VSB – TECHNICAL UNIVERSITY OF OSTRAVA
FACULTY OF ECONOMICS

DEPARTMENT OF APPLIED INFORMATICS

Návrh a implementace vyhledávacího systému pro data brownfieldů
Design and Implementation of Search System for Data Concerning Brownfields

Student: Bc. Patrik Polaček
Supervisor of diploma thesis: Ing. Roman Danel, PhD.

Ostrava 2021

VŠB - Technical University of Ostrava
Faculty of Economics
Department of Applied Informatics

Diploma Thesis Assignment

Student: **Bc. Patrik Polaček**

Study Programme: N6209 Systems Engineering and Informatics

Study Branch: 6209T017 Informatics in Economics

Title: Design and Implementation of Search System for Data Concerning Brownfields
Návrh a implementace vyhledávacího systému pro data brownfieldů

The thesis language: English

Description:

1. Introduction to Design of Search System for Data Concerning Brownfields
2. Theoretical Foundation and Methodology for System Architecture and Applied Technologies
3. Analysis of Existing Search Capabilities in the Current Version of Search System for Brownfield Related Data
4. Design and Implementation of New Search System for Data Concerning Brownfields
5. Conclusion

Bibliography
List of Abbreviations
Declaration of Utilisation of Results from the Diploma Thesis
List of Annexes
Annexes

References:

CARNELL, John. *Spring Microservices in Action.* Shelter, Island, NY: Manning Publications Co, 2017. 384 p. ISBN 978-1617293986.
WALLS, Craig. *Spring in Action, Fifth Edition.* Shelter, Island, NY: Manning Publications Co, 2018. 520 p. ISBN 978-1617294945.
WONG, Wai T. *Advanced Elasticsearch 7.0: a practical guide to designing, indexing and querying advanced distributed search engines.* Birmingham: Packt Publishing, 2019. 560 p. ISBN 978-1789957754.

I hereby declare that I have elaborated the entire thesis including annexes myself.
I have supplemented the provided annex 1 myself.

Ostrava dated 12.04.2021

Bc. Patrik Polaček

I would like to express gratitude to my supervisor Ing.Roman Danel, Ph.D for all his help and expertise provided during the work on this diploma thesis.

Contents

1 Introduction to Design of Search System for Data Concerning Brownfields 5
2 Theoretical Foundation and Methodology for System Architecture and Applied Technologies 6
2.1 Microsoft SQL Server 6
2.2 Elastic Stack 7
2.2.1 Elasticsearch 8
2.2.2 Kibana 9
2.2.3 Logstash 10
2.3 Full-Text Search 11
2.3.1 Implementation in Elasticsearch 11
2.3.2 Search Queries in Elasticsearch 13
2.3.3 Implementation in Microsoft SQL Server 14
2.3.4 Search Queries in Microsoft SQL Server 16
2.4 Java 17
2.5 Methodology for Creating Search System Architecture 18
2.5.1 Usage of NoSQL Databases 18
2.5.2 Comparison of benefits between SQL and NoSQL databases for brownfield search system 22
3 Analysis of Existing Search Capabilities in the Current Version of Search System for Brownfield Related Data 26
3.1 The Current Data Structure of the Brownfield Database 26
3.2 Entity Relationship Diagram of Existing Brownfield Database 26
3.3 Requirements for Full-Text Search Capabilities of Text Data 27
4 Design and Implementation of New Search System for Data Concerning Brownfields 29
4.1 Search System Implementation Using Elastic Stack 29
4.1.1 Forms of Deployment and Installation of the Elastic Stack 29
4.1.2 Installing Elastic Stack on Personal Workstation 29
4.1.3 Pricing of Elastic Stack 33
4.1.4 Importing Brownfield Related Data into Elasticsearch 36
4.1.5 Elasticsearch General Configuration 38
4.1.6 Defining Search Queries 38
4.1.7 Kibana Search Tools Implementation 43
4.2 Search System Configuration Using Microsoft SQL Server 45
4.2.1 Pricing of Microsoft SQL Server 45
4.2.2 Enabling Full-Text Search in Microsoft SQL Server 46

4.2.3 Examples of Microsoft SQL Server Full-Text Search Queries 48
4.3 Recommending Search System Solution 50
5 Conclusion 53
Bibliography 54
List of Abbreviations 56

1 Introduction to Design of Search System for Data Concerning Brownfields

The main goal of this diploma thesis is to provide solutions for creating a software search system with a focus on text search capability.

Core data that will serve as a basis for search system creation are currently stored in the relational database management system Microsoft SQL Server. Data stored are in text format. The primary requirement is to provide the client with new ways to perform effective searching of text data. Text data describe the best practice methodology of working with brownfields. A new search system will provide the client with a more sophisticated and effective way of searching best practice brownfield methodology and allow them to find the required information in a faster and more user-friendly way.

The client can be in our case characterized as a small research group consisting of different domain experts, students, academic personnel, and private sector employees that participate in the research of solutions for brownfield rehabilitation.

Search system creation as the main topic of the diploma thesis will focus on two main areas. The first area will describe providing text search capability with existing technologies used by the client, mainly Microsoft SQL Server. Analysis of the possibility to implement a search system solution with a focus on text search in Microsoft SQL Server will be conducted from the standpoint of its technological architecture, pricing, and possible implementation. The second area will describe the use of a different set of technologies to provide the required functionality for the client. Elastic Stack and its main component, Elasticsearch has been chosen to provide search capability to fulfil requirements set by the client. Possibilities of integration of Elastic Stack components and their analysis from standpoint of their technological architecture, technology, pricing, and implementation will be performed, similarly to implementation by Microsoft SQL Server.

Solutions described in both areas of the diploma thesis will be compared from a theoretical and practical standpoint and the final recommendation and implementation of the search system will be presented to the client.

2 Theoretical Foundation and Methodology for System Architecture and Applied Technologies

This section describes the set of technologies applied to the design and implementation of a search system concerning brownfield related data. Common methodology and usage of best practices are also discussed in this chapter.

"A brownfield is a property, the expansion, redevelopment, or reuse of which may be complicated by the presence or potential presence of a hazardous substance, pollutant, or contaminant", As defined by (Overview Of EPA's Brownfields Program, 2020).

This diploma thesis is primarily focused on technical architecture and implementation of a search system for data that are being part of the brownfield database, as is more closely described in chapter 3.

2.1 Microsoft SQL Server

Microsoft SQL Server is an RDMS (relational database management system) that holds currently available data related to brownfields. SQL (Structured Query Language) is a standard language used to query and manage relational database management systems. It can be characterised as being declarative by programming paradigm standards. Microsoft SQL Server uses T-SQL (Transact Structured Query Language) as its proprietary extension and a slight modification to SQL.

The current implementation of the brownfield database is done in OLTP (Online Transactional Processing) manner. *The primary focus of an OLTP system is data entry and processing, not data analytics - transactions mainly insert, update, and delete data. The relational model is targeted primarily at OLTP systems, where a normalized model provides both good performance for data entry and data consistency (Ben-Gan, 2016).*

The most recent version of Microsoft SQL Server (2019) is used to hold data about brownfields.

2.2 Elastic Stack

Elastic Stack can be characterized as a set of software applications used for search and analytics solutions. We can import data into Elastic Stack from a wide sphere of data sources and use it for data search, analytics, and visualizations in real-time.

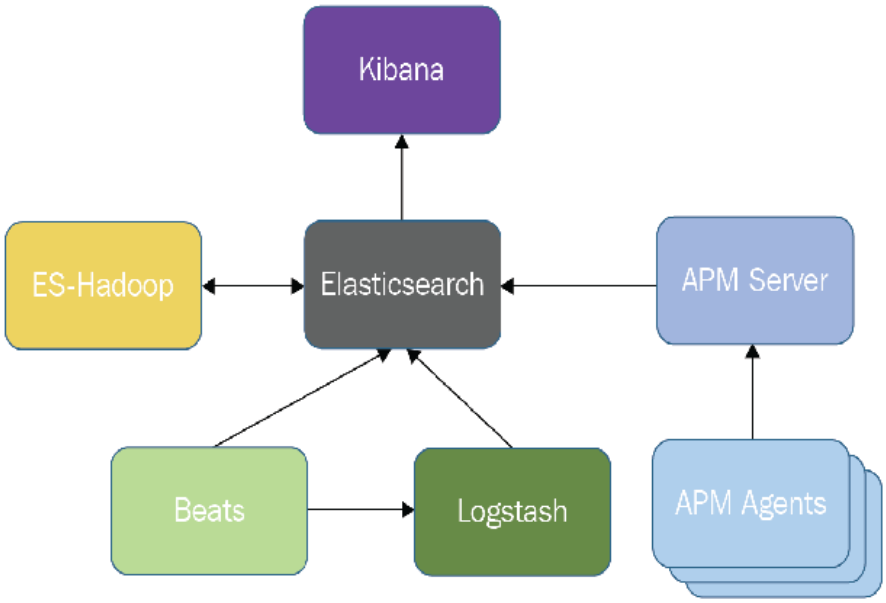

Figure 2.2 Diagram of possible Elastic Stack integrations

Reference: [WONG Wai, 2019]

Figure 2.2 shows some of the different possible applications and technologies from Elastic Stack and connected applications and services which can be integrated together.

This group of application is owned by company Elastic NV, that provides several subscription models and paid support for all its applications. Elastic Stack can be deployed on personal workstations, servers or in the public cloud with providers such as AWS (Amazon Web Services).

Integration of different applications from Elastic Stack is a crucial feature to accommodate for complex data system engineering requirements which generally entail importing data from multiple internal and external sources, transforming the data if necessary and help use the data effectively for analytical, reporting, and other purposes by the users of the search system.

In the following subchapters, we will characterize the main applications and services provided by Elastic Stack used in the development of a search system for brownfields related data.

2.2.1 Elasticsearch

"Elasticsearch is a real-time distributed search and analytics engine with high availability. It is used for full-text search, structured search, analytics, or all three in combination. It is built on top of the Apache Lucene library. It is a schema-free, document-oriented data store", As characterized by Wong (2019, p.85).

Elasticsearch is a NoSQL document-oriented database system, it organizes data into indexes. Indexes are created according to user-specified mapping or automatically based on the character of data by the Elasticsearch engine.

The basic unit of data in Elasticsearch is document. The document consists of JSON (JavaScript Object Notation) data.

We can make certain analogies between relational databases and Elasticsearch concepts based in NoSQL databases. A field in Elasticsearch can be viewed as similar to a column in relational databases. Fields can hold multiple values of specific data types. Filtering in Elasticsearch can be implemented on the level of documents.

Elasticsearch does not provide a comparable concept to a schema in RDBM systems, it is schema-less. The structure of data is dynamically derived from the actual structure of the document. As mentioned before, indexation is done automatically or manually through mapping, which can be viewed as a comprehensive model, or schema of our data. This way of functioning can also be called schema-on-read.

The aforementioned engine can also work in distributed mode as a distributed system, which can allow parallel processing of significant amounts of data for real-time search solutions. From the standpoint of Elasticsearch technological architecture, core concepts can be characterized as:

- Node – unit that is the core instance of Elasticsearch. We can have more than one node. Nodes have general information about each other and can communicate together. There are several types of nodes, such as master, data, ingest and other types of nodes,

- Cluster – a collection of nodes. Can be viewed as the database instance for general RDBM systems. Nodes in a cluster hold data in a specific way as a whole and provide a way for federated indexing, search and other capabilities,
- Index – a collection of documents with similar characteristics. Indexes have associated shards representing physical data organization.
- Shard – a subset of data contained by Elasticsearch index. Each shard can be characterized as a Lucene index. Shards are distributed within each cluster. Document units are stored within shards that are then allocated within specific nodes of a cluster. Elasticsearch balances the number of shards within each node based on the cluster requirements and its changes.

Elasticsearch can be viewed as a near real-time search solution. There is a slight delay between indexing a new document and it being available for search. Speed and adaptability to new content is a key focus of the Elasticsearch platform. This characteristic can be useful in the future when new requirements may arise to accommodate for increasing data volumes and new data insertions within the brownfield database.

2.2.2 Kibana

Kibana can be characterized as a visualization tool and UI interface for Elasticsearch. This technology is able to provide a GUI (Graphical User Interface) management tool for a whole range of applications and services from Elastic Stack. Elasticsearch can be fully managed from the command line, Kibana can be viewed as an alternate solution to work with Elasticsearch with the use of GUI.

Data visualization, reporting, management, and other Elastic Stack functions can be provided through Kibana not only for search system architects but also directly for customers.

Mapping of documents from Elasticsearch index to Kibana is done through index pattern, which represents mapping that integrates document structure with Kibana search platform. This integration allows us to define a wide variety of filters for our search queries defined within Kibana search tools. Kibana uses KQL (Kibana Query Language) or Lucene Query Syntax as search languages for performing search queries. The semantics of both query languages and Elasticsearch query DSL (Domain Specific Language) defined in chapter 2.3.2 are very similar.

2.2.3 Logstash

Logstash is an open-source data collection engine that provides a real-time data processing pipeline from multiple sources and then sends the collected logs to a defined target, such as an Elasticsearch server. The data processing usually includes conversion, unification, and normalization (Wong Wai, 2019).

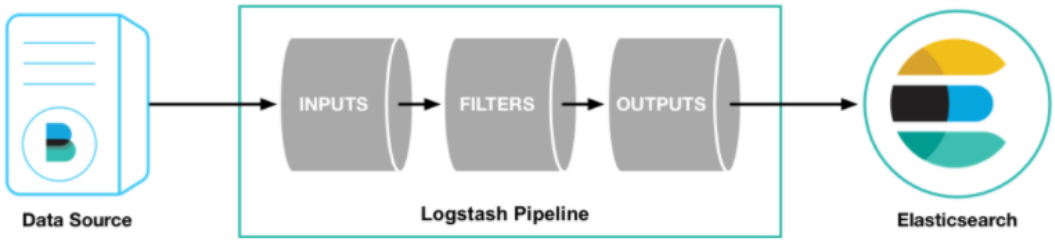

Figure 2.2.3 Architectural overview of Logstash
Reference: [Stashing Your First Event, 2020]

As figure 2.2.3 shows, Logstash has three core parts:

- Inputs – representing data sources Logstash can digest data from. Logstash supports dozens of data sources, including usage of JDBC (Java Database Connectivity) API (Application Programming Interface), which allows us to use Microsoft SQL Server as a data source and many other RDBM systems and different services.
- Filters – representing plugins with the ability to modify input data for conversion of data types and formats between input and output system, data normalisation or other processing tasks. This layer is optional.
- Outputs – representing different system compatible with Logstash with the ability to intake its data. Elasticsearch as part of the Elastic Stack is a dominant output system of Logstash.

We can define more than one input source. The whole architecture of the Logstash engine can be characterised as a data pipeline between source and target systems. This engine can process data in real-time as they are generated within the source system.

Logstash can be configured through a configuration file, where we specify the input and the output system, data processing logic and event handling.

2.3 Full-Text Search

Full-Text search can be described as a procedure for searching text data. Search engine performing this procedure generally retrieves all units representing given text data. Units that are retrieved, analysed, and used by the search engine are called tokens. Tokens can be thought of as words. There are different strategies for extracting tokens, some are language specific. A search engine is a set of software libraries that define the fundamental search logic.

2.3.1 Implementation in Elasticsearch

Elasticsearch provides robust end to end full-text search capabilities. Apache Lucene library is an open-source java search engine used by Elasticsearch and works as a core component of its full-text search system. Text data imported into Elasticsearch are transformed into documents. "*An inverted index is created by tokenizing the terms in the document, creating a sorted list of all unique terms, and associating the document list with the location where the terms can be found. An index consists of one or more shards. A shard is a Lucene index that uses a data structure (inverted index) to store data", as described by Wong (2019, p.85).*

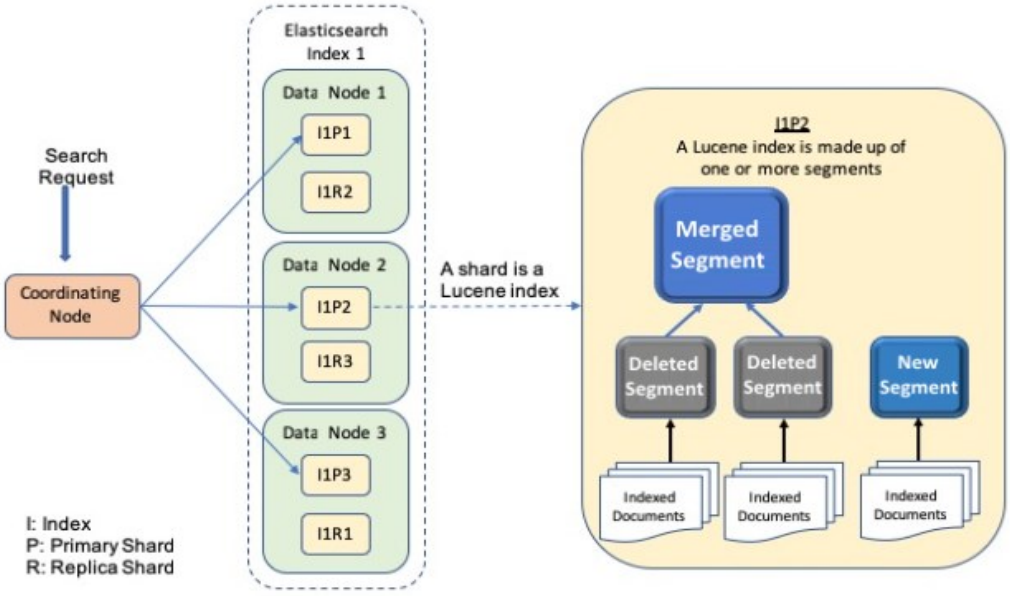

Figure 2.3.0 Diagram of Lucene index architecture within Elasticsearch
Reference: [WONG Wai, p.87]

Location in inverted index points to documents in which tokenized term can be found. One specific term can be present in more than one document. Generally, all indexes point to documents contained within a specific Elasticsearch index.

Shards are contained within different nodes and can be replicated. Two or more same replicas cannot be placed in the same node for the sake of node failure mitigation. Each shard is a Lucene index and uses the inverted index as a data structure to store how are tokenized terms allocated within different documents.

One Lucene index can store more than one inverted index, in form of inverted index segments, or parts of the whole index segment. Elasticsearch is built to process continual data additions in form of new documents. For this purpose, different segments of inverted indexes allocated within one Lucene index can be merged to one inverted index inside the Lucene index. After data representing inverted index are persisted to disk, old segments of inverted indexes are deleted. All segments of Lucene indexes inside shards associated with the given Elasticsearch index will be searched on every new search request by a client. This process is fully automatic and represents low-level functioning of data allocation and full-text search implementation in Elasticsearch and is depicted by figure 2.3.0.

Elasticsearch implements different techniques for full-text search in form of analysers. Analysers define the general procedure of transforming text data into tokens stored in the inverted index.

The analyser has three main components *WONG (2019, p.80)*:

- *Character Filter: Receives the raw text as a stream of characters and can transform the stream by adding, removing, or changing its characters.*
- *Tokenizers: Splits the given streams of characters into a token stream.*
- *Token filters: Receives the token stream and may add, remove, or change tokens.*

Character filters are mainly used to pre-process streams of characters before tokenization for purposes such as transforming characters from one language to another, changing HTML (HyperText Markup Language) elements into text representation or custom regular expression character replacements.

There are many pre-defined algorithms for tokenization in Elasticsearch that can be categorized into several groups. Word oriented tokenizers include standard, letter,

lowercase, whitespace, and other algorithms. Partial word tokenizers use N-Gram or Edge N-Gram algorithms. Structured text tokenizers include a keyword, pattern, char group, path, and other procedures. Tokenizers determine when to split sequences of words into tokens.

Token filters are most often used to transform tokens produced by tokenizers to standardized form, for example to all lowercase letters to potentially increase search performance. Another common use case is to remove stop words that are defined for most languages to optimize for data volume storage.

2.3.2 Search Queries in Elasticsearch

Elasticsearch supports many different categories of queries and uses its own query domain-specific language based on JSON format to define them. Some of the categories are described below, mainly the ones used in our search engine implementation.

Match Query

This type of query returns documents containing specified text, number, date or Boolean parameter. Elasticsearch compares queried value to terms, term creation is defined by the specific analyser. We must specify fields we want to search within documents of the existing Elasticsearch index. Match query also supports fuzzy parameter matching, which can greatly improve the user experience while searching by returning results even in a situation where the specified search value was typed incorrectly.

"When querying text or keyword fields, fuzziness is interpreted as a Levenshtein Edit Distance — the number of one character changes that need to be made to one string to make it the same as another string", As is defined by (Common options, 2021).

"Levenshtein distance (LD) is a measure of the similarity between two strings, the source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform s into t. The greater the Levenshtein distance, the more different the strings are", As characterised by (Haldar and Mukhopadhyay, 2011)

Fuzzy searching can be configured in several different modes, mode AUTO will automatically choose the maximum number of term edits that can be done by the engine when searching for a specific term. The number of edits will be determined from the length of queried value. We can also manually specify the number of edits.

Multi-Match Query

Multi-match query can be described as a modification of Match Query that allows us to specify more than one field of Elasticsearch document for search inside a specific Elasticsearch index. This type of query also supports fuzzy search functionality.

Both match and multi-match query can be configured to have specific functionality when searching for sequence of words and not only for one word. Default option *or* will search for every word match in different fields even when the original ordering of words in the sequence is not maintained. Option *and* will search only for a combination of all words in exact sequence as they were defined. Our search system will use default *or* option unless specified otherwise.

2.3.3 Implementation in Microsoft SQL Server

This chapter describes the architecture of full-text search as implemented within Microsoft SQL Server. Microsoft SQL Server provides a possibility to define full-text indexes on different character-based columns of Microsoft SQL Server database. Supported data types of columns for the full-text index are *char, varchar, nchar, nvarchar, text, ntext, image, xml, or varbinary(max) and FILESTREAM, as stated by (Full-Text Search, 2020).* There can only be one full-text index defined per table.

Microsoft SQL Server allows us to define full text queries that search within specified columns with full-text index defined, these columns are also called documents. In case of the positive result between searched for value or phrase and values within the indexed document, Microsoft SQL Server returns all documents containing specified queries or documents. A positive match is called a hint within the Microsoft SQL Server context.

Full-text queries are generally not case sensitive and allow to search for proximity matches between words, similarly to fuzzy search queries defined in chapter 2.3.2.

A whole range of possible full-text query options are *(Full-Text Search, 2020)*:

- *One or more specific words or phrases (simple term)*
- *A word or a phrase where the words begin with the specified text (prefix term)*
- *Inflectional forms of a specific word (generation term)*

- *A word or phrase close to another word or phrase (proximity term)*
- *Synonymous forms of a specific word (thesaurus)*
- *Words or phrases using weighted values (weighted term)*

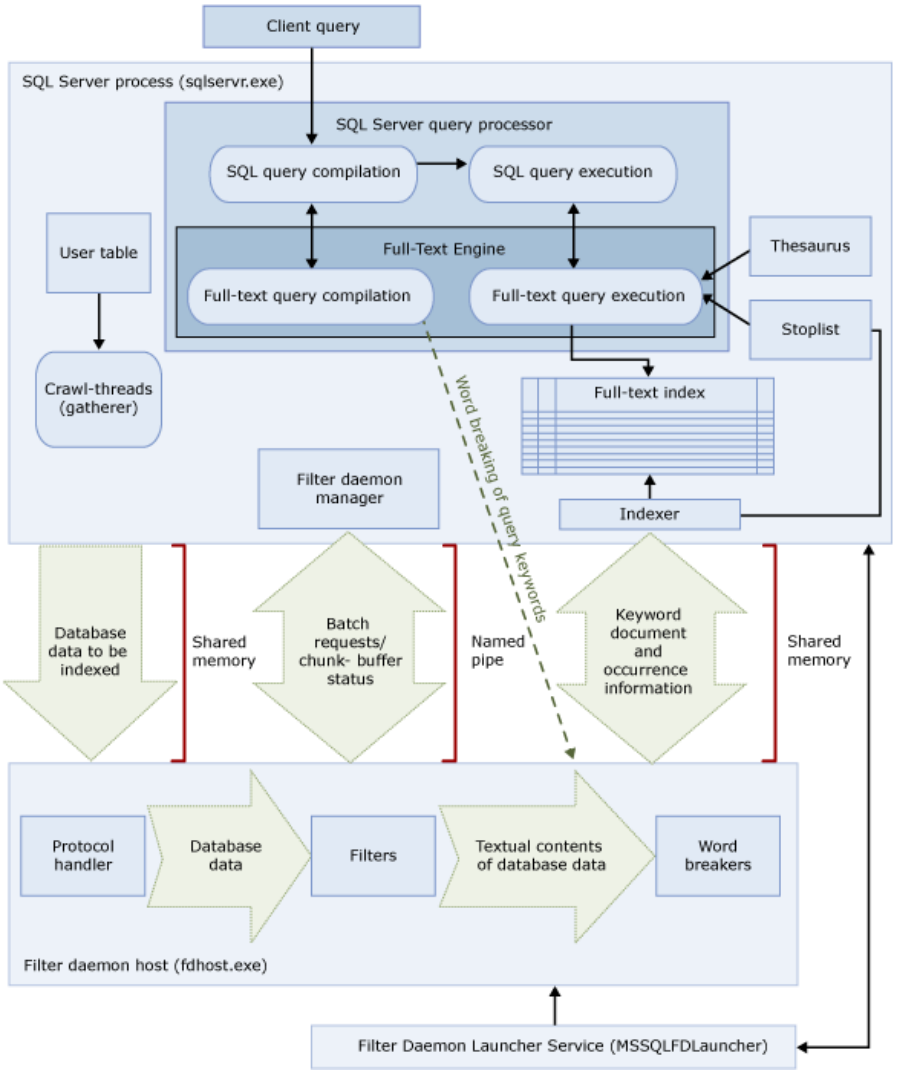

Figure 2.3.1 Overview of Microsoft SQL Server full-text search architecture
Reference: [Full-Text Search, 2020]

Different concepts of full-text search are implemented through different processes, as shown in figure 2.3.1. The core full-text capability is implemented with the *SQL Server process (sqlservr.exe)*. User tables represent tables containing data to be used for full-text search. *Gatherer crawl-threads* are used to populate full-text indexes with data from designated compatible columns in user tables, this can happen periodically or after detecting changes – for example, new insertions into user tables. *SQL Server query*

processor compiles and processes user full-text queries. Core processing logic happens in the *full-text engine* during compilation and query processing by *SQL Server query processor*. During processing, *thesaurus* and *stoplist* objects are used to enhance full-text search. *Thesaurus* can be used to identify synonymous word to queried values. *Stoplist objects contain a list of common words that are not useful for the search (Full-Text Search, 2020). Stoplist* objects are similar to token filters and are used to optimize stored data volume by removing not useful words for full-text search. *Indexer* manages the full-text index.

Additional data filters are maintained by separate daemon processes managed by the *Filter Daemon Manager*. Filters provide functionality for processing specific data formats like excel, word or XML documents. Another functionality of filters is word breaking and stemming. This set of functionalities provides the ability to split specified columns into different terms that can be then stored within full-text indexes pointing different terms to their respective location within documents, similarly to Lucene indexation in Elasticsearch as more closely described in chapter 2.3.1.

2.3.4 Search Queries in Microsoft SQL Server

Microsoft SQL Server provides several predicates to work with full-text queries. This chapter describes the most common predicates and their basic functioning.

Contains

This predicate matches single word and phrases with precise or fuzzy matching (Query with Full-Text Search, 2017). It is a similar concept to match query used by Elasticsearch defined in chapter 2.3.2. We can specify the proximity of words to be accepted as hits, the weighting of matches according to proximity to searched for value and additional chaining with other logical commands such as AND, OR, NOT operators.

Freetext

Freetext predicate matches the meaning of searched for words, phrases, or sentences and not only the exact wording (Query with Full-Text Search, 2017). It matches any terms or forms of any terms that are contained within specified full-text indexed columns. Forms of terms are generated through stemming or converting words into their inflectional base forms. Base forms are also expanded, replaced and modified in different ways in consideration of values from the thesaurus.

2.4 Java

Java can be characterized as an object-oriented programming language. Java was created by James Gosling while working for Sun Microsystems. Some of the main characteristics and paradigms can be characterized as:

- Object-oriented
- Imperative
- Reflective
- Static and Strong typed, nominative,
- Supporting Multithreading

"Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code", As described by Schildt (2019, p.74)

Class-based object-oriented language like Java consists of classes that can be viewed as definitions or blueprints for creating objects that can communicate with each other. Communication between objects is a crucial principle of object-oriented languages. The fundamental paradigm used by object-oriented languages is an abstraction, allowing us to translate complex real-world systems into a digital, ephemeral state of software programs. Encapsulation, inheritance and polymorphism complete the set of the fundamental paradigms of object-oriented programming.

The imperative nature of Java allows us to define in detail operations performed by computer programs by using statements that can modify the state of the program.

Reflection allows the program to some extent intersect in its own structure after its execution. Java allows for partial reflection. We can introspect or gain meta-information about certain features of Java program on runtime, however, our ability to modify its structure or behaviour is limited.

A strong typing system makes sure every variable and expression have a type that is strictly defined. Due to the definition of the typing system as being static, the compiler checks enforcement of constraints for each type on compile time.

Java uses a combination of compilation and interpretation. As part of the JRE (Java Runtime Environment), we can find Java Compiler and Java Virtual Machine. Java

Compiler first compiles code instructions into bytecode, which is then interpreted by Java Virtual Machine. This allows for the portability of our programs into different platforms, without the need to write native code for different hardware platforms and promotes security. There are exceptions to this rule and Java can also be directly compiled into native code.

Java also supports multithreading or a process that allows us to define an arbitrary number of threads to work within our application. This attribute of Java allows parallel computation, communication, and other computational and programming features to be used.

Elasticsearch and other Elastic Stack application are based on Java programming language, Apache Lucene search engine is an open-source data search engine also based on Java programming language, for these reasons the author of the diploma thesis decided to dedicate one chapter for Java language basic definition.

2.5 Methodology for Creating Search System Architecture

This section describes the common principles, rules and methodology of core disciplines to which technologies used to provide solutions for search system belong to. The main focus is on the dichotomy between NoSQL and SQL databases and technologies belonging to these respective groups – Elasticsearch as part of the Elastic Stack and Microsoft SQL Server.

2.5.1 Usage of NoSQL Databases

NoSQL is an abbreviation for not only SQL. Some of the basic characteristics of NoSQL databases are, as stated by *MEIER and KAUFMANN (2019, p.202)*:

- *"The database model is not relational.*
- *The focus is on distributed and horizontal scalability.*
- *There are weak or no schema restrictions.*
- *Data replication is easy.*
- *Easy access is provided via an API.*
- *The consistency model is not ACID (instead, e.g., BASE)."*

The main focus of a NoSQL database can be characterized as being able to effectively process an extensive amount of data. Core need for NoSQL databases can be

traced to the early 2000s and the commercialization of the internet around the world which led to new requirements in amounts of data needed to be effectively processed. Working with significant amounts of data is not the only benefit we can expect from NoSQL databases. Different advantages come from various core types of NoSQL databases.

Different types of NoSQL databases:

- Key-Value Stores – use key and value pairs to store data and are mostly used for fast query performance. Keys are identifying data structures bound with value data object containing actual data. Key-Value stores do not support referential integrity and are often used in form of in-memory databases, storing data in RAM with replicas stored on hard drives. They are distributed by subsets of keys that are stored on different shards. Well known key-value stores are Redis, DynamoDB and other NoSQL databases.

- Column-Family Stores – are partially based on concepts used by key-value stores. Column-family stores optimize read queries by columns and not by rows to accommodate for the existence of more complex business entities with a significant number of columns of which only a subset is needed for retrieval. They store data in column family structure representing a subset of columns of a specific table which is part of the row key, a specific key containing column family and a row of the table. "*The advantages of column-family stores are their high scalability and availability due to their massive distribution, just as with key-value stores. Additionally, they provide a useful structure with a schema offering access control and localization of distributed data on the column family level", as stated by MEIER and KAUFMANN (2019, p.215).* Column families serve as integrity structures with different access rules and other options. Some of the popular Column Family stores are Apache Cassandra and HBase.

- Document stores – Elasticsearch is a document-oriented store and is more closely described in chapter 2.2.1. Document stores are another subset of key-value stores. The core unit of data storage in document stores are documents. Documents are usually in JSON format containing data. Each

document has an associated id working as a key. Documents are distributed and replicated in different shards within different nodes of the system, as is more closely described in the context of full-text search in chapter 2.3.1. One of the mechanisms for maintaining near real-time search capability in a distributed environment is to retrieve a document with the highest numbers of changes. If the client queries document NoSQL database that should retrieve specific documents, the document with the highest numbers of changes is retrieved even when this change is potentially not yet replicated within all document replicas. For this purpose, each document has an associated number of changes. The main advantages of document store databases are easy replication and database scalability, ability to process significant amounts of unstructured and heterogeneous data thanks to their schema-less structure and flexible document format. Document stores do not offer referential integrity of data. Apart from Elasticsearch, NoSQL databases such as MongoDB, DocumentDB, CosmosDB and other can be characterised as document store NoSQL database systems.

- XML, graph and other types of NoSQL databases.

The fundamental requirements of relational database management systems are to always maintain consistency and data integrity. RDBM systems use to compel with specified requirements set of transaction rules called ACID, which stands for atomicity, consistency, isolation, and durability.

NoSQL databases have a different set of general requirements to fulfil such as processing significant amounts of data, ability to process heterogeneous data, high flexibility, be able to scale horizontally according to usage, work with unstructured data formats and other. To fulfil listed requirements, NoSQL databases use a different set of rules labelled as the BASE, which stands for basically available, soft state, eventually consistent.

As a result of the distributed nature of NoSQL databases, where several parts of the database system, usually nodes, store data, data insertions or changes take time to propagate within all parts of the database system (nodes), which may lead to data retrieval

returning database state not representing the most actual data representation. This attribute of NoSQL databases is described as eventual consistency.

Singular nodes within the database system are available for data retrieval, this characteristic is labelled as basic availability. Due to data being eventually consistent, data retrieval from basically available nodes may return data not representing the most actual state of the database, this state can be described as a soft state of the NoSQL database system.

Distributed nature allows NoSQL database systems to process data parallelly which can lead to making data processing effort much more effective than in relational databases. BASE rules deal with maintaining consistency across a distributed system.

CAP theorem describes the potential for achieving fundamental properties of a distributed storage system. Three attributes making CAP theorem are listed below, as characterized by *MEIER and KAUFMANN (2019, p.135)*:

- *"Consistency (C): When a transaction changes data in a distributed database with replicated nodes, all reading transactions receive the current state, no matter from which node they access the database.*
- *Availability (A): Running applications operate continuously and have acceptable response times.*
- *Partition tolerance (P): Failures of individual nodes or connections between nodes in a replicated computer network do not impact the system as a whole, and nodes can be added or removed at any time without having to stop operation."*

CAP theorem states, we can only achieve two of the three properties. Different combinations are consistency and partition tolerance (CP), consistency and availability (CA) and availability and partition tolerance (AP). Graphical representation of CAP theorem property combinations is shown in figure 2.4.1.

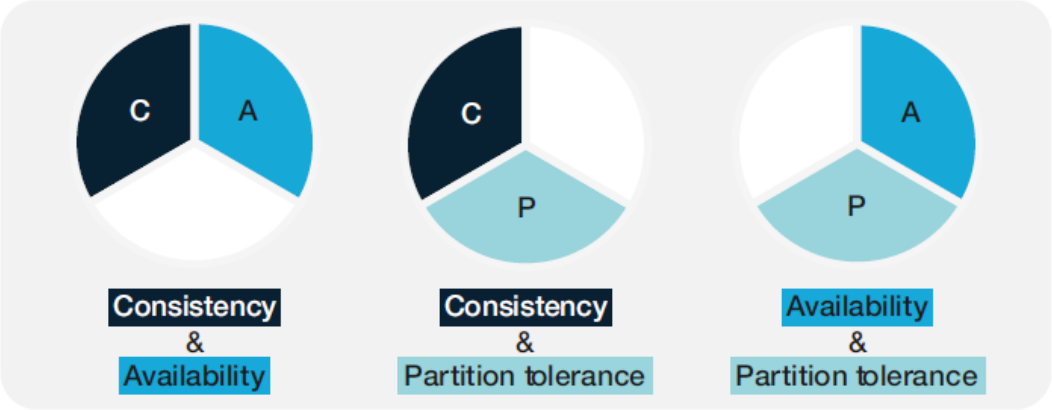

Figure 2.5.1 Combination of different achievable properties defined by the CAP theorem

Reference: [MEIER and KAUFMANN, p.135]

Elasticsearch is focused primarily on achieving availability and consistency by the definition of the CAP theorem. Consistency is maintained in form of eventual consistency due to distributed node architecture of Elasticsearch. Partition tolerance cannot be fully achieved with Elasticsearch and Elastic Stack in general. We can imagine situations in which enough nodes could fail which could lead to complete system failure of Elastic Stack. Even in case of system failure, Elastic Stack still responds to users indicating an inability to process requests, due to this characteristic response availability is maintained.

The aforementioned attributes of Elastic Stack are beneficial for the creation of a search system. We can maintain near real-time search capabilities of significant volumes of data, which could fulfil future requirements emerging after the brownfield database increases in data volume size.

2.5.2 Comparison of benefits between SQL and NoSQL databases for brownfield search system

The general definition of NoSQL databases provided in chapter 2.5.1 describes the focus and attributes of NoSQL databases, mainly its distributed nature, schema-less architecture, working with unstructured data formats and other. Relational SQL databases focus on maintaining the integrity and defined relationships of data entities with pre-defined structure. Search system and mainly the full-text search function which provides the ability to search unstructured text data as described in chapter 2.3 can be implemented in both SQL and NoSQL databases. In the following subchapter, we will compare the

main differences between Microsoft SQL Server, an SQL database and Elasticsearch, a NoSQL database.

Microsoft SQL Server and Elasticsearch

Microsoft SQL server described in chapter 2.1 provides full-text search capability more closely described in chapter 2.3.3. Microsoft SQL server is already used to host brownfield related data, full-text capabilities can be configured within existing Microsoft SQL Server instance. This configuration does not require the usage of new technology by the client, unlike in the case of Elasticsearch. The initial source of data for full-text search is also contained within the Microsoft SQL Server database, which makes integration of source data and full-text search capability easier, and we do not generally need ETL (Extraction-Transformation-Load) or ELT (Extraction-Load-Transformation) tools for achieving our full-text search goals. Elasticsearch implementation requires additional integration ETL/ELT tools like Logstash described in chapter 2.2.3. The advantage of using external integration tools can be an increased range of possible data integrations from different internal or external sources, which can be useful when requirements for data search system as described in chapter 3 develop to accommodate for more complex search solution. ETL/ELT tools can be independently integrated to also import data into Microsoft SQL server. Elasticsearch provides the benefit of bundled deployment within the Elastic Stack platform, which contains Logstash for integration and Kibana for data visualisation, reporting and other front-end data analytics functions. Microsoft SQL Server can also be integrated with data visualisation and reporting tools, however, Kibana provides GUI usable by end-users to perform full-text search, unlike Microsoft SQL Server, this can be considered as an advantage to using Elastic Stack in comparison with Microsoft SQL Server, where we would need to make specific frontend application. In some cases, making custom frontend service for clients would be required despite the discovery function and other frontend reporting tools offered by Kibana, as more closely described in chapter 4.1.7.

