HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Multi-Source Search Engine

with focus on relational Databases

Robin Buchta, Marius Luding, Philip Ohm, Maximilian Senge

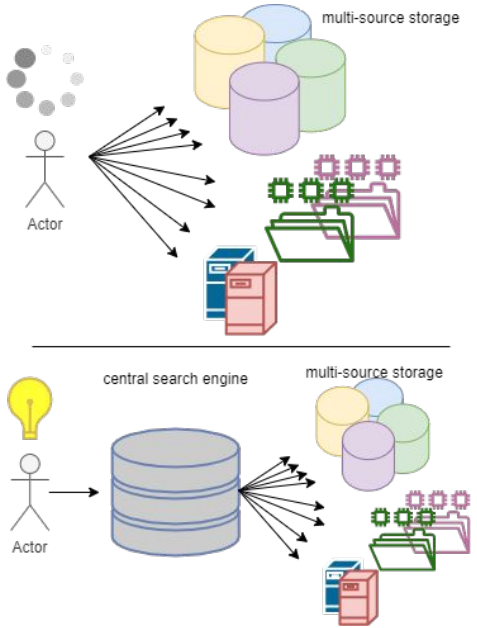
Structure

Introduction - The different Worlds

- SQL databases:
  - Preserves ACID: **a**tomicity, **c**onsistency, **i**solation, **d**urability.
  - Difficult to scale.
  - Fast for the operations: **c**reate, **u**pdate, **d**elete.
- NoSQL
  - Offers BASE: **b**asically **a**vailable, **s**oft state, **e**ventual consistency.
  - Easy to scale.
  - Fast for the operation: **r**etrieve.
  - Many functionalities in the search can be used, offered from some databases.
- Combine these worlds to ensure operational excellence while still taking advantage of the NoSQL world e.g. for filling information needs.

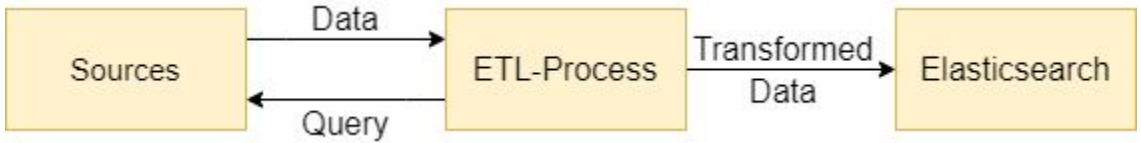
Introduction - Problem and Solution

- No longer need to know the individual locations of the information you are looking for.
- No need to understand the data model, if given.
- No need to know the query syntax of the different source systems.
- Only one location and one query language.
- The operational side remains unaffected.
- The search engine must get and refresh the data.
- The user can be redirected from the search engine directly to the source system.



Structure

Definition ETL-Process



- Stands for Extract, Transform and Load.
- **Extract**: Load data from different sources.
- **Transform**: Transforms the data into the desired format of the target database.
- **Load**: Load the data into die target database (in our case Elasticsearch).
- Process:
  - ETL-Tools queries the data of the source databases.
  - ETL-Tool receives the data.
  - ETL-Tool transforms the data in the desired format.
  - ETL-Tool loads the data to Elasticsearch.

Kind of sources and mapping

- Any kind of sources can be used for the ETL-Process.
- In this work, two kinds of sources are considered:
  - Relational databases:
    - Transform relational data to documents (called mapping).
    - Mapping defines how the index looks like.
  - Events:
    - Using the Elasticsearch REST-API directly.
- Kind of mappings:
  - Implicit mapping:
    - Elasticsearch automatically creates an index -> Good for simple structures and queries.
    - All Attributes are used for creating index -> Maybe Overhead.
  - Explicit mapping:
    - Programmer defines how the mapping looks like -> Individual index.

Relational data mapping (1/3) - Denormalization

- Denormalization of normalized data.
- Each Record is on Document.
  (One Employee works has to roles -> Two documents)
- Select the attributes of interest.
- Historic truth can be archived with:
  - Timestamp,
  - Header,

  attribute.


```
{
    "_index": "employnorm",
    "_type": "doc",
    "_id": "someID",
    "_source": {
        "name": "Christian Koblink",
        "birthday": "12.12.1966",
        "department": "d004",
        "role": "Senior Engineer",
        "hired": "06.06.2012",
        "department.name": "Production"
    }
}
```


- Pros:
  - Simplicity,
  - Good for simple queries,
  - Good Kibana support.
- Cons:
  - Loading data from source is expensive -> Extra load on the database.
  - High memory requirements, as each combinationation is stored redundantly.

Relational data mapping (2/3) - Aggregation + Filtering

- Instead of each row gets an document, each entity (for example employee) gets one document.
- Each document has the attributes of interest (for example role, destination, …).
- Pros:
  - One document per entity,
  - Less overhead when saving data.
- Cons:
  - No out of the box solution for Kibana (we have to specify the queries),
  - Query expensive when execute the mapping process.


```
{
    "_index": "empnested",
    "_type": "doc",
    "_id": "someID",
    "_source": {
        "name": "Christian Koblink",
        "birthday": "12.12.1966",
        "hired": "06.06.2012",
        "roles": [
            {
                "department.name": "Production",
                "role": "Senior Engineer",
                "role.from": "06.06.2012",
                "role.to": "null",
                "department.number": "d004"
            },
            {
                "department.name": "Production",
                "role": "Junior Engineer",
                "role.from": "01.01.2008",
                "role.to": "06.06.2012",
                "department.number": "d004"
            }
        ]
    }
}
```


Relational data mapping (3/3) - Parent and Child

- Approach to archive a kind of normalized structure within a based database.
- Low redundancy, because entities can share attributes.
- Entities are one kind of documents and the attributes of interest are one kind of documents.
- Changing one document can affect other documents -> Consistency.
- Pros:
  - One document per entity,
  - Less redundancy,
  - Consistency.
- Cons:
  - No Out-Of-The-Box solution available,
  - ETL tool needs to be well understood,
  - Kibana support not available, much rework needed.

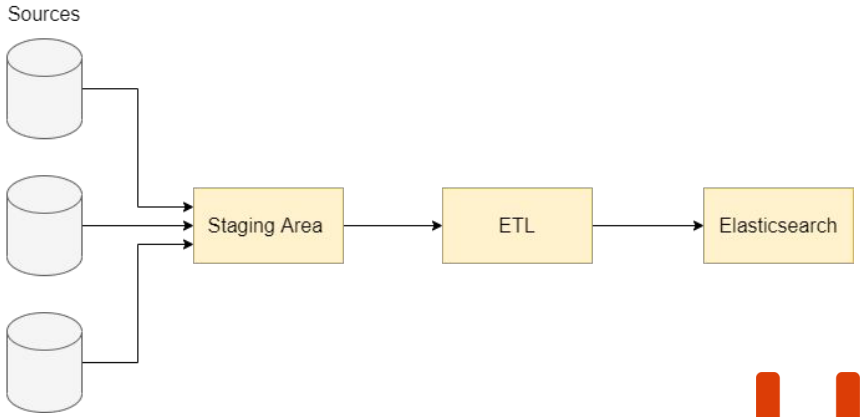

```
{
    "_index": "employnorm",
    "_type": "doc",
    "_id": "someID",
    "_source": {
    "name": "Christian Koblink",
    "birthday": "12.12.1966",
        "employeenumber": "1337",
        "hired": "06.06.2012",
        "department.name": "Production"
    }
},
{
    "role": {
        "hits": {
            "total": 1,
            "hits": [{
                "_type": "doc",
                "_id": "dsome id",
                "routing": "1337",
                "source": {
                    "role.from": "06.06.2012",
                    "role.to": "null",
                    "rname": "Senior Engineer",
                    "enumber": "1337",
                    "dnumber": "d004",
                    "dname": "Production"
                }
            }]
        }
    }
}
```

Syncing Elasticsearch and RDBMS

- **Polling at regular intervals:**
  - ETL-Tool is polling the data in regular intervals.
  - Sourcedata needs timestamps (createdAt, updatedAt).
  - ETL-Tools queries only the delta after the last run -> Performance gain.
  - Pros:
    - Only the delta is loaded,
    - Easy to implement,
    - No logic within the database.
  - Cons:
    - Source data needs timestamp,
    - If interval to big -> no realtime.
- **Use of triggers:**
  - Logic must be moved to the database.
  - Databases must support HTTP-Calls.
  - ETL is done by the triggers, which use the REST-Interface of Elasticsearch.
  - Pros:
    - Realtime,
    - No ETL-Tool needed.
  - Cons:
    - Logic within the database,
    - Debugging of triggers is hard work,
    - Database needs support for HTTP-Calls.

Optimizations

- The ETL-Process can produce extra load on the database.
- This is a general data warehouse problem.
- Solution:
  - Introduce a staging area.
  - Decouples ETL from the sources.
  - 1 by 1 copy -> 3nf.
  - Staging area in best case on another Node.
  - ETL-Process queries staging area.
  - Staging area can be filled by triggers dynamically.

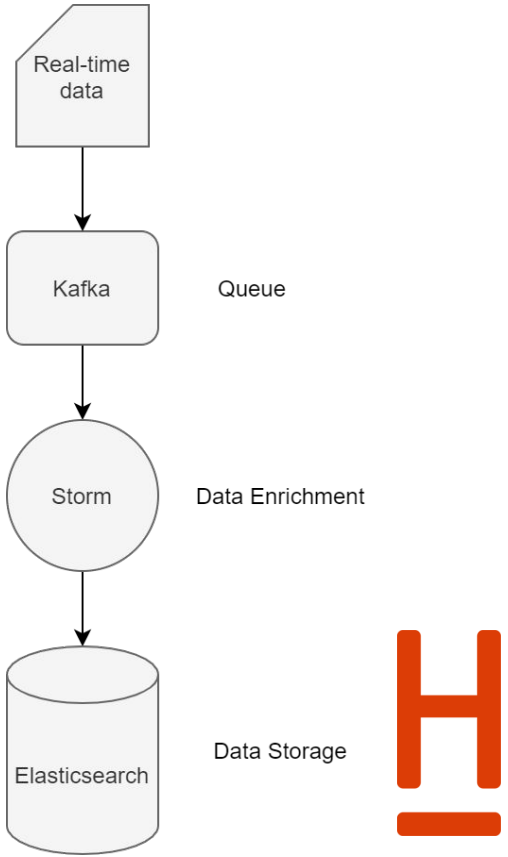


ETL-Tools

- **Logstash:**
  - Belongs to Elastic stack.
  - Easy integration in Elasticsearch workflow.
  - Good Documentation.
  - Scalable.
  - Big community.
  - Open Source.
- **Apache NiFi:**
  - Open Source.
  - High-Level ETL Tool with UI.
  - Scalable.
  - Consistency problems if changing the main node.
- **Apache Spark:**
  - Good for processing streams and batches.
  - Good for big data volumes.
  - Scalable.
  - Not good for small files.

Event based Data insertion

- Data can also be integrated directly into Elasticsearch using the REST API.
- Elasticsearch offers optimizations for this, the Bulk API, so that not every entry has to be indexed individually.
- Instead of adding the data directly, a queue can be used to ensure reliability.
- In addition or only a data enrichment tool can be used to combine several events from one source, or several, before they are stored.



Structure

Chapter 1	Introduction
Chapter 2	ETL
Chapter 3	Searching / Indexing
Chapter 4	Visualisation
Chapter 5	Demo

Searching - Different Data Fields

Structured Data:

