Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Metadata Management in the Data Lake Architecture

Rebecca Kay Eichler

Course of Study:	Software Engineering
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Corinna Giebler, M.Sc.
Commenced:	November 29, 2018
Completed:	May 29, 2019

Abstract

The big data era has introduced a set of new challenges, one of which is the efficient storage of data at scale. As a result, the data lake concept was developed. It is a highly scalable storage repository, explicitly designed to handle (raw) data at scale and to support the big data characteristics. In order to fully exploit the strengths of the data lake concept, pro-active data governance and metadata management are required. Without data governance or metadata management, a data lake can turn into a data swamp. A data swamp signifies that the data has become useless, or has lost in value for a variety of reasons, therefore it is important to avoid this condition.

In the scope of this thesis a concept for metadata management in data lakes is developed. The concept is explicitly designed to support all aspects of a data lake architecture. Furthermore, it enables to fully exploit the strengths of the data lake concept and it supports both classic data lake use cases as well as organization specific use cases. The concept is tested by applying it to a data inventory, data lineage and data access use case. Furthermore, a prototype is implemented demonstrating the concept through exemplary metadata and use case specific functionality. Finally, the suitability and realization of the use cases, the concept and the prototype are discussed. The discussion yields that the concept meets the requirements and is therefore suitable for the initial motivation of metadata management and data governance.

Kurzfassung

Mit der Big Data Ära sind eine Reihe neuer Herausforderungen entstanden. Unter anderem gehört dazu das effiziente Persistieren der Daten. In diesem Kontext wurde das Data Lake Konzept entwickelt. Es ist ein Repository für besonders große Mengen an (Roh-) Daten unterschiedlichster Art. Daten liegen in Data Lakes oft in großen Mengen undokumentiert vor. Das führt dazu, dass nicht klar ist welche Daten vorhanden und brauchbar sind. In diesem Zustand bezeichnet man den Data Lake als Data Swamp. Um die Entwicklung eines Data Swamps zu vermeiden, muss proaktiv Data Governance und Metadatenmanagement durchgeführt werden.

Im Rahmen dieser Arbeit wird ein Konzept für das Metadatenmanagement in Data Lakes entwickelt. Das Konzept ist explizit darauf ausgelegt, die unterschiedlichen Ebenen einer Data Lake Architektur zu unterstützen. Darüber hinaus ermöglicht es, die Stärken des Data Lake Konzepts besser auszuschöpfen und unterstütz sowohl klassische Data Lake Use Cases als auch unternehmensspezifische Use Cases. Der Entwurf des Konzept basiert auf einer Reihe von Data Lake typischen Anwendungsfällen und wird später anhand dieser getestet. Dazu gehören die Inventar-, Daten-Abstammungs- und Daten-Zugriffs Anwendungsfälle. Zudem wird ein Prototyp implementiert, welcher das Konzepts durch exemplarische Metadaten und Anwendungsfall spezifische Funktionen demonstriert. Zuletzt wird die Eignung und Umsetzung der Anwendungsfälle, des Konzepts und des Prototyps diskutiert. Es wird gezeigt, dass das Konzept den definierten Anforderungen entspricht und somit Metadatenmanagement und Data Governance im Data Lake ermöglicht.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

ADMS Asset Description Metadata Schema. 33

CLI command-line interface. 55

EDM Enterprise Data Management. 33

EMM Enterprise Metadata Management. 23

GDPR General Data Protection Regulation. 20

GEMMS Generic and Extensible Metadata management System. 34

Goods Google Dataset Search. 33

HIPAA Health Insurance Portability and Accountability Act. 21

M4DG Metadata Model for Data Goods. 33

NISO National Information Standards Organization. 22

NoSQL Not only SQL. 9

RDB Relational Database. 28

RDF Resource Description Framework. 9

SOX Sarbanes-Oxley Act. 21

SQL Structured Query Language. 30

1 Introduction

In recent years, the rise of the big data era has presented many organizations with both new opportunities and challenges. The term big data is defined through the so called V-properties, such as volume, variety and velocity [MT16]. These respectively signify the amount, diversity and speed with which data is generated today. The big data related challenges are often closely coupled to these properties. More specifically, one challenging aspect is the storage and management of data. As existing storage solutions, such as the data warehouse, weren't designed to handle these big data properties, they no longer meet all of the requirements for data at scale [Hua15]. They have shortcomings in aspects such as scalability, storage cost, integrating new data and can't exploit the data's full potential, as they are implemented according to predefined use cases. As a result the data lake concept was developed, designed to deal with exactly these big data characteristics. In short, a data lake is a highly scalable storage repository. One of its most distinguishing features is the absence of a predefined data schema. The schema-less approach provides advantages such as faster integration of new data types, as no schema adjustment is required. The data is directly integrated in its raw form without complex and costly modeling and transformations for schema conformity. Therefore, more data can be stored faster and no information is lost through transformations, which benefits both the big data's volume and velocity properties. Lastly, the data lake can also incorporate data of any structure, meaning structured, semi-structured and unstructured data, thereby supporting the big data's variety characteristic.

Although the mentioned data lake characteristics solve many problems, the new concept also introduces new challenges [Hua15]. Storing data in its native format, without transformations, enables to quickly store large amounts of data with very little effort. This tempts people to store large amounts of data without a strategy, which often results in a large collection of undocumented data. Without documentation of any kind data loses its expressiveness. In addition, it is also not clear what data resides in the lake if no inventory is maintained. Under such conditions, the stored data becomes useless due to lacking documentation, or cannot be found. A data lake in this state is called a data swamp [Mat17]. The creation of a data swamp can be prevented by conducting data governance and metadata management tasks.

In general, data governance involves specifying guidelines for handling and maintaining data [HGHM11]. The guidelines are centered around the topics data quality, lifecycle management and data security [Mat17]. Dealing with these topics involves the collection of metadata, which is data on data [Ril17]. Handling and maintaining the collected metadata is part of the metadata management domain [SDE18b]. Metadata management is a cross-sectional task and covers several use cases including the data governance use case.

It has been established why the storage concept data lake was created, what it is and that metadata management is essential for utilizing its potential. In order to conduct comprehensive metadata management on the data lake, it has to be integrated into an all-encompassing data lake architecture. The data lake architecture groups all processes, tasks and components related to a data lake into

levels, thereby providing a framework for the creation and maintenance of a data lake [GGH19]. Ultimately, the metadata management should be designed to incorporate all of the level's different aspects in order to be comprehensive.

The following Section 1.1 gives an overview of the thesis' goals and Section 1.2 provides insight into the thesis' structure through a general outline.

1.1 Goal of the Thesis

The main objective of this thesis is to provide a general design for implementing metadata management based on an all-encompassing data lake architecture. The resulting metadata management enables fully exploiting the strengths of the data lake concept. Furthermore, the design supports a variety of classic data lake uses cases as well as organization specific use cases. The following four goals are used as a framework for achieving the main objective:

1. To begin with a set of classic data lake use cases will be created. Each of these should highlight the necessity for metadata management in the data lake.
2. The created use cases will serve as a reference for the design of a metadata management concept, which will be the central outcome of the thesis. It should enable storing and handling diversely structured metadata of any type and quantity. The established concept should be designed to support characteristics of the underlying data lake architecture. While its design is partly inspired through the given use cases, it should also be abstract enough to cover a wide variety of both classic data lake and organization specific metadata management use cases.
3. In order to demonstrate the established concept, a prototype will be created. It should highlight both the realizability of the concept as well as its aptness for answering the use cases.
4. Lastly, both the concept and prototype will be evaluated based on the main objective as well as their suitability for implementing the use cases.

1.2 Outline

The goals presented in Section 1.1 are dealt with as follows. To begin with, Chapter 2 "Background" features explanations on several topics, which are required for understanding the necessity for and the design of the use cases, a metadata management concept and a prototype. The next Chapter 3 "Related Work" illustrates research and products in the same topic area and how the work conducted in this thesis differs from it. Chapter 4 contains the created use cases. Chapter 5 demonstrates the established metadata management concept. The prototype is presented in Chapter 6. "Discussion and Conclusion", in Chapter 7, critically evaluates the concept and prototype. The last Chapter 8 "Summary and Outlook" recapitulates all topics discussed in the previous chapters and provides ideas for future work.

2 Background

This chapter covers the basics on all topics, which are required to understand the necessity and design of the created use cases, metadata management concept and prototype. To begin with, the concept of the data lake is explained in Section 2.1, followed by a description on how it can turn into a data swamp, in Section 2.2. The section also explains the necessity for data governance, which in turn uses metadata and metadata management. The three topics are explained in Sections 2.3 to 2.5, respectively. The created metadata management concept is based on architecture levels, which are described in the section 2.6. Lastly, Section 2.7 contains information on types of storage systems, one of which will be used for the prototype.

2.1 Data Lake

The *data lake* storage concept was developed to tackle challenges, which arose with the big data era. These challenges are tightly coupled with the big data's so-called V-properties. Amongst others, these include: volume, variety, value, veracity, velocity and validity [QSC16]. Volume refers to the sheer amount of data generated, whereas variety alludes to its diversity. Value signifies whether useful insights can be generated from the data. Velocity indicates the speed with which the data is generated. Lastly, validity describes its expressiveness.

Existing storage solutions, such as the data warehouse, weren't designed or even intended to handle these big data properties [Hua15]. Hence, they don't fare well in some aspects of big data use cases such as scalability, storage cost, integrating new data and so on. As a result, a new concept was developed, designed to deal with exactly these big data characteristics, which is now called a "data lake." At its core the data lake is a highly scalable storage repository [MT16]. It is intended to improve the capture, refinement, archival and exploration of data [Hua15]. The name was given to it by James Dixon in the 2010s [MT16].

One of the data lakes most prominent characteristics is the absence of a predefined data schema [Hua15]. In traditional storage solutions, like the data warehouse, the data schema is a central aspect, which defines which data and in what form is integrated. The schema is defined prior to data integration, which is called "schema on write" or "early binding" [Hua15; Mat17]. As opposed to data warehouses, the schema is defined when the data is required for use in data lakes. This is called "schema on read" or "late binding". The no schema approach has several advantages. To begin with, new data types can be incorporated without requiring a schema adjustment [Hua15]. Hence, this is how the data lake supports the big data's variety characteristic. Next, the absence of a comprehensive schema eliminates complex and costly modeling. However, not having a schema during integration does not eliminate the necessity of a schema in general. A schema can also be seen as a sort of registry of what data is stored. Therefore, the absence of a schema means there is no predefined specification of what data can be found within the data lake. This specific issue is addressed in more detail in Section 2.2.

Since there is no schema, the data does not have to be transformed for schema conformity. It is integrated directly in its native format as "Raw data" [Hua15]. This is also one of the data lake's distinguishing features. Since time-consuming transformation steps are avoided upon data ingestion, data is directly accessible, which is helpful e.g. if it is required in real-time. Through the performance aspect of faster integration, this facilitates the volume and velocity aspects of big data. Furthermore, the flexibility in the use of the data is better as no information is lost through foregoing transformations. This is an opportunity to extract the data's value to its full potential and therefore, accommodates the big data's value property. Another consequence of no schema is the ability to incorporate structured data as well as semi-structured and unstructured data. The data lake also supports all forms of data processing.

The above-mentioned characteristics have so forth covered all the stated big data properties apart from validity. The significance of this property is further discussed in Section 2.2.

2.2 Data Swamp

As mentioned in Section 2.1, a data lake contains data in its native format [MT16]. Since the data does not require transformation, this enables quickly storing large amounts of data with very little effort. Naturally, this tempts people to store anything that might be useful later on. However, integrating any data without a strategy or concept has several pitfalls. A common outcome is a large collection of undocumented data [Riv15]. Lacking documentation results in the loss of knowledge of what data is stored, what it represents, as well as the inability to ascertain and ensure data quality, data lineage and security aspects [Hua15]. In other words, the data loses its expressiveness, therefore, breaking the big data's "validity" characteristic. Not knowing what data is stored or how to find it, renders it useless. If the data is found, its quality is essential for its fitness for use. Moreover, the disability to implement security aspects or data lineage features can have legal implications [Mat17]. For instance, with the enforcement of the General Data Protection Regulation (GDPR) in May 2018, it is lawfully required to provide transparency on data ownership, retention and guarantee abilities for deletion and correction of any form of personal data [Eur18]. In order to facilitate transparency, knowledge is required on exactly what data lies where, what it represents and what it is used for and by whom. A data lake which does not satisfy these aspects is called a data swamp or data graveyard [Riv15; SM]. Essentially, a data swamp is a data lake in which data has become useless as it cannot be found or used in the intended manner, due to quality or legal reasons [Mat17]. Cultivating a data swamp can be prevented by pro-actively planning the data's collection, management and protection through data governance and metadata management.

2.3 Data Governance

Generally, governance is an activity which defines the way in which organizations develop, review and implement strategies [HGHM11]. In contrast, data governance specifies guidelines for handling and maintaining data in accordance with the company's business objectives. To clarify the difference between governance and management: data governance provides a framework for all data management activities by specifying the topics, roles and responsibilities needed for management [KB10]. In short, governance determines what decisions are made and by whom,

whereas management involves executing and implementing these. The main topics handled in data governance are data quality, data lifecycle and data security [KB10; Mat17]. These topics are realized with the help of metadata management. As mentioned in Section 2.2, these are the topics which need to be addressed to avoid the creation of a data swamp.

Data Quality is essential as it determines whether the data can be used for its intended purpose [HGHM11]. If it cannot be used, it becomes useless and there is no need to store it. The quality of data is ascertained through characteristics like accuracy, reliability, timeliness, completeness and accessibility. Accuracy specifies how precisely the data represents the real object or event [FLR94]. Reliability refers to the probability of correctness. Similarly, completeness features to what extent the data contains the expected amount of values. The timeliness feature indicates whether the data describes the current state of reality or is outdated [HGHM11]. Lastly, the degree to which data is retrievable and available is signified through accessibility. These are just a few of many data quality characteristics. In order to retain data quality, the data must comply to defined thresholds for each of these characteristics [KB10]. Defining thresholds is part of the data governance responsibilities. It does not suffice to define a general threshold for each characteristic as the thresholds strongly depend on the context of the data. For instance, 80% accuracy in customer data for an online shop may suffice, whereas 80% accuracy for a medical diagnosis may be inadequate. Hence, first, the context for data must be determined, secondly, thresholds need to be defined and lastly, the data's compliance to these must be checked before integration. Data governance should enable data quality management, which includes processing, storage, maintenance and presentation of the data [HGHM11].

In terms of processing, storage and maintenance, data governance is also used for data lifecycle management. The lifecycle starts with the data's creation, moves on through its storage, usage, sharing and archival and ends with its deletion [Spi]. An understanding of these steps and data's growth trends can be used to find optimal storage media minimizing storage costs [KB10]. Apart from cost issues, this information is relevant for compliance to legislation issues, such as the GDPR, Basel II, Sarbanes-Oxley Act (SOX) or Health Insurance Portability and Accountability Act (HIPAA). Lastly, tracing data through the above-mentioned lifecycle steps helps establish an understanding where security constraints must be enforced, e.g for the protection of sensitive data [Spi].

Data quality and data lifecycle management are enablers for data security. Accordingly, insufficient data quality or lacking information about the data's lifecycle stages may lead to security issues [Kue]. The three pillars of data security are confidentiality, integrity and availability. Confidentiality, which is securing data from unauthorized access, requires restrictions on data access as well as encryption and anonymisation rules. In terms of lifecycle stages, this is especially important in the usage and sharing stages. Integrity signifies that data is free from unauthorized changes. It is guaranteed through extensive data lineage tracking and accountability and authenticity metrics. Lastly, availability is managed through data loss and downtime prevention. In the data's lifecycle, integrity must be guaranteed throughout all lifecycle stages, whereas availability mainly concerns the storage and archival stages. Data governance involves preparing strategies and concepts for dealing with the explained aspects [KB10].

The previously discussed data on quality, lifecycle and security aspects are metadata. Maintaining and handling metadata is a part of the metadata management tasks. As previously mentioned, governance delivers a framework which is an enabler for management jobs. However, metadata

management involves more use cases than the data governance aspect, as depicted in Figure 2.1 [SDE18b]. Both the topics “meatdata” and “metadata management” are discussed in more detail in the following Section 2.4 and Section 2.5 respectively.

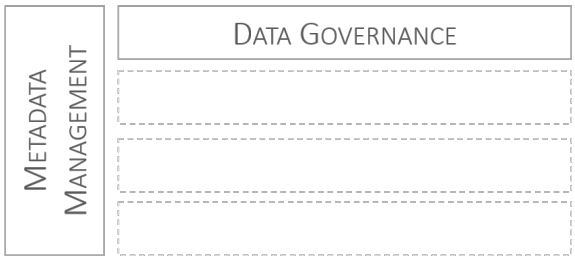

Figure 2.1: Metadata Management and Data Governance Connection

2.4 Metadata

Metadata is information on data context, in other words, data about data [Ril17]. It gives data consumers, including applications, a better understanding of the data and its properties, thereby improving its usability [TSRC]. For example, a book such as “Harry Potter and the Sorcerer’s Stone” might be the data. Metadata on this book would be the author “J.K. Rowling,” the year it was published, the publisher, the warehouse or shelf number where the books are stored and so on.

Metadata belongs to different categories depending on its context [TSRC]. Depending on the category the metadata is relevant for different stakeholders and serves different purposes. For instance, the title and author are related to the books content and are relevant for identifying the correct book, whereas information on its storage is related to sales and logistics.

There are several approaches for categorizing metadata. The National Information Standards Organization (NISO) distinguishes between three types of metadata: *descriptive*, *structural* and *administrative* metadata [Ril17]. Descriptive metadata is used to identify and discover a data set through properties such as author, title or other keywords. Structural metadata describes relations between subsets of data and how these belong together. Lastly, the administrative category comprises *rights management, technical* and *preservation* metadata. Examples for these subcategories are the copyright status, file type and checksum, respectively. The paper “Data Wrangling” classifies metadata as either *schematic*, *semantic* or *conversational* [TSRC]. Schematic metadata covers information on the schema and formatting, thus containing part of NISOs administrative and structural metadata. Semantic metadata is equivalent to NISOs descriptive metadata with additional information on associations, thus covering part of the structural metadata. Lastly, conversational metadata covers all of the information on people who previously accessed the data, their insights and experiences with the data. The idea being that insights and understanding of the data only have to be gained once and are then shared.

Figure 2.2: Metadata Classification Types

Another classification scheme distinguishes between *business*, *technical* and *operational* metadata [Rus17]. Business metadata puts the data in a context, indicating what it represents. Information of this sort provides visibility, thereby enabling self-service data access and discovery use cases. Assuming the date "June 26, 1997" is the data, then the information that this date is "the Harry Potter book's publication date in the UK" would be the according business metadata, thus distinguishing it from other dates. Technical metadata describes the form and structure of the data. It is used for storage, maintenance and re-use purposes. For example, the data is of the type "date" and stored in the column "publication date" in the table "books." Lastly, operational metadata captures the data's provenance, quality, lineage and legal requirements. It is used to implement capacity, lifecycle as well as security and privacy requirements. This could be anything ranging from the data's creator, the last modification, the modificator, access rights or quality attributes. Table 2.1 contains more example metadata for each of the three categories. This classification is used by companies which develop metadata management solutions, such as IBM or Informatica, and will therefore, also be the classification of choice for this thesis [IBM; Inf].

Category	**Examples**
Business:	book title, book publisher, book author
Technical:	size, data type, table name, column name
Operational:	access rights, provenance, quality requirements

Table 2.1: Metadata Classification

2.5 Metadata Management

Metadata management involves activities centered around organizing and handling information assets with metadata [SDE18b]. Its primary goal is to maximize the value of the stored data [Sim12]. Generally, metadata management is performed on the level of a project, single program or initiative [SDE18a]. If the scope ranges across the entire organization, it is called Enterprise Metadata Management (EMM) [SDE18b]. As depicted in Figure 2.3, metadata management starts with the creation of a data inventory [SDE18a]. Once it is clear what data exists, it can be assessed and analyzed. Data is analyzed to find either additional datasets or metadata. If found, these are used to enrich the data, thereby adding value.

Figure 2.3: Metadata Management Practices (adapted from [SDE18a])

Gartner maps the practices illustrated in Figure 2.3 to four metadata management use cases, listed in Figure 2.4 [SDE18b]. The inventory practice is mapped to the data governance use case. The link to governance lies in the fact that an inventory is a result of proficiently governed metadata and data. It enables the creation of a business glossary, an audit trail, impact analysis and more. According to Gartner, the data governance use case boils down to supporting the implementation of policies and rules. The second use case, risk and compliance, is mapped to the assessment practice. Data elements need to be mapped and identified, the processing and associated risks need assessing and the lineage as well as impact analysis need to be managed continuously to enable the realization of risk and compliance requirements. This use case also includes managing data access according to risk and security needs. The third use case identified by Gartner is data analysis. It delivers insight into the organization's previous performance. The goal is to learn from previously leveraged metadata in order to improve in the future. Lastly, the "value" practice also constitutes a metadata management use case. In contrast to the data analysis use case, its attention lies on future insights. The use cases each depend on the previous ones.

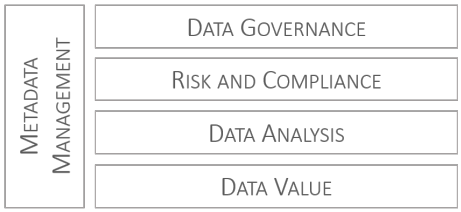

Figure 2.4: Metadata Management Use Cases

According to Gartner, the metadata management landscape is comprised of data sources, other existing metadata systems and the central metadata management system, as depicted in Figure 2.5 [Sim12]. Within this landscape metadata from several sources is consolidated in a metadata database. The source can be both another metadata system and the data sources. A metadata discovery engine is used to derive metadata from the sources. All of the configurable aspects can be addressed through the management interface. Lastly, the data catalog provides a presentation layer. However, metadata management products, such as data catalogs, seem to offer both the presentation as well as configurable aspects of metadata through one interface [Ala; SDE18a].

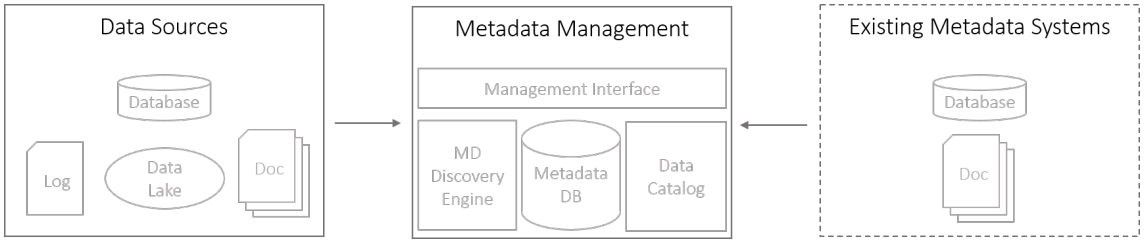

Figure 2.5: Metadata Management Landscape (adapted from [Sim12])

According to Gartner, more than 50% of organizations manage their data and metadata in a reactive manner, as a opposed to a proactive fashion [SDE18b]. However, Gartner also predicts that the organizations will have to adapt their metadata management in the near future. For instance, the new requirements for the GDPR forced European companies to redo their metadata management, which was expected to cost them an average of 1.3 million euros. Gartner also estimates that the companies' investments in metadata management will double by 2020.

2.6 Data Lake Architecture Levels

The architecture described in this section is a framework for the creation of a data lake [GGH19]. It will be used as the basis for the creation of the metadata management concept illustrated in Chapter 5. The architecture divides all processes, tasks and components related to the data lake into levels depending on their focus and abstraction level. This includes tasks and processes described in the previous sections, Data Governance and Metadata Management. As depicted in Figure 2.6, the architecture comprises a total of six levels: the *data process, data services/access, data organization, data model, data storage* and *metadata* level. The use cases, metadata management concept and prototype developed in the scope of this thesis are based on these levels.

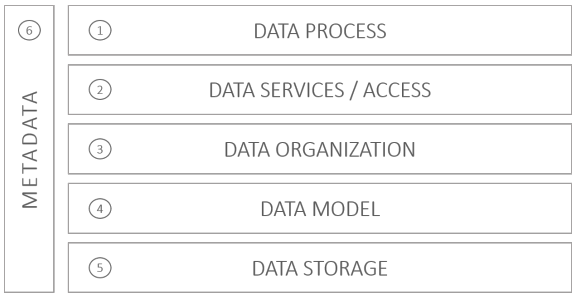

Figure 2.6: Data Lake Architectural Levels (adapted from [GGH19])

1. The **Data Process Level** is the most abstract level [GGH19]. It provides information on how data is inserted into or extracted from the data lake. The focus lies on the process steps of the insertion and extraction. The process level also includes the steps taken to finding the right data. The steps are abstract enough that the data lake remains a black box and neither user groups nor interfaces are defined. Figure 2.7 illustrates an example process, which covers

both data ingestion and extraction. The process begins by finding a new source, cataloging the content and then adding the content to the data store. Subsequent steps include the discovery of data through the catalog, checking if it is the right data and finally accessing the data.

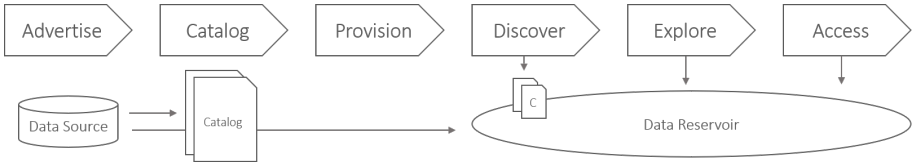

Figure 2.7: Data Reservoir Process (adapted from [CSN+])

2. The second level, **Data Services**, defines how users can access the data lake [GGH19]. User groups, services and interfaces are specified on this level. Furthermore, it can also include specific definitions for offered functionality. The definitions are abstract and the data lake is viewed as a black-box. Organizations today are challenged to offer services on the data lake for progressively diverse use cases [Rus17]. Self-service interfaces enable users to access services independently without having to address a contact person from IT. This optimizes costs and enables a faster delivery of results [Rou]. For example, Figure 2.8 roughly depicts a set of interfaces and potential user groups.

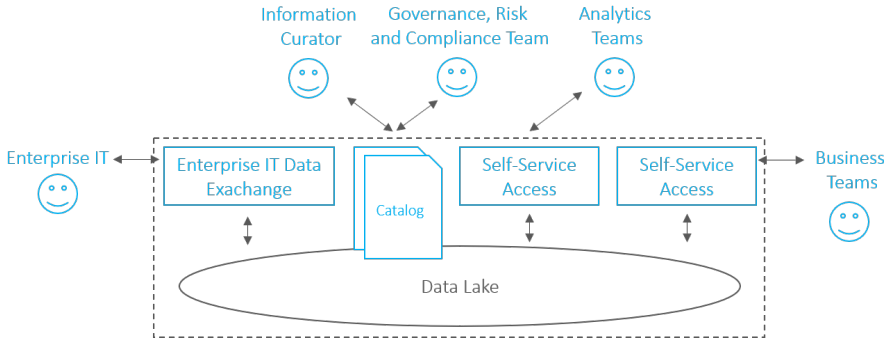

Figure 2.8: User Groups and Interfaces (adapted from [Ram])

3. The **Data Organization level** provides information on the separation and organization of data within the data lake [GGH19]. There are two approaches for organizing the data, firstly, assigning it to so-called *zones* and secondly, storing it in *ponds* [Inm16; LS16]. The former organizes the data according to user needs [LS16]. A user will find all of the relevant data in a single zone. The latter approach organizes the data according to the properties of the data [Inm16]. As opposed to the zone approach, the user groups and their needs are not taken into consideration. Consequently, users might have to access multiple ponds to collect the data relevant to them. In the pond approach a dataset is stored in exactly one pond. Using zones, the same dataset may exist in several zones simultaneously in different transformation degrees [LS16]. How data is transformed and moved between the zones varies depending on the zone model.

Figure 2.9 illustrates a zone architecture created by Zaloni [Zal]. This architecture focuses on four zones, the *transient loading zone*, the *raw zone*, the *trusted zone, refined zone* and a *sandbox*. The transient loading zone is an optional zone which contains data loaded directly from the sources. Basic quality checks are executed on the data in this zone. If the data passes the checks, it is moved into the raw data zone. As opposed to all other zones, the data in the transient loading zone is only stored temporarily. The focus of the raw zone lies on identifying and masking sensitive data, cataloging the data and enriching it with metadata. Data scientists and business analysts have access to this zone. The trusted zones data is transformed to fit business needs and corporate policies. Data transformed to a common format or in any other form to fit a specific line of business is found in the refined zone. If data scientists wish to perform wrangling, discovery or exploratory analysis, the data is moved into the sandbox zone.

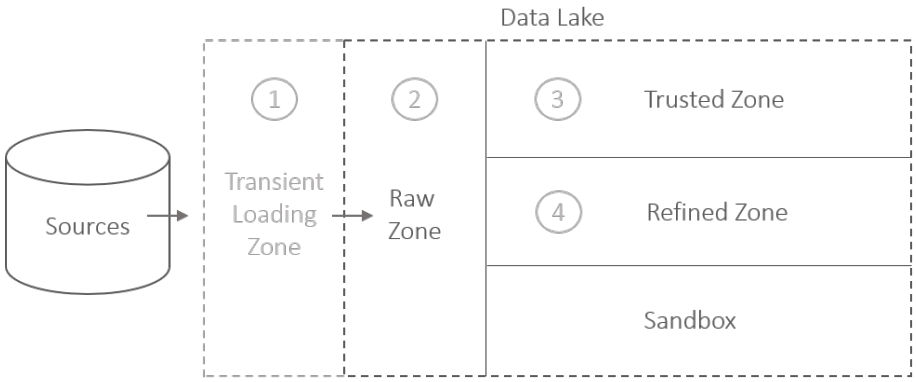

Figure 2.9: Zaloni's Data Lake Zones (adapted from [Zal])

Inmon's data ponds are based on a distinction of three data types: *analog, application* and *textual* data [Inm16]. Analog data is produced in an automated fashion by some machine or device and is very voluminous and repetitive. Measurements of room temperature could be analog data. Application data is produced by an application and is also repetitive. For example, sales, payment and shipping information used in any application fit this category. Lastly, textual data is unstructured and non-repetitive data, such as call center conversations. Figure 2.10 illustrates a structure based on the previously mentioned data categories. To begin with, all of the data is loaded into the raw data pond. It stays there until it is needed for analysis. Depending on the type of data, it is transferred into the according pond where it is processed. When it is no longer of use, it is moved into the archival pond. Since the data is moved and not copied from one pond to the next and is then further processed, the original data is lost in this approach.

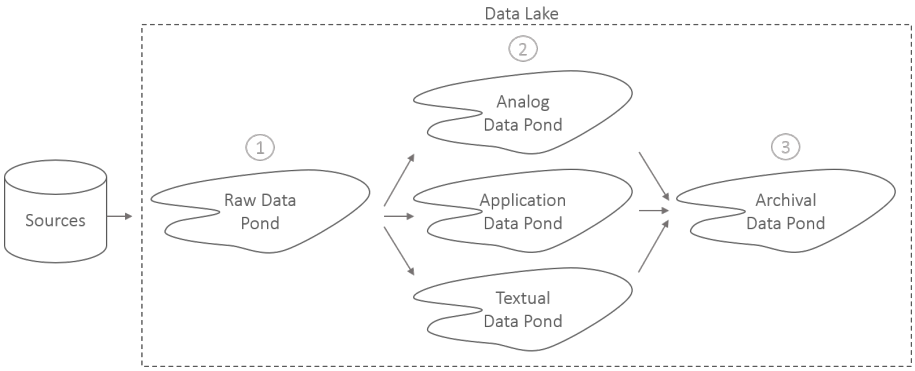

Figure 2.10: Inmon's Data Ponds (adapted from [Inm16])

As the raw data is lost in Inmon's approach this thesis uses the zone approach.

4. The **Data Model level** comprises information on the structure and relationships of data elements [GGH19]. In data warehousing, the Star and Snowflake schema or Data Vault Model are commonly used to model the schema. The Data Vault Model is most suited for data lakes as it is very flexible and extensible [NRD18]. As data lakes contain semi- and unstructured data as well, modeling is only possible to a limited extent. There are other methods for defining the data's structure. Keeping track of related elements in the form of a list is called a Taxonomy [IL15]. If relationships are defined in between the elements of the list, it becomes an Ontology. Figure 2.11 illustrates an example ontology, modeled with the RDF. In this case, the relationship between the data element book and its three properties year, title and isbn is depicted. Extensive modeling is contradictory to the data lake's schema-less characteristic and should be kept in mind when constructing data models.

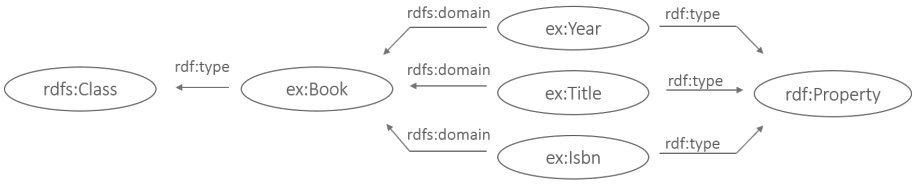

Figure 2.11: Relationships expressed through RDF

5. The lowest level, **Data Storage**, contains information on the physical storage of the data [GGH19]. The storage requirements depend on the structure of the data. For example, the data lake's storage solution may be a Relational Database (RDB), a NoSQL database or some combination of these. Depending on the data, the choice of storage solution could be based on properties such as scalability, query language, the supported data structures or necessity of schemata.

6. The **Metadata Level** is a cross-sectional level, covering all levels starting with the storage level, up until the Process level [GGH19]. Each level has metadata. It differs in terms of what the levels focus is. Table 2.2 depicts an extract of metadata for a dataset throghout all levels.

There are two options for storing metadata. It can either be stored with the data it belongs to or in a separate metadata repository [Lay13]. A metadata repository is a database used for managing metadata [KBM10]. The former option of storing metadata with its data has the disadvantage that searching the metadata requires looking at each data element in order to collect its metadata [Lay13]. An index for the collected metadata would improve the search. Nonetheless, this circumstance makes the former option inapt for metadata management purposes and thus, is not considered in the further course of this thesis.

Data Process	process.ids: “1,5,9...”
Data Services/Access	usergroups.access: “admin, analytics” user.access: “author, editor”
Data Organization	zones: “Loading Zone, Raw Zone, Trusted Zone”
Data Model	book.properties: “Title, Year, Isbn...”
Data Storage	path.tz: “hdfs://ex.metadata.com/TrustedZone/book” path.rz: “hdfs://ex.metadata.com/RawZone/book” format: “txt”

Table 2.2: Metadata Examples for each Level

2.7 Data Storage Systems

In order to demonstrate the metadata management concept created in the course of this thesis, a storage system is required for persisting the metadata in question. This section presents two categories of databases, as illustrated in Figure 2.12. Relational databases are considered and presented in Section 2.7.1, as these have prevailed over other types for many years [Cat10]. The second type considered are the NoSQL databases, as these are gaining in significance in the big data context. These are presented in 2.7.2. The information provided will be used for choosing the storage type for the metadata repository.

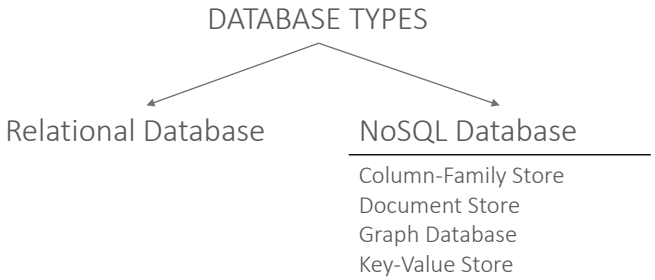

Figure 2.12: Database Types

2.7.1 Relational Databases

Relational databases are an established standard and it is the database type which had the highest market share for several decades [Cat10]. They use the Structured Query Language (SQL) to manipulate and query data. SQL is also a widely spread standard. Furthermore, relational databases require a pre-defined schema and are based on the ACID properties: *Atomicity, Consistency, Isolation* and *Durability* [V14]. Due to the Consistency property, relational databases don't work well in a distributed environment, as retaining a valid state in a distributed environment is expensive in terms of performance costs. It requires constant synchronization and locking. Consequently, relational databases do not scale well horizontally, as this increases distribution.

2.7.2 NoSQL Databases

In contrast to relational databases, NoSQL databases do not require a fixed schema and can usually scale horizontally as they do not support the ACID properties [SF13; V14]. According to the CAP-Theorem by Eric Brewer, a distributed service cannot be simultaneously *consistent, available* and *partition-tolerant* [V14]. In accordance to the CAP-Theorem, most NoSQL databases support the BASE properties: *Basically Available, Soft State* and *Eventual Consistent*. The focus of BASE lies on availability, as opposed to consistency as with ACID.

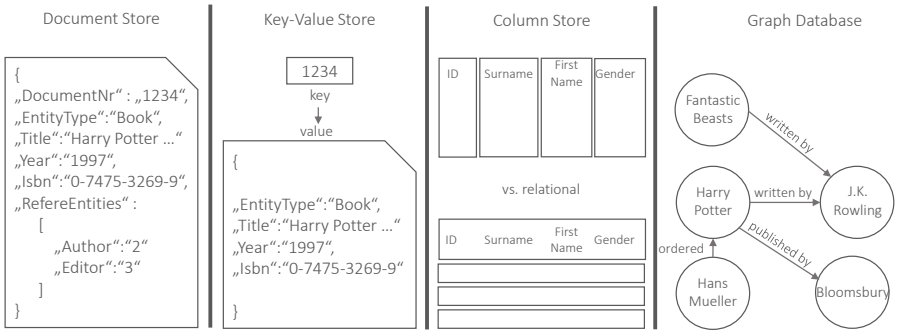

Figure 2.13: NoSQL Database Types (adapted from [Klö])

The subtypes of NoSQL databases are the *column-family store*, *document store*, *graph database* and *key-value store*. Figure 2.13 illustrates some of their fundamental properties and differences.

- **Column-Family Store:** Apart from Column-Family Store, this kind of database is also called a Column Oriented Store, Extensible Record Store or Wide Columnar Store [V14]. In a way, this kind of store is similar to the relational database. Instead of being stored in a row-like fashion, the values of the entire columns are stored sequentially. Therefore, if only a few column values are required, as opposed to entire rows, then the column-family store has faster read requests [KR]. Furthermore, the column wise storage supports horizontal scaling. Each entry in a column consists of the dataset's key, the column value and timestamp [SF13]. The timestamp is used to deal with write conflicts, expired data and so on. If they are often accessed together, columns can be grouped into column families. If a column consists of a

map of columns, it is called a super column. A row in a column-family store is a collection of columns linked to a specific key which enables the stores specialty that each row may contain a different set of columns.

- **Document Store:** Datasets are stored in documents, which are uniquely identifiable by their key property, "ID" [V14]. Standard formats for the stored documents include BSON, XML, PDF and JSON [SF13; V14]. The documents are schema-less and may be structured diversely, containing very different content. The content of the documents can be queried. Documents may be embedded within each other or contain a property referencing one another [SF13]. The document store is well suited for storing diverse datasets, not, however, for heavily linked data [V14].
- **Graph Database:** A graph consists of nodes and edges [SF13]. The nodes represent entities and the edges the relationships between the entities. Edges have directional significance and a property defining the relationship. Nodes can be connected though several edges. Depending on the underlying graph database concept, nodes and edges can have attributes. This kind of database is specialized on managing strongly linked data. Its structure eliminates the need for complex queries with intensive joins and enables an efficient traversal through index-free adjacency [KR]. As opposed to most other NoSQL databases, graph databases are often ACID-compliant [SF13]. Scaling a graph database is not as easy as with other NoSQL databases. Splitting the graph and distributing it among several servers would result in performance loss while traversing it. An option for horizontal scaling is sharding through domain-specific knowledge, that is, according to content. The applications using the database must be built to accommodate this kind of sharding. Graph databases are suited for use cases with strongly linked data.
- **Key-Value Store:** The key-value store is somewhat similar to the document store. Each dataset consists of a key and value [V14]. Similarly to the document store, the value can be a blob, text, JSON, XML etc., and is schema-less [SF13]. As opposed to the document store, the key is stored outside the document [V14]. The values are uninterpreted byte arrays, thus, they cannot be queried and are only accessible through the key. The key-value database is not well suited for querying by data, storing relationships among data and multioperation transactions [SF13].

Even though all of the presented subtypes are classified as NoSQL databases, they are very different in their properties, strengths and weaknesses. The evaluation, which classification and sub type is best suited for the metadata repository, is conducted in Section 6.1.

3 Related Work

This chapter provides an extract of both available products and research conducted on four topic areas related to metadata management and data lakes. The topics include modeling metadata, general purpose metadata management systems, data lake specific metadata management systems and lastly, metadata models explicitly created for data lakes.

Metadata Model: The purpose of metadata models is the provision of a general schema for structuring metadata. The models are not necessarily designed for a specific storage solution. An example model is the Metadata Model for Data Goods (M4DG) [MDSB]. It was developed in order to enable trading and selecting data goods. The metadata model based on the Asset Description Metadata Schema (ADMS) and designed for describing data sources in a standardized way with a defined set of properties. An inventory software was developed to manage the data and support data governance processes. The software collects all data sources in a central document-based datastore, according to the uniform model. The demonstrated metadata model and implementation are distinctly created for the use case of trading data and are also not specifically designed for data lakes. Although the topic is somewhat similar, this thesis aims to provide a general purpose metadata management concept which is specifically created for data lakes and their architecture.

General Purpose Metadata Management Systems: Work available in this area provides general frameworks for managing metadata. These often provide additional functionality, such as a keyword search on metadata, to improve the management aspects. The general frameworks often work for a variety of storage solutions. Google Dataset Search (Goods) is an example for such a system [HKN+16]. It is created for organizing Google's structured datasets distributed throughout a variety of storage systems. It extracts metadata from the systems and aims to provide a global and unified perspective on the data. In contrast to Enterprise Data Management (EDM), in which actions on data have to be performed through this one system, Goods enables users to work with a wide variety of tools and systems and operates in a post-hoc, non intrusive fashion in the background. It provides services for accessing, monitoring, annotating and analyzing data. Goods may be applicable to data lakes, is, however, not specifically designed for them. It also only covers structured data. Furthermore, the underlying metadata management concept is not accessible in detail.

Besides Goods, a wide range of metadata management solutions exist on the market which offer functionality such as metadata repositories, provision of a business glossary, lineage functionality, impact analysis, rule management, semantic frameworks as well as metadata ingestion and translation features [SDE18a]. With the listed features, they cover many metadata management use cases and many are applicable to a variety of storage solutions. According to Gartner's ranking of August 2018, the vendors Alation, Collibra and Informatica offer the most capable solutions such as the Alation Data Catalog, Collibra's Data Governance Center, Catalog and Collibra Connect as well as Informatica's Enterprise Data Catalog, Metadata Manager, Business Glossary and Axon Data

Governance. This is a small extract of solutions which are specifically designed for metadata management tasks. With all of these solutions the underlying metadata management concept is not readily accessible to the public, thus not providing the sought-after concept.

Data Lake Specific Metadata Management Systems: In contrast to the aforementioned topic area, work in this area was designed specifically for the data lake concept. Constance, a data lake system with metadata management functionality, is one such example system [HGQ16]. As opposed to many application, project or organization specific data lake solutions in practice today, Constance was developed as a more general data lake solution. It provides a framework for managing structural and semantic metadata with an integrated user interface. Amongst other functionality, it can extract metadata, supports matching semantic metadata and its enrichment. It also incorporates search functionality through a simple keyword search as well as a formal query language. As with the other systems mentioned so far, Constance's metadata management concept is not openly accessible. Furthermore, this thesis aims to provide a metadata management concept which can be applied to an existing data lake and doesn't come with the storage included.

Another example is the metadata management solution called Generic and Extensible Metadata management System (GEMMS) [QHV16]. It was developed explicitly for data lakes and its functionality includes extracting metadata from heterogeneous sources, storing it according to an extensible metamodel and providing annotation and querying services. In contrast to all other products and systems so far, GEMMS corresponds most to the content of this thesis, especially because the metadata management concept is accessible. In which respect the concept differs from the one in this thesis, is addressed in the next paragraph.

Metadata Model for Data Lake Systems: There are concepts for metadata management explicitly for data lakes with metadata models that are accessible. As mentioned in the previous paragraph the system GEMMS provides a generic model for metadata. However, even though it has an accessible metadata model for data lakes, it does not incorporate the characteristics of an underlying data lake architecture, which the concept of this thesis requires. The same is true for the metadata storage methodology introduced in the paper "Personal Data Lake with Data Gravity Pull" [WA15].

Many approaches have been developed for modeling and managing metadata throughout a variety of systems. Nevertheless, a general purpose metadata management concept for data lakes, which is based on a data lake architecture , does not exist and is therefore provided through this thesis.

4 Self-Service Use Cases

To demonstrate the necessity for metadata management and show an explicit application for the concept designed in this thesis, this chapter describes typical data lake use cases. Furthermore, the use case scenarios are implemented in the prototype and later used as the basis for the discussion of the concept and prototype.

In order to establish realistic use cases, a hospital example scenario is used. Hospitals can collect data on a variety of topics. For example, data on employees like doctors, nurses and so on, as well as personal and demographic data on patients are required. Furthermore, information on diseases, medications, treatments and operations are of interest as well as conducted research and experiments in the medical field. The data is of diverse structures, for example, employee data is structured, a report may be semi-structured and therapeutic audio recordings are unstructured. Moreover, the data can be processed in different ways. For instance, it can be used to generate reports on recoveries and so on. In this scenario the data is stored in a data lake, therefore supporting the persistence of diversely structured data of diverse topics and various processing options. In this context, the three use cases Data Inventory, Data Lineage and Data Access were created. Generally, many use cases involve some form of search in the metadata repository. While a keyword search is a standard feature in many metadata management solutions, this thesis focuses on other use cases which better highlight the necessity for metadata management. Nevertheless, the concept and prototype enable such a keyword search in one form or the other.

4.1 Inventory Use Case

In the first use case, a hospital employee requests an overview of data stored on a specific patient. The data may be needed for checking medications or investigating lab results. The first step in retrieving the requested data is finding out what data is generally stored on patients. As data lakes usually persist a large amount of very diverse data, a *data inventory* needs to be maintained, which provides information on what data is stored. Finding data related to a specific topic without an inventory would involve crawling the entire content. The latter option is highly inefficient and error prone. The inventory must comprise some sort of list on what data resides in the data lake. This list should contain a description of the data and information through which the data can be accessed. As a data lake will likely contain a large variety of data, the list should cluster the data in one way or another, providing some structure.

In the given scenario, a hospital employee will browse the inventory and compile a list of data collected on the patients. As depicted in Figure 4.1, the inventory could constitute structured information like tables on the patients and their treatments as well as unstructured information such as radiograms or audio recordings of therapy sessions. Within the inventory list shown below, the

data elements are grouped by their source. In order to answer the hospital employee's request, the tables containing patient information, i.e the patient and treatment tables as well as radiology source etc., need to be filtered for entries on the patient.

Source Name	Element Name	Element Description
Bellvue Center DB	Patient Table	Personal information on hospital patients
	Treatment Table	Treatments prescribed for patients…
	…	
Bellvue Center Radiology Images	Patient X – Radiology Knee	Radiology img of knee taken for treatment xxx …
	…	

Figure 4.1: Data Inventory

4.2 Data Lineage Use Case

Data may be used in a variety of ways, such as deriving new data sets or generating reports in order to gain new insights. The data lineage use case is based on the scenario that a hospital employee, like a doctor, wishes to know what data was used to generate a specific report. Figure 4.2 depicts an example of how data elements may originate from one another. The lineage use case takes a specific data element as input. The output contains a collection of elements from which the specified element was derived and a collection of elements, which were in turn derived from the specified element. Compiling the lineage information requires metadata mapping all elements together which are derived from each other.

In this scenario a doctor might want to know what data the patient recovery report is based on. As depicted in Figure 4.2, the report is derived from the treatment table. Further investigation on the treatment table can then yield the result that the treatment table in turn is derived from both the disease and patient table. This analysis can be continued as required.

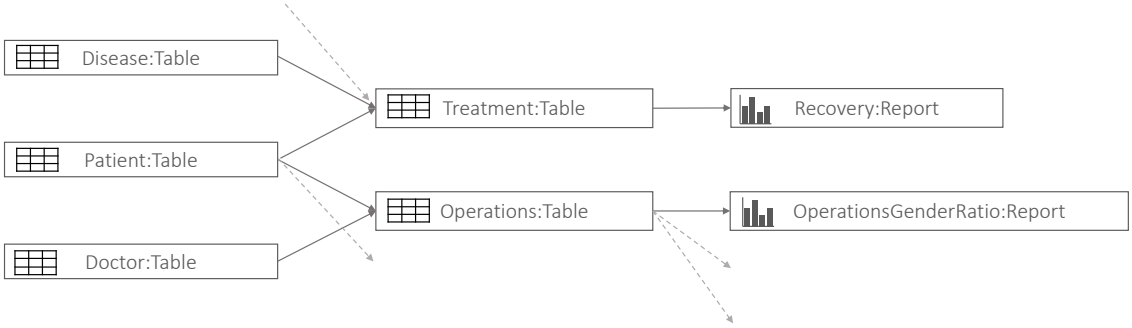

Figure 4.2: Lineage Extract of Patient Data

4.3 Data Access Use Case

Given the scenario that a data element, in this case the treatment table, contains incorrect data and a hospital employee wishes to find out who performed an action on the data in question. The use case output should return a list of people that performed an action on a specific data set. As demonstrated in the previous lineage use case, information, including personal information, may be propagated into other data elements. As the incorrect data might have originated from another data element, the hospital employee might require a comprehensive list of actors accessing data elements along the entire lineage path. Figure 4.3 depicts an extract of the treatment tables lineage information with actors that performed actions on some part along the path. With the given lineage information, the access information can be retrieved for each data affected element. In the given example, the required access information would include the access information on the treatment table and elements in the earlier stages, namely the disease and patient table.

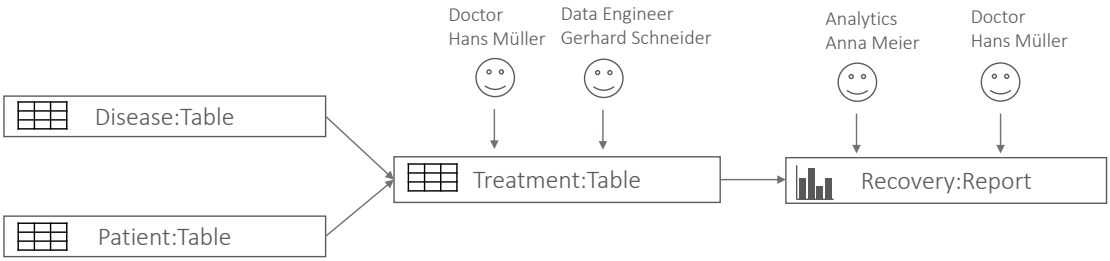

Figure 4.3: Data Access Log

5 Metadata Management Concept

The previous chapter introduced several use cases which underline the necessity for metadata management and consequently, the necessity for some storage concept for the metadata. This chapter introduces and explains the metadata management concept for data lakes which has been developed within the scope of this thesis and should enable implementing the described use cases. The concept is introduced in the following two sections. The first one, Section 5.1, demonstrates the conceptual metadata model. The model consists of both a core model and use case specific extensions. The second section, Section 5.2, contains specific models for the use cases described in Chapter 4.

As mentioned in Section 2.6, the storage solution of choice for the metadata is a separate metadata repository, as opposed to storing it with the data. The concept is explicitly designed for a separate metadata repository.

5.1 The Conceptual Metadata Model

The concept established in the course of this thesis aims to be applicable to a wide variety of metadata management use cases. Therefore, the concept is designed to consist of an abstract core model and use case specific model extensions. The core model is abstract enough to cover every variation of metadata. The extension models are a suggestion on how parts of the core model may be designed in more detail. These are designed for very specific scenarios. For example, the first extension, described in Section 5.1.1, explains how to collect metadata throughout different zones and assumes the data lake is structured according to Zaloni's zone model as explained in Section 2.6. The second extension, illustrated in Section 5.1.2, explains how to collect relational data on varying granular levels. The last extension is also based on relational data and explains how its schema can be stored in Section 5.1.3.

Figure 5.1 illustrates the core metadata model for the metadata repository. The *data* entity, in blue, forms the basis of the model. In order to avoid storing data redundantly, the data entity in the metadata repository represents a pointer to the data in the data lake. The path to the location is stored in the *storageLocation* attribute. Thus, the actual data can only be retrieved by following this path. A data element can be connected to zero or many other data elements. For instance, a data element representing a table's row can be connected to the superior table data element. The data has two entities attached to it, the *zoneIndicator* and the *granularityIndicator*. The zoneIndicator is a reference to the data lake architecture's organization level, described in Section 2.6. As a recap, the organization level supplies information on how the data is organized and separated within the lake, i.e. in zones or ponds, however this thesis only focuses on zones. In order to understand within which zone the data is stored, the data entity has exactly one zoneIndicator. Section 5.1.1 contains an extension of the core model, describing how the zoneIndicator could be used with

Zaloni's zones. The data entity has zero or one granularityIndicator. This entity is strongly tied to the data's structure. It enables collecting metadata on variable granular levels. For instance, if the lake contains data with a relational structure, metadata could be collected on the schema, table, row, column or field level. More details and an example on the granularityIndicator are offered in Section 5.1.2.

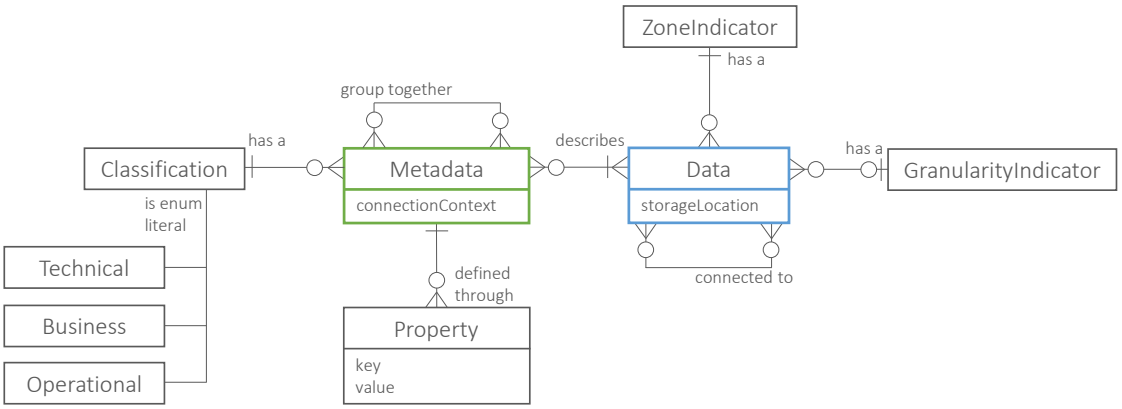

Figure 5.1: The Core Metadata Model

The second central entity in the model is the *metadata* entity. A metadata element is connected to one or more data elements and each data entity may have zero or many metadata entities connected to it. For example, when tracking who accessed the data element, one could store the people accessing the data as metadata elements. The data may have been accessed by more than one person, thus, it would have several "person" metadata elements attached to it. A person may have accessed more than one data element. Therefore, the person metadata element is connected to several data elements. An attribute called *connectionContext* describes what information the metadata element contains. For example, a metadata element containing a description of the data may have a connection context called "content description." In other words, the connection context sheds light on which aspect of the data is described. The information itself is stored in the form of key-value pairs, modeled as *properties*. A metadata entity may contain zero or more properties. Continuing the scenario from chapter 4, properties might be: "name: Patient Table" and "longDescription: General information on patients." The metadata entity's self-link enables it to group zero or more other metadata elements together. A metadata element may belong to zero or many groups. Grouping the elements according to some sort of context is helpful if a lot of metadata is collected on the same topic for a single data element. Continuing with the data access example, over time the data element might be swamped with metadata on accesses in addition to the other collected metadata. Grouping these, by connecting them to an intermediate metadata element with the "connectionContext: access" adds structure and reduces the number of immediate connected elements as demonstrated in Figure 5.2. While the groups add more structure and improve the comprehensibility of the metadata, they are an extra intermediary node which might impact the performance when querying the metadata. Therefore, the groups are optional and should be used depending on the metadata management use case's requirements. As described in Section 2.4, metadata can be classified as either *business, technical* or *operational* metadata. In the model, the categorization is defined through a *classification* entity. Each metadata element has exactly one

classification assigned to it. The three types are modeled as enumerations. The previous example's description element would be assigned a business classification as it puts the data's content into a context, whereas the access and people elements are operational information and are, therefore, classified as operational.

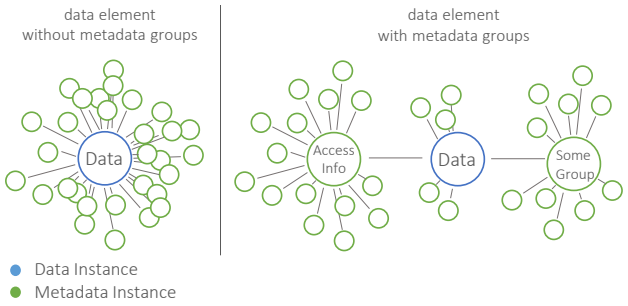

Figure 5.2: Comparison of a Data Element with and Without Metadata Groups

Figure 5.3 depicts an example instantiation of the core model. The data element has a storageLocation containing the path to the data in the lake. The granularityIndicator "table" implies that the path points to a table. Furthermore, the referenced content is stored in the raw zone, implied through the zoneIndicator. The data element has one metadata element, which describes the data's content, as declared through the connectionContext. The metadata element has the properties "name" and "longDescription". Lastly, it is classified as a business metadata element.

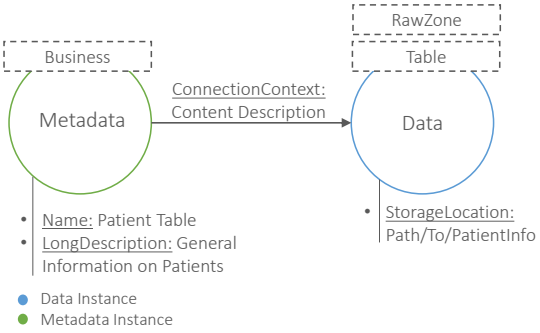

Figure 5.3: Example Instance of the Core Metadata Model

5.1.1 The Zone Indicator

The model demonstrated in this subsection is a use case specific extension of the core model intended for data lakes which use a zone model. It is based on Zaloni's data lake zones which are described in Section 2.6. Figure 5.4 shows the zoneIndicator model.

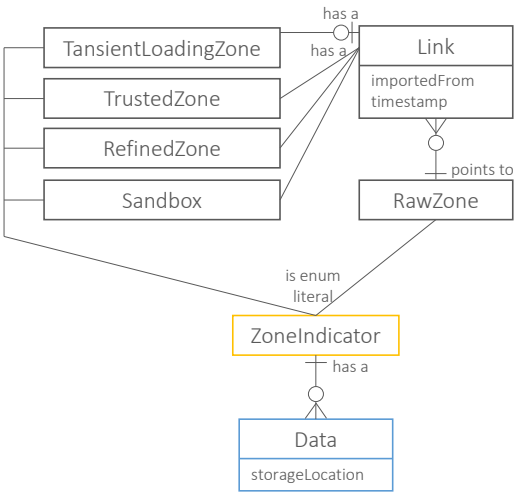

Figure 5.4: The ZoneIndicator Model

As mentioned previously, the zoneIndicator entity is a label supplying information about the location of the data element in the data lake's zone architecture. Depending on the zone definition, the data's transformation degree is immediately apparent through it. Zaloni's zones are modeled as enumerations for the zoneIndicator. In order to use another kind of architecture, the zone enumerations simply need to be substituted and the relationships between the enumerations need to be changed to fit the new architecture.

The model illustrates that every data element must have exactly one zoneIndicator, but the indicators may be applied to zero or many data elements. In this model, the *rawZone* entity is designed to be the central zoneIndicator. The reason behind this design decision is that the data is sometimes directly loaded into the raw zone, even though it is the second zone, because the first zone, the transient loading zone, is optional and can be omitted. Furthermore, data loaded into the transient loading zone is only stored temporarily and may not move on into other zones if it does not pass the quality checks. Consequently, if data is stored in any of the other zones, it will have a corresponding data element in the raw zone, however, not necessarily in the transient loading zone. Thus, the raw zone is the most stable reference. The other zones, the transient loading zone, trusted zone, refined zone and sandbox, have a *link* entity, connecting them to the corresponding data element in the raw zone. The information from where the data was imported into the zone as well as the corresponding timestamp is stored with the link. The *importedFrom* attribute may contain the name of a zone or the original source. Within Zaloni's zones, the data should first move through the transient loading zone, the raw zone, the trusted zone and lastly, through the refined zone. In the case of the raw zone and sandbox, the data may be loaded directly from the source. The importedFrom attribute enables tracing the data's progress through the zones. As the data may not be moved into the raw zone from the transient loading zone this enumeration can exist without a link to the rawZone element. If it was moved into the raw zone, then it must have a link connecting them.

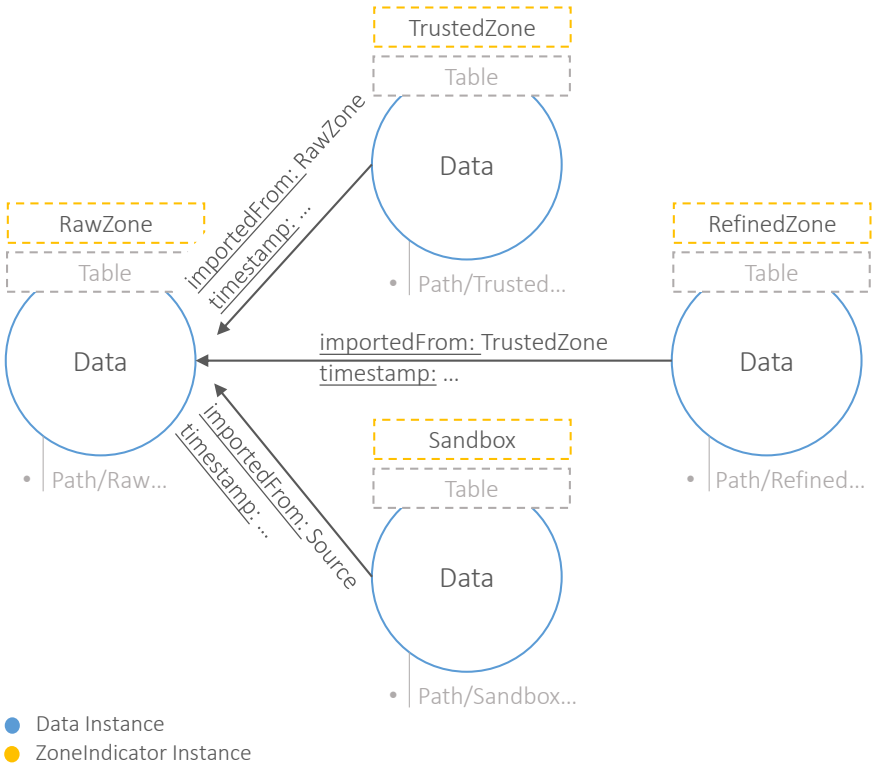

Figure 5.5: Instance of the ZoneIndicator Model

Figure 5.5 shows an exemplary instantiation of the zoneIndicator model. The four illustrated data elements are all versions of the same table stored in different zones. The left element is stored in the rawZone and, therefore, points to the original unaltered version of the data. The other data elements are connected to it through a link, illustrated by an arrow. As depicted, the data has moved from the raw zone into the trusted zone and then into the refined zone. It was also loaded into the sandbox. The importedFrom attribute specifies that the data element in the sandbox was not loaded from any zone in the data lake, but from the source directly.

5.1.2 The Granularity Indicator

Like the zoneIndicator model, the granularityIndicator model is an extension to the core model. As described in Section 5.1, the granularityIndicator entity enables collecting metadata on different granular levels. The granularity levels are closely tied to some kind of structure in the data. For example, the object, key, value or key-value pair instances in a JSON Document may be used as granularity levels. They are not limited to "structured data." For instance, videos are often categorized as "unstructured data" and yet, one may want to collect metadata on single frames of the video. In this case, there would be a video and frame level.

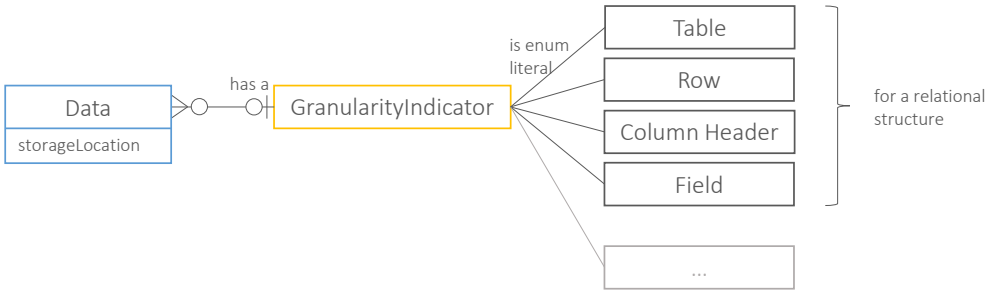

Figure 5.6: The GranularityIndicator Model

The granularityIndicator model, illustrated in Figure 5.6, defines that a data entity has zero or one granularityIndicator. A few example enumerations are listed, which can be used to indicate the levels of relational data. The "..." indicates that other enumerations may be added as needed. In order to collect metadata on different levels, a corresponding data element must be created that points to the granular instance. So, there may be a set of data elements all referring to the same data set, simply pointing to more or less specific granular levels. Figure 5.7 depicts an exemplary scenario with three data elements all pointing to the patient table. The element on the left has a "table" label, meaning it points to the overall table. The middle element has a "column header" label and points to the exact location of the "first name" column. The third element has a "row" label and points to a specific row, in this case the first one.

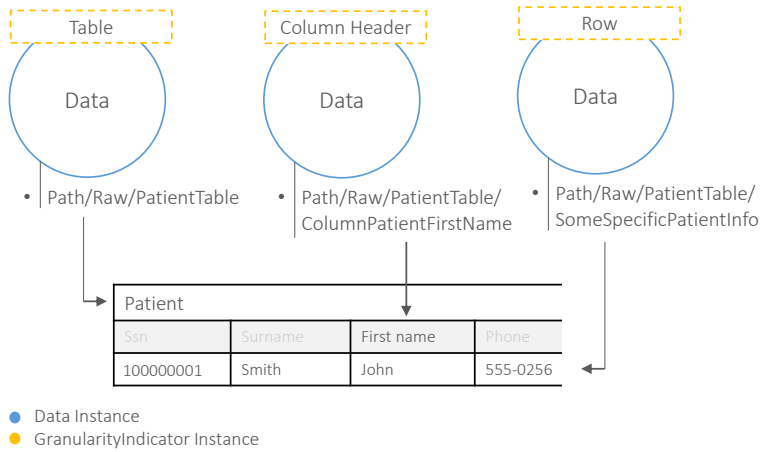

Figure 5.7: The GranularityIndicator Example

5.1.3 Schema Modeling

When storing data in the data lake, retaining information on the original structure is essential. It enables understanding data's relations and its context. The structure is usually defined in a schema. So far, the proposed models do not cover all of the information required for storing the schema. The previous section 5.1.2 provides a list of elements which constitute the overall structure. However, information on relations in between the data elements is missing entirely. The model presented in this section takes the granularityIndicators from the last section and uses them, among other things, to map parts of the schema. Section 5.1.2 explains that structured data can be found in many forms, such as relational data, xml files, json files and so on. As discussed, each form requires an individual set of granularityIndicators. Like in Section 5.1.2, the following subsection specifically focuses on modeling the schema of relational data. For storing schemata of otherly structured data, an according model has to be designed with the corresponding granularityIndicators.

Element connection

To begin with, a set of granularityIndicators needs to be established, as described in 5.1.2. In the next step, the connection between the granularityIndicators needs to be defined. This involves specifying the order from higher to lower granular levels. For example, a definition is required that a table element has rows, which in turn consist of fields. From here, the connection between elements with the same granular level needs to be determined. For example, this involves specifying how tables within a schema are interconnected. Figure 5.8 depicts an example of how the schema of relational data can be represented with the granularity levels: *table, column header, row* and *field*. The connections in between the granularity enumerations indicate there must be a connection between the corresponding data elements that are labelled with these, as illustrated in Figure 5.9 by the orange arrows.

To begin with, the following paragraph describes how the elements in a single table are interconnected. The model specifies that both column headers and rows are part of exactly one table. A table may have one ore more columns and zero or more rows. For instance, a table with exactly one column and zero rows is empty and collects information on only one attribute. A field has one corresponding column header, but the column header may be connected to zero or more fields. If the column header has no fields, the column is empty. It must not necessarily mean the table is empty. Depending on the type of database, "NULL" or "empty" fields can be added or left away entirely. For instance, when using a relational database each row has a constant number of fields that must contain some value, which can also be NULL. In contrast, when using a graph database, an empty field node can simply be omitted. If, in the case of a relational database, a field element is instantiated, it belongs to exactly one row and a row may have one ore more fields. Zero fields for a row is not possible as it would mean the row is empty and, therefore, the row element wouldn't exist.

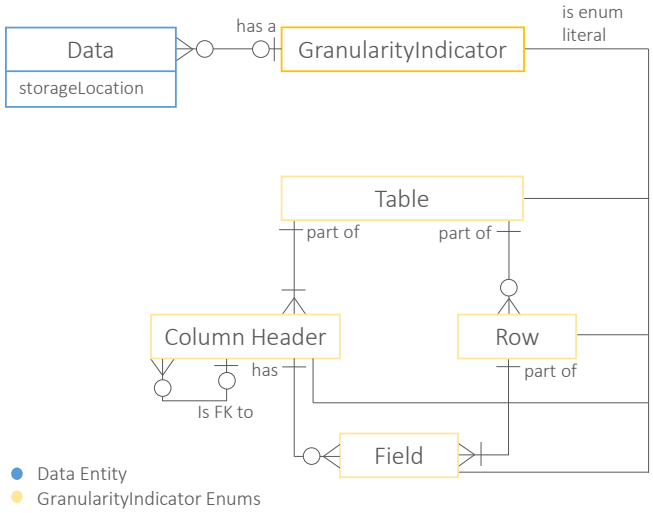

Figure 5.8: Element Connection in a Relational Schema

In order to represent a schema that is the logical grouping and relations in between data, the tables must be interconnectable. Within relational schemata, tables are connected by foreign keys pointing to the corresponding table's primary key. In Figure 5.8 this ability is represented by the column header's link to itself. A column header may be a foreign key pointing to another column header. The other column header is then a primary key or part of a primary key group. Currently, the model does not reflect what kind of key type the column represents. Furthermore, it cannot incorporate the described constraints. The only information reflected is the cardinality on both ends of the self-link. The zero or many cardinality defines that several column headers representing a foreign key may point to the same primary key. The zero to one cardinality reflects that the column header may be either a standard column with no connection or a foreign key pointing to exactly one primary key.

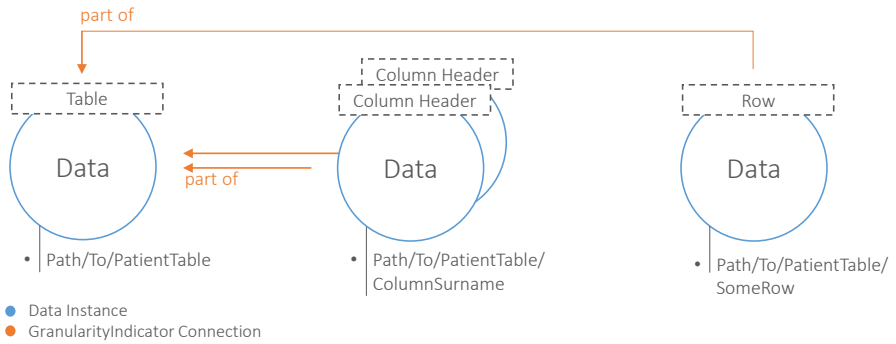

Figure 5.9: Example of Connections Between Diverse Granular Data Levels

Technical and Context Information

So far the connection between the granularityIndicator elements is the only information collected on the schema. A schema may contain more information such as specific constraints, context information on what the data represents, the datatype of a field and so on. In order to store such information, additional metadata elements are required, as described in Section 5.1. Figure 5.10 depicts the elements required for storing the additional information. Two types of metadata instances are introduced called *granularDescription* and *structure*.

In order to understand a data element's context, the granularDescription entity provides a content description through its properties *name* and *description*. It is classified as business metadata because it provides context information. For instance, the context for a columnHeader may be "name: Social Security Number" and "description: Patient ssn." A data element may have zero or more granularDescriptions. For instance, a data element with the row label does not require a granularDescription. The description of the row's content can be provided once on the table level as it is identical for each row. Storing the information with each row would be redundant. Equally, a data element with a field label does not require a description due to the fact that it can be stored with the column header. The table entity has more than one granularDescription. One describes the table and another the overall schema. The element describing the schema can be connected to one or many tables. It is the only granularDescription element connected to more than one data element. It enables easy access to a schema's collection of tables via one central element. Alternatively, the schema information could be added to each table's granularDescription as a property. However, when looking for all tables of a specific schema, each table element of all of the stored schemata in the entire metadata repository with its description would have to be inspected. This is unnecessarily cumbersome and unperformant. The granularDescriptions can also be assigned to different granular levels of other structures, such as JSON. Furthermore, the granularDescription element could be used to deduce the data's semantics and may provide the basis for establishing which data elements are semantically identical or similar.

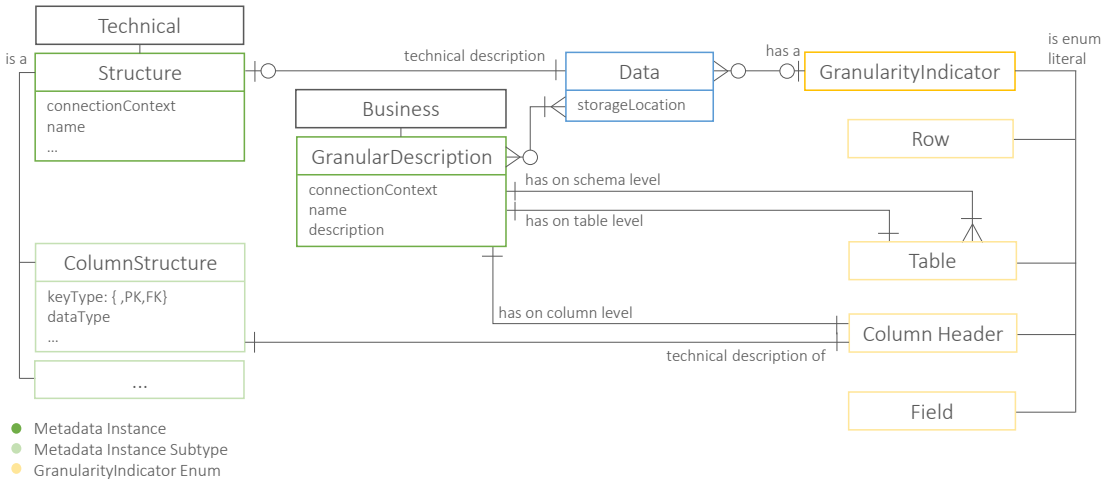

Figure 5.10: Additional Metadata Related to the Schema

The structure entity, which can be seen on the top left in Figure 5.10, contains technical details about the schema and is therefore, classified as technical metadata. It belongs to exactly one data element. A data element can have zero or one structure element. The model depicts one exemplary subtype of the structure element, *columnStructure*, which can be viewed on the bottom left side in Figure 5.10. The parent entity specifies the properties all subtypes have in common. Hence, all the subtypes have the *name* property in addition to their own. The technical information collected can differ strongly depending on the use case. That is why the structure element is modeled to support a variety of properties, signified through the "...". The specified name property refers to the element's actual identifier in the schema. In relational schemata a lot of technical information is collected on the column level. The subtype *columnStructure* has the properties, *keyType, dataType* and more. The keyType property can take three values: *empty, PK* and *FK*. PK is short for primary key and FK for foreign key. If the column is neither of these, the property will be empty. The dataType property can contain any data type such as integer, float and so on. The "..." could further signify its maximum size, its default value, whether it is nullable, specific constraints and so on. The model demonstrates that other subtypes can be added as required through the "..." entity. If desired, subtypes such as a *tableStructure* or *fieldStructure* entity can be added to collect structural information on the according levels.

Figure 5.11 illustrates an exemplary instantiation of Figure 5.10. It depicts a data element representing a column header with both its columnStructure and granularDescription elements. Apart from the technical and context information, also the connection to other data elements is shown through the orange arrow demonstrating a foreign key to primary key connection. In this case the depicted data element must be a primary key, which is also reflected in the columnStructure element, which has the property "KeyType: PK." The example also shows how the name attributes in the granularDescription and columnStructure elements can differ. The column header with the granularDescription attribute "name: Social Security Number" has the corresponding technical identifier of "name: SSN," stored in the columnStructure element.

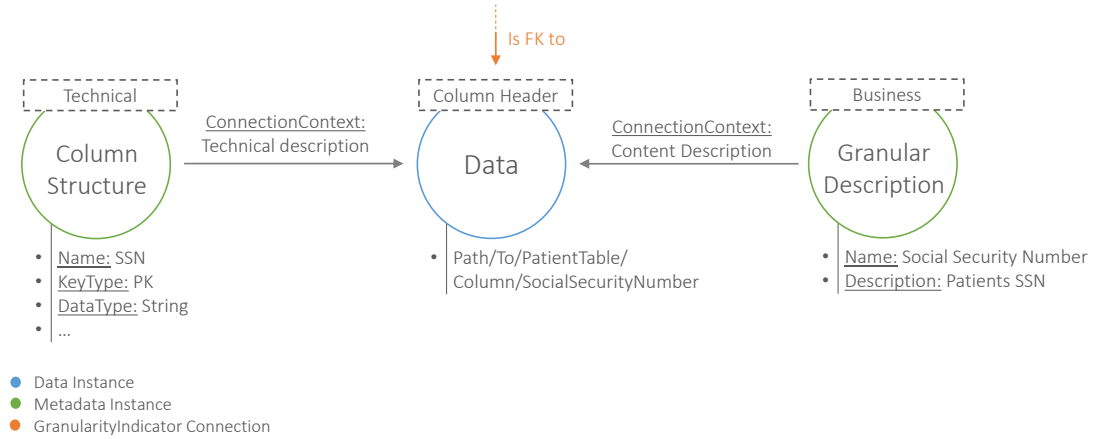

Figure 5.11: Example Instantiation for Additional Metadata on Schema Information

5.2 The Use Case Models

The models demonstrated in the following subsections are based on the core model, which was explained in Section 5.1. All of these are exemplary instantiations of it and cover one of the three use cases described in Chapter 4, namely data inventory, data lineage and data access.

5.2.1 Data Inventory Model

The data inventory is a catalog that covers all the data in the data lake. The only requirement it must fulfil is the ability to produce a searchable list of data elements through which each element can be found and accessed. The information portrayed in the list can be anything ranging from a content description to an enumeration of source names.

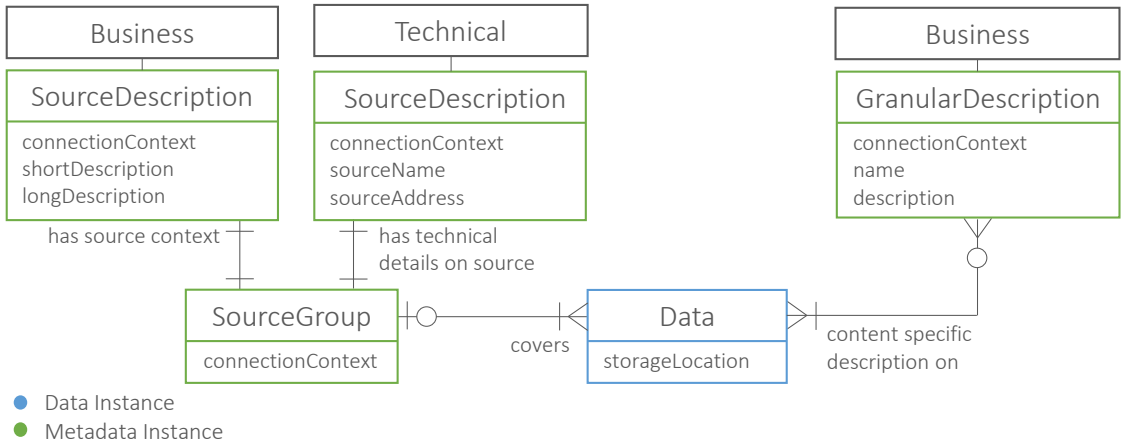

Figure 5.12: Data Inventory Model

Figure 5.12 portrays the suggested model for the inventory. Four metadata instances are used, two of which are a technical and a context description of the source. Next, a *sourceGroup* entity is used to cluster all data elements from the same source together, providing a way of structuring data elements in the inventory. Lastly, the granularDescription entity, introduced in Section 5.1.3 is also used in the inventory. Each sourceGroup element has both a technical and a context description of the source. One or many data elements can be connected to a sourceGroup element. For instance, a set of radiology images may come from the same source and, thus, share the sourceDescription elements. Also, a set of tables in a relational schema or even several schemata may come from the same source. Not every data element must be connected to a sourceGroup element. The idea is that only the high-level data elements are connected. For instance, in a set of relational elements, as described in Section 5.1.3, it suffices to connect the tables to the sourceGroup element, as the row, column and field elements can be reached through the table element. The granularDescription entity provides the ability of distinguishing between the same type of elements from the same source. It contains a content specific description. For example, a data element pointing to one of many

radiology images may have a granularDescription with information about the patient and why he/she needed it. Figure 5.13 illustrates an exemplary instantiation of inventory model demonstrating the radiology image scenario.

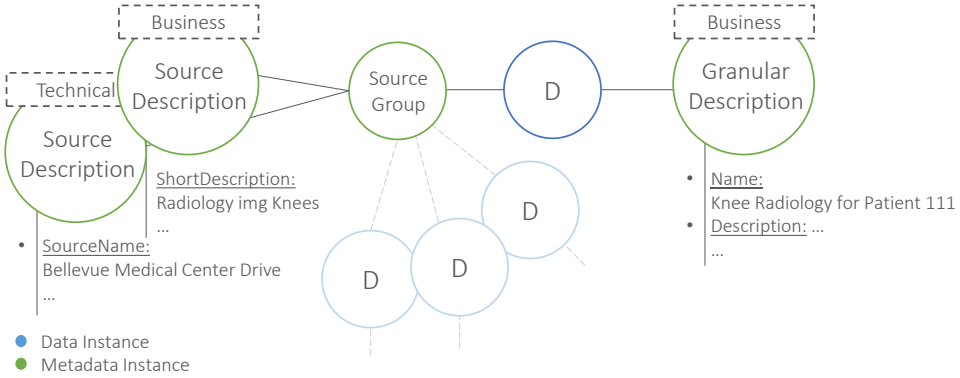

Figure 5.13: Data Inventory Instantiation Example

5.2.2 Data Lineage Model

The lineage model is designed to support the lineage use case described in Section 4.2. As Figure 4.2 shows, the lineage use case in this thesis focuses on the more abstract business perspective, as opposed to the detailed technical perspective. The technical perspective would offer enough information to recreate the data. Technical details such as the transformation types, like blackbox, dispatcher or aggregator transformations, used to create the data are omitted. The information, which elements the data was derived from and in which other elements it is used, suffices for the given use case.

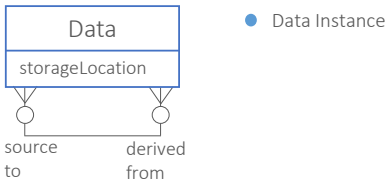

Figure 5.14: Data Lineage Model

Figure 5.14 illustrates the lineage model. A data element can be *derived from* zero or many other data elements. Based on Figure 4.2, the data element may represent a table with information about patient treatments. The data in this table may, for instance, be derived from both a disease and patient table. Thus, it would have two derived-from links to data elements representing the patient and disease table. If desired, the links can be created at the column level, thereby, describing the lineage on a more granular level. For example, Figure 5.15 shows that the treatment tables attribute "dose" is, amongst others, derived from the patient table's attributes "birthdate" and "gender."

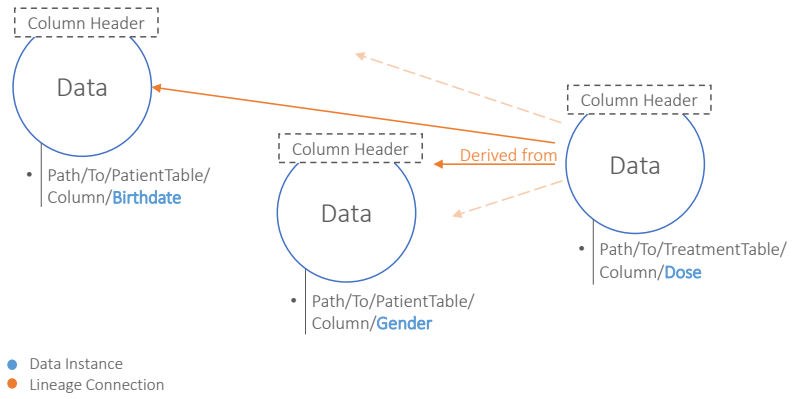

Figure 5.15: Data Lineage Example

5.2.3 Data Access Model

Based on the use case described in Section 4.3, the metadata needs to include a list of people that accessed the data. The term access covers a variety of actions, such as create, read, update or delete actions. Henceforth, the access use case is demonstrated through read actions, as the other operations each raise a new set of questions, or require more concepts, which would exceed the scope of this thesis. For example, the update action requires change management and a deletion requires a new method for accessing the metadata, as the storageLocation attribute no longer points to a corresponding dataset in the lake.

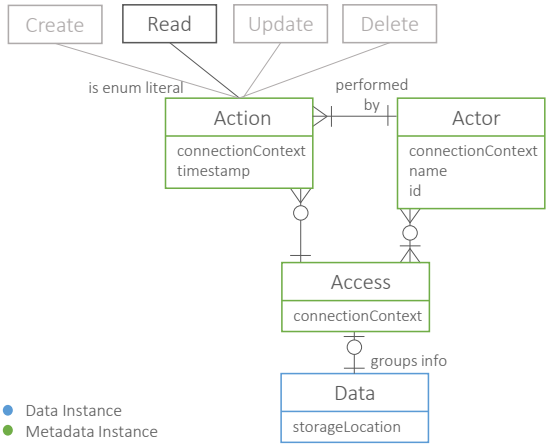

Figure 5.16: Model for Data Access Use Case

Three metadata instances are introduced in Figure 5.16, *action, actor* and *access*. A data entity has zero or exactly one access entity. In order to avoid the data element being swamped by indefinitely increasing access information, all access related nodes are connected to the access entity as an intermediate node. The access entity is a way of adding structure and understandability.

The structural difference with and without an intermediate node is demonstrated in Figure 5.2. Nevertheless, grouping the elements is not strictly necessary. The model suggests that an action element is created for every executed action. It is connected to the involved data's access element and stored with the time it was performed. An action belongs to exactly one access element, but an access element can have zero or many actions. An action is performed by exactly one actor. An actor is connected to one or many actions in an access group. For instance, a specific doctor may repeatedly load information from the patient table. The access group for the patient table will have one actor element with the doctor's name and id. This actor element will be connected to a variety of *read* actions, one for every time he or she loaded the data.

An actor can be connected to one or more access groups. The advantages of the "one or more" cardinality over the "exactly one" cardinality is demonstrated using Figure 5.17. The two images are an example for both of the above mentioned cardinalities. The left side represents the "exactly one" cardinality, so an actor element can only be connected to one access group. The right side illustrates the "one or more" connection as modeled in Figure 5.16. Both images show three data instances, D1, D2 and D3. Each have an access group, a few read action elements and an actor called "Hans," who performed the read actions. In order to highlight the advantages of the "one or more" cardinality, the images will be compared in respect to some ordinary queries. Two possible queries someone might want to answer are, firstly, which data elements were accessed by "Hans" and secondly, which actions "Hans" performed. Executing the first query on the left model requires looking at every single data element and its access group. Each needs to be checked for the presence of the "Hans" actor. The second query is identical, except for the additional loading of the performed actions. In the right model, once a "Hans" actor is found, all accessed data elements are directly connected. Therefore, no additional data elements need to be checked, thus improving performance. This example nicely demonstrates the impact a slightly different cardinality can have.

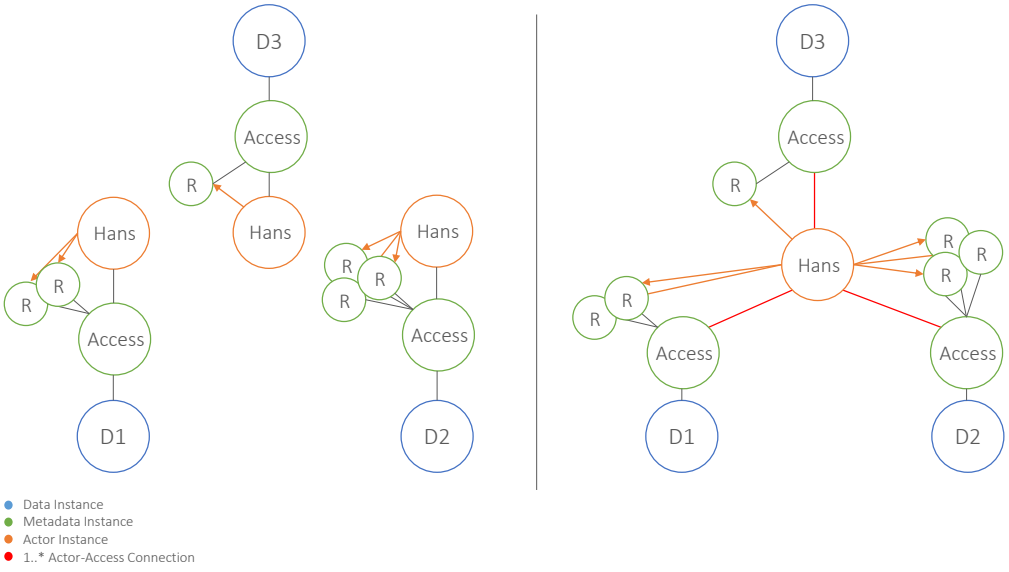

Figure 5.17: Comparison of Cardinalities for the Access to Actor Connection

6 Implementation

While the previous chapter explains the metadata management concept established in the scope of this thesis, this chapter demonstrates how the concept was implemented. To begin with, Section 6.1 covers which type of storage system is used and why. Section 6.2, "The Prototype," provides information on the programming language, database and query language used as well as the implemented functionality. The last section, Section 6.3, demonstrates the implementation of the concept, its extensions and the use cases.

6.1 Choice of Metadata Storage Type

The choice of database type for the metadata repository is based on its scalability and structure related capabilities. The size and growth of the repository depend on what metadata is collected and its granularity. For example, operational metadata, such as modification or lineage data, can be collected on the higher-level record or lower-level attribute level. Fields that require frequent editing will have a lot of modification metadata, whereas other fields, e.g. a patient's birth date, don't change and will have very little modification metadata. The point is the amount of collected metadata can be very different for each data element. Nonetheless, the repository must be able to incorporate large amounts of metadata for each dataset. The metadata repository must measure up to the data lake, which is designed to handle large amounts of data, and must, therefore, be highly scalable. Consequently, a distributed system and database that scales horizontally would best fit the metadata repositories needs. In terms of structure, metadata can be fit into a schema, although it would be preferable to enable the schema-less approach, thus retaining flexibility in adding new metadata. From this follows that a NoSQL database would be better suited for the metadata repository.

At this point the questions remains which subtype of NoSQL database fits the needs of a metadata repository. The choice of the database subtype is mainly based on requirements derived from the use cases, described in Chapter 4. To begin with, the key-value store can be ruled out as even standard use cases like a keyword search, requires the ability of querying the stored content. The created metadata management concept and the use cases are not only focused on storing certain types of information but are strongly centered around the relations among the data and metadata. For example, retaining lineage information can be done solely through storing relationships and the access information is only valuable as long as the relationships between the data element, action and actor are stored. This is why, the document store and column-family store are ruled out as well, as they are not well suited for storing strongly linked data. The only NoSQL database specialized on storing relations is the graph database, which is why this type will be the database of choice for the metadata repository.

6.2 The Prototype

The prototype was implemented with Java, jdk version 12.0.1 and built with the build-management tool Apache Maven, version 3.6.1. It is intended to be used through the jar file mm_prototype-1.0.jar. The database and query language used, as well as the implemented functionality are described in the following subsections Section 6.2.1 and Section 6.2.2 respectively.

6.2.1 Database and Query Language

The prototype is set up with the graph database Neo4j. The choice is based on the DB-Engines ranking of graph database management systems in which Neo4j is by far the most popular graph database [DBE]. For Mai, 2019 it is currently leading with 51,03 points, followed by the Microsoft Azure Cosmos DB with 27,59 points and then the OrientDB with 6,37 points. Apart from it being the leading graph database, it also fulfills the requirements discussed in Section 6.1. The graph query language Cypher is used for interacting with Neo4j.

Listing 6.1 Example Cypher Statement

```
1  //Find accessed node
2  MATCH (accessedNode:Data) WHERE ID(accessedNode)=1234
3  //Create or find access-grouping node
4  MERGE (accessedNode)<-[:GROUPS_ACCESS_INFO]-(access:Metadata:Operational:Access)
5  //Create or find actor and connect him to access group
6  MERGE (actor:Metadata:Operational:Actor {name: 'Hans Mueller', id: '0000'})
7  MERGE (access)<-[:PERFORMED_ACTION_ON]-(actor)
8  //Create action and connect to actor and access group
9  CREATE (action:Read:Metadata:Operational:Action {timestamp: timestamp()})
10 MERGE (access)<-[:PERFORMED_ON]-(action)-[:PERFORMED_BY]->(actor)
```

Cypher uses parenthesis to represent nodes [Neoa]. The content of the square brackets is used to define a specific type of relationship between the nodes. Curly brackets specify properties through a key value notation , e.g “{age: 10}”. These can be added to both nodes and relationships and are therefore embedded in either parentheses, for nodes, or square brackets, for relationships. Apart from properties, labels can be specified for both nodes and relationships, by adding a colon, e.g “:SomeLabel”. These can be used to classify the nodes and relationships as certain types. A relationship between two nodes is expressed through two dashes, optionally with the square brackets in the middle. If the relationship is directed, an angle bracket is used to indicate the direction, for example “- ->”. Lastly, nodes and relationships can be stored in variables. These are specified after the opening bracket, e.g “(varNameNode)” or “[varNameRelationship]”.

Listing 6.1 shows an exemplary cypher statement used for adding access information to the graph, according to the access use case. In line two, the keyword “MATCH” is used to find a node with the id “1234” and the label “Data” [Neoa]. The node is stored in a variable called “accessedNode”. The “MERGE” keyword, as used in line four, either finds an existing node with the specified labels, relationships and properties, or creates one with these if it does not exist [Neob]. In this case, the query is looking for a node with the labels “Metadata, Operational” and “Access”, which is connected to the “accessedNode”. The relationship must have the label “GROUPS_ACCESS_INFO”

and the node will be stored in the variable called “access”. In line six, a node with the listed labels and the properties “name: Hans Mueller” and “id: 0000” is either found, or created if it does not exist, and stored in the variable “actor”. Line seven makes sure a relationship exits between the nodes “access” and “actor” with the label “PERFORMED_ACTION_ON”. Line nine creates a new node with the listed labels and the property “timestamp”. The function “timestamp()” returns the current timestamp [Neo19]. Lastly, the new “action” node is connected to the “access” and “actor” node in line ten.

6.2.2 Implemented Functionality

Depending on the arguments passed through the command-line interface (CLI), the following functionality can be invoked:

1. **Populate Database:** Create example data, utilized in the use cases to demonstrate the metadata management concept.
   CLI option: `-p, --populateDB`
2. **Empty Database:** Delete all nodes and relationships.
   CLI option: `-e, --emptyDB`
3. **Inventory Use Case:** Print inventory with all data nodes, ordered by their original source.
   CLI option: `-i, --printInventory`
4. **Lineage Use Case:** Print lineage information for a specified node as well as its parent and child nodes recursively.
   CLI option: `-pl, --printLineage <NodeId>`
5. **Access Use Case:**
   a) **Add Access Info:** Add access information, namely an action node, actor node and if not existent an access node, to a specific node.
      CLI option: `-a, --addAccess <NodeId> <Action> <ActorFirstName> <ActorSurName> <ActorId>`
   b) **Print Access Info:** Print list of actors together with the dates they accessed the specified node.
      CLI option: `-pa, --printAccess <NodeId>`

The output and result of these actions is demonstrated in Section 6.3.

6.3 Realization of the Metadata Management Concept

This section demonstrates the implementation of the metadata management concept, explained in Chapter 5. Section 6.3.1 shows the realization of the core model and the proposed extensions and Section 6.3.2 focuses on the manifestation all of the use case models and the use case specific output of the prototype’s functionality.

6.3.1 Realization of the Core Model and the Extensions

Figure 6.1: Data Node and Metadata Node Relationship

The core model, illustrated in Figure 5.1, is based on two main entities, the data and metadata entity. Figure 6.1 depicts an instantiation of both these entity types. Both the left and right side show a data node, the yellow circle, and a metadata node, the green circle. Both sides of the image show the same nodes. On the bottom lefthand side, some of the data node's attributes and labels are listed and on the righthand side, most of the metadata node's properties and labels are listed. Based on the information represented on the left side, the node has a label called "Data" and the attribute "storageLocation: Path/to/TablePatient." As defined in the core model, the metadata entity has a connection context property, exposing how it describes the corresponding data entity. In the scope of this implementation, the connection context property is implemented as a label on the relationship connecting the nodes. In this example, the metadata node contains a "table description" on the data node it points to. According to the core model, the metadata entity may have a set of properties. In the image, the two properties name and description are shown. The core model also specifies that each metadata entity has a classification. As can be seen in the image above, the metadata node has a label called "Business." The entities' classification, zoneIndicator and granularityIndicator are all implemented as labels in this prototype, as demonstrated in the following sections.

Realization of the ZoneIndicator Extension

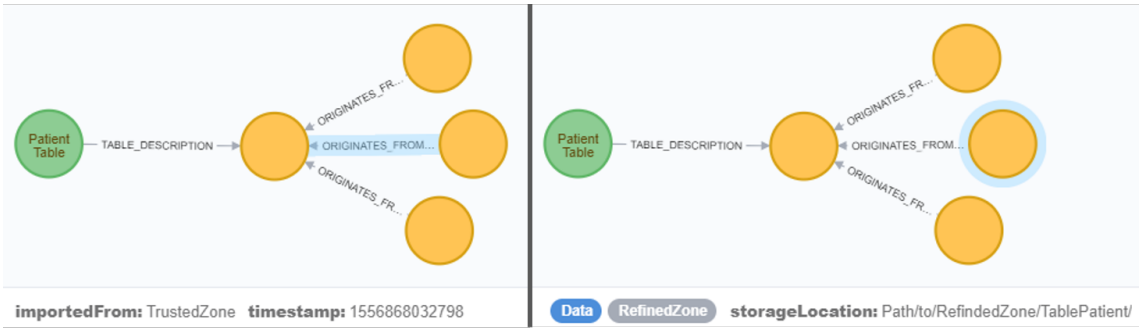

Figure 6.2: Data Node Propagated Through Zones

Figure 6.2 demonstrates how data can be tracked through zones, according to the zoneIndocator extension, illustrated in Figure 5.4. The four yellow nodes are data nodes, each pointing to a different zone. The Link entity, connecting all zone nodes to the raw zone node, was realized as a relationship with the label "originates from rawzone element", pointing to the according raw zone

node. In the image, three data nodes have such a relationship, pointing to the fourth data node, which represents the data in the raw zone. The lefthand side of the image shows the link's attributes, whereas the righthand side shows the connected node's labels and attributes. According to the links "importedFrom" attribute and the node's label "RefinedZone," the patient table was moved from the trusted zone into the refined zone at the given time, stored in the attribute "timestamp".

Realization of the GranularityIndicator Extension

As mentioned in Section 6.3.1, the granularityIndicator entity is implemented as a label on the data nodes. Figure 6.3 shows four data nodes, each with one of the proposed indicators for a relational structure for different granular levels, as illustrated in Figure 5.6. The indicators belong to the nodes with the corresponding color.

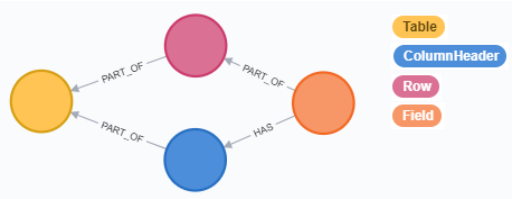

Figure 6.3: Data Nodes with Different Granularity Indicators

Realization of the Schema Extension

As described in Section 5.1.3, retaining schema information includes storing how elements on different granular levels are related as well as storing the according technical and context information on each level. Besides the used granularityIndicators, Figure 6.3 also shows how the elements on different granular levels are related to each other, according to the schema extension's model in Figure 5.8. The four nodes are an extract from a set of nodes, representing a relational table. The orange field node is "part of" a row, represented by the red node. The row in turn is "part of" a table, represented by the yellow node. A table has one or more columns. In this case one column header is listed in the form of the blue node. Lastly, a field belongs to a certain column, therefore, the orange field node "has" a column header node.

Figure 6.4 depicts an extract of the schema information stored in the database. The four yellow nodes are data nodes, each representing a table. The blue nodes are also data nodes and represent column headers. As illustrated in Figure 5.10, a table element has two granularDescription elements, one describing the content of the schema, the other the content of the table. In addition, each column header has a granularDescription, describing the content of the column. In Figure 6.4, each of the light green nodes is a granularDesciption element with the name attribute rendered in the circle. The image nicely shows that the granularDescription on the overall schema is shared by all table nodes, whereas all other granularDescriptions belong to exactly one node. The dark green nodes are instantiations of the structure entity and contain a technical description.

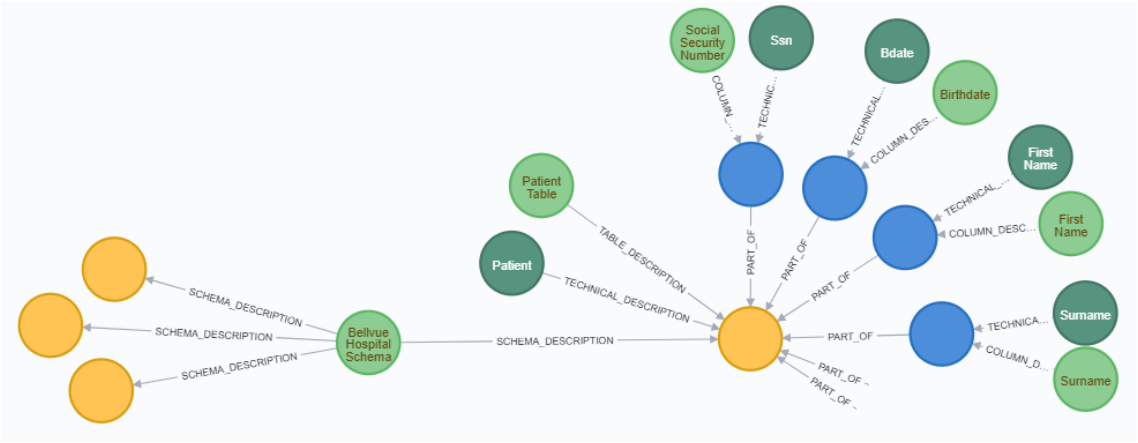

Figure 6.4: Technical and Context Metadata Nodes for each Data Node

Figure 6.5 illustrates the "FK_TO" relationship through which the tables are interconnected. Each column header representing a foreign key has such a relationship with the corresponding node, representing the primary key column. In this case, the treatment table has a column called patient which is a foreign key, pointing to the patient table's ssn column.

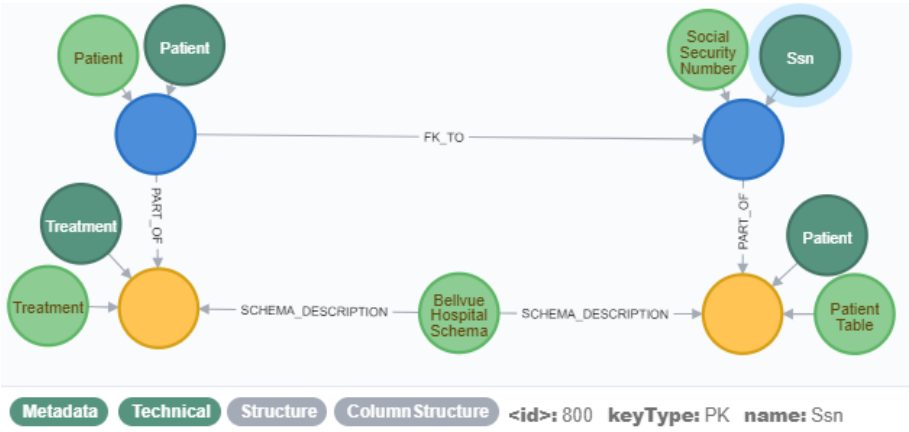

Figure 6.5: Table Interconnection Example

6.3.2 Use Case Realization

Now that the realization of the core model and its extensions has been demonstrated, that only leaves a demonstration of the use case models. In the following sections, use case specific data is presented through the database's browser visualization as well as the printed output of the prototype.

Inventory Use Case Realization

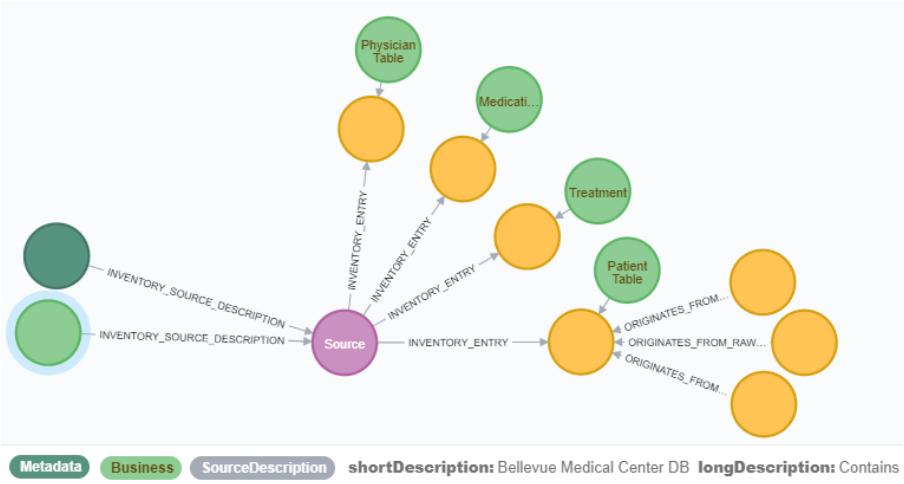

Figure 6.6: Inventory List - Neo4j Visualization

The inventory supplies an overview of the data stored in the data lake. As defined in Section 5.2.1 and visualized in Figure 6.6, the yellow data elements are linked to a purple node called source. The source node enables structuring the elements in the list, according to their source context. As explained in Section 5.2.1, only the elements of the highest granularity are connected, so in this case, only the table elements are listed. The other data elements, row, column and field nodes, can be reached through the table nodes. The inventory also includes a list of zones each element resides in. As specified in Figure 5.12, the purple source node has both a business and technical sourceDescription node, which are light green and dark green respectively.

```
> java -jar mm_prototype-1.0.jar --printInventory

_____PRINT INVENTORY:_____________
_____Sources:_____________________

 "Bellvue Medical Center DB"

     ___
     Name: "Treatment Table"
     Description: "List of treatments prescribed in the Bellvue hospital"
     Zones: RawZone

     ___
     Name: "Medication Table"
     Description: "List of all the medications prescribed in the Bellvue hospital"
     Zones: RawZone

     ___
     Name: "Patient Table"
     Description: "Detailed demographic and other personal information on hospital patients"
     Zones: RawZone RefinedZone TrustedZone Sandbox

     ___
     Name: "Physician Table"
     Description: "Detailed demographic and other personal information on hospital physicians"
     Zones: RawZone
```

Figure 6.7: Inventory List - Prototype Output

In order to retrieve the inventory information, all the source nodes are collected together with their technical and business description. For each source node, a list of data elements is compiled together with their granularDescription. Furthermore, the existence of corresponding nodes in other zones is checked for each of these data elements. The result comprises groups of nodes, resembling those in Figure 6.6. The inventory printed through the prototype produces a list as depicted in Figure 6.7. Each data element's name, description and zone information is printed. The elements are listed with the name of their source.

Lineage Use Case Realization

Figure 6.8: Lineage Information

The elements required for storing lineage information merely comprise a single link entity between data entities, which are derived from each other, as can be seen in Figure 5.14. This link is realized through a relationship between data nodes with the label "derived from". The lefthand side in Figure 6.8 depicts an instantiation of the lineage example shown in Figure 5.15. The treatment table's column dose is derived from the patient table's columns gender and birth date as well as the medication table's column code.

The lineage information is collected by first finding the node with the specified id, together with its granularDescription. In the next step, all nodes connected to it with a "derived from" relationship are gathered. These either belong to the list of nodes to which the given node is the source, or to those from which the given node is derived. As the specified node might not have lineage information, but its child or parent nodes might, this information is also collected for each of the child and parent nodes as well as the child node's children and parent node's parents and so on. This information can be printed through the prototype, as shown on the righthand side of Figure 6.8. If they exist, it prints both a list of nodes to which the node is the source as well as a list of nodes the specified node itself is derived from. In the example, the "Treatment Table" node does not have any such information. The output recursively includes all the parent and child nodes with their ids and lineage information. Therefore, the output also shows that the child node "dose" is derived from the three nodes gender, birth date and code. The lineage can be traced further by requesting the same information for one of the printed nodes through its id.

Access Use Case Realization

Figure 6.9 illustrates an extract of the created access data. As suggested in Section 5.2.3, the access information on each data element is grouped together by the orange nodes, which are an instance of the access entity, introduced in Figure 5.16. The rose colored nodes are actions and the pink node represents an actor. The nodes show that Hans Müller performed a read action on the patient table, at the time represented through the highlighted action node's timestamp attribute. The example also demonstrates that access information can be collected on varying granular levels. In the example, Hans Müller performed a read action on the entire patient table as well as a read action on the treatment table's medication column.

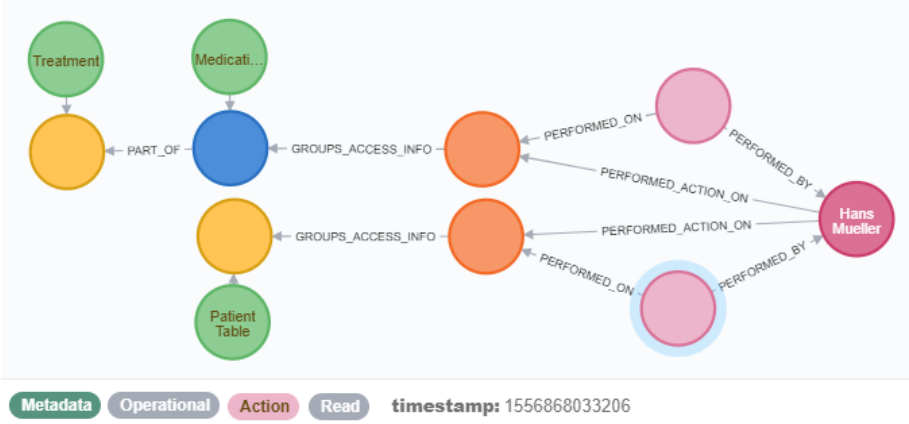

Figure 6.9: Access Information - Neo4j Visualization

Access information is added as described in Section 6.2.1. Querying access information for a specific node involves finding the node in question, together with its orange access node. From here on, the connected actor nodes can be returned as well as the connected action nodes. In order to find all the data elements an actor, e.g. Hans Müller, has accessed, the actor node with the according name property needs to be matched. Then the connected access nodes can be found and through them the data element with its granuarDesciption.

```
>java -jar target\mm_prototype-1.0.jar -pa 847

_______PRINT ACCESS INFO_______
       "Robert Schneider" : 2019-05-27 13:31:56
       "Max Mustermann" : 2019-05-27 13:31:29 | 2019-05-27 13:31:16
```

Figure 6.10: Access Information - Prototype Output

Figure 6.10 demonstrates the prototype's output for a specific node. All the actors are listed together with the dates on which they accessed the according data element.

7 Discussion and Conclusion

This section ascertains whether the thesis' main objective has been successfully achieved. As described in Section 1.1, the main objective is reached by fulfilling a set of sub goals. To begin with, the accomplishment of the first three goals is examined. These involve the creation of firstly, a set of use cases, secondly, the creation of a metadata management concept and thirdly, a prototype. The fourth goal is a critical evaluation of the results of the previous three goals. The evaluation is provided in this section, thus, fulfilling goal four. Section 7.1, Section 7.2 and Section 7.3 each discuss if the corresponding goals were fulfilled. The discussion also includes if there were problems during the realization and if the solutions are based on simplifications. Furthermore, the discussion also contains whether the solutions have limitations and lastly, which problems might arise during future use. After the discussion on the fulfillment of the goals one to three, the achievement of the thesis' main objective is evaluated in Section 7.4.

7.1 Goal 1: Use Cases

The first goal involved creating a set of use cases, which highlight the necessity for metadata management in the data lake. They served as a guideline for the design of the metadata management concept. Furthermore, they show whether or not the concept is suitable for providing the desired content for each use case.

The created set comprises three classic data lake use cases, the inventory use case, the lineage use case and access use case, each described in Chapter 4. All three fulfill the requirement of demonstrating the necessity for metadata management. With the collection and storage of metadata, metadata management is a compulsory task. The inventory use case demands the collection of metadata in the form of descriptions on data sources and data elements. The metadata collected for the other two use cases, lineage and access, involve tracking which elements are derived from each other and by whom and when data was accessed.

The lineage use case was simplified to contain only the business perspective and not the technical perspective on the lineage information. Similarly, the access use case focused on people accessing the data and didn't consider access through applications. Even though simplifications imply certain aspects aren't considered, the simplified use cases focus on the concept's main aspects and are therefore well suited for demonstrating the realizability and applicability of the concept. Generally speaking, the applied simplifications do not impact the general applicability of the created concept and omitted aspects can be added at a later time.

7.2 Goal 2: Concept

The second goal specified that a metadata management concept should be created. The requirements for the concept include that it should enable storing diversely structured metadata of any type and quantity. It should also support the characteristics of the architecture levels, described in Section 2.6. Lastly, it should cover a variety of both classic data lake use cases and organization specific use cases.

The established concept consists of a conceptual metadata model with an abstract core model and more specific model extensions, both described in Section 5.1. In addition, it contains specialized models for the specified use cases from goal one, as can be seen in Section 5.2. The core model covers how metadata is mapped to the according data elements in the data lake, how the metadata itself is structured and that the metadata is always classified as one of the three types, business, technical, or operational. The core model also specifies where the data is located within the data lake and how granular the specific data is. The models extending the core concept demonstrate in detail how to keep track of data throughout the data lakes internal organization, in which granularity metadata can be collected and how a data schema can be stored in the form of metadata. The use case specific models are based on the core model and incorporate all the mentioned aspects of it.

The question remains whether the established conceptual metadata model meets the requirements specified in goal two. To begin with, the core model's metadata entity can possess any number of properties, as specified in Section 5.1. Each metadata entity may have its own set of properties. Therefore, the core model supports storing diversely structured metadata. The requirement for storing metadata of different types is also fulfilled, as the core model supports storing metadata classified as business, technical as well as operational metadata. The model is also not limited in regard to the quantity of metadata, as each data element can have an unlimited number of metadata elements. Therefore, the conceptual model supports the specified structural, type and quantity requirements.

The model also supports storing metadata generated on all the data lake's architecture levels. The metadata generated on the different levels and throughout the same levels may differ in context and structure. As the core model can handle diversely structured metadata of different types, the metadata's diverseness throughout the levels does not pose a problem. Furthermore, the core model's extensions as well as the use case models demonstrate the use of metadata throughout most of the levels, as described in Section 5.1 and Section 5.2. The inventory and access use cases thematically belong to the data ingestion and extraction process, which are the central topic of level one, called data process. The usage of metadata belonging to level three, data organization, is explicitly demonstrated through the zoneIndicator extension, which shows how data is tracked through a data lake's zones, in Section 5.1.1. The schema extension together with the granularityIndicator extension, described in Section 5.1.3 and Section 5.1.2, demonstrate how a schema is stored, which is metadata generated on level four, the data model level. Lastly, some information belonging to the fifth level, data storage, is represented through the core model's data entity's attribute storagePath. The only level not explicitly demonstrated through an extension or use case is the second level, data services, which' metadata is supported nonetheless, as stated above.

The third requirement for the concept is that it must support a large variety of use cases. In order to provide this feature, the core model must be flexible enough that it can incorporate any more specific use case design. The core model provides the necessary flexibility through its high-level

definitions on the data and metadata elements and their interconnections. The data entity and metadata entity can be used to represent any more specific data and metadata. For example, the data entity can represent data of any structure, i.e a table, but also a video, as well as data of any granularity, i.e. the overall table, but also a single field. As mentioned in this section, the metadata can be structured diversely and can therefore, represent a variety of more specific entities. For example, the access use case demonstrates this through its three differently structured entities, actor, action and access. In addition, Section 5.1 defines very flexible options for connecting the data and metadata entities. The data entity can have any number of metadata elements and the metadata entity can likewise be connected to several data entities. The metadata entity can also be connected to other metadata elements. Through its abstract and flexibility definitions, the core model supports modeling a variety of use cases. In Section 5.2, the inventory, lineage and access use case models demonstrate how a more specific model is created, according to the core models specifications.

In terms of limitations, several topic areas had to be omitted due to a lack of time. For example, handling semantically identical data through metadata, or storing schemata for structured, but not relational data, could not be covered. Furthermore, several simplifications were introduced to reduce complexity. For example, with the access use case, the concept considers read actions, as other actions, such as updates or deletes, raise an extended set of new questions outside the scope of this thesis, as explained in Section 4.3.

With the main focus of this thesis being the provision of a metadata management concept, other topics, such as the automated creation of models weren't examined. The automated creation of use case specific models is a relevant topic, as the amount of metadata management use cases is increasing. In terms of automatisation, the use case models as well as the schema model are suboptimal because they are manually designed. Nonetheless, the mentioned models are merely a suggestion on how the use case specific data can be stored according to the core model. The core model also supports connecting all the collected metadata directly to the according data entity, which would then no longer require a use case specific, handmade design. Therefore, the inclusion of new metadata management use cases can be carried out in an automated fashion, if desired. However, the omission of use case specific models also entails disadvantages. The handmade models are designed based on potential queries and therefore, consider aspects such as performance, which the automated variant does not support. Ultimately, the core model supports both variants.

Nevertheless, as discussed previously, the core metadata model is abstract and highly flexible and therefore supports implementing the mentioned expanded versions of the use cases as well as further topic areas like the automatisation aspect.

7.3 Goal 3: Prototype

In order to demonstrate the created concepts realizability, goal three asks for a prototype. The prototype should implement the concept and provide a method for testing the concept's aptness for answering the use cases.

As shown in Section 6.3, exemplary metadata sets were stored according to the specifications of all of the created models, ranging from the abstract core model, its extensions to the use case models. All of the modeled aspects could be implemented, thus proving that the models are realizable. Furthermore, the questions derived from the use cases were answered successfully, demonstrating that the created concept also fulfills its purpose for metadata management in real life scenarios.

As discussed in Section 6.1, the prototype is based on a graphdatabase. Section 2.7 explains that these aren't particularly well suited for horizontal scaling. Given that the amount of metadata may expand drastically, this might become an issue at some point in time. Section 2.7 also explains that the issue can be addressed through content based sharding of data. The established concept has entities which can be used for splitting the data. For example, the data could be sharded according to the core models classification types of business, technical and operational metadata. Therefore, the established concept also supports scalability requirements.

7.4 Main Objective

As described in Section 1.1, the main objective of this thesis involves the provision of a design, which enables implementing metadata management based on the data lake architecture, described in Section 2.6. The established design is supposed to support a variety of both classic data lake use cases and organization specific use cases.

As mentioned in Section 1.1, the thesis' main objective can be reached by fulfilling the specified goals, one through four. The previous sections demonstrate that each of the defined goals were achieved successfully and from this follows that the thesis' main objective has also been accomplished.

In conclusion, this thesis' has successfully provided a concept for metadata management in the data lake context, which is based on Giebler et al.'s data lake architecture, supports diverse use cases and is therefore universally applicable. The initial motivation for creating the concept was the enabling of data governance in order to exploit the data lake's strengths as well as preventing it from turning into a data swamp. With its characteristics, the established metadata management concept serves as a basis for data governance tasks and thus satisfies the original incentive for conducting this thesis.

8 Summary and Outlook

In order to fully exploit the strengths of the data lake concept and to prevent it from turning into a data swamp, pro-active data governance and metadata management are required. The central goal of this thesis was the development of a concept, which enables performing metadata management on a data lake. It distinguishes itself from existing research and solutions by considering the underlying data lake architecture as well as supporting a wide variety of metadata management use cases. The illustrated goal was achieved by firstly setting up exemplary data lake use cases. Secondly, the concept was developed based on these use cases and lastly, the concept was tested by implementing a prototype.

To begin with, the established set of use cases are classic data lake use cases and constitute a data inventory, data lineage and data access use case. Each of these use cases accentuate the necessity for metadata management. The metadata management concept which was developed next, is based on an architecture structuring all data lake processes, tasks and components into the six levels: data process, data access, data organization, data model, data storage and metadata. The metadata level is cross sectional and deals with metadata generated throughout the other five levels and is therefore the generated concepts main point of reference. The centerpiece of the concept is the conceptual metadata model, which consists of a core model and some model extensions. The abstract core model defines how metadata is structured, classified and mapped to the data in the data lake. It also specifies where the data is located in the data lake and how granular it is. The additional models extending the core model demonstrate in detail how to track data through data lake zones, specify granularity levels and how to store schemata. In order to demonstrate the established metadata model, a specific model for each use case was created based on the core models specifications. Lastly, a prototype was developed using the graphdatabase Neo4j as a metadata repository. Each of the models, including the core model, its extensions and use case models were demonstrated by storing exemplary metadata according to each models specification. The prototype provides functionality for populating and emptying the metadata repository with the example data. It also offers use case specific functions for printing the data inventory and lineage or access information on a specific data element. Lastly, it enables adding access information to the metadata repository for a specific data element.

The analysis of the generated concept showed that it can store diversely structured metadata of any type and quantity. Therefore, it supports storing metadata created throughout each data lake architecture level. It is also flexible enough to support diverse use cases and is therefore universally applicable in the data lake context. In conclusion, the concept meets the requirements and is therefore suitable for the initial motivation of metadata management and data governance.

The next steps in developing the concept include removing the adopted simplifications and limitations. In terms of removing simplifications, this involves expanding the concept to support all possible actions within the access use case. That means topics such as change management and data deletions must be dealt with. Furthermore, the concept can be enhanced to support applications accessing

data as well as people. Many more topics can be considered and incorporated in the concept, such as dealing with semantically identical data. Currently the concept explicitly demonstrates storing schemata for relational data. Therefore, it can be expanded to store otherly structured data as well. Other classic use cases such as a keyword search or storing the technical perspective on data lineage can be incorporated. So far the mentioned topics are centered around extending the concepts metadata model. There are other topics which can be discussed in more detail such as the process of incorporating new metadata and generally hooking up new metadata sources. The concept can also be combined with a solution for automated extraction of metadata. Furthermore, the prototype can be improved and extended. For example, a graphical user interface could be implemented that displays the results and supports active user interaction. In conclusion, future work can be applied to different aspects of the concept, ranging from the metadata model to processes and lastly the prototype.

Bibliography

[Ala] Alation. *The Data Catalog Company*. URL: https://alation.com/product/ (cit. on p. 24).

[Cat10] R. Cattell. "Scalable SQL and NoSQL Data Stores". In: *SIGMOD Record* (2010) (cit. on pp. 29, 30).

[CSN+] M. Chessell, F. Scheepers, N. Nguyen, R. van Kessel, R. van der Starre. *Governing and Managing Big Data for Analytics and Decision Makers* (cit. on p. 26).

[DBE] DB-Engines. *DB-Engines Ranking von Graph DBMS*. URL: https://db-engines.com/de/ranking/graph+dbms (cit. on p. 54).

[Eur18] European Comission. *2018 reform of EU data protection rules*. 2018. URL: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en#background (cit. on p. 20).

[FLR94] C. Fox, A. Levitin, T. Redman. "The notion of data and its quality dimensions". In: *Information Processing & Management* 30.1 (1994), pp. 9–19. ISSN: 0306-4573. DOI: 10.1016/0306-4573(94)90020-5. URL: http://www.sciencedirect.com/science/article/pii/0306457394900205 (cit. on p. 21).

[GGH19] C. Giebler, C. Gröger, E. Hoos. "The Unified Data Architecture for Data Lakes: Unplublished Technical Report". University of Stuttgart, 2019 (cit. on pp. 18, 25, 26, 28, 66).

[HGHM11] K. Hildebrand, M. Gebauer, H. Hinrichs, M. Mielke, eds. *Daten- und Informationsqualität*. Wiesbaden: Vieweg+Teubner, 2011. ISBN: 978-3-8348-1453-1. DOI: 10.1007/978-3-8348-9953-8 (cit. on pp. 17, 20, 21).

[HGQ16] R. Hai, S. Geisler, C. Quix. "Constance: An Intelligent Data Lake System". In: *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. Ed. by F. Özcan, G. Koutrika, S. Madden. New York, New York, USA: ACM Press, 2016, pp. 2097–2100. ISBN: 9781450335317. DOI: 10.1145/2882903.2899389 (cit. on p. 34).

[HKN+16] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, S. E. Whang. "Goods: Organizing Google's Datasets". In: *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. Ed. by F. Özcan, G. Koutrika, S. Madden. New York, New York, USA: ACM Press, 2016, pp. 795–806. ISBN: 9781450335317. DOI: 10.1145/2882903.2903730 (cit. on p. 33).

[Hua15] F. Huang. *Managing Data Lakes in Big Data Era: What's a data lake and why has it became popular in data management ecosystem*. Piscataway, NJ: IEEE, 2015. ISBN: 9781479987283. URL: http://ieeexplore.ieee.org/servlet/opac?punumber=7274858 (cit. on pp. 17, 19, 20).

[IBM] IBM Knowledge Center. *Types of metadata*. URL: https://www.ibm.com/support/knowledgecenter/es/SSZJPZ_9.1.0/com.ibm.swg.im.iis.mdwb.doc/topics/c_metadata_types.html (cit. on p. 23).

[IL15] W. H. Inmon, D. Linstedt. *Data architecture: a primer for the data scientist: Big Data, data warehouse and Data Vault*. Waltham, MA et al.: Morgan Kaufmann imprint of Elsevier and Elsevier, 2015. ISBN: 9780128020449 (cit. on p. 28).

[Inf] Informatica. *What is metadata?* URL: https://www.informatica.com/services-and-training/glossary-of-terms/metadata-definition.html#fbid=fitHgv8HWYN (cit. on p. 23).

[Inm16] W. H. Inmon. *Data lake architecture: Designing the data lake and avoiding the garbage dump*. First edition, first printing. [Bradley Beach]: Technics Publications, 2016. ISBN: 1634621174 (cit. on pp. 26–28).

[KB10] V. Khatri, C. V. Brown. "Designing data governance". In: *Communications of the ACM* 53.1 (2010), p. 148. ISSN: 00010782. DOI: 10.1145/1629175.1629210 (cit. on pp. 20, 21).

[KBM10] H.-G. Kemper, H. Baars, W. Mehanna. *Business Intelligence - Grundlagen und praktische Anwendungen: Eine Einführung in die IT-basierte Managementunterstützung*. 3., überarbeitete und erweiterte Auflage. Studium. Wiesbaden: Vieweg+Teubner Verlag / GWV Fachverlage GmbH Wiesbaden, 2010. ISBN: 9783834807199. DOI: 10.1007/978-3-8348-9727-5. URL: http://dx.doi.org/10.1007/978-3-8348-9727-5 (cit. on p. 29).

[Klö] K. Klöckner. *Im Vergleich: NoSQL vs. relationale Datenbanken* (cit. on p. 30).

[KR] K. Kaur, R. Rani. "Modeling and Querying Data in NoSQL Databases". In: *2013 IEEE International Conference on Big Data* (cit. on pp. 30, 31).

[Kue] M. Kuesters. *Interactions between Data Security and Data Quality*. URL: https://de.slideshare.net/MichaelKuesters/data-qualitysecurity (cit. on p. 21).

[Lay13] J. Layton. *The Metadata Storage Problem*. 2013. URL: https://www.enterprisestorageforum.com/storage-management/metadata-storage-problem.html (cit. on p. 29).

[LS16] A. LaPlante, B. Sharma. *Architecting Data Lakes: Data Management Architectures for Advanced Business Use Cases*. Sebastopol: O'Reilly, 2016. ISBN: 9781492042518 (cit. on p. 26).

[Mat17] C. Mathis. "Data Lakes". In: *Datenbank-Spektrum* 17.3 (2017), pp. 289–293. ISSN: 1618-2162. DOI: 10.1007/s13222-017-0272-7 (cit. on pp. 17, 19–21).

[MDSB] Markus Spiekermann, Daniel Tebernum, Sven Wenzel, Boris Otto. "A Metadata Model for Data Goods". In: *Multikonferenz Wirtschaftsinformatik 2018* (cit. on p. 33).

[MT16] N. Miloslavskaya, A. Tolstoy. "Big Data, Fast Data and Data Lake Concepts". In: *Procedia Computer Science* 88 (2016), pp. 300–305. ISSN: 18770509. DOI: 10.1016/j.procs.2016.07.439 (cit. on pp. 17, 19, 20).

[Neoa] Neo4j. *Cypher Basics I*. URL: https://neo4j.com/developer/cypher-query-language/ (cit. on p. 54).

[Neob] Neo4j. *Cypher Basics II*. URL: https://neo4j.com/developer/cypher-basics-ii/ (cit. on p. 54).

[Neo19] Neo4j. *4.2. Scalar functions - Chapter 4. Functions*. 24.05.2019. URL: https://neo4j.com/docs/cypher-manual/current/functions/scalar/ (cit. on p. 55).

[NRD18] I. D. Nogueira, M. Romdhane, J. Darmont. "Modeling Data Lake Metadata with a Data Vault". In: *Proceedings of the 22nd International Database Engineering & Applications Symposium on - IDEAS 2018*. New York: ACM Press, 2018, pp. 253–261. ISBN: 9781450365277. DOI: 10.1145/3216122.3216130 (cit. on p. 28).

[QHV16] C. Quix, R. Hai, I. Vatov. "Metadata Extraction and Management in Data LakesWith GEMMS". In: *Complex Systems Informatics and Modeling Quarterly* 9 (2016), pp. 67–83. DOI: 10.7250/csimq.2016-9.04 (cit. on p. 34).

[QSC16] QSC AG. *Die 9 V von Big Data – von Validity bis Volume*. 2016. URL: https://digitales-wirtschaftswunder.de/die-9-v-von-big-data/ (cit. on p. 19).

[Ram] J. Ramos. *The Data Reservoir: Architecture, Best Practices and Governance*. URL: http://www.northtexasdama.org/wp-content/uploads/2015/12/Data_Reservoir_Governance_Best_Practices_Final.pdf (cit. on p. 26).

[Ril17] J. Riley. *Understanding metadata: What is metadata, and what is it for*. NISO Primer series. Baltimore, MD: National Information Standards Organization, 2017. ISBN: 978-1-937522-72-8. URL: http://marciazeng.slis.kent.edu/metadatabasics/types.htm (cit. on pp. 17, 22).

[Riv15] P. Rivett. *Are you creating a data swamp*. 2015 (cit. on p. 20).

[Rou] M. Rouse. *Was ist Self Service im Internet?* URL: https://www.searchenterprisesoftware.de/definition/Self-Service-im-Internet (cit. on p. 26).

[Rus17] P. Russom. *Data Lakes: Purposes, Practices, Patterns, and Platforms*. 2017 (cit. on pp. 23, 26).

[SDE18a] G. de Simoni, A. Dayley, R. Edjlali. *4 Use Cases That Drive Critical Capabilities in Metadata Management*. 2018 (cit. on pp. 23, 24, 33).

[SDE18b] G. de Simoni, A. Dayley, R. Edjlali. *Magic Quadrant for Metadata Management Solutions*. 2018 (cit. on pp. 17, 22–25).

[SF13] P. J. Sadalage, M. Fowler. *NoSQL distilled: A brief guide to the emerging world of polyglot persistence*. Upper Saddle River NJ: Addison-Wesley, 2013. ISBN: 0321826620 (cit. on pp. 30, 31).

[Sim12] G. de Simoni. *Five Ways to Use Metadata Management to Deliver Business Value for Data*. 2012 (cit. on pp. 23–25).

[SM] B. Stein, A. Morrison. "The enterprise data lake- Better integration and deeper analytics: Data lakes that can scale at the pace of the cloud remove integration barriers and clear a path for more timely and informed business decisions." In: *PWC Technology Forecast*. Vol. 1. URL: www.pwc.com/technologyforecast (cit. on p. 20).

[Spi] Spirion. *Data Lifecycle Management (DLM): What is Data Lifecycle Management?* URL: https://www.spirion.com/data-lifecycle-management/ (cit. on p. 21).

[TSRC] I. Terrizzano, P. Schwarz, M. Roth, J. E. Colino. "Data Wrangling: The Challenging Journey from the Wild to the Lake". In: *Presented in the 7th Biennial Conference on Innovative Data Systems Research* (cit. on p. 22).

[V14] M. V. “Comparative Study of NoSQL Document, Column Store Databases and Evaluation of Cassandra”. In: *International Journal of Database Management Systems* 6.4 (2014), pp. 11–26. ISSN: 09755985. DOI: 10.5121/ijdms.2014.6402 (cit. on pp. 30, 31).

[WA15] C. Walker, H. Alrehamy. “Personal Data Lake with Data Gravity Pull”. In: *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*. IEEE, 26.08.2015 - 28.08.2015, pp. 160–167. ISBN: 978-1-4673-7183-4. DOI: 10.1109/BDCloud.2015.62 (cit. on p. 34).

[Zal] Zaloni. *The Data Lake Reference Architecture: Leveraging a Data Reference Architecture to Ensure Data Lake Success* (cit. on p. 27).

All links were last followed on Mai 23, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature