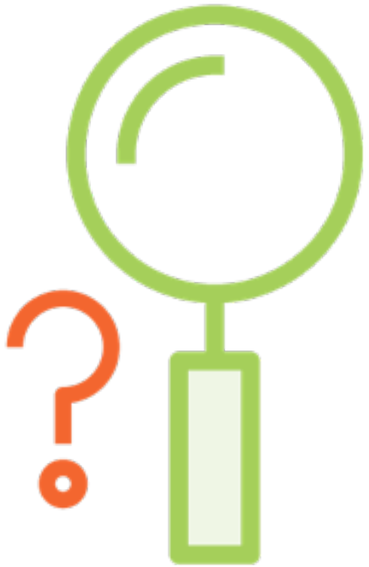
Doesn't have a yes/no answer. Requires a score



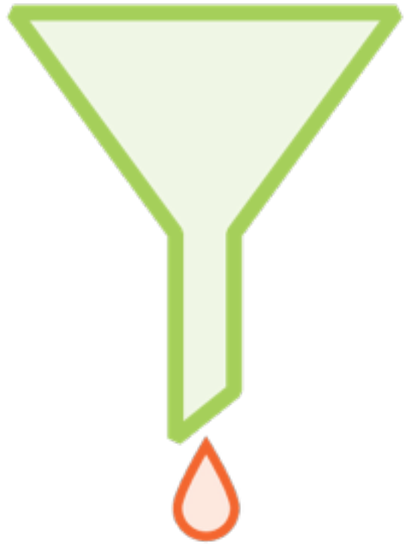
Does this document match this query clause?

Simple yes/no answer

Two Contexts of Search



Query Context



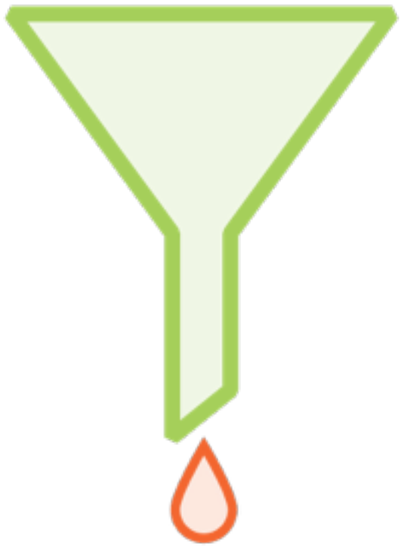
Filter Context

Query Context

Included or not: Determine whether the document should be part of the result

Relevance score: Calculated for every search term the document maps to

High score, more relevant: More relevant documents, higher in the search rankings



Filter Context

Included or not: Yes/no determines whether included in the result

No scoring: No additional relevance ranking in the search results

Structured data: Exact matches, range queries

Faster: Only determine inclusion in results, no scoring to consider

Demo

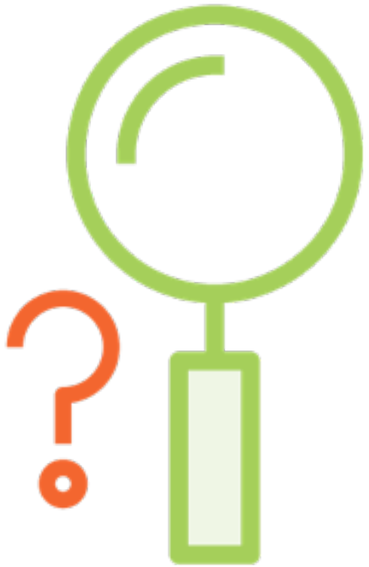
Generate JSON data for a 1000 customers to server as documents for search

www.json-generator.com

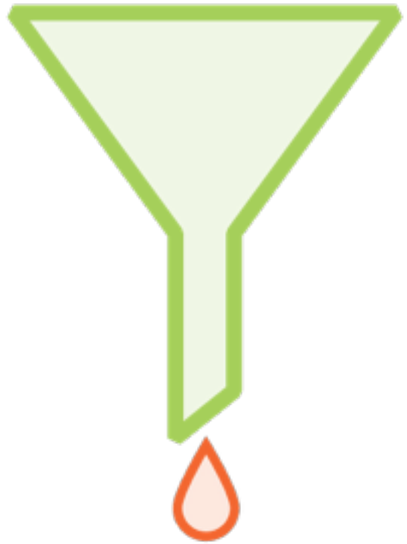
To convert from JSON to ES format:
1. Remove outer array notation - remove “[” and “]” so the JSON objects are not within an array.
2. Regex find and replace “,” between JSON objects with a new line.
3. Add the JSON “index” field {“index” : {}}

The Query Context

Two Contexts of Search

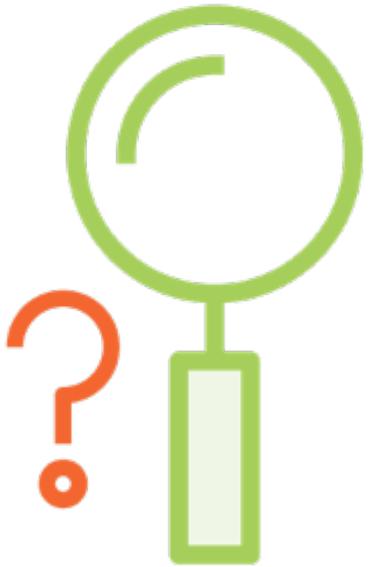


Query Context



Filter Context

Two Contexts of Search



Query Context

Filter Context

Query Terms Specification

Search terms as URL query parameters

Search terms within the URL request body

Query Terms Specification

Search terms as URL query parameters

Search terms within the URL request body

Demo

Search using query parameters

The default number of results returned from ES is 10.

When sorting the result, e.g. age in descending order, there is no relevance score.

The from field indicates from which index we want the results to be returned.

The explain parameter gives you insight as to why the relevance score for a particular document is a certain value. This is good for debugging.

Query Terms Specification

Search terms as URL query parameters

Search terms within the URL request body

Demo

Search using the request body to specify parameters

Use the _search API

Add “match_all” to the query to get all documents. Only 10 will be returned as default. The score of every document will be 1.0 as the relevance score isn’t calculated for match_all.

The ES server is stateless for search queries. This means that ES maintains no open cursor or session for your search. Search results come back in one go (no pagination). For pagination, you use a session ID or some other marker to maker requests to the same session to retrieve search result pages one after the other.

Multiple indices can be searched by specifying both indices within your URL e.g. ‘localhost:9200/customer,products/_search?pretty

You can also search multiple doc types e.g. ‘localhost:9200/products/shoes,laptops/_search?pretty’

Demo

Source filtering to include only those fields that we're interested in

Term searches should have an exact match in the inverted index.

Elasticsearch considers first name matches to be more relevant than last name matches.

To reduce unnecessary data transfer across the network and speed up the searches, you can set source to false so the document contents aren't returned.

You can use regex in for the source parameter to indicate what fields you're interested in e.g. "_source" : "st*" will return source fields starting with 'st'. Multiple regex can be specified by putting them in an array.

Source filtering doesn't affect the relevance ranking of documents.

Within source you can also have includes and excludes parameters.

* is wildcard.

Demo

Full text queries using:

- match — Not an exact term match by itself.
  Can be used with other parameters e.g. operator (OR AND)
  When no operator is specified it defaults to OR
- match_phrase — Entire phrase has to be matched (exact match)
- match_phrase_prefix — Specify the prefix of what you want to look for.
  Like an autocomplete.

Relevance in Elasticsearch

The Meaning of Relevance

The search results answered your question or solved your problem

The user understands easily why the search engine retrieved these results

The Meaning of Relevance

Early engines

If the results contain all the search terms then the query was successful

Web search engines

Initial emphasis on high performance, with huge document sets, moved on to finding the one correct document

Second generation engines

Relevancy score in relation to other documents in the database, better for research queries, rather than lookup