Elasticsearch, part of Elastic Stack as described in chapter 2.2, provides full-text search as one of its core functionalities, more closely described in chapter 2.3.1. Full-text capabilities are similar for general use cases between Microsoft SQL Server and Elasticsearch. Elasticsearch has a more robust configuration and range of options to customize the full-text search. Elasticsearch being a NoSQL database provides wider

options for data processing, especially in the domain of big data, sometimes characterised as Four or Five V´s of big data.

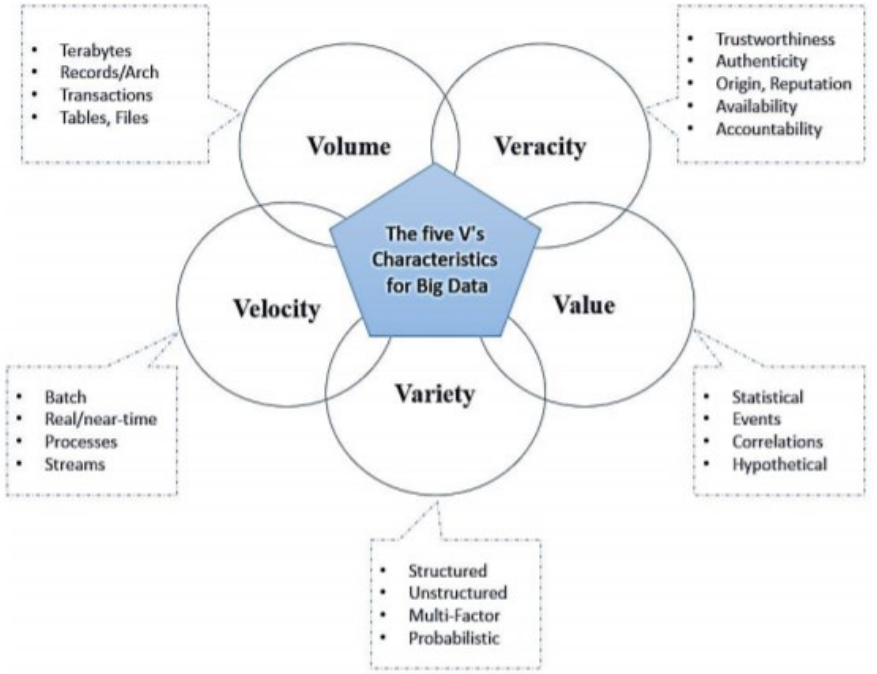

Figure 2.5.2 Diagram of core attributes of Big Data
Reference: [Hiba & al, 2015]

Currently, information about best practices of working with brownfields consists of text data. We can consider this format to be unstructured data. Elasticsearch and its schema on read architecture allow us to work with unstructured data formats with different attributes and provides benefit for a potential increase in source data formats of data concerning brownfields. This provides greater flexibility for the future and is generally the attribute of a *variety* of big data as shown in Figure 2.5.2.

Another benefit of Elasticsearch could be working with high-velocity data that are generated constantly over certain time periods, which would allow different external data to be integrated on a real-time basis into the brownfields search system. However, Microsoft SQL Server is also able to handle periodical data imports. The important comparison is the size of data being generated and the general data volume expected to be handled by the database. If future requirements arise to handle vast volumes of data on a periodical and constant basis to be imported into the brownfields search system, Elasticsearch could be better options to maintain a higher level of user experience through faster data retrieval.

From the perspective of implementation, Microsoft SQL Server and other RDBMS are older solutions that are more widely known in practice than NoSQL technologies like Elasticsearch and finding domain experts to configure respective systems may be easier for Microsoft SQL Server implementation. The deployment of an Microsoft SQL Server can be generally considered simpler and requiring less configuration than Elasticsearch. The main reason behind increased complexity is distributed nature of Elasticsearch and its wider range of possible functionalities. Distributed nature of Elasticsearch can provide clients with more robust near real-time data replication functionality than replication possibilities of Microsoft SQL Server and is a core requirement for all applications within Elastic Stack.

3 Analysis of Existing Search Capabilities in the Current Version of Search System for Brownfield Related Data

The current state of the search system for brownfield relate data can be characterised by an RDBMS Microsoft SQL Server that holds data describing different brownfields located in Moravian-Silesian Region in the Czech Republic. We can find 21 tables in this database describing geographical, technical, environmental, and other categories of data concerning brownfields. The database is also holding data about former login information of users accessing database data through RDBMS.

3.1 The Current Data Structure of the Brownfield Database

We can find various data types in the brownfield database, some of the most common categories that are used by the Microsoft SQL Server and can also be found in the database are *(Data types (Transact-SQL), 2021)*:

- *Exact numeric data types such as tinyint, int, bigint, money;*
- *Character strings data types such as char, varchar, text;*
- *Date and Time data types such as date, smalldatetime, datetime;*
- *Other data types such as Unicode Character strings and Binary strings.*

Not all applicable data to our system are stored inside the current RDBMS system. Some specific data types such as images in JPEG format are only stored by their name and reference to their external physical location in a different storage medium.

3.2 Entity Relationship Diagram of Existing Brownfield Database

ERD diagram (Entity-relationship diagram) can provide us with a visual representation of relational database architecture. This diagram allows us to see different entities and their attributes, relationships, and their cardinality.

"An entity is a "thing" which can be distinctly identified. A specific person, company, or event is an example of an entity. A relationship is an association among entities. For instance, "father-son" is a relationship between two "person" entities", As stated by Chen (1976, p.10). In our case entity is represented by each relation/table in our relational database.

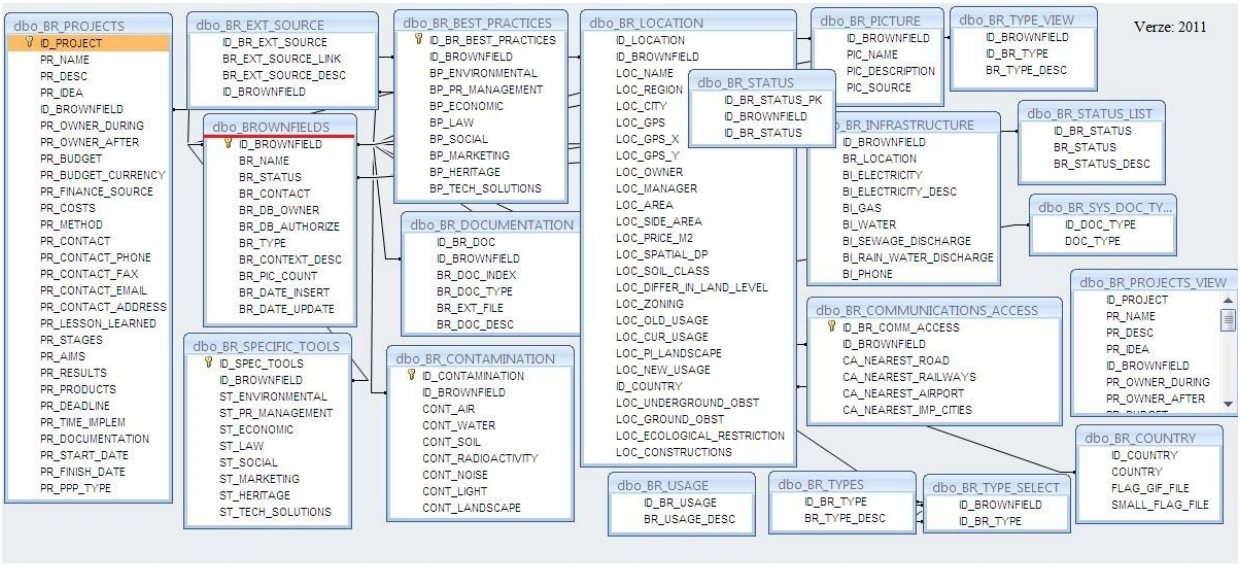

Figure 3.1 ERD Diagram of Brownfield Database

Reference: [Own Creation]

3.3 Requirements for Full-Text Search Capabilities of Text Data

Most of the tables inside the brownfield database have several attributes with varchar(max) data type. This specific data type holds wide ranges of text-based data. Users of the database concerning brownfield data would benefit from the comprehensive search capability of text-based information stored in the database.

In the current state of the search system, we have only access to the aforementioned relational database holding data in a specified way, without any applicable methods, procedures, or holistic strategies for giving users useful information from search results of text-based data.

One of the main goals of this diploma thesis is to find effective ways of providing users with useful results from a search system focusing primarily on text data in the brownfield database. We can use different methods provided by an existing RDBMS Microsoft SQL Server, but also to try new technologies and applications like Elasticsearch, Grafana or other database management systems.

The client currently uses his own version of the search system without any full-text specific search functionality as described by this diploma thesis and has his own frontend built to perform a search of the brownfield database. Search results of the clients own search system are not satisfactory to effectively search in the brownfield database

which created the initial need of the client to request a more comprehensive full-text search capability solution for the brownfield database. The client can however use his own frontend to work with specific Microsoft SQL Server configuration and predicates as described by the following chapters, which leads to having no requirement to create a frontend solution for the search by the Microsoft SQL Server as part of this diploma thesis. No parts of the clients existing search system were used in this diploma thesis due to not providing any real full-text search solution, except for the relational database holding brownfield related data.

Special attention is focused on the table concerning best practices for working with Brownfields, where search capabilities would be most needed by users. Providing a more effective way of searching through best practice methodology descriptions will be the general focus of this diploma thesis.

4 Design and Implementation of New Search System for Data Concerning Brownfields

This section describes the design and implementation of a new search system using Elastic Stack and Microsoft SQL Server.

4.1 Search System Implementation Using Elastic Stack

4.1.1 Forms of Deployment and Installation of the Elastic Stack

Different applications and services of Elastic Stack – Elasticsearch, Kibana, Logstash and other can be installed on Linux, Windows, or other OS environments. Elastic Stack also supports containerization, in form of Docker or Kubernetes deployment. Elastic Stack applications can be installed on local machines, workstations, servers, or in the cloud environment. Public cloud implementations can be done manually or on pre-defined implementation platforms offered by Amazon Web Services, Microsoft Azure or Google Cloud, the biggest public cloud providers.

4.1.2 Installing Elastic Stack on Personal Workstation

Installation of Elastic Stack applications on a local workstation can be done directly or with the use of virtual environments and containerization platforms. This type of usage, installation and deployment is useful for development and testing purposes. Simple search systems can be implemented in this type of environment. Complex systems requiring much greater hardware resources and configuration considerations would need to be deployed in a server environment.

Installing Logstash on Windows Workstation

General recommendation on how to install Logstash on Windows machines can be found at *(Running Logstash on Windows, 2021).*

To install Logstash on Windows OS, we need to have pre-installed Java JDK (Java Development Kit) and properly set up environmental variables to be able to invoke JDK from the command line. We also need to download and extract Logstash binaries to our workstation (Running Logstash on Windows, 2021).

Logstash can be then run manually from the command line after invoking the Logstash batch file. We can additionally specify the location of the Logstash configuration file by using flag f.

Example of command to start Logstash, as specified by *(Running Logstash on Windows, 2021)*:

```
C:\logstash-7.11.1> .\bin\logstash.bat -f .\config\syslog.conf
```

We can also specify additional flags to alter the runtime characteristics of Logstash. This command starts Logstash and uses a configuration file, an example of using a configuration file for our search system can be found in chapter 4.1.4. Logstash can also be run as a process, to allow for periodical checkouts of updates in data sources we want to import data from into Elasticsearch, and then import these newly updated data into Elasticsearch.

Installing Elasticsearch on Windows Workstation

Elasticsearch can be installed similarly to Logstash. *To install Elasticsearch on a Windows workstation, we need to download and extract Elasticsearch binaries to a directory (Install Elasticsearch with .zip on Windows, 2021).*

Elasticsearch on default uses bundled JDK but can be configured to use different JDK, configuration of Elasticsearch is done through different configuration files.

Files used to configure Elasticsearch are (Configuring Elasticsearch, 2021):

- *elasticsearch.yml for configuring Elasticsearch*
- *jvm.options for configuring Elasticsearch JVM settings*
- *log4j2.properties for configuring Elasticsearch logging*

Within specified files, we can define the runtime settings of Elasticsearch in terms of hardware resource allocation. We can also define cluster size, index and other types of settings for Elasticsearch. These settings can be to some extent specified also in Kibana GUI after Kibana has been integrated with Elasticsearch.

Elasticsearch can be started by invoking batch configuration file, as specified by *(Install Elasticsearch with .zip on Windows, 2021):*

```
.\bin\elasticsearch.bat -Ecluster.name=my_cluster Enode.name=node_1
```

The command above uses additional options for the configuration of Elasticsearch directly from the command line, in this case, we specify cluster and node name. The same settings can be applied within configuration files of Elasticsearch.

Installing Kibana on Windows Workstation

To install Kibana on a Windows workstation we need to download and extract binaries to a directory (Install Kibana on Windows, 2021).

