

Akeneo **Workshop**
Elasticsearch edition

12/02/2019

Objectives

- To gain a **basic** understanding of:
 - How ES works
 - How to use ES
 - How we use ES in the PIM

*(**Disclaimer:** I'm not an ES expert 😊)*

Summary

- The **Basics**
- **Part I:** Mapping & Indexing
- **Part II:** Searching
- **Part III:** Elasticsearch & PIM

The **Basics**

- **Search engine** based on Lucene
- Open source
- Asynchronous (Near real time)
- Runs on JVM

```
$> java -Xms512m -Xmx512m ...
```

Usage

CREATE INDEX WITH Settings

PUT es.local/my_index



settings



ES

INDEX Documents

PUT es.local/my_index/doc_1



data



ES

Query

GET es.local/my_index



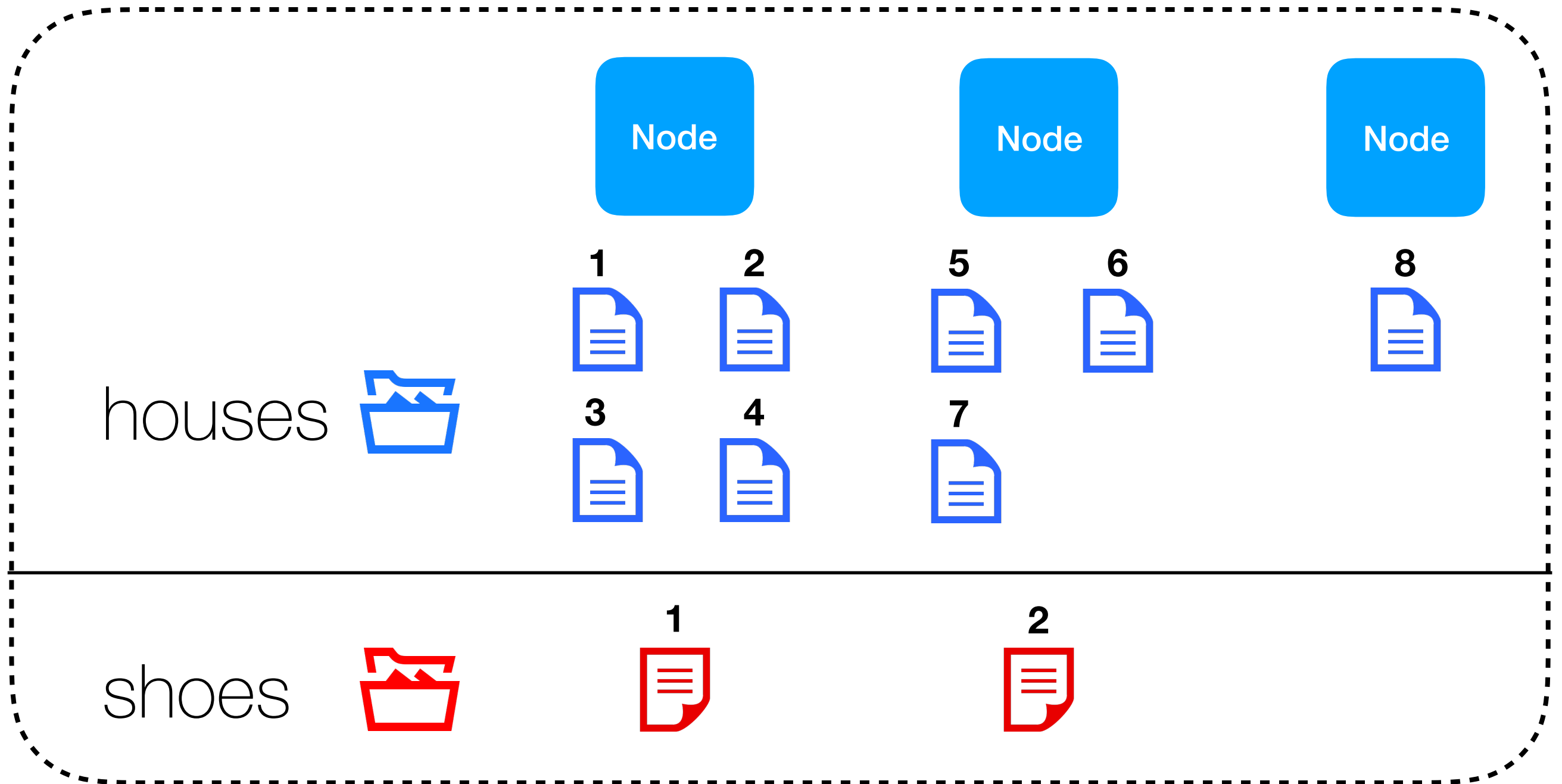
Document



ES

Infrastructure

Cluster



Part I: **Mapping & Indexing**

- What is a **Mapping** ?
- Normalizers & **Analysers**
- The **inverted dictionary**
- Exercices

What is a mapping ?

- It's a simple json file
- It's a way of telling ES about:
 - ▶ How should it interpret the data ?
 - ▶ What should it do with it ?

Mapping by default

```
{  
  "id": "document_1",  
  "content": "The Brown Fox Is Very Quick",  
  "speed": 200,  
  "is_first": true,  
  "record_date": "20/02/2019"  
}
```

Diagram illustrating the mapping of JSON fields to their default types:

- `"content": "The Brown Fox Is Very Quick"` maps to **text**
- `"speed": 200` maps to **long**
- `"is_first": true` maps to **boolean**
- `"record_date": "20/02/2019"` maps to **text**

Custom mapping (you choose)

- Map properties

```
mappings:
  my_type:
    properties:
      content:
        type: 'keyword'
      speed:
        type: 'number'
      is_first:
        type: 'boolean'
      record_date:
        type: 'date'
```

```
{
  "id": "document_1",
  "content": "The Brown Fox Is Very Quick",
  "speed": "200",
  "is_first": "true",
  "record_date": "20/02/2019"
}
```

"content": "The Brown Fox Is Very Quick",	—————→	Text
"speed": "200",	—————→	Integer
"is_first": "true",	—————→	Boolean
"record_date": "20/02/2019"	—————→	Date

Custom mapping (you choose)

