Document relations

Martijn van Groningen
@mvgroningen

elasticsearch.

Overview

- Background
- Document relations with joining.
- Various solutions in Lucene and Elasticsearch

Background - Lucene model

- Lucene is document based.
- Lucene doesn’t store information about relations between documents.
- Data often holds relations.
- Good free text search over relational data.

Background - Common solutions

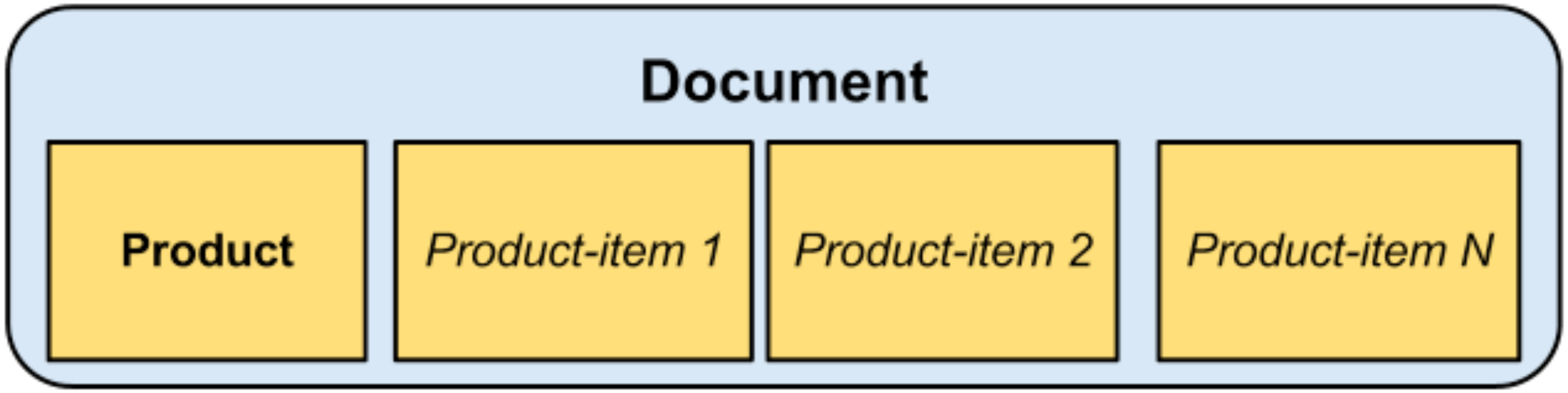
- Compound documents.
    - May result in documents with many fields.
- Subsequent searches.
    - May cause a lot of network overhead.
- Non Lucene based approach:
    - Use Lucene in combination with a relational database.

Background - Example

- Product
  - Name
  - Description
- Product-item
  - Color
  - Size
  - Price

Background - Example

- Compound Product & Product-items document.
- Each product-item has its own field prefix.



Background - Other solutions

- Lucene offers solutions to have a 'relational' like search.
  - Joining
  - Result grouping
- Elasticsearch builds on top of the joining capabilities.
- These solutions aren't naturally supported.

Joining

Joining

- Join support available since Lucene 3.4
  - Not a SQL join!
- Two distinct joining types:
  - Index time join
  - Query time join
- Joining provides a solution to handle document relations.

elasticsearch.

Joining - What is out there?

- Index time:
  - Lucene’s block join implementation.
  - Elasticsearch’s nested filter, query and facets.
    - Built on top of the Lucene’s block join support.
- Query time:
  - Lucene’s query time join utility.
  - Solr’s join query parser.
  - Elasticsearch’s various parent-child queries and filters.

Index time join

And nested documents.

Joining - Block join query

- Lucene block join queries:
  - ToParentBlockJoinQuery
  - ToChildBlockJoinQuery

- Lucene collector:
  - ToParentBlockJoinCollector

- Index time join requires block indexing.