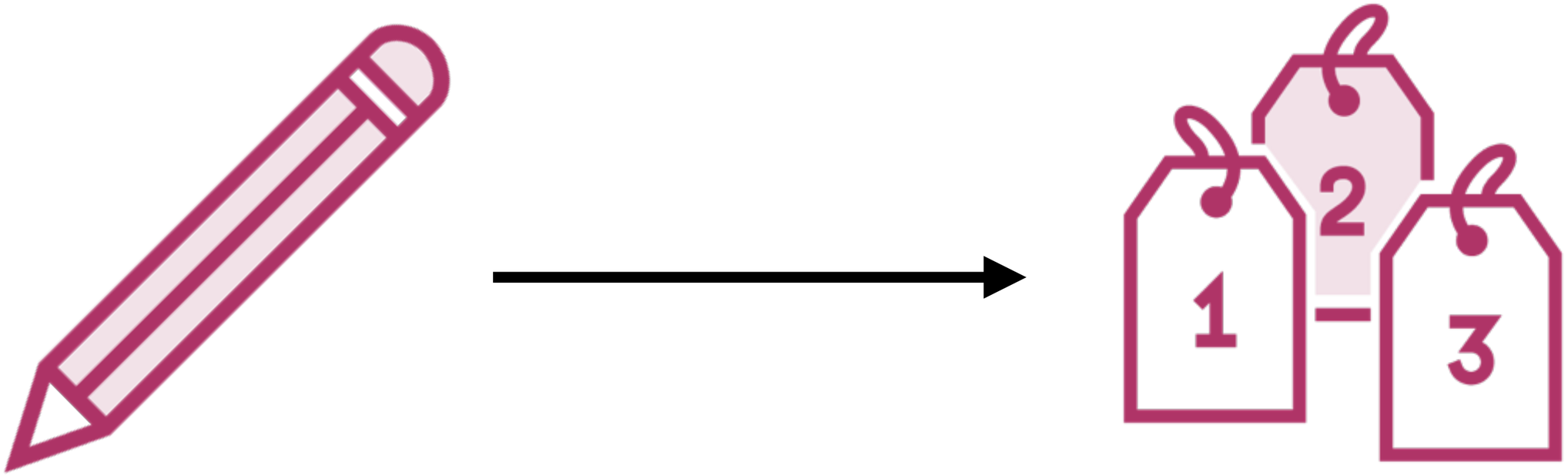
Relevance in Elasticsearch

Represented by the `_score` field in every search result

Relevance in Elasticsearch

Higher the value of `_score`
more relevant the document

Relevance in Elasticsearch

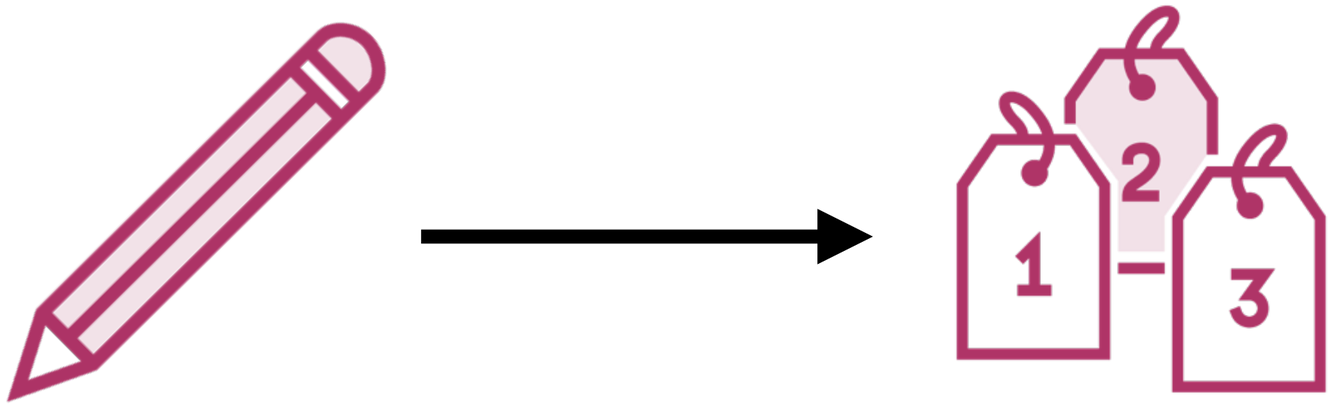


Query clause

Relevance score

Each document has a different relevance score
based on the query clause

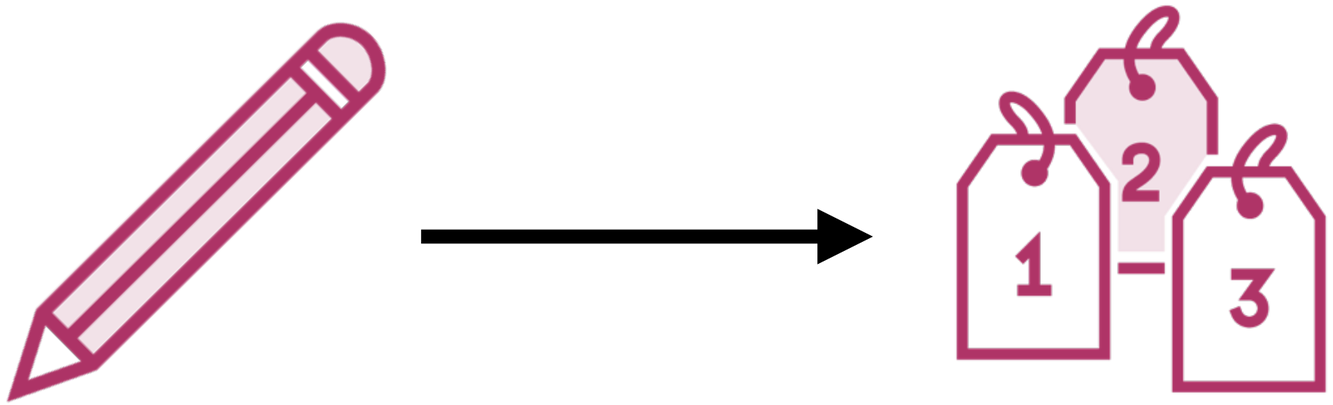
Relevance in Elasticsearch



i.e.non-exact

Fuzzy searches might look at how similar the search term is to the word present in the document

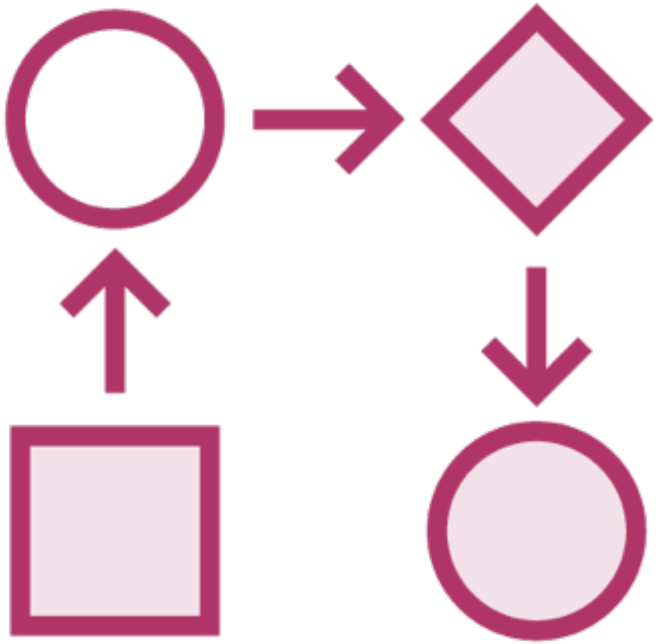
Relevance in Elasticsearch



i.e. exact

Term searches might look at the percentage of search terms that were found in the document

Elasticsearch Relevance Algorithm



TF/IDF

Term Frequency/Inverse Document Frequency

TF/IDF Relevance Algorithm

Term frequency

How often does the term appear in the field?

Inverse document frequency

How often does the term appear in the index?

Field-length norm

How long is the field which was searched?

Term Frequency

Term frequency

How often does the term appear in the field?

More often, more relevant

A field containing 4 mentions of a term is more relevant than one which has just one mention

Inverse Document Frequency

Inverse document frequency

How often does the term appear in the index?

More often, less relevant

If the term is really common across documents in the index, its relevance for a particular document is low

e.g. stopwords such as “the”, “this”

