THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 1 Capitole (UT1 Capitole)*

Présentée et soutenue le *7 Décembre 2021* **par :**

Yan ZHAO

Metadata Management for Data Lake Governance

JURY

Anne LAURENT	Professeure, Université de Montpellier	Rapportrice
Claudia RONCANCIO	Professeure, Institut Polytechnique de Grenoble	Rapportrice
Jérôme DARMONT	Professeur, Université Lumière Lyon 2	Examinateur
Chantal SOULÉ-DUPUY	Professeure, Université Toulouse I Capitole	Examinatrice
Grégory LOPEZ	Chef du service Gestion de la Donnée, CHU de Toulouse	Invité
Imen MEGDICHE	Maître de Conférences, Institut National Universitaire Jean-François-Champollion	Co-encadrante
Franck RAVAT	Professeur, Université Toulouse I Capitole	Directeur de thèse

École doctorale et spécialité :
EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse : Informatique et Télécommunications
Unité de Recherche :
IRIT: Institut de Recherche en Informatique de Toulouse (UMR 5505)
Directeur de Thèse :
Franck RAVAT
Rapporteures :
Anne LAURENT et *Claudia RONCANCIO*

Acknowledgement

The three-year research journey would not have been possible without the efforts of all those who have directly or indirectly helped me. I want to acknowledge a select few, although many others remain unacknowledged but are greatly appreciated.

Foremost, I would like to express my deep and sincere gratitude to my supervisor **Professor Franck RAVAT** for the quality of his supervision, his patience, enthusiasm, motivation and immense knowledge. He convincingly guided and challenged me throughout this research. I am also deeply grateful for his great humor, natural cheerfulness, boundless encouragement and extensive and comprehensive support which have always pushed me to do things better and which have helped me a lot to overcome difficulties.

I would like to acknowledge with much appreciation my co-supervisor **Ph.D Imen Megdiche** for the quality of her supervision, her consistent support, tirelessly guidance, continuous encouragement through the process of researching and writing this thesis. Her great humor, great listening skills and great patience kept me going on.

I would like to express my deep appreciation to all committee members for letting my defense be an enjoyable moment. My sincere thanks goes to my thesis reviewers **Professor Anne LAURENT** and **Professor Claudia RONCANCIO** for agreeing to evaluate my thesis work. Their insightful comments, valuable suggestions and inspiring questions also helped me to improve my work.

I would like to thank my thesis examiners, **Professor Jérôme DARMONT** and **Professor Chantal SOULÉ-DUPUY**, and invited committee member **Mr. Grégory LOPEZ** for their valuable comments, shared ideas and helpful suggestions.

My sincere thanks goes to the *Direction du Système d'Information et de l'Organisation* of **Centre Hospitalier Universitaire de Toulouse** for funding the project, offering me the opportunity and giving me a lot of autonomy for my scientific research.

My sincere thanks also goes to **all my fellow colleagues** in *Univertité Toulouse I Capitole*, *Institut de Recherche en Informatique de Toulouse* and *Centre Hospitalier Universitaire de Toulouse* for their sincere encouragement and all the fun we have had in the last three years. I wish to extend my special thanks to **Mr. Prosper BURQ** for his continuous encouragement, precious advice with his skills and talents in his own field.

I am extremely grateful to **my parents**, Zhihua YANG and Xinxiu ZHAO, for their love, their care and sacrifices. I am very much thankful to **my boyfriend**, Gérard GOACHET, for his love, understanding and never-ending support.

Résumé

À l'ère du Big Data, les données sont caractérisées par le volume, la vitesse, la variété, la véracité et la valeur (5V). L'enjeu majeur du Big Data, au-delà du stockage, est d'extraire de la valeur de qualité à travers des analyses avancées sur des données volumineuses, véloces et variées. Depuis une décennie, le Lac de données (LD) est apparu comme une nouvelle solution répondant à cet enjeu de Big Data Analytics.

En tant que concept relativement nouveau, le lac de données n'a pas de définition standard ni d'architecture reconnue. Les propositions de la littérature sont insuffisantes au regard de l'ampleur du contexte. Notre première contribution se résume en une définition complète ainsi qu'une architecture générique du lac de données qui contient une zone d'ingestion, une zone de préparation, une zone d'analyse et une zone de gouvernance de données.

De plus, afin que les lacs de données ne soient ni invisibles ni inaccessibles par ses différents d'utilisateurs, une gouvernance est vitale. L'élément central d'une bonne gouvernance est un système de management de métadonnées. Dans la littérature, les approches de management des métadonnées sont parcellaires et pas nécessairement génériques pour les LD.

La contribution majeure de cette thèse est une solution complète de management de métadonnées permettant aux utilisateurs de trouver, d'accéder, d'interopérer et de réutiliser facilement aussi bien des données que des processus ou des analyses effectuées par le LD.

Dans un premier temps, nous avons proposé un modèle de métadonnées permettant de gérer tout le cycle de vie des données dans un LD comme suit : (i) métadonnées représentant différents types de données ingérées (structurées, semi structurées et non structurées) et différents modes d'ingestion (batch et en temps réel), (ii) métadonnées représentant différents processus de transformation des données (ETL, exploration statistiques et phase de préparation en science des données) au travers de la spécification d'opérations de haut niveau, (iii) métadonnées orientées analyse et notamment l'apprentissage automatique pour caractériser les analyses effectuées dans le LD et de pouvoir réutiliser et paramétrer rapidement les futures analyses.

Dans un second temps, nous avons défini un système de gestion de métadonnées, nommé DAMMS. DAMMS permet (i) d'ingérer de manière semi automatique des métadonnées et (ii) d'explorer le contenu du LD (données, processus de transformation ou analyses) de manière ergonomique afin de pouvoir les réutiliser ou les adapter. DAMMS présente ainsi l'avantage de répondre au besoin d'industrialisation de la science des données. Enfin, pour évaluer la faisabilité et l'utilisabilité de notre proposition, nous avons mené conjointement une étude de performance de l'ingestion des métadonnées et une étude analysant l'expérience utilisateur de DAMMS.

Summary

In the era of Big Data, data is characterized by volume, velocity, variety, veracity and value (5V). The major challenge of Big Data, beyond storage, is to extract quality value through advanced analytics on voluminous, fast and varied data. Over the past decade, Data Lake (DL) has emerged as a new solution to address this Big Data Analytics challenge.

As a relatively new concept, data lake has no standard definition or recognized architecture. The proposals in the literature are insufficient for the scope of the context. Our first contribution is a comprehensive definition and a generic architecture of data lake that contains an ingestion zone, a preparation zone, an analysis zone and a data governance zone.

Furthermore, in order to ensure that a data lake is neither invisible nor inaccessible to its various users, the data lake governance is vital. The central element of good governance is a metadata management system. In the literature, approaches to metadata management are fragmented and not necessarily generic for DL.

The major contribution of this thesis is a comprehensive metadata management solution that allows users to easily find, access, interoperate and reuse data as well as processes or analyses performed by the DL.

As a first step, we proposed a model to manage the entire data life-cycle in a DL as follows: (i) metadata representing different types of ingested data (structured, semi-structured and unstructured) and different ingestion modes (batch and real-time), (ii) metadata representing different data transformation processes (ETL, statistical mining and the preparation phase of data science) through the specification of high-level operations, (iii) metadata that are oriented to analysis and in particular machine learning to characterize the analyses performed in the DL and to be able to reuse and quickly parameterize future analyses.

In a second step, we defined a metadata management system, named DAMMS. DAMMS allows users (i) to automatically ingest metadata and (ii) to explore the content of the DL (data, transformation processes or analyses) in an ergonomic way in order to be able to reuse or adapt them. DAMMS thus has the advantage of responding to the need for data science industrialization. Finally, in order to evaluate the feasibility and usability of our proposal, we have jointly conducted a performance study of metadata ingestion and a study analyzing the user experience of DAMMS.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Research Context

According to the report "Data Age 2025" (Reinsel et al., 2017), published by International Data Corp, the global data sphere is expected to grow to 163 zettabytes (ZB) in 2025, a volume ten times greater than the 16.1 ZB of data generated in 2016. Beyond the volume, the user have to deal with the variety of data without forgetting the data velocity which is commonly supported by social networks or Internet of Things (IoT). Exploiting large, veloce and varied data requires processes to be put in place to manage its veracity. So that we are able to carry out relevant analyses for users. All these characteristics constitute today, what we cal l the 5V big data: Volume, Velocity, Variety, Value and Veracity (Demchenko et al., 2013).

Among the 5Vs, **Value** is the most important characteristic regarding business concern. Back to 2006, Clive Humby declared that "*data is the new oil*" (Arthur and editor, 2013). Afterwords, (Palmer, 2006) extended the opinion, he thinks "*data is just like crude*", to obtain the value, data have to be analyzed. For enterprises, it is important to analyze data to compile comprehensive information for decision making or problem solving. However, the fact of voluminous, heterogeneous and fast created data makes the value digging and veracity checking difficult. The ability to effectively and efficiently extract knowledge from data is now seen as a key competitive advantage (Cavanillas et al., 2016).

Today, one of the most popular big data analytics solutions dedicated to industrialize data science and to reduce the time and money consumption of data analysis is data lake (Syed, 2020). The concept data lake was put forward by (Dixon, 2010) in the industrial world and then has been studied at the academy. A data lake ingests raw data (structured, semi-structured and unstructured) from various sources, stores data in their native format and processes data upon usage (Fang, 2015; O'Leary, 2014; Miloslavskaya and Tolstoy, 2016). Its ambition is to offer various capacities such as data ingestion, processing and analysis to different users (e.g. data scientists, analysts and BI professionals).

1.2 Problem Statement

Data lake is a relatively new concept which was put forward only a decade ago, even though there are already some so-called data lake solutions proposed by Information Technology (IT) companies (Amazon, 2016; Azure, 2016) and some academic works that focus on this subject (Fang, 2015; Madera and Laurent, 2016; Inmon, 2016; Munshi and Mohamed, 2018), there is not a standard definition nor an acknowledged architecture of data lake.

Moreover, one of the basic characteristics of data lake is "schema on read", which means that raw data are ingested without transformation, stored in their native format and processed until needed. This characteristic ensures the flexibility and the possibility of being able to find the original data of analyses at any time of data lake. Nevertheless, the ingestion of a great volume of heterogeneous data (structured, semi-structured and unstructured) without implicit information makes data lake easily turning into a data swamp which is invisible, inaccessible and unreliable to users (Paschalidi, 2015).

To prevent a data lake from turning into a data swamp, metadata management is emphasized by different authors (Walker and Alrehamy, 2015; Hai et al., 2016; Halevy et al., 2016a,b; Diamantini et al., 2018). Metadata can help users to find data that correspond to their needs, accelerate data accesses, verify data origin and processing history to gain confidence and find relevant information to enrich their analyses. However, the existing metadata solutions can not perfectly be adapted to data lakes. Firstly, current metadata solutions mainly only focus on ingested datasets whereas the transformation processes and preformed analyses in data lakes are not described with metadata to ensure their reuse. Secondly, most of the existing works of metadata management of data lakes are not introduced with a formalized model, the works that presented metadata model do not show details of used metadata (no attributes). Thirdly, the presented metadata management systems rarely provide both the functions of automatic metadata generation and user-friendly exploration with predefined and free queries.

1.3 Research Overview

The objective of this thesis is to propose a metadata management solution for data lakes to improve the effectiveness and efficiency of big data analytics by helping users to find, access, interoperate and reuse ingested datasets, transformation processes and performed analyses. This solution must be based on a complete metadata model and a system making it easy to ingest and above all explore the metadata of all the elements stored in the data lake.

1.4 Manuscript Outline

The thesis is organized in three parts. The first part concerns the presentation of the context of data lake metadata management. It is presented in one chapter.

- In the **chapter 2**, we study the existing data lake definitions and functional architectures and propose our proper definition and architecture of data lake to agree on the basis of the thesis. Moreover, we compare the existing metadata solutions of data lakes.

The second part concerns the presentation of our proposed metadata model for data lakes. It is composed of three chapters.

- In the **chapter 3**, we focus on the metadata of data ingestion in data lakes. We bring up a ingestion metadata management solution that can be adapted to different types of data sources (IoT objects, databases, files), different structural types of datasets (structured, semi-structured and unstructured) and different ingestion modes (batch and real-time). The solution is twofold: a metadata model and formalized metadata integration processes for the relative metadata.

- In the **chapter 4**, we focus on the metadata of data preparation in data lakes. We bring up a preparation metadata management solution that can be adapted to different types of data processes (transformation process of ETL used in Business intelligence (BI), data mining, machine learning (ML)) and different ingestion modes (batch and real-time). The solution is twofold: a metadata model and formalized metadata integration processes for the relative metadata.

- In the **chapter 5**, we focus on the metadata of data analysis in data lakes. We bring up an analysis metadata management solution that capitalizes all relevant experiences, such as used datasets, characteristics of analyses (e.g. name, type, creation date, description), information of analysis implementations (e.g. implementation environment, used algorithms, parameters, script URL), evaluation of analysis result (e.g. output model, evaluation measure, result) in a data lake to improve the efficiency of data analysis. The solution is twofold: a metadata model and formalized metadata integration processes for the relative metadata.

The third part concerns the validation of our proposed solution. It is composed of two chapters.

- In the **chapter 6**, we present our implemented system DAMMS: DAta lake Metadata Management System which is based on our proposed metadata model. This system has two main function: (1) it integrates the proposed metadata in a semi-automatic way and (ii) it allows users to explore all the metadata of stored elements in the data lake with hierarchical navigation and ergonomic interfaces.
- In the **chapter 7**, we present the results of experiments of DAMMS. The experiments have objective of validating the effectiveness of metadata generation and the satisfaction of users utilization.

Finally, we conclude the manuscript in the **chapter 8** which concerns a balance sheet of our contribution and a presentation of our future work.

Chapter 2

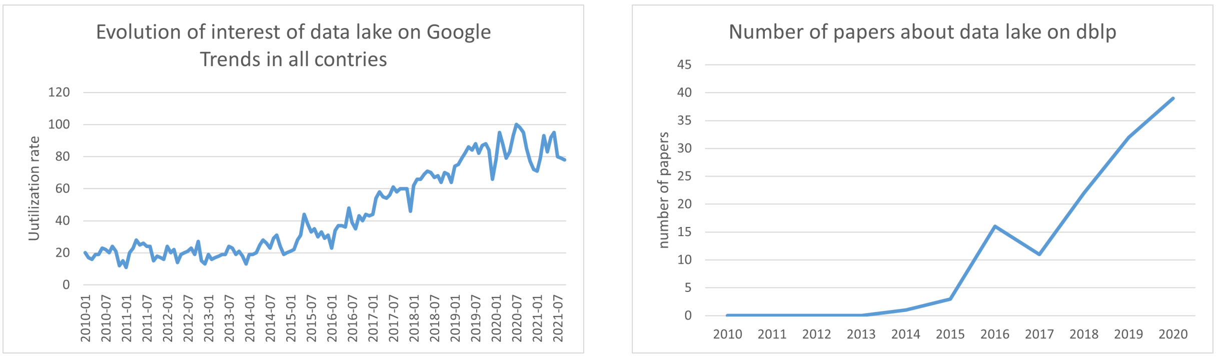
Data lake Context

Contents

This chapter is dedicated to define the scope of the PhD proposal. The scope of the thesis is based on three key concepts: data lake definition, data lake architecture and metadata management. In this chapter, we will study the three concepts in details. Firstly, we will introduce and compare different data lake definitions in eight aspects and then propose our own data lake definition to cover different aspects of data lake. Secondly, we will present and discuss different existing data lake architectures and then propose our own architectures to consider different functions of data lake. Last but not least, we will study metadata management which is one of the most essential concept of data lakes. We will introduce and discuss different solutions in the aspects of metadata model and metadata management system.

2.1 Data Lake Definition

In both academia and industry, data lake is a trendy and popular solution of big data analytics (see Fig. 2.1a[1]). However, today, data lake does not have a commonly acknowledged definition, different definitions have different emphases. In the following, we give an overview of the different definitions in the literature then we present our proper definition of data lake which is more complete and detailed then the state-of-the-art.



(a) General evolution (b) Evolution in academic world

Figure 2.1: Evolution of the interest of data lake

2.1.1 Related Work

The concept of data lake was initially emitted in the industrial world in 2010 to show the interest of the first Hadoop release. It was (Dixon, 2010) who put forward the concept at the aim of improving the data reporting and analysis based on Hadoop:

"If a data warehouse may be a store of bottled water – cleansed and packaged and structured for easy consumption – the data lake is a large body of water in a more natural state. The contents of the data lake stream in from a source to fill the lake, and various users of the lake can come to examine, dive in, or take samples".

From this presentation, Dixon pointed up that a data lake is a data repository which stores heterogeneous data in their native format and serves different users. However, this

[1]https://trends.google.fr/trends/explore?date=2010-01-01%202021-11-14&q=data%20lake

explanation cannot be treated as a formal definition and it had been unclear for a few years.

Since 2014, the academic world has started to study data lake and researchers have become more and more enthusiastic about this new big data analysis solution (see Fig. 2.1b[2]).

(O'Leary, 2014) and (Fang, 2015) proposed the first academic works about data lake. In their work, data lake has the following characteristics: (i) single data repository, (ii) designed for an enterprise and (iii) closely tied to Apache Hadoop technologies. They also compared data lake with data warehouse which is another popular data analytics solution. Both of them indicated some important data lake characteristics such as the data sources of data lakes are various (structured, semi-structured and unstructured), data are stored in their native format and are processed upon usage (schema-on-read).

A flurry of definitions have succeeded Dixon's definition from 2014 until today. To analyze different definitions, we propose 8 features that can characterize data lakes: data sources, data source types, ingestion mode, process, analysis, data storage, data governance, architecture, technology and user types. We synthesize in table 2.1-2.2 the state-of the art definitions according to the features:

- Regarding **data sources**, they are expanded from enterprise data (Fang, 2015) to internal and external data (Suriarachchi and Plale, 2016b; Maccioni and Torlone, 2018). The type of data sources can be unstructured, semi-structured or structured.
- Regarding **data ingestion**, its mode is expanded from batch (Fang, 2015; Miloslavskaya and Tolstoy, 2016) to real-time and batch (Azure, 2016).
- Regarding **data process and analysis**, the first data lake definitions do not allow users to process and analyze data while new definitions integrate these functions (Amazon, 2016; Azure, 2016; Couto et al., 2019; Giebler et al., 2021).
- Regarding **data storage**, all the approaches ingest raw data as native format.
- Regarding **data governance**, metadata management and data security, privacy and quality control are applied to new data lake definitions (Walker and Alrehamy, 2015; Alserafi et al., 2016; Madera, 2018; Giebler et al., 2021).
- Regarding **data lake architecture**, it is developed from flat architecture (Fang, 2015; Walker and Alrehamy, 2015) which has a single repository to multi ponds/-zones architecture (Azure, 2016; Inmon, 2016; Panwar and Bhatnagar, 2020).
- Regarding **data lake technologies**, various techniques (RDBMS, NoSQL, machine learning) (Giebler et al., 2021) are used to replace the mono Hadoop system.
- Regarding **users**, we notice that most approaches do not specify the types of users for whom the data lake is intended.

2.1.2 Our Data Lake Definition

As mentioned in the previous subsection, the different definitions are vague, they are not integrated with each other or are even contradictory. To be as complete as possible, we propose a definition that includes input, process, output and governance of data lakes (Ravat and Zhao, 2019a).

[2]`https://dblp.org/search?q=data+lake`

	Data sources	**Data source types**	**Ingestion**	**Process**	**Analysis**
(O'Leary, 2014)		all structural types	batch, real-time		
(Walker and Alrehamy, 2015)					
(Fang, 2015)	enterprise data	unstructured or multi-structured			
(Hai et al., 2016)				on-demand integration	
(Farid et al., 2016)					
(Alserafi et al., 2016)					
(Inmon, 2016)		all structural types			
(Suriarachchi and Plale, 2016b)	different sources	all structural types		support data transformations	support data analysis
(Miloslavskaya and Tolstoy, 2016)			ingest without compromising		
(Amazon, 2016)		structured and unstructured data	allow processing		allow different types of data analytics
(Azure, 2016)		different types	batch, streaming	allow data processing	all data analytics
(Maccioni and Torlone, 2018)	internal or external sources		without ETL process		
(Madera, 2018)		all formats	no transformation		
(Couto et al., 2019)		all structural types	support data ingestion	support data processing	support data analysis
(Panwar and Bhatnagar, 2020)		various schemata and structural forms			
(Giebler et al., 2021)				allow data processing	

Table 2.1: Different data lake definitions (part 1)

	Data storage	**Data governance**	**Architecture**	**Technology**	**Users**
(O'Leary, 2014)			flat architecture - single repository	Hadoop	
(Walker and Alrehamy, 2015)	raw data		flat architecture - single repository		
(Fang, 2015)	raw data		flat architecture - data repository	Hadoop, low cost technologies	
(Hai et al., 2016)	raw data				
(Farid et al., 2016)	data in native format				
(Alserafi et al., 2016)	raw data	metadata management			
(Inmon, 2016)	raw data		multi ponds		
(Suriarachchi and Plale, 2016b)					
(Miloslavskaya and Tolstoy, 2016)	raw data				
(Amazon, 2016)	raw data				
(Azure, 2016)	raw data	metadata management, rules of data governance			data scientists
(Maccioni and Torlone, 2018)	keep original format				
(Madera, 2018)	raw data		reference multi zones architecutre	Hadoop ecosystem	
(Couto et al., 2019)	raw data	metadata management, security, privacy, quality	multi zones technical architecture	different technologies	
(Panwar and Bhatnagar, 2020)	raw data				
(Giebler et al., 2021)					

Table 2.2: Different data lake definitions (part 2)

Definition 2.1.2. Data lake is a big data analytics solution that allows different types of users (data scientists, data analysts, BI professionals etc) to:

- ingest heterogeneous (structured, semi-structured, unstructured) raw data from various sources (local or external to the organization) and store these raw data in their native format,
- process data to prepare for new business requirements,
- analyze stored available data for different types of data analysis (statistical analysis, data reporting, queries, data visualization and Machine Learning (ML), etc),
- govern data to ensure the data quality, data security and data life cycle.

2.2 Data Lake Architecture

After having presented the definition of a data lake, the second important concept is the architecture of the data lake. We can distinguish data lake architecture between functional and technical architectures. We are interested in the functional architecture from a conceptual point of view. In this section, we summarize the different definitions of functional architecture proposed in the literature and we propose our own definition.

2.2.1 Related Work

The functional architecture of data lake has evolved from mono-zone to multiple zones.

2.2.1.1 Flat architecture

Until now, researchers have been working on technical architectures of data lakes. The first vision of data lake functional architecture was a flat architecture with mono-zone that is closely tied to the HADOOP environment (Fang, 2015; Walker and Alrehamy, 2015). This zone allows users to collect and store raw data in their native formats. It includes the data from web logs, sensor devices, operational data store (ODS) and online transaction processing (OLTP) systems. The advantage of this architecture is to enable loading heterogeneous and voluminous data with a low cost.

Discussion: The flat architecture is simple and can be hardly realized especially in the context of big data analytics. As a matter of fact, this architecture does not reflect the activities performed by data scientists, such as the pre-processing phases inherent in decisional analyzes with a set of intermediate data storages. To overcome these drawbacks, some extended visions of architecture with multiple ponds or zones were proposed with a more diverse technological environment.

2.2.1.2 Multi-ponds architecture

Inmon (2016) proposed an architecture with multi-ponds which deals with different types of data in different ponds (see Fig. 2.2):

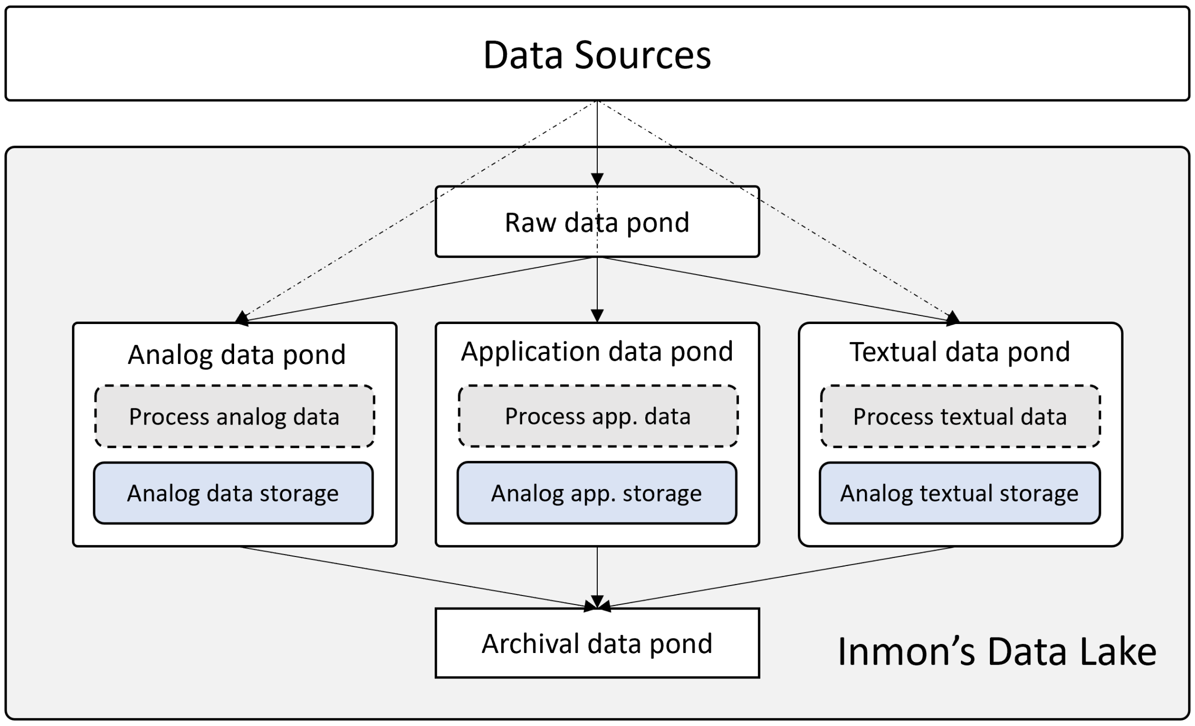


Figure 2.2: Multi ponds data lake architecture

- Data sources can be ingested in the *raw data pond*. This pond is not used for performing analysis but for storing jumble data. Data that are stored in the raw data pond should be processed and sent to the supporting data ponds (analog, application, textual data ponds) as quickly as possible. Once raw data are refined and passed to other ponds, they are removed from the raw data pond. It is as well possible that data do not pass the raw data pond but directly go the supporting data pond.

- The *analog data pond* is designed for analog data. Analog data are typically generated by machines or connected objects, they have voluminous and repetitive characteristics. These data are usually reduced or compressed then stored in the analog data pond.

- The *application data pond* is designed for data that are created by applications or transactions. Raw data need to be integrated and stored in this pond. The application data are uniformly structured and relevant to business activities.

- The *textual data pond* is designed for unstructured text data. These data should pass the textual disambiguation phase to be able to be analyzed by computers.

- The *archival data pond* is used to store data that are not actively needed. All other ponds can send data to the archival data pond.

Discussion: This architecture ensures the ingestion of different structural types of data. The classification of three types of data (analog, application and textual) makes data finding faster and data analytics easier. However, not all of the data types are considered, such as images, videos, sound files. Moreover, in this data lake, once the raw data are processed and sent to other ponds, they are no longer remained in the raw data pond.

This design is contrary to one of the basic characteristics of data lake which is to keep all raw data and to avoid creating data silos.

2.2.1.3 Multi-zones architecture

There are different multi-zone architectures proposed in the academic and industry world. In this subsection, we introduce five architectures: the architectures of (Mehmood et al., 2019), (John and Misra, 2017), (Panwar and Bhatnagar, 2020), Amazon Web Services (Nadipalli, 2017) and Zaloni (LaPlante, 2016).

Architecture of (Mehmood et al., 2019)

(Mehmood et al., 2019) proposed a data lake architecture of data lakes that strongly relies on Hadoop Ecosystem, in particular, Cloudera Distribution including Hadoop. This architecture allows data collection, storage and processing of diverse data (see Fig. 2.3).

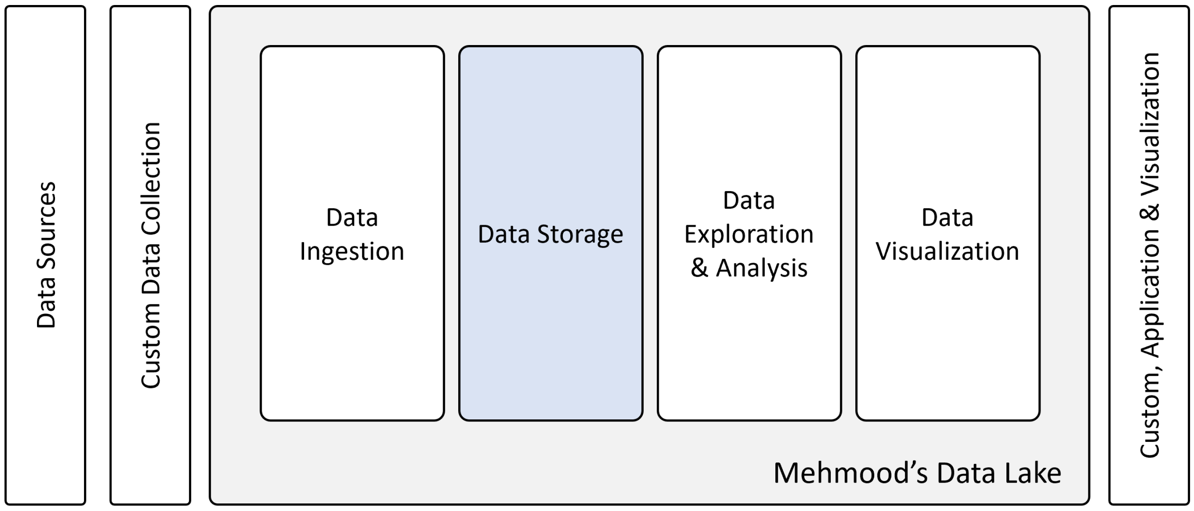


Figure 2.3: Data lake architecture of (Mehmood et al., 2019)

- *Custom data collection* enables data retrieval from data sources which requires custom scripts. It allows users to pre-process data sources.
- *Data ingestion* concerns the process of importing data from external sources to the data lake in different modes (batch or real-time).
- *Data storage* is used to store data in Hadoop Distributed File System (HDFS).
- *Data Exploration & Analysis* is for processing and analyzing data.
- *Data visualization* is for drawing insights from data.

Discussion: The architecture indicates the different functions of a data lake: data ingestion, storage, exploration, analysis and visualization. However, this architecture allows users to customize data collection before the ingestion phase and outside of the data lake, which makes it difficult to retrieve the data source. In addition, there is no data governance in the data lake which results in the impossibility of data security and quality control.

Architecture of (John and Misra, 2017)

(John and Misra, 2017) proposed a data lake architecture leveraging Lambda architecture to ensure batch and real-time data processing. This architecture divided the overall data lake architecture into different layers with the help of Lambda Architecture pattern:

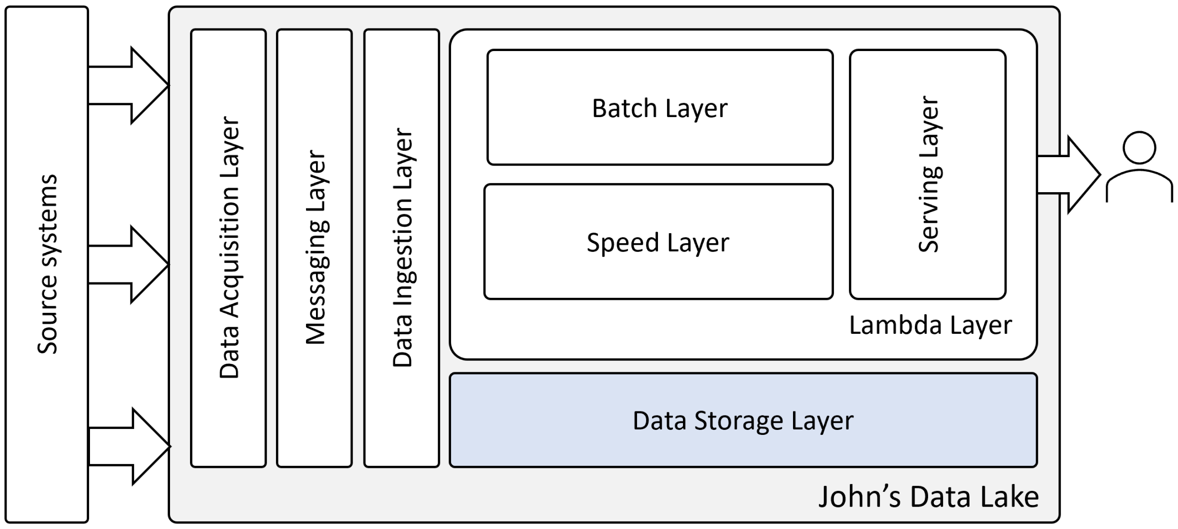


Figure 2.4: Data lake architecture of (John and Misra, 2017)

- *Data acquisition layer* is used to get data from various sources (structured, semi-structured and unstructured).
- *Messaging layer* is used to guarantee data delivery.
- *Data ingestion layer* is used to ingest data for processing and storage. It should allow highly scalable loading, data recovery and resiliency and multi-thread execution.
- *Data storage layer* is used to store all data.
- *Lambda layer:*
  - *Batch layer* is used to ensure batch processing of ingested data. Its primary responsibility is the conversion of raw data to modeled data.
  - *Speed layer* is used to ensure near-real-time data processing on the data that are received from the ingestion layer.
  - *Serving layer* is used to delivery and export data to the consuming application.

Discussion: This architecture considers both batch and (near) real-time data processing by integrating the Lambda architecture. Moreover, the data lake allows users to ingest, process and analyze various types of data (structured, semi-structured and unstructured). However, the authors did not consider the governance of data lake. So, the data life-cycle, security and quality can not be ensured.

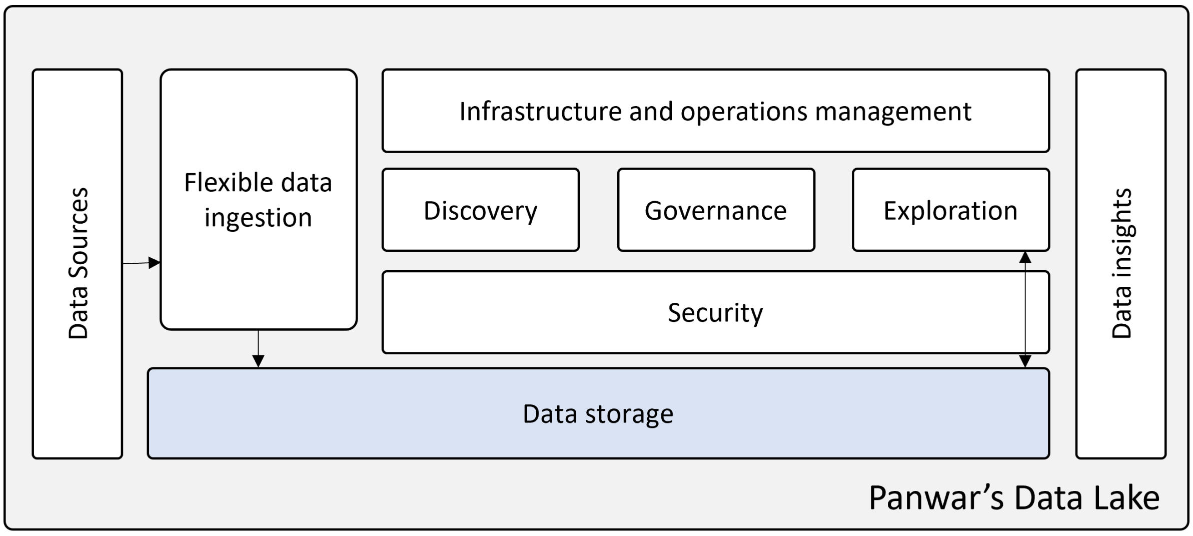


Figure 2.5: Multi-tiers data lake architecture

Architecture of (Panwar and Bhatnagar, 2020)

(Panwar and Bhatnagar, 2020) proposed a reference architecture of data lake which contains 9 different tiers and each tier corresponds to one function/responsibility (see Fig. 2.5):

1. *Data source tier* contains different types of data sources (batch /real-time, structured /semi-structured /unstructured).
2. *Flexible data ingestion tier* is for extracting data from various data sources as shown in the first tier.
3. *Data discovery tier* helps users to understand and search data by ensuring the metadata storing, auto-tagging and keyword search.
4. *Data governance tier* ensures (i) data quality which is essential for users to get actual business insights; (ii) data lineage which concerns the data life-cycle from the beginning source to final destination; and (iii) data auditing which is the technique of recording data access and modifications.
5. *Exploration tier* helps users to explore data by getting more details after data discovery.
6. *Data security tier* protects sensitive data in different levels (dataset, entity/attribute and cluster).
7. *Infrastructure and operation management tier* concerns the establishment of an Hadoop ecosystem for a data lake.
8. *Data storage tier* stores different types (structured, semi-structured and unstructured) and stages (raw, in-process and curated) of data and provides accesses of the stored data.
9. *Insights tier* concerns the business values of data which can be got from data reporting, analysis and visualization.

Discussion: This architecture has the advantages of centralizing data store, being ensured by security and governance control and being able to help users to discover and explore data. However, the architecture is ambiguous. Firstly, data sources which should be external to the data lake are shown in the architecture. Secondly, different tiers have common functions and they are not well explained. For instance, both *infrastructure and operation management tier* and *governance tier* have the auditing function; both *discovery tier* and *security tier* have the searching function. It seems that functions are overlapping in different tiers and the border or objective of the different tiers is not clear.

Architecture of Amazon Web Services (AWS)

Amazon uses a data lake architecture containing four zones: ingestion zone, storage zone, processing zone and a govern & secure zone (Nadipalli, 2017). Data from various sources are extracted and loaded in the ingestion zone. These raw data are stored in the storage zone. When data are needed, they are structured in the processing zone. Govern & secure zone is for controlling the data security, data quality, metadata management and data life cycle (see Fig. 2.6).

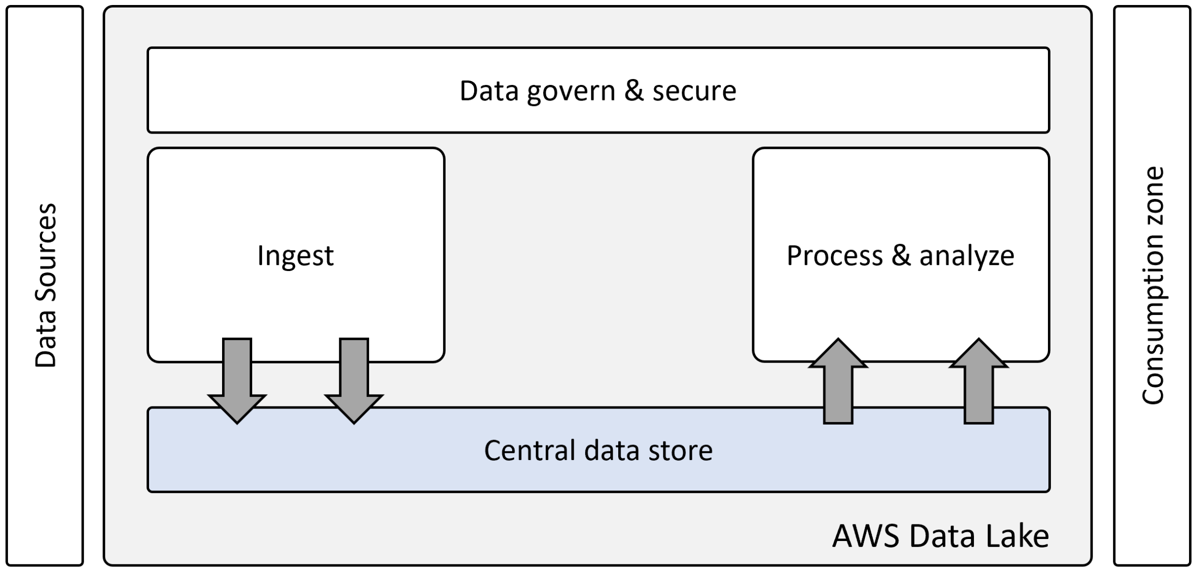


Figure 2.6: AWS data lake architecture

Discussion: This architecture clearly presents the data flow in a data lake. All data are ingested and centralized upon analysis. However, data flow is unidirectional in the schema. Raw data are ingested and stored in the central data store, but there is no data created by transformation processes and stored in the central store. Nevertheless, the cleansed and transformed data can be directly reused by users to improve the data processing effectiveness.

Architecture of Zaloni (LaPlante, 2016)

Zaloni data lake architecture (LaPlante, 2016) goes beyond the AWS architecture: it separated the processing and storage zones into refined data zone, trusted data zone and discovery sandbox zone (see Fig. 2.7). In the refined zone, data scientists and business analysts integrate and structure the data that they want to discover. Trusted data zone stores all the cleansed and validated data. Data for discovery and exploratory analysis moves from the trusted area to the discovery sandbox.

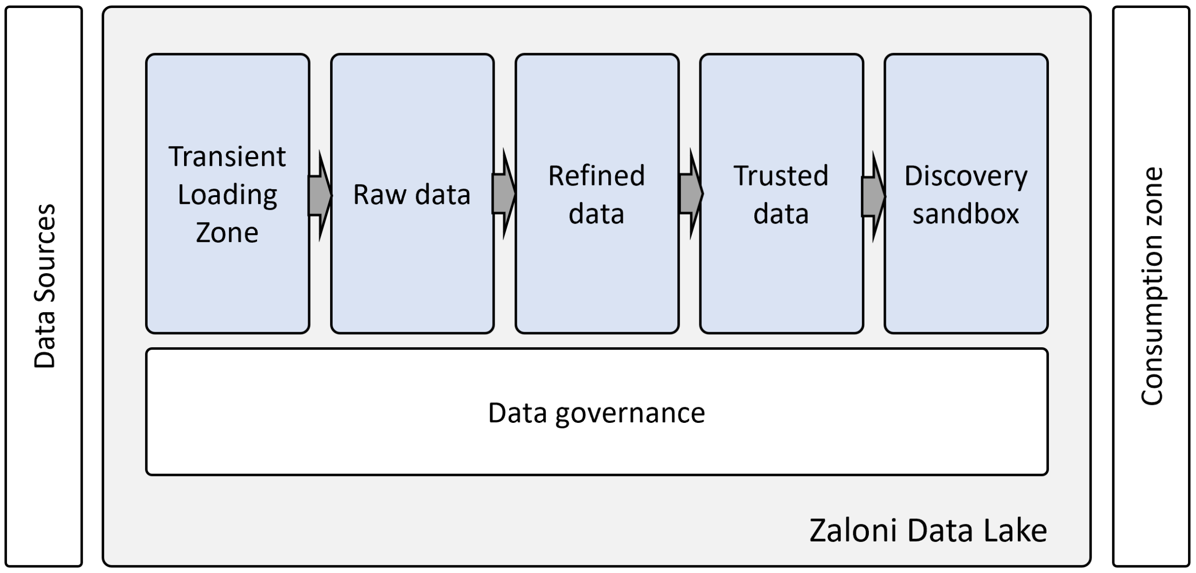


Figure 2.7: Zaloni data lake architecture (LaPlante, 2016)

Discussion: This architecture divided data into different zones according to their maturity. Its advantages are that users can more efficiently find refined or trusted data and that data are governed no matter they are in which zone. However, data lake is characterized by "schema on read", all the raw data are stored and the same data may need to be transformed in different ways for different objectives. For instance, a cancer table which contains missing values can be analyzed directly for one project, for which this dataset is trusted. But the same dataset may need to be transformed to replace all the missing values to be ready for the analysis, in this case, the dataset is not trusted anymore.

2.2.2 Our Data Lake Functional Architecture

Most the above-mentioned proposals are supported by software editors or IT companies. They are not independent of the inherent technical environment. Consequently, none of the existing architectures draws a clear distinction between functionality-related components and technology-related components. Nevertheless, the concept of multi-zone architecture is interesting and deserves further investigations. We believe that some zones are essential, while others are optional. Concerning the essential zones, based on our data lake definition, a data lake should be able to ingest raw data, process data upon usage, store processed data, provide access for different users and govern data.

In order to have a homogeneous definition of the zones and to meet the limits of the current solutions, we propose a generic functional data lake architecture which contains the four essential zones (Ravat and Zhao, 2019b). Our proposal is depicted in Figure 2.8.

Definition 2.2.2. A data lake functional architecture consists of four essential zones each of which contains two layers: data processing layer and data storage layer:

- **Ingestion zone**: all types of data are ingested, with the help of metadata tags, in their native format without processing. Big Data ingestion involves connecting to various data sources, extracting the data, and detecting the changed data. The ingestion processes may be implemented in batch, real time or hybrid processing.

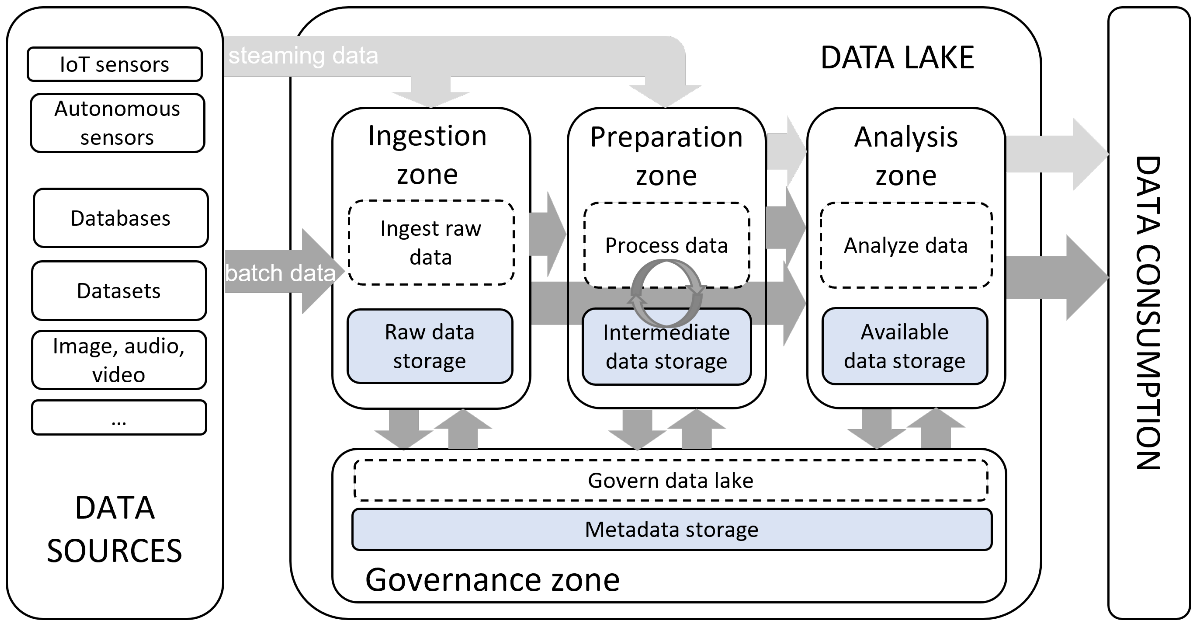


Figure 2.8: Our data lake architecture

- **Preparation zone**: in this zone, users can transform raw data into standardized formats according to their requirements. Data processing is an unavoidable step for data analysis. The processing layer includes batch and/or real-time processing.
- **Analysis zone**: this zone allows self-service data consumption and data are formatted for different analyses (reporting, statistical analysis, business intelligence analysis, machine learning algorithms).
- **Governance zone**: data governance is applied to all the other zones. It is in charge of insuring data security, data quality, data life-cycle, data access and metadata management. It is essential to prevent a data lake from deteriorating into a data swamp which is a massive data repository that is completely inaccessible to end users.

2.3 Data Lake Metadata Management

Once we have defined a data lake and its architecture, we now turn to the third key concept of this thesis, namely metadata management.

Data lake is characterized by "schema on read". In a data lake, different types of datasets (structured, semi-structured and unstructured) can be ingested in different modes (batch and real-time) without explicit needs or schemas, and different transformations and analyses can be applied on these datasets by different users (data scientists, engineers, analysts) upon usage. Therefore, a data lake can easily turn into a data swamp which is invisible, inaccessible and incomprehensible by users. To avoid this situation, data governance is emphasized (Derakhshannia et al., 2019, 2020) and many authors highlighted metadata management as the core of a data lake (Quix et al., 2016; Sawadogo et al., 2019; Diamantini et al., 2018; Halevy et al., 2016a).

2.3.1 Concepts

Before introducing different solutions of metadata management for data lakes, we define the concepts of metadata, metadata management and metadata management system.

2.3.1.1 Metadata

Metadata are often called "*data about data*" or "*information about information*". They have been used in information management long before computer science in fields related to documentation and cataloging (Zgolli et al., 2020). Metadata, according to the point of view of the National Information Security Organization (NISO, 2017), are defined as structured information for describing, explaining, localizing and aiding the retrieval, the management of an information resource.

Definition 2.3.1.1. In a data lake, metadata are structured information to describe and explain all the resources (datasets, preparation, analyses) stored in the data lake, and relationships between them as well as common sense about them.

2.3.1.2 Metadata Management

The metadata management, according to (whatls, 2021) and (Zgolli et al., 2020), seeks to provide business and technical users with easier access to integrated and high quality metadata. Among the benefits of metadata management, metadata management ensures the: (i) consistency of definitions of metadata, (ii) less redundancy of effort and greater consistency across multiple instances of data because data can be reused appropriately, (iii) maintenance of information across the organization and (iv) greater efficiency, leading to faster product and project delivery. We highlight that metadata management is used in different contexts such as in data preparation (Alexandropoulos et al., 19ed; Zhou et al., 2017; García et al., 2015), ontologies for machine learning (Esteves et al., 2016; Keet et al., 2015; Panov et al., 2014) or simply data cataloging in general.

Definition 2.3.1.2. In a data lake, the metadata management ensures the availability, quality and persistence of metadata through a set of feeding, updating and querying processes. So that users can find, access, interoperate and reuse all the stored elements (datasets, preparation, analyses) in the data lake.

2.3.1.3 Metadata Management System

To ensure a proper and efficient metadata management, a system should be implemented to help users to manage and use metadata (Quix et al., 2016; Liu et al., 2021; Sawadogo and Darmont, 2021). a unified schema of metadata (said also metadata model) shared transparently between users, and a system for metadata management that can help users to generate and explore metadata are essential.

Definition 2.3.1.3. For a data lake, a metadata management system is a system which is based on defined metadata standards and which can help users to generate and maintain metadata in an automatic way and explore them with ergonomic interfaces.

In this chapter we focus on and discuss only the flagship papers for the metadata management in data lakes.

2.3.2 Related Work

In this section we summarize the work related to metadata in data lakes. We study each of the state-of-the-art work according to three viewpoints: general presentation, proposed metadata and metadata management system.

2.3.2.1 Personal Data Lake

Walker and Alrehamy (2015) proposed a personal data lake design which has a unified storage of data regardless of their format (structured, semi-structured and unstructured) and a metadata management to help users to analyze and query metadata of personal data.

Metadata

In a Personal Data Lake, when a dataset (also called data fragments in this paper) is submitted, metadata about the dataset are submitted at the same time to record the information of the dataset. The metadata contains the following properties:

- *source*: a URI (Uniform Resource Identifier) that defines the provider of the dataset.
- *context*: a URI that defines the context of the dataset by pointing to referenced sources which indicate dataset categories, classes and schema. In addition, other semantic information, such as a set of ontologies, can also be added.
- *type*: a URI that indicates the type of the dataset.
- *assertion*: used to indicate whether the dataset schema is extracted directly from structured / semi-structured data or it requires more analysis for unstructured data.
- *custom abstraction (optional)*: a key-value pair which is added to represent facets of raw data.

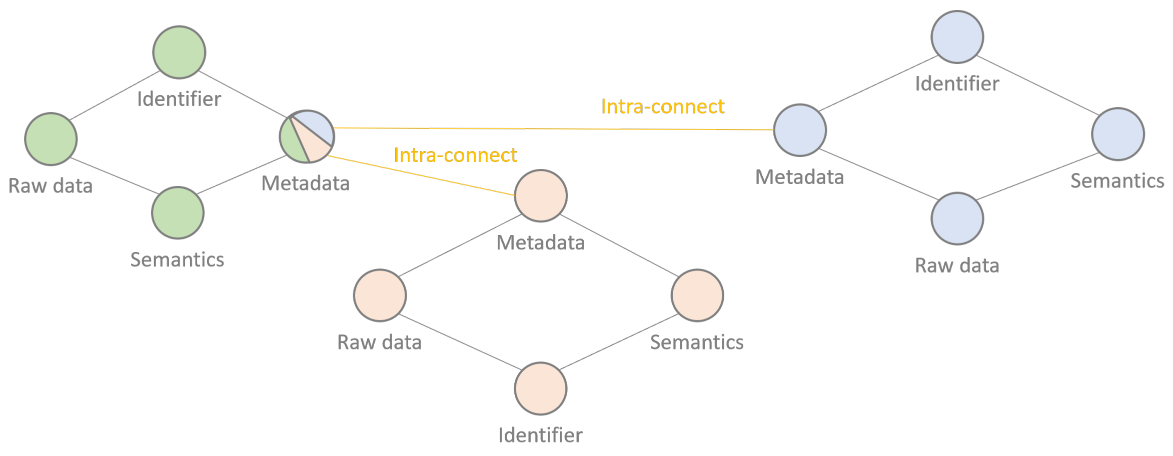


Figure 2.9: Metadata storage of Personal Data Lake

Metadata management system

A metadata management system is integrated in the Personal Data Lake Architecture. Each submitted dataset is serialized into a special JSON object (PLSF object). The PLSF object is then flattened into a sequence of string that carries metadata. The sequence is finally sliced and generated into different segments and stored in a graph database (Neo4j).

In the graph database, each dataset can be represented by four nodes which are grouped up by undirected edges (see Fig. 2.9): (i) a *metadata-node* which includes the previously presented information; (ii) a *rawdata-node* which refers to the raw dataset; (iii) a *semantics-node* which is a key-value pair representing a specific facet of raw data; and (iv) a *identifier-node* which is created to store the unique identifier of a dataset. In the graph, the metadata-node acts as a bridge to inter-connect different datasets when they share common metadata, for instance, two datasets share the context keyword "health".

All the stored datasets can be searched through a Functional Query Interface (FQI).

2.3.2.2 Constance Data Lake

Hai et al. (2016) presented the data lake system *Constance* which contains a metadata management framework to help users to discover, extract and summarize metadata from structured and semi-structured data (relational databases, JSON, spreadsheets and XML, etc.). The system can provide a common access interface to a data lake which is a schema-less repository of raw data.

Metadata

Constance can extract metadata from the data sources. According to (Hai et al., 2016), the extracted metadata are generated to fit a unified model. However, they did not present a formalized model, instead, they indicated different categories of metadata:

- Explicit schematic metadata from relational datasets, for instance, tables, attributes and constraints of relational database;
- Implicit schema metadata of semi-structured datasets, for instance, entity types, relationship types, and constraints of JSON or CSV files. These metadata are extracted by the *structured metadata discovery* component.
- Semantic metadata such as attribute annotation, record linkage. These metadata can help users to link different labels if they have the same tag and they are managed by the component *Semantic Metadata Matching*. The output of this component a graph representation of the extracted metadata nodes and their relationships.

Metadata management system

In Constance, a metadata management system is integrated which can extract explicit, implicit metadata and semantic annotations. Moreover, based on the extracted metadata, *Constance* can also cluster different schemas according to the distance between them and summarize schemas by filtering the most important elements or relationships. Nevertheless, there is an integrated user interface that allows users to query stored data in the data lake.

2.3.2.3 GEMMS

To reduce the upfront integration costs and provide a more flexible solution of data integration and analysis, (Quix et al., 2016) proposed GEMMS (Generic and Extensible Metadata Management System). GEMMS can automatically extract metadata from various datasets when they are loaded in their original format into the data lake repository.

Metadata
The proposed metadata in GEMMS can be classified into three categories:

- *Structure metadata* provide information about the structure of a dataset and its basic data types. GEMMS specifically extracts internal abstract tree (only element names and cardinalities) from tree-like datasets (e.g. XML, JSON files) and extracts matrix structural metadata from structured metadata.
- *Semantic metadata* provide annotations of datasets to describe their meaning. The simplest form of an annotation is an ontology term (an URI), but it can also to other types of identifiers.
- *Properties metadata* are a set of properties of datasets. These metadata are stored in key-value pairs to describe raw data, such as filename, size, location and last modification date.

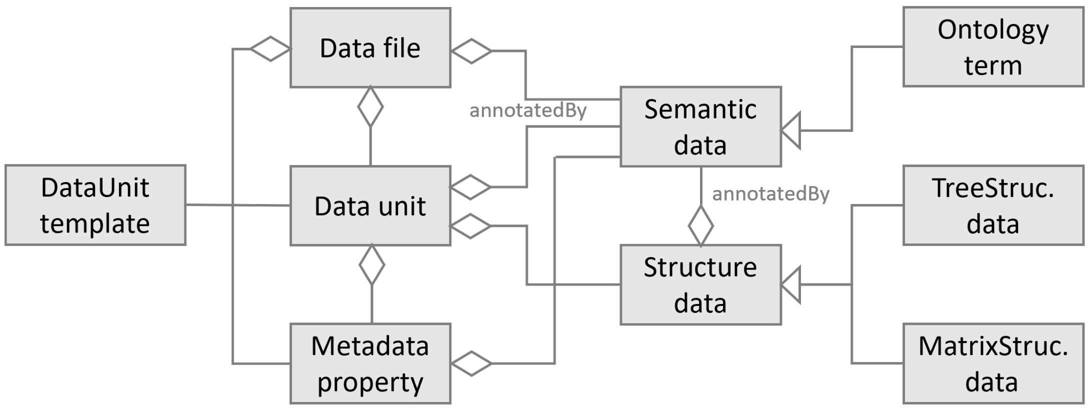


Figure 2.10: Conceptual model of GEMMS

These metadata are modeled into the model shown in Fig. 2.10. In the model, a *data file* is used to present a dataset (also called file in this paper), such as a database, a separate sheet or a XML file. A *data unit* is a piece of data in a dataset such as a table in a relational database or sub-tree in a XML file. Both *data file* and *data unit* can be described by *metadata property* and *semantic data*. Besides, a *data unit* can also be described by *structure data* because it carries raw data.

Metadata management system
The system has three main functions: (i) *Metadata extraction* which concerns the extraction of metadata from data sources, such as file type and structural information. This function is done by the *extractor component*. (ii) *Metadata transformation* which performs the transformation between models and the storage format. This function is

realized through the *serialization component*. And (iii) *Metadata storage* concerns storing the serialized metadata in a MongoDB database. This function is done with the*persistence component*.

2.3.2.4 GOODS (Google Dataset Search)

To handle structured and semi-structured datasets (structured files, databases, spreadsheets, etc.) at scale, (Halevy et al., 2016a,b) proposed a post-hoc system dedicated to organize the datasets that are generated and used in Google. The post-hoc manner allows the system to collect and aggregate metadata about datasets after the creation, access or updating of different pipelines without interfering with dataset owners or users.

Metadata

GOODS aggregates metadata in a central catalog and it also correlates metadata of a specific dataset with other dataset information. The catalog contains different types of information:

- *Basic metadata*: basic information of each dataset such as size, timestamp, format, owners, access permissions.
- *Provenance*: information extracted from productions or consumption, such as reading jobs, writing jobs, downstream and upstream datasets.
- *Schema*: information of the dataset structure.
- *Content summary*: information that summarizes the content of the dataset, such as frequent tokens and data fingerprint.
- *User-provided annotations*: information entered by users to describe the dataset such as description and annotations.
- *Semantics*: comments extracted from datasets who conform to a protocol buffer.
- *Relationships*: relationships among datasets including dataset containment, provenance, logical clusters and content similarity.

Metadata management system

A metadata management system is integrated in the GOODS system which has four principal services: (i) a *search engine* to narrow search results to help users find relevant datasets; (ii) a per-dataset *profile page* displays metadata of a specific dataset; (iii) a monitoring service to monitor datasets and alter users if they change unexpectedly; (iv) a *annotation service* to allow users to annotate datasets to extend dataset metadata.

2.3.2.5 Data Lake of Suriarachchi

Suriarachchi and Plale (2016b) focused on data provenance metadata in the case of process transformation for data lakes. They introduced the provenance integration challenge of data lakes and they proposed a reference architecture which contains a central provenance collection subsystem.

Metadata

The solution of Suriarachchi stores the fine-grained provenance information (source and result data of each process). As a simple example, for the data flow diagram presented in Fig. 2.11: the dataset (also called data product in this work) *d1* is transformed (*T1*) to produce datasets *d2* and *d3*. *T2* uses *d3* and a new dataset *d4* to generate *d5*, *d6* and *d7*. Finally, the dataset *d8* is created by *T3* with datasets *d6* and *d7*. The stored metadata are shown in Fig. 2.12.

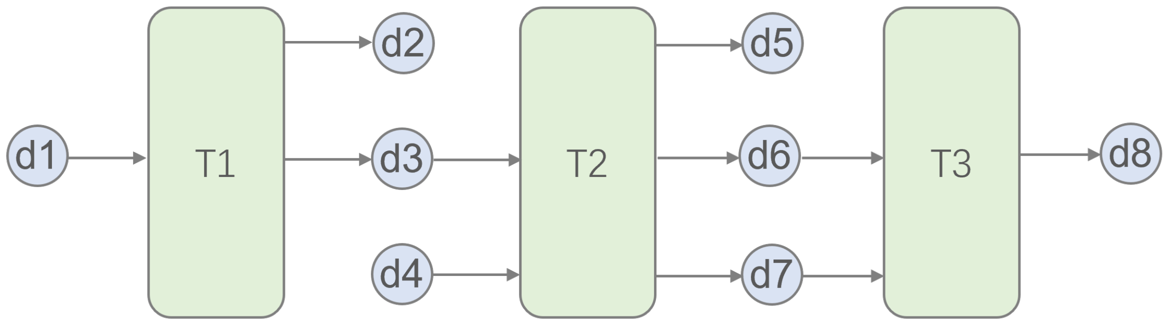


Figure 2.11: An example of data transformation flow

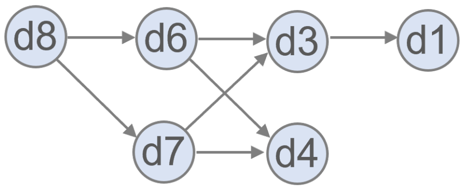


Figure 2.12: Data lineage of the example according to Suriarachchi

Metadata management system

The system proposed by Suriarachchi is not a complete metadata management system. Instead, it extracts only provenance information through its Ingest API and store this information in a graph database.

2.3.2.6 KAYAK Framework

Maccioni and Torlone (2018) proposed the framework KAYAK which can help data scientists to define, execute and optimize data preparation pipelines in a data lake.

Metadata

Regarding metadata management, KAYAK extracts metadata from datasets with ad-hoc primitives or tasks. The extracted metadata can be classified into intra-metadata and inter-metadata:

- Intra-metadata concern information of each signal dataset, for instance, descriptive, statistical, structural and usage information.
- Inter-metadata concern relationships between datasets including integrity constraints such as inclusion dependencies, and user defined relationships such as joinability and affinity.

Metadata management system

Maccioni and Torlone (2018) did not present a metadata management system in this work, they focused mainly on the data transformation pipelines.

2.3.2.7 Metadata model of Diamantini

To extract thematic views from heterogeneous and generally unstructured data sources, Diamantini et al. (2018) presented a metadata model that can structure unstructured data.

Metadata

The metadata model concerns a network-based model for business (business rules, such as upper and lower limit of a specific field and integrity constraints) and technical metadata (data format and schema). The model consists of two elements:

- *Objects* can be tables and attributes of relational databases, complex/simple element and their attributes of XML or JSON documents, keywords of unstructured datasets.
- *Relationships* are used to link different objects. There are three types of relationships: (i) *Structural relationship* which is used to present that an object "contains" another object. For instance, a relational database "contains" a table and this table "contains" its attributes, or a unstructured dataset "contains" a keyword. (ii) *Similarity relationship* which is used to present that an object is "similarTo" another object. (iii) *Lemma relationship* is used to present that a target object is a "lemma" of the source one. For instance, "impureness" is a lemma of "pollution".

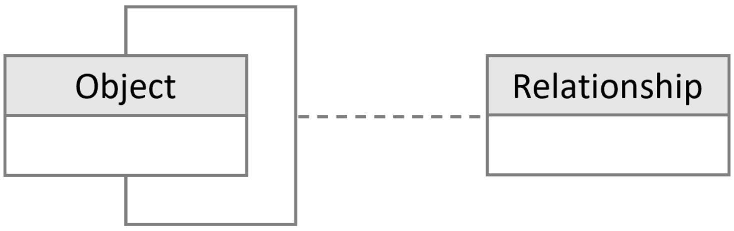


Figure 2.13: Metadata model of Diamantini

The generated metadata are represented in a graph, each object becomes a node and each relationship becomes an edge. There are two examples of generated datasets. The first one is a XML (semi-structured source) called *Climate* (see Fig. 2.14a). The second one is a unstructured source called *Environment Video* (see Fig. 2.14b).

Metadata management system

Diamantini et al. (2018) did not present a metadata management system in this work.

2.3.2.8 MEDAL

To ensure the efficiency and comprehension of metadata management, Sawadogo et al. (2019) proposed evaluation criteria through a list of features for data lake metadata systems and a metadata typology for MEDAL (MEtadata model for DAta Lakes).

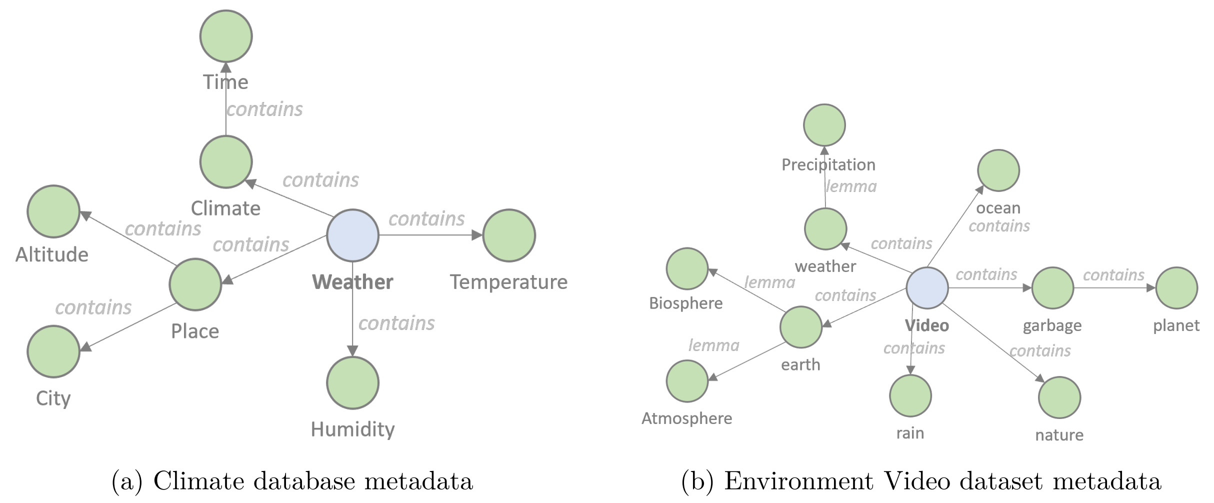


Figure 2.14: Examples of generated metadata model of Diamantini

Metadata

The authors proposed different metadata to manage datasets (also called data objects in this paper) that are stored in a data lake. The proposed metadata can be classified into three categories :

- *Intra-metadata* refer to information associated with datasets and include (i) *properties* which concern a general description of a dataset, such as dataset title, size, last modification date and access path, etc.; (ii) *summaries and previews* which concern a overview of the content (e.g. word cloud of textual datasets) or structure of a dataset (schema of structured or semi-structured datasets); (iii) *versions* which concern different updated datasets; (iv) *representations* which concern different transformation results of datasets; and (v) *semantic metadata* which concern annotations of dataset meaning, such descriptive tags, textual descriptions or business categories.

- *Inter-metadata* refer to relationships between different datasets and they can be: (i) *objects groupings* which organize datasets into collections according to semantic metadata (tags, business categories) or some properties (e.g. format or language); (ii) *similarity links* concern the similarity between datasets. Similarity are calculated based on summaries and previews metadata (content or schema information); (iii) *parenthood relationship* concern data transformation source and result facts to ensure the traceability of data in a data lake.

- *global-metadata* are applied to the entire data lake instead of specific datasets and include (i) *semantic resources* which concern knowledge bases (e.g. ontologies, taxonomies, thesauri, dictionaries); (ii) *indexes* which are data structures such as keywords of textual datasets, patterns or colors of images; and (iii) *logs* which are used to track user interactions in the data lake, such as logging in, viewing, modifying records.

Metadata management system

Sawadogo et al. (2019) did not present a metadata management system in this work. However, they proposed six main features that can be used to evaluate a metadata management system: semantic enrichment (semantic annotation / profiling), user tracking, link generation and conservation, data polymorphism, versioning and indexing.

2.3.2.9 HANDLE

To ensure the exploitation of data value in a data lake, Eichler et al. (2020) proposed HANDLE (Handling metAdata maNagement in Data LakEs). HANDLE is a generic model for metadata which enables comprehensive metadata management.

Metadata

HANDLE consists of two parts, a core model and three core model extensions adapted to different implementations (see Fig. 2.15).

The core model defines all the required elements and relations to model metadata. The core model consists of three entity classes: *data*, *metadata* and *property*. (i) A *data* represents a pointer to a stored data element. To consider different granularity of a dataset, a data element can be a dataset or its component (a row, a key-pair or a frame etc.). The attribute *storageLocation* is used to store the path to the data element. (ii) A *metadata* is a piece of information used to describe a data element. For instance, to record the access information of a data element, the metadata element *accessing user* can be used. This information is stored in the attribute *connectionContext.*(iii) A *property* is a key-value pairs to indicate the value of a metadata, for instance, "name: Hans Muller" can be used to link to the metadata *accessing user*.

The core model extensions defines more details about zone, granularity and categorization topics. (i) *The granularity extension* is used to describe data element granularity, such as table, row, column, header and field etc. (ii) *The zone extension* is used to indicate data maturity, such as transient loading zone, trusted zone, refined zone, sandbox and raw zone. (iii) *The categorization extension* is indicated metadata category (technical, business and operational).

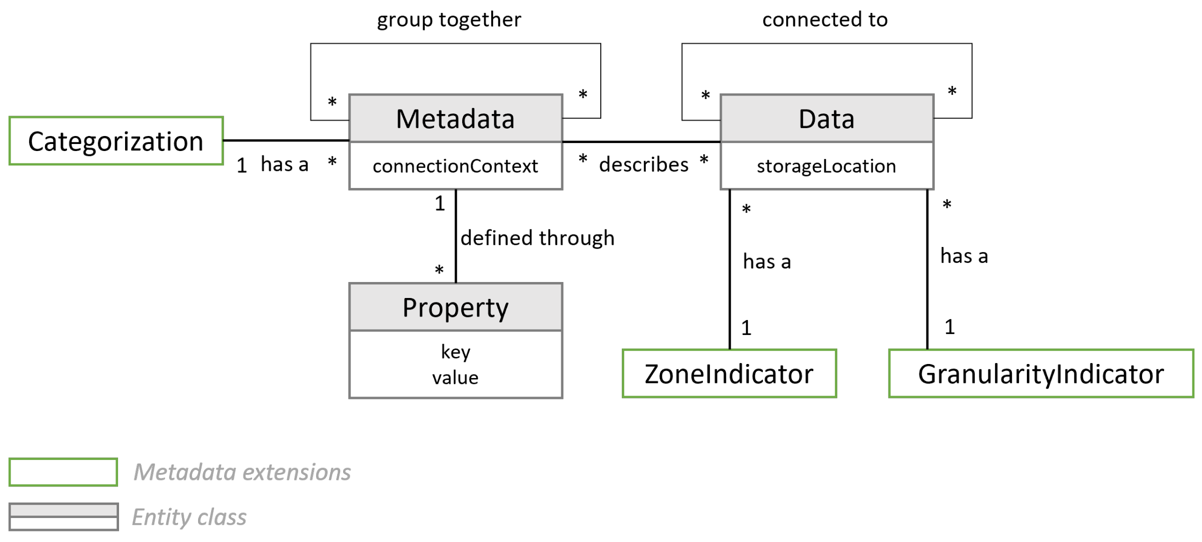


Figure 2.15: Metadata model of HANDLE

Metadata management system

HANDLE is not yet implemented as a complete metadata management system, but a graph database (Neo4j) is implemented as the metadata storage.

2.3.2.10 goldMEDAL

To propose a generic metadata model and to enable the data lineage tracing with the concept of process, Scholly et al. (2021) introduced a revision of MEDAL model. The model is compared with different state-of-the-art metadata models and is implemented for a textual and tabular data lake (AUDAL).

Metadata

The goldMEDAL metadata model is presented in Fig. 2.16. Comparing to its previous version MEDAL, goldMEDAL (i) uses *data entity* to generalize raw data, representations and versions, a data entity can be a dataset of any type; (ii) adds the concept *process* to record updates, transformations and parenthood relationship information for data lineage tracking; (iii) uses *link* to replace *similarity links*, a link can associate different data entities or data groups; and (iv) remains the concept of *grouping* to ensure multiple data granularity levels.

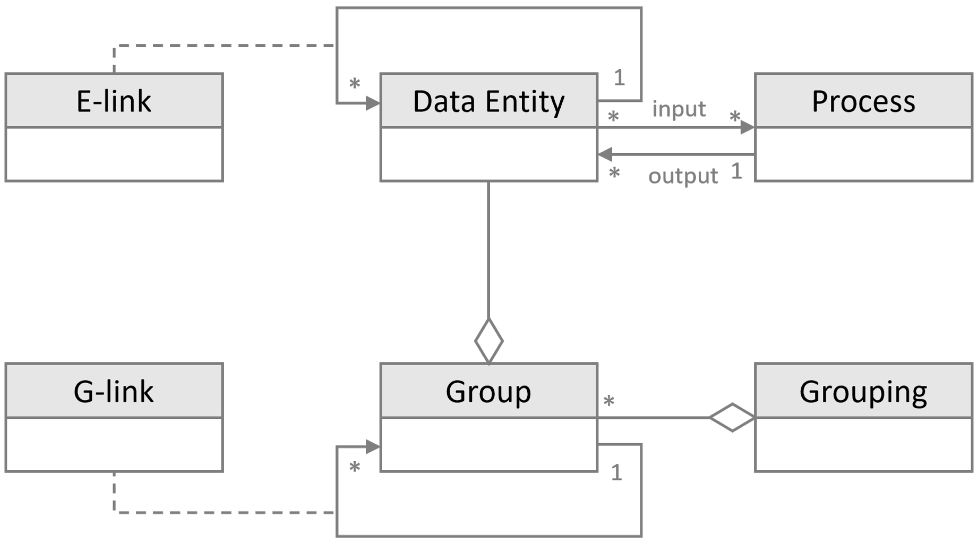


Figure 2.16: Metadata model of goldMEDAL

Metadata management system

To validate the proposition, goldMEDAL metadata model is applied into three different data lake systems. The first one is HOUDAL (Public Housing Data Lake). in this data lake, the metadata are stored in a graph database (Neo4j) and can be interacted with Neo4j Cypher queries. The second one is AUDAL which is a textual and tabular data lake. In this data lake, documents are stored in a document-oriented database (MongoDB) and their metadata are stored in a Neo4j database which can be accessed with Cypher queries. The third one is the Archaeological Data Lake. In this data lake, a metadata management system is integrated and is implemented by using the Apache Atlas framework.

2.3.3 Global analysis

All the works that we introduced in the section 2.3.2 proposed their metadata solution for data lakes with different aspects of information. Our global analysis is divided into two parts: a first part where we compare the two components of the approaches, namely the metadata model and the metadata management system, a second part concerns a FAIR evaluation.

Metadata Management Approach

All the solutions introduced different categories of useful metadata except Eichler et al. (2020) who proposed only a model of metadata without detailing used metadata (see table 2.3).

	Metadata model	**System of metadata management (MMS)**
(Walker and Alrehamy, 2015)	JSON format with different properties	a metadata system with searching interface
(Hai et al., 2016)	no model	a metadata system with searching interface
(Quix et al., 2016)	UML class diagram without attribute	a metadata system with searching interface
(Halevy et al., 2016a,b)	no model	a metadata system that generates and manages metadata
(Suriarachchi and Plale, 2016b)	no model	no system
(Maccioni and Torlone, 2018)	no model	no system
(Diamantini et al., 2018)	UML class diagram without attribute	no system
(Sawadogo et al., 2019)	no model	no system
(Eichler et al., 2020)	UML class diagram without attribute	no system
(Scholly et al., 2021)	UML class diagram without attribute	Apache Atlas metadata system for Archaeological data lake

Table 2.3: Global Analysis

Regarding **metadata model**, a few approaches formalized metadata in a standard model. Walker and Alrehamy (2015) introduced that, in their data lake, dataset metadata are firstly serialized in a JSON object and they described the sections and properties of the JSON object. Quix et al. (2016); Diamantini et al. (2018); Eichler et al. (2020); Scholly et al. (2021) introduced their metadata solutions with formalized UML metadata models. However, these metadata models are presented only with entity classes and their relationships, none of the metadata details (attributes of classes) are introduced.

Regarding **metadata management systems**, Walker and Alrehamy (2015); Hai et al. (2016); Quix et al. (2016); Halevy et al. (2016a,b); Scholly et al. (2021) presented their solutions that can extract, transform and store metadata. Among them, the metadata management systems of Walker and Alrehamy (2015), Scholly et al. (2021) and Hai et al. (2016) contain interfaces that allow users to search stored metadata.

FAIR Analysis

We evaluate the presented approaches according to the FAIR principles (gof, 2016). FAIR principles concern the guidelines to improve the findability, accessibility, interoperability and reuse of internet digital assets. The FAIR principles are originally proposed for scientific data through implementation network. We rely on the FAIR principles in the specific context of data lakes so some definitions was simplified (marked with asterisk) in the following list:

- **Findable**
  F1. (Meta)data are assigned a globally unique and persistent identifier
  *F2. Data are described with rich metadata (dataset, process and analysis metadata) **
  F3. Metadata clearly and explicitly include the identifier of the data they describe
  F4. (Meta)data are registered or indexed in a searchable resource

- **Accessible**
  *A1. (Meta)data are retrievable in the data lake system **
  A2. Metadata are accessible, even when the data are no longer available

- **Interoperable**
  *I1. Metadata use a formal, accessible, shared representation **
  *I2. Metadata use standard vocabularies **
  I3. (Meta)data include qualified references to other (meta)data

- **Reusable**
  *R1. (Meta)data are associated with detailed provenance **

We summarized the presented approaches and checked if they respect the FAIR principles in table 2.4.

Regarding the **findability**, all of the approaches emphasize that their proposed metadata can help users to search datasets stored in a data lake. However, most of them do not indicate that each stored element is assigned a globally unique identifier and this information is stored as the element metadata. Moreover, none of the approaches contain metadata of all datasets, transformation processes and analyses.

Regarding **accessibility**, the approaches of Walker and Alrehamy (2015), Hai et al. (2016), Quix et al. (2016), Halevy et al. (2016a,b) and Scholly et al. (2021) present metadata systems that allows users to search and access datasets through interfaces. None of these authors mentioned metadata in their approaches, metadata are always available even if the linked data are no longer available.

Regarding **interoperability**, the authors of all of the approaches explained their metadata solution with or without a formalized model. With all of the approaches, different datasets can be linked with common metadata (e.g. a common tag) and/or provenance metadata. However, only Hai et al. (2016), Quix et al. (2016), Halevy et al. (2016a,b), Diamantini et al. (2018) and Sawadogo et al. (2019) mentioned that datasets can have semantic annotations (e.g. ontology).

	Findable				Accessible		Interoperable			Reusable
	F1	F2	F3	F4	A1	A2	I1	I2	I3	R1
(Walker and Alrehamy, 2015)	✓	-	✓	✓	✓	?	✓	-	✓	-
(Hai et al., 2016)	?	-	?	✓	✓	?	✓	✓	✓	-
(Quix et al., 2016)	?	-	?	✓	✓	?	✓	✓	✓	-
(Halevy et al., 2016a,b)	✓	-	✓	✓	✓	?	✓	✓	✓	✓
(Suriarachchi and Plale, 2016b)	?	-	?	✓	-	?	✓	-	✓	✓
(Maccioni and Torlone, 2018)	?	-	?	✓	-	?	✓	-	✓	-
(Diamantini et al., 2018)	?	-	?	✓	-	?	✓	✓	✓	-
(Sawadogo et al., 2019)	?	-	?	✓	-	?	✓	✓	✓	✓
(Eichler et al., 2020)	?	-	?	✓	-	?	✓	-	✓	-
(Scholly et al., 2021)	?	-	?	✓	✓	?	✓	-	✓	✓

✓: *mentioned or explained with details*
? : *not mentioned*
- : *not considered in the approach*

Table 2.4: FAIR analysis

Regarding **reusability**, Halevy et al. (2016a,b), Suriarachchi and Plale (2016b), Sawadogo et al. (2019) and Scholly et al. (2021) introduced process provenance metadata to trace the data life-cycle.

2.3.4 Analysis of metadata models

Different authors introduced their metadata models that include information of datasets and processes. To clarify the comparison of different metadata models on different aspects, we firstly propose a classification of dataset metadata. According to the characteristics, we propose to classify them into three categories:

- **Intra-metadata** that concern the information of each single dataset, including: (i) *Dataset properties* that are dataset basic information, such as name, size, creation date, etc.. (ii) *Schematic metadata* that are the information used to describe dataset structure, such as, table and attribute information of relational datasets, entity class and property information of semi-structured datasets. (iii) *Basic semantic metadata* that are the basic information describing dataset meaning, such as tags, descriptions. Intra-metadata are tied to single datasets, to ensure that there are intra-metadata stored in a metadata management system, the data lake should have at least one dataset. Note that even different datasets can be linked with their common intra-metadata, these metadata should not be treated as inter-metadata. For instance, two datasets have the same tag "health", this information is always an intra-metadata because one dataset and its metadata are deleted, the tag still exists for another dataset.

- **Inter-metadata** that concern the relationships among different datasets, such as containment (e.g. a relational database instance contains different users/schemas), similarity/dissimilarity between datasets or user indicated relationship. An inter-metadata can be added only when there are at least two datasets.

- **Global-metadata** that concern advanced semantic annotations such as ontologies. The global metadata are independent from any datasets, they can exist in a data lake when there is no dataset.

In table 2.5, we summarized the ingestion mode, ingested dataset type and generated metadata type of different metadata solutions.

2.3.4.1 Dataset Ingestion

Regarding the data ingestion or process, all the solutions can deal with the batch mode, only Suriarachchi and Plale (2016b) indicated that they can work with real-time data transformation processes.

Regarding the ingested or processed dataset type, Quix et al. (2016) only works with structured datasets; Hai et al. (2016) and Maccioni and Torlone (2018) can only deal with structured and semi-structured datasets; the other solutions can adapt to all types of datasets (structured, semi-structured and unstructured).

2.3.4.2 Metadata type

Although data lakes allow users to ingest, process and analyze different types of data, we observe that the majority of metadata solutions focus only on dataset metadata or data catalog. Few of them consider the metadata of transformation processes and none of them studies the metadata of performed analyses.

Regarding **dataset metadata**, most of the authors explained different types of metadata used in their models except Suriarachchi and Plale (2016b) who worked on data processes and Eichler et al. (2020) who only proposed a model of metadata without introducing the metadata that can be used in their model.

Among authors who proposed dataset metadata, all of them considered intra-metadata with more or less attributes. Hai et al. (2016) and Diamantini et al. (2018) introduced schematic and basic semantic metadata categories without describing dataset basic properties. Scholly et al. (2021) introduced dataset properties and basic semantic metadata. In their model, different granularity of data and the links among them may implicitly present dataset schema, but since the author mainly focus on unstructured datasets, they did not emphasize schematic metadata. Walker and Alrehamy (2015), Quix et al. (2016), Halevy et al. (2016a) and Maccioni and Torlone (2018) presented their dataset intra-metadata with properties, schematic and basic semantic metadata. However, Walker and Alrehamy (2015) only listed 5 metadata and the other authors only introduced the three categories of metadata with examples, none of the work listed in details all the possible metadata. Sawadogo et al. (2019) proposed presentations and versions metadata besides the three categories. However, the authors need to explain with more details their solution in which data (e.g. a dataset presentation) are stored as metadata (*presentations*). For instance, regarding *representations* and *parenthood relationship*, when a dataset A is transformed to dataset B, the B should be *representation metadata* of A or a new *dataset* or *both*? If B is *representation metadata*, why choose to use *parenthood relationship* to link a dataset and its metadata? If B is a new dataset, are *representation metadata* still needed? If B is both, what are the difference between them?

Inter-metadata of datasets are considered by less authors. GOODS (Halevy et al., 2016a,b) uses four types of relationships of datasets: containment, provenance, logical

	Ingestion		Metadata type				
	Mode	Dataset type	Datasets			Processes	Analyses
			intra	inter	global		
(Walker and Alrehamy, 2015)	batch	structured, semi-structured, unstructured	properties, schematic, basic semantic				
(Hai et al., 2016)	batch	structured, semi-structured	schematic, basic semantic		semantic annotations		
(Quix et al., 2016)	batch	structured	properties, schematic, basic semantic		semantic annotations		
(Halevy et al., 2016a,b)	batch	structured, semi-structured, unstructured	properties, schematic, basic semantic	dataset relationship	semantic annotations	process provenance	
(Suriarachchi and Plale, 2016b)	batch, real-time	structured, semi-structured, unstructured				process provenance	
(Maccioni and Torlone, 2018)	batch	structured, semi-structured	properties, schematic, basic semantic	dataset relationship			
(Diamantini et al., 2018)	batch	structured, semi-structured, unstructured	schematic, basic semantic	dataset relationship	lemma metadata		
(Sawadogo et al., 2019)	batch	structured, semi-structured, unstructured	properties, basic semantic, presentations, versions	dataset relationship	semantic annotations, indexes, logs	parenthood relationship	
(Eichler et al., 2020)	batch	structured, semi-structured, unstructured					
(Scholly et al., 2021)	batch	structured, semi-structured, unstructured	properties, basic semantic	link		process provenance	

Table 2.5: Context analysis of metadata models

clusters and content similarity. KAYAK (Maccioni and Torlone, 2018) calculates joinability and affinity between datasets. The solution of (Diamantini et al., 2018) calculate dataset similarity. MEDAL (Sawadogo et al., 2019) considers objects grouping and similarity links. GoldMEDAL(Scholly et al., 2021) uses *grouping* metadata to cluster datasets.

Global-metadata are also proposed by different solutions. Hai et al. (2016), Quix et al. (2016) and Halevy et al. (2016a,b) considered to use semantic annotations (e.g. ontology) to create global links among datasets. Diamantini et al. (2018) proposed to use lemma metadata in their semantic models of datasets. Sawadogo et al. (2019) considered, besides semantic annotations, dataset indexes and data lake logs as global metadata too. However, in our metadata classification, dataset index and access/process logs are intra-metadata because they are tied to specific datasets.

Regarding **process metadata**, Halevy et al. (2016a,b), Suriarachchi and Plale (2016b), Sawadogo et al. (2019) and Scholly et al. (2021) considered process provenance metadata to trace data life-cycle. Among them, only Scholly et al. (2021) mentioned that transformation script can be stored as metadata. Nevertheless, the proposed process metadata are limited. Other process information, for instance, process execution environment and logs, is not considered. Users do not have enough metadata to find and reuse transformation processes easily. We will also discuss these solutions and other preparation metadata approaches that are not in the data lake domain in section 4.5.

Regarding **analysis metadata** in data lakes, to the best of our knowledge, there is no approach in the literature that deals with this aspect. We will discuss other analysis metadata approaches that are not in the data lake domain in section 5.4.

2.3.5 Analysis of metadata management systems

Different metadata management system for data lakes are implemented to validate the related metadata models and to facilitate the metadata generation and searching for users. To compare the different solutions, we summarized them in four features: metadata generation, metadata storage, interface of exploration and metadata querying (see table 2.6).

Regarding **metadata extraction**, all the authors explained how to extract metadata from different types of data sources. Among their solutions, most of the authors worked on the automatic extraction of metadata from structured and semi-structured datasets. Scholly et al. (2021) and Liu et al. (2021) focused on the metadata extraction from unstructured data sources.

Regarding the **metadata storage system**, all of the users chose to use NoSQL systems. Walker and Alrehamy (2015) chose to use a graph database because it focus on relationships between entities and it is optimized for processing of dense interrelated datasets. Quix et al. (2016) chose to use a document-oriented database adapted to their metadata model. Halevy et al. (2016a) chose to use Bigtable, a key-value database, because it offers per-row transactions consistency and it fits the most their GOODS system. Scholly et al. (2021); Liu et al. (2021) chose to stored metadata in HBase which is integrated in Apache Atlas.

Regarding **interfaces of the systems**, Hai et al. (2016) showed a hierarchical searching interface and Scholly et al. (2021); Liu et al. (2021) uses the Apache Atlas interface.

Regarding **metadata querying**, Hai et al. (2016), Halevy et al. (2016a,b) and Scholly et al. (2021); Liu et al. (2021) provide predefined searching functions for users so that users who do not know the metadata querying language can explore the data lakes. Walker and Alrehamy (2015), (Hai et al., 2016) and Quix et al. (2016) offered free query function of deep and complicated search for users who can write queries with the metadata system querying language.

		(Walker and Alrehamy, 2015)	**(Hai et al., 2016)**	**(Quix et al., 2016)**	**(Halevy et al., 2016a,b)**	**(Scholly et al., 2021; Liu et al., 2021)**
Generation of metadata	**Extraction**	Automatic for structured and semi-structured data, manual for unstructured data	Automatic for structured and semi-structured data	Automatic for semi-structured data	Automatic for structured and semi-structured data	Automatic for unstructured data
	Metadata storage system	Graph database (Neo4j)	Not mentioned	Document-oriented database (MongoDB)	Key–value database (Bigtable)	Distributed database (HBase)
Exploratior of metadata	**Interface**	Mentioned but not shown	Ergonomic	Mentioned but not shown	Mentioned but not shown	Apache Atlas interface
	Metadata querying	Ad-hoc query	Predefined and ad-hoc query	Limited query	Predefined	Predefined

Table 2.6: Analysis of metadata management systems

2.4 Conclusion

In this chapter we presented the three main concepts defining the scope of our Ph.D proposals. The concepts are: data lake, data lake architecture and metadata management.

Regarding data lake, we compared different related work and we introduced our proper definition of data lake. In our definition, data lake is a big data analytics solution that allows different types of users to ingest, process and access different structural types of data and guarantees data quality, data security and data life cycle. This definition is validated by our publication (Ravat and Zhao, 2019a).

Regarding data lake architecture, an overview of the main architectures in industrial and academic fields were presented. Our proper architecture was also presented which contains four essential zones: ingestion, preparation, analysis and governance. This architecture is validated by our publication (Ravat and Zhao, 2019b).

Regarding metadata management which is the core of data lake governance, we analyzed different solutions in different aspects. Firstly, we defined the concepts of metadata, metadata management as well as metadata management system. Secondly, we presented the key works of data lake metadata management in the literature, in particular, we introduced their metadata design and implemented metadata management system. Finally, we conducted initial cross-cutting discussions of these works from different aspects: global, metadata model and metadata management system.

After comparing existing solutions, we see the current metadata management are limited. There is no formalized and detailed metadata model that can be applied to different types of datasets (structured, semi structured and unstructured) and different phases of data life-cycle (ingestion, process and analysis). Moreover, there is no system of metadata management that respects FAIR principles to help users to find, access, interoperate and reuse different elements stored in the data lake with ergonomic interfaces. To remedy these deficiencies, in chapters 3, 4 and 5, we will introduce our own proposals and discuss in more details the positioning of the approaches specifically on each part of the data life cycle (ingestion, processing and analysis).

Chapter 3

Metadata on Data Ingestion

Contents

3.1 Introduction

Data integration is the phase during which data are moved from one or more sources to a destination where data are stored and can be accessed by users (Lenzerini, 2002; Meehan et al., 2017). Today, data integration is commonly discussed under the name of (i) Extract, Transform and Load (ETL) for data warehousing (Bansal and Kagemann, 2015; Bansal, 2014) and (ii) ingestion for data lake.

ETL of data warehousing has objective of providing an access of integrated and manageable data for decision-making (Simitsis et al., 2009; Vassiliadis, 2009) by the following steps:

- Extract which is responsible for accessing to different sources and extracting part of the source data that are defined as necessary for decision making during the conception phase.
- Transform which is responsible for converting the extracted data into coherent and reliable data for decision making. This phase is also called data consolidation which is based on data sorting, cleaning, standardization and calculation.
- Load which is responsible for inserting the transformed data into a permanent storage system such as a data warehouse.

Although ETL tools are well developed for data warehousing, it is not adapted to the big data era. Big data have the characteristics of 3Vs: (i) Various means that the ingested data are various in data sources (databases, IoT objects, files, etc), data modes (real-time and batch) and data structural types (structured, semi-structured, unstructured). (ii) Velocity means that the speed of data creation can be high. And (iii) Volume means that a great amount of data can be ingested. However, ETL, characterized by "schema on write", integrates data to a predefined schema which can not adapt to various data and the cost of data warehouse can grow exponentially for a greater volume, a better performance or demanded changes. Therefore, a more flexible solution "schema on read" of data ingestion is proposed for data lakes.

Regarding data lakes, data ingestion concerns only extraction and loading; data are ingested without transformation and stored in their native format (Dixon, 2010; Fang, 2015). However, data ingestion can not just be a copy-paste process. Data with a large variety, volume and velocity increase the difficulty of finding and reusing datasets. On the contrary, all the ingested data are expected to be found, accessed, interacted with and reused by different types of users (data analysts, data engineers ans data scientists) who do not have the same needs in the use of data. In addition, trust and confidence are also excepted from users for data exploitation.

Therefore, to ensure the finding and reusability of data in a data lake, during the ingesting phase, not only data, but also their metadata should both be extracted and stored.

Different authors proposed data lake metadata management solutions to ensure the data ingestion phase. Nevertheless, current researches only partially respond to this problem. Most of the works that are dedicated to ingestion processes are not generic and focus on a specific aspect. Some works focus only on one type of data (structured (Halevy et al., 2016a), semi-structured (Alserafi et al., 2016) or unstructured (Sawadogo et al., 2019)), an ingestion mode (batch (Alserafi et al., 2016; Sawadogo et al., 2019) or real-time (Gupta and Giri, 2018)) or a specific domain (Walker and Alrehamy, 2015). Some works address metadata partially (Halevy et al., 2016a; Alserafi et al., 2016). Some works deal with metadata but do not propose a formalized model to organize them (Quix et al., 2016). To the best of our knowledge, there is no solution that formalizes the ingestion phase of data lake with a metadata model dedicated to this phase.

With the aim of facilitating the search and exploitation of all the ingested data in a data lake, we propose a data lake ingestion solution. The solution is twofold: (i) an exhaustive metadata model related to the data ingestion that is introduced in section 3.2, (ii) a formalized process of data ingestion including the generation of all the relative metadata that is introduced in section 3.3. We also compare our solution with the state of the art in section 3.4 and we conclude the chapter in section 3.5.

3.2 Metadata of Data Ingestion

To ensure the findability and reuse of ingested data by different users (data analysts, data scientists, data engineers), the metadata applied on data ingestion (see Fig. 3.1) should include as much information as possible and should be comprehended by experts who work in different areas.

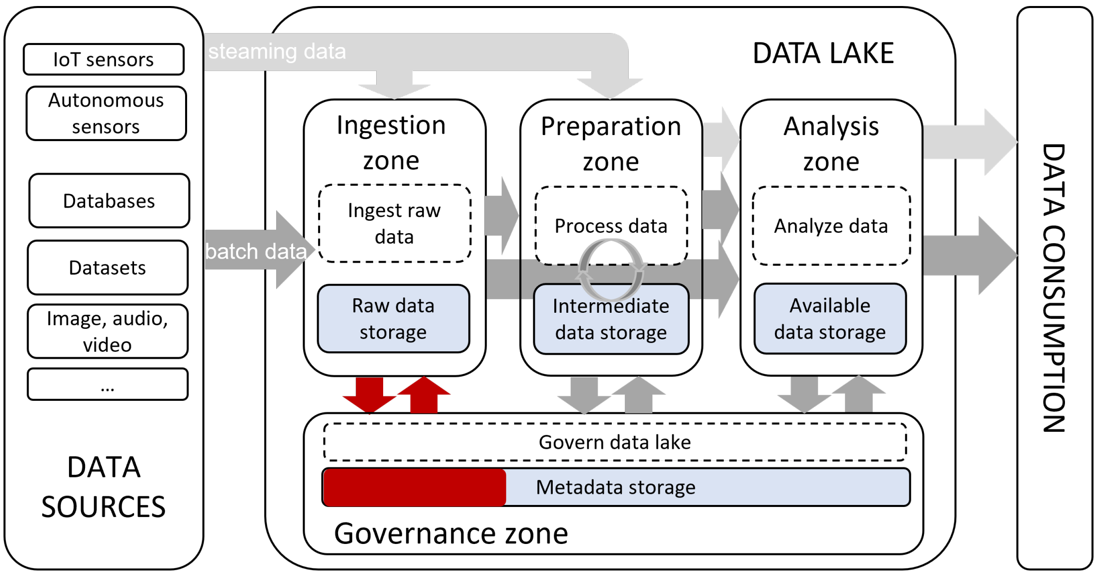


Figure 3.1: Data ingestion in a data lake

To include different aspects of information which can help users to comprehend how data are ingested and to achieve a methodical thinking, we chose to use the 5W1H method (Shimazu et al., 2006) (abbreviation summarizing what? who? where? when? why? how?) . Note that for data ingestion in data lakes, the question "why" is not answered for the reason that data are ingested into a data lake in their native format without business reasons.

The questions concerning one data ingestion activity are:

- What are the external data sources? What is the data ingestion activity? What dataset is ingested? What is the ingested dataset quality? What is the ingested dataset security level? What are the relationships between different ingested datasets?
- Who owns the source data? Who ingested the data? Who is in charge of the ingested dataset?

- Where are the ingested dataset stored? Where is the data ingestion code stored? Where is the ingested dataset stored?
- When are the source data created? When did the ingestion start/end ? When is datasets created in the data lake?
- How are the data ingested?

The above questions can be classified into four categories: questions of the data lake common knowledge, external data source, data ingestion activity and ingested dataset (in our work, the word *dataset* is used for all different types of collections of data, for instance, relational databases, spreadsheets, comma-separated values (CSV) files, text files, images, music files and videos, etc).

3.2.1 Metadata Model on Data Ingestion

To answer all the questions and to consider different types of data sources, datasets and ingestion modes, we propose a metadata model which includes the following categories of information (see Fig. 3.2):

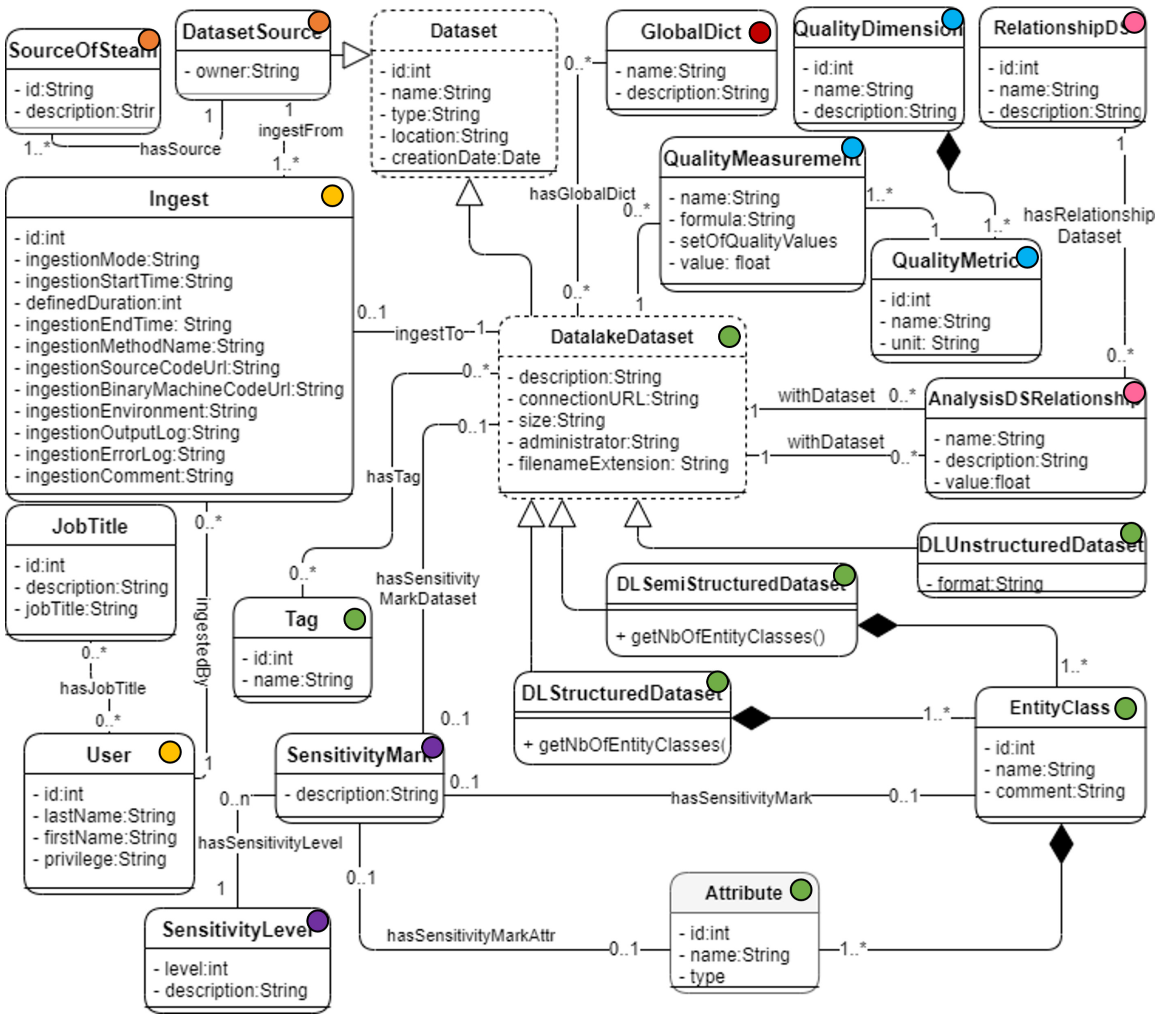


Figure 3.2: Metadata model of data ingestion

- With the aim of helping users to understand the common knowledge of the data lake and to facilitate data processing and analysis, we propose **global metadata** (class *GlobalDict*) (marked in red). The global metadata concern the knowledge that is not tied to a specific element in the data lake, but they include:

  - **Word knowledge**, as in a dictionary, thesauri and thesaurus, etc. For instance, in a hospital data lake, the organization structure information (relationships between hospital sites, departments, services and teams etc.) should be added in the thesauri, so that users can easily understand the structure regardless if they work on the electronic health record (EHR) database or invoicing database.
  - **Semantic annotations**, such as ontology.

- With the aim of answering all the questions concerning data source to help users to confirm the origin of ingested data, we propose the **external data sources metadata**. These metadata are regrouped in 2 classes: *DatasetSource* (subclass of *Dataset*) and *SourceOfStream* (marked in orange). *DatasetSource* will be generated for all types of data sources while *SourceOfStream* is for IoT (Internet of Things) data so that users can find the origin of the stream data and regroup data from the same source easily. The metadata include:
  - *name*, *type* (structured, semi-structured, unstructured) of *DatasetSource* and *description* of *SourceOfStream* answer the question of "what".
  - *owner*, which is the organization who owns the data source, to answer the question of "who".
  - *location* indicates where the data source is stored to answer the question of "where", this information can help users to find the root of data to give them more confidence.
  - *creationDate* to answer the question of "when".

- With the aim of answering all the questions concerning data ingestion activity to help users to understand how data are ingested and provide the possibility of reusing the ingesting process, we propose the **data ingestion process metadata** (class *Ingest* and its association relationships)(marked in yellow) including:
  - *ingestionMode* (batch, real-time), *ingestionComment* and relationships *ingestFrom*, *ingestTo* to answer the question of "what".
  - relationship *ingestedBy* that links to class *User* to answer the question of "who".
  - *ingestionSourceCodeUrl* and *ingestionBinaryMachineCodeUrl* to answer the question of "where".
  - *ingestionStartTime*, *definedDuration* and *ingestionEndTime* to answer the question of "when". For batch and real-time ingestion, the start time (*ingestionStartTime*) and end time (*ingestionEndTime*) is the real start and end time of an ingestion process, however, the duration (*definedDuration*) is only required in the case of real-time ingestion.
  - *ingestionMethodName*, *ingestionEnvironment*, *ingestionOutputLog* and *ingestionErrorLog* to answer the question of "how".

- With the aim of answering all the questions concerning ingested dataset to help users to have a global view of ingested data, we propose intra- and inter- metadata. Intra-metadata describe each single dataset, such as basic characteristics, security and quality information. while inter-metadata describe the relationships and common knowledge between different datasets:
  - The **ingested dataset characteristics metadata** have the objective of helping users to easily find, access and understand a dataset without opening it and checking its content by themselves. The classes *DatalakeDataset*, *EntityClass*, *Attibute* and *Tag* (marked in green) are involved and they include:
    - *name*, *format*, *filenameExtension*, *type* (structural type), *size* that are the basic information to answer the question of "what".
    - *description* and *tag* that are the semantic information to answer the question of "what".
    - lined *entityClass* and *attributes* that are the schematic information for (semi-) structured datasets to answer the question of "what".
    - *administrator* to answer the question of "who".
    - *location*, *connectionURL* to answer the question of "where".
    - *creationDate* to answer the question of "when".
  - The **quality metadata** have the objective of helping users to have more confidence on the ingested data by providing a first glimpse of data quality. The classes *QualityMeasurement*, *QualityDimension* and *QualityMetric* (marked in blue) are involved. The most commonly used five dimensions (accuracy, integrity, consistency, completeness and readability) are set by default in the model (Gyulgyulyan et al., 2019), and users can add more dimensions or metrics according to their needs.
  - The **security metadata** have the objective of protecting sensitive data. The classes *SensitivityMark*, *SensitivityLevel* (marked in purple) are involved. Sensitivity level is predefined by the administrator of data lake, for example, 0 to 5 and 0 indicate the least sensible level, 5 indicate the most sensible level. And for each dataset or entity class (table) or attribute, a sensible mask which indicates the sensible level can be added. For instance, in a hospital, the production database of the electronic health record (EHR) should be marked with level 5. If the production data are pseudonymized and stored in a dataset, this dataset can then be marked with a lower level. The sensitivity information can help data lake administrator to limit data accesses of users.
  - The **dataset relationship metadata** have the objective of helping users to find relevant datasets and enrich their analyses. The classes *RelationshipDS, AnalysisDSRelationship* (marked in pink) are involved. There are some predefined relationships that can be automatically detected by the system such as the similarity, correlation, containment and logical cluster. In addition, users can define relationships by themselves, for instance, they indicate overlapping data or common objects in two different datasets. Moreover, for the same type of relationship, users can choose different algorithms.

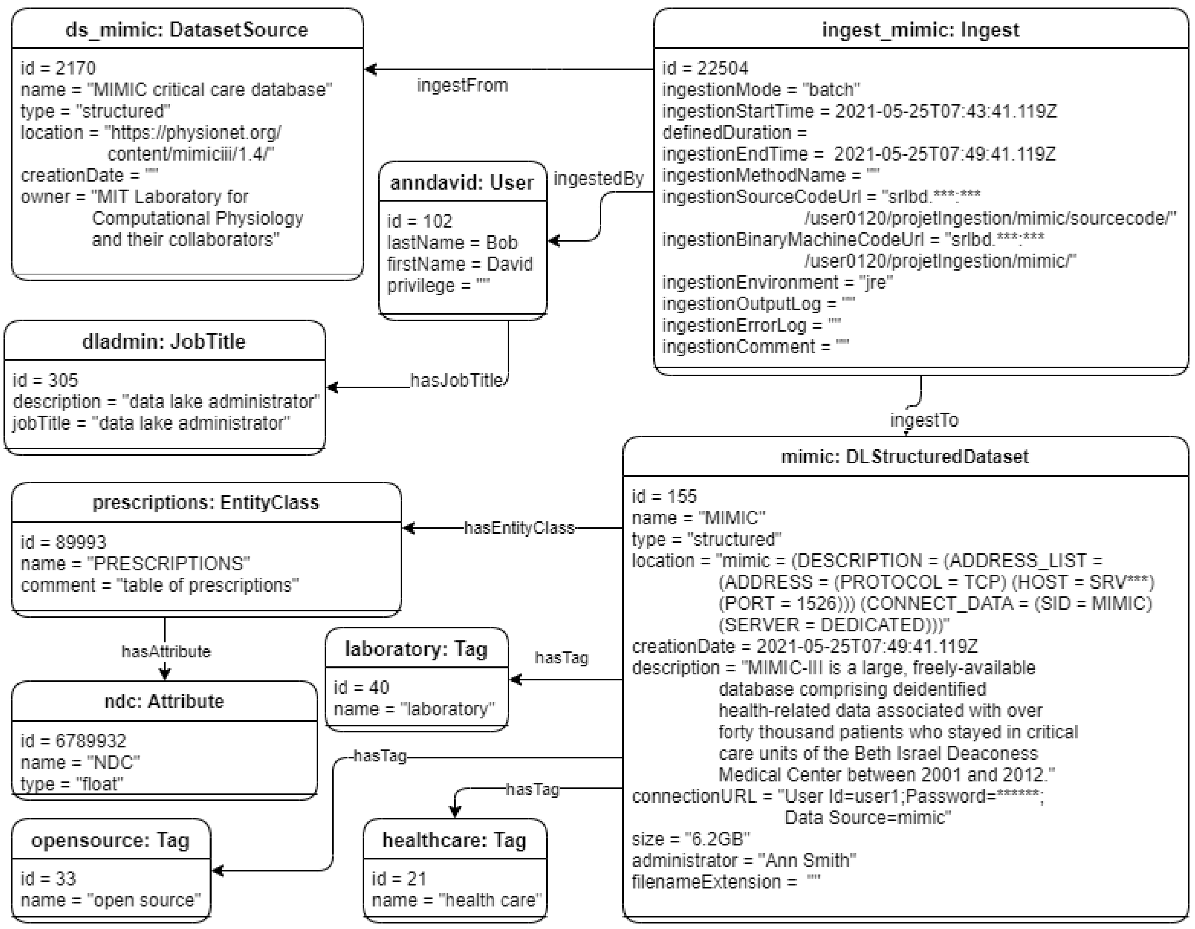
A metadata management system is developed which allows users to search datasets and consult the different categories of metadata easily (see a use case in section 6.4.1).

3.2.2 Example of Metadata Instantiation

To demonstrate our metadata model, we introduce two examples of data ingestion result with real datasets. The used datasets are of different types.

Dataset	Description	Type	Entities	Columns	Size
MIMIC	a freely accessible critical care database	relational database	40	534	6.2 GB
CHSI cancer	health indicators for the US counties	csv file	11	587	15 MB

Table 3.1: Ingested datasets



ds_mimic: DatasetSource
id = 2170
name = "MIMIC critical care database"
type = "structured"
location = "https://physionet.org/content/mimiciii/1.4/"
creationDate = ""
owner = "MIT Laboratory for Computational Physiology and their collaborators"

ingest_mimic: Ingest
id = 22504
ingestionMode = "batch"
ingestionStartTime = 2021-05-25T07:43:41.119Z
definedDuration =
ingestionEndTime = 2021-05-25T07:49:41.119Z
ingestionMethodName = ""
ingestionSourceCodeUrl = "srlbd.***:*** /user0120/projetIngestion/mimic/sourcecode/"
ingestionBinaryMachineCodeUrl = "srlbd.***:*** /user0120/projetIngestion/mimic/"
ingestionEnvironment = "jre"
ingestionOutputLog = ""
ingestionErrorLog = ""
ingestionComment = ""

anndavid: User
id = 102
lastName = Bob
firstName = David
privilege = ""

dladmin: JobTitle
id = 305
description = "data lake administrator"
jobTitle = "data lake administrator"

prescriptions: EntityClass
id = 89993
name = "PRESCRIPTIONS"
comment = "table of prescriptions"

ndc: Attribute
id = 6789932
name = "NDC"
type = "float"

laboratory: Tag
id = 40
name = "laboratory"

opensource: Tag
id = 33
name = "open source"

healthcare: Tag
id = 21
name = "health care"

mimic: DLStructuredDataset
id = 155
name = "MIMIC"
type = "structured"
location = "mimic = (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP) (HOST = SRV***) (PORT = 1526))) (CONNECT_DATA = (SID = MIMIC) (SERVER = DEDICATED)))"
creationDate = 2021-05-25T07:49:41.119Z
description = "MIMIC-III is a large, freely-available database comprising deidentified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012."
connectionURL = "User Id=user1;Password=******; Data Source=mimic"
size = "6.2GB"
administrator = "Ann Smith"
filenameExtension = ""

Figure 3.3: Object diagram of MIMIC dataset ingestion

MIMIC is a relational database of health care data[1]. The data lake administrator Bob David ingested the dataset on the 25 may 2021 into the data lake. He inputs the data source name (MIMIC critical care database), location (http://physionet.org), type (structured), owner (MIT) information and chose to ingest the dataset in the mode of batch, these information is used to instantiate the *DatasetSource* class. When the dataset is ingested in the data lake, firstly, the ingestion information, user (Bob) and the ingested dataset basic information (name, type, location, creation date, size, connection URL) is

[1]`https://mimic.mit.edu/`

stored in the metadata system. Then Bob added tags (health care, open source, laboratory) to the dataset, and chose to let the system detect the schematic information of the dataset. The system detected 40 tables and 535 attributes of the dataset in total and all the information is stored as metadata of the dataset. We present the object diagram of the ingested dataset in Fig. 3.3 (due to the page limit, we only present the metadata of one table and one attribute).

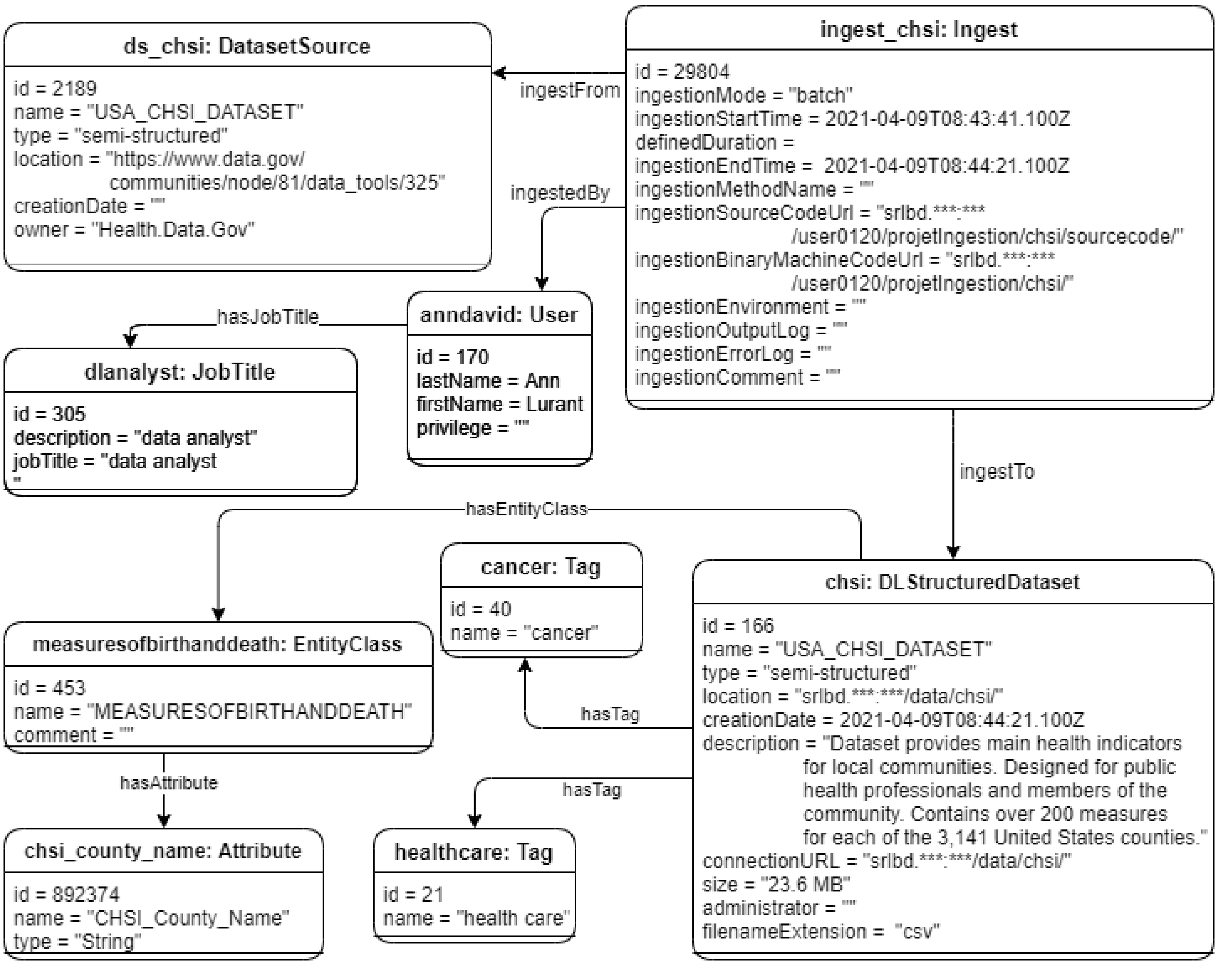

ds_chsi: DatasetSource
id = 2189
name = "USA_CHSI_DATASET"
type = "semi-structured"
location = "https://www.data.gov/communities/node/81/data_tools/325"
creationDate = ""
owner = "Health.Data.Gov"
ingest_chsi: Ingest
id = 29804
ingestionMode = "batch"
ingestionStartTime = 2021-04-09T08:43:41.100Z
definedDuration =
ingestionEndTime = 2021-04-09T08:44:21.100Z
ingestionMethodName = ""
ingestionSourceCodeUrl = "srlbd.***:***/user0120/projetIngestion/chsi/sourcecode/"
ingestionBinaryMachineCodeUrl = "srlbd.***:***/user0120/projetIngestion/chsi/"
ingestionEnvironment = ""
ingestionOutputLog = ""
ingestionErrorLog = ""
ingestionComment = ""
ingestFrom
ingestedBy
hasJobTitle
anndavid: User
id = 170
lastName = Ann
firstName = Lurant
privilege = ""
dlanalyst: JobTitle
id = 305
description = "data analyst"
jobTitle = "data analyst"
ingestTo
hasEntityClass
cancer: Tag
id = 40
name = "cancer"
chsi: DLStructuredDataset
id = 166
name = "USA_CHSI_DATASET"
type = "semi-structured"
location = "srlbd.***:***/data/chsi/"
creationDate = 2021-04-09T08:44:21.100Z
description = "Dataset provides main health indicators for local communities. Designed for public health professionals and members of the community. Contains over 200 measures for each of the 3,141 United States counties."
connectionURL = "srlbd.***:***/data/chsi/"
size = "23.6 MB"
administrator = ""
filenameExtension = "csv"
measuresofbirthanddeath: EntityClass
id = 453
name = "MEASURESOFBIRTHANDDEATH"
comment = ""
hasTag
hasTag
hasAttribute
chsi_county_name: Attribute
id = 892374
name = "CHSI_County_Name"
type = "String"
healthcare: Tag
id = 21
name = "health care"

Figure 3.4: Object diagram of CHSI dataset ingestion

CHSI cancer is a set of CSV files which contain health indicators for each of the 3,141 United States counties[2]. This dataset was ingested by Ann Rick the 25 may 2021 and is stored in the server *srlbd*. This dataset is ingested in the same way as MIMIC dataset, we present the object diagram of the ingestion metadata in Fig. 3.4 (due to the page limit, we only present the metadata of one table and one attribute of the dataset).

The presented metadata model is dedicated to data ingestion of data lakes. It is a complete model that is adapted to different structural types of datasets (structured, semi-structured and unstructured) and different ingesting modes (batch and real-time). Moreover, the model well describes all data sources, data ingestion activities and ingested datasets to help users search all elements concerning data ingestion. Especially for data lake datasets, we consider both inter- and intra- metadata. Inter-metadata help users find relevant datasets and enrich their analyses by providing relationships between datasets.

[2]`https://catalog.data.gov/dataset/community-health-status-indicators-chsi-to-combat-obesity-heart-disease-and-cancer`

Intra-metadata help users to understand each single dataset from different aspects: basic characteristics, semantic, schematic, sensitivity as well as quality information.

3.3 Ingestion Process and Generation of the Ingestion Metadata

The data ingestion process consists of not only ingesting datasets in the data lake but also instantiating the metadata schema.

Regarding the dataset ingestion, we propose a process for which we combine user actions and automatic processes done by the system to explain the interactions between users and the system. Normally, when ingesting a dataset, users can choose (i) to manually input all the information to ensure the metadata accuracy, (ii) to leave the system automatically detect all the metadata, or (iii) to let the system firstly detect the metadata then secondly ask users to validate the result so that quality is ensured. We chose the third solution for our data lake management system because it not only is convenient for users to balance the efficiency and accuracy but also ensures the sensitive data protection. Particularly in the healthcare domain, patient medical data need to be protected so that it is important to let users input or validate the sensitivity metadata to ensure the certainty of the data usage, such recommendations are mentioned in (Halevy et al., 2016b).

We formalize the data ingestion process by BPMN for the following reasons: (i) BPMN diagrams can represent complex processes with an intuitive language, (ii) BPMN diagrams can clearly define the scope of a process from the beginning to the end and (iii) non-experts can easily understand processes through the graphical language (El Akkaoui and Zimanyi, 2009; Oliveira and Belo, 2012). The last point is particularly important in a DL environment with different types of users. Moreover, we also use algorithms to precise instantiated metadata.

Regarding the generation of metadata, we introduce it through algorithms so as to clearly identify the complementarity between the metadata instantiation and metadata storage for each step of data ingestion.

The data ingestion in a data lake consists of two sub-processes (see Fig. 3.5):

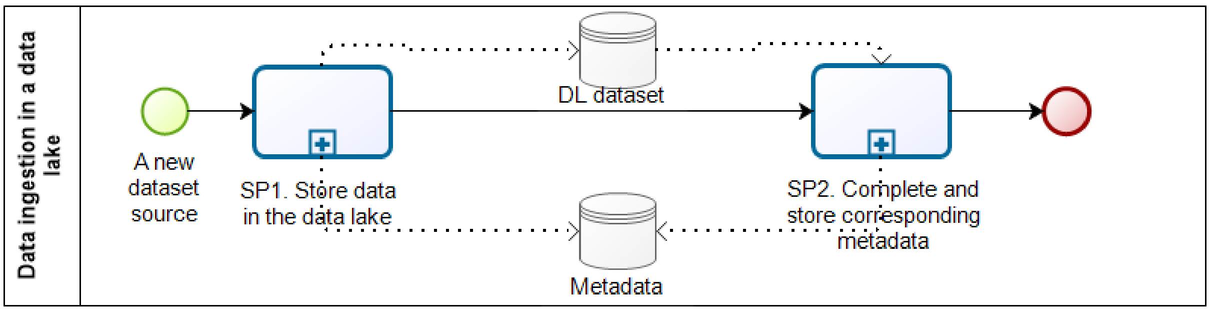

Data ingestion in a data lake
A new dataset source
SP1. Store data in the data lake
DL dataset
Metadata
SP2. Complete and store corresponding metadata

Figure 3.5: The data ingestion process

- *SP1:* The first sub-process concerns the storage of a new dataset. This phase permits to establish the data source connection, store the dataset in data lake and store the metadata of the data source, ingestion process and ingested dataset. In order to meet the needs of data lake users, we offer different ingestion modes and several alternatives of storage.

- *SP2:* In order to generate all the metadata in the proposed model, the objective of the second sub-process is to completely instantiate the metadata schema. The metadata include inter-metadata which concern the relationships among different datasets and intra-metadata which concern the semantic, schematic, quality and sensitivity metadata.

3.3.1 SP1. The process of Storing data in the data lake

The objective of this sub-process is to ingest source data into data lake and store metadata of the data source, ingestion activity as well as some metadata of the ingested datasets.

To do so, firstly, the source data connection needs to be established. Users need to input the data source name, location and/or connection information manually. Moreover, in the aim of making the data provenance checking easier, users can input data source creation date and owner. The data lake framework verifies the entered information by connecting the data source. Once the connection is established, the metadata of the data source are stored.

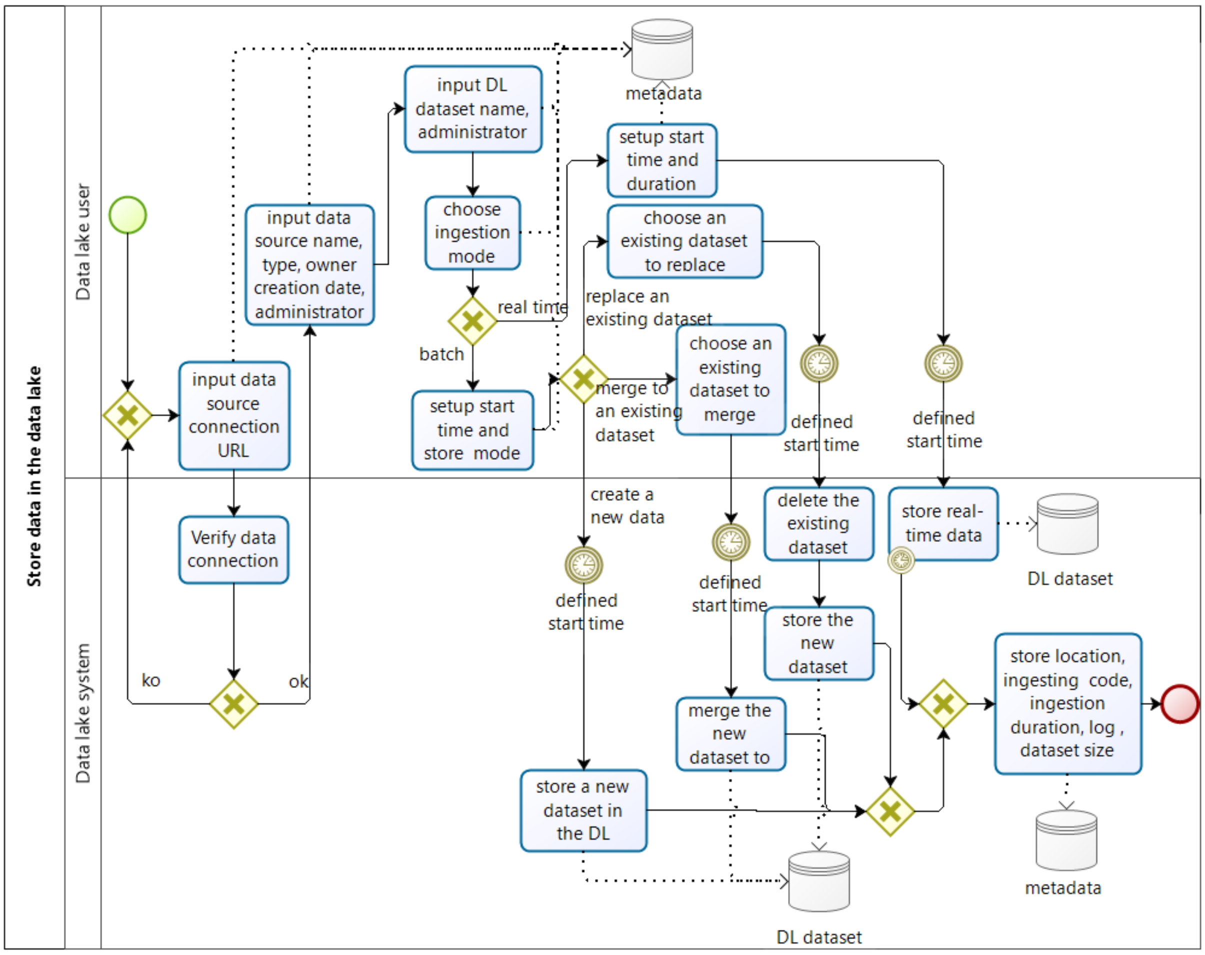

Store data in the data lake
Data lake user
Data lake system
input data source connection URL
Verify data connection
ko
ok
input data source name, type, owner creation date, administrator
input DL dataset name, administrator
choose ingestion mode
real time
batch
setup start time and store mode
metadata
setup start time and duration
choose an existing dataset to replace
replace an existing dataset
merge to an existing dataset
choose an existing dataset to merge
create a new data
defined start time
defined start time
defined start time
defined start time
delete the existing dataset
store the new dataset
merge the new dataset to
store a new dataset in the DL
store real-time data
DL dataset
store location, ingesting code, ingestion duration, log , dataset size
metadata
DL dataset

Figure 3.6: Store the dataset in the data lake

Secondly, according to the operation mode and data storage chosen by the user, dataset is stored in the right repository at the right moment. For this sub-process, as depicted in Fig. 3.6, we offer two operation modes: (i) real-time which concerns the ingestion of streaming data for a period and (ii) batch for which we proposed three alternatives for physical data storage in the data lake: create a new dataset, replace an existing dataset and merge new data to an existing dataset. During this step, data ingestion process information (class *Ingest*, *User*) and ingested data information (class *DatalakeDataset*) is generated and stored.

We also introduce Algo. 1 which includes two main functions of the process of metadata generation for this step. The first function is used to instantiate data source metadata during the connection phase. After the data connection, all the user input information of the data source are stored (*line 1-8*). When a dataset is ingested in the data lake, the ingestion process information and the ingested dataset basic information (location, name, creation time) are instantiated and stored (*line 9-13*).

Algorithm 1: Functions used to ingest datasets into a data lake

```
   /* establish source dataset connection                                    */
1  Function connectDataSource(dataSourceConnection, dataSourceType,
     dataSourceLocation, dataSourceName, dataSourceOwner, sourceOfStream,
     owner):
2      if tryConnectDataSource(dataSourceConnection, dataSourceLocation) then
3          datasetSource ← instantiateClass('DatasetSource',
             createProperties('name', dataSourceName, 'type', dataSourceType,
             'location', dataSourceLocation, 'owner', owner))
4          if sourceOfStream then
5              createRelation('DatasetSource-SourceOfStream', datasetSource,
                 sourceOfStream)
6          return datasetSource
7      else
8          return None
   /* ingest dataset                                                         */
9  Function ingestDataset(datasetSource, ingestionComment, sarteTime,
     duration):
10     ingestionProcess, ingestedDataset ← ingestDatasetInDL(datasetSource,
         ingestionMode, startTime, duration)
11     ingest ← instantiateClass('Ingest', getPropertiesIngest(ingestionProcess),
         'comment', ingestionComment)
12     createRelation('DatasetSource-Ingest', datasetSource, ingest)
13     datalakeDataset ← instantiateClass('DatalakeDataset',
         getPropertiesDatalakeDataset(ingestedDataset))
```

3.3.2 SP2. The process of completing and storing metadata

The objective of this phase is to facilitate the findability, accessibility and interoperability of ingested datasets for future analyses by recording more useful information. To achieve this objective, we need to complete and store not only the information of each single dataset but also the relationships and shared information between different datasets. Therefore, during this phase, we instantiate intra-metadata, inter-metadata as well as global-metadata (see Fig. 3.7).

The generation of intra-metadata includes three sub-processes:

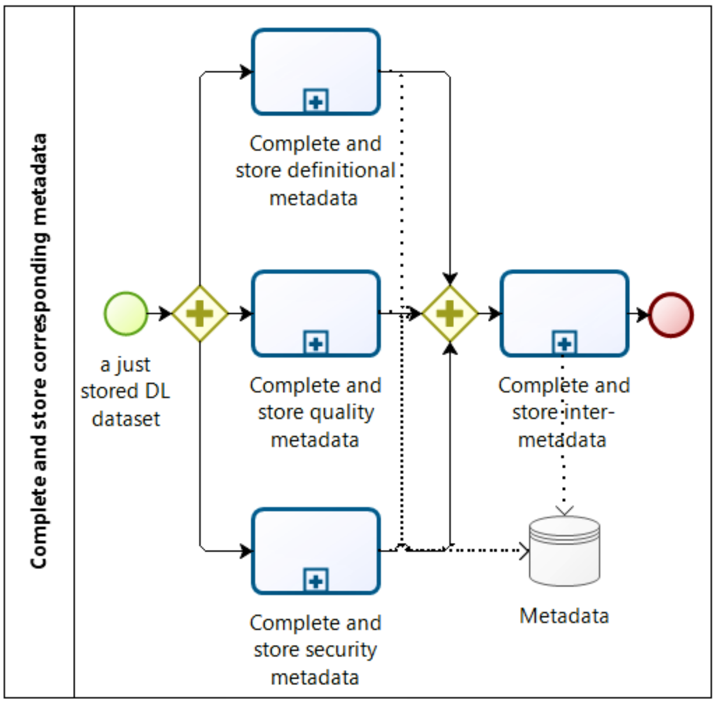

Complete and store corresponding metadata
a just stored DL dataset
Complete and store definitional metadata
Complete and store quality metadata
Complete and store security metadata
Complete and store inter-metadata
Metadata

Figure 3.7: Complete and store corresponding metadata

- The generation of definitional metadata (see Fig. 3.8) concerns storing semantic and schematic metadata. Semantic metadata consist a list of tags (class *Tag*) and a description (attribute *datalakeDataset.description*) linked a dataset. These metadata can be manually input by users. Schematic metadata of structured and semi-structured databases consist of the database model with the tables/entities (class *EntityClass*) and attributes (class *NumericAttribute* and *NominalAttibute*) information. For unstructured datasets, the data format is detected and stored.

  We precise the details of the generation of definitional metadata in Algo. 2. Regarding semantic metadata, users need to input a description and a set of tags linked to the dataset. The description is stored in the *DatalakeDataset* object as a propriety (*line 2*). For each of the input tags, firstly, if it exists already in the data lake, the system retrieves the existing tag, if not, the system creates an new tag object; secondly, the tag is linked to the dataset (*line 3-9*). Regarding schematic metadata, for a structured or semi-structured dataset, the system firstly detects automatically all its table/entity with attributes and store this information (*line 12-21*). Then for each predefined attribute relationship, the system calculates and stores the result value between every two attributes (*line 22-17*). For unstructured dataset, the system detects its format (image, video, text files etc.) and adds this information to the dataset (*line 28-29*).

- The generation of quality metadata is about collecting predefined measures (accuracy, integrity, consistency, completeness and readability) and user defined measures to adapt to different use-cases (see Fig. 3.9). This information is calculated according to the approaches of (Kwon et al., 2014; Cai and Zhu, 2015) then validated by users. During this step, information of classes *QualityMeasurement*, *QualityMetric* and *QualityDimension* is generated.

- The generation of security metadata consists of the security levels of different data granularity (dataset, table, column, etc) (see Fig. 3.10). The security metadata is not necessary, they can be manually input or not, the default value is zero which means that all users can access the dataset. However, for some special domains, for example, hospitals, to protect the personal data of patients, the sensitivity metadata are mandatory. During this step, information of classes *SensitivityMark* and *SensitivityLevel* is generated. We do not precise the metadata generation process because it is a manual entry.

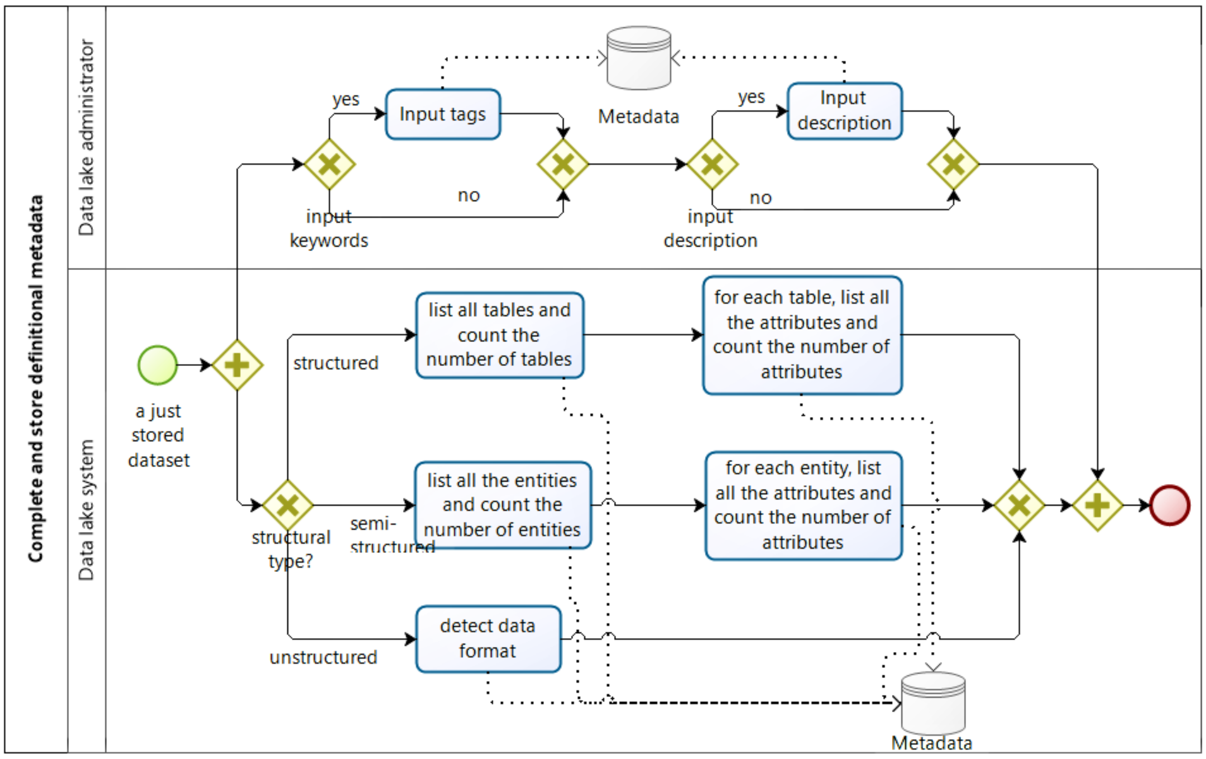

Complete and store definitional metadata
Data lake administrator
Data lake system
yes
Input tags
Metadata
yes
Input description
no
input keywords
no
input description
a just stored dataset
structured
list all tables and count the number of tables
for each table, list all the attributes and count the number of attributes
list all the entities and count the number of entities
for each entity, list all the attributes and count the number of attributes
semi-structured
structural type?
detect data format
unstructured
Metadata

Figure 3.8: Complete and store definitional metadata

Algorithm 2: Functions used to complete definitional metadata

```
   /* instantiating definitional metadata - semantic */
1  Function instantiatingSemanticMetadata(datalakeDataset,
    datalakeDatasetDesc, datalakeDatasetTags):
2      addProperties('DatalakeDataset', datalakeDataset,
        createProperties('description', datalakeDatasetDesc))
       /* store tags for datalakeDataset */
3      allTags[] ← getAllTags()
4      foreach t ⊂ datalakeDatasetTags do
5          if t in allTags[] then
6              tag ← getTag(t)
7          else
8              tag ← instantiateClass('Tag', createProperties('name', t))
9          createRelation('DatalakeDataset-Tag', DatalakeDataset, tag)
   /* instantiating definitional metadata - schematic */
10 Function instantiatingSchematicMetadata(datalakeDataset):
11     if (datalakeDataset.type = "structured" or "semi-structured") then
12         entities[] ← getEntityClasses(datalakeDataset)
13         foreach e ⊂ entities do
14             entityClassProperties ← createProperties('name', getEntityName(e))
15             entityClass ← instantiateClass('EntityClass', entityClassProperties)
16             createRelation('DatalakeDateset-EntityClass',
                datalakeDataset, entityClass)
17             atts[] ← getAttributes(e)
18             foreach att ⊂ atts[] do
19                 attributeProperties ← createProperties('name', getAttName(att),
                    'type', getAtttype(att))
20                 attribute ← instantiateClass('Attribute', attributeProperties)
21                 createRelation('EntityClass-Attribute', entityClass, attribute)
               /* For each predefined RelationshipAtt, calculate the value of
                  relationship between attributes */
22             analysisAttributes[] ← getAnalysisAttribute(atts[], rsAtts[])
23             foreach an ⊂ analysisAttributes[] do
24                 relationArr ← instantiateClass('AnalysisAttribute',
                    createProperties('value', an.value)
25                 createRelation('AnalysisAttribute-Attribute',
                    relationArr, an.attribute1)
26                 createRelation('AnalysisAttribute-Attribute',
                    relationArr, an.attribute2)
27                 createRelation('AnalysisAttribute-RelationshipAtt',
                    relationArr, an.relationshipAtt)
28     else
29         addProperties('DatalakeDataset', datalakeDataset,
            getDatasetFormat(datalakeDataset))
```

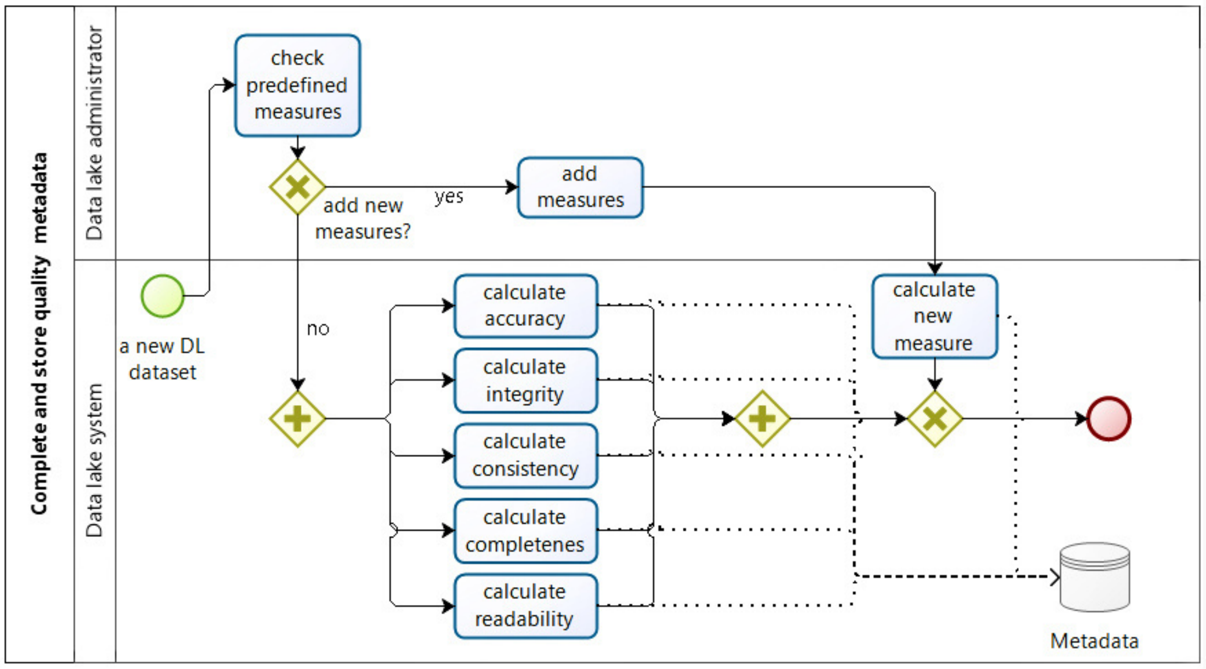


Figure 3.9: Complete and store quality metadata

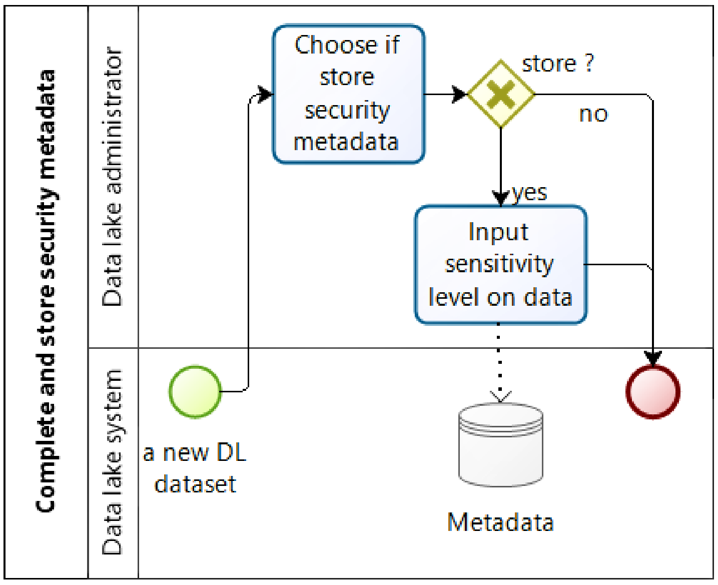

Complete and store security metadata
Data lake administrator
Data lake system
Choose if store security metadata
store ?
no
yes
Input sensitivity level on data
a new DL dataset
Metadata

Figure 3.10: Complete and store security metadata

Inter-metadata help users to link different datasets, which are important for data finding, reuse and interoperability in a data lake (see Algo. 3). The relationships among datasets (similarity, correlation, containment and logical cluster) can be automatically generated by the system (Halevy et al., 2016a; Quix et al., 2016; Suriarachchi and Plale, 2016a) (*line 1-7*) or input manually by users (*line 8-12*). Note that the generation of inter-metadata is not limited in this sub-process, for example, the provenance metadata can be instantiated during the data storage phase as we mentioned previously. Moreover, the relationships between different datasets are not limited in the inter-metadata, with some other intra-metadata attributes, for example, keywords and administrators, we can also find relationship between datasets.

Global-metadata help users to reduce the time on understanding datasets. The thesaurus is manually input or imported by users.

Algorithm 3: Functions used to complete inter metadata

```
/* for chosen dataset, calculate automatically relationships */
1 Function calculateRelationships($datalakeDataset$):
    /* For each predefined RelationshipDS we calculate the value of relationship
       between datasets */
2   $analysisDSRelationships[] \leftarrow$
      getAnalysisDSRelation($datalakeDataset, datalakeDatasets[], relationshipDSs[]$)

3   foreach $anDs \subset analysisDSRelationships[]$ do
4       $relationDs \leftarrow$ instantiateClass('AnalysisDSRelationship',
          createProperties('value', $anDs.value$))
5       createRelation('AnalysisDSRelationship-DatalakeDataset',
          $relationDs, anDs.datalakeDataset1$)
6       createRelation('AnalysisDSRelationship-DatalakeDataset',
          $relationDs, anDs.datalakeDataset2$)
7       createRelation('AnalysisDSRelationship-RelationshipDS',
          $relationDs, anDs.relationshipDS$)
/* for chosen datasets, users input a relationship manually */
8 Function inputRelationships($datalakeDataset1$, $datalakeDataset2$,
    $relationshiDS$, $dsRelationshipName$, $dsRelationshipdesc$,
    $dsRelationshipValue$):
9   $analysisDSRelationship \leftarrow$ instantiateClass('AnalysisDSRelationship', )
10  createRelation('AnalysisDSRelationship-DatalakeDataset',
      $relationDs, anDs.datalakeDataset1$)
11  createRelation('AnalysisDSRelationship-DatalakeDataset',
      $relationDs, anDs.datalakeDataset2$)
12  createRelation('AnalysisDSRelationship-RelationshipDS',
      $relationDs, anDs.relationshipDS$)
```

3.4 Discussion

We introduced our metadata solution of data ingestion in data lakes in the previous sections. Our solution consists of a metadata model as well as formalized processes of data ingestion and metadata generation.

Our metadata model includes all the information to answer the 5W1H questions that we listed in section 3.2. Nevertheless, to present the completeness of our metadata management and in order to discuss our proposal compared to the state of the art approaches, we summarize in table. 3.2 - 3.4 the different types of metadata that should be addressed. In the table, we can see that we have filled in all the metadata categories and for each categories, we proposed more metadata than the other solutions.

Regarding the data lake management aspects, we can observe that our approach is the only one that covers different types of ingested data (structured, semi-structured and unstructured) supported with a metadata model. Moreover, regardless its importance, the state of the art approaches do not precise the ingesting mode. Our approach satisfies both batch and real-time data ingestion which are both crucial in data lake environment.

Regarding data source metadata, we proposed external data source metadata that allow users to check the provenance of ingested datasets so that they can easily find the data source in case of updating datasets. Six attributes were dedicated in our model for this subject but we have noticed that the other approaches neglected this point.

Regarding datasets metadata, we distinguish three types of metadata: intra-metadata, inter-metadata and global-metadata, we underline that:

- Intra-metadata provides a vision of each single dataset which help users to understand datasets easier and faster. Globally most of the approaches mentioned the characteristics, semantic and schematic metadata. In our approach, we have the originality of sensitivity and quality metadata to ensure the access control and data governance. Note that in our model, different versions, representations or updates of one datasets are stored as different datasets. Moreover, even a dataset is deleted, we always keep its metadata to ensure the tracability and findability of data.

- Inter-metadata indicate the relationships between different datasets to improve the efficacy of data finding and version tracing. Inter-metadata are also important to help users find relevant datasets to enrich their analyses. We remark that CM4DL, DL-Wrangling and GEMMS do not define this type of metadata. Our model is more complete with particularly the possibility for users to define their own relationships between datasets.

- Global-metadata concern the common knowledge and logs of a data lake, which reduces user learning costs. A few authors (Sawadogo et al., 2019)(Terrizzano et al., 2015) define these metadata. In our approach, we use a global dictionary for shared information among datasets and it can be simply used by research engines.

Regarding the ingestion process metadata, to the best of our knowledge, there is only GOODS (Halevy et al., 2016a,b) which address this point without a formalized processing model. We highlight that ingestion process metadata are as important as other metadata because the ingestion information allows users to reuse, update and modify ingestion process to ensure the efficiency of data ingestion. For the ingestion process, program, execution and data flow metadata should be saved. The approach of Goods (Halevy et al., 2016a) saves reading and writing jobs. In our approach, we are more generic as we save source code URL, binary machine code URL and the whole environment. GOODS does not track the execution metadata while we back up ingestion start time, duration, output log, error log and further comments. Our approach is then compliant with lineage on processes.

Concerning the process of data ingestion in data lakes, to the best of our knowledge, there are only a few solutions. (Alserafi et al., 2016) introduced metadata management during the data ingestion phase by three phases (ingest, digest and exploit) for batch data. They focused on the extraction of inter-metadata from semi-structured data. (Gupta and Giri, 2018) presented a data ingestion framework which consists of a data collector and a data integrator. They explained different approaches to bring data into a Hadoop data lake without integrating a formalized ingestion processing and a metadata model for this phase. (Terrizzano et al., 2015) introduced data ingestion by three steps without a formalized processing. These data ingestion solutions did not introduce a formalized

Metadata			**MEDAL (Sawadogo et al., 2019)**
Management	**Ingested data**		semi-structutred, unstructured data
	Metamodel		-
	Ingesting mode		*not precised*
Data source metadata			
Dataset metadata	**Intra-metadata**	**Characteristics**	*(mention with attr.)*
		(basic infomraion)	object title, size, access path
		(update)	last motif. Date
		(representation/versioning)	representations, versions
		Semantic	*(mention with attr.)*
			tags, descriptions/ business categories
		Schematic	*(mention with attr.)*
		(schema)	schema
		(volume)	
		(representation)	word cloud for text
		Sensitivity	
		Quality	
	Inter-metadata	**Relationships**	*(mention 3 types)*
		(logical clusters)	object grouping
		(similarity)	similarity links
		(provenance)	parenthood relationship
		(containment)	
		(duplicated)	
		(others)	
	Global metadata		*(mentioned 3 types of glocal metadata)*
			semantic, indexes, resources, logs
Metadata of data ingestion	**Program**		
	Execution		
	Data flow		

Table 3.2: A comparative table of Metadata for the ingestion phase (part I)

goldMEDAL(Scholly, 2021)	**HANDLE(Eichler, 2020)**	**CM4DL (Alserafi, 2016)**	**DL data wrangling (Terrizzano, 2015)**
		semi-structured data	
A core model	A core model with extensions	-	-
	not precised	*not precised*	*not precised*
(mention without attr.)			
(mention without attr.)	*(mention without attr.)*		*(mention without attr.)*
(mention without attr.)	*(mention without attr.)*	*(mention without attr.)*	*(mention with attr.)*
grouping metadata			tags
(mention without attr.)	*(mention with granularity extension)*	*(mention without attr.)*	*(mention with attr.)*
	granularity extension		schema
(mention with grouping and link metadata)		*(focused on 3 types)*	
		duplicated	
		joinable, outlier	
(mention with grouping metadata)			*(mentioned)*
			conversational metadata

Table 3.3: A comparative table of Metadata for the ingestion phase (part II)

Constance (Hai et al., 2016)	**GEMMS (Quix et al., 2016)**	**GOODS (Halevy et al., 2016a,b)**	**Our metadata model**
		structured data	structured, semi-structured, unstructured
-	A formalized model	-	A formalized model
not precised	*not precised*	*not precised*	batch and real-time
			(list 5 attr.)
			name, type, creation date, administrator, owner
	(mention without attr.)	*(list attr.)*	*(list 6 attr.)*
		aliases, access, size,format,control	name, size, type, connection url, creation date, admin.
		last motif. Date, change history	*(an updated dataset is a new dataset)*
			(a new version/representation is a new dataset)
(mention with attr.)	*(mention without attr.)*	*(list attr.)*	*(list 2 attr.)*
labels		key field, description	tags, descriptions
(mention with attr.)	*(mention without attr.)*	*(list attr.)*	*(list attr.)*
schema		schema	schema
		number of records	number of records
			(a representation is a new dataset)
			sensitivity level, description
			quality dimensions
(mentioned 2 types)		*(mentioned 4 types)*	*(4 predifined + user defined)*
schema grouping		logical clusters	logical cluster (predifined)
similarity		similarity	similarity (predifined)
		provenance	*(association relationship)*
		containment	containment (predifined)
			partial overlap (predifined)
		data fingerprint	user defined relationships
			(global metadata are generated)
			thesaurus
		reading jobs, writing jobs	source code url, environment, binary machine code url,
			ingestion start time, duration, output log, comment
		downstream/upstream datasets	target data, source data

Table 3.4: A comparative table of Metadata for the ingestion phase (part III)

and detailed ingestion process. Moreover, they did not include both the aspects of data ingestion process and the metadata generation of data ingestion. Therefore, we proposed a more complete metadata model and a data ingestion process that includes the interaction of the data lake system and users.

3.5 Conclusion

In this chapter, we focus on the metadata of data ingestion in data lakes. Data ingestion is the first phase of data life-cycle in data lakes, it concerns only data extraction and loading. During this phase, data are ingested without transformation and stored in their native format. However, with the aim of facilitating data analysis in data lakes by ensuring the findability and reusability of datasets, the ingestion process is much more complicated than a simple copy and paste of data.

To answer to this question, we brought up a complete metadata management solution that can be adapted to different types of data sources (IoT objects, databases, files), different structural types of datasets (structured, semi-structured and unstructured) and ingestion modes (batch and real-time).

Our solution is twofold: firstly, we proposed a metadata model which contain more categories of information and more attributes in different categories than the existing solutions. Moreover, the model is formalized in a class diagram that can help users to easily understand different aspects of metadata. We also instantiated the ingestion metadata for two real datasets to validate the model.

Secondly, with the aim of facilitating data lake user work, we proposed a formalized process of data ingestion for data lakes which can be rarely found in the state of the art. The ingestion process is formalized by BPMN to show the interactions between the data lake system and users. Moreover, we used three algorithms to introduce the details of metadata generation for different tasks of data ingestion. Our solution balances the efficiency and accuracy of data ingestion and also can protect sensitive data.

The metadata model with a use-case of IoT (Internet of Things) data is validated by our published work in IDEAS2021 (Zhao et al., 2021b).

Data lake is a solution of big data analytics and it is not limited to the ingestion of raw data. In the next chapter, we will introduce the next step - data preparation.

Chapter 4

Metadata on Data Preparation

Contents

4.1 Introduction

Data preparation is a set of acts that transform raw data into a form which is appropriate to be analyzed (Brownlee, 2020; Coussement et al., 2017). In the context of data mining (DM) and machine learning (ML), data preparation is also called data pre-processing, it concerns the construction of final dataset on which analysis algorithms can be applied by data cleaning, integration, transformation, reduction and discretization. (Alasadi and Bhaya, 2017; Alexandropoulos et al., 19ed; García et al., 2016; Zhou et al., 2017).

Data wranglers (data analysts, engineers and scientists) spend a bulk of time to cleaning and refining data workflows to answer analytical questions (Rezig et al., 2019). It is commonly agreed that up to 80% of a data wrangler time can be spent on transforming data into a usable dataset for either descriptive or predictive analysis (Jin et al., 2017). Data lakes offer the possibility of facilitating the data preparation phase by helping data wranglers to find, consult and reuse existing data preparation processes across different stored elements. Additionally, in a data lake, data lineage can be presented to help users easily understand how a dataset is created, so that users can have more trust on the data that they use.

To ensure the findability, accessibility, interoperability and reusability of existing processes to facilitate data preparation in data lakes, a metadata management system dedicated to data preparation is essential. On one hand, the metadata should describe not only the general information of processes, such as name, created time and execution log, but also the content of processes to help data wranglers to find and to understand a process in a easy way. On the other hand, the metadata generation should be standardized to facilitate the work of data lake users.

Different metadata solutions for data lakes have been proposed or implemented in academia and industry. Nevertheless, only a few academic data lake metadata solutions deal with information about data preparation processes for specific objectives (Hidalgo et al., 2009; Maccioni and Torlone, 2018; van Vlymen and de Lusignan, 2005; Zhang and Ives, 2020) which can not be applied on all different types of data preparation (transformation process of ETL used in BI, data mining, machine learning). The industrial products, such as Zaloni[1] or Azure[2], provide lineage metadata by retrieving the information about data source, process, and result data but these process metadata primarily provide basic process information such as program name, created time, user and free text description. This basic information is not enough to help different data wranglers, such as business intelligence (BI) experts, data scientists and data analysts on preparing data.

In the aim of doing this, we propose a data preparation metadata solution which not only contains basic information of processes but also integrates a **controlled vocabulary** to describe them. Moreover, we formalize the metadata generation through algorithms to help data wranglers better understand how to manage the preparation metadata.

To present our contribution, the purpose of the chapter is threefold. Firstly, in section 4.2, we present the metadata of data preparation in data lakes. Secondly, in section 4.3, we introduce in detail a list of predefined operations of data preparation which is the core of our metadata solution. Thirdly, in section 4.4, we explain how to generate the metadata by three algorithms. We discuss other metadata solutions of data preparation in data lakes and compare them with our solution in section 4.5. And we conclude the chapter in section 4.6.

[1]https://www.zaloni.com/aws/

[2]https://azure.microsoft.com/en-us/solutions/data-lake/

4.2 Metadata of Data Preparation

To ensure the efficiency and efficacy of data preparation by helping different data wranglers (data analysts, data scientists, data engineers) to find and reuse existing processes, the data preparation metadata (see Fig. 4.1) are essential and should include as much information as possible.

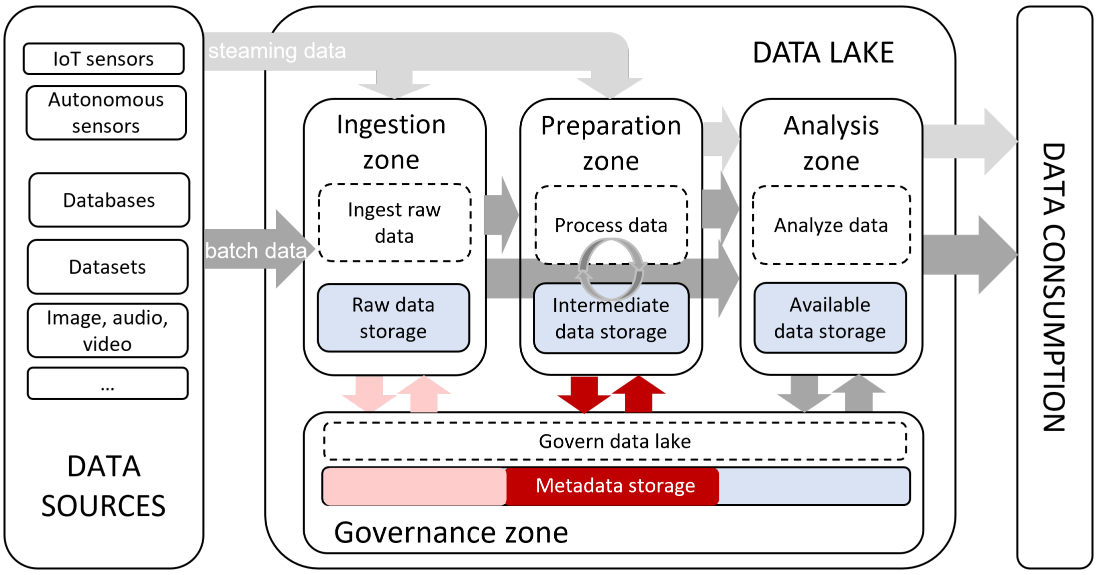


Figure 4.1: Data preparation in a data lake

To consider different aspects of information to help data wranglers comprehend how data are prepared, we chose to use the 5W1H method (abbreviation summarizing what? who? where? when? why? how?) (Shimazu et al., 2006) to achieve a methodical thinking.

The questions concerning a data process are:

- What is the source data of process? What is the result data of process? What is the process (It contains what operations)? What is the execution environment?
- Who created the processes?
- Where is the source code stored? Where is the binary machine code stored?
- When is the process created? When is the process modified? When is the process executed?
- Why the process is performed?
- How data are processed?

The above questions concern the source datasets, processes and result datasets, in this chapter, we only introduce the information of processes for the reason that dataset metadata are already introduced in the previous chapter. Regarding the questions of processes, they can be classified into three categories: process technical information, business objectives and content details.

4.2.1 Metadata Model on Data Preparation

To answers all the questions to facilitate data preparation of data wranglers, we propose a metadata model (see Fig. 4.2) that includes the three different aspects of information:

- In order to ensure that data wranglers can easily find, access and re-execute existing data processing programs, we propose process technical metadata. We decompose these metadata into characteristics, location and execution metadata:
  - **Characteristics metadata** has the objective of helping data wranglers to search all the data processing programs of data preparation and to have a global view of them (for instance, the programming language, input and output of a program) without going into details. These metadata are modeled through attributes in classes *Process*, *User* and *JobTitle*) (marked in red) which include:
    - *name* to answer the question of "what"
    - *programLanguage* to answer the question of "what"
    - *creationTime* to answer the question of "when".
    - *lastModifTime* to answer the question of "when".
    - relationships *sourceData*, *targetData* and *realtimeProcess* to help data wranglers to understand the data lineage. The metadata *sourceData* and *targetData* are for batch process which transforms one or more *datalakeDataset* and create new datasets. While *realtimeProcess* is for real-time process which connects to a data source and transforms it directly. This information is for answering the quesiton of "what".
    - relationship *hasSubprocess* to link subprocess to parent process to answer the question of "what".
    - *user.lastName*, *user.firstName*, *user.Privilege*, *jobTitle.description* and *jobTitle.jobtitle* to answer the question of "who".
  - **Location metadata** have objective of helping data wranglers to access processing program codes when they want to check, modify, or reuse them. These metadata are modeled through attributes in class *Process* (marked in orange) which include:
    - *binaryMachineCodeUrl* to help data wranglers find the executable program to answer to question of "where".
    - *sourceCodeUrl* to help data wranglers find the program source code so that they can reuse the source code directly or after modifications. This information answers the question of "where".
  - **Execution metadata** have the objective of helping data wranglers to understand how data are processed and providing the possibility of reusing or re-executing data preparation processes. These metadata are modeled through attributes in class *Process* (marked in green) which contains:
    - *executionEnvironment*, *executionOutputLog*, *executionErrorLog* and *executionComment* to answer the question of "how".
    - *executionDate* to answer the question of "when".
- In order to help data wranglers to understand the business objective of processes besides technical information, we propose the **business metadata**. These metadata are modeled through attributes in classes *Process* and *Tag* (marked in yellow) which include:
  - *process.description* to answer the question of "why".

  - *tag.name* and the relationship *hasTag* to tie some keywords to a process to facilitate the process understanding and searching. This information answers as well the question of "why".

- In order to help data wranglers to fast understand the content of a process without requiring them to open and check the programs source code line by line, we propose the **operation metadata**. These metadata are modeled through classes *OperationOfProcess* and *Operation* (marked in pink) which include:
  - *operation* which is a set of data preparation operations that are predefined in the system to answer the question of "what" (we will introduce the operations in the next section with more details).
  - *operationOfProcess* to describe the details of used operation to answer the question of "what".

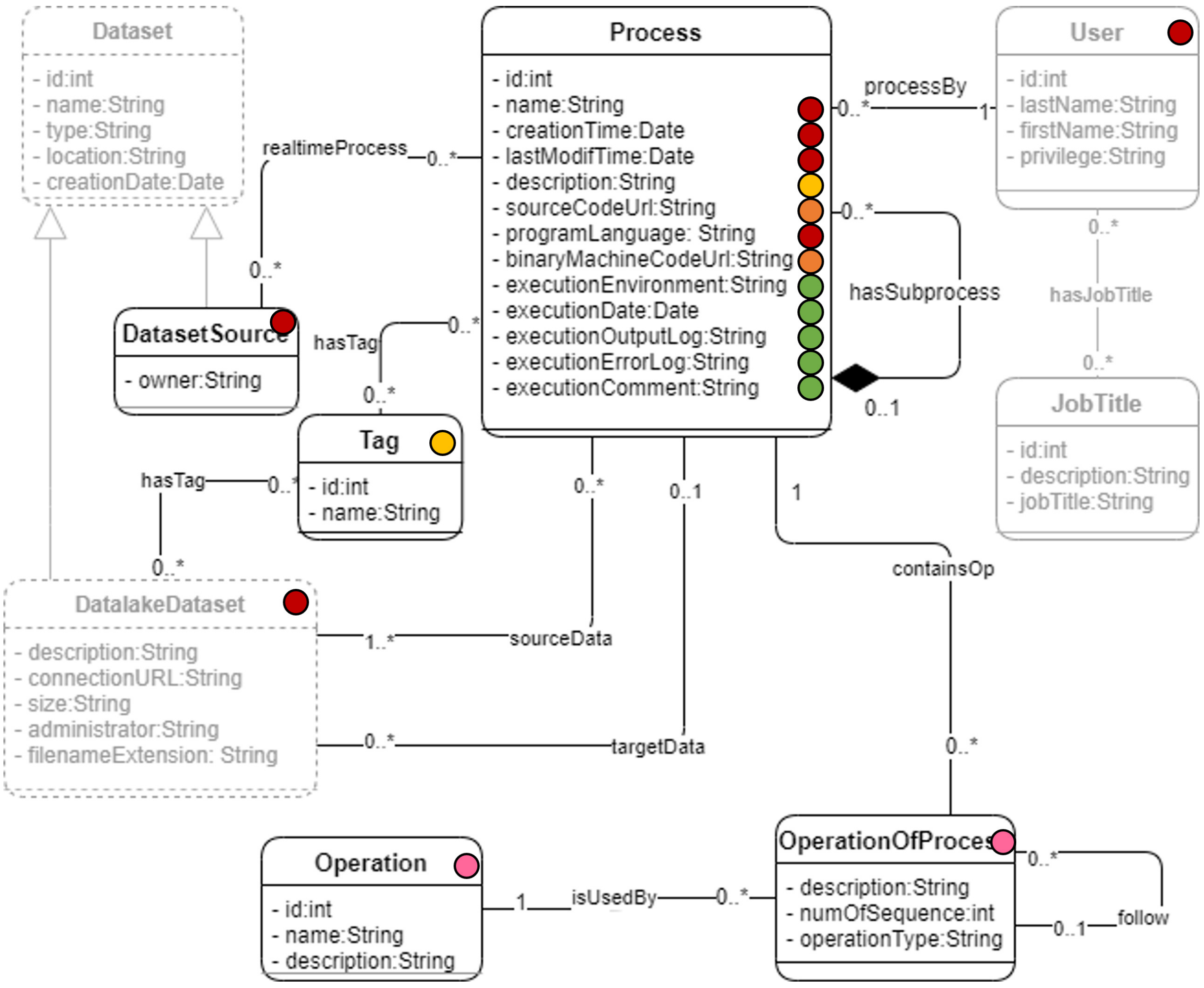


Figure 4.2: Metadata model for data preparation

A metadata management system is developed which allows users to search preparation processes and consult different categories of metadata easily (see a use case in section 6.4.2).

4.2.2 Example of Metadata Instantiation

To demonstrate our metadata model of data preparation, we introduce two examples of real data preparation cases (see table 4.1). The used datasets are of different types and are transformed by different modes.

Preparation process	Source dataset	Target dataset
MIMIC-OMOP	MIMIC	OMOP
CHSI-Cancer analysis	CHSI cancer	Colon cancer, Breast cancer, lung cancer

Table 4.1: Preparation examples

MIMIC-OMOP[3] is a process that uses ETL to transform the MIMIC dataset to a relational database OMOP that respects the OMOP CDM (common data model) for future analysis. This process includes both mapping structural clinical data and standardizing local terminologies. We present the object diagram of the data preparation in Fig. 4.3.

Due to the size of the page, we only present part of the project metadata that concerns a sub-process (etl/person) and its operations. The process *MIMIC-OMOP* is created at the 22 Jun 2021 to transform *MIMIC* dataset to *OMOP* dataset. Its source code is stored in *srlbd.***:***/user0120/ preparation/mimic-omop/sourceCode/*. The process has an sub-process which concerns the creation of the person table of OMOP, to create this table, the sub-process joined different useful tables, split dob attribute, converted patient gender information, selected useful features, and used aggregation function.

CHSI-Cancer analysis is a process to prepare an analysis that has the objective of identifying the indicators having an important impact on the colon cancer. The process is about extracting three datasets (colon, breast and lung) from the CHSI initial dataset for the reason that with an high number of indicators dedicated to multiple types of illnesses (especially on cancers), an analysis can be difficult to be performed directly in a unique dataset. The extracted datasets contain indicators oriented on the analysis of the *colon, breast and lung cancers*, respectively, through individuals with measures of *obesity*, *high blood pressure* and *smoker*. Each line represents a set of individuals living in a same *county* for each *state* of the United States. Thus, 3,141 rows are available. We present the object diagram of the data preparation in Fig. 4.4.

Due to the page limit, we only present the metadata of one sub-process that creates *colon* table and its operations. To start the metadata generation, user inputs data source (*USA_CHSI_DATASET*), prescription description (*create colon cancer dataset*), creation time (*2021-06-22T00:00:00Z*), source code URL (*srlbd.***:***/user0120/preparation/ chsi_cancer_analysis/*). For this process, user also inputs the used operation himself, the process concerns only feature selection and this information is added in the metadata system.

The presented metadata model is dedicated to data preparation of data lakes. It is a complete model that is adapted to different types of data preparation (transformation process of ETL used in BI, data mining, machine learning) and different ingesting modes (batch and real-time) by different data wranglers (data analysts, data scientist and data engineers). Moreover, the model well describes the data preparation process by different aspects to help data wranglers to search and reuse existing process easily. Especially for the data process content, we chose to use a set of predefined operations to describe it

[3]`https://github.com/MIT-LCP/mimic-omop`

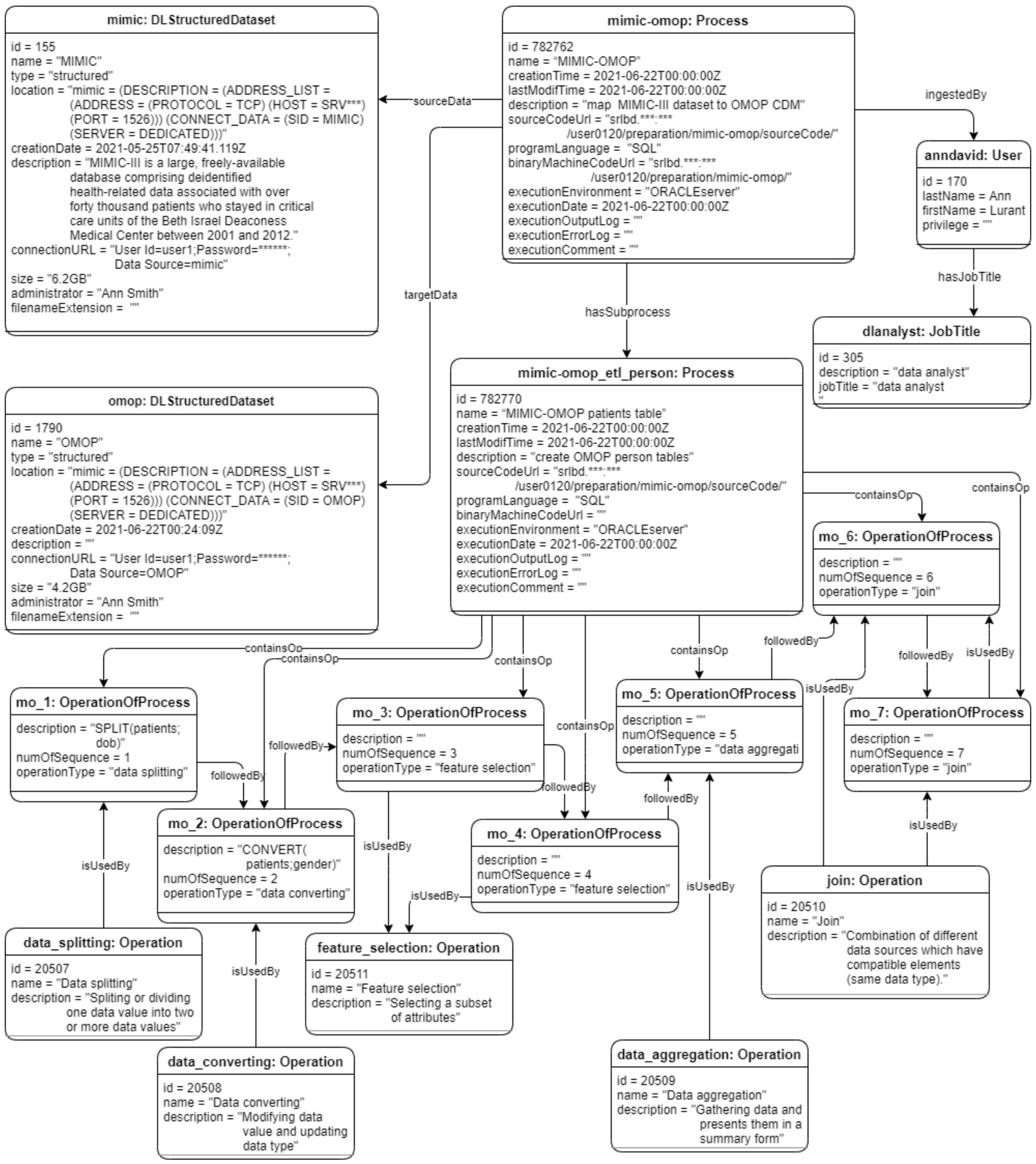

mimic: DLStructuredDataset
id = 155
name = "MIMIC"
type = "structured"
location = "mimic = (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP) (HOST = SRV***) (PORT = 1526))) (CONNECT_DATA = (SID = MIMIC) (SERVER = DEDICATED)))"
creationDate = 2021-05-25T07:49:41.119Z
description = "MIMIC-III is a large, freely-available database comprising deidentified health-related data associated with over forty thousand patients who stayed in critical care units of the Beth Israel Deaconess Medical Center between 2001 and 2012."
connectionURL = "User Id=user1;Password=******; Data Source=mimic"
size = "6.2GB"
administrator = "Ann Smith"
filenameExtension = ""
mimic-omop: Process
id = 782762
name = "MIMIC-OMOP"
creationTime = 2021-06-22T00:00:00Z
lastModifTime = 2021-06-22T00:00:00Z
description = "map MIMIC-III dataset to OMOP CDM"
sourceCodeUrl = "srlbd.***:*** /user0120/preparation/mimic-omop/sourceCode/"
programLanguage = "SQL"
binaryMachineCodeUrl = "srlbd.***:*** /user0120/preparation/mimic-omop/"
executionEnvironment = "ORACLEserver"
executionDate = 2021-06-22T00:00:00Z
executionOutputLog = ""
executionErrorLog = ""
executionComment = ""
sourceData
targetData
ingestedBy
anndavid: User
id = 170
lastName = Ann
firstName = Lurant
privilege = ""
hasJobTitle
dlanalyst: JobTitle
id = 305
description = "data analyst"
jobTitle = "data analyst "
hasSubprocess
mimic-omop_etl_person: Process
id = 782770
name = "MIMIC-OMOP patients table"
creationTime = 2021-06-22T00:00:00Z
lastModifTime = 2021-06-22T00:00:00Z
description = "create OMOP person tables"
sourceCodeUrl = "srlbd.***:*** /user0120/preparation/mimic-omop/sourceCode/"
programLanguage = "SQL"
binaryMachineCodeUrl = ""
executionEnvironment = "ORACLEserver"
executionDate = 2021-06-22T00:00:00Z
executionOutputLog = ""
executionErrorLog = ""
executionComment = ""
omop: DLStructuredDataset
id = 1790
name = "OMOP"
type = "structured"
location = "mimic = (DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP) (HOST = SRV***) (PORT = 1526))) (CONNECT_DATA = (SID = OMOP) (SERVER = DEDICATED)))"
creationDate = 2021-06-22T00:24:09Z
description = ""
connectionURL = "User Id=user1;Password=******; Data Source=OMOP"
size = "4.2GB"
administrator = "Ann Smith"
filenameExtension = ""
containsOp
containsOp
containsOp
containsOp
containsOp
containsOp
containsOp
mo_6: OperationOfProcess
description = ""
numOfSequence = 6
operationType = "join"
mo_1: OperationOfProcess
description = "SPLIT(patients; dob)"
numOfSequence = 1
operationType = "data splitting"
mo_3: OperationOfProcess
description = ""
numOfSequence = 3
operationType = "feature selection"
mo_5: OperationOfProcess
description = ""
numOfSequence = 5
operationType = "data aggregati
mo_7: OperationOfProcess
description = ""
numOfSequence = 7
operationType = "join"
mo_2: OperationOfProcess
description = "CONVERT(patients;gender)"
numOfSequence = 2
operationType = "data converting"
mo_4: OperationOfProcess
description = ""
numOfSequence = 4
operationType = "feature selection"
followedBy
followedBy
followedBy
followedBy
followedBy
followedBy
isUsedBy
isUsedBy
isUsedBy
isUsedBy
isUsedBy
isUsedBy
isUsedBy
isUsedBy
data_splitting: Operation
id = 20507
name = "Data splitting"
description = "Spliting or dividing one data value into two or more data values"
data_converting: Operation
id = 20508
name = "Data converting"
description = "Modifying data value and updating data type"
feature_selection: Operation
id = 20511
name = "Feature selection"
description = "Selecting a subset of attributes"
data_aggregation: Operation
id = 20509
name = "Data aggregation"
description = "Gathering data and presents them in a summary form"
join: Operation
id = 20510
name = "Join"
description = "Combination of different data sources which have compatible elements (same data type)."

Figure 4.3: Object diagram of process MIMIC - OMOP

so that data wranglers can easily and fast find useful processes without read the process source code in details (we will introduce the list of operation in the following section).

4.3 Coarse-Grained Data Preparation Operations

To help users to easily understand the content of a data preparation process, it is important to describe the used operations in the processing program. In a data lake, different types of data preparations (transformation process of ETL used in BI, data mining, machine learning etc.) can be applied on different types of datasets (structured, semi-structured and unstructured). However, to the best of our knowledge, there is no solution

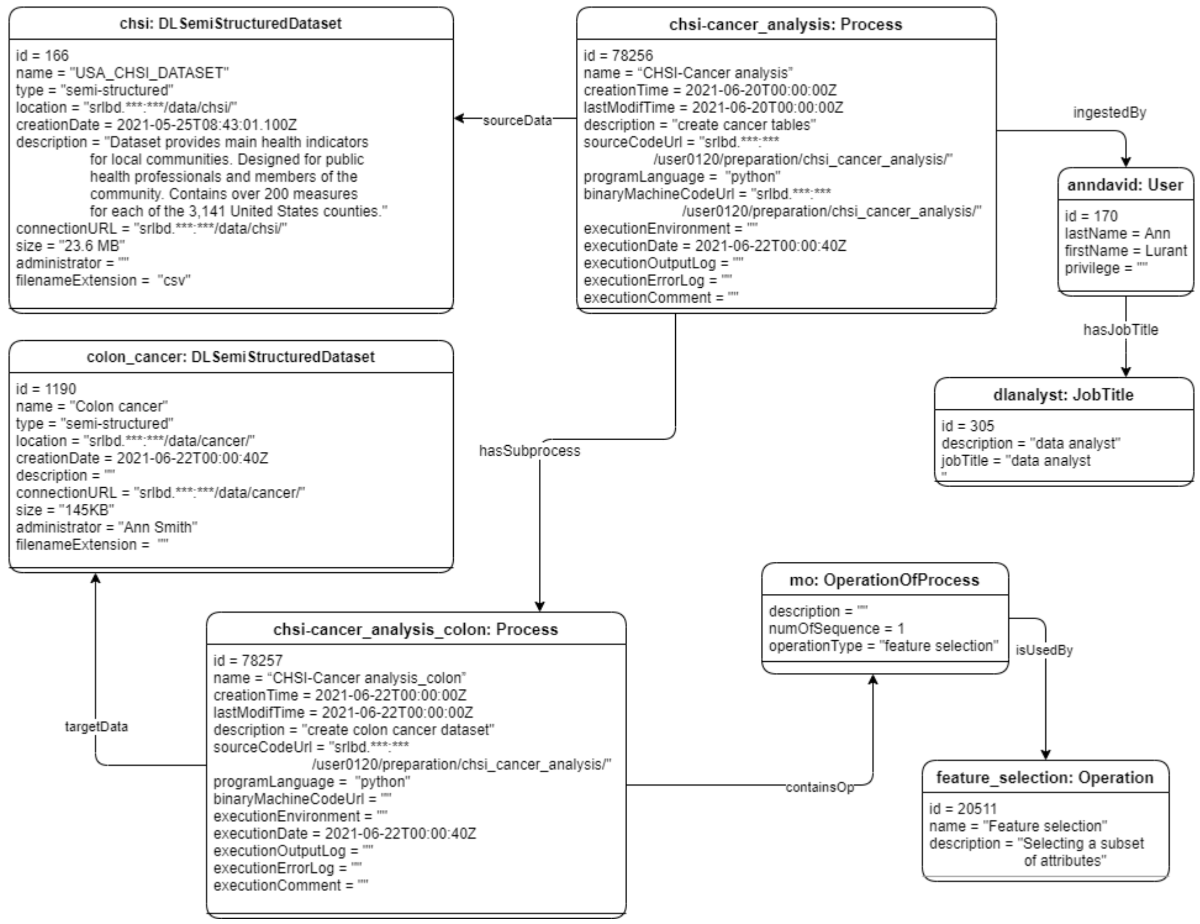

chsi: DLSemiStructuredDataset
id = 166
name = "USA_CHSI_DATASET"
type = "semi-structured"
location = "srlbd.***:***/data/chsi/"
creationDate = 2021-05-25T08:43:01.100Z
description = "Dataset provides main health indicators for local communities. Designed for public health professionals and members of the community. Contains over 200 measures for each of the 3,141 United States counties."
connectionURL = "srlbd.***:***/data/chsi/"
size = "23.6 MB"
administrator = ""
filenameExtension = "csv"
sourceData
chsi-cancer_analysis: Process
id = 78256
name = "CHSI-Cancer analysis"
creationTime = 2021-06-20T00:00:00Z
lastModifTime = 2021-06-20T00:00:00Z
description = "create cancer tables"
sourceCodeUrl = "srlbd.***:*** /user0120/preparation/chsi_cancer_analysis/"
programLanguage = "python"
binaryMachineCodeUrl = "srlbd.***:*** /user0120/preparation/chsi_cancer_analysis/"
executionEnvironment = ""
executionDate = 2021-06-22T00:00:40Z
executionOutputLog = ""
executionErrorLog = ""
executionComment = ""
ingestedBy
anndavid: User
id = 170
lastName = Ann
firstName = Lurant
privilege = ""
hasJobTitle
dlanalyst: JobTitle
id = 305
description = "data analyst"
jobTitle = "data analyst "
colon_cancer: DLSemiStructuredDataset
id = 1190
name = "Colon cancer"
type = "semi-structured"
location = "srlbd.***:***/data/cancer/"
creationDate = 2021-06-22T00:00:40Z
description = ""
connectionURL = "srlbd.***:***/data/cancer/"
size = "145KB"
administrator = "Ann Smith"
filenameExtension = ""
hasSubprocess
targetData
chsi-cancer_analysis_colon: Process
id = 78257
name = "CHSI-Cancer analysis_colon"
creationTime = 2021-06-22T00:00:00Z
lastModifTime = 2021-06-22T00:00:00Z
description = "create colon cancer dataset"
sourceCodeUrl = "srlbd.***:*** /user0120/preparation/chsi_cancer_analysis/"
programLanguage = "python"
binaryMachineCodeUrl = ""
executionEnvironment = ""
executionDate = 2021-06-22T00:00:40Z
executionOutputLog = ""
executionErrorLog = ""
executionComment = ""
containsOp
mo: OperationOfProcess
description = ""
numOfSequence = 1
operationType = "feature selection"
isUsedBy
feature_selection: Operation
id = 20511
name = "Feature selection"
description = "Selecting a subset of attributes"

Figure 4.4: Object diagram of process CHSI - Cancer analysis

that summarizes and standardizes all the operations that can be carried out in a data lake. Moreover, current solutions mainly work on a special type of data preparation and focus on the automatization of data transformation (Maccioni and Torlone, 2018; El Akkaoui and Zimanyi, 2009) by translating every line of the code of existing data processes. Nevertheless, fine-grained operational metadata require a long and tedious work that decreases the effectiveness and efficiency of data lake management.

Therefore, with the objective of helping data wranglers to know what are the main activities of preparation processes to ensure the findability and reusability of processes and at the same time to guarantee the efficiency and effectiveness of metadata management, we propose to use a set of coarse-grained operations to describe data processes. Moreover, with the aim of defining clear and standardized metadata, we propose to use a controlled language to describe all different data preparation operations.

In the context of data lake, a data process is composed of a set of operations. We define five categories that can include all the different operations: data integration, data cleaning, data transformation, data discretization and data reduction (see Fig. 4.5). Each operation can be defined as $OP((DS_{input1}, [DS_{input2}...DS_{inputN}]);[CONDITION])$, where:

- OP is the data preparation operation.
- DS_{input} is the source dataset(s).
- CONDITION is the optional argument of the operation.

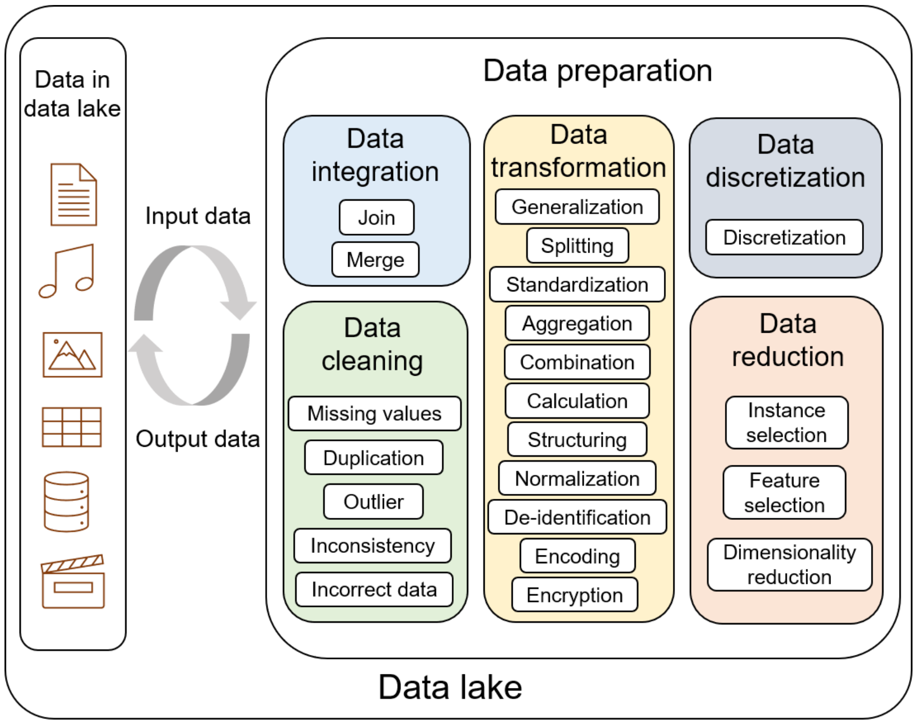


Figure 4.5: Data preparation in data lake

To be complete, for each category, we explain its operations with a definition and a controlled language (see table. 4.3).

- **Data integration** operations are used for combining multiple datasets. A data lake stores different types of datasets, whether they are schema-less or not. If data wranglers can find existing integrating operations, they can prepare data more easily and find relative datasets to enrich their data analysis.

Data integration	
Merge/Appending	$MERGE(DS_{input1}, DS_{input2})$
	Combination of different data sources which have compatible elements (same data type).
Join	$JOIN((DS_{input1}, DS_{input2}); [JOINED\_ATT])$
	Combination of different datasets with common values.

Table 4.2: Data preparation operations - data integration

- **Data cleaning** operations are used to solve the missing values, noise data, outlier, redundancy and inconsistencies problems. This step is important to ensure the required level of data quality.

Data cleaning	
Missing values	$MISSING\_VALUES\ (DS_{input}; [ATTRIBUTES])$
	Imputation of missing values and (i) discard the instances containing missing values or (ii) fill the missing value.
Incorrect data	$INCORRECT\_DATA\ (DS_{input}; [ATTRIBUTES])$

Table 4.3 continued from previous page

Data cleaning	
	Correction of error data by (i) filtering or (ii) replacing them.
Outlier	$OUTLIER\ (DS_{input};\ [ATTRIBUTES])$
	Detection and correction of outliers.
Duplication	$DUPLUICATION\ (DS_{input};\ [ATTRIBUTES\ /OBSERVATIONS])$
	Detection and limitation of same instances or columns that appears for more than one time.
Inconsistency	$INCONSISTENCY\ (DS_{input};\ [ATTRIBUTES])$
	Consistency checking and converting or replacing inconsistent data.

Table 4.3: Data preparation operations - data cleaning

- **Data transformation** operations aim to convert or manipulate data to meet the analysis requirements. In a data lake, raw data are stored in their native format and are processed while in use, in other words, data that are not formalized or cleansed are still stored. The relevant metadata can help data wranglers to save time and effort by helping them understand the native format of data and reuse existing transformations.

Data transformation	
Data aggregation	$AGGREGATE\ (DS_{input};\ [AGGREGATEDATTRIBUTES])$
	Gathering data and presents them in a summary form
Data normalization	$NORMALIZE\ (DS_{input};\ [NORMALIZEDATTRIBUTES])$
	Adjusting data values to a specific range
Data generalization	$GENERALIZE(DS_{input};\ [GENERALIZED\_\ (ATTRIBUTES])$
	Abstracting data in a dataset from a low conceptual level to a higher one
Data standardization	$STANDARDIZE(DS_{input};\ [STANDARDIZED\_\ ATTRIBUTES])$
	Transforming data to a common format according to a standard
Data converting	$CONVERT\ (DS_{input};\ [CONVERTED\_\ ATTRIBUTES])$
	Modifying data value and updating data type
Data splitting	$SPLIT\ (DS_{input};\ [SPLITTED\_\ ATTRIBUTES])$
	Spliting or dividing one data value into two or more data values
Data combination	$COMBINE\ (DS_{input};\ [ATTRIBUTES])$
	Combining two or more data values into one data value
Data calculation	$CALCULATE\ (DS_{input};\ [CALCULATED\_\ ATTRIBUTES])$
	Appliying method on existing values to judge the number
Data de-identification	$DEIDENTIFY\ (DS_{input};\ [DEIDENTITIED\_\ ATTRIBUTES])$
	Anonymizing data to protected user privacy
Data encryption	$ENCRYPT(DS_{input};\ [RESULT\_\ FORMAT])$

Table 4.4 continued from previous page

Data transformation	
	Translating dataset from the original format into another form or code
Data encoding	$ENCODE\ (DS_{input};\ [RESULT\_\ FORMAT])$
	Encoding dataset into a specified format
Data structuring	$STRUCTURE\ (DS_{input};\ [RESULT\_\ STRUCTURE])$
	Restructuring a dataset format into another format

Table 4.4: Data preparation operations - transformation

- **Data discretization** transforms continuous variables, models or functions into discrete form: $DISCRETIZE(DS_{input};\ [ATTRIBUTES])$. For instance, data binning, histogram, entropy based and clustering. Data discretization metadata allow data wranglers to understand what data is transformed and how the transformations are done so data wranglers can directly reuse the result or process.

Data discretization	
Data discretization	$DISCRETIZE(DS_{input};\ [ATTRIBUTES])$
	Transforming continuous variables, models or functions into discrete form

Table 4.5: Data preparation operations - discretization

- **Data reduction** has the objective of reducing dataset size by selecting a portion of data, compressing data or replacing or mapping data to an alternate or smaller representation of data. Data reduction information can help data wranglers to better prepare data for analysis. For instance, the feature selection metadata helps data wranglers to understand potentially relative attributes; the dimensionality reduction process allows data wranglers to represent data in a reduced dimension space.

Data reduction	
Instance selection	$IS\ (DS_{input};\ [INSTANCES\_\ NUMBER\ /CONDITION])$
	Reducing the quantity of data by removing instances
Feature selection	$FS\ (DS_{input};\ [FEATURES\_\ NUMBER\ /CONDITION])$
	Selecting a subset of attributes
Dimensionality reduction	$DR\ (DS_{input});\ [DIMENSION\_\ NUMBER])$
	Transforming data from a high-dimensional level into a low-dimensional level to compress data

Table 4.6: Data preparation operations - reduction

Example of the controlled language. To better introduce the operations, we present an example application of a process which have for objective of mapping MIMIC patient data to OMOP standardized person table[4]. This process concerns extracting data from three MIMIC tables (*patients*, *admissions*, *gcpt_ethnicity_to_concept*), transforming data

[4]`https://ohdsi.github.io/CommonDataModel/cdm531.html`

according to OMOP standards and loading data into the *person* table. Regarding the extraction of *patients* table, four attributes (features) are selected (*subject_id, mimic_id, gender, dob*) before which gender is converted to OMOP concept_id ('F' → 8532, 'M' → 8507) and dob (data of birth) is split into year, month and day. Regarding the extraction of *gcpt_ethnicity_to_concept*, two attributes (features) are selected (*ethnicity, race_concept_id, ethnicity_concept_id*). Regarding the extraction of *admissions* table, two attributes (*subject_id, ethnicity*) are selected before which data are aggregated by the attribute *subject_id*. Then all the three tables are joined to load all extracted data into one OMOP table *person*. This process uses the following operations:

- patients ← FS ($patients$; $subject\_id$, $mimic\_id$, $CONVERT$ ($patients$; $gender$), $SPLIT$ ($patients$; dob))
- gcpt_ethnicity_to_concept ← FS ($gcpt\_ethnicity\_to\_concept$; $ethnicity$, $race\_concept\_id$, $ethnicity\_concept\_id$)
- admissions ← $AGGREGATE$ ($admissions$; $subject\_id$)
- OMOP_SCHEMA.PERSON ← $JOIN$(($gcpt\_ethnicity\_to\_concept$, $JOIN$(($patients$, $admissions$); $subject\_id$)) ; $race\_source\_value$)

4.4 Generation of the Preparation Metadata

In the aim of facilitating the tasks of data wranglers and ensuring homogeneous integration of metadata, the generation of metadata should be standardized. To do so, we propose to use algorithms to systematize the metadata generation. Metadata can be generated in two ways: manual and automatic. Manual generation requires data wrangler to input all the used operations by hand. The manually input metadata have a better veracity but it costs wranglers effort and time. Automatic generation is done by the metadata system, this type of generation saves time and effort of data wranglers but the result is less veracious than manual input. To balance the efficiency and correctness of operation metadata generation, we propose to mix the two ways: (i) data wranglers need to input metadata that can not be detected by the system, such as process description, tags, process binary machine code URL and source code URL, they can also input or validate the operation metadata in the case that they want to improve the veracity. (ii) The metadata system should automatically detect execution and operation metadata to facilitate the work of data wranglers.

We explain the metadata generation through two algorithms, in Algo. 4, we introduce how the system can store the general preparation metadata and in Algo. 5, we introduce the detection of operation metadata in details.

To start the generation of preparation metadata, users need to indicate the source data; input the process description, creation data, process source code location, binary machine code location, execution environment; upload execution log and process source code file; and set the operation detection level (see Algo. 4). The system firstly detect the process name and program language from the process code file (*line 1-2*). Secondly, it detects execution date and error log from the log file (*line 3-5*). Thirdly, all the input and detected information are stored for the process object (*line 6-7*). Finally, the system detects and stores operation metadata.

Algorithm 4: Detect general data preparation metadata

```
Input: data_source, prcs_desc, prcs_creation_date, prcs_source_code_url,
       prcs_binary_machine_code_url, prcs_execution_env,
       prcs_execution_comment, detection_level
Data: process file, prcs_execution_log, dic_file_extensions
1  prcs_name ← get_process_name(process)
2  prcs_lang ← get_process_language(get_file_extension(process),
     dic_file_extensions)
3  execution_date ← get_execution_date(prcs_execution_log)
4  execution_log ← get_execution_log(prcs_execution_log)
5  execution_error_log ← get_execution_error_log(prcs_execution_log);
6  proc ← createNode('Process', createProperties('name', prcs_name,
     'creationDate', prcs_creation_date, 'description', prcs_desc, 'sourceCodeUrl',
     prcs_source_code_url, 'language', prcs_lang, 'binaryMachineCodeUrl',
     prcs_binary_machine_code_url, 'executionEnvironment', prcs_execution_env,
     'executionDate', execution_date, 'executionOutputLog', execution_log,
     'executionErrorLog', execution_error_log, 'executionComment',
     prcs_execution_comment)
7  createRelation('DatalakeDataset-Process', data_source, proc)
8  if detection_level == 'comment_level' then
9      detecte_operation_comment(proj_lang, process, proc)
10 else
11     detecte_operation_code(proj_lang, process, proc)
```

The automatic operation metadata generation can be done in two levels (see Algo. 5). (i) The first level is the comment level for which the operation detection will run on all the comments of a processing program. The advantage of this level is that explicit words in the comments can facilitate the coarse-grained operation detection. The disadvantage is that when a program code does not have comment or the comments are scribble, then there is no result or the result does not make sense. (ii) The second level is the code level for which the auto-detect program will run on the process script code. This level can always provide a result but the detection can be lengthy when the program code is complicated. For the automatic operation detection, the following prerequisites are required:

- A dictionary of filename extension (*Dic_file_extensions*) to recognize different types of language (.py, .java ...).
- A dictionary of comment syntax (*Dic_comment_syntax*) of different languages to extract comments from different languages ('#' for Python, '//', '/*' and '*/' for Java...).
- A dictionary of operations (*Dic_operations*) to link keywords in comments to operations. For instance, the words "join", "joins", "leftjoin", "rightjoin" are all linked to the operation JOIN(). The process of detecting keywords from comments can be optimized by tokenization borrowed from nature language processing (NLP) techniques, for instance, Byte Pair Encoding (BPE).

- A dictionary of predefined words used in the syntax of different languages (*Dic_syntax _words*) to extract operations from source code of processes. For instance, *select* is a word used in SQL language to select features, *SelectKBest* is a word (function name) used in Python for the same objective.

When users choose comment level, the function *detect_operation_comment()* is called (see Algo. 5). The system finds firstly the comment syntax of the programming language with the help of *Dic_comment_syntax* (*line 2*). Secondly, all the comments in the program code will be extracted (*line 3*). Then, for each line of comment, the system detects key annotation words to generate and store operation metadata (line 4-9). When code level is selected, the function *detect_operation_code()*, the code level has the same logic then the comment one, but it get all the code lines and detects syntax key words with (*Dic_syntax _words*) to retrieve used operations.

Algorithm 5: Detect operation metadata at comment level and code level

```
   Input: proj_lang, process, proc
   Data: dic_comment_syntax, Dic_syntax_words
1  Function detect_operation_comment(proj_lang, process, proc):
2      comment_syntaxes[] ← get_comment_syntaxe(proj_lang,
        dic_comment_syntax)
3      comments[] ← get_all_comments(process, comment_syntaxes[])
4      foreach com ⊂ comments[] do
5          op ← check_operation(com, dic_operations)
6          if op then
7              operation_of_process ← createNode(‘OperationOfProcess’,
                createProperties(‘description’, com))
8              createRelation(‘Process-OperationOfProcess’, proc,
                operation_of_process)
9              createRelation(‘OperationOfProcess-Operation’, operation_of_process,
                op)
10 Function detect_operation_code(proj_lang, process, proc):
11     comment_syntaxes[] ← get_comment_syntaxe(proj_lang,
        dic_comment_syntax)
12     code_lines[] ← get_all_code_lines(process, comment_syntaxes[])
13     foreach cl ⊂ code_lines[] do
14         op ← check_operation(cl, dic_syntax_words)
15         if op then
16             operation_of_process ← createNode(‘OperationOfProcess’,
                createProperties(‘description’, cl))
17             createRelation(‘Process-OperationOfProcess’, proc,
                operation_of_process)
18             createRelation(‘OperationOfProcess-Operation’, operation_of_process,
                op)
```

4.5 Discussion

In this chapter, we introduced a metadata model dedicated to data preparation and a formalized process of metadata generation.

Regarding metadata of data preparation in data lakes, we proposed a model that includes process technical (characteristics, location and execution), definitional and operation metadata to allow data wrangler to find, access, interoperate and reuse existing data preparation processes. Our model is more complete than current solutions. To the best of our knowledge, there are only a few works (Maccioni and Torlone, 2018; Hidalgo et al., 2009; van Vlymen and de Lusignan, 2005; Zhang and Ives, 2020) in the literature that introduced metadata on data preparation in the context of big data or data lake.

The author of (Maccioni and Torlone, 2018) presented a framework KAYAK that helps data scientists to define and optimize data preparation pipelines of data mining in data lakes. Their data lake has a metadata management which manages inter- and intra- metadata of datasets. Although their solution uses pipelines (a set of primitives), primitives (a set of tasks) and tasks (an atomic operation in KAYAK) to describe data preparation process, this information is not included in their metadata system.

The authors of (Hidalgo et al., 2009) define a metadata schema describing data preparation tasks in the context of data mining. The system aims to automate data preparation by identifying its requirements which are classified into eight categories: objective, output, definition, control, flow, content, composition and execution. This approach focuses on the automatization of data preparation by identifying tasks of processes. However, the proposed metadata model does not have enough information to help users easily reuse a process, such as, process source code location, process binary machine code location, program language and execution environment, etc.

The authors of (van Vlymen and de Lusignan, 2005) introduced a metadata system for primary care big data to control the process of transformation and analysis. The system adds six elements of metadata to the Primary Care Data Quality (PCDQ) renal program: study/audit name, queries of data extraction, data collection number, data type, repeat number and a processing suffix. As described in this approach, we observe that there is a focus on quality aspects which does not cover all the transformations and problems that we find in DL. Moreover, in this work there is not a generic metadata model presented.

The authors of (Zhang and Ives, 2020) develop a DL solution of search and management for the Jupyter Notebook data science platform. The framework additionally takes into account the data process to help users specify the type of task to perform. However, their solution focuses on the datasets relatedness searching in Machine Learning (ML) domain. It is not enough for data lakes that allow different types of data processing.

In the industrial world, different solutions of data lake with integrated metadata management are offered, eg. Microsoft Azure, Amazon AWS. There are also well developed open-source solutions such as Apache Atlas. Nevertheless, these metadata solutions mainly take into account the data stored in data lake. Regarding data preparation, metadata are limited to basic information, eg. source data, target data and process time which are not enough to help wranglers to find and reuse processes.

Moreover, our metadata model can be applied on different types of data preparation (transformation process of ETL used in BI, data mining, machine learning). To check the completeness of our proposed operation metadata, we compare in table. 4.7-4.8 the list of our data preparation operations with other solutions. The work we refer to in the table is mostly for data mining or machine learning data preparation, because to the best of

Data preparation activities	**Operations**	**Our solution**	**Alasadi and Bhaya (2017)**	**Iliou et al. (2018)**	**Alexandropoulc et al. (19ed)**	**Zhou et al. (2017)**
Data cleaning	Missing values	●	●	○	●	
	Incorrect data	●		○		●
	Outlier	●	●	○	●	
	Duplication	●				●
	Inconsistency	●		○		●
Data integration	Merge (Appending)	●	○	○		
	Join	●	○	○		
Data transformation	Data aggregation	●	●	○		
	Data normalization	●	●	○	●	
	Data generalization	●	●			
	Data standardization	●				
	Data converting	●				
	Data splitting	●				
	Data combination	●				
	Data calculation	●				
	Data de-identification	●				
	Data encryption	●				
	Data encoding	●				
	Data structuring	●				
Data reduction	Instance selection	●			●	
	Feature selection	●	●	○	●	●
	Dimensionality reduction	●	●			
Data discretization	●	●				

● : *explained with details;* ○ : *mentioned without details*

Table 4.7: Comparison of different solutions of data preparation operations (Part I)

Data preparation activities	**Operations**	**García et al. (2015)**	**García et al. (2016)**	**Berti-Equille (2019)**	**Wirth and Hipp (2000)**	**Megdiche et al. (2021)**
Data cleaning	Missing values	●	●	●	●	○
	Incorrect data	●	●		●	○
	Outlier			●		
	Duplication			●		
	Inconsistency			●	●	
Data integration	Merge (Appending)	○				●
	Join	○				●
Data transformation	Data aggregation					●
	Data normalization	●		●		○
	Data generalization					
	Data standardization					
	Data converting					○
	Data splitting					○
	Data combination					○
	Data calculation					●
	Data de-identification					○
	Data encryption					
	Data encoding					○
	Data structuring					●
Data reduction	Instance selection	●	●		●	●
	Feature selection	●	●	●	●	●
	Dimensionality reduction		●			
Data discretization						

● : *explained with details;* ○ : *mentioned without details*

Table 4.8: Comparison of different solutions of data preparation operations (Part II)

our knowledge, there is no data lake metadata solution summarized possible preparation operations. From the table, we can see that some authors only mentioned names of operations without explaining them in details (Iliou et al., 2018), some authors explained limited operations with details (Alasadi and Bhaya, 2017; Alexandropoulos et al., 19ed; Zhou et al., 2017; García et al., 2015, 2016; Berti-Equille, 2019; Wirth and Hipp, 2000). Our list of coarse-grained operations covers all different tasks of data preparation (integration, cleaning, transformation, reduction and discretization) and can be applied on different types of preparations (Extract, Transform, Load (ETL) procedure, data mining, machine learning) for all different structural types of data (structured, semi-structured and unstructured).

Regarding the preparation metadata generation, to the best of our knowledge, our solution is the only one that can generate all the preparation metadata thought wranglers manual input and system automatic generation at comment and code levels in data lakes.

4.6 Conclusion

In this chapter, we focus on the metadata of data preparation in data lakes. Data preparation is the phase during which raw data are transformed by a set of acts into a form that is appropriate to analyze. It is commonly agreed that is the most time-consuming and effort-consuming phase of data analysis. Data lakes can facilitate the data preparation phase by helping data wranglers to find, access and reuse existing data transformations if an efficient and effective metadata management system is integrated. However, today, only few works focus on data preparation in data lakes and none of them are dedicated to data preparation searching and reuse.

To address this deficiency, we proposed a complete metadata management solution dedicated to data preparation that can be adapted to different types of data processes (transformation process of ETL used in BI, data mining, machine learning) and ingestion modes (batch and real-time). Our solution is twofold: firstly, we borough up a formalized metadata model which is complete and which includes different categories of process information (technical, business, operation). The operation metadata is based on a predefined list of operations which can be applied to different types of data preparation and which are formalized by a controlled language. Secondly, in order to facilitating the work of data wranglers by automatizing metadata generation, we proposed a formalized process through algorithms. This process combines data wranglers manual input and system automatic metadata detection to balance the efficiency and correctness of metadata management. Moreover, the automatic detection can be done on two levels (program code level and comment level) for different use cases.

The metadata model is validated by our published work in SOFSEM2021 (Megdiche et al., 2021).

Data lake is a solution of big data analytics and data preparation is dedicated to prepare data for final analyses. In the next chapter, we will introduce the next step - data analysis.

Chapter 5

Metadata on Data Analysis

Contents

5.1 Introduction

Data analysis is the activity during which different users (such as data scientists, data analysts and BI professionals) study and examine data by different techniques (such as data mining, machine learning and reporting) to discover useful information for supporting decision-making (Judd et al., 2011; O'Neil and Schutt, 2013).

Data lakes, one of the most popular data analytic solutions of today, should help users to perform more efficient and efficacious data analysis on different aspects:

- helping users to find useful and relevant existing datasets or analyses by crossing all stored elements.
- helping users to interoperate or reuse existing analyses by providing information of used algorithms with parameters and evaluation results etc.
- helping users to choose the most appropriate model/algorithm by providing information or pre-executed landmarkers results.
- helping users to generate and maintain the above information to facilitate data lake management.

Although the ultimate goal of data lake is to facilitate data analysis, few data lake solution is proposed to improve the effectiveness and efficiency of the work of data analysts through metadata dedicated to support decisional analysis. Regarding data analysis without considering data lakes, in the field of meta-learning, different metadata models of machine learning or data mining analysis are proposed by W3C (Esteves et al., 2016) and other authors (Keet et al., 2015; Panov et al., 2014). Although these solutions can support data analysts to make better choices on data mining process for specific datasets, users can not find needed information by looking though all the existing datasets and analyses as well as the relationships between them.

Therefore, to address this absence, with the aim of facilitating data analysis by improving the findability, accessibility, interoperability and reusability of existing analysis and providing datasets and attributes descriptive information to help users on dataset and feature selection, we propose a metadata solution dedicated to data analysis. Our solution is twofold, firstly, we propose an exhaustive metadata model related to the data analysis phase in section 5.2. Secondly, we formalize the process of analysis metadata generation to make it systematic, standardized for all users and to ensure the automatization of metadata generation in section 5.3. Finally, we compare our solution with the state of the art in section 5.4 and conclude the chapter in section 5.5.

5.2 Metadata Model of Data Analysis

In order to help users to find and reuse performed model or algorithms, the metadata applied on data analysis (see Fig. 5.1) should include as much information as possible and should be comprehended by different types of users (data analysts, data scientists and data engineers).

To consider different aspects of information that can help users to search existing analyses, to understand how an analysis is designed and performed, to know interesting datasets detailed information, we choose to apply the 5W1H method (abbreviation summarizing what? who? where? when? why? how?) to achieve a methodical thinking.

The questions concerning a data analysis are:

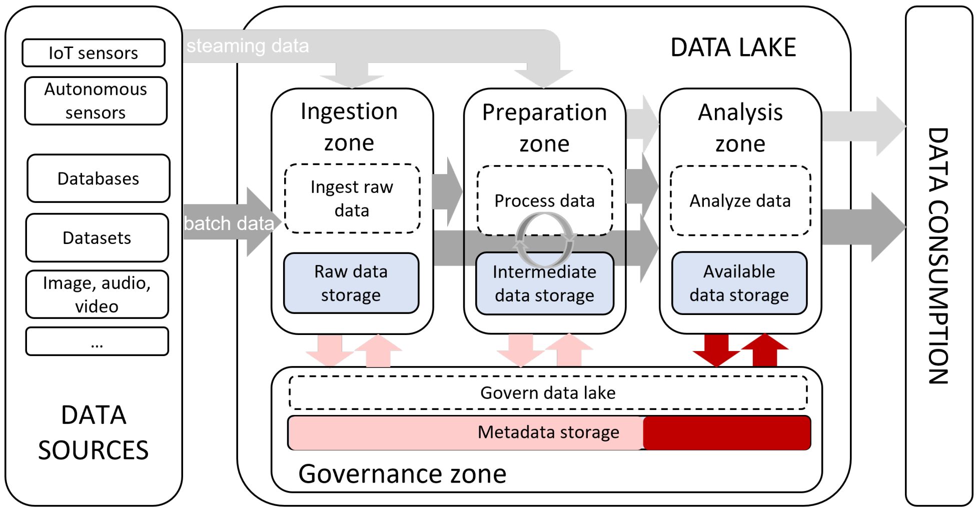


Figure 5.1: Data analysis in a data lake

- What is the analysis? What is the analyzed dataset? What are the analyzed attributes? What are the selected features? What is the selected class (target attribute)? What is the output model? What are the relevant datasets of the analyzed dataset? What are the datasets that correspond to a new analysis subject? What are the details of an interesting dataset? What are the attributes in an interesting dataset?
- Who performed the analyses?
- Where is the analysis implementation stored?
- When was the analysis performed?
- Why was the analysis performed?
- How was the analysis performed (used tool, algorithm, parameters)? How was the analysis result evaluated (measures, results)?

The above questions can be classified into three categories: questions of the dataset, attribute and analysis information.

5.2.1 Metadata Model on Data Analysis

To answer the questions, we propose a metadata model that capitalizes the full experiences of data analysis which includes (i) descriptive information about datasets (statistics of dataset values), (ii) descriptive information about attributes and (iii) analytical information about analyses (performed implementations and the used algorithms with parameters). Note that we already introduced dataset metadata including characteristics, quality and security information in section 3.2. However, the introduced dataset metadata are not enough to help users to decide whether or not analyze a dataset. Therefore in this section, we will introduce in details more datasets metadata oriented to data analysis.

5.2.1.1 Metadata on Datasets

The dataset metadata have the objective to allow users to have a clear vision of dataset schema and the statistical information of its value without necessarily opening and brows-

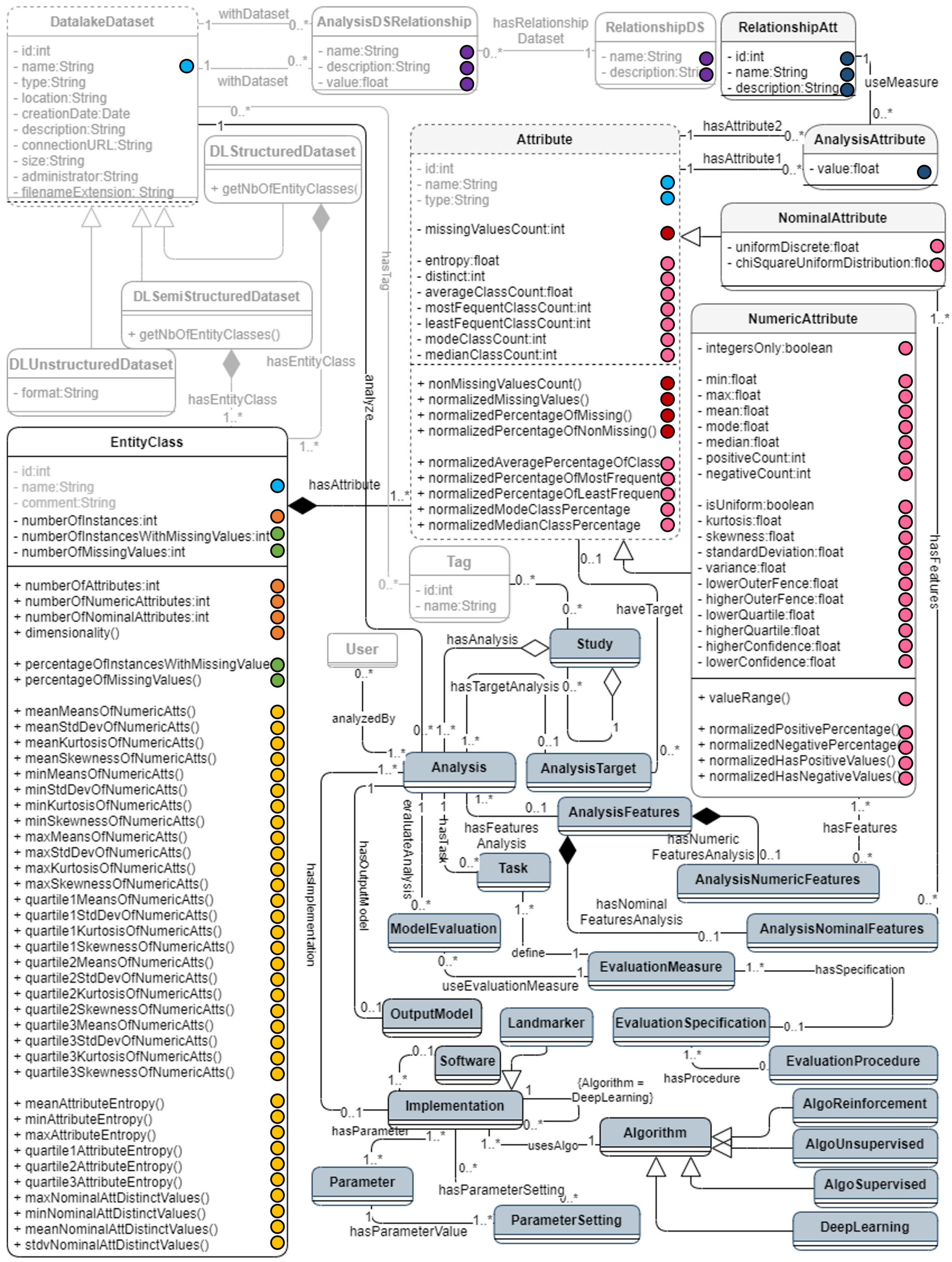


Figure 5.2: Metadata on datasets and attributes

ing the dataset to facilitate data examining. With this objective, we propose metadata of datasets which include five categories of information (see white classes in Fig. 5.2):

- To facilitate the feature engineering step (including feature selection problems and target class understanding) by helping users to understand the structure of datasets, we propose **schema metadata** which relate to tables/entities and attributes of dataset. These metadata are modeled through the attributes marked in blue in three classes (*DatalakeDataset*, *EntityClass* and *Attribute*).

- To help users to solve dimensionality issues such as reducing dimensionality of size limits related to execution environments, we propose **dimension metadata** which relate to the size of dataset. The metadata include the number of attributes, number of instances as well as the dimensionality (number of attributes divided by number of instances) of each table/entity in a dataset. They are modeled through the statistical information of datasets which is marked in orange in the class *EntityClass*.

- To be transparent on the quality of analysis results, data analysts should be aware of the presence of missing data in dataset and adopt the best strategy. To handle this need in data analysis, we propose **missing value metadata**. These metadata include the information of the number and percentage of missing values as well as the number and percentage of instances containing missing values of each table/entity of a dataset. They are modeled through the attributes marked in green in the class *EntityClass*.

- When users want to do an analysis, they have to agree on the bias of the data that they want to manage. To help users to understand the data bias (such as measurement bias, sampling bias, representation bias(Dorleon et al., 2021) we propose **distribution metadata**. These metadata are expressed by 34 information, such as attribute entropy, kurtosis of numeric attributes and standard deviation of numeric attributes of a dataset. They are modeled through the attributes marked in yellow in the class *EntityClass*.

- To select analysis algorithm, past experiences in similar datasets can be very helpful, so the **dataset relationship** metadata are proposed. These metadata refer to relationships between different datasets such as similarity/dissimilarity and correlation. They are modeled through the attributes marked in purple in the classes *AnalysisDSRelationship* and *RelationshipAtt*.

5.2.1.2 Metadata on Attributes

In order to allow users to have detailed information of attributes in the chosen dataset and facilitate the feature selection and algorithm selection phase of data analysis. We propose the attribute metadata to help users to understand a dataset at the attribute level. For example, to select features, users may need to choose attributes that do not contain any missing values for their predictive analysis. Or when dimensionality reduction approaches (Principal Component Analysis (PCA)) are employed, users need to know indicators of correlation or covariance.

The metadata on attributes includes five categories of information (see the gray classes in Fig. 5.2):

- **Attribute type metadata** concern two levels of classification, the first one indicates if an attribute is nominal or numeric (classes *NominalAttribute* and *NumericAttribute*, the second one indicates precisely the type of an attribute (*Attribute.type*).

- **Missing value metadata** (marked in red) consist of the number of missing values, number of non-missing values, normalized missing values count and their percentages of each attribute.
- **Distribution metadata** (marked in pink) concern the statistical analysis of attribute values, such as the entropy, distinct values, Kurtosis of numerical attribute and whether the values of a nominal attribute follow a discrete uniform distribution.
- **Attribute relationship metadata** concern the relationships between two different attributes (marked in dark blue), for which different relationships such as the Spearman's rank correlation, Pearson's correlation and mutual information can be calculated.

5.2.1.3 Metadata on Data Analysis

The purpose of metadata on analysis is to ensure collaborative capabilities. As far as machine learning is concerned, especially supervised machine learning, metadata on datasets and attributes are useful but not sufficient. Indeed, the same dataset can be used for different analyses with different classes/targets or features (in this chapter, we define the class/target attribute for supervised analysis as the target attributes, all the other attributes are features.). It is important to help users to obtain information on previous analyses, as existing analyses can be directly reused or contribute to improve current analyses. In addition, a user can also be interested in landmarker information which is obtained by performing basic learning algorithms.

Therefore, we propose the analytical metadata (see the dark blue classes in Fig. 5.3) to help users find and potentially reuse existing analyses with their basic characteristics, selected features and target class, implementation technical information, output model and evaluation of the result.

- In order to facilitate analysis searching and to help users to have a general view of an analysis, we propose analysis **basic characteristics metadata**. These metadata are modeled in classes *Analysis*, *Study*, *Task*, *Tag* and *User* (marked in red). A study is a project which is a set of analyses that have the same subject. An analysis can link to a task which concerns an aspect of objective of the project.
  - *Analysis.name*, *Study.name*, *Task.name*, *Analysis.typeAnalysis* (for instance, machine learning), *Analysis.subTypeAnalysis* (for instance, supervised) are the basic information of analysis and they answer the question of "*what*".
  - *Analysis.creationDate* is the date of creation of an analysis, it answers the question of "*when*".
  - *Analysis.descriptionAnalysis*, *Study.description*, *Task.description* describe the objective or subject of analyses, they answer the question of "*why*".
  - *user* is the person who preformed the analysis, it answers the question of "*who*".
- In order to help users to know the selected features and target/class of an analysis so that they can decide easier weather or not use the same features for other analyses, we propose **analysis of features/target metadata**. These metadata concern statistical analysis of features/target values and they answer the question of "*what features/target*". They are modeled in the classes *AnalysisTarget*, *AnalysisFeatures*, *AnalysisNumericFeatures*, *AnalysisNominalFeatures* (marked in purple).

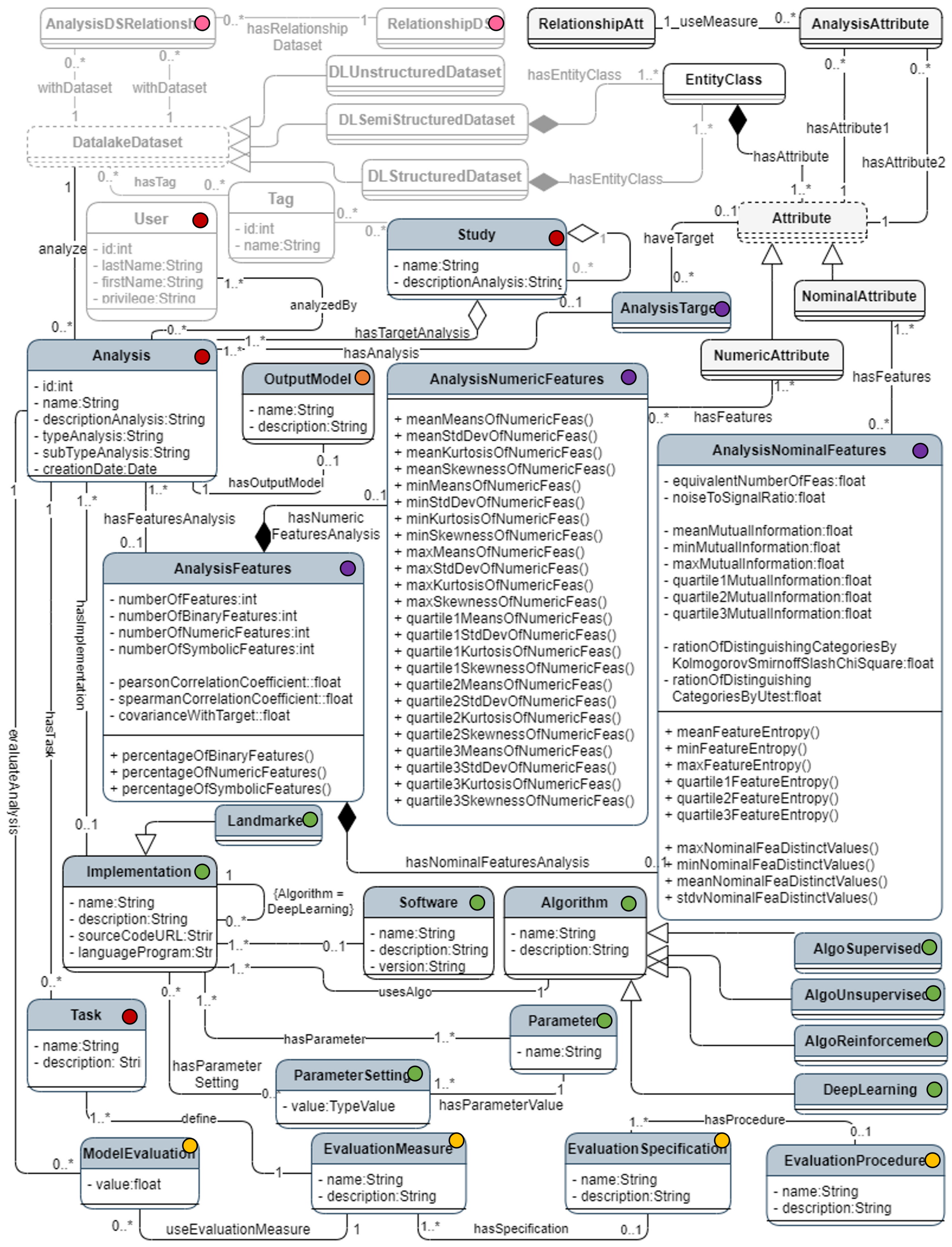

AnalysisDSRelationship
hasRelationshipDataset
RelationshipDS
RelationshipAtt
useMeasure
AnalysisAttribute
withDataset
DLUnstructuredDataset
hasEntityClass
EntityClass
DLSemiStructuredDataset
DatalakeDataset
hasAttribute1
hasAttribute
hasAttribute2
DLStructuredDataset
hasTag
Tag
- id:int
- name:String
User
- id:int
- lastName:String
- firstName:String
- privilege:String
Study
- name:String
- descriptionAnalysis:String
haveTarget
Attribute
analyze
analyzedBy
AnalysisTarget
NominalAttribute
hasTargetAnalysis
hasAnalysis
NumericAttribute
Analysis
- id:int
- name:String
- descriptionAnalysis:String
- typeAnalysis:String
- subTypeAnalysis:String
- creationDate:Date
OutputModel
- name:String
- description:String
AnalysisNumericFeatures
+ meanMeansOfNumericFeas()
+ meanStdDevOfNumericFeas()
+ meanKurtosisOfNumericFeas()
+ meanSkewnessOfNumericFeas()
+ minMeansOfNumericFeas()
+ minStdDevOfNumericFeas()
+ minKurtosisOfNumericFeas()
+ minSkewnessOfNumericFeas()
+ maxMeansOfNumericFeas()
+ maxStdDevOfNumericFeas()
+ maxKurtosisOfNumericFeas()
+ maxSkewnessOfNumericFeas()
+ quartile1MeansOfNumericFeas()
+ quartile1StdDevOfNumericFeas()
+ quartile1KurtosisOfNumericFeas()
+ quartile1SkewnessOfNumericFeas()
+ quartile2MeansOfNumericFeas()
+ quartile2StdDevOfNumericFeas()
+ quartile2KurtosisOfNumericFeas()
+ quartile2SkewnessOfNumericFeas()
+ quartile3MeansOfNumericFeas()
+ quartile3StdDevOfNumericFeas()
+ quartile3KurtosisOfNumericFeas()
+ quartile3SkewnessOfNumericFeas()
hasFeatures
AnalysisNominalFeatures
- equivalentNumberOfFeas:float
- noiseToSignalRatio:float
- meanMutualInformation:float
- minMutualInformation:float
- maxMutualInformation:float
- quartile1MutualInformation:float
- quartile2MutualInformation:float
- quartile3MutualInformation:float
- rationOfDistinguishingCategoriesByKolmogorovSmirnoffSlashChiSquare:float
- rationOfDistinguishingCategoriesByUtest:float
+ meanFeatureEntropy()
+ minFeatureEntropy()
+ maxFeatureEntropy()
+ quartile1FeatureEntropy()
+ quartile2FeatureEntropy()
+ quartile3FeatureEntropy()
+ maxNominalFeaDistinctValues()
+ minNominalFeaDistinctValues()
+ meanNominalFeaDistinctValues()
+ stdvNominalFeaDistinctValues()
hasOutputModel
hasFeaturesAnalysis
hasNumericFeaturesAnalysis
AnalysisFeatures
- numberOfFeatures:int
- numberOfBinaryFeatures:int
- numberOfNumericFeatures:int
- numberOfSymbolicFeatures:int
- pearsonCorrelationCoefficient::float
- spearmanCorrelationCoefficient::float
- covarianceWithTarget::float
+ percentageOfBinaryFeatures()
+ percentageOfNumericFeatures()
+ percentageOfSymbolicFeatures()
hasImplementation
hasTask
evaluateAnalysis
Landmarker
hasNominalFeaturesAnalysis
Implementation
- name:String
- description:String
- sourceCodeURL:String
- languageProgram:String
{Algorithm = DeepLearning}
Software
- name:String
- description:String
- version:String
Algorithm
- name:String
- description:String
AlgoSupervised
AlgoUnsupervised
AlgoReinforcement
usesAlgo
Task
- name:String
- description: String
hasParameter
Parameter
- name:String
hasParameterSetting
ParameterSetting
- value:TypeValue
hasParameterValue
DeepLearning
define
hasProcedure
ModelEvaluation
- value:float
EvaluationMeasure
- name:String
- description:String
EvaluationSpecification
- name:String
- description:String
EvaluationProcedure
- name:String
- description:String
useEvaluationMeasure
hasSpecification

Figure 5.3: Metadata on data analysis

- In order to help users understand how an analysis is preformed to facilitate the algorithm selection or parameter setting, we propose **implementation metadata** which concerns technical information of analysis. These metadata are modeled through classes *Implementation*, *Landermarker*, *Software*, *Algorithm*, *Parameter* and *ParameterSetting* (marked in green).

  – *name* and *description* of class *Implementation* present the semantic information of implementations, they answer the question of "*what*".
  – sourceCodeURL of class *Implementation* indicates the location of implementation source code to answer the question of "*where*".
  – *languageProgram* of class *Implementation* and the information in class *Software* explain how the analysis is implemented.
  – information in classes *Algorithm*, *Parameter*, *ParameterSetting* shows the algorithms details of an implementation to answer the question "*how*".

- In order to help users to check the result of an analysis to let them evaluate different implementations and easily make decisions on algorithm selection and parameter selection, we propose **evaluation metadata**. These metadata answer the question of "*what result*" and they are modeled through classes *OutputModel*, *ModelEvaluation*, *EvaluationMeasure*, *EvaluationSpecification* and *EvaluationProcedure* (marked in yellow).
- In order to help users to find relevant tables to enrich their analysis or to help them to find possibly useful implemented analyses, we propose the dataset relationship metadata (marked in pink). These metadata are introduced in section 3.2.

A metadata management system is developed which allows users to search analyses and consult different categories of metadata easily (see a use case in section 6.4.3).

5.2.2 Example of Metadata Instantiation

To demonstrate our metadata model, we introduce an example of data analysis cases. The analyzed dataset is the *breast cancer* dataset which is transformed from the *CHSI* dataset (we introduced the *CHSI* dataset in section 3.3 and the preparation process of the creation of breast cancer in section 4.4.)

The first part of the example concerns the dataset and attribute metadata. The dataset *breast cancer* contains only one one-dimension table which has six attributes (*CHSI_County_Name,CHSI_State_Name,Obesity,High_Blood_Pres,Smoker,Brst_Cancer*) and 3141 instances. The statistical metadata are presented in dataset and attribute objects. The attribute relationships are calculated for the dataset and the dataset relationships are calculated among three datasets (breast cancer, lung cancer, colon cancer datasets). In Fig. 5.4, we present the object diagram of metadata of *breast cancer* dataset, due to the page limit, we only show details of one numeric attribute, one nominal attribute, one relationship of datasets (dissimilarity) and one relationship of attributes (Pearson correlation).

The second part of the example concerns an analysis performance. In this example, users carried out an analysis about the *breast cancer* dataset. For this analysis, the attributes *county*, *state*, *obesity*, *high blood pressure* and *smoker* are selected as features and *breast cancer* is selected as the class/target. The statistical and distribution information are calculated to instantiate the *AnalysisFeaturesm*, *AnalysisNumericFeatures*, and *AnalysisNominalFeatures* classes. Moreover, the algorithm KNN with parameters *n_neighbors=10* and *test_size=0.2* is used to analyze the dataset. An object diagram which contains all the above metadata of the analysis is shown in Fig. 5.5.

The presented metadata model is dedicated to data analysis of data lakes. It is a complete model that contains not only descriptive information of datasets and attributes

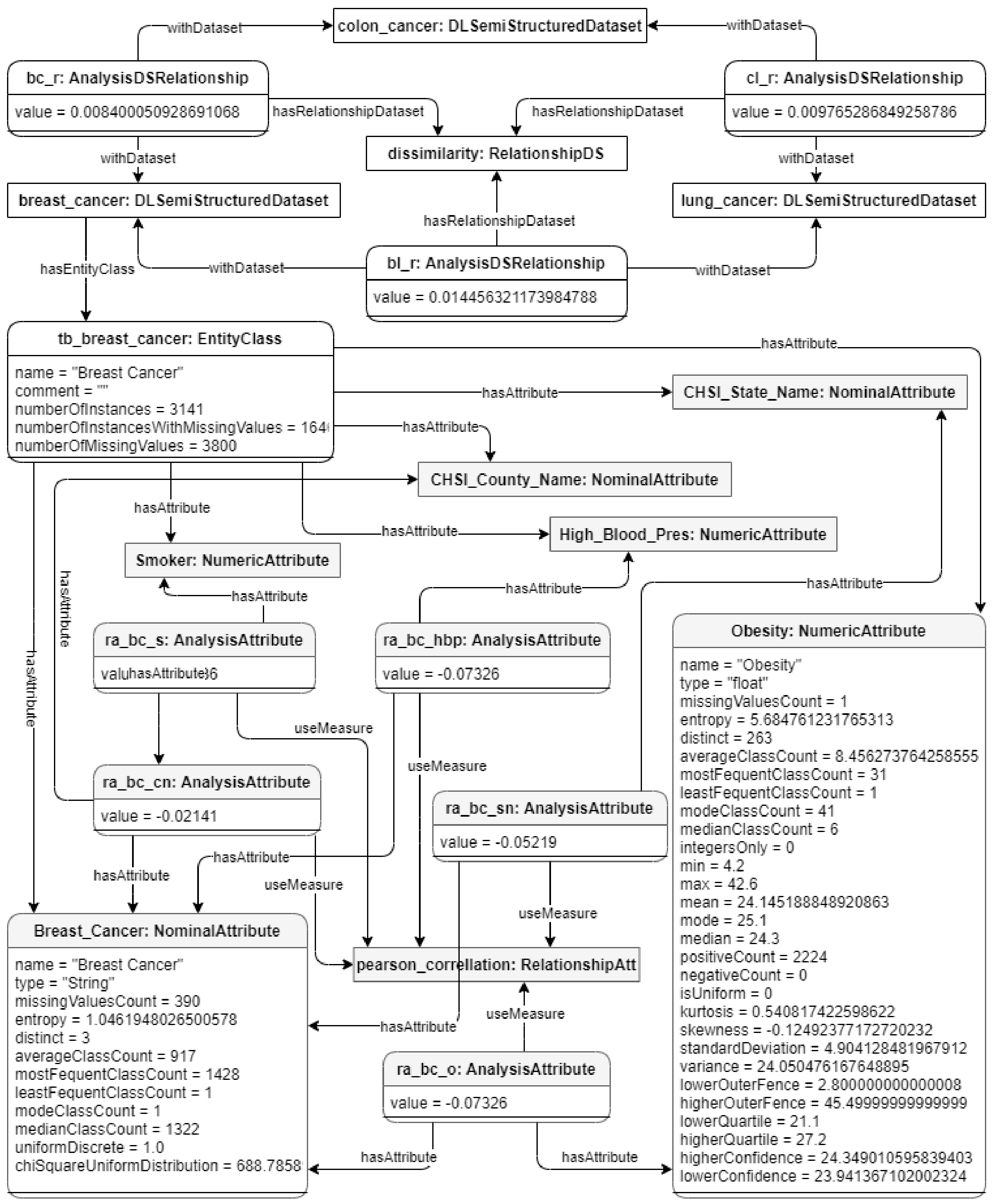

colon_cancer: DLSemiStructuredDataset
withDataset
bc_r: AnalysisDSRelationship
value = 0.008400050928691068
hasRelationshipDataset
dissimilarity: RelationshipDS
cl_r: AnalysisDSRelationship
value = 0.009765286849258786
breast_cancer: DLSemiStructuredDataset
lung_cancer: DLSemiStructuredDataset
hasEntityClass
bl_r: AnalysisDSRelationship
value = 0.014456321173984788
tb_breast_cancer: EntityClass
name = "Breast Cancer"
comment = ""
numberOfInstances = 3141
numberOfInstancesWithMissingValues = 164
numberOfMissingValues = 3800
hasAttribute
CHSI_State_Name: NominalAttribute
CHSI_County_Name: NominalAttribute
High_Blood_Pres: NumericAttribute
Smoker: NumericAttribute
ra_bc_s: AnalysisAttribute
value = -0.07326
ra_bc_hbp: AnalysisAttribute
value = -0.07326
ra_bc_cn: AnalysisAttribute
value = -0.02141
ra_bc_sn: AnalysisAttribute
value = -0.05219
useMeasure
pearson_correllation: RelationshipAtt
ra_bc_o: AnalysisAttribute
value = -0.07326
Breast_Cancer: NominalAttribute
name = "Breast Cancer"
type = "String"
missingValuesCount = 390
entropy = 1.0461948026500578
distinct = 3
averageClassCount = 917
mostFequentClassCount = 1428
leastFequentClassCount = 1
modeClassCount = 1
medianClassCount = 1322
uniformDiscrete = 1.0
chiSquareUniformDistribution = 688.7858
Obesity: NumericAttribute
name = "Obesity"
type = "float"
missingValuesCount = 1
entropy = 5.684761231765313
distinct = 263
averageClassCount = 8.456273764258555
mostFequentClassCount = 31
leastFequentClassCount = 1
modeClassCount = 41
medianClassCount = 6
integersOnly = 0
min = 4.2
max = 42.6
mean = 24.145188848920863
mode = 25.1
median = 24.3
positiveCount = 2224
negativeCount = 0
isUniform = 0
kurtosis = 0.540817422598622
skewness = -0.12492377172720232
standardDeviation = 4.904128481967912
variance = 24.050476167648895
lowerOuterFence = 2.800000000000008
higherOuterFence = 45.49999999999999
lowerQuartile = 21.1
higherQuartile = 27.2
higherConfidence = 24.349010595839403
lowerConfidence = 23.941367102002324

Figure 5.4: Object diagram of breast cancer analysis - datasets and attributes

(statistics on attribute values, management of missing values and relationships between datasets and attributes), but also analytical information on performed analyses of these datasets (studies, analyses, implementations with used algorithms and parameters and evaluations of result). All these metadata can facilitate data analysis by providing statistical information of dataset values to help users select features and class/target and providing performed analyses experience information to help users to choose algorithms and parameters.

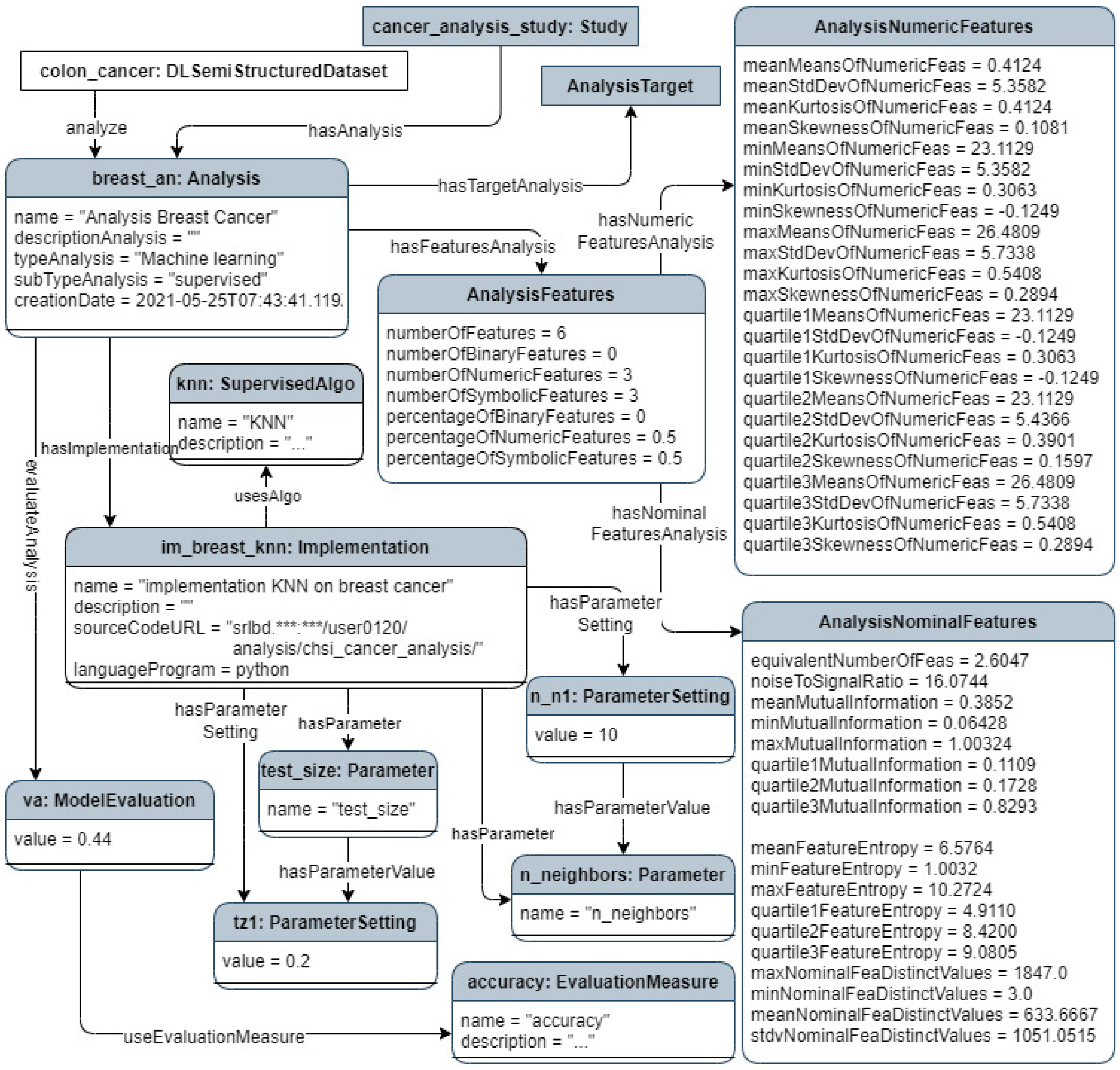

cancer_analysis_study: Study
colon_cancer: DLSemiStructuredDataset
AnalysisTarget
analyze
hasAnalysis
breast_an: Analysis
name = "Analysis Breast Cancer"
descriptionAnalysis = ""
typeAnalysis = "Machine learning"
subTypeAnalysis = "supervised"
creationDate = 2021-05-25T07:43:41.119
hasTargetAnalysis
hasFeaturesAnalysis
hasNumeric FeaturesAnalysis
AnalysisFeatures
numberOfFeatures = 6
numberOfBinaryFeatures = 0
numberOfNumericFeatures = 3
numberOfSymbolicFeatures = 3
percentageOfBinaryFeatures = 0
percentageOfNumericFeatures = 0.5
percentageOfSymbolicFeatures = 0.5
AnalysisNumericFeatures
meanMeansOfNumericFeas = 0.4124
meanStdDevOfNumericFeas = 5.3582
meanKurtosisOfNumericFeas = 0.4124
meanSkewnessOfNumericFeas = 0.1081
minMeansOfNumericFeas = 23.1129
minStdDevOfNumericFeas = 5.3582
minKurtosisOfNumericFeas = 0.3063
minSkewnessOfNumericFeas = -0.1249
maxMeansOfNumericFeas = 26.4809
maxStdDevOfNumericFeas = 5.7338
maxKurtosisOfNumericFeas = 0.5408
maxSkewnessOfNumericFeas = 0.2894
quartile1MeansOfNumericFeas = 23.1129
quartile1StdDevOfNumericFeas = -0.1249
quartile1KurtosisOfNumericFeas = 0.3063
quartile1SkewnessOfNumericFeas = -0.1249
quartile2MeansOfNumericFeas = 23.1129
quartile2StdDevOfNumericFeas = 5.4366
quartile2KurtosisOfNumericFeas = 0.3901
quartile2SkewnessOfNumericFeas = 0.1597
quartile3MeansOfNumericFeas = 26.4809
quartile3StdDevOfNumericFeas = 5.7338
quartile3KurtosisOfNumericFeas = 0.5408
quartile3SkewnessOfNumericFeas = 0.2894
knn: SupervisedAlgo
name = "KNN"
description = "..."
hasImplementation
evaluateAnalysis
usesAlgo
hasNominal FeaturesAnalysis
im_breast_knn: Implementation
name = "implementation KNN on breast cancer"
description = ""
sourceCodeURL = "srlbd.***:***/user0120/ analysis/chsi_cancer_analysis/"
languageProgram = python
hasParameter Setting
AnalysisNominalFeatures
equivalentNumberOfFeas = 2.6047
noiseToSignalRatio = 16.0744
meanMutualInformation = 0.3852
minMutualInformation = 0.06428
maxMutualInformation = 1.00324
quartile1MutualInformation = 0.1109
quartile2MutualInformation = 0.1728
quartile3MutualInformation = 0.8293
meanFeatureEntropy = 6.5764
minFeatureEntropy = 1.0032
maxFeatureEntropy = 10.2724
quartile1FeatureEntropy = 4.9110
quartile2FeatureEntropy = 8.4200
quartile3FeatureEntropy = 9.0805
maxNominalFeaDistinctValues = 1847.0
minNominalFeaDistinctValues = 3.0
meanNominalFeaDistinctValues = 633.6667
stdvNominalFeaDistinctValues = 1051.0515
n_n1: ParameterSetting
value = 10
hasParameter Setting
hasParameter
test_size: Parameter
name = "test_size"
va: ModelEvaluation
value = 0.44
hasParameterValue
hasParameter
n_neighbors: Parameter
name = "n_neighbors"
hasParameterValue
tz1: ParameterSetting
value = 0.2
accuracy: EvaluationMeasure
name = "accuracy"
description = "..."
useEvaluationMeasure

Figure 5.5: Object diagram of breast cancer analysis - implementation

5.3 Generation of the Analysis Metadata

To facilitate the work of users, the data science process including analysis metadata generation and maintenance should be industrialized. To achieve this objective, we propose a standardized process of analysis metadata generation through algorithms. With the three algorithms, we can not only differentiate metadata that should be entered by hand and those can be detected automatically, but also explain how metadata are generated and stored.

Algorithm 6: DatasetAttributesMetadata

```
   Input: connectionURL, descriptionDS
   Data: newDS
   /* store dataset properties */
1  $datalakeDatasetProperties \leftarrow$ createProperties('description',
     getDatasetName(newDS), 'connectionURL', $connectionURL$, 'size',
     getFileSize(newDS))
2  $datalakeDataset \leftarrow$ createNode('DatalakeDataset', $datalakeDatasetProperties$)
   /* store schematic metadata */
3  $datasetStruct \leftarrow$ getDatasetStructurality(connexionURL)
4  if ($datasetStruct.type$ = "structured" or "semi-structured") then
5      $entities[] \leftarrow$ getEntityClasses(newDataset)
6      foreach $e \subset entities$ do
7          $atts[] \leftarrow$ getAttributes($e$)
8          $entityClassProperties \leftarrow$ createProperties('name', getEntityName($e$),
             getEntityStatistics($atts[]$))
           // function getEntityStatistics() returns an array including the name and
              value of each statistical metadata
9          $entityClass \leftarrow$ createNode('EntityClass', $entityCLassProperties$)
10         createRelation('DatalakeDateset-EntityClass',
             $datalakeDataset, entityClass$)
11         foreach $att \subset atts[]$ do
12             if $getAttType(att)$ = 'numeric' then
13                 $numericAttributeProperties \leftarrow$ createProperties('name',
                     getAttName($att$), getNumericAttStat($att$)) $attribute \leftarrow$
                     createNodeNeo4j('NumericAttribute',
                     $numericAttributeProperties$)
14             else
15                 $nominalAttributeProperties \leftarrow$ createProperties('name',
                     getAttName($att$), getNominalAttStat($att$))
16                 $attribute \leftarrow$ createNode('NominalAttribute',
                     $nominalAttributeProperties$)
17             createRelation('EntityClass-Attribute]', $entityClass, attribute$)
           /* For each predefined RelationshipAtt we calculate the value of
              relationship between attributes */
18         $analysisAttributes[] \leftarrow$ getAnalysisAttribute(atts[], $relationshipAtts[]$)
19         foreach $an \subset analysisAttributes[]$ do
20             $relationArr \leftarrow$ createNode('AnalysisAttribute',
                 createProperties('value', $an.value$)
21             createRelation('AnalysisAttribute-Attribute',
                 $relationArr, an.attribute1$)
22             createRelation('AnalysisAttribute-Attribute',
                 $relationArr, an.attribute2$)
23             createRelation('AnalysisAttribute-RelationshipAtt',
                 $relationArr, an.relationshipAtt$)
```

```
24 else
25     addAttToNeo4jNode($datalakeDataset$, getDatasetFormat(newDS))
   /* For each predefined RelationshipDS we calculate the value of relationship
      between datasets                                                          */
26 $analysisDSRelationships[] \leftarrow$
     getAnalysisDSRelation($datalakeDataset, datalakeDatasets[], relationshipDSs[]$)

27 foreach $anDs \subset analysisDSRelationships[]$ do
28     $relationDs \leftarrow$ createNode('AnalysisDSRelationship', createProperties('value',
         $anDs.value$))
29     createRelation('AnalysisDSRelationship-DatalakeDataset',
         $relationDs, anDs.datalakeDataset1$)
30     createRelation('AnalysisDSRelationship-DatalakeDataset',
         $relationDs, anDs.datalakeDataset2$)
31     createRelation('AnalysisDSRelationship-RelationshipDS',
         $relationDs, anDs.relationshipDS$)
```

The first algorithm has the objective of generating analysis metadata of datasets and attributes (see Algo. 6). Users need to input dataset location or connection URL and a description, so that the system stores the location and description metadata of the dataset (*line 1-2*). Then data structural type is detected (*line 3*). For a structured or semi-structured dataset, the system retrieves all its tables/entities (*line 4-5*), and for each of them (i) calculates statistical information of dataset values (*line 6-10*); (ii) calculates different statistical information of each attribute (*line 11-17*) according to its type (numeric or nominal); (iii) relationships between different attributes (*line 18-23*). For an unstructured dataset, only its format is detected and stored (*line 24-25*). Finally, the relationships between different datasets are calculated (*line 26-31*).

The second algorithm is for collecting pre-analyzing metadata which concern features/-target analysis and landmarker information (see Algo. 7). For each business requirement, users can create a study. Within a study, users can perform different analyses. For each analysis, users need to describe the analysis with a name and a description, indicate the used dataset, choose class/target attribute and features. All the input information is stored in the system (*line 1-8*). Moreover, the system calculates automatically statistical metadata of the set of features (*line 9-14*). Besides the statistical information, the system can perform predefined or user defined basic learning algorithms (landmarkers) and store the relative metadata (*line 15-25*).

The third algorithm is for generating metadata of implementations of analyses (see Algo. 8). The system provides an interface with which users can input descriptive information, for instance, name and description of implementation, preformed algorithm with the set of parameters. The system can store all these metadata appropriately in the metadata database.

Algorithm 7: Pre_analysis

```
Input: $studyName, studyDesc, datalakeDataset, targetAtt, features[],$
       $analysisName, analysisDesc$
Data: selected dataset $ds$
/* calculate statistics for analysis */
1  Function statisticsAnalysis($study, datalakeDataset, targetAtt, features[],$
    $analysisName, analysisDesc$):
2      $analysis$ ← createAnalysis('Analysis', createProperties('name',
         $analysisName$, 'description', $analysisDesc$)
3      createRelation('Analysis-DatalakeDataset', $analysis$, $datalakeDataset$)
4      createRelation('Analysis-Study', $analysis$, $study$)
5      $analysisTarget$ ← createNode('AnalysisTarget')
6      createRelation('Analysis-AnalysisTarget', $analysis$, $analysisTarget$)
7      createRelation('AnalysisTarget-Attribute', $analysisTarget$, $targetAtt$)
8      $analysisFeatures$ ← createNode('AnalysisFeatures',
         createProperties(getStatAnalysisFeatures($features[]$)))
9      createRelation('Analysis-AnalysisFeatures', $analysis$, $analysisFeatures$)
10     $analysisNumericFeatures$ ← createNode('AnalysisNumericFeatures',
         createProperties(getStatAnalysisNumericFeatures($features[]$)))
11     createRelation('AnalysisNumericFeatures-AnalysisFeatures',
         $analysisNumericFeatures$, $analysisFeatures$)
12     $analysisNominalFeatures$ ← createNodeNeo4j('AnalysisNominalFeatures',
         createProperties(getStatAnalysisNominalFeatures($features[]$)))
13     createRelation('AnalysisNominalFeatures-AnalysisFeatures',
         $analysisNominalFeatures$, $analysisFeatures$)
14     return analysis
/* run predefined landmarkers */
15 Function runLandmarkers($study, datalakeDataset, targetAtt, features[]$):
16     $landmarkers[]$ ← getLandmarkers()
17     foreach $lm \subset landmarkers[]$ do
18         $analysis$ ← statisticsAnalysis($study$, $datalakeDataset$, $targetAtt$,
             $features[]$, $lm.name$, $lm.description$)
19         $output$ ← runAlgo($analysis$, $lm$)
20         $outputModel$ ← createNode('OutputModel', createProperties('name', ,
             'description', $output$))
21         createRelation('Analysis-OutputModel', $analysis$, $outputModel$)
22         foreach $m \subset getEvaluationMeasures()$ do
23             $modelEvaluation$ ← createNode('MoedlEvaluation',
                 createProperties('value', evaluate($m$, $analysis$))
24             createRelation('Analysis-ModelEvaluation', $analysis$,
                 $modelEvaluation$)
25             createRelation('ModelEvaluation-EvaluationMeasure',
                 $modelEvaluation, m$)
26 $study$ ← createNode('Study', createProperties('name', $studyName$, 'description',
    $studyDesc$))
27 runLandmarkers($study$, $datalakeDataset$, $targetAtt$, $features[]$)
```

Algorithm 8: Analyse_dataset

```
Input: $study, analysisName, analysisDesc, datalakeDataset, targetAtt, features[],$
       $sourceCode, algo, paras[], paraSets[]$
Data: selected dataset $ds$
1  $analysis$ ← statisticsAnalysis($study$, $datalakeDataset$, $targetAtt$, $features[]$,
     $analysisName$, $analysisDesc$)
2  $implementation$ ← createNode('Implementation', createProperties('sourceCode',
     sourceCode))
3  createRelation('Implementation-Analysis', $implementation, analysis$)
4  $algo$ ← createNode('Algorithm', createProperties('name', algo.name,
     'description', algo.description))
5  createRelation('Implementation-Algorithm', $implementation, algo$)
6  foreach $para \subset paras[]$ do
7      $parameter$ ← createNode('Parameter', createProperties('name', para))
8      createRelation('Implementation-Parameter', $implementation, parameter$)
9  foreach $paraSet \subset paraSets[]$ do
10     $parameterSetting$ ← createNode('ParameterSetting',
         createProperties('value', paraSet.value))
11     createRelation('Implementation-ParameterSetting',
         $implementation, parameterSetting$)
12     createRelation('ParameterSetting-Parameter', $parameterSetting$,
         getParameter($paraSet.para$))
13 $output$ ← runAlgo($analysis, algo$)
14 $outputModel$ ← createNode('OutputModel', createProperties('name', ,
     'description', $output$))
15 createRelation('Analysis-OutputModel', $analysis, outputModel$)
16 foreach $m \subset$ $getEvaluationMeasures()$ do
17     $modelEvaluation$ ← createNode('MoedlEvaluation', createProperties('value',
         evaluate($algo, analysis$)))
18     createRelation('Analysis-ModelEvaluation', $analysis, modelEvaluation$)
19     createRelation('ModelEvaluation-EvaluationMeasure', $modelEvaluation, m$)
```

5.4 Discussion

In this chapter, we introduced a metadata model dedicated to data analysis and a formalized process of metadata generation.

Regarding metadata of data analysis, in the literature, the metadata challenge for data analysis is especially studied in the meta-learning field. Meta-learning solutions are able to learn from the past analysis experiences (Hutter et al., 2019) in order to improve the effectiveness of machine learning algorithms. In this paradigm, metadata describe learning tasks (algorithms) and previously learned models (results). These metadata concern, for instance, hyper-parameters of predictive models, pipelines of compositions or also meta-features.

In a high level vision of these works, a lack of consensus is emerging about the metadata to be used, not only in the machine learning context, but also in data analysis more generally. The works defining in a formal setting the classification of the relevant metadata

Our Schema	**Term from ML Schema**	**OntoDM-core**	**DMOP**
Task	Task	Data mining task	DM-Task
Algorithm	Algorithm	Data mining algorithm	DM-Algorithm
Software	Software	Data mining software	DM-Software
Implementation	Implementation	Data mining algorithm implementation	DM-Operator
Patamater	HyperParameter	Parameter	Parameter
PatamaterSetting	HyperParameterSetting	Parameter setting	OpParameterSetting
Study	Study	Investigation	
Study	Experiment		DM-Experiment
AnalysisEntityClass	Run	Data mining algorithm execution	DM-Operation
DatalakeDataset	Data	Data item	DM-Data
EntityClass	Dataset	DM dataset	DataSet
Attribute	Feature		Feature
DatalakeDataset.attributes	DataCharacteristic	Data specification	DataCharacteristic
EntityClass.attributes	DatasetCharacteristic	Dataset specification	DataSetCharacteristic
Attribute.attributes	FeatureCharacteristic	Feature specification	FeatureCharacteristic
OutputModel	Model	Generalization	DM-Hypothesis
OutputModel.attributes	ModelCharacteristic	Generalization quality	HypothesisCharacteristic
ModelEvaluation	ModelEvaluation	Generalization evaluation	ModelPerformance
EvaluationMeasure	EvaluationMeasure	Evaluation datum	ModelEvaluationMeasure
EvaluationSpecification	EvaluationSpecification		
EvaluationProcedure	EvaluationProcedure	Evaluation algorithm	ModelEvaluationAlgorithm
AnalysisDSRelationship			
AnalysisFeatures			
AnalysisAttribute			
Landmarker			

Table 5.1: Comparison of different ML schemas and ontologies

are scarce. The works of (Bilalli et al., 2016; Rivolli et al., 2018; Keet et al., 2015; Panov et al., 2014) and a W3C machine learning model (Esteves et al., 2016) investigate a classification of metadata related to data analysis.

Regarding machine learning model, different ontologies of data mining are proposed (Keet et al., 2015; Panov et al., 2014). (Keet et al., 2015) contains detailed descriptions of data mining tasks, data, algorithms, hyper-parameters and workflows. (Panov et al., 2014) defines data mining entities in three layers: specification, implementation and application. The W3C ML model (Esteves et al., 2016) is oriented to machine learning process. Nevertheless, the relationships between datasets and attributes are not considered whereas we intend to do it in the context of data lakes. However, one of the important advantages of data lakes is to cross multiple sources of data, so that the relationships between these data are crucial. Hence, adding relationship metadata is inevitable in order to facilitate the work of analysts by selecting and querying relevant datasets and corresponding attributes.

We dress in table 5.1 a comparison between our model and the basic W3C ML specification (Esteves et al., 2016) and ontologies schemas (Keet et al., 2015; Panov et al., 2014). We remark a compatibility between the approaches on the basic components of ML such as task algorithm or parameter. However, our model contains not only data modeling and algorithm metadata, but also the metadata generated during the data collection and preparation phases, including characteristics, statistical metadata, and relationships between datasets and between attributes. This information is essential to deal with the Feature Selection (FS) (Solorio-Fernández et al., 2020) problem. It also facilitates data analysis tasks by allowing users to broaden their perspective on the data by viewing the results of performed analyses (Al-Tashi et al., 2020).

Regarding formalized process of analysis metadata generation, to the best of our knowledge, we are the only one who offered standardized process to generate metadata in an semi-automatic way for the enhancement of the metadata model.

5.5 Conclusion

In this chapter, we focus on the metadata of data analysis in data lakes. Data analysis is the activity during which different users recover and examine data to discover useful information for decision-making. Data analysis is the last but not the least phase of data life-cycle in a data lake. Data lakes can facilitate the data analysis phase by helping users to search, analyze, reuse existing datasets and analyses to select features, choose algorithms, analysis models or establish reports. However, in the context of data lake, only few solutions focus on data analysis.

To overcome this shortfall, we proposed an analysis metadata solution that capitalizes all relevant experiences in a data lake to improve the efficiency of data analysis. Our solution is twofold: firstly, we proposed a complete and formalized metadata model that includes not only descriptive information on datasets and attributes (statistics, missing values and relationships between datasets and attributes), but also analytical information on performed analyses of these datasets (studies, tasks, implementations with algorithms and their parameter settings and evaluations of analyses). These metadata allow users to cross different elements in a data lake to get inspired by the existing analysis experiences to make better decisions on dataset, feature and algorithm selection. Secondly, in order to facilitate the metadata generation and maintenance tasks for users, we formalized the analysis metadata generation and we demonstrated it through a real example.

The metadata model and the metadata generation process are validated by our published work in IDEAS2021 (Zhao et al., 2021c).

We already proposed a complete data lake metadata solution which includes ingestion, preparation and analysis metadata. In the next chapter, we will introduce a developed system which allows users to generate and search all the proposed metadata for different data lake elements.

Chapter 6

Implementation

Contents

6.1 Introduction

The metadata management in a data lake has for objective to facilitate the findability, accessibility, interoperability, and reusability of different elements (datasets, preparation processes, analyses) stored in a data lake for different types of users (data engineers, scientists, analysts). For this purpose, we proposed metadata model dedicated to different aspects in previous chapters. To ensure that the model can adapt to real word scenarios, to guarantee that the proposed metadata can easily be generated and explored by all types of users and to verify if it respects the FAIR principles, we propose the **DAMMS** system, a **DA**ta lake **M**etadata **M**anagement **S**ystem[1].

DAMMS, as presented in Fig. 6.1, is positioned in the govern zone of data lake and can interact with the other DL zones and with different types of users. Regarding the users of a data lake, we find data lake managers, data engineers, data scientists, data analysts or simple data lake visitors. These users will participate in the development of different projects for which elements stored in the data lake are accessed and used. During their experiments on the data lake, DAMMS will be a support that allows users to generate the metadata corresponding to different stages of the life-cycle of data (ingestion, preparation and analysis). We can therefore designate these users as creators from the point of view of DAMMS. The same types of users can also rely on DAMMS to search metadata for exploring the data lake. We refer to these users as explorer when they take advantage of the metadata of the stored elements. Note that we distinguish creators and explorers according to the way that they use the data lake at a time, a user can change his role at different point of time. For instance, when an explorer have identified the datasets or the processes that he needs via the metadata exposed by DAMMS and he starts a new analysis, he will then take over the hat of creator and works on the underlying activities.

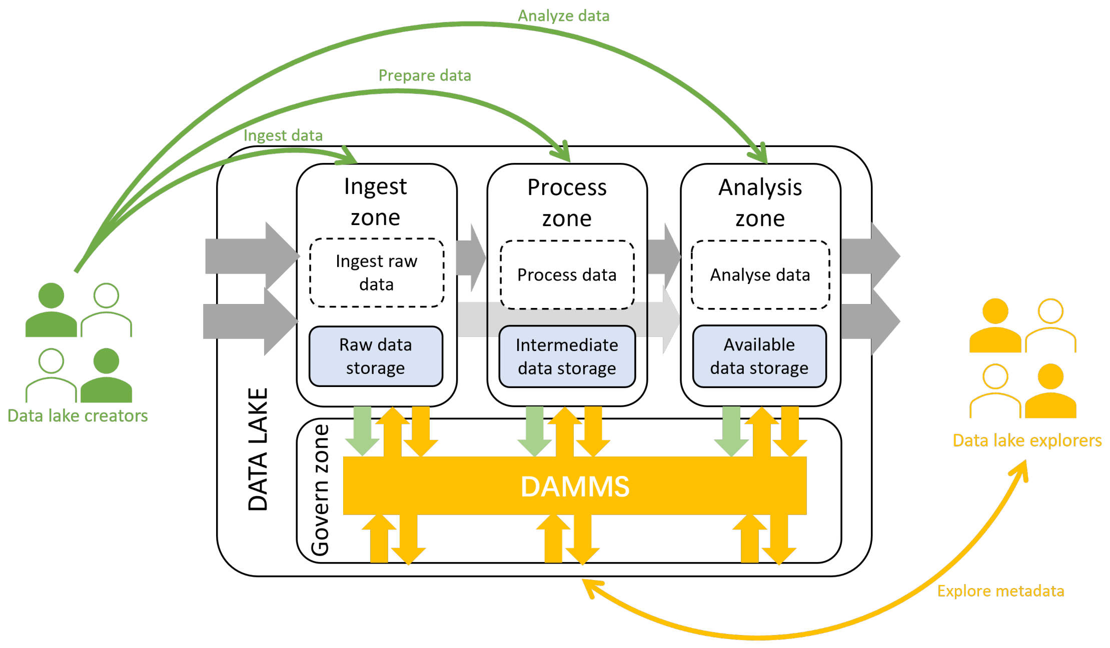


Figure 6.1: DAMMS in a data lake

In section 6.2, we introduce the system DAMMS and its functional and technical archi-

[1]https://github.com/yanzhao-irit/data-lake-metadata-management-system

tectures. In section 6.3, we introduce in details the semi-automatic metadata generation. In section 6.4, we present in details the exploration of metadata.

6.2 DAMMS : A Data Lake Metadata Management System

To ensure the usability of the system, we developed an application which allows users to generate metadata in a semi-automatic way and to explore all the existing elements in the data lake through intuitive search results. In the following, we will introduce the functional and technical architectures of DAMMS then we present the metadata storage system with details.

6.2.1 Functional Architecture of DAMMS

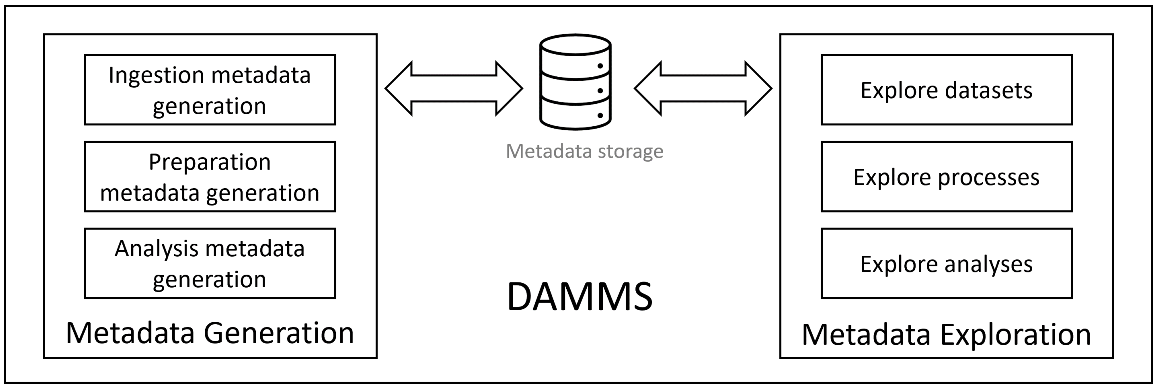


Figure 6.2: DAMMS - functional architecture

DAMMS has two principal functions (see Fig. 6.2):

- The first function aims to help creators to generate metadata. To ensure that all the information in the proposed model are covered, metadata are generated during all ingestion, preparation and analysis phases. To facilitate the tasks of users, metadata are generated in a semi-automatic way. All the generated metadata are stored in the metadata database in the data lake.
- The second function aims to help explorers to explore metadata. To ensure that users can find different elements and consult their details, the application allows users to find datasets, processes and analyses. For each element, users can find more different details depend on its type according to the description of different metadata in chapters 3-5. For instance, properties, lineage and relationships for datasets; parent / children processes and used main operations for preparation transformations; used algorithms, parameters and features for analyses.

6.2.2 Technical Architecture of DAMMS

The technical architecture of DAMMS is presented in Fig. 6.3. We chose to apply the client-server architecture to our application. This architecture is able to centralize the

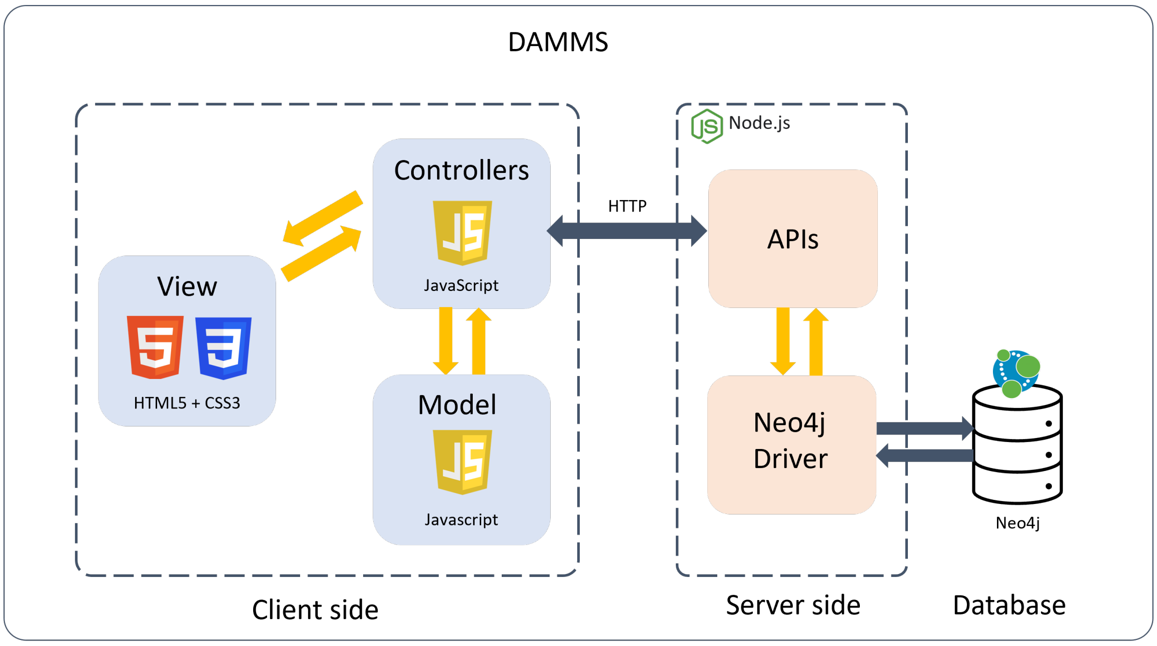


Figure 6.3: DAMMS - technical architecture

control of the application, to share resources from clients to servers, to reduce data replication and to ensure the security and scalability of the application (Sulyman, 2014).

The application is developed with the framework Electron. Electron is a framework dedicated to build cross-platform desktop applications using JavaScript. We chose this framework for the following reasons (Kredpattanakul and Limpiyakorn, 2019; Peguero and Cheng, 2021):

- It is an open-source project which is maintained by an large and active community.
- An Electron application can run on all major desktop operating systems, such as Windows, macOS, and Linux.
- It uses Chromium and Node.js so that the application can simply be built with HTML, CSS, and JavaScript.

For DAMMS, we built the application under the Electron framework with the front-end running in a Chromium browser and a back-end using the Node.js platform.

Regarding the front-end, the application interface is developed with JavaScript, HTML and CSS which are the core technologies of the World Wide Web.

Regarding the back-end, the environment Node.js is used. Node.js an open-source runtime that is built on Chrome's V8 JavaScript and is geared towards highly competitive event web applications. We chose Node.js for the following reasons:

- JavaScript is a common skill of web developers, Knowing JavaScript simplifies the learning of Node.js.
- Node.js provides different APIs to interact with the operating system. In DAMMS, different APIs (Application Programming Interface) are applied, for instance, the API node-postgres is used to connect to PostgreSQL database and generate metadata and node-oracle db is used to connect to ORACLE database and generate metadata.

Regarding the metadata storage, a graph database (Neo4j) is implemented. We will introduce more details on this point in the following subsection.

6.2.3 Metadata Storage

We chose to use a graph database (Neo4j) to store metadata for the reason that:

- Graph database has a good scalability. As introduced previously, in a data lake, a great amount of datasets can be stored, we need to ensure the volume and the velocity of metadata storage.
- Graph database has a good flexibility. We introduced metadata models for different elements stored in data lake in chapters 3-5, users can always extend or reduce the model to fit their requirements.
- The search depth is ensured when querying graph database. One of most important advantages of data lake is to search information across different datasets, processes and analyses, a graph database can facilitate the search without lots of self-joins.

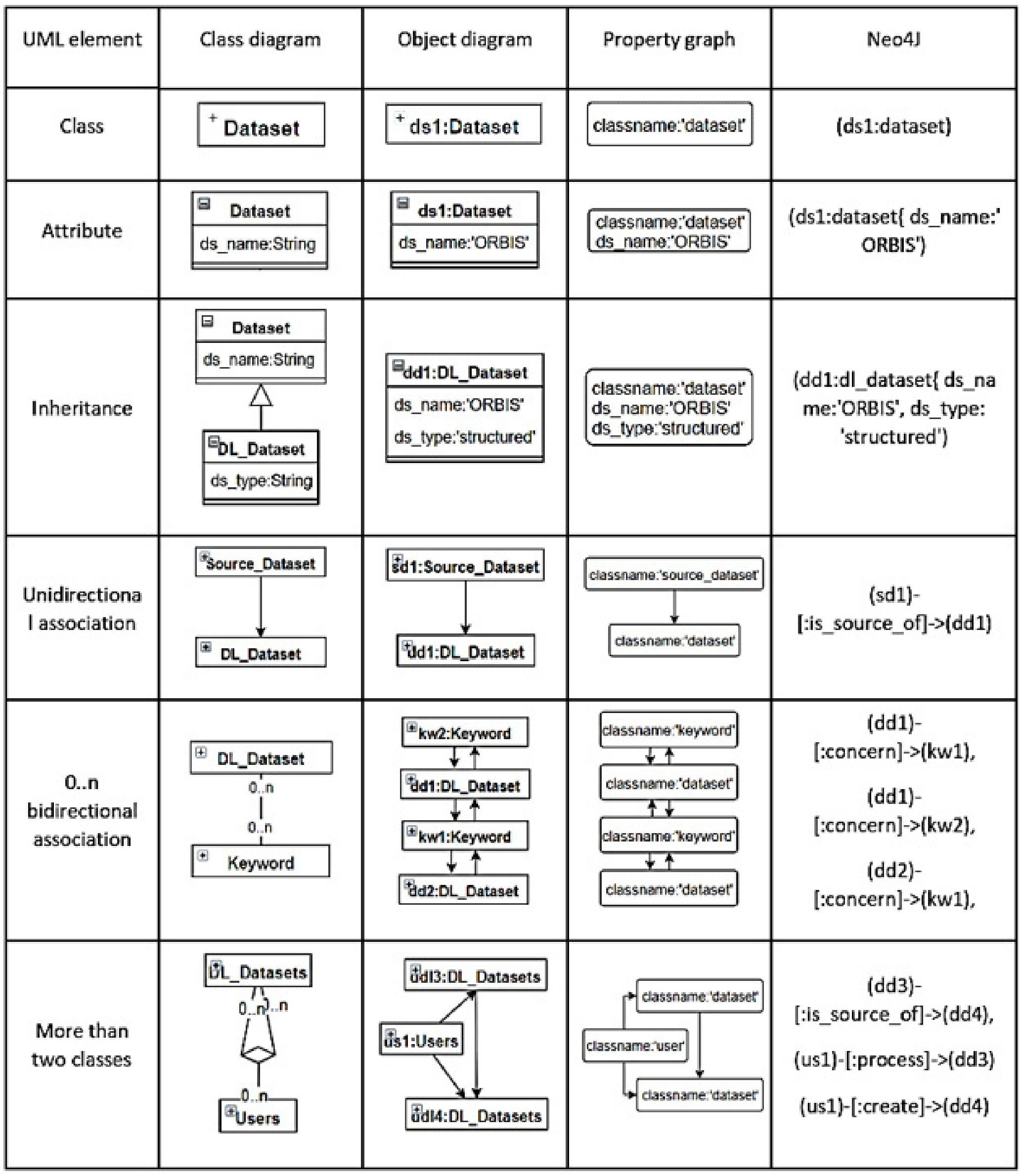
To model a graph, the following concepts are used (Lal, 2015): (i) *Nodes* represent entities, a graph database contains at least one node. (ii) *Labels* are used to group nodes into sets. (iii) *Relationships* are used to connect nodes. (iv) *Properties*, name-value pairs, are used to describe nodes.

A mapping from UML class diagram to property graphs was proposed by (Delfosse et al., 2012). We extended the mapping to Neo4j Cypher query language to transform and implement our model (see Fig. 6.4). All the *classes* become *labels*; all the instantiated *classes* become *nodes*; all the *attributes* become *properties* and *relationships* are used to connect nodes. Note that Neo4j does not support bidirectional relationship and direction can be ignored while querying, so that for bidirectional association, only one direction needs to be created. To illustrate the mapping from UML model to Neo4j database, we instantiated three examples that we have presented in the form of object diagram in the previous chapters (see Fig. 6.5-6.7).

6.3 Metadata Generation

Metadata generation has the objective of extracting metadata from data ingestion, preparation and analyses. So that users need to do as less as manual work to store as much as possible metadata. In the actual version of DAMMS, we provide functional interfaces that allow data lake users, in particular, creators to generate ingestion metadata. The algorithms for preparation and analysis metadata generation are implemented but not yet integrated in the front view interfaces.

According to our proposal in chapter 3, different types of datasets (structured, semi-structured and unstructured) can be ingested in the data lake. In section 3.2, we introduced different metadata that should be generated and stored in the system to ensure that metadata operators can find and reuse ingested datasets. For this objective, we developed different interfaces to ingest different types of datasets. With these interfaces, ingestion metadata are generated according to the processes proposed in section 3.3.



UML element	Class diagram	Object diagram	Property graph	Neo4J
Class	Dataset	ds1:Dataset	classname:'dataset'	(ds1:dataset)
Attribute	Dataset ds_name:String	ds1:Dataset ds_name:'ORBIS'	classname:'dataset' ds_name:'ORBIS'	(ds1:dataset{ ds_name:' ORBIS')
Inheritance	Dataset ds_name:String DL_Dataset ds_type:String	dd1:DL_Dataset ds_name:'ORBIS' ds_type:'structured'	classname:'dataset' ds_name:'ORBIS' ds_type:'structured'	(dd1:dl_dataset{ ds_na me:'ORBIS', ds_type: 'structured')
Unidirectiona l association	Source_Dataset DL_Dataset	sd1:Source_Dataset dd1:DL_Dataset	classname:'source_dataset' classname:'dataset'	(sd1)-[:is_source_of]->(dd1)
0..n bidirectional association	DL_Dataset 0..n 0..n Keyword	kw2:Keyword dd1:DL_Dataset kw1:Keyword dd2:DL_Dataset	classname:'keyword' classname:'dataset' classname:'keyword' classname:'dataset'	(dd1)-[:concern]->(kw1), (dd1)-[:concern]->(kw2), (dd2)-[:concern]->(kw1),
More than two classes	DL_Datasets 0..n 0..n 0..n Users	dd3:DL_Datasets us1:Users dd4:DL_Datasets	classname:'dataset' classname:'user' classname:'dataset'	(dd3)-[:is_source_of]->(dd4), (us1)-[:process]->(dd3) (us1)-[:create]->(dd4)

Figure 6.4: Mapping from UML class diagram to Neo4j Cypher language

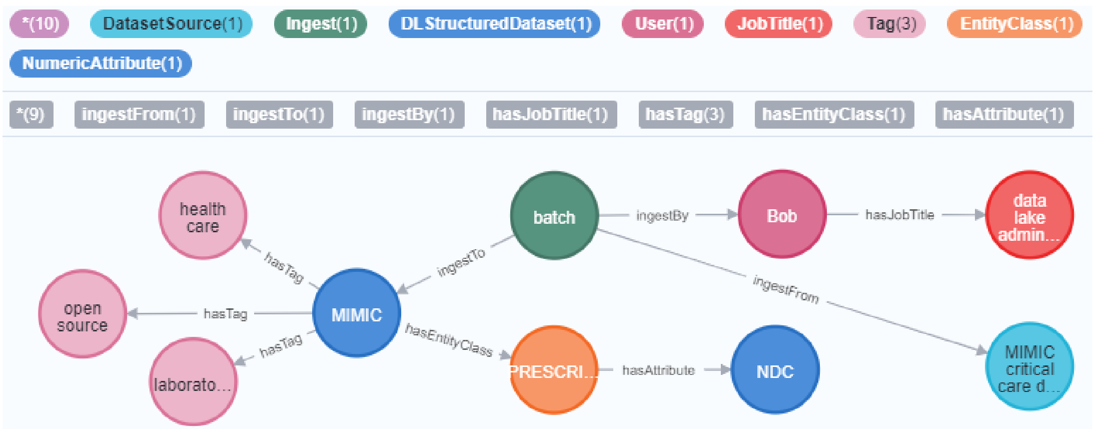


Figure 6.5: Mapping result of the object diagram 3.3

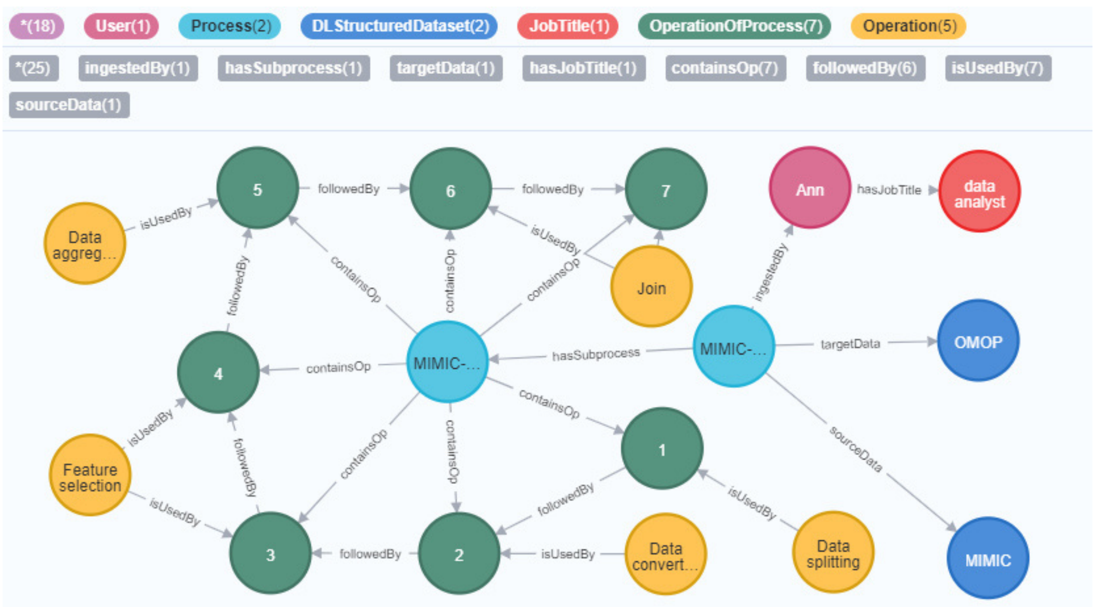


Figure 6.6: Mapping result of the object diagram 4.3

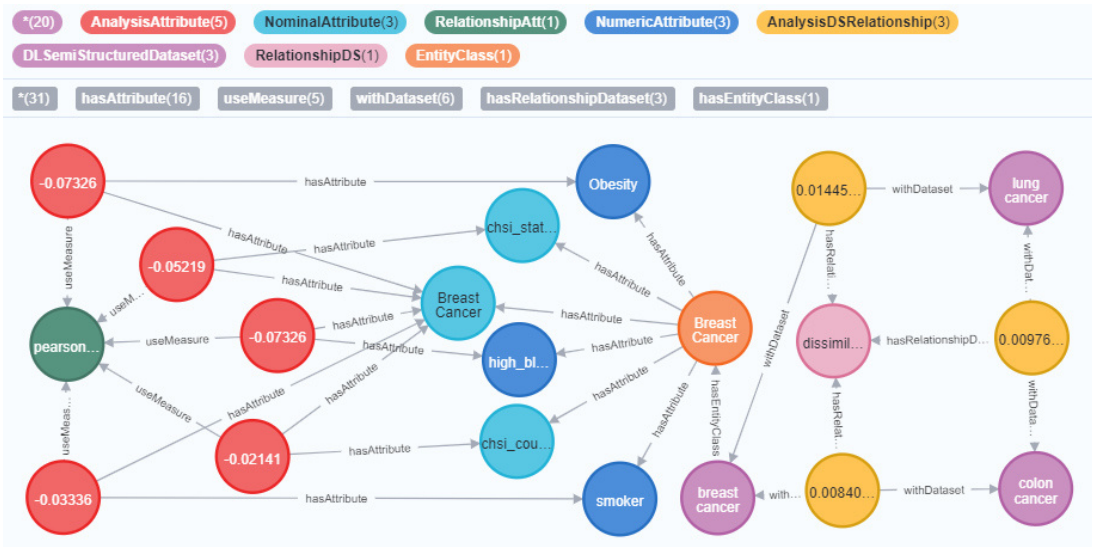


Figure 6.7: Mapping result of the object diagram 4.4

6.3.1 General Functioning

To generate metadata, the system should (i) firstly allow metadata creators to enter essential information, in particular, connection information and descriptive information to start the metadata generation and (ii) secondly extract database basic and schematic information automatically. For this purpose, we implemented functions that can generate partial metadata from PostgreSQL and ORACLE databases for the proof of concept.

6.3.1.1 Front-end

To start to generate dataset metadata, creators need to be able to (i) establish dataset connections and and (ii) add basic properties and semantic information (see Fig. 6.8).

Regarding the establishment of dataset connection, for **structured datasets** (PostgreSQL and ORACLE databases), users need to enter *user name*, *password*, *server host*,

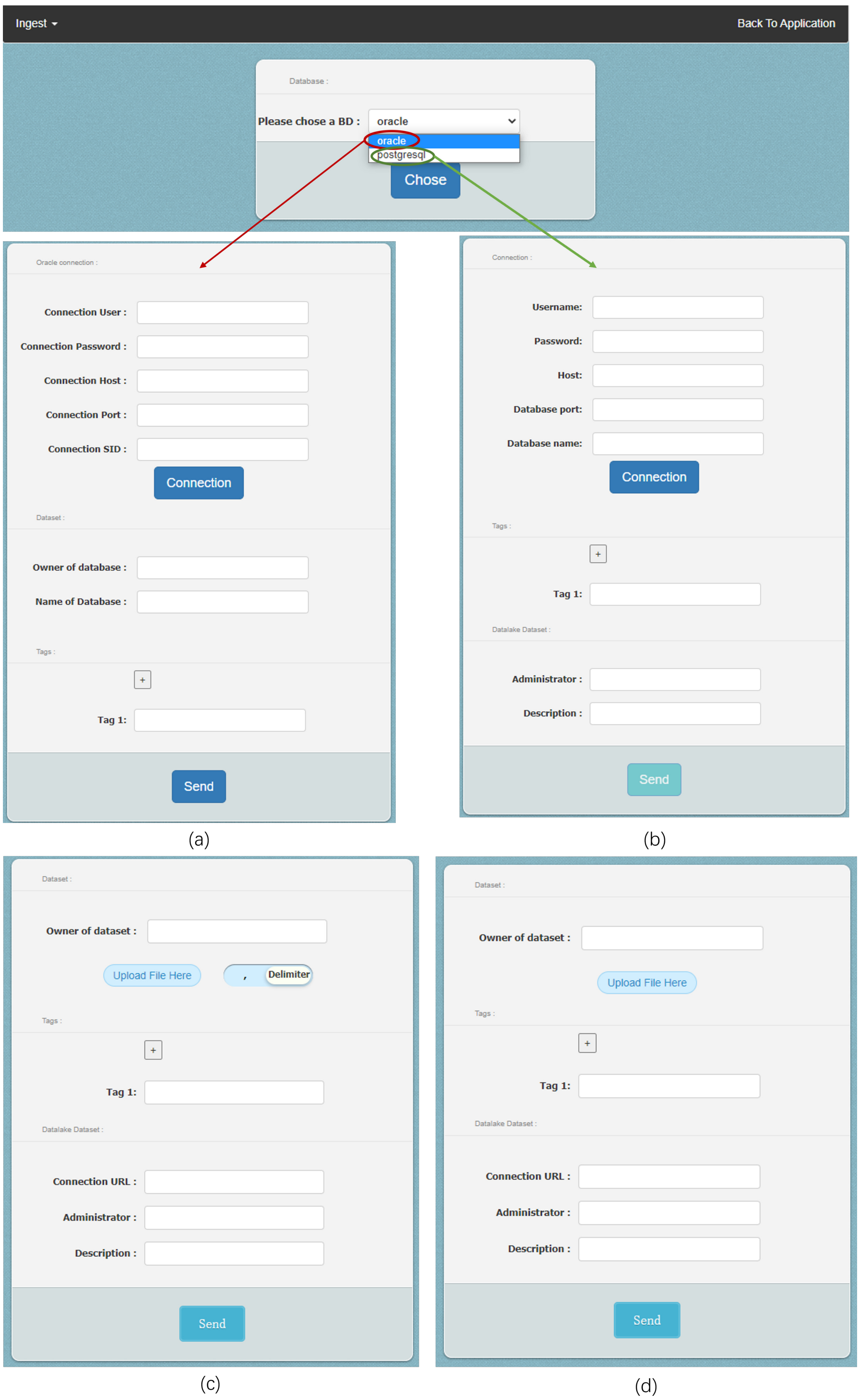


Figure 6.8: DAMMS interface - metadata generation

port and *database name* (see Fig. 6.8a,b). For **semi-structured datasets** (CSV files), users need to enter the *owner* and *location* of the dataset and choose the *delimiter* (see Fig. 6.8c). For **unstructured datasets**, users only need to enter the *owner* and *location* information (see Fig. 6.8(d)).

Regarding the basic properties and semantic information adding, for all three different types of datasets, users can add *tags* and *description*.

6.3.1.2 Back-end

Once users start the metadata generation in the front end, the back-end starts to (i) connect to the dataset, (2) extract metadata automatically and (iii) stored input and extracted metadata in the Neo4j database.

Regarding the dataset connection, for **structured datasets**, it is done with the help of different API. For instance, Oracle database is connected by *Oracledb*, PostgreSQL is connected by *pg*. For **semi-structured** and **unstructured datasets**, the API *filereader* is used.

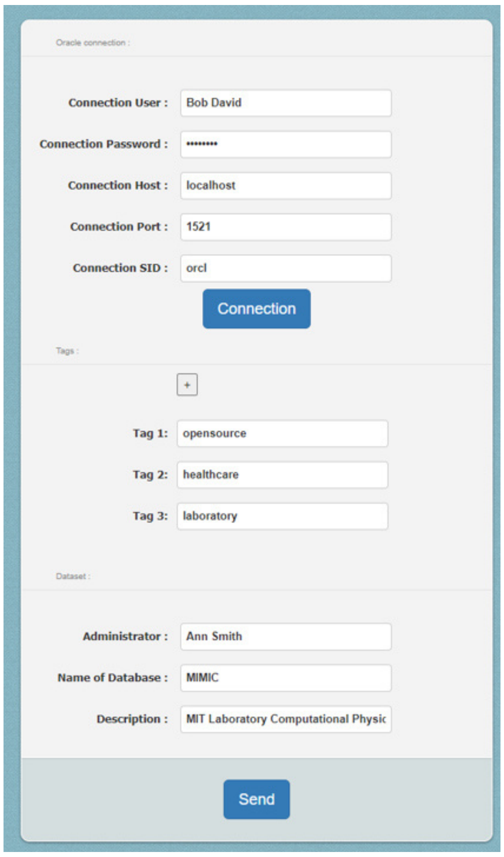
Regarding the metadata extraction, the same APIs are used to extract metadata. For **structured datasets**, different types of dataset metadata are extracted and calculated, such as basic properties (e.g. database name, type, location (server), creation date, size), schematic metadata (e.g. tables, table comment, attributes) and statistic metadata (e.g. number of attributes and instances, number of instances with missing value, number of missing value, nominal and numeric attribute count). Note that we extract the same metadata from ORACLE and PostgreSQL databases, but for ORACLE database, we use "*DBA_OBJECTS*" and "*DBA_TAB_COLUMNS*" to extract directly metadata such as max, min, missing value count wheres for PostgreSQL, we extract all the values and then calculate the same metadata. For **semi-structured datasets**, we extract almost the same metadata as structured dataset, the differences are that the name extension and format (e.g. music, image, video, etc.) are extracted for semi-structured datasets but not the table comment. For **unstructured datasets**, less metadata are extracted (*name*, *location*, *ingestion mode*, *file name extension*, *creation date*, *size*, *format*).

Regarding the metadata storage, the system stores all the automatically extracted and manually input metadata according to our proposed metadata model. To explain this funciton with more details, the script that stores metadata of ORACLE databases in the Neo4j database can be found in the Listing A.1) in annex A.

6.3.2 Example

To illustrate our design, we introduce an example of metadata generation for an ORACLE database *OT*[2] which is a database storing computer selling information. This database was ingested by Bob David. Once Bob entered all the information in the interface (see Fig. 6.9a), DAMMS tries to (i) connect to the database, (ii) extract metadata and (iii) store the input and extracted metadata in the Neo4j database (see an excerpt of the used Cypher query in listing A.2 in annex A). The result of the metadata generation is shown in Fig. 6.9b.

[2] `https://www.oracletutorial.com/getting-started/oracle-sample-database/`



Oracle connection :

Connection User :	Bob David
Connection Password :	••••••••
Connection Host :	localhost
Connection Port :	1521
Connection SID :	orcl

Connection

Tags :

+

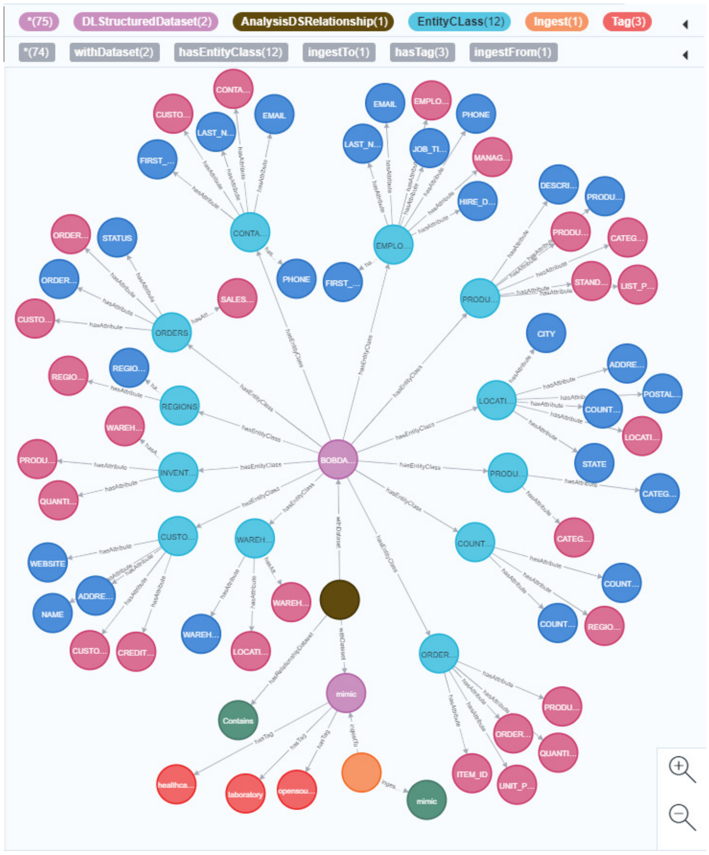
Tag 1:	opensource
Tag 2:	healthcare
Tag 3:	laboratory

Dataset :

Administrator :	Ann Smith
Name of Database :	MIMIC
Description :	MIT Laboratory Computational Physic

Send

(a) Interface of metadata generation



(b) Stored metadata

Figure 6.9: Metadata generation example - *OT* ORACLE database

6.4 Exploration of Metadata

The goal of the data lake exploration is to help explorers to model and analyze data more efficiently and efficaciously by facilitating the findability, accessibility, interoperability and reusability of existing elements (datasets, preparation processes and analyses). Such a system should make it possible to industrialize the use of data lake and implicitly ensure the FAIR recommendations when a large volume of data is shared between different users (data analysts, engineers and scientists).

To archive this goal, DAMMS is designed in such a way that allows users to explore everything stored in the data lake without getting lost in the amount of information provided. Indeed, to facilitate the search of users who have different skills, we provide two accesses in the system:

- The first access concerns graphical interfaces that dedicated to all users of the data lake (data scientists, statisticians and analysts) to then find, access and reuse existing elements. With this access, users only need to enter one or more keyword(s) to launch a search, the system will find all the elements (datasets, processes and analyses) whose name, description, related tags contain the entered keyword(s). We chose to apply keyword search for the reason that the result can correspond the best to the subject of user needs (Deng et al., 2015). In addition, to ensure that users can efficiently find useful information and easily understand selected elements with ergonomic interfaces, we chose to present the search result through a hierarchical navigation. The degree of depth of information seeking was also considered, as we will exemplify in sections 6.4.1-6.4.3.

- The second access concerns querying the metadata database directly for elaborated searches for specialists who have the skills of Neo4j querying language Cypher.

In the following, we present the different exploration facts while searching datasets, processes and analysis.

6.4.1 Explore Datasets

Datasets are the base of data preparation processes or analyses. To improve the efficiency and efficacy of the work of different types of data lake explorers (data analysts, scientists and engineers), datasets should be easily found and understood.

6.4.1.1 User Needs

When users explore datasets stored in the data lake, they should be able to find all the subject-related datasets they need. In addition, to avoid having too many results, they should be able to add filters such as created date and type of dataset to limit the search result. Moreover, if they are interested in a dataset, they should be able to look at all of its metadata for more detail so that they can decide weather or not to use the dataset.

To be more precised, users should find all the following information:

- Properties: the basic information of dataset, such as its name, type, size, owner and creation date, etc.
- Lineage: visualized data lineage which shows the origin of the dataset (dataset source) and all the ingestion, transformation processes until the selected dataset.
- Hyper-graph: the process that creates the dataset.
- Relationship of dataset: the already calculated relationships between the selected dataset and other datasets in the data lake. The predefined relationships include 'contain' (a relational database contain different schemas/users) and 'dissimilarity' (dissimilarity between different datasets), etc.
- Relationship of attribute: the calculated relationships between every two attributes in the selected structured or semi-structured dataset. Relationship can be predefined in the system, such as Pearson correlation coefficient, mutual information, Spearman correlation coefficient and covariance, etc. or can be defined by users.
- Similarity: Node Similarity algorithm from Neo4j Graph Data Science (gds) to calculate the similarity of nodes in the graph.

6.4.1.2 General Functioning

Front-end

To ensure the datasets search, DAMMS should have (i) a search bar with which explorers can use keywords to obtain a list of all corresponding datasets, (ii) filters that can be applied on dataset search to limit the result, such as dataset type (structured, semi-structured, unstructured) and creation date and (iii) a result panel where all the datasets that correspond to the search are listed (see Fig. 6.10).

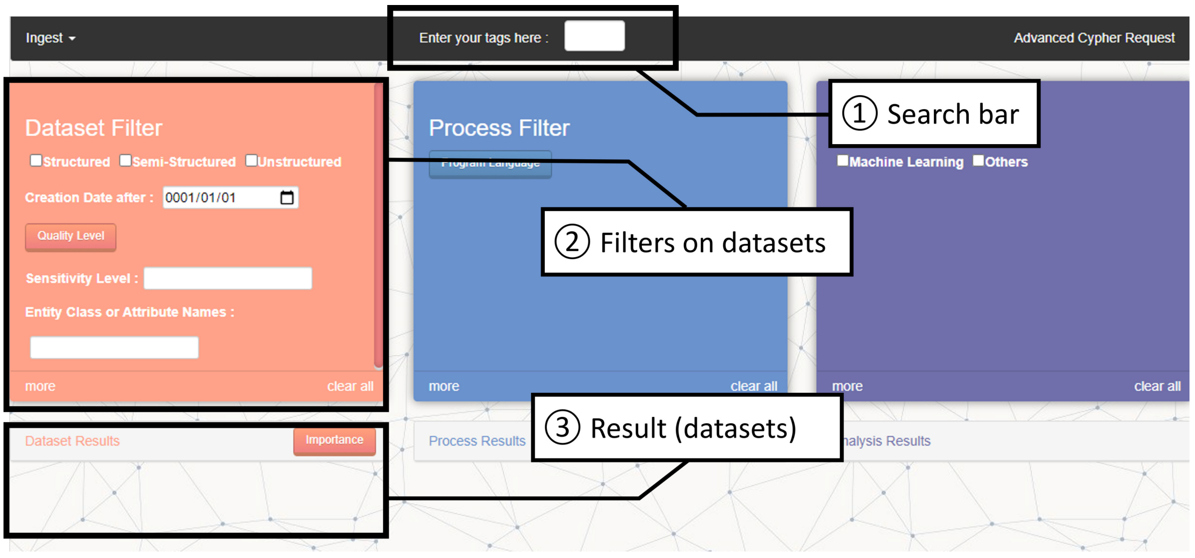
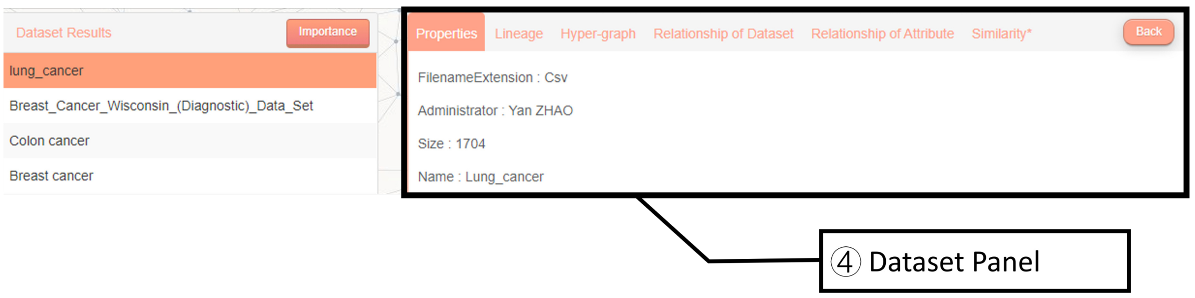


Figure 6.10: DAMMS - element search interface with filters on datasets



FilenameExtension : Csv
Administrator : Yan ZHAO
Size : 1704
Name : Lung_cancer

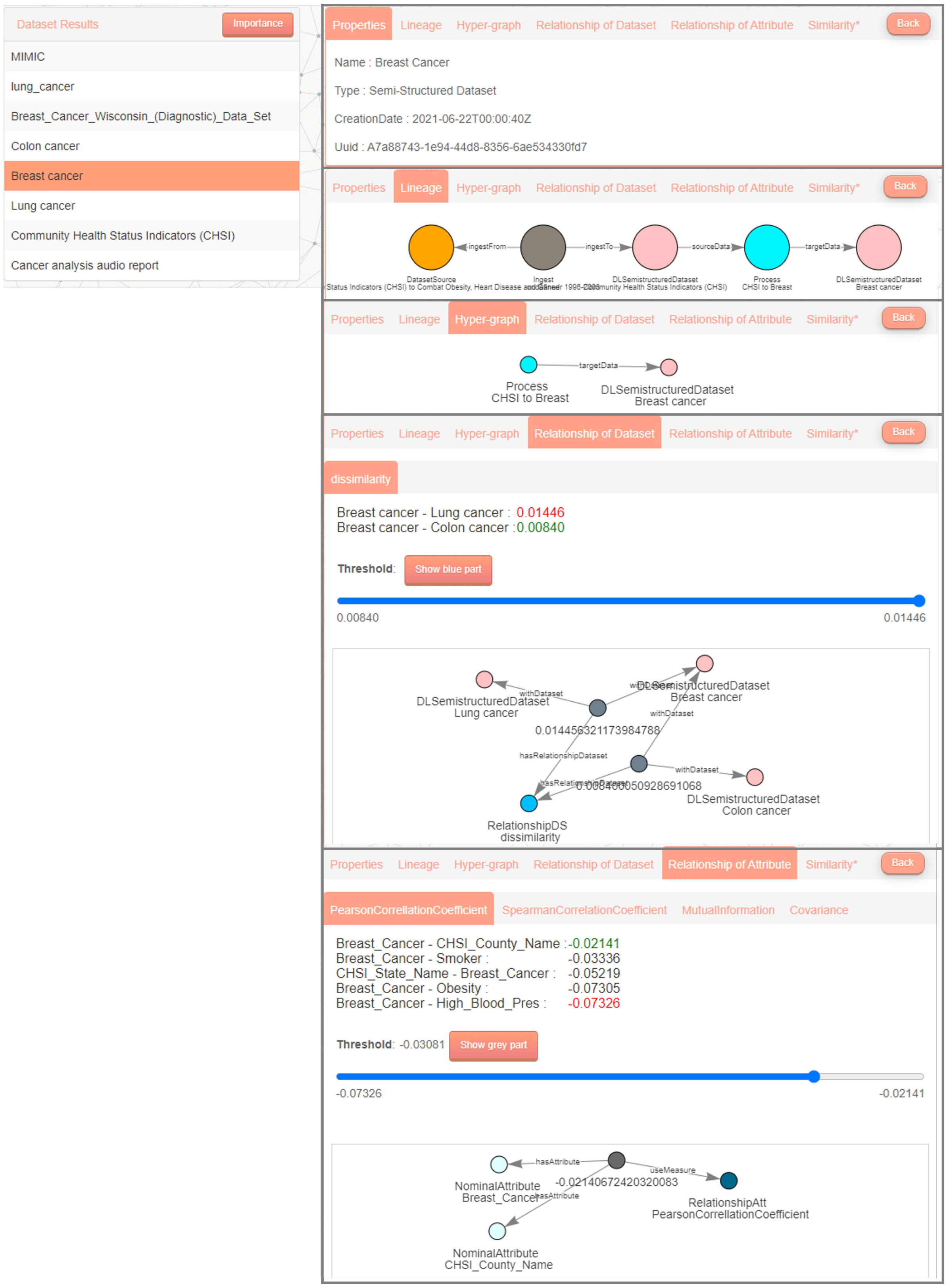
Figure 6.11: DAMMS - dataset panel

To ensure the datasets understanding, DAMMS uses a hierarchical navigation. The first level is the list of datasets presented with their name. The second level concerns different aspects information of each dataset. Each aspect of information is regrouped in one tab that contains the metadata that we proposed in section 3.2 (see Fig. 6.11 and an example in fig. 6.12).

Back-end

When users enter one or more keywords to start the dataset search, DAMMS will launch a query (see Listing B.1 in Annex B) to find all the datasets whose name, description or lined tags contain the input keyword(s) to ensure that all the relative datasets are listed and shown with their name in the result area. If different filters are added to limit the result, the above query will be modified by adding more conditions and be launched automatically to refresh the result (see annex B.1).

In the result list, if users want to find more details of one dataset, they can click on it. DAMMS will find and load the metadata about the selected dataset to display the dataset detail panel. Metadata are classified and displayed in different tabs in the panel. Different queries that are used to find metadata in the Neo4j database can be found in Listing B.2 in Annex B.



MIMIC
lung_cancer
Breast_Cancer_Wisconsin_(Diagnostic)_Data_Set
Colon cancer
Breast cancer
Lung cancer
Community Health Status Indicators (CHSI)
Cancer analysis audio report

Name : Breast Cancer
Type : Semi-Structured Dataset
CreationDate : 2021-06-22T00:00:40Z
Uuid : A7a88743-1e94-44d8-8356-6ae534330fd7

Breast cancer - Lung cancer : 0.01446
Breast cancer - Colon cancer : 0.00840

Threshold:
0.00840 0.01446

Breast_Cancer - CHSI_County_Name : -0.02141
Breast_Cancer - Smoker : -0.03336
CHSI_State_Name - Breast_Cancer : -0.05219
Breast_Cancer - Obesity : -0.07305
Breast_Cancer - High_Blood_Pres : -0.07326

Threshold: -0.03081
-0.07326 -0.02141

Figure 6.12: DAMMS - dataset result

6.4.1.3 An example of use case

The system allows users to easily find and consult existing datasets in the data lake via the interface. Our example concerns a data analyst user interested in cancer analysis. For this goal, he needs to find the useful datasets in the data lake to create a project in cancer analysis. To do so, he needs to firstly find all the datasets related to cancer in the data lake by launching the search with the keyword *'cancer'*. Secondly, he needs to chose the best dataset fitting with his business needs (see Fig. 6.12). He should (i) check the type and creation date of the selected dataset in the *properties* tab to exclude those that are not of the required type or are too old; (ii) check the origin of the dataset in the *lineage* tab to have more confidence; (iii) check the previous process created on the dataset in the *hyper-graph* tab; (iv) check the relationships between different attributes of the dataset in the *relationship of attribute* tab to have a first idea of the dataset content and (v) check the relationships between this dataset and other datasets in the *relationship of dataset* tab to find relevant datasets to enrich the dataset finding result.

6.4.2 Explore Data Preparation Processes

Data preparation exploration has for objective to help users to understand how a dataset is transformed and created and enable users to find and reuse existing processes to improve the data preparation effectiveness.

6.4.2.1 User Needs

When users explore preparation processes that are stored in the data lake, they should be able to find all the subject-related processes with a search bar where they can input keywords. Moreover, the result should be limited by filters to avoid having a too long list. For preparation process, different filters than those of datasets are proposed, such as program language, creation date, used operations and execution environment. In addition, if they are interested in a process, they should be able to consult the details of the process, including:

- Properties: basic information of the selected process, such as its name, description, creation date, binary machine code, source code URL and execution environment etc. This information helps users to have the general idea of a process.
- Lineage: visualized data lineage which shows the source and target dataset of a process. This information helps users to understand the life-cycle of processes and relevant datasets.
- Hyper-graph: for each process that contains sub-processes, show all its sub-processes; for each sub-process, show its parent process. This information helps users to understand the containing relationship of different processes.
- Operation: used operations of the selected process. This information helps users to understand the main activities of a process without reading its source code line by line.

6.4.2.2 General Functioning

Front-end

To ensure the process search, similar to dataset search, DAMMS also should have (i) a search bar with which explorers can use keywords to obtain a list of all corresponding processes, (ii) filters that can be applied on processes to limit the result, such as process language, creation date and used operation(s) and (iii) a result panel where all the processes that correspond to the search are listed (see Fig. 6.13).

To help users to understand a process, the result information is also displayed in different hierarchies. The first level is the list of processes presented with their name. The second level concerns different aspects of information of selected process, each aspect of information is presented in a tab in DAMMS. In different tabs, users can find different metadata that are introduced in section 4.2 (see Fig. 6.14 and an example in Fig. 6.15).

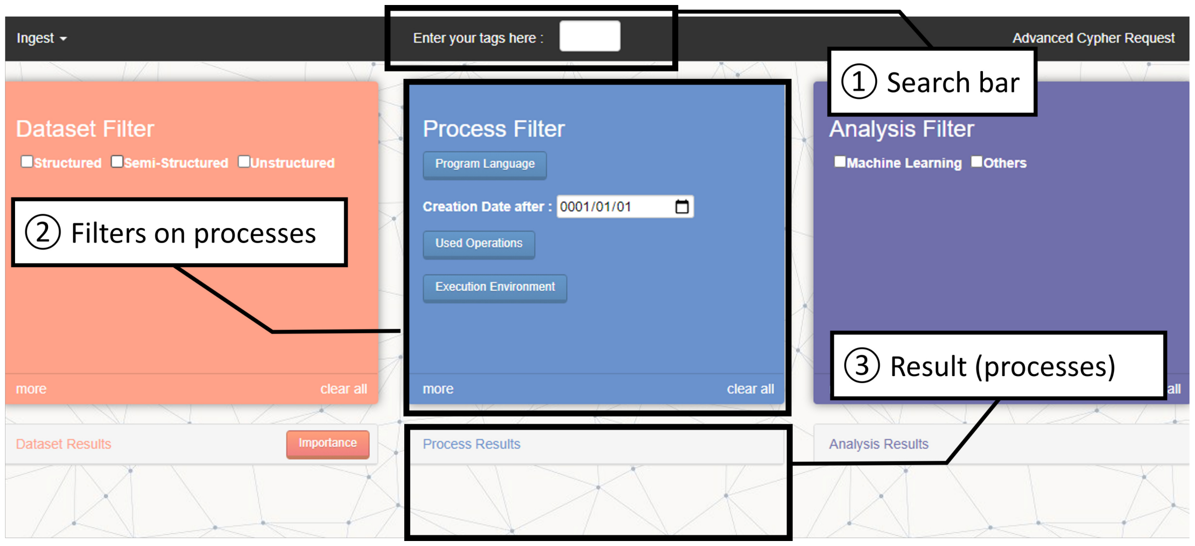
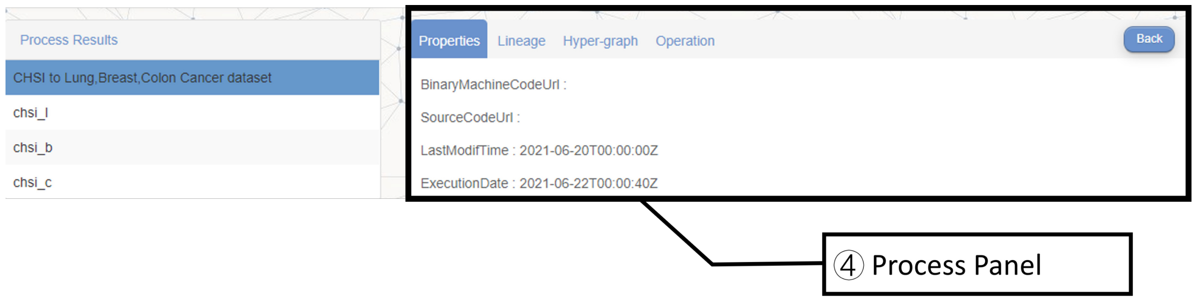


Figure 6.13: DAMMS - element search interface with filters on processes



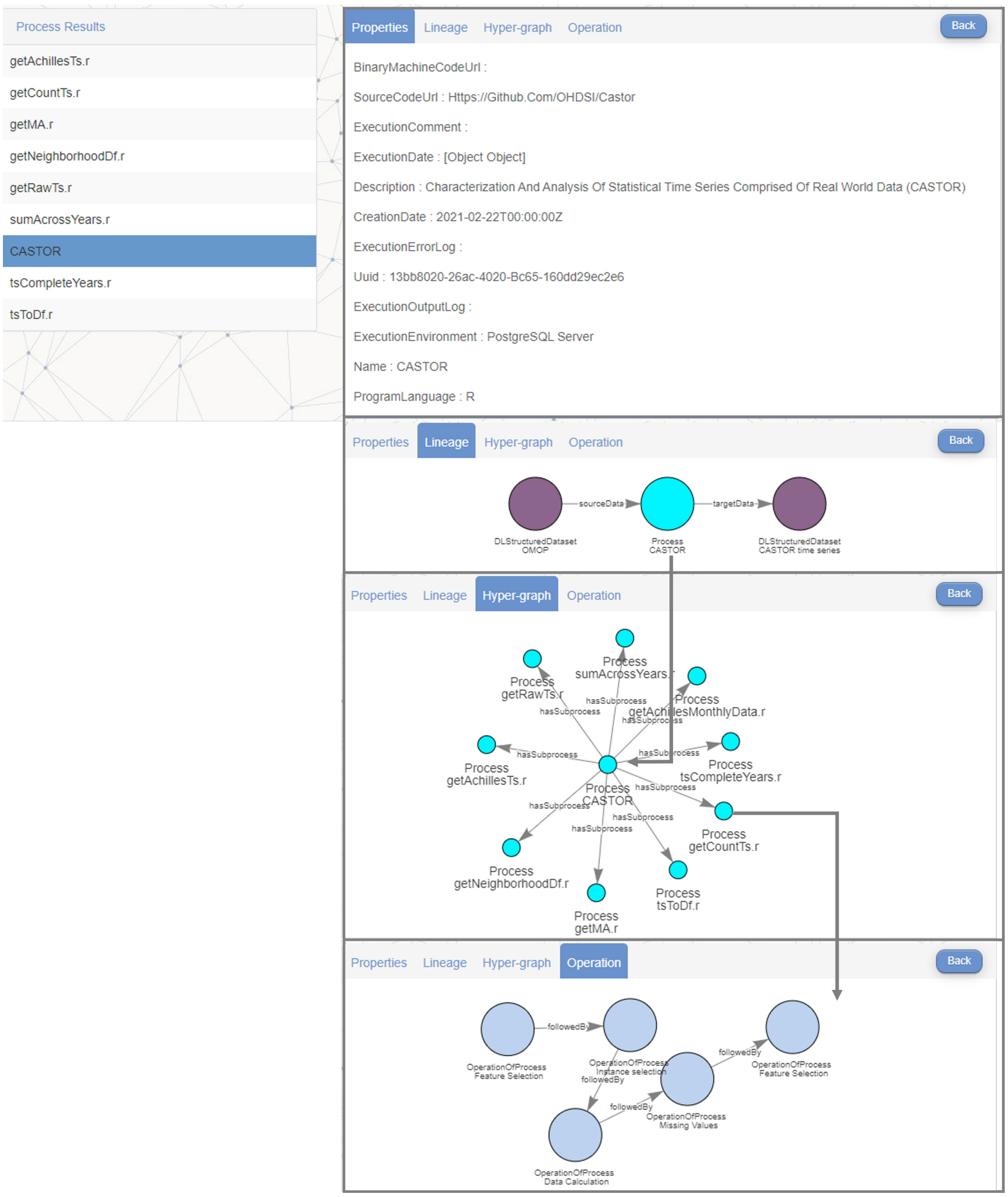
BinaryMachineCodeUrl :
SourceCodeUrl :
LastModifTime : 2021-06-20T00:00:00Z
ExecutionDate : 2021-06-22T00:00:40Z

Figure 6.14: DAMMS - process panel

Back-end

To meet user needs, we use a process similar to datasets finding. Users use the same search bar to enter keyword(s), DAMMS will launch a query to find all the processes whose name, description or lined tags contain the entered keyword(s) (see Listing C.1 in Annex C). If users add filters to limit the result, the above query will be modified to adapt to the filters and be automatically launched to obtain new result (see Listing C.1).

When users select a process in the result list, all the metadata about the process will also be searched and loaded in the process panel. Different queries that are used for this task can be found in Listing C.2 in Annex C.



BinaryMachineCodeUrl :

SourceCodeUrl : Https://Github.Com/OHDSI/Castor

ExecutionComment :

ExecutionDate : [Object Object]

Description : Characterization And Analysis Of Statistical Time Series Comprised Of Real World Data (CASTOR)

CreationDate : 2021-02-22T00:00:00Z

ExecutionErrorLog :

Uuid : 13bb8020-26ac-4020-Bc65-160dd29ec2e6

ExecutionOutputLog :

ExecutionEnvironment : PostgreSQL Server

Name : CASTOR

ProgramLanguage : R

Figure 6.15: DAMMS - data preparation use case 1

6.4.2.3 Use case 1 : Data preparation guidance

Bob, a data analyst, wants to start a time series analysis on diabetic data. There are several ways to perform an analysis with time series, with the application, Bob can search for existing project on diabetic and/or through time series. To do so, Bob launches a search with the keywords “time series” and checks the process box to have a first batch of potentially useful processes.

The application returns a list of processes / sub-processes that relate to time series (see Fig. 6.15). The Bob discovers the CASTOR process which is a process containing several sub-processes. He can check all the operations used by CASTOR in the *OPERATION* tab and verify the operations used by each sub-process.

He gets a holistic view of a time-series process without reading the source code. Since the process is closely linked to his analysis, he decides to start preparing a database on this existing process.

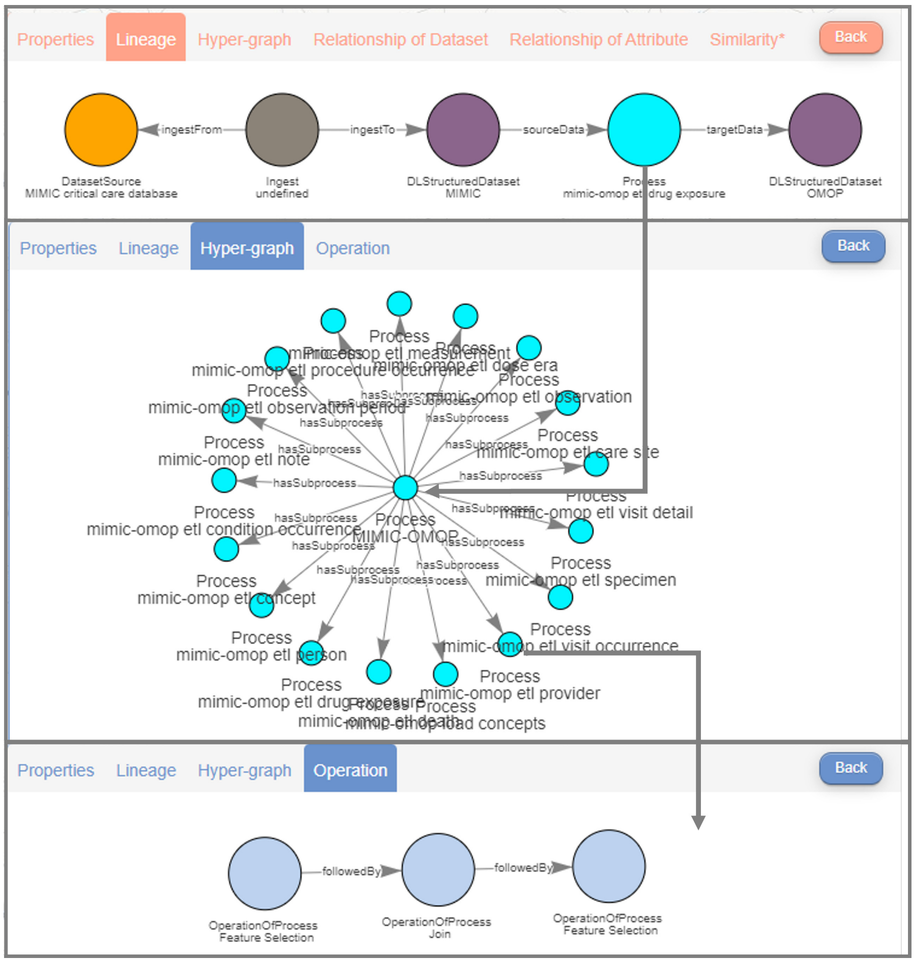


Figure 6.16: DAMMS - data preparation use case 2

6.4.2.4 Use Case 2 : Data origin check

For another project, Bob wants to analyze patient history to help doctors improve medical treatment, the data source is the OMOP dataset. Bob needs to firstly verify how OMOP dataset has been created. In addition, to ensure the quality of his analysis, he needs to know if the data are already cleaned up when creating the OMOP dataset. Therefore, he uses the application to find the OMOP dataset and uses the lineage tab to check the dataset creation process, and he searches the details of the creation process to check if it contains data cleaning operations (see Sec. 4).

With the application result (see Fig. 6.16), Bob finds out the lineage of OMOP dataset: a external dataset MIMIC is ingested in data lake and then is transformed to create OMOP dataset through the process *mimic-omop etl observation*. So Bob checks the hyper-graph of the process to see all its sub-process. He finds one process *mimic-omop etl visit occurrence* concerning the patient consultation results and stays which is important for his research. So he checks if these data are cleaned up during the data preparation phase. In the *Operation* tab, he finds out that visit table is created by using *feature selection* and *join* without data cleaning operations. Therefore, to ensure his patient history analysis, he needs to clean OMOP dataset data.

6.4.3 Explore Analyses

Analyses exploration has for objective to help users to find preformed useful analyses, consult features and landmarkers information and reuse existing analyses.

6.4.3.1 User Needs

When users explore analyses that are stored in the data lake, they should at least be able to use keyword(s) to search all the subject-related analyses (same as datasets and processes). To limit the result of search, we provide filters that are related to data analysis, such as type of analysis (machine learning, other), creation date. Moreover, we provide