Joining - Block indexing

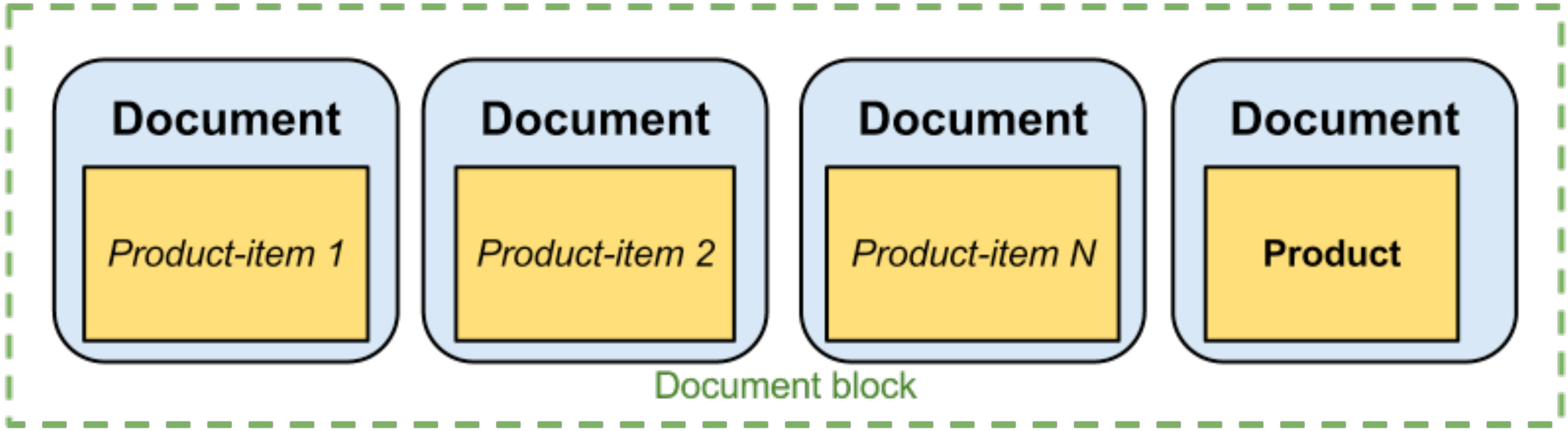
- Atomically adding documents.
  - A block of documents.
- Each document gets sequentially assigned Lucene document id.
- IndexWriter#addDocument<u>s</u>(docs);

Joining - Block indexing

- Index doesn't record blocks.
- App is responsible for identifying block documents.
- Segment merging doesn’t re-order documents in a segment.
- Adding a document to a block requires you to reindex the whole block.

Joining - block join query

- Parent is the last document in a block.



Block join - ToChildBlockJoinQuery

Marking parent documents

```
private static Document createProduct(String name, String description) {
  Document document = new Document();
  document.add(new Field("name", name, TextField.TYPE_STORED));
  document.add(new Field("docType", "product", StringField.TYPE_UNSTORED));
  document.add(new Field("description", description, TextField.TYPE_STORED));
  return document;
}

private static Document createProductItem(String color, String size, int price) {
  Document document = new Document();
  document.add(new Field("color", color, TextField.TYPE_STORED));
  document.add(new Field("size", size, TextField.TYPE_STORED));
  document.add(new IntField("price", price));
  return document;
}
```

Block join - ToChildBlockJoinQuery

```
IndexWriter writer = new IndexWriter(directory, config);
List<Document> documents = new ArrayList<~>();
documents.add(createProductItem("red", "s", 999));
documents.add(createProductItem("red", "m", 1099));
documents.add(createProductItem("red", "l", 1199));
documents.add(createProduct("...Polo Shirt", "Made of 100% cotton..."));
writer.addDocuments(documents);          <-------------------- Add block
documents.clear();

documents.add(createProductItem("light blue", "s", 1999));
documents.add(createProductItem("blue", "s", 1999));
documents.add(createProductItem("dark blue", "s", 1999));
documents.add(createProductItem("light blue", "m", 2099));
documents.add(createProductItem("blue", "m", 2099));
documents.add(createProductItem("dark blue", "m", 2099));
documents.add(createProduct("...White Colored...", "...stripe pattern..."));
writer.addDocuments(documents);          <-------------------- Add block

IndexReader indexReader = DirectoryReader.open(writer, false);
IndexSearcher indexSearcher =  new IndexSearcher(indexReader);
```

Block join - ToChildBlockJoinQuery

- Parent filter marks the parent documents.

```
Query parentQuery = new TermQuery(new Term("docType", "product"));
Filter parentsFilter = new CachingWrapperFilter(
    new QueryWrapperFilter(parentQuery)
);
```

- Child query is executed in the parent space.

```
Query childQuery = new TermQuery(new Term("size", "m"));
ScoreMode scoreMode = ScoreMode.Max;

BooleanQuery mainQuery = new BooleanQuery();
mainQuery.add(userQuery, BooleanClause.Occur.MUST);

ToParentBlockJoinQuery productItemQuery = new ToParentBlockJoinQuery(...);
mainQuery.add(productItemQuery, BooleanClause.Occur.MUST);
TopDocs result = indexSearcher.search(mainQuery, 10);
```

Block join & Elasticsearch

- In Elasticsearch exposed as nested objects.

- Documents are constructed as JSON.
  - JSON's nested structure works nicely with block indexing.

- Elasticsearch takes care of block indexing and also keeps track of the nested documents.

Elasticsearch’s nested support

- Support for a nested type in mappings.
- Nested query.
- Nested filter.
- Nested facets.

Nested type

- The nested types enables Lucene’s block indexing.

index

type

Nested offers

```
curl -XPUT 'localhost:9200/products' -d '{
  "mappings" : {
    "product" : {
      "properties" : {
        "offers" : { "type" : "nested" }
      }
    }
  }
}'
```

Indexing nested objects

index

type

nested objects

```
curl -XPOST 'localhost:9200/products/product' -d '{
  "name" : "Polo shirt",
  "description" : "Made of 100% cotton",
  "offers" : [
       {
         "color" : "red",
         "size" : "s",
         "price" : 999
       },
       {
         "color" : "red",
         "size" : "m",
         "price" : 1099
       },
       {
         "color" : "blue",
         "size" : "s",
         "price" : 999
       }
  ]
}'
```

Nested query

```
curl -XPOST 'localhost:9200/products/product/_search' -d '{
  "query" : {
    "nested" : {
        "path" : "offers",
        "score_mode" : "total",
        "query" : {
            "bool" : {
                "must" : [
                    {
                        "term" : {
                            "color" : "blue"
                        }
                    },
                    {
                        "term" : {
                            "size" : "m"
                        }
                    }
                ]
            }
        }
    }
  }
}'
```

The nested field path in mapping.

Sum the individual nested matches.

Color *red* would match the previous document.

Nested facets

```
curl -XPOST 'localhost:9200/products/product/_search' -d '{
   "facets" : {
      "color" : {
         "terms_stats" : {
            "key_field" : "size",
            "value_field" : "price"
         },
         "nested" : "offers"
      }
   }
}'
```

A facet for nested field offers.

```
"facets":{
    "color":{
        "_type":"terms_stats",
        "missing":0,
        "terms":[
            {
                "term":"s",
                "count":2,
                "total_count":2,
                "min":999.0,
                "max":999.0,
                "total":1998.0,
                "mean":999.0
            },
            ...
        ]
    }
}
```

Counts 2 *nested* documents for term: *s*

elasticsearch.

Query time join

and parent & child relations.