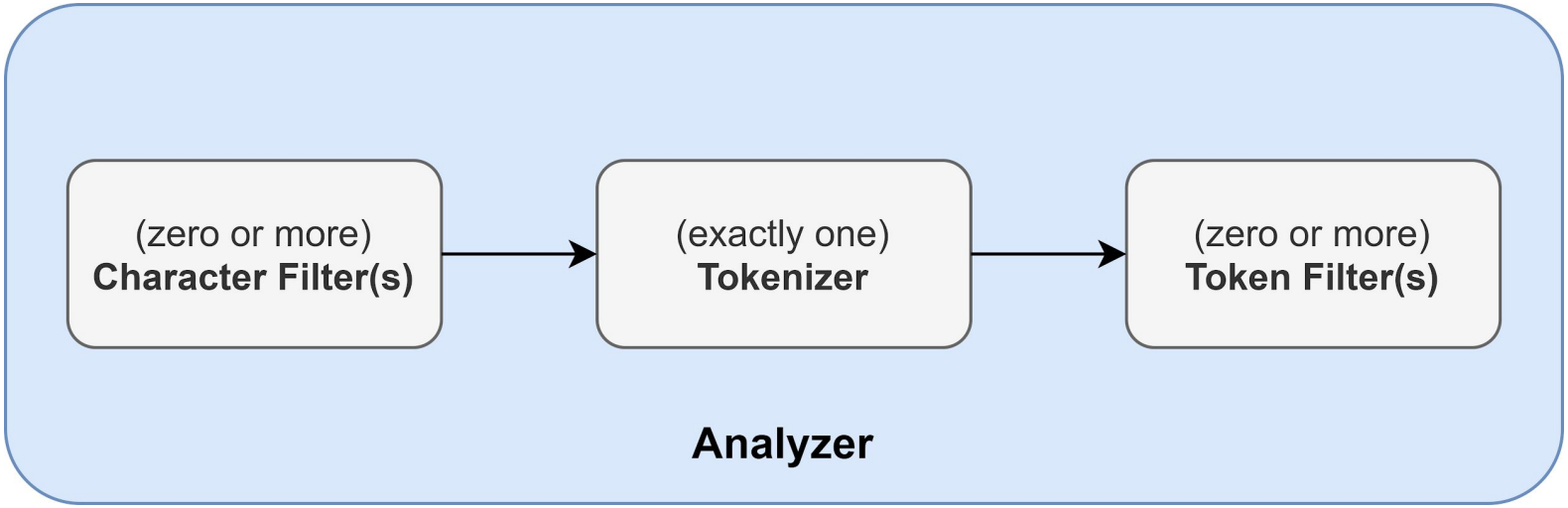
- Numbers, Dates.
- Definitive ordinal relations

  -> Searchable with ranges/exact matches.

Text:

- Can contain combinations of words and their declinations/abbreviations.
- Relevance in search context can depend on occurrences of search terms in a document, changes needed to get a match or other metrics.
- Different Analyzers can be used during ingest and search to account for special requirements.

Searching - Analyzing text



Source: S. Shukla and s. Kumar, Learning Elastic Stack 6.0.Birmingham, UK: Packt Publishing, 2017.

Searching - Query types

- Factors for _score depend on field type and query
- Example query types for numerical data:
  - ‘Range‘ and ‘Term‘ (exact match) will always return a score of 1.
  - ‘Exists‘ can boost the score of a document if a certain field is present.
- Example query types for text:
  - ‘Match’ will look up individual tokens of the query in the inverted index.
  - ‘Match phrase queries’ search for either a complete match of all terms (‘slop’ may be used to leave out n number of terms).
  - ‘Fuzzyness’ defines how many changes may be done on a word to still match the query.

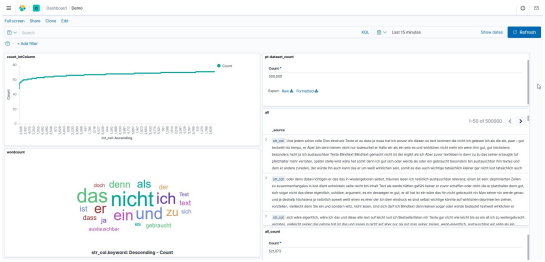

```
{
 ...,
 "hits": {
 "total": 1,
 "max_score": 2,
    "hits": [
        {
        "_index": "library_content",
        "_type": "books",
        "_id": "AV5rBfasNI_2eZGciIbg",
        "_score": 2,
        "_source": {
         "title": "gods & heroes: rome rising",
         ...
        }
       }
    ]
 }
}
```


Structure

Visualization of the search results

- Since Elasticsearch results are a json-object, it can be used anywhere.
- Custom frontend can be build.
- Data in json has to be processed to visualize result.
- Elasticsearch URL needs authentication/authorization:
  - Having access to the URL means being able to do CRUD operations.
  - Old way: A proxy was setup between Elasticsearch and the frontend.
  - This is not necessary anymore, since multiple security features are provided.
- Security features such as encrypted communications, role-based access control and authentication is included in the basic license which is free.
- For auto-completion, Elasticsearch provides completion-suggester feature.
  - Mapping has to be done first, as well as index special fields.
- Elasticsearch also provides Kibana for data visualization.

Kibana



- Web based application.
- Provides many tools to search, filter, manage, analyze and visualize data.
- Targets administrators, business users and analysts.
- Uses the Elasticsearch REST-API.
- Tools which are not based on the X-Pack extension:
  - Discover: Enables user to search through the data with filters by using either the Kibana Query Language (KQL) or Lucene search terms.
  - Visualize: Provides different visualization techniques for data, like charts, tables, heat maps etc.
  - Dashboard: Combines multiple visualizations and searches in one page for analytical purposes.
- Kibana has a plugin API if the provided tools are not sufficient.
- With the X-Pack extension (where some features require a commercial license) additional features such as: Machine Learning, Canvas and Maps are unlockable.

Referencing back to the Source System

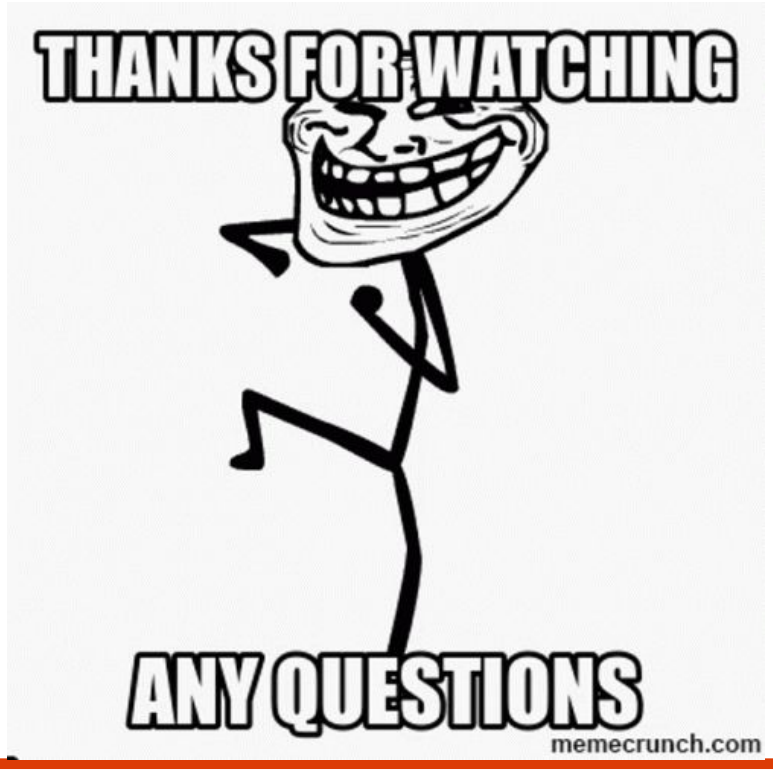
- Data in form of a URL/Path or SQL query has to be included into the result to point to the origin of the source.
- Has to be preprocessed in the frontend.
  - Can visualized as link in the frontend.

