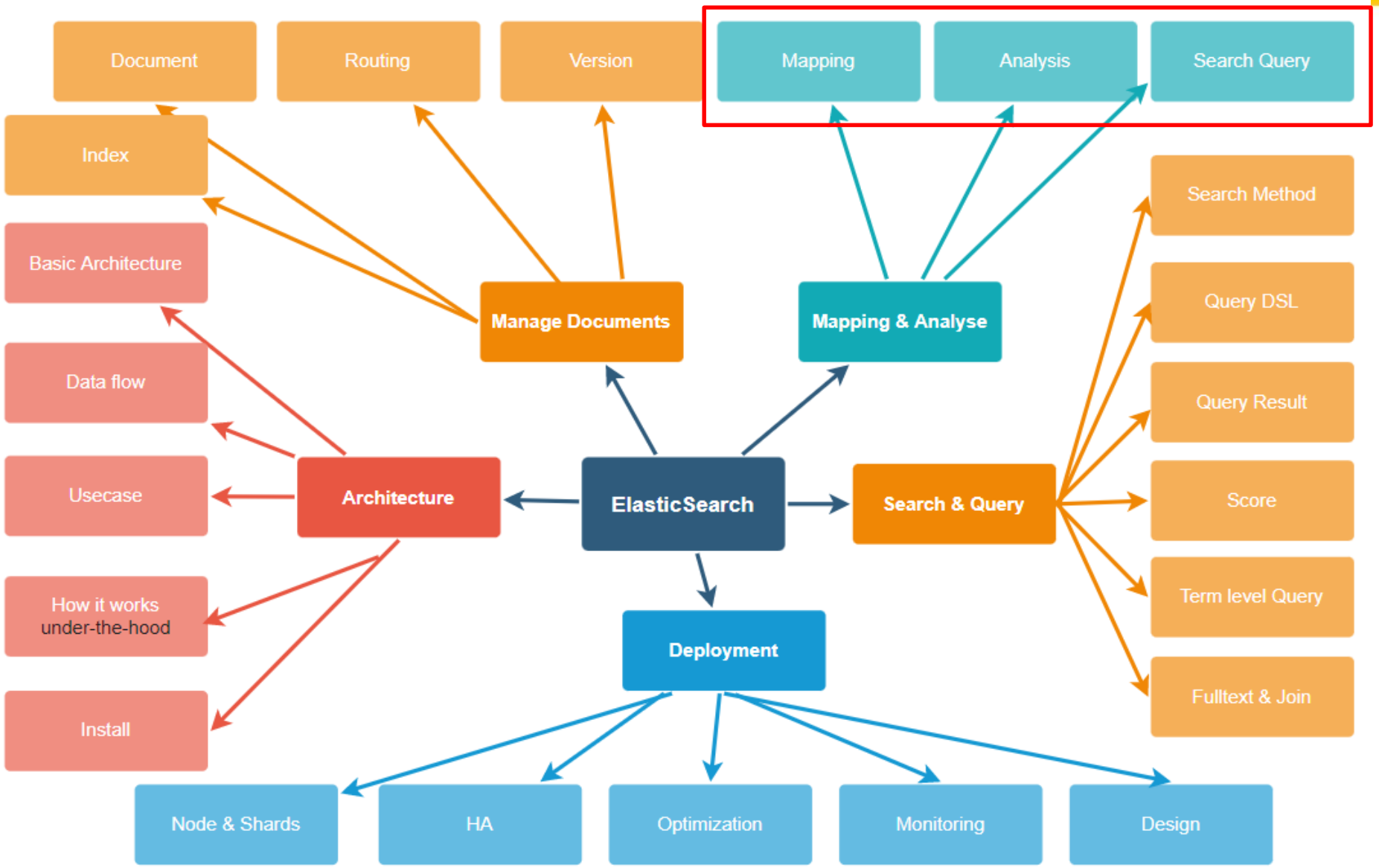
TechMaster
Học là có việc

ElasticSearch

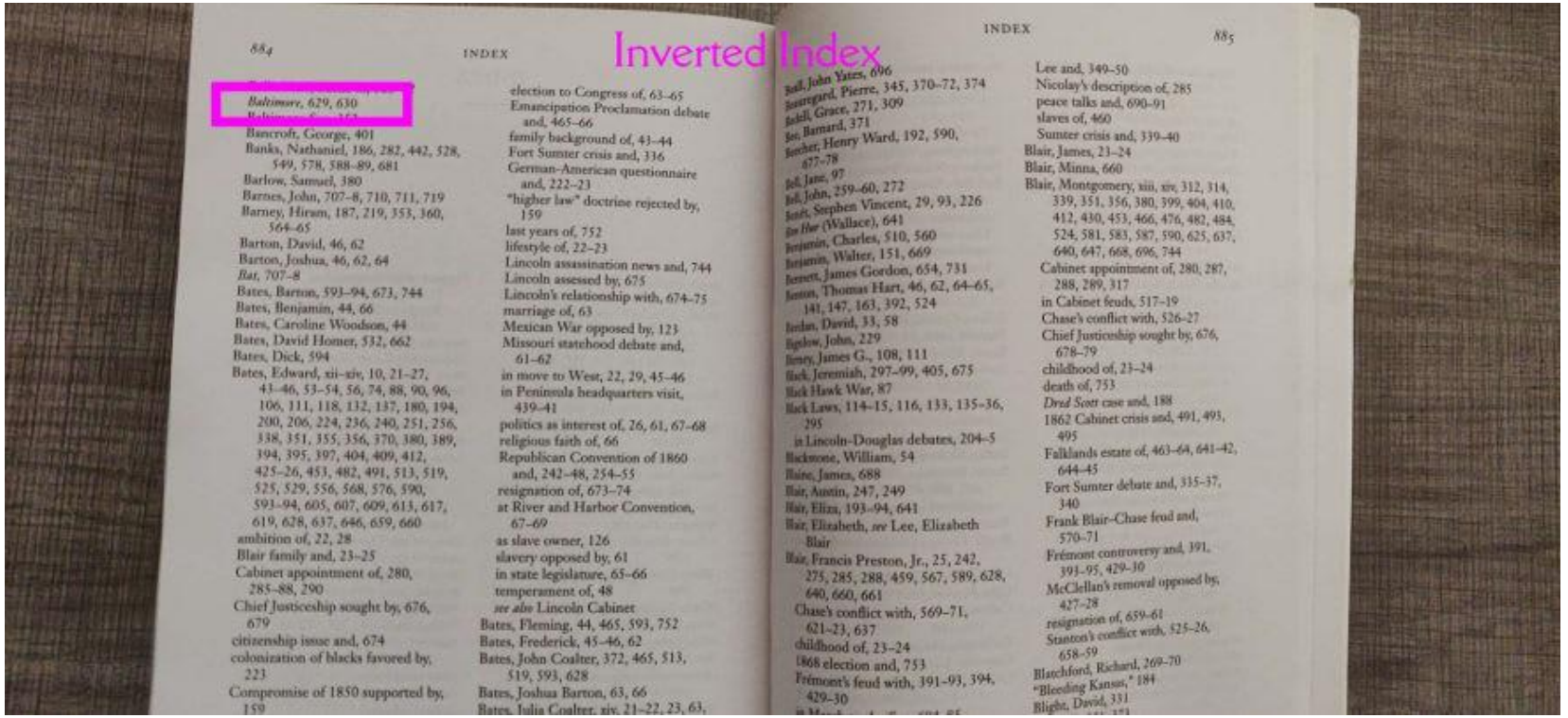
caosao@techmaster

Nội dung


Document
Routing
Version
Mapping
Analysis
Search Query
Index
Basic Architecture
Data flow
Usecase
How it works under-the-hood
Install
Manage Documents
Mapping & Analyse
Architecture
ElasticSearch
Search & Query
Deployment
Search Method
Query DSL
Query Result
Score
Term level Query
Fulltext & Join
Node & Shards
HA
Optimization
Monitoring
Design

Inverted index

- **An inverted index consists of a list of all the unique words that appear in any document, and for each word, a list of the documents in which it appears.**
- **An inverted index consists of all of the unique terms that appear in any document covered by the index.**



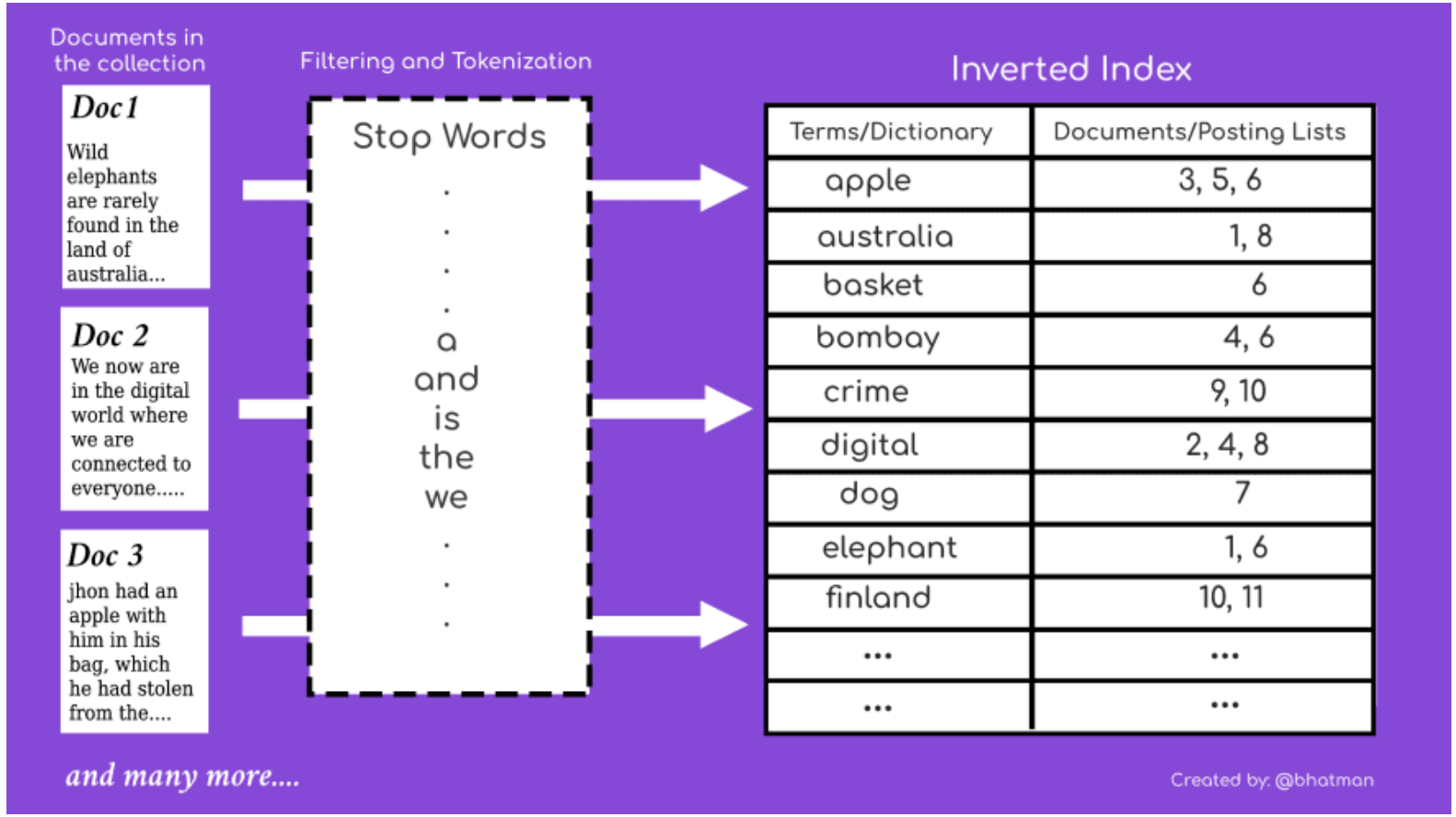
Inverted index

- A field’s values are stored in one of several data structure
  - The data structure depends on the field’s data type
- Ensure efficient data access – eg searches
- Handed by apace lucene, not Elasticsearch

Inverted index

- Mapping between term and which documents contain them
- Outside the context of analyzers, we use the terminology “term”

Sentence => Tokens



Inverted Index

Introduction

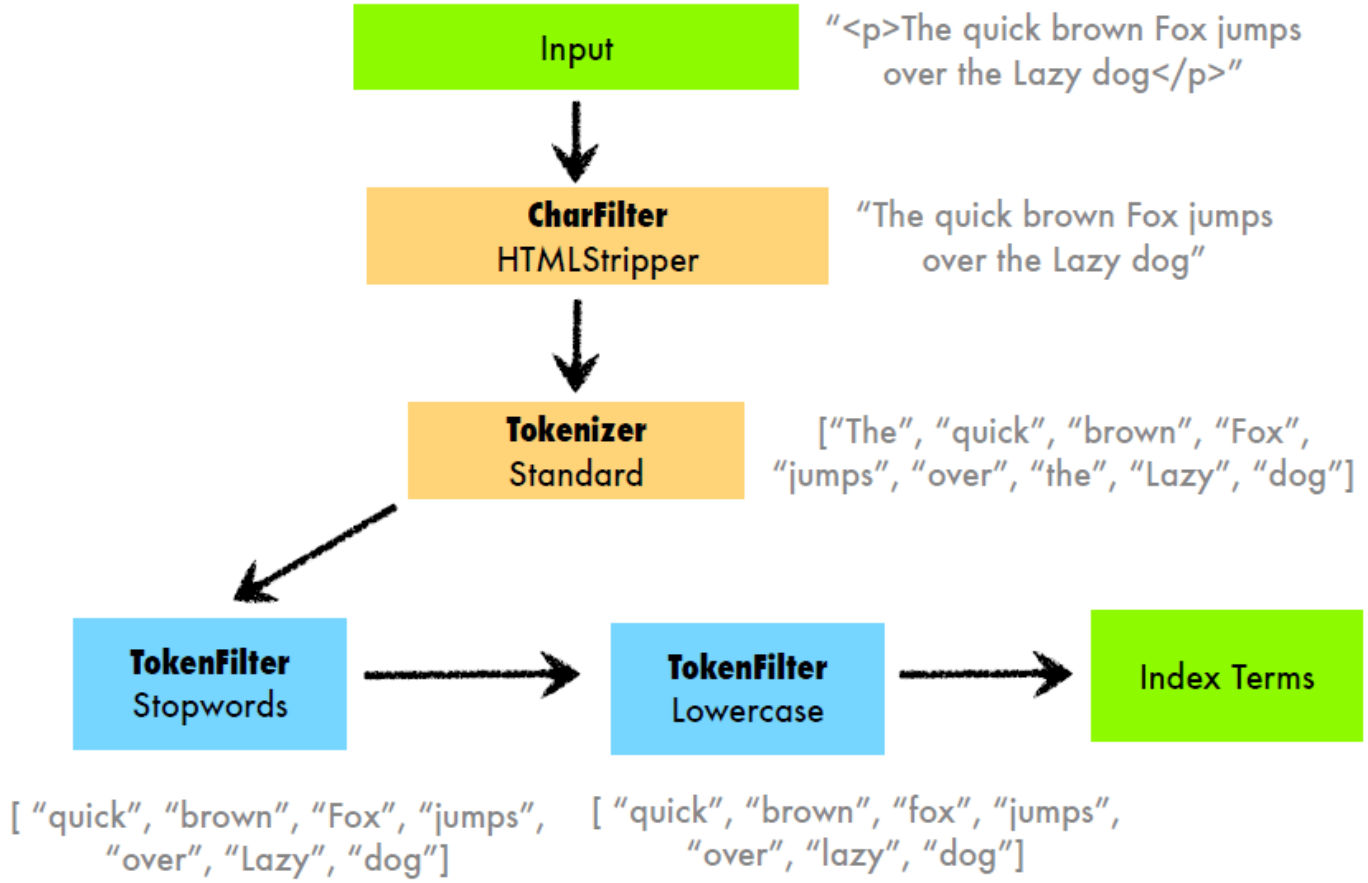
- Each field has dedicated index
- Terms are sorted alphabetically for performance reasons
- Values for a text field are analyzed and the results are sorted within an inverted index

Analysis

- Sometimes referred to as text analysis
- Applicable to text field/values
- Text values are analyzed when indexing documents
- The result is stored in data structures that are efficient for searching etc
- The _source object is not used when searching for documents
  - It contains the exact values specified when indexing a document

Document analysis process

- **The same process of document normalization (during inverted index) needs to be applied during document search query.**



Analysis & analyzer

- Analysis is a process that consits of the following:
  - First, tokenizing a block of text into individual terms suitable for use in an inverted index
  - Then normalizing these terms into a standard form to improve their “searchability”, or recall.
- An analyzer is really just a wrapper that combines three functions into a single package:
  - Character filters
  - Tokenizer
  - Token filters

Character filters

- Adds, removes, or change characters
- Analyzers contain zero or more chacraters filters
- Character filters are applied in the order in which they are specified
- The result is stored in data structures that are efficient for searching etc
- The _source object is not used when searching for documents
  - It contains the exact values specified when indexing a document

Token filters

- Receive the output of the tokenizer as input
- A token filter can add, remove, or modify tokens
- An analyzer contains zero or more token filters
- Token filters are applied in the order in which they are specified
- Example (lowercase filter)
  - **Input**: [“I”, “REALLY”, “like”, “beer”]
  - **Output:** [“I”, “really”, “like”, “beer”]

Build-in and custom components

- Built-in analyzers, character filters, tokenizers, and token filters are available
- We can also build custom ones

Example

POST /_analyze

```
{
  "text": "2 guys walk in to bar, but third...DUCK :)",
  "analyzer": "standard"
  }
```

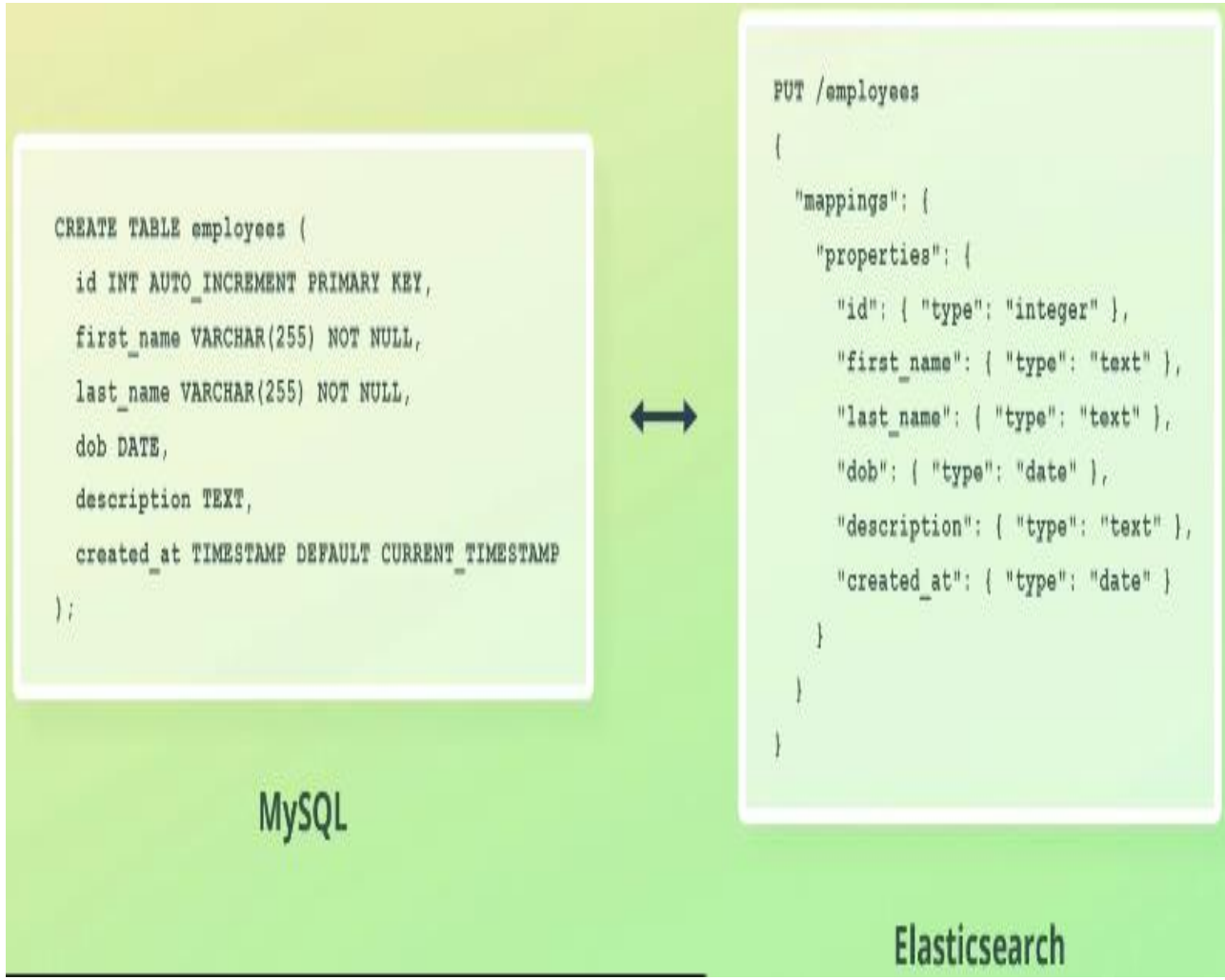

```
{
  "tokens" : [
    {
      "token" : "2",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "<NUM>",
      "position" : 0
    },
    {
      "token" : "guys",
      "start_offset" : 2,
      "end_offset" : 6,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "walk",
      "start_offset" : 7,
      "end_offset" : 11,
      "type" : "<ALPHANUM>",
      "position" : 2
    },
```