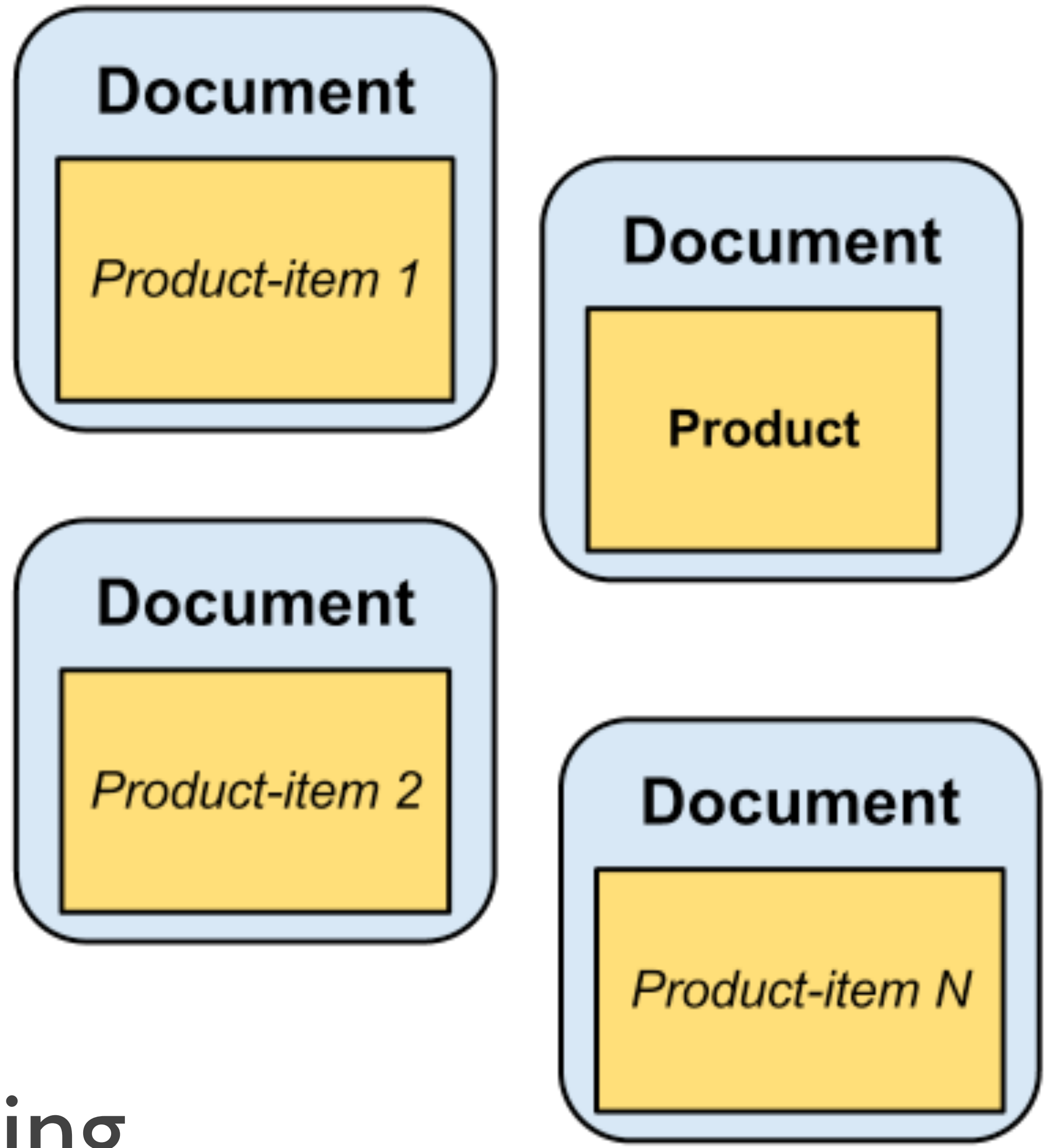
Query time joining

- Documents are joined during query time.
  - More expensive, but more flexible.

- Two types of query time joins:
  - Parent child joining.
  - Field based joining.

Lucene's query time join

- Query time joining is executed in two phases.
- Field based joining:
  - 'from' field
  - 'to' field



- Doesn't require block indexing.

Query time join - JoinUtil

- First phase collects all the terms in the fromField for the documents that match with the original query.
- The second phase returns the documents that match with the collected terms from the previous phase in the toField.
- One public method:
    - JoinUtil#createJoinQuery(...)

Joining - JoinUtil

```
private static Document createProduct(String id, String name, String description) {
  Document document = new Document();
  document.add(new Field("id", id, TextField.TYPE_STORED));
  document.add(new Field("name", name, TextField.TYPE_STORED));
  document.add(new Field("description", description, TextField.TYPE_STORED));
  return document;
}

private static Document createProductItem(String color, String size, int price,
                                          String productId) {
  Document document = new Document();
  document.add(new Field("color", color, TextField.TYPE_STORED));
  document.add(new Field("size", size, TextField.TYPE_STORED));
  document.add(new IntField("price", price));
  document.add(new Field("productId", productId, TextField.TYPE_STORED));
  return document;
}
```

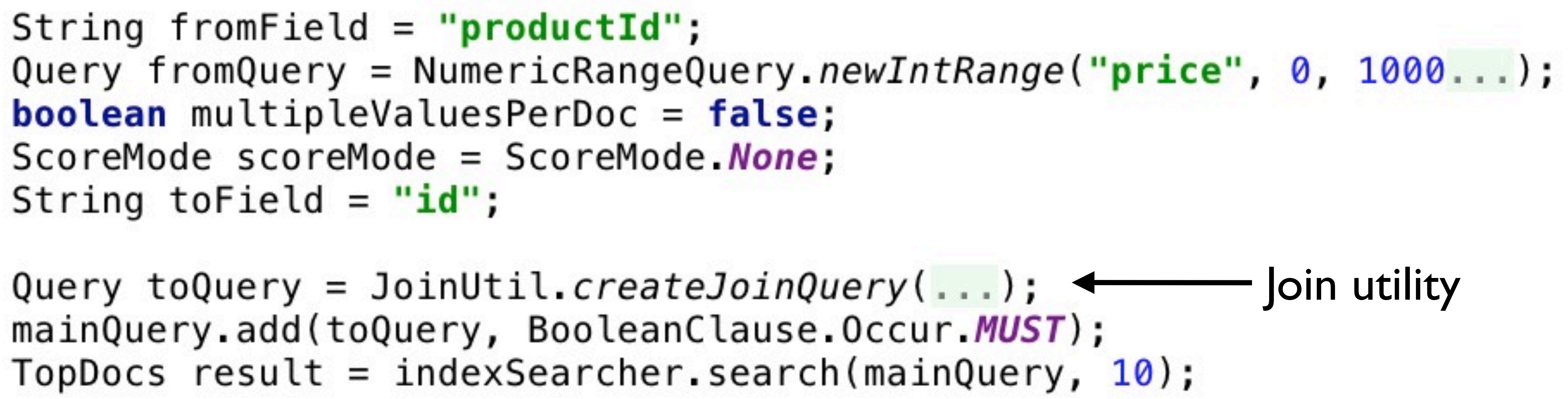
Referrer the product id.