Field-length Norm

Field-length norm

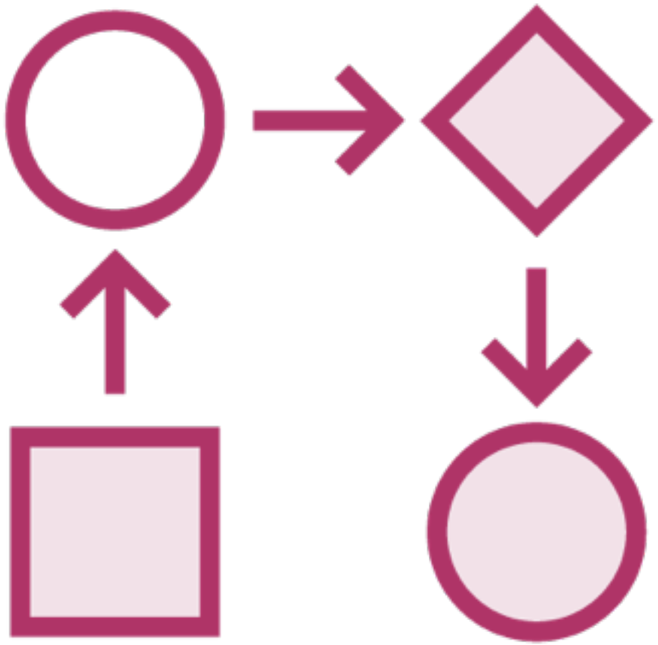
How long is the field which was searched?

Longer fields, less relevant

A term appearing in a longer field is one of a larger set, so less relevant

e.g. words in the title of a book are more relevant than words in the contents

Elasticsearch Relevance Algorithm



The TF/IDF score can be combined with other factors based on the query clause

Relevance in Elasticsearch is calculated using TF/IDF in combination with other factors

The Common Terms Problem

Common Terms

Common terms are called stopwords.

Search for `“The quick brown fox”`

Common Terms

The word "the" is likely to match a huge number of documents

With low relevance to the actual search

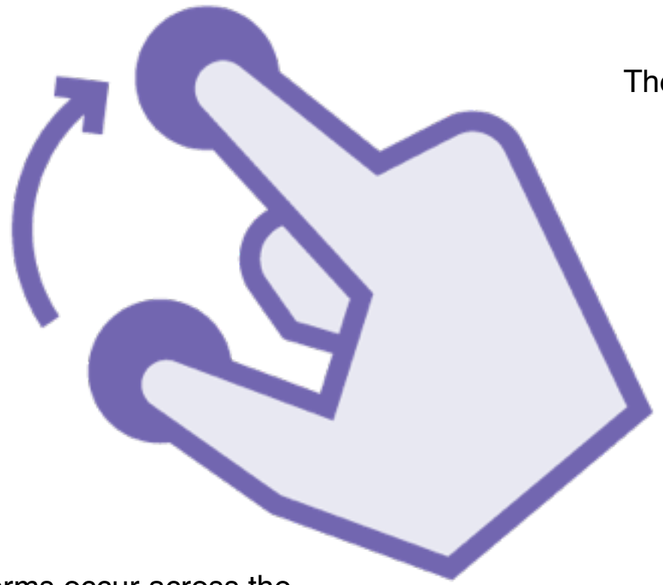
Common Terms

Leaving out stopwords can have unexpected impact

Unable to distinguish between "great" and "not great"

Split Terms: Low and High Frequency

Low: “quick brown fox”



The algorithm automatically adjusts itself based on what documents are indexed.

Frequency of terms is determined by how frequently the terms occur across the indexed documents in ES

High: “the”

Split Terms: Low and High Frequency

Low: “quick brown fox”

Search for documents which have the rarer terms first

High: “the”

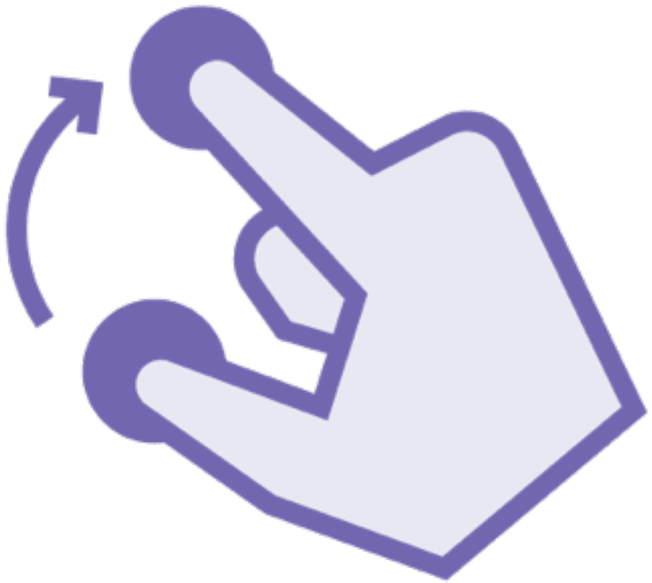
Split Terms: Low and High Frequency

Low: “quick brown fox”

Look for the high frequency terms in the document subset which match the low frequency terms

High: “the”

Split Terms: Low and High Frequency



Improved relevance, good performance

Demo

Common terms queries with cutoff_frequency specified

Use the keyword “common” within the query keyword to indicate that there may be common terms and to specify a cutoff_frequency.