- Map dynamic properties

```
{  
  "id": "document_1",  
  ...  
  "race-of-the-rock": 10.50,  
  "race-of-the-pine-tree": 9.50,  
  "race-of-the-garden": 11.50  
}
```

```
mappings:  
  my_type:  
    dynamic_templates:  
      -  
        map_races_records:  
          path_match: 'race-*'  
          mapping:  
            type: 'float'
```

Custom mapping (you choose)

- Map dynamic properties

```
{
  "id": "document_1",
  "races": {
    "race-of-the-rock": 10.50,
    "race-of-the-pine-tree": 9.50,
    "race-of-the-garden": 11.50
  },
}
```

```
mappings:
  my_type:
    dynamic_templates:
      -
        map_races_records:
          path_match: 'race.*'
          mapping:
            type: 'float'
```

Available types

- Available types:
 - ▶ **"Keyword"** (aggregation and sorting)
 - ▶ **"text"** (for full text search)
 - ▶ Date, Long, Double, Boolean, Ip, etc.

Analysers

Analysers

- Happens at **index time**

Analysers

- Happens at **index time**
- 1. **Tokenizing** a block of text into terms

Tokenize: standard

"The White Rabbit Is Also Very Quick"  ["The", "White", "Rabbit", "Is", "Also", "Very", "Quick"]

Analysers

- Happens at **index time**
- 1. **Tokenizing** a block of text into terms

Tokenize: standard

"The White Rabbit Is Also Very Quick"  ["The", "White", "Rabbit", "Is", "Also", "Very", "Quick"]

Analysers

- Happens at **index time**
- 1. **Tokenizing** a block of text into terms

Tokenize: standard

"The White Rabbit Is Also Very Quick"  ["The", "White", "Rabbit", "Is", "Also", "Very", "Quick"]

- 2. **Normalizing** terms to improve searchability

["The", "White", "Rabbit", "Is", "Also", "Very", "Quick"]



Normalize: lowercase

["the", "white", "rabbit", "is", "also", "very", "quick"]

Declaring an Analyser

- In the settings we declare an analyser
 - **Character filter**: (strip html)
 - **Tokenizer**: Split the sentence into terms using a strategy (language)
 - **Token filter**: remove terms (a, and, or) lowercase

```
settings:
  analysis:

    analyzer:
      my_analyzer:
        filter: ['lowercase']
        char_filter: ['html_strip', 'newline_pattern']
        type: 'custom'
        tokenizer: 'standard'

    char_filter:
      newline_pattern:
        pattern: '\\n'
        type: 'pattern_replace'
        replacement: ''

    filter:
      text_area_truncate:
        type: 'truncate'
        length: 100

    normalizer:
      my_normalizer:
        filter: ['lowercase']
```

Use them in mapping

```
mappings:  
  pim_catalog_product:  
    properties:  
  
      my_property:  
        type: 'keyword'  
        normalizer: 'my_normalizer'  
  
      my_other_property:  
        type: 'keyword'  
        analyzer: 'my_analyzer'
```

But, Why bother with
normalizers and analyzers
?

But, Why bother with
normalizers and analyzers
?

*To improve **Searchability***

The Inverted index

```
{  
  "id": "document_1"  
  "content": "The brown fox is very quick"  
}
```


The Inverted index

```
{  
  "id": "document_1"  
  "content": "The brown fox is very quick"  
}
```



Index

The Inverted index

```
{  
  "id": "document_1"  
  "content": "The brown fox is very quick"  
}
```

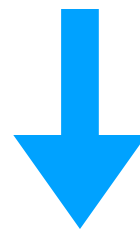


Index

Token	document_1
The	x
brown	x
fox	x
is	x
very	x
quick	x
white	
rabbit	

The Inverted index

```
{  
  "id": "document_2"  
  "content": "The white rabbit is also very quick"  
}
```



Index

Token	document_1	document_2
The	x	x
brown	x	
fox	x	
is	x	x
very	x	x
quick	x	x
white		x
rabbit		x

The Inverted index

Token	document_1	document_2
The	x	x
brown	x	
fox	x	
is	x	x
very	x	x
quick	x	x
white		x
rabbit		x



Find documents having terms:
"very" and "quick"

Token	document_1	document_2
very	x	x
quick	x	x

The Inverted index

Token	document_1	document_2
The	x	x
brown	x	
fox	x	
is	x	x
very	x	x
quick	x	x
white		x
rabbit		x



Find documents having terms:
"quick" and "rabbit"

Token	document_1	document_2
rabbit		x
quick	x	x

The Inverted index

Token	document_1	document_2
The	x	x
brown	x	
fox	x	
is	x	x
very	x	x
quick	x	x
white		x
rabbit		x



Find documents having terms:
"the" and "rabbit"

Token	document_1	document_2
the		
rabbit		x

The Inverted index

Token	document_1	document_2
the	x	x
brown	x	
fox	x	
is	x	x
very	x	x
quick	x	x
white		x
rabbit		x

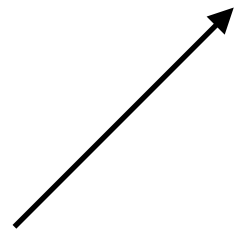


Find documents having terms:
"the" and "rabbit"

Token	document_1	document_2
the	x	x
rabbit		x

The Inverted index

ES can do that for us
at index time thanks to
Analysis



Token	document_1	document_2
the	x	x
brown	x	
fox	x	
is	x	x
very	x	x
quick	x	x
white		x
rabbit		x



Find documents having terms:
"the" and "rabbit"

Token	document_1	document_2
the	x	x
rabbit		x

Exercises on Mapping



```
git clone git@github.com:samirboulil/es-workshop  
--branch=workshop
```

Ok,

so which **type** should I use ?

which **transformation** should we
perform ?



you need to rely on your
search use-cases
to actually
write your **mapping**



Part II: **Searching**

- Generic request model
- Term level queries
- Compound queries
- Exercises

Searching - Generic model

```
{
  "_source": ["id", "title"], // Properties you want
  "sort": [{"title": {"order": "ASC"}}, // sort order
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            // AND clauses
          ],
          "must_not": [
            // NOT clauses
          ],
          "should": [
            // OR clauses
          ],
        }
      }
    }
  }
}
```

Searching - Quiz 1

Elasticsearch

SQL ?

```
{
  "_source": ["title", "release_year"],
  "sort": [
    {"release_year": {"order": "DESC"}}
  ],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "term": {
                "title": "ARMAGEDDON"
              }
            }
          ],
        }
      }
    }
  }
}
```

Searching - Quiz 1

Elasticsearch

SQL ?

```
{
  "_source": ["title", "release_year"],
  "sort": [
    {"release_year": {"order": "DESC"}}
  ],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "term": {
                "title": "ARMAGEDDON"
              }
            }
          ],
        }
      }
    }
  }
}
```

```
SELECT title, release_year
FROM movies
WHERE title LIKE "%ARMAGEDDON%"
ORDER BY release_year DESC
```

Searching - Quiz 2

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "term": {
                "title": "ARMAGEDDON"
              }
            },
            {
              "term": {
                "language": "French"
              }
            }
          ]
        }
      }
    }
  }
}
```


Searching - Quiz 2

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "term": {
                "title": "ARMAGEDDON"
              }
            },
            {
              "term": {
                "language": "French"
              }
            }
          ],
          "minimum_should_match": 1
        }
      }
    }
  }
}
```

```
SELECT title
FROM movies
WHERE
  title LIKE "%ARMAGEDDON%"
AND
  language = "French"
```

Searching - Quiz 3

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "term": {
                "language": "French"
              }
            },
            {
              "term": {
                "release_year": 2006
              }
            }
          ]
        }
      }
    }
  }
}
```

Searching - Quiz 3

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "term": {
                "language": "French"
              }
            },
            {
              "term": {
                "release_year": 2006
              }
            }
          ]
        }
      }
    }
  }
}
```

```
SELECT title
FROM movies
WHERE
  language = "French"
AND
  release_year <> 2006
```

Searching - Quiz 4

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "should": [
            {
              "term": {
                "release_year": "2006"
              }
            },
            {
              "term": {
                "release_year": "2007"
              }
            }
          ]
        }
      }
    }
  }
}
```

Searching - Quiz 4

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "should": [
            {
              "term": {
                "release_year": "2006"
              }
            },
            {
              "term": {
                "release_year": "2007"
              }
            }
          ]
        }
      }
    }
  }
}
```

```
SELECT title
FROM movies
WHERE
  release_year = "2006"
OR
  release_year = "2007"
```

Searching - term level queries

- The operators you can use depend on the data type (defined in the mapping):
 - ▶ strings: "term" / "terms" / "query_string"
 - ▶ dates / numbers: "range"
 - ▶ check a field exists: "exists"

Searching - Quiz 5

Elasticsearch ?

SQL

```
SELECT title
FROM movies
WHERE
  release_year = "2006"
OR
  release_year = "2007"
```

Any way to simplify this in SQL?

Searching - Quiz 5

Elasticsearch

SQL

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "terms": {
                "release_year": [
                  "2006",
                  "2007"
                ]
              }
            },
            {}
          ],
          "minimum_should_match": 1
        }
      }
    }
  }
}
```

```
SELECT title
FROM movies
WHERE
  release_year IN ("2006", "2007")
```

Yes! (It's an implicit "OR")



Searching - Quiz 6

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "query_string": {
                "default_field": "title",
                "query": "*armag*"
              }
            }
          ],
          "minimum_should_match": 1
        }
      }
    }
  }
}
```

```
SELECT title
FROM movies
WHERE
  title LIKE "%armag%"
```

Searching - Quiz 7

Elasticsearch

SQL ?

```
{
  "_source": ["title"],
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "exists": {
                "field": "director",
              }
            },
          ],
        },
      },
    },
  },
}
```

```
SELECT title
FROM movies
WHERE
  director IS NOT NULL
```

Searching - Quiz 7

Elasticsearch

Exists Matches

```
{ "director": "jane" }  
{ "director": "" }  
{ "director": "-" }  
{ "director": [jane] }  
{ "director": [jane, null] }
```

SQL ?

Exists Does not Match

```
SELECT title  
FROM movies  
WHERE director IS NOT NULL  
{ "user": null }  
{ "user": [] }  
{ "user": [null] }  
{ "foo": "bar" }
```

Searching - Compound queries

- It's all about the **Bool**

Searching - Quiz 8

Elasticsearch

Logical expression ?

```
{
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "bool": {
                "should": [
                  {CLAUSE A},
                  {CLAUSE B}
                ]
              }
            },
            {
              "bool": {
                "should": [
                  {CLAUSE C},
                  {CLAUSE D}
                ]
              }
            }
          ]
        }
      }
    }
  }
  ...
}
```

Searching - Quiz 8

Elasticsearch

Logical expression ?

```
{
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "bool": {
                "should": [
                  {CLAUSE A},
                  {CLAUSE B}
                ]
              }
            },
            {
              "bool": {
                "should": [
                  {CLAUSE C},
                  {CLAUSE D}
                ]
              }
            }
          ]
        }
      }
    }
  }
}
```

(A || B) && (C || D)

Exercises on Searching



```
git clone git@github.com:samirboulil/es-workshop  
--branch=workshop
```

Part III: **ES** and the **PIM**

- Define a new index in the PIM
- Product query builder and ES
- How do we test ?
- Case study: full-text search reference entities

ES & PIM : New index

- Via configuration:

parameters:

record_index_configuration:

- 'path/to/record_index_configuration.yml'

akeneo_elasticsearch:

hosts: "%index_hosts%"

indexes:

-

service_name: "akeneo_referenceentity.client.record"

index_name: "%record_index_name%"

configuration_files: "%record_index_configuration%"

- ES Client generated at kernel compile time

ES & PIM : New index

- We have a ESClient wrapper which can do much more things easily
- Akeneo\Tool\Bundle\ElasticsearchBundle\Client
 - ▶ Client::index && Client::bulkIndexes
 - ▶ Client::delete && Client::bulkDelete
 - ▶ Client::refreshIndex
 - ▶ Client::resetIndex