Joining - JoinUtil

```
IndexWriter writer = new IndexWriter(directory, config);
writer.addDocument(
    createProduct("1", "...Polo Shirt", "Made of 100% cotton,...")
);
writer.addDocument(createProductItem("red", "s", 999, "1"));
writer.addDocument(createProductItem("red", "m", 1099, "1"));
writer.addDocument(createProductItem("red", "l", 1199, "1"));

writer.addDocument(
    createProduct("2", "...White Colored...", "...stripe pattern...")
);
writer.addDocument(createProductItem("light blue", "s", 1999, "2"));
writer.addDocument(createProductItem("blue", "s", 1999, "2"));
writer.addDocument(createProductItem("dark blue", "s", 1999, "2"));
writer.addDocument(createProductItem("light blue", "m", 2099, "2"));
writer.addDocument(createProductItem("blue", "m", 2099, "2"));
writer.addDocument(createProductItem("dark blue", "m", 2099, "2"));
```

Joining - JoinUtil



```
String fromField = "productId";
Query fromQuery = NumericRangeQuery.newIntRange("price", 0, 1000...);
boolean multipleValuesPerDoc = false;
ScoreMode scoreMode = ScoreMode.None;
String toField = "id";

Query toQuery = JoinUtil.createJoinQuery(...);
mainQuery.add(toQuery, BooleanClause.Occur.MUST);
TopDocs result = indexSearcher.search(mainQuery, 10);
```

← Join utility

- Result will contain one products.
- Possible to do 'join' across indices.

Elasticsearch's query time join

- A parent child solution.
- Not related to Lucene's query time join.
- Support consists out of:
  - The *_parent* field.
  - The *top_children* query.
  - The *has_parent* & *has_child* filter & query.
  - Scoped facets.

The _parent field

- Points to the parent type.
- Mapping attribute to be define on the child type.

```
curl -XPUT 'localhost:9200/products' -d '{
   "mappings" : {
      "offer" : {
         "_parent" : {
            "type" : "product"
         }
      }
   }
}'
```

- Elasticsearch uses the *_parent* field to build an id cache.
  - Makes parent/child queries & filters fast.

Indexing parent & child documents

- Parent document:

```
curl -XPOST 'localhost:9200/products/product/1' -d '{
   "name" : "Polo shirt",
   "description" : "Made of 100% cotton"
}'
```

- Child documents:

```
curl -XPOST 'localhost:9200/products/offer?parent=1' -d '{
   "color" : "red",
   "size" : "s",
   "price" : 999
}'
```

```
curl -XPOST 'localhost:9200/products/offer?parent=1' -d '{
   "color" : "red",
   "size" : "m",
   "price" : 1099
}'
```

The id of the parent document. Also used for routing.

The 'top_children' query

```
curl -XPOST 'localhost:9200/products/_search' -d '{
   "query" : {
      "top_children" : {
         "type" : "offer",
         "query" : {
            "term" : {
               "size" : "m"
            }
         },
         "score" : "sum"
      }
   }
}'
```

Child type

Child query

Score mode

- Internally the child query is potentially executed several times in order to get enough parent hits.

The ‘has_child’ query

```
curl -XPOST 'localhost:9200/products/_search' -d '{
   "query" : {
      "has_child" : {
         "type" : "offer",
         "query" : {
            "term" : {
               "size" : "m"
            }
         }
      }
   }
}'
```

Child type

Child query

- Doesn’t map the child scores into the matching parent doc. Works as a filter.
- The *has_parent* query matches child document instead.

Scoped facets

```
curl -XPOST 'localhost:9200/products/_search' -d '{
   "query" : {
      "has_child" : {
         "type" : "offer",
         "query" : {
            "term" : {
               "size" : "m"
            }
         },
         "_scope" : "my_scope"
      }
   },
   "facets" : {
      "color" : {
         "terms_stats" : {
            "key_field" : "size",
            "value_field" : "price"
         },
         "scope" : "my_scope"
      }
   }
}'
```

Execute facets inside a specific scope.

Conclusion

- Block join & nested object are fast and efficient, but lack flexibility.
- Query time and parent child join are flexible at the cost of performance and memory.
  - Field based query time joining is the most flexible.
  - Parent child based joining is the fastest.
- Faceting in combination with document relations gives a nice analytical view.

Any questions?