YOUR TURNS

Khám phá một analyzer bất kì với đoạn text bất kỳ

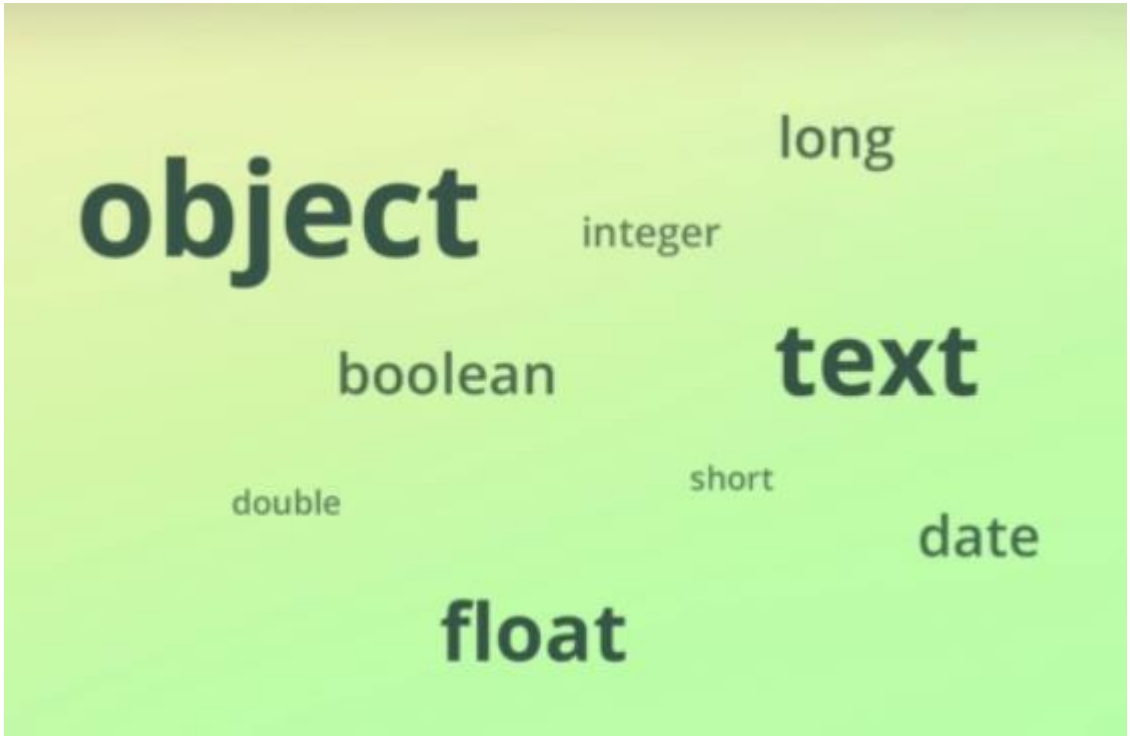
Mapping

- Defines the structure of documents (e.g fields and their data types)
  - Also used to configure how values are indexed
- Similar to the tables's schema in a RDBMS
- Explicit mapping
  - We define filed mappings ourselves
- Dynamic mapping: ElasticSearch defined



Datatype

- Object data type
  - Used for any JSON object
  - Objects may be nested
  - Mapped using the properties parameter
  - Objects are not stored as objects in Apache lucene
    - Objects are transformed to ensure that we can index any valid JSON



Keyword data type

- Used for exact matching of values
- Typicallly used for filtering, aggregation, and sorting
- E.g. searching for articles with status of PUBLISHED
- For full-text searchs, use the **text** data type instead
  - E.g. searching the body text of an article

How Keyword fields are analyzed

- **Keyword** fileds are analyzed with the **keyword** analyzer
- The **keyword** analyzer is a no-op analyzer
  - It outputs the unmodified string as a single token
- For full-text searchs, use the **text** data type instead
  - E.g. searching the body text of an article

```
POST /_analyze
{
 "text": "2 guys walk in to bar, but third ... DUCK:)",
 "analyzer": "keyword"
 }
```

```
{
 "tokens" : [
  {
   "token" : "2 guys walk in to bar, but third ... DUCK:)",
   "start_offset" : 0,
   "end_offset" : 43,
   "type" : "word",
   "position" : 0
  }
 ]
```