ES & PIM : PQB overview



PQB

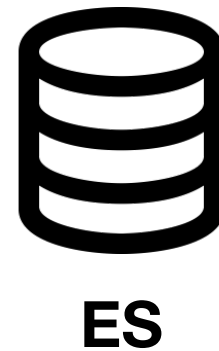
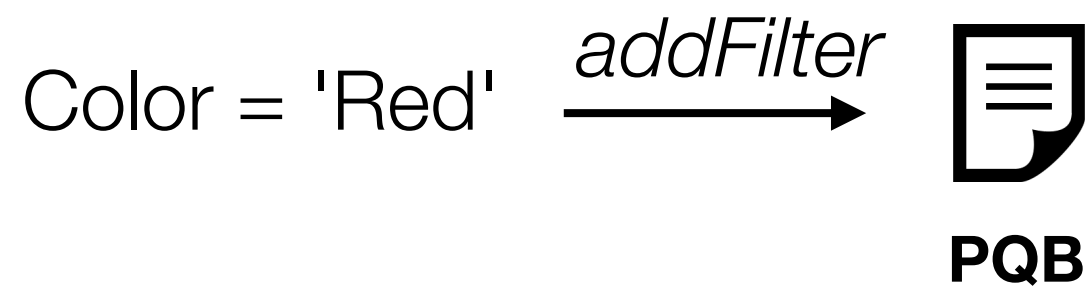


ES



Mysql

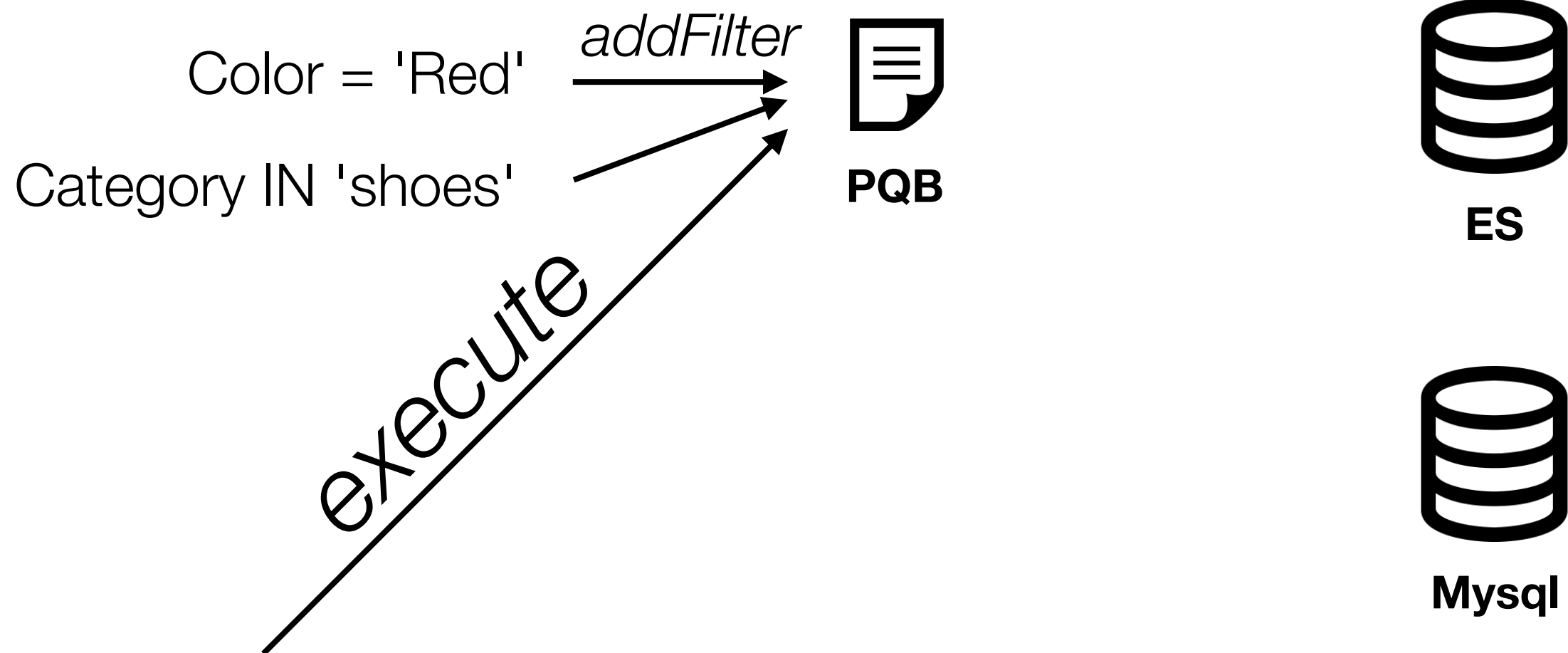
ES & PIM : PQB overview



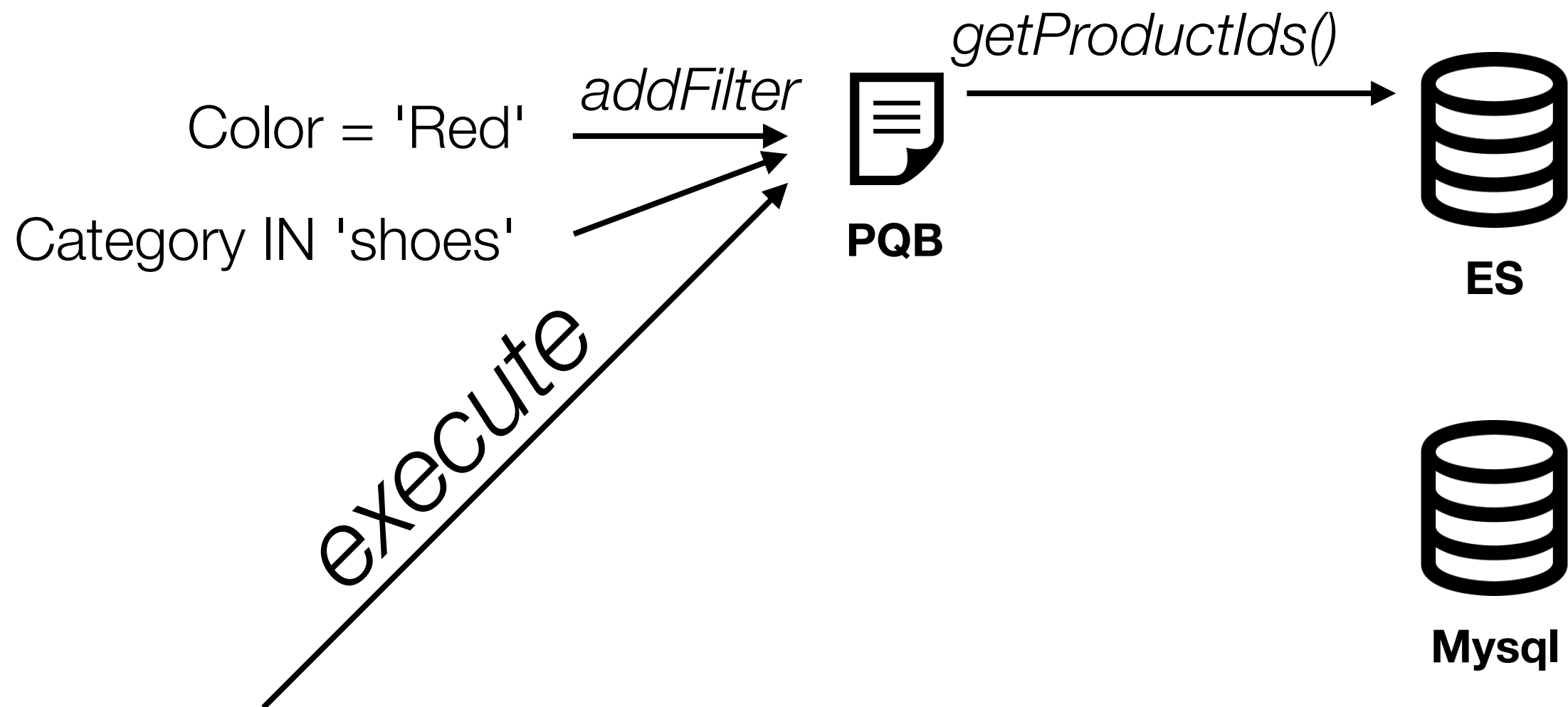
ES & PIM : PQB overview



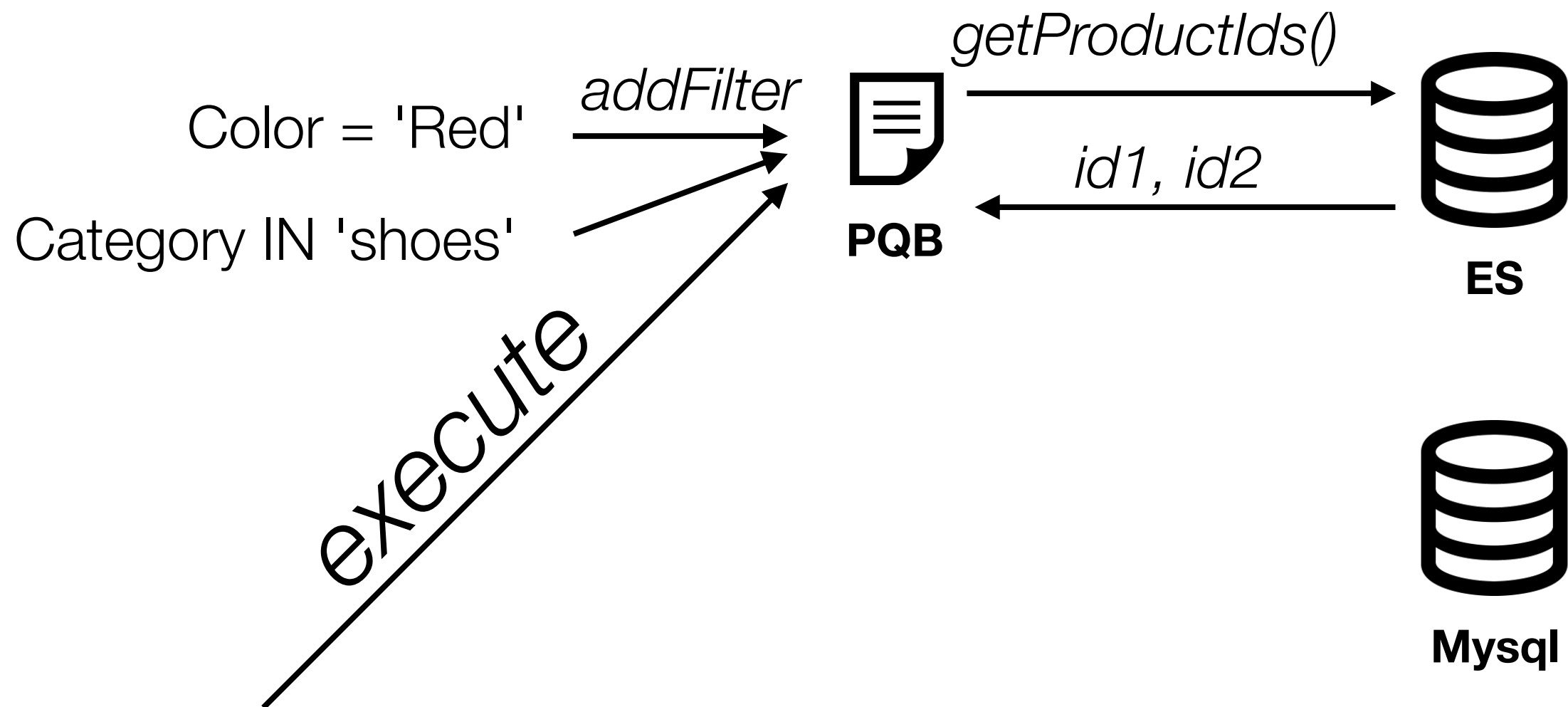
ES & PIM : PQB overview



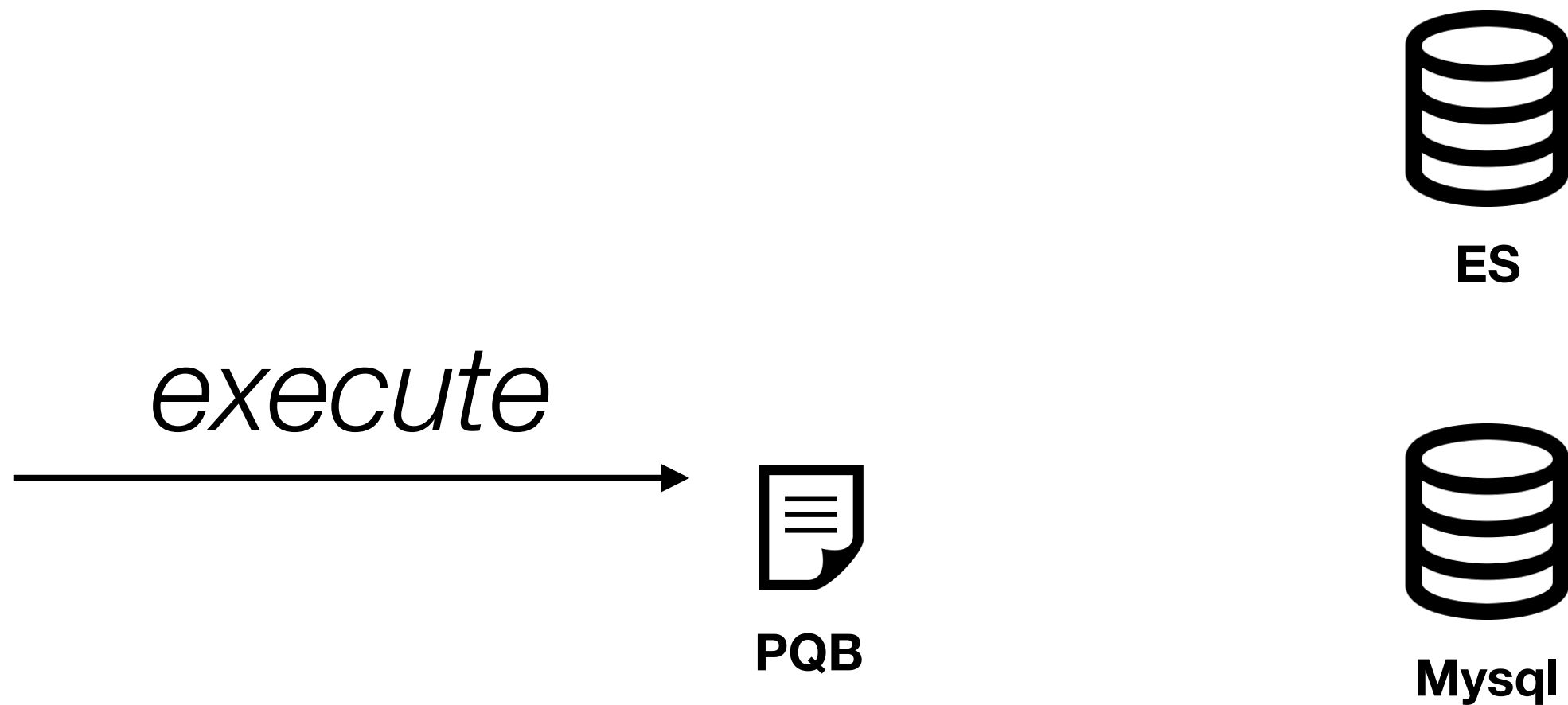
ES & PIM : PQB overview



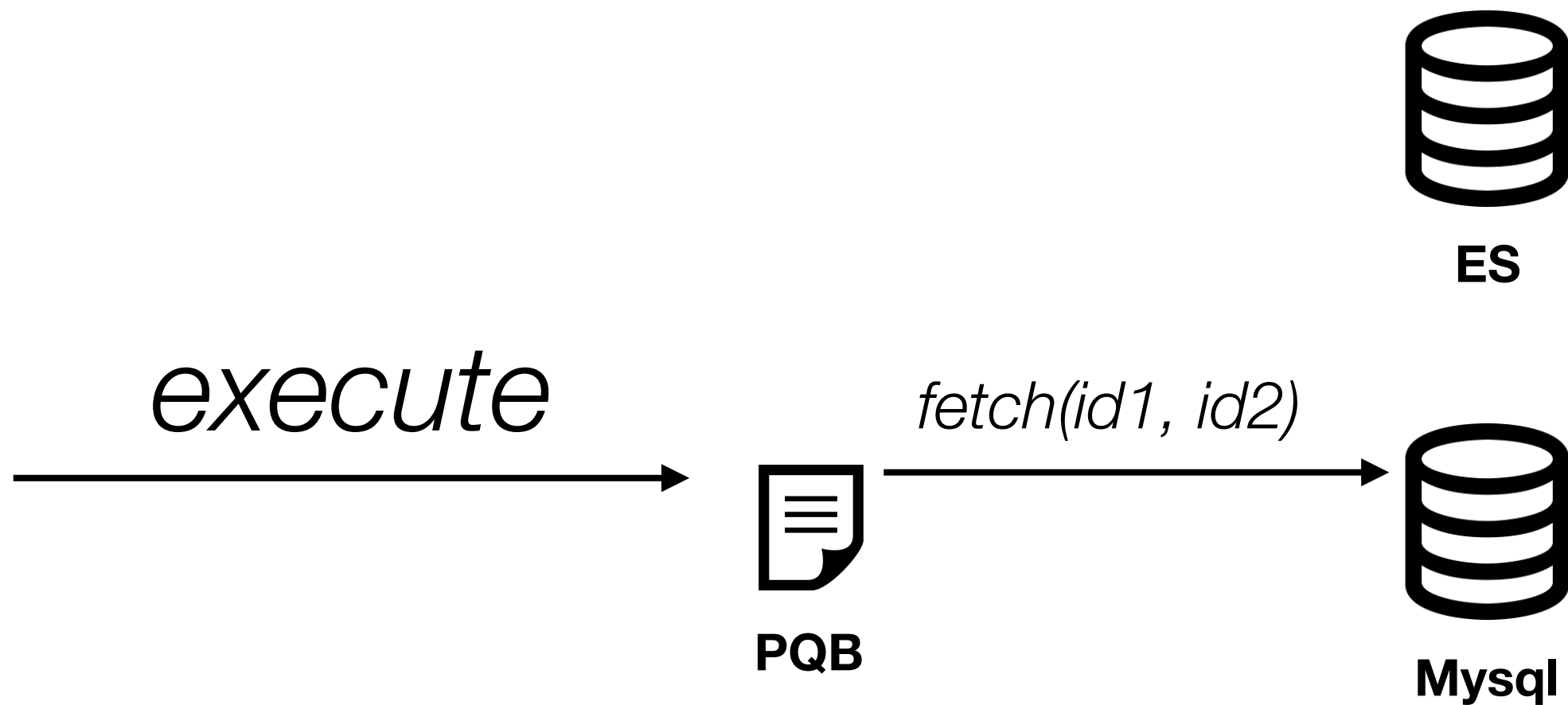
ES & PIM : PQB overview



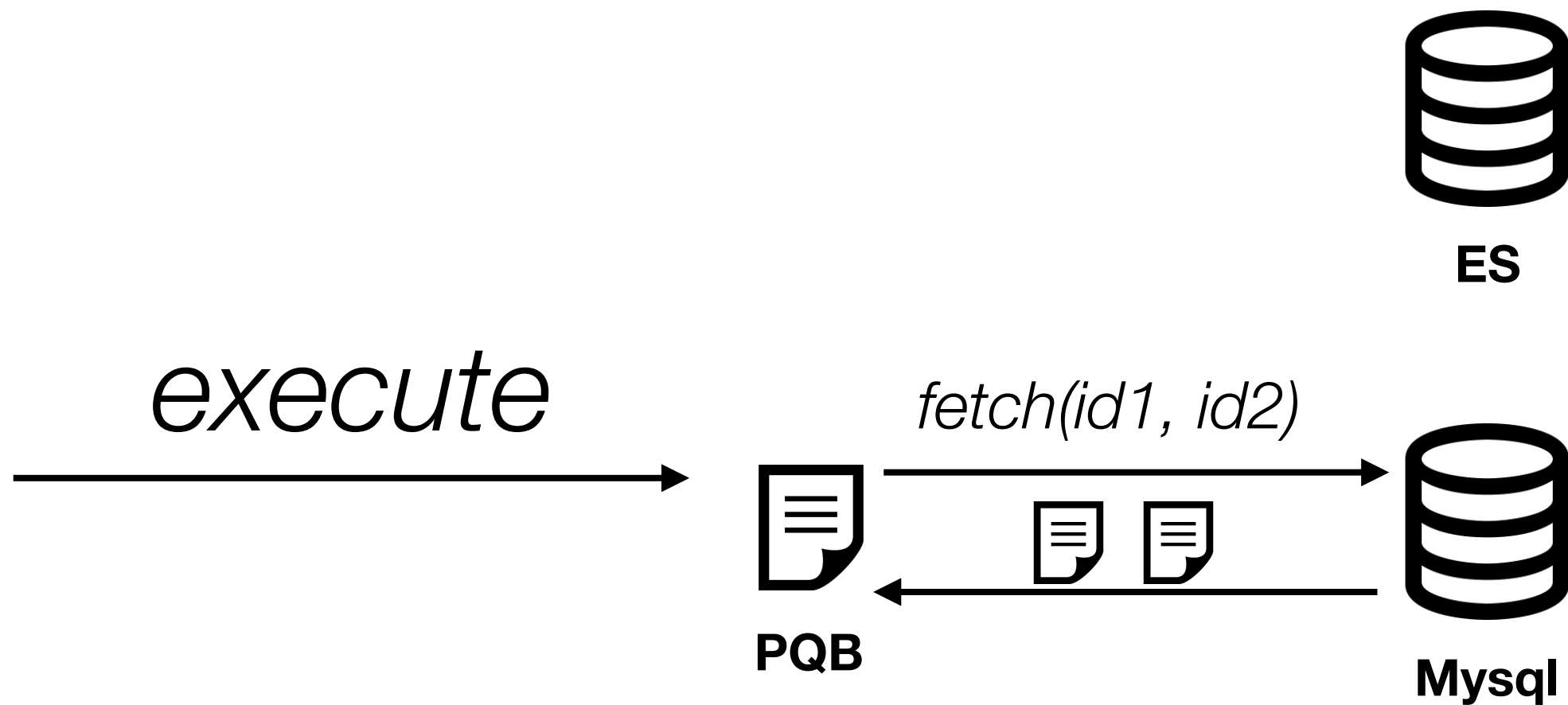
ES & PIM : PQB overview



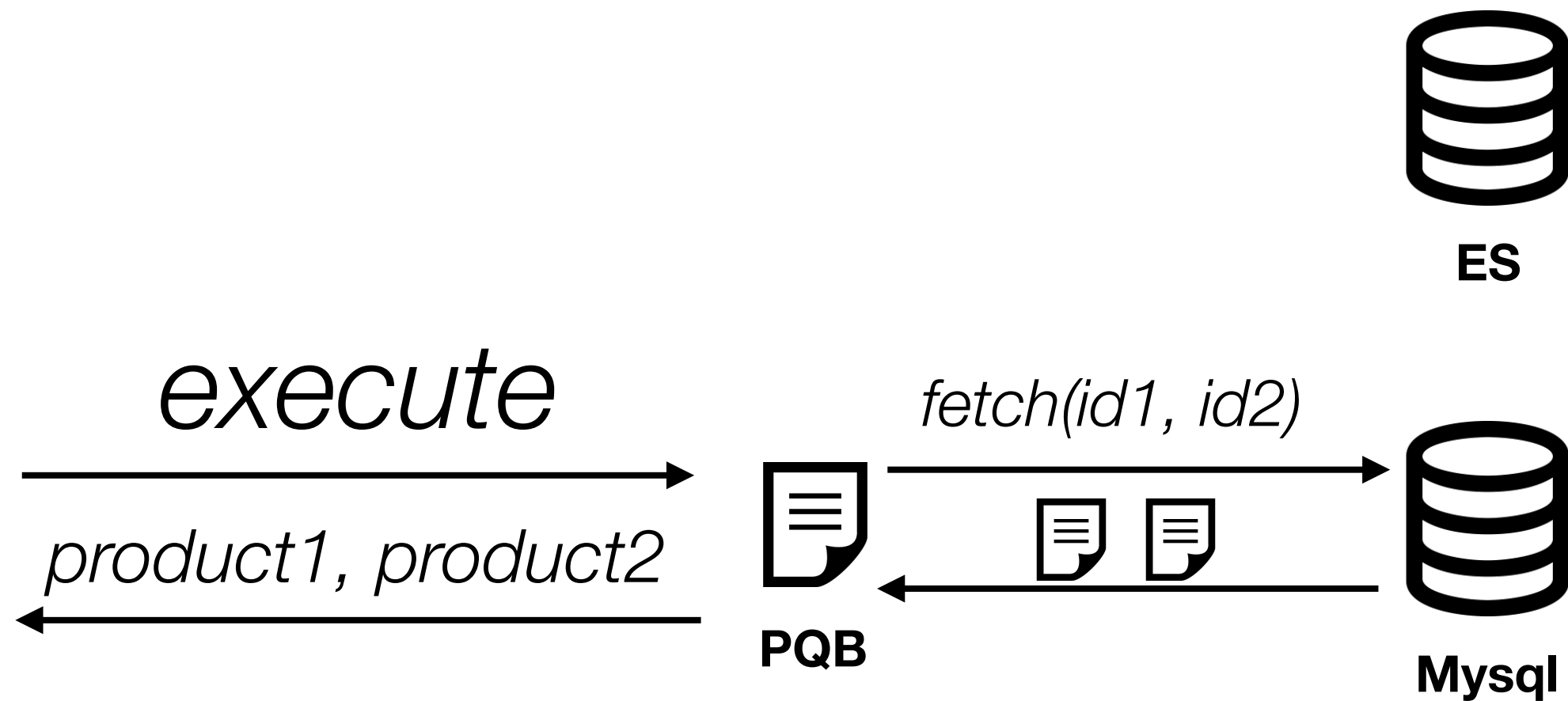
ES & PIM : PQB overview



ES & PIM : PQB overview



ES & PIM : PQB overview



ES & PIM : Compiling the ES search ?

- Thanks to the SearchQueryBuilder
 - ▶ Sqb::addFilter
 - ▶ Sqb::addMustNot
 - ▶ Sqb::addShould
 - ▶ Sqb::addSort
- Sqb::getQuery => Generates a complete ES Request
 - ▶ very useful for Debugging



ES & PIM : How many indexes ?



- 3 for products (p, p&pm, pm)
- 1 published products
- 1 reference entities
- maybe more ?

ES & PIM : Why so many PQBs ?



- Because search use-cases on the UI is not the same as the rest (exports / mass edits, etc.)
- Because it depends on the way you want to iterate
 - ▶ Search after: Given an ID, give me the next IDs
 - ▶ FromSize: from 155, give me next 40

ES & PIM : Testing

- Dedicated integration tests for the mapping
 - ▶ Setup the index with some raw data
 - ▶ Query on it (just like the exercises)
 - ▶ Make sure your mapping works for your search use-cases

ES & PIM : Testing

- Integration tests for each filter of the PQB
 - ▶ Makes sure your filter generates correctly the clauses for the request

*Disclaimer: this is what we **is** done, I maybe would do it differently now*

Any questions ?



ES & PIM : Case study

- the case of "Reference entity full text search-*like*"
- ***Workflow is counter-intuitive***

ES & PIM : Case study

- Workflow:
 - ▶ **1.** Gather the requirements
 - ▶ **2.** Determine how you would want to the request to look like
 - * ***Find the simplest query you could imagine***
 - ▶ **3.** *Find the Mapping that let's you execute this query successfully*

ES & PIM : Case study

- **1.** Requirements

- ▶ *"On the record grid, I would like to enter some words that may correspond to the label, code or any text value the record may have given a reference entity, a channel and a locale"*

ES & PIM : Case study

- **2.** The simplest query ? *(Using fiddling)*

```
{
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "filter": [
            {
              "term": { "reference_entity_code": "MY_REF" },
            },
            {
              "query_string": {
                "default_field": "full_text.ecommerce.fr_FR"
                "query": "*a* AND *few* AND *words*"
              }
            }
          ]
        }
      }
    }
  }
  ...
}
```

ES & PIM : Case study

- **3.** Mapping

```
settings:
  analysis:
    normalizer:
      text_normalizer:
        filter: ['lowercase']
mappings:
  pimee_reference_entity_record:
    properties:
      reference_entity_code:
        type: 'keyword'
    dynamic_templates:
      -
        record_full_text_search:
          path_match: 'record_full_text_search.*.*'
          mapping:
            type: 'keyword'
            normalizer: 'text_normalizer'
```

ES & PIM : Case study

- **3.** Normalization

```
$kartell = [  
  'reference_entity_code' => 'brand',  
  'identifier'            => 'brand_kartell',  
  'record_full_text_search' => [  
    'ecommerce' => [  
      'en_US' => 'kartell' . ' ' . 'Kartell – The Culture of Plastics'...
```

... In just over 50 years, this famous Italian company has revolutionised plastic, elevating it and propelling it into the refined world of luxury. Today, Kartell has more than a hundred showrooms all over the world and a good number of its creations have become cult pieces on display in the most prestigious museums. The famous Kartell Louis Ghost armchair has the most sales for armchairs in the world, with 1.5 million sales! Challenging the material, constantly researching new tactile, visual and aesthetic effects – Kartell faces every challenge! With more than 60 years of experience in dealing with plastic, the brand has a unique know-how and an unquenchable thirst for innovation. Kartell harnesses technological progress: notably, we owe them for the first totally transparent plastic chair, injection moulds, laser welding and more!' . ' ' . 'Philippe Starck',

```
  ],  
],  
];
```


ES & PIM : Case study

- Please, use this workflow:
 - ▶ **1.** Usecases
 - ▶ **2.** Query
 - ▶ **3.** Mapping + normalization



ES & PIM : Going even further

- Documentation:

<https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

- Explanation of the Smart search:

Search_of_products_and_product_models.md (in CE)

- Visualizing Lucene's segment merge:

<http://blog.mikemccandless.com/2011/02/visualizing-lucenes-segment-merges.html>

Last Questions ?

(Don't worry, I'm not going anywhere anyway...)

Feedback



Thank
you