By setting the cutoff frequency of 0.001, terms with a frequency of > 0.1% will be considered common terms.

Compound Queries: The Boolean Query

Boolean Query

Matches documents by combining multiple queries using boolean operators such as AND, OR

Boolean Query

must

The clause must appear in matching documents

should

The clause may appear in matching documents but may not sometimes

must_not

The clause must not appear in the document results

filter

The clause must appear in results but results are not scored

relevance score doesn't matter when you're excluding terms

Boolean Query

must

The clause must appear in matching documents

should

The clause may appear in matching documents but may not sometimes

must_not

The clause must not appear in the document results

filter

The clause must appear in results but results are not scored

Demo

Run boolean compound queries using:

- must
- should
- must_not

Term Queries

Term Query

The exact term needs to be found in the inverted index for indexed documents

Term Query

The terms found in the index may vary based on how you analyze them

There are different analysers you can use

Demo

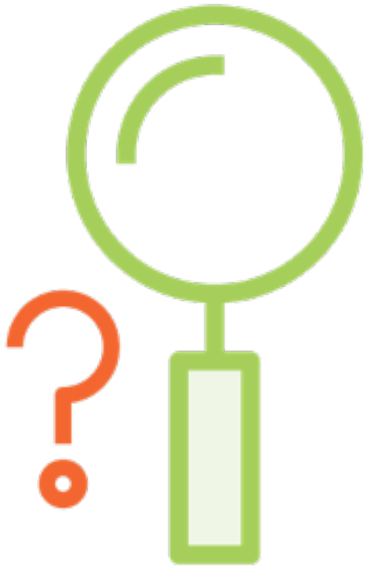
Simple term queries

Boost some terms over others

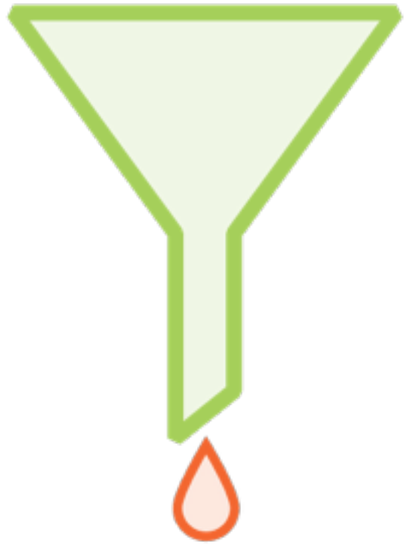
You can add a boost factor to some terms which means you can indicate the importance of one term over another.

Filters

Two Contexts of Search



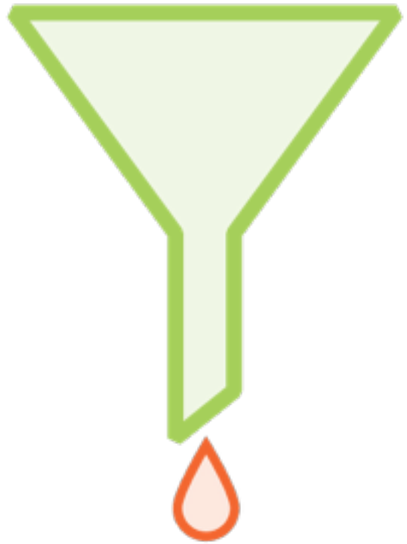
Query Context



Filter Context

Two Contexts of Search

Query Context



Filter Context

Filters

The documents in the result are not scored

Each document responds yes/no to whether it should be included in the result

Demo

Queries to filter documents

Summary

Understood the Query DSL that Elasticsearch uses for search queries

Worked with searches which need relevance and filters where relevance is not required

Worked with full text searches, term searchers, compound searches, filters

Understood the basics of the TF/IDF algorithm for relevance

Implemented all queries using the Elasticsearch REST API