How Keyword fields are analyzed

Understanding type coercion

- **Data types are inspected when indexing documents**
  - **They are validated, and some invalide values are rejected**
  - **E.g. trying to index to an object for a text field**
- Sometimes, providing the wrong data type is okay
  - Example

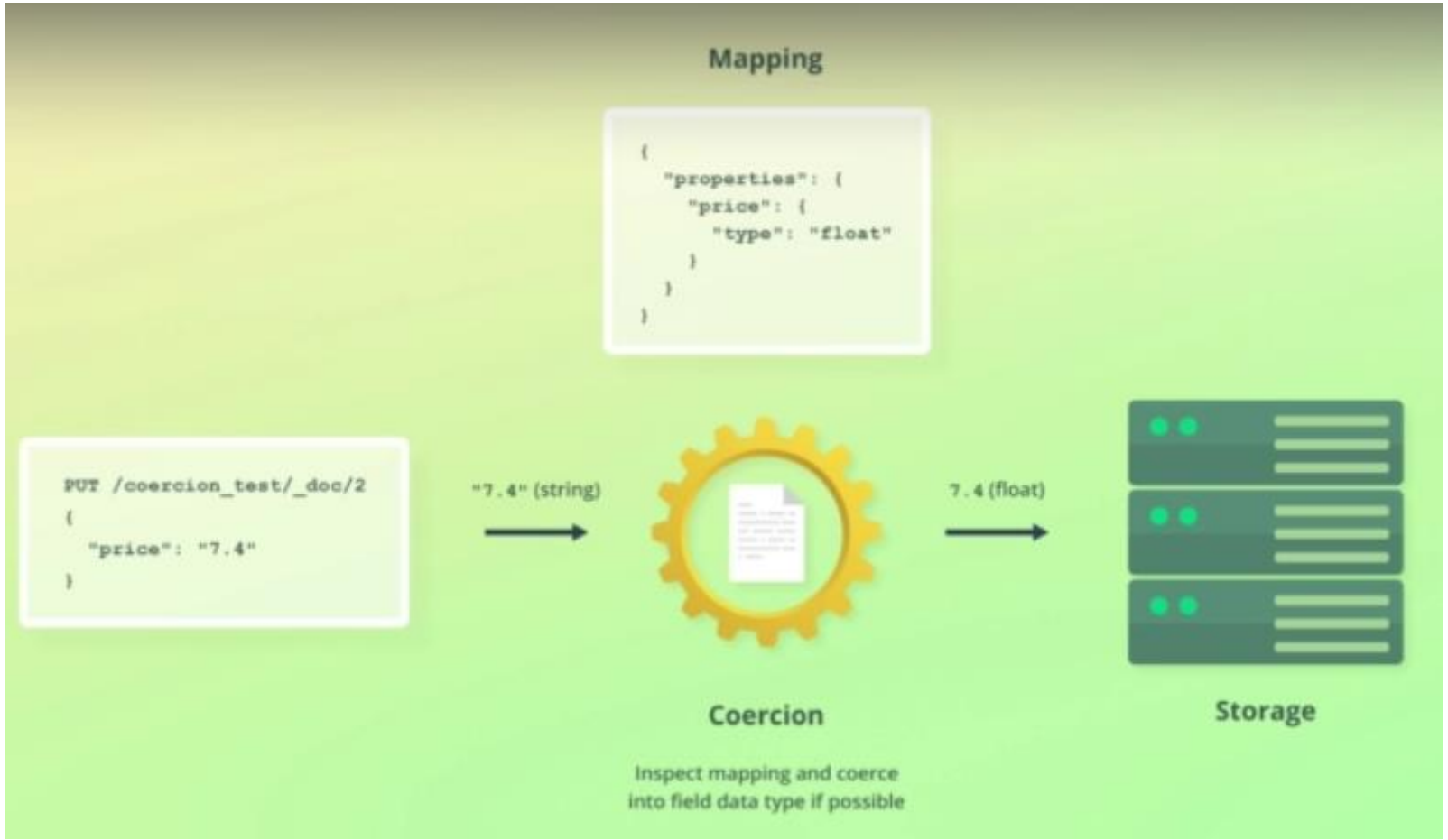
```
POST /_analyze
{
  "text": "2 guys walk in to bar, but third ... DUCK:)",
  "analyzer": "keyword"
  }
```

```
{
  "tokens" : [
    {
      "token" : "2 guys walk in to bar, but third ... DUCK:)",
      "start_offset" : 0,
      "end_offset" : 43,
      "type" : "word",
      "position" : 0
    }
  ]
```

Understanding type coercion

```
PUT /coercion_test/_doc/1
{
  "price":7.4
}

PUT /coercion_test/_doc/2
{
  "price":"7.4"
}
PUT /coercion_test/_doc/3
{
  "price":"7.4m"
}
GET /coercion_test/_doc/2
```



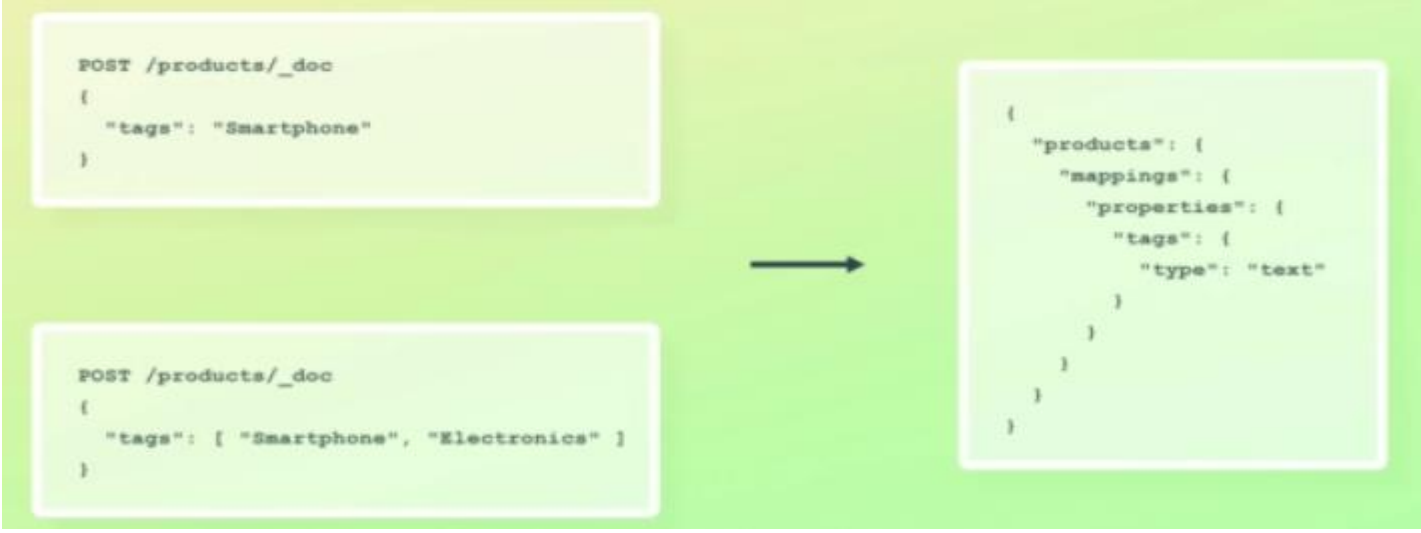
Continues

- Supplying a floating point for an integer filed will truncate it to an integer
- Coercion is not used for dynamic mapping
  - Supplying “7.4” for a new filed will create a text mapping
- Always try to use the correct data type
  - Especially the first time you index a filed
- Disabling coercion is a matter of preference
  - Enabled by default

Understanding arrays

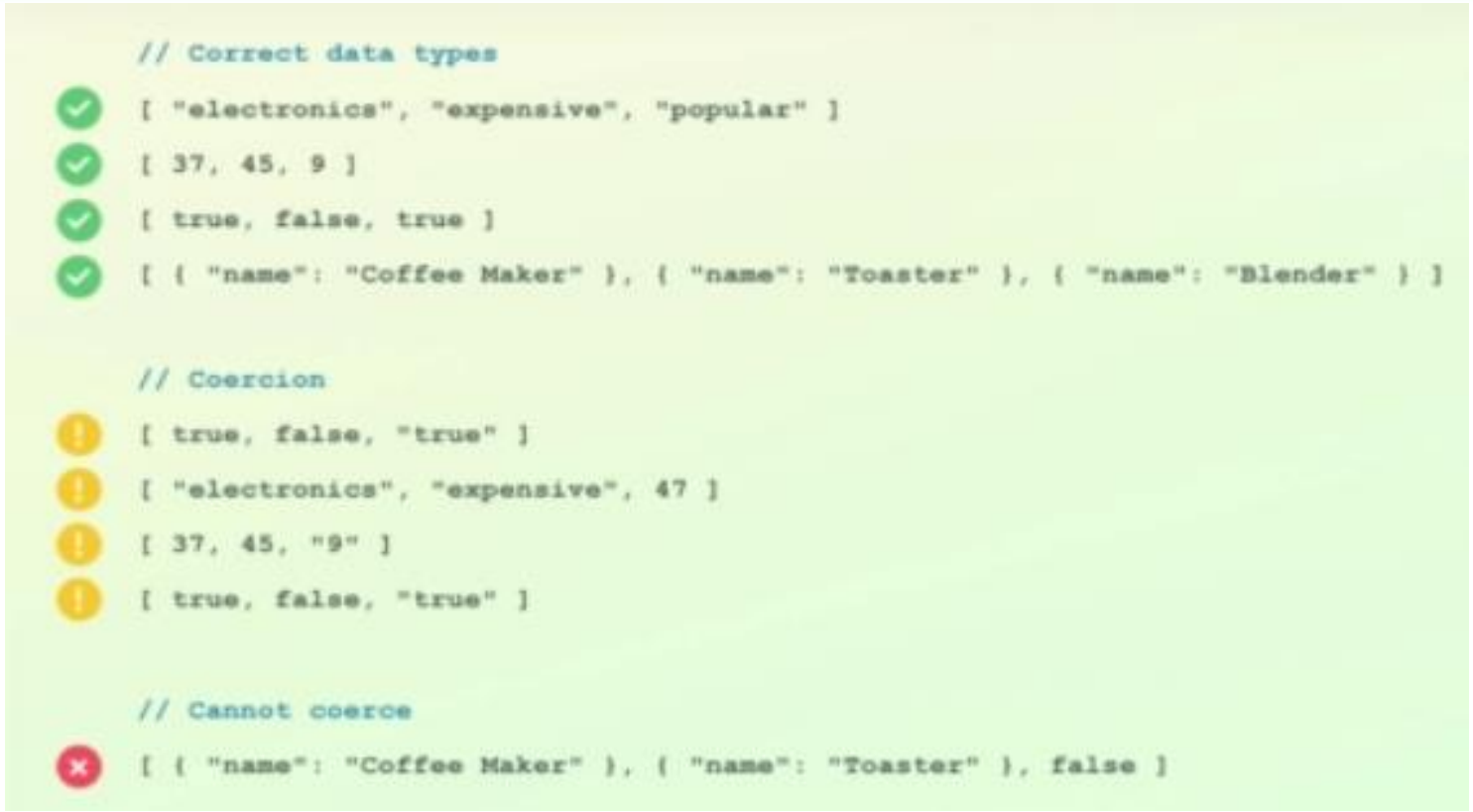
- **There is no such thing as an array data type**
- Any filed may contain zero or more values
  - No configuration or mapping needed
  - Simply supply an array when indexing a document
- We did this for the tags filed for the products index

```
POST /_analyze
{
  "text": ["String are simply", "merge together"],
  "analyzer": "standard"
  }
```