Structure

DEMO

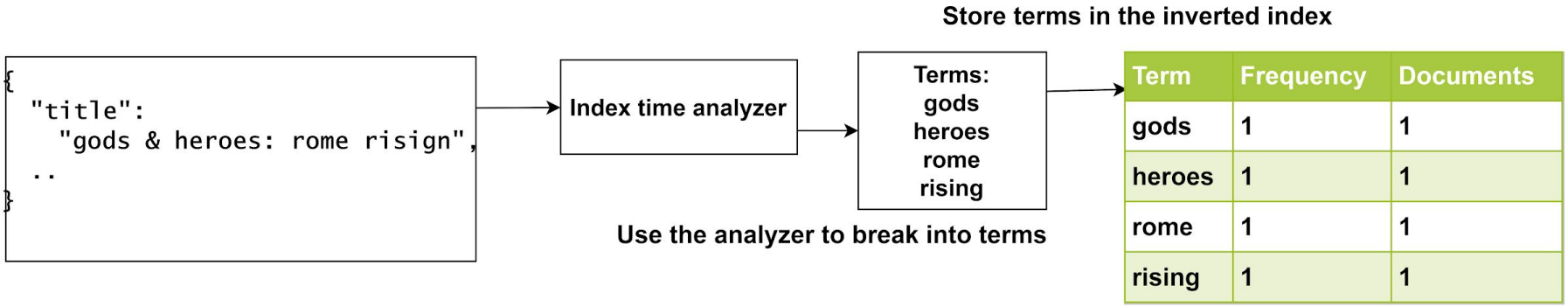

TO START
PRESS ANY KEY
"To start, press any key."
Where's the "any" key?
90s90s90s

Time for questions :-)

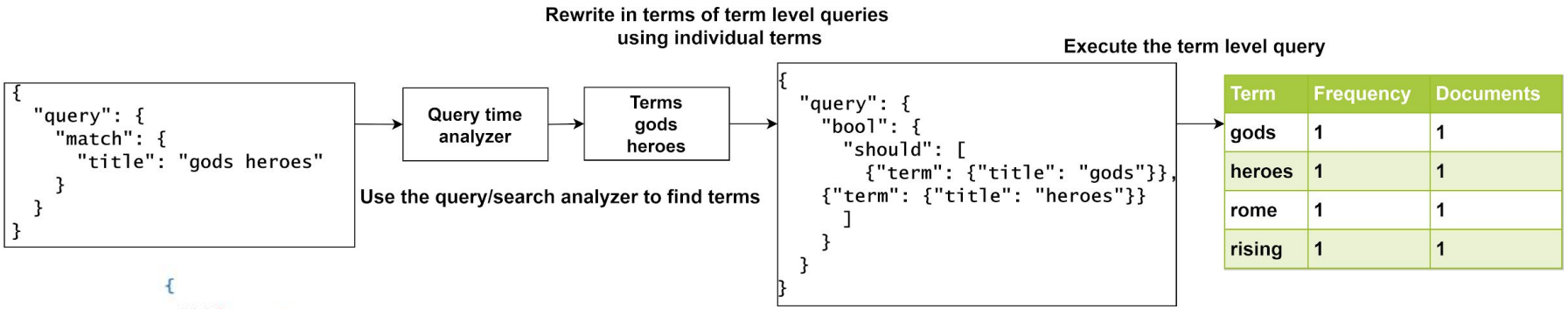


Backup

Searching - Indexing



Searching - Query time




```
{
 ...,
 "hits": {
 "total": 1,
 "max_score": 2,
    "hits": [
         {
         "_index": "library_content",
         "_type": "books",
         "_id": "AV5rBfasNI_2eZGciIbg",
         "_score": 2,
         "_source": {
          "title": "gods & heroes: rome rising",
          ...
         }
         }
    ]
 }
}
```

Searching - Calculating relevance

$$\text{score}(q,d) = \text{coord}(q,d) \cdot \text{queryNorm}(q) \cdot \sum_{t \text{ in } q} \left(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot \text{t.getBoost}() \cdot \text{norm}(t,d) \right)$$

- q = query term.
- d = current document.
- t = individual, tokenized part of the original query term.
- coord(q,d) is higher, if more of the query terms have been found in the document.
- queryNorm(q), makes multiple similar queries more comparable.
- tf(t in d) is the frequency of the term appearing in the document.
- idf(t) relates to how often the term occurs in all documents in the index.
- t.getBoost() can be set at query time to change the importance of a part of the query.
- norm(t,d) normalize certain aspects, such as if a very short field has already exhausted all terms are still met with a high score compared to long fields, where overall a lower percentage is hit.

Source:https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html (retrieved: 22.02.21)