Kibana can be then started by invoking batch file, as specified by *(Install Elasticsearch with .zip on Windows, 2021):*

```
.\bin\kibana.bat
```

Configuration of Kibana can be done with the use of kibana.yml configuration file, located on default in $KIBANA_HOME/config directory (Configure Kibana, 2021).

Important configuration setting useful to integrate Elasticsearch with Kibana is elasticsearch.hosts, which specifies the host URL on which Elasticsearch is running on the local machine.

Deploying and Using Elasticsearch and Kibana from Docker Container

An integrated combination of Elasticsearch and Kibana can be deployed on a Windows workstation in form of a docker container. The first step of this process is to have Docker installed on the workstation.

The next step consists of creating a docker-compose file that specifies docker container images and their integration. Example of a docker-compose.yml file, as defined by (*Running the Elastic Stack on Docker, 2021).*

```
version: '2.2'
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:{version}
    container_name: es01
    environment:
      - node.name=es01
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es02,es03
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - elastic
  es02:
    image: docker.elastic.co/elasticsearch/elasticsearch:{version}
    container_name: es02
    environment:
      - node.name=es02
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es03
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data02:/usr/share/elasticsearch/data
    networks:
      - elastic
  es03:
    image: docker.elastic.co/elasticsearch/elasticsearch:{version}
    container_name: es03
    environment:
      - node.name=es03
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es02
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data03:/usr/share/elasticsearch/data
    networks:
      - elastic
  kib01:
    image: docker.elastic.co/kibana/kibana:{version}
    container_name: kib01
    ports:
      - 5601:5601
    environment:
      ELASTICSEARCH_URL: http://es01:9200
      ELASTICSEARCH_HOSTS: '["http://es01:9200","http://es02:9200",
                                                          "http://es03:9200"]
    networks:
      - elastic

volumes:
  data01:
    driver: local
  data02:
    driver: local
  data03:
    driver: local
```

Figure 4.1.0 Example of docker.compose.yml file for Elastic Stack deployment

This configuration sets up three Elasticsearch nodes within one cluster. Kibana is integrated with all three nodes and its GUI is running at localhost:5601, as shown in figure 4.1.0.

4.1.3 Pricing of Elastic Stack

The price of using applications from Elastic Stack and their combinations differs between deployment types.

Elasticsearch, Kibana and Logstash are all open-source applications. They can be used at no cost and their source code can be freely accessed via GitHub at URL address https://github.com/elastic. Usage at no cost does not offer any direct support and does not contain all functionalities that can be provided by commercial paid for licenses.

The direct cost of using most Elastic Stack applications comes in form of infrastructure and support cost. Development of complex systems built upon Elastic Stack applications can have additional costs in form of the labour cost of domain experts – software and data engineers, analysts, quantitative experts, product and project managers and others. Additional costs can be associated with general change, R&D, governance, operations, managerial and other areas associated with building and effectively using software products or offering software services and maintaining them in the future.

Infrastructure cost is deduced from the price of servers and platforms Elastic Stack applications are deployed in. Support for Elastic Stack applications is priced based on its level. Required infrastructure and associated costs can differ between each application from Elastic Stack – Kibana, Elasticsearch and Logstash,

Elasticsearch B.V, the company owning Elastic Stack applications offers Elastic Cloud – the platform for Elastic Stack deployment where customers can choose a specific public cloud provider – Amazon Web Services, Microsoft Azure or Google Cloud whose infrastructure will be used for the deployment. Customers can also specify data canter and availability zone for each of the public cloud providers.

Given the open-source nature of Elastic Stack applications, there are other Elastic Stack services provided by the same public cloud providers that are however not connected to Elasticsearch B.V, for example, Amazon Elasticsearch Service or Amazon Elasticsearch. Pricing is different between Elastic Cloud and Amazon services and offers different features, structure, and levels of support.

The price of usage differs between public cloud providers and is determined by the size of data, requests, required functionality and other parameters. Different providers have different levels of hardware bundling, it is important to know how each provider documents their deployment process.

Elasticsearch B.V offering

We need to first determine the amount of data we are planning to import into Elastic Stack on daily basis over a specific period. Let us assume a small company or a research group that generates a maximum of 500 MB of data daily, or 0.5 GB. Generated data can be from multiple sources, for example from changes in relational databases that are then imported into Elastic Stack, which is the current use case in our brownfield database. Other common sources of data can be application logs, results from monitoring applications and other. The company wants to create a deployment for one year period.

For estimation of memory requirements for deployment in question, we can compute several characteristics about Elasticsearch, as described by (*Benchmarking and sizing your Elasticsearch cluster for logs and metrics, 2020)*

- *"Total Data (GB) = Raw data (GB) per day * Number of days retained * (Number of replicas + 1)*
- *Total Storage (GB) = Total data (GB) * (1 + 0.15 disk Watermark threshold + 0.1 Margin of error)*
- *Total Data Nodes = ROUNDUP(Total storage (GB) / Memory per data node / Memory:Data ratio)"*

In our case,

Total Data required (GB) would be 0.5 * (12*30) * 2, in total 360 GB. Assuming we would import 0.5GB of data per day for a total of 360 days. The number of replicas is in this case set to 1, which is a minimum requirement for running Elastic Stack with general fault tolerance present. *Total Storage (GB) would be* 360 * (1 + 0.15 + 0.1), in total 450 GB. This characteristic takes into consideration the margin of error for data allocation and recommended levels of disk usage, also called watermark levels. *Total Data Nodes* can be calculated as 450GB/8GB/30 which equals 2 nodes. Memory to data ratio is calculated from options provided by Elasticsearch B.V on the amount of physical storage and of RAM memory per node. The most aligned option for our use case is 240 GB of storage with 8GB of RAM per data node. This accounts for the ratio of 30.

The price of hourly deployment for a configuration consisting of two Elasticsearch nodes each with 240 GB of storage and 8 GB of ram in two availability zones would be

0,455$. This hourly price would account for 10,92$ daily, 327,6$ monthly and 3931,2 $ for 360 days or one year of continuous deployment.

Two 1 GB nodes allocated for Kibana would cost hourly addition 0,0594$, 2GB of memory should be sufficient for general reporting abilities, performing search queries and using the discovery function. Two nodes allocated within two different availability zones would provide basic fault tolerance ability for Kibana.

The total price of deployment would be 0,5147$, which accounts for 12,35$ daily, 370$ for 30 days – one month and 4440$ per 360 days of continual deployment using Elasticsearch B.V offering on AWS servers using standard edition support option. All options have free of charge APM monitoring instance present used for monitoring of resource usage by our Elastic Stack deployment. The Standard edition consists of ticket service management support from Elasticsearch B.V for our deployment. Standard edition also provides the complete core functionality of Elastic Stack.

For purpose of maintaining brownfield related methodology data that would be in its current state, without further need to process any significant amount of data on daily basis, the price of the deployment would be significantly decreased. 30 GB of total storage distributed within two availability zones while maintaining the same level of Kibana deployment would cost 0,1381$ when deployed on AWS public cloud servers in the Frankfurt region.

This cost would propagate to 3,3144$ daily, 99,432$ monthly and 1193,184$ yearly of continuous Elastic Stack usage.

Other infrastructure costs to consider are follow up costs for additional years of deployment while maintaining historical data. In the case of primarily static data deployment that does not generate data periodically, we can predict the approximately same level of pricing for the following years. When we need to maintain historical data from the first year of deployment and still need to generate new data with certain velocity and volume, we will need more memory and a higher number of nodes for our deployment, which will increase the hourly price of our deployment in following years. Changes in hourly pricing may also arise as a result of a change in the pricing policy of the deployment provider.

Other public cloud offerings

Elastic stack deployments from public cloud providers such as Amazon Elasticsearch Service and other public cloud providers are generally more complex to configure. They can provide synergies when used with other deployments in the public cloud environment, which are not applicable in their current form for a small company or public research institution. The complexity of this configuration and additional labour cost in form of cloud architecture expertise would make this option inefficient for our use case requiring search system implementation that is smaller in scale.

Self-managed Elastic Stack deployment

Elastic Stack could be deployed on Linux or Windows server without any form of support. Initial installation and integration of Elastic Stack services could be done by an external domain expert. An external entity would also manage requests for change of deployment and different Elastic Stack applications by the company. General time allocation for this external entity would be in a matter of hours per month. Hiring an internal employee with domain expertise would be likely an ineffective and relatively expensive solution due to the limited size of the brownfield database in its current state.

4.1.4 Importing Brownfield Related Data into Elasticsearch

Logstash configuration file has been set to accept input from RDBMS Microsoft SQL Server and import data into Elasticsearch.

```
input {
    jdbc {
        jdbc_driver_class => "com.microsoft.sqlserver.jdbc.SQLServerDriver"
        jdbc_connection_string => "jdbc:sqlserver://**;databaseName=cobraman_db;"
        jdbc_user => "**"
        jdbc_password => "**"
        statement => "select * from dbo.BR_BEST_PRACTICES"
       }
}

output {
  elasticsearch {
    hosts => ["**"]
    index => "db_brownfield"
    #user => "**"
    #password => "**"
  }
}
```

Figure 4.1.1 Logstash configuration file

Logstash configuration file from Figure 4.1.1 has been set up to connect to Microsoft SQL Server with residing Brownfield database. Input settings specify JDBC connection attributes like JDBC driver class, JDBC database connection string, username and password of a database client account with proper authorization to read data from Brownfield database. Statement attribute specifies SQL query performed by Logstash. The result from this SQL query will be then imported into the Elasticsearch instance. Mapping to the Elasticsearch document is automatically done by Elasticsearch.

Output settings specify host URL (Uniform Resource Locator) of Elasticsearch instance, username and password for Elasticsearch client account. The index specifies the pre-defined Elasticsearch index use by Elasticsearch for document specification and shard allocation as defined more closely in chapter 2.2.1.

Attributes of the configuration file specifying private information have been replaced with the star symbol (*).

```
db_brownfield:
    mappings:
        properties:
            '@timestamp':
                type: date
            '@version':
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_economic:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_environmental:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_heritage:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_law:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_marketing:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_pr_management:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_social:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            bp_tech_solutions:
                type: text
                fields:
                    keyword:
                        type: keyword
                        ignore_above: 256
            id_br_best_practices:
                type: long
            id_brownfield:
                type: long
```

Figure 4.1.2 Elasticsearch document mapping

As Figure 4.1.2 shows, the document has attributes equal to attributes of dbo.BR_BEST_PRACTICES table. Document attributes bp economic, environmental, heritage, law, marketing, management, social and tech solutions are all text fields storing different best practice recommendations for specific brownfields and fully consist of text data, which is the type of attributes with field keyword configuration. Both configurations of type text and field keyword are separate and used in different use cases, text type for full-text search and field keyword for aggregations and sorting.

4.1.5 Elasticsearch General Configuration

Our implementation uses automatic mapping of external data to Elasticsearch documents as shown in chapter 4.1.4, due to single text-based data formats in source database Microsoft SQL Server that is mapped to appropriate data types of attributes in Elasticsearch documents automatically.

We use a standard analyser that uses grammar-based tokenization which splits text into tokens on basis of the Unicode Text Segmentation algorithm. *Unicode Text Segmentation algorithm detects the boundaries between different text elements with help of Unicode character analysis to optimize for performance and speed of tokenization* (*UNICODE TEXT SEGMENTATION, 2020).* The standard analyser also uses a lower case token filter to change text tokens into all lowercase letters, to optimize for search performance.

Different analysers were tried, the standard analyser was chosen due to providing good and consistent performance results.

4.1.6 Defining Search Queries

Queries defined in this chapter show general search full-text search logic functionality in Elasticsearch. Definitions of specific Elasticsearch queries serve as a base for API which can be used by different client software systems to retrieve data from Elasticsearch.

```
"query": {
  "match": {
    "bp_environmental": {
      "query": "methane"
    }
  }
}
```

Figure 4.1.3 example of match query

Match query shown in figure 4.1.3 searches inside field bp_environmental of all documents within db_brownfield Elasticsearch index and returns found occurrences of queried value. Match query is more closely described in chapter 2.3.2.


```
{
  "took" : 2,
  "timed_out" : false,
  "_shards" : {
    "total" : 3,
    "successful" : 3,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 1.9506652,
    "hits" : [
      {
        "_index" : "db_brownfield",
        "_type" : "_doc",
        "_id" : "_yNEk3cBdvYqYKM_BeXE",
        "_score" : 1.9506652,
        "_source" : {
          "id_brownfield" : 16,
          "bp_pr_management" : "",
          "bp_heritage" : "",
          "bp_law" : "",
          "bp_social" : "",
          "@version" : "1",
          "bp_environmental" : "\"Effective limit the greenhouse effect should be useful by burning methane for the leisure complex
           .\r\nMethane escaping from the rock mass during mining of coal, by far the most part, escapes into the atmosphere.  For the
           Tropical Islands will be a cheap source of electricity and heat produced in CHP aggregates.  Near the location of this investment
           was discovered substantial sources of water, which in this respect and ensure self-sufficiency project.\"\r\n",
          "bp_tech_solutions" : "",
          "@timestamp" : "2021-02-11T22:44:04.427Z",
          "bp_marketing" : "",
          "id_br_best_practices" : 16,
          "bp_economic" : ""
        }
      }
    ]
  }
```


Figure 4.1.4 result from match query

The result from the match query shown in figure 4.1.4 contains information about the search process, in this case, the search took 2 milliseconds and was successful. The returned document contains in its field bp_environmental value “methane” which was specified in the match query.

Max_score result attribute represents numerical representation of specific document adherence to containing the queried value.

```
GET db_brownfield/_search
{
  "query": {
    "multi_match" : {
      "query": "Stuttgart",
      "fields": []
    }
  }
}
```

Figure 4.1.5 Example of a multi-match query

Multi-match query from figure 4.1.5 will find queried value “Stuttgart” in all fields in all documents within Elasticsearch brownfield index.

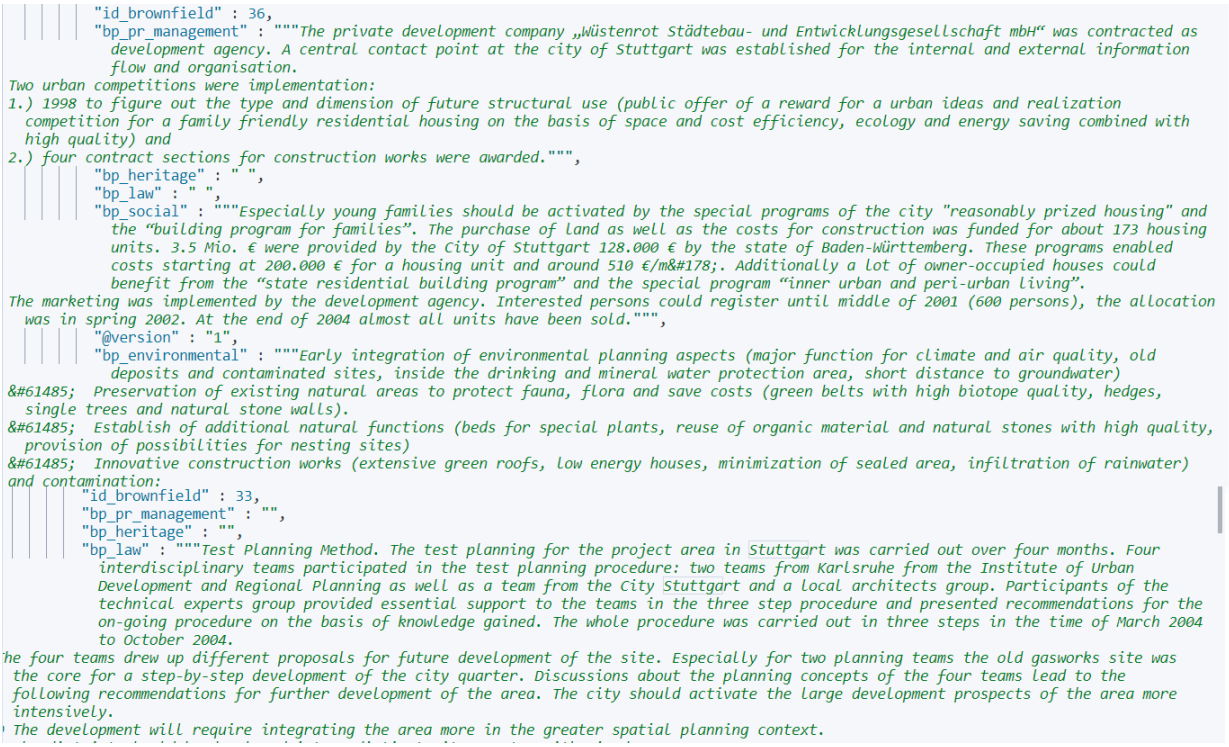

Figure 4.1.6 Compiled result of the multi-match query

The Multi-match query shown in figure 4.1.6 returned several documents containing searched value “Stuttgart” in different fields. Multi-match query allows for greater flexibility of search requirements from users and is more closely described in chapter 2.3.2.

```
GET db_brownfield/_search
{
  "query": {
    "match": {
      "bp_pr_management": {
        "query": "shazeholders",
        "fuzziness": "AUTO"
      }
    }
  }
}
```

Figure 4.1.7 Example of match query with fuzziness

Query depicted in figure 4.1.7 uses fuzziness term matching described in chapter 2.3.2, searched for value “shareholders” has been typed by mistake as “shazeholders”.

Figure 4.1.8 Result from match query with fuzziness parameter

The result shown in figure 4.1.8 returned the document containing the word shareholders. The same query without the fuzziness parameter would not return the document. This type of queries can enhance the user experience.

```
GET db_brownfield/_search
{
  "query": {
    "multi_match" : {
      "query" : "2004 Bydgokcz",
      "operator": "and",
      "fuzziness": "AUTO",
      "fields" : [ "bp_economic",
        "bp_pr_management" ]

    }
  }
}
```

Figure 4.1.9 Example of a multi-match query with fuzziness parameter

The Multi-match query shown in figure 4.1.9 has defined fuzzy querying with mode AUTO. Lucene engine will find similar terms to our queried value and after finding some level of similarity, will try and change different letters to match our original value. This process is further described in chapter 2.3.2 and allows Elasticsearch to reasonably

find also incorrectly typed queried words. The query will also search for a group of words in documents containing both words together in a sequence due to the configuration option of attribute *operator* being set to *and*. Correct spelling would be “2004 Bydgoszcz” instead of the one specified.


```
  "hits" : {
    "total" : {
      "value" : 2,
      "relation" : "eq"
    },
    "max_score" : 1.9357065,
    "hits" : [
      {
        "_index" : "db_brownfield",
        "_type" : "_doc",
        "_id" : "-yNEk3cBdvYqYKM_BeXE",
        "_score" : 1.9357065,
        "_source" : {
          "id_brownfield" : 12,
          "bp_pr_management" : "\"In 2004 Bydgoszcz authorities prepared the programme \r\nRevitalisation of cultural and natural heritage on
            the Mill Island\r\nand its closest surroundings aiming at complex reconstruction of\r\nthe area and giving it new urban functions
            . The decisions taken\r\nassumed transformation of degraded post-industrial area in the\r\nspace of culture, leisure and tourism.
            The programme provided\r\nfor keeping the unique historical character of the Mill Island\r\ntaking into account the requirements
            and needs of modern\r\ntimes. The following works were to be executed: renovation and\r\nadaptation of historical architecture
            (six buildings from a unique\r\ncomplex of industrial architecture), technical infrastructure\r\n(new water supply and sewage
            networks, gas and power\r\ngrids; reconstruction of wharfs) and transport infrastructure\r\n(renovation of a historical street,
            construction of footbridges)\r\nas well as leisure infrastructure (the construction of a complex\r\nof park docks, a playground,
            an amphitheatre, development\r\nof green areas), sports and tourist facilities (construction of a\r\nmarina). The assumption was
            the programme realisation with\r\nthe use of EU support funds. Due to content diversity of the tasks\r\nplanned and the projected
            high investment implementation cost,\r\nthe programme realisation was to be performed in the years\r\n2006-2010 within the
            frameworks of four projects:\r\nRevitalisation of the Mill Island for enterprise development;\r\nRevitalisation of items of
            cultural heritage on the Mill Island in\r\nBydgoszcz;\r\nConstruction of leisure infrastructure on the Mill Island and in\r\nits
            closest surroundings;\r\nRevitalisation of degraded sports areas on the Mill Island.\"\r\n",
          "bp_heritage" : """conservator's guidelines
""",
```


Figure 4.1.10 Result from Multi-match query with fuzzy parameters

The result from the multi-match query with fuzziness shown in figure 4.1.10 returned document containing correctly spelt word sequence with the highest score for adherence to queried value. Different documents containing the same term were also returned but are not shown in Figure 4.1.10 due to the graphical size of the result.

4.1.7 Kibana Search Tools Implementation

Kibana allows us to create a complex search bar, dashboard, and other visualization tools for our Elasticsearch index representing documents containing data about the brownfield database.

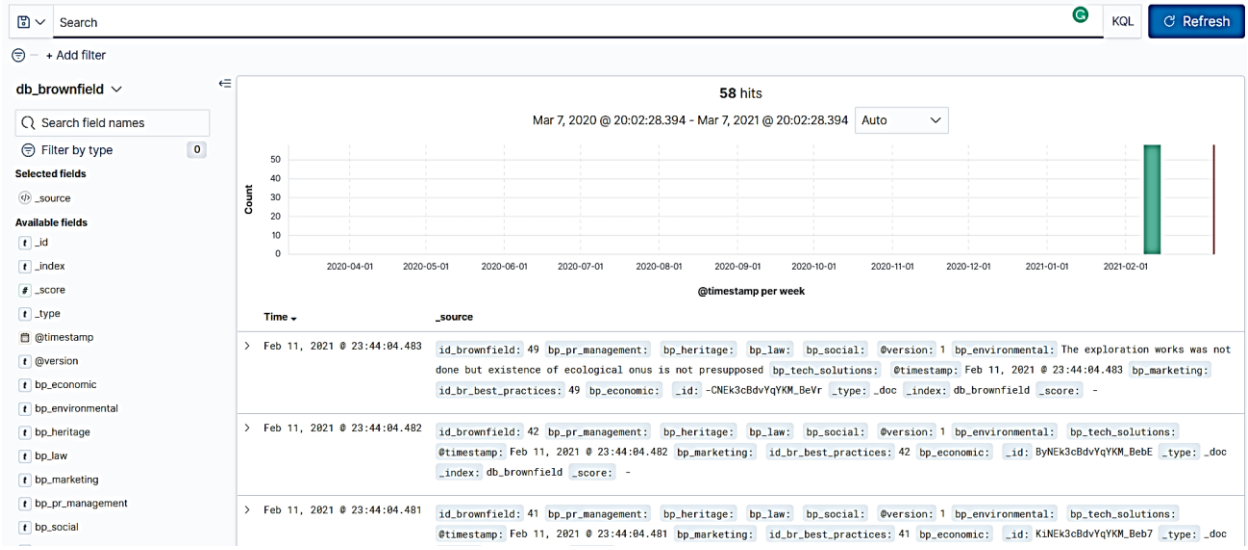

Figure 4.1.11 Overview of Kibana discover search function

Figure 4.1.11 shows the general Kibana discover function that allows us to perform searching using Kibana Query Language or older Lucene query syntax. This search tool can be used directly by end-users of our search system.

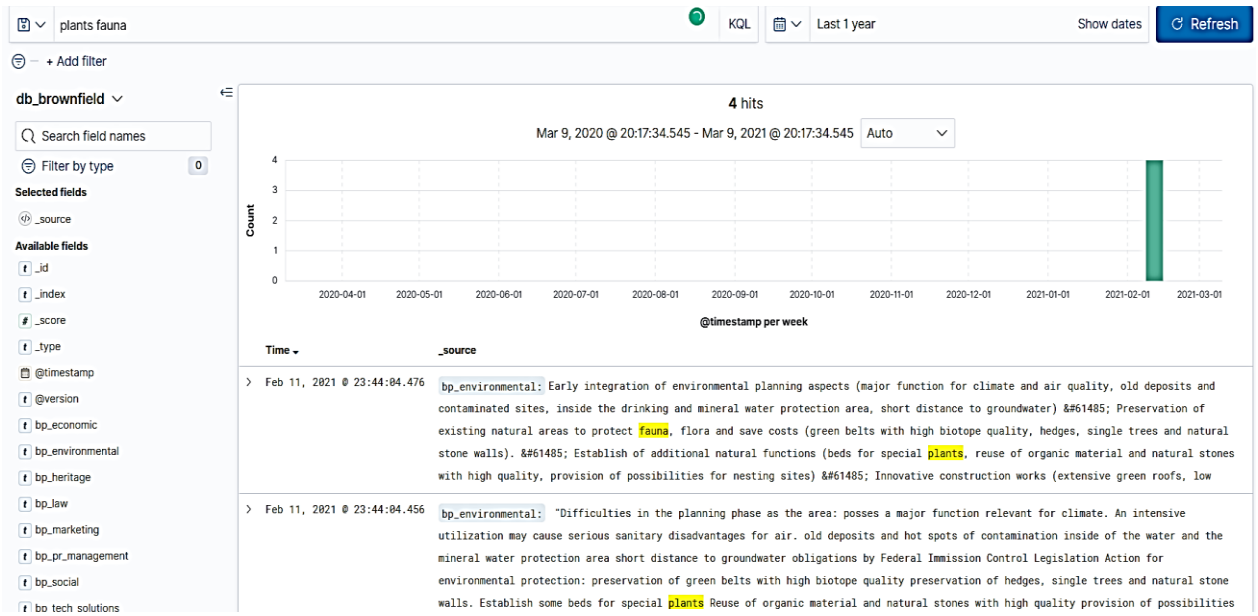

Figure 4.1.12 Result of the unfiltered term search

Figure 4.1.12 shows returned documents based on the specified search terms. The resulting documents can contain any number of terms in any field from our specified set of terms.