Array

- Array values should be of the same data type
- Coercion only works for fields that are already mapped
  - Must contain the same data type
  - Can nested array



Adding explicit mappings

```
PUT /reviews
{
  "mappings": {
    "properties": {
      "rating": {"type": "float"},
      "content": {"type": "text"},
      "product_id": {"type": "integer"},
      "author": {
        "properties": {
          "first_name":{"type":"text"},
          "last_name":{"type":"text"},
          "email":{"type":"keyword"}
        }
      }

    }
  }
}
```

```
PUT /reviews/_doc/1
{
   "rating": 5.0,
    "content": "test review",
    "product_id": 123,
      "author": {
          "first_name":“cao",
          "last_name":“sao",
          "email":“scd@gmail.com"
      }
}
```

Retrieving mappings

```
GET /reviews/_mapping
GET /reviews/_mapping/field/content
```

Can use dot notation

```
 "properties": {
     "rating": {"type": "float"},
     "content": {"type": "text"},
     "product_id": {"type": "integer"},
     "author.first_name ":{"type":"text"},
"author.last_name":{"type":"text"},
         "author.email":{"type":"keyword"}
```

```
"reviews" : {
  "mappings" : {
   "properties" : {
    "author" : {
     "properties" : {
      "email" : {
       "type" : "keyword"
      },
      "first_name" : {
       "type" : "text"
      },
      "last_name" : {
       "type" : "text"
      }
     }
    },
    "content" : {
     "type" : "text"
    },
    "product_id" : {
     "type" : "integer"
    },
```

Adding mappings to existing indices

```
PUT /reviews/_mapping
{
  "properties":{
    "created_at": {
      "type": "date"
    }
  }
}}

GET /reviews/_doc/1
```

```
{
  "_index" : "reviews",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "rating" : 5.0,
    "content" : "test review",
    "product_id" : 123,
    "author" : {
      "first_name" : "tran",
      "last_name" : "thuyet",
      "email" : "thuyettv1@mail.com"
    }
  }
}
```

Mapping parameters

- Format parameter (customize date format)
- Properties parameter (nested, object)
- Coerce parameter (disable coercion)

```
PUT /sales
{
  "mappings": {
    "properties": {
      "purchased_at": {
        "type": "date",
        "format": "dd/MM/yyyy"
      }
    }
  }
}
```

```
PUT /sales
{
  "mappings": {
    "properties": {
      "sold_by": {
        "properties": {
          "name": { "type": "text" }
        }
      }
    }
  }
}
```

```
PUT /sales
{
  "mappings": {
    "properties": {
      "amount": {
        "type": "float",
        "coerce": false
      }
    }
  }
}
```

Doc_values

- Doc values are the on-disk data structure
- They store the same values as the _source but in a column-oriented fashion that is way more efficient for sorting and aggregations
- If you are sure that you don't need to sort or aggregate on a field, or access the field value from a script, you can disable doc values in order to save disk space

```
PUT my-index-000001
{
  "mappings": {
    "properties": {
      "status_code": {
        "type":      "keyword"
      },
      "session_id": {
        "type":      "keyword",
        "doc_values": false
      }
    }
  }
}
```

Norms

- Norms store various normalization factors that are later used at query time in order to compute the score of a document relatively to a query

- Although useful for scoring, norms also require quite a lot of disk (typically in the order of one byte per document per field in your index, even for documents that don't have this specific field). As a consequence, if you don't need scoring on a specific field, you should disable norms on that field. In particular, this is the case for fields that are used solely for filtering or aggregations

```
PUT my-index-
000001/_mapping
{
  "properties": {
    "title": {
      "type": "text",
      "norms": false
    }
  }
}
```

Mapping parameters

- Index parameter (can disable)
- null_value (cannot be indexed or searched)
- Copy_to parameter

```
PUT /sales
{
  "mappings": {
    "properties": {
      "first_name": {
        "type": "text",
        "copy_to": "full_name"
      },
      "last_name": {
        "type": "text",
        "copy_to": "full_name"
      },
      "full_name": {
        "type": "text"
      }
```

```
POST /sales/_doc
{
  "first_name": "John",
  "last_name": "Doe"
}
```

Hand-on labs