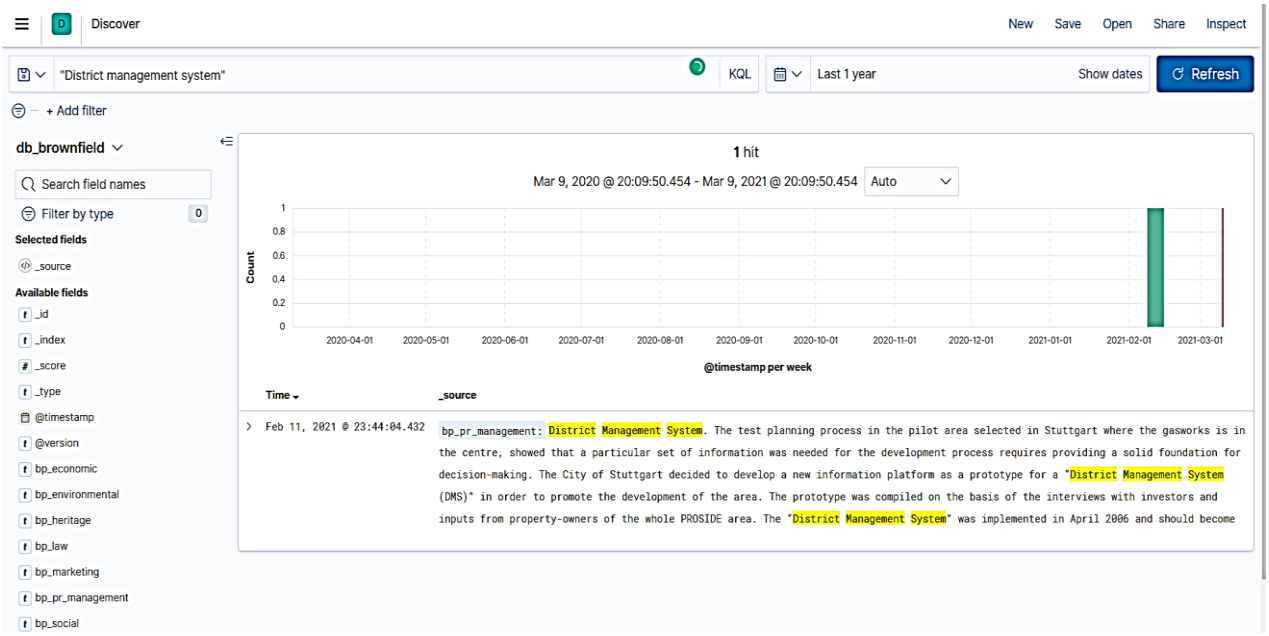

Figure 4.1.13 Result of unfiltered sequence search

Figure 4.1.13 shows searching without any filter specified. We can find documents containing exactly matched expression sequences to our search values after prefixing and postfixing our search for value in quotation marks. This search type use *and* logic described in chapter 2.3.2.

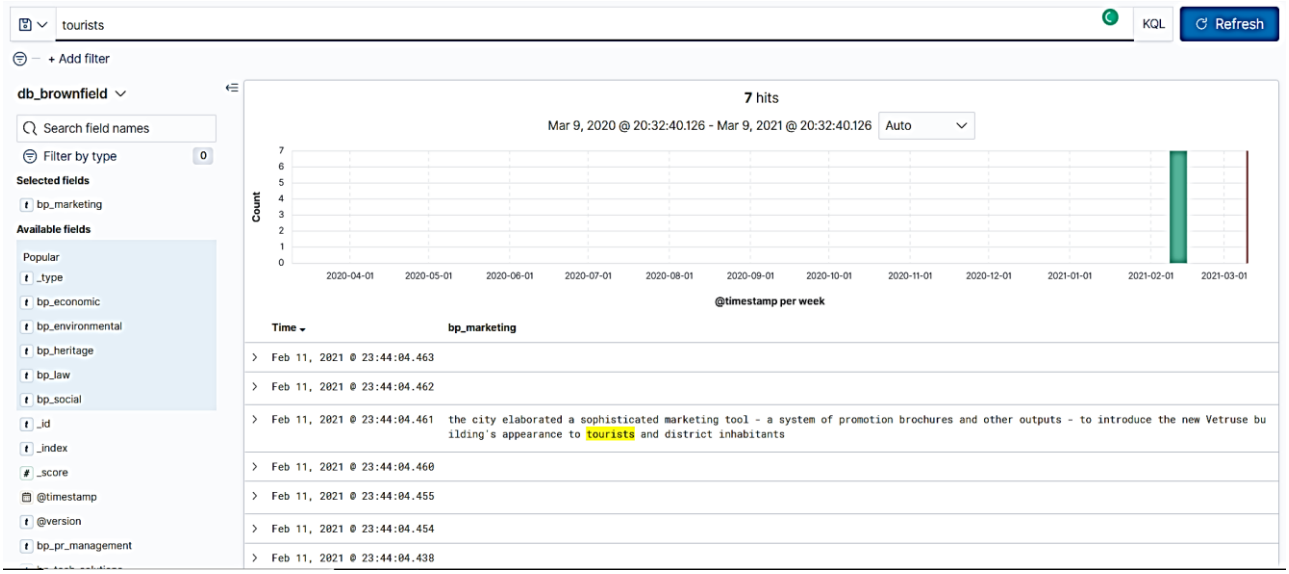

Figure 4.1.14 Result of the filtered term search

Figure 4.1.14 displays documents returned containing searched for value only in filtered out field bp_marketing, rest of the fields are hidden.

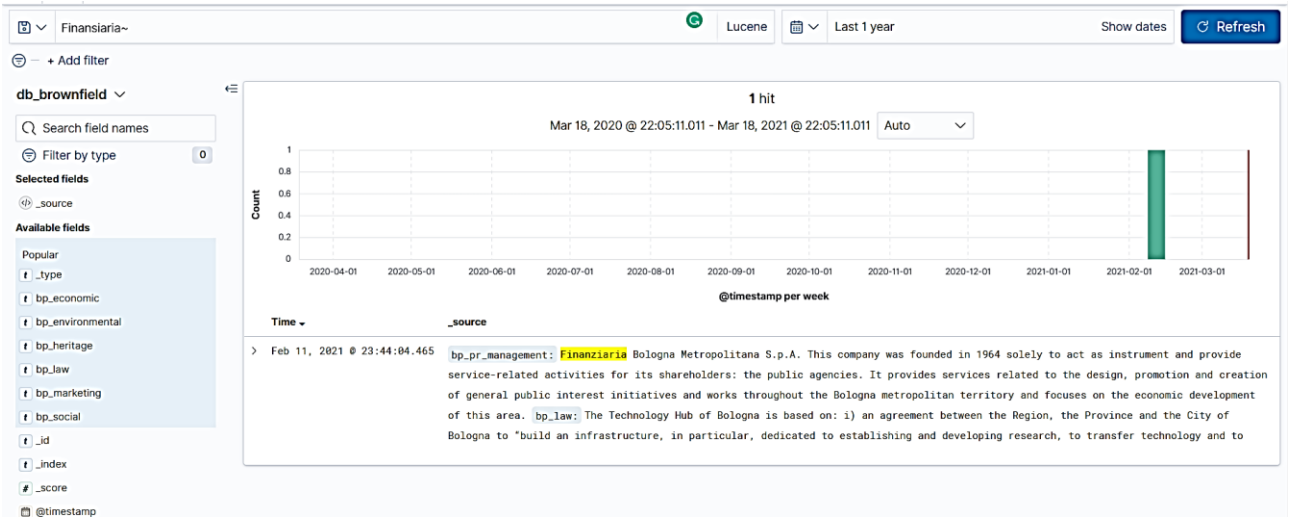

Figure 4.1.15 Result of the fuzzy term search

Figure 4.1.15 shows the ability of fuzzy searching by the Kibana discovery function. After searching for the incorrect term “Finansiara” instead of “Finanziara”, the resulting document contains the correct term. This functionality is similar to match queries with fuzziness enabled in chapter 4.1.6, specifically defined in figures 4.1.7 and 4.1.9.

4.2 Search System Configuration Using Microsoft SQL Server

4.2.1 Pricing of Microsoft SQL Server

Brownfield database is currently running on Microsoft SQL Server 2019 express edition. Express edition can be used free of charge for commercial usage. The free edition is limited to 10 GB of storage, 1 GB of RAM and using one physical processor core. In its current state, resources provided by free of charge express edition of Microsoft SQL Server are enough to maintain the effective functioning of the brownfield search system.

In the eventuality of requirement scope increase for brownfield database, we can use different deployment options of Microsoft SQL Server.

On-Premises Deployment

There are several licensing options for Microsoft SQL Server 2019, mainly standard and enterprise edition. Standard edition is limited to 24 maximum number of used processor cores, 128 GB of RAM used per instance and 524 PB maximum size. Enterprise edition has only one limitation in terms of its maximum size of 524 PB, use of cores and RAM can be unlimited and bound only by server characteristics. Standard edition generally uses server or core licensing model. Price can differ between different deployment options and additional support options and can also differ between offers of different implementation partners and providers of Microsoft. The server license costs 899 $ per connected device and the core license costs 3,586 $ per two cores which are sold in two-core packs.

Cloud Deployment

Microsoft SQL Server can be deployed through public cloud providers, mainly Microsoft Azure. Microsoft Azure provides the possibility to deploy Microsoft SQL Server into a cloud environment. SQL Server on Virtual Machines is a more traditional way of deployment, that combines Microsoft SQL Server functionality with general deployment scalability and flexibility of the public cloud. Pricing options differ between specific deployment requirements and level of support. There are additional possibilities to integrate the on-premises deployment with cloud deployment through hybrid cloud options, which can be beneficial to handle fast unexpected spikes in processed data volumes and other use cases. In general, the price for the brownfield database deployed within Microsoft Azure would be thousands of dollars per year, depending on the potential increase in data volume and new requirements from clients. Future price estimation without specific details that are currently unknown provides only a very general overview and estimation of the potential cost. There are additional types of services offered by Microsoft Azure and different public cloud providers like Amazon Web Services or Google Cloud to deploy and use Microsoft SQL Server within a public cloud environment.

4.2.2 Enabling Full-Text Search in Microsoft SQL Server

Full-text search in Microsoft SQL Server requires having full-text search package installed and enabled within Microsoft SQL Server instance. This process can be done

while installing Microsoft SQL Server or additionally after it was installed through installation wizard.

The core object for full-text search in Microsoft SQL Server is a full-text catalogue, with which all full-text indexes are associated.

```
USE brownfield_db;
GO
CREATE FULLTEXT CATALOG brownfieldFullTextCatalog;
```

Figure 4.2.1 Fulltext Catalog Creation

Figure 4.2.1 shows an example of T-SQL syntax for creating a new full-text catalogue object named brownfieldFullTextCatalog. USE predicate specifies database name in Microsoft SQL Server with which given full-text catalogue will be associated.

A full-Text index can be defined on specific columns through management studio configuration or by T-SQL commands. Each table can have a maximum of one Full-Text index and needs to have a unique identifier column, usually in form of a primary key.

```
CREATE FULLTEXT INDEX ON  [dbo].[BR_CONTAMINATION]([CONT_AIR],
[CONT_WATER],[CONT_SOIL])
KEY INDEX [PK_CONTAMINATION] ON brownfieldFullTextCatalog
WITH CHANGE_TRACKING AUTO
GO
```

Figure 4.2.2 Example of definition of full-text index

Figure 4.2.2 defines full-text index on table BR_CONTAMINATION for columns CONT_AIR, CONT_WATER, CONT_SOIL. Given columns are all text-based and full-text index definition allows performing full-text queries in Microsoft SQL Server as defined in chapter 2.3.4. KEY INDEX predicate defines new unique index PK_CONTAMINATION that will be the part of brownfieldFullTextCatalog object described previously. The default language is set for British English, which is the same as the language used in the brownfield database and can additionally be changed. CHANGE_TRACKING AUTO defines propagation of changes in columns defined under full-text index to the given index. The AUTO option will automatically propagate changes made to given columns. Other options include MANUAL, where *change propagations must be made via ALTER FULLTEXT INDEX ... START UPDATE POPULATION (CREATE FULLTEXT INDEX (Transact-SQL). 2017).* Population

options set to OFF does not populate full-text index and all populations must be made manually, similarly to option MANUAL.

Additional options for full-text index creation include setting specific stoplist object, as further described in chapter 2.3.3. By default, Microsoft SQL Server uses a system stoplist, that already has a list of stopwords for the English language, other languages can be manually specified.

4.2.3 Examples of Microsoft SQL Server Full-Text Search Queries

This chapter describes full-text queries similarly in comparison to search query definitions made for Elasticsearch implementation described in chapter 4.1.6.

```
SELECT BP_ENVIRONMENTAL
FROM dbo.BR_BEST_PRACTICES
WHERE CONTAINS(BP_ENVIRONMENTAL, 'Methane')
```

Figure 4.2.3 Example of CONTAINS predicate usage

Figure 4.2.3 shows an example of contains predicate. Predicate specifies the exact column in which value Methane will be searched for. This query will return a specific document returning BP_ENVIRONMENTAL column in which searched for value “Methane” will be found.

	BP_ENVIRONMENTAL
1	"Effective limit the greenhouse effect should be useful by burning methane for the leisure com...

Figure 4.2.4 Result from CONTAINS predicate defined in figure 4.2.3

The result shown in figure 4.2.4 shows returned document containing the term “methane”. This result is the same as the result shown in figure 4.1.4 from the match query defined in Elasticsearch.

```
SELECT *
FROM dbo.BR_BEST_PRACTICES
WHERE CONTAINS(*, '"Stuttgart"')
```

Figure 4.2.5 Example of CONTAINS predicate in multi-column search

The query shown in figure 4.2.5 will find the value Stuttgart in all columns within the full-text index defined in table dbo.BR_BEST_PRACTICES.

	ID_BR_BEST_PRACTICES	ID_BROWNFIELD	BP_ENVIRONMENTAL	BP_PR_MANAGEMENT
1	6	6	Initial environmental assessments provide a suffi...	District Management Sy
2	7	7	Early integration of environmental planning aspe...	Development agency: W
3	24	24	Demolition and remediation of contaminated site...	Management by the City
4	33	33		
5	36	36	Early integration of environmental planning aspe...	The private developmen

Figure 4.2.6 Result from CONTAINS predicate defined in figure 4.2.5

Result of the query shown by figure 4.2.6 returns all rows/documents containing the term "Stuttgart" in any of the defined indexed columns in table dbo.BR_BEST_PRACTICES. This result is in its core same as a result from the multi-match query shown by figure 4.1.6 in chapter 4.1.6.

```
SELECT BP_PR_MANAGEMENT
FROM dbo.BR_BEST_PRACTICES
WHERE CONTAINS(BP_PR_MANAGEMENT, 'shazeholders')
```

Figure 4.2.7 Example of CONTAINS predicate searching for a mistyped term

Unfortunately, the query shown in figure 4.2.7 to find the mistyped term "shazeholders", instead of "shareholders" did not manage to find the desired document containing the word in its correct form, unlike the Elasticsearch query shown by figure 4.1.7 in chapter 4.1.6 in column BP_PR_MANAGEMENT.

```
SELECT BP_PR_MANAGEMENT
FROM dbo.BR_BEST_PRACTICES
WHERE FREETEXT(BP_PR_MANAGEMENT, 'shareholder')
```

Figure 4.2.8 Query using FREETEXT predicate to find the plural form of the specified term

Figure 4.2.8 defines query to search for value "shareholder" instead of "shareholders". CONTAINS predicate did not manage to return any document. Freetext predicate query managed to find the correct document depicted in Figure 4.2.9.

	BP_PR_MANAGEMENT
1	Finanziaria Bologna Metropolitana S.p.A. This company was fou...

Figure 4.2.9 Result from FREETEXT predicate shown by figure 4.2.8.

```
SELECT *
FROM dbo.BR_BEST_PRACTICES
WHERE FREETEXT((BP_ECONOMIC,BP_PR_MANAGEMENT), '"2004 Bydgokcz"')
```

Figure 4.2.10 Example of the query using FREETEXT predicate to find incorrectly typed value

Figure 4.2.10 shows FREETEXT predicate query that was incorrectly typed to find value "2004 Bydgokcz" instead of "2004 Bydgoszcz" in columns BP_ECONOMIC or BP_PR_MANAGEMENT. This query is similar to the multi-match query defined in Elasticsearch shown by Figure 4.1.9 in chapter 4.1.6.

	ID_BR_BEST_PRACTICES	ID_BROWNFIELD	BP_ENVIRONMENTAL	BP_PR_MANAGEMENT
1	2	5		The Project is an example of cooperation between
2	3	1		In 2004 Bydgoszcz authorities prepared the progra
3	12	12		"In 2004 Bydgoszcz authorities prepared the progr.
4	38	38		The Project is an example of cooperation between

Figure 4.2.11 Result of the query defined in Figure 4.2.10

The result shown in Figure 4.2.11 shows returning of appropriate documents containing searched for values by Microsoft SQL Server, similarly to Elasticsearch.

4.3 Recommending Search System Solution

Chapter 4 described the general configuration and procedures required to perform a full-text search with the use of Elastic Stack and mainly Elasticsearch and Microsoft SQL Server. The client currently uses the brownfield database within Microsoft SQL Server. Configuration and implementation of full-text search capabilities in Microsoft SQL Server provided by chapter 4.2 can be recommended to the client, especially while using express edition of Microsoft SQL Server with free license as described in chapter 4.2.1. Express edition has its limitations but in its current state can fully comprehend the requirements for data volume and velocity of brownfield data. The client is also recommended to integrate the newly proposed full-text search solution with his existing frontend GUI, primarily the new functionality of Microsoft SQL Server and search queries on basis of the ones defined in chapter 4.2.3. Implementation of Elastic Stack as described in chapter 4.1 can also be recommended, mainly in the way of self-deployment due to the current size of brownfield data volume and velocity that could be effectively handled by self-deployed and managed instances of Elastic Stack application. The client can also think about buying commercially deployed Elastic Stack as described in chapter

4.1.3, mainly the most basic configuration that would also suffice current data size and volume requirements but provide for a more standardized solution with application support included. Elastic Stack implementation will also provide the client with a new way to use a graphical user interface to perform a full-text search function. Additionally, the client can integrate his own frontend solution with the full-text search design of Elasticsearch search queries defined in chapter 4.1.6. The document data format may require wider changes to the client's current frontend solution, however, the frontend graphical user interface for Elasticsearch is provided through Kibana as previously stated.

In the case of future requirements to accommodate for increased data volume, velocity, variety or other characteristics, the client can use more robust deployment options for Elastic Stack, as described in chapter 4.1.3. If the client were to start using commercial Elastic Stack deployment in the current circumstances, it could be automatically scalable to a more robust solution that could accommodate for greater data volume, velocity and variety.

Microsoft SQL Server could be used in the case of future increase in data requirements in case of insufficient hardware resources with a commercial license on-premise solution, which is generally priced in chapter 4.2.1. The client could also think about complete or hybrid public cloud deployment of Microsoft SQL Server, the decision should be based on specific future requirements, such as infrastructure scalability due to for example seasonality of usage of the search system or global deployment options within different regions and other possible use cases, to decide whether cloud option will generate real business benefits.

The future need and budget of the client can help decide whether the future solution would consist only of using Microsoft SQL Server, Elastic Stack, or their combination. In the case of increased data variety due to for example increased number of source data systems and data formats being used by the search system, Elastic Stack could provide more benefits to the customer. The same can be stated about increased data velocity or new requirements for increased availability of the search system. Basic comparison between Elastic Stack and Microsoft SQL Server as defined in chapter 2.5.2 can be used by the client to help decide which solution would provide more value and benefits.

The labour cost of implementation experts should be also considered by the client. The small research group can use the help of students to implement or additionally

configure either Microsoft SQL Server or Elastic Stack solution. For more robust search system solutions with required periodical support and more robust infrastructure, the client should think about the budget allocation for experts who can help develop and maintain the search system in a more systematic and structured way.

5 Conclusion

Solutions to create a search system for the data concerning brownfields were analysed and proposed to the research group together with their concrete implementations.

The first area of focus of the diploma thesis analysed and provided a solution to use existing technology known by the members of the research group, Microsoft SQL Server to create a search system with text search capability. The developed solution fulfils the requirements of the research group and provides the possibility to develop this solution further, mainly in the case of requirements changing in the future due to a shift in the data structure of the brownfield methodology data. General pricing and implementation information were also provided as part of the analysis to give the client holistic information and knowledge about this particular solution.

The second area of focus of the diploma thesis similarly analysed and provided a solution to use a new set of technologies in form of the Elastic Stack and its components to create a search system with specialization on text data together with its implementation. The procedure of configuration and following integration of different Elastic Stack components, together with its pricing and additional implementation options were also provided to the members of the research group. The general need for future changes of the search system was considered and additional development and implementation options were described as part of the diploma thesis.

Differences between both solutions were described from a theoretical and practical perspective and can be used by the members of the research group in future decision making about maintaining or further developing the search system for brownfield related data.

Bibliography

1. Ben-Gan, I. *T-SQL fundamentals*. Redmond, WA: Microsoft Press, 2016. ISBN 978-1-5093-0200-0.

2. *Common options.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2021 [9. 1. 2021]. Available on: https://www.elastic.co/guide/en/elasticsearch/reference/current/common-options.html

3. *Configure Kibana.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2021 [11. 2. 2021]. Available on: https://www.elastic.co/guide/en/kibana/7.12/settings.html

4. *Configuring Elasticsearch.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2021 [6. 2. 2021]. Available on: https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html

5. *Install Elasticsearch with .zip on Windows.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2021 [3. 2. 2021]. Available on: https://www.elastic.co/guide/en/elasticsearch/reference/current/zip-windows.html

6. *Install Kibana on Windows.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2021 [10. 2. 2021]. Available on: https://www.elastic.co/guide/en/kibana/current/windows.html

7. *Running Logstash on Windows.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2021 [1. 2. 2021]. Available on: https://www.elastic.co/guide/en/logstash/current/running-logstash-windows.html

8. *Running the Elastic Stack on Docker.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2021 [13. 2. 2021]. Available on: https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-docker.html

9. *Stashing Your First Event.* Elasticsearch B.V. Elastic Stack and Product Documentation. elastic.co [online]. 2020 [19. 12. 2020]. Available on: https://www.elastic.co/guide/en/logstash/current/first-event.html

10. Haldar, R., Mukhopadhyay, D.: Levenshtein distance technique in dictionary lookup methods: an improved approach. CoRR abs/1101.1232 (2011)

11. Hiba Jasim Hadi, Ammar Hameed Shnain, Sarah Hadishaheed, Azizahbt Haji Ahmad. BIG DATA AND FIVE V'S CHARACTERISTICS. *International Journal of Advances in Electronics and Computer Science*. 2015, 2(1), 2393-2835. ISSN 2393-2835.

12. Chen, P. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems*.1976, 1(1), 9-36. ISSN 0362-5915.

13. MEIER, Andreas and KAUFMANN, Michael. *SQL & NoSQL databases: models, languages, consistency options and architectures for Big data management*. Wiesbaden: Springer, 2019. ISBN 978-3-658-24548-1.

14. CREATE FULLTEXT INDEX (Transact-SQL). Microsoft. Docs. docs.microsoft.com [online]. 2017 [24.3.2021] Available on https://docs.microsoft.com/en-us/sql/t-sql/statements/create-fulltext-index-transact-sql?view=sql-server-ver15

15. *Data types (Transact-SQL)*. Microsoft. Docs. docs.microsoft.com [online]. 2017 [28.12.2020] Available on https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15

16. Query with Full-Text Search. Microsoft. Docs. docs.microsoft.com [online]. 2017 [22.3.2021] Available on https://docs.microsoft.com/en-us/sql/relational-databases/search/query-with-full-text-search?view=sql-server-ver15

17. Schildt, H. *Java The Complete Reference, Eleventh Edition*. New York: McGraw-Hill Education, 2019. ISBN 978-1-26-044024-9.

18. UNICODE TEXT SEGMENTATION. Unicode, Inc. Technical Reports. Unicode.org [online]. 2020 [27.3.2021] Available on https://unicode.org/reports/tr29/

19. WONG, Wai T. *Advanced Elasticsearch 7.0*: *a practical guide to designing, indexing and querying advanced distributed search engines.* Birmingham: Packt Publishing, 2019. 560 p. ISBN 978-1789957754.

20. *Overview Of EPA's Brownfields Program*. United States Environmental Protection Agency. Brownfields. epa.gov [online]. 2020 [28.11.2020] Available on https://www.epa.gov/brownfields/overview-epas-brownfields-program

21. Benchmarking and sizing your Elasticsearch cluster for logs and metrics. Yacine Younes. Blogs. elastic.co [online]. 2020 [2. 3. 2021]. Available on: https://www.elastic.co/blog/benchmarking-and-sizing-your-elasticsearch-cluster-for-logs-and-metrics

List of Abbreviations

API - Application Programming Interface

AWS - Amazon Web Services

ELT- Extraction-Load-Transformation

ERD – Entity Relationship Diagram

ETL - Extraction-Transformation-Load

GUI - Graphical User Interface

HTML - Hypertext Markup Language

JDBC - Java Database Connectivity

JDK - Java Development Kit

KQL – Kibana Query Language

OLTP - Online Transactional Processing

RDBMS – Relational Database Management System

T-SQL - Transact Structured Query Language

URL - Uniform Resource Locator

Herewith I declare that

- I was acquainted with the fact that my diploma thesis is fully covered by Act No. 121/2000 Coll. - Copyright Act, in particular § 35 - use of the work in the framework of civil and religious ceremonies, in school performances and use of school work and § 60 - school work;
- I acknowledge that by submitting my diploma (bachelor) thesis, I agree to the publication of my work under Act No. 111/1998 Coll. on universities and on amendments to other acts (the Higher Education Act), as amended, regardless of the outcome of the defence;
- I acknowledge that the VSB – Technical University of Ostrava (hereinafter VSB-TUO) has a non-profit right to use the diploma (bachelor) thesis for its own internal needs (Section 35 (3));
- I agree that the thesis (bachelor's thesis) will be archived in electronic form in the Central Library of VSB-TUO. I agree that bibliographic data on the thesis will be published in the VSB-TUO information system;
- it was agreed that I would conclude a license agreement with VSB-TUO, if it was interested in it, with the right to use the work within the scope of Section 12 (4) of the Copyright Act;
- it was agreed that I can use my work, diploma thesis, or grant a license for its use only with the consent of VSB-TUO, which is entitled to request a reasonable contribution from me because of costs incurred by VSB-TUO for the creation of the work (up to their actual amount).

Ostrava dated 12.04.2021

Patrik Polaček

List of Annexes

Annex 1: File Archive

Annex 1: File Archive

File archive contains:

- configuration documents and query definition documents defined in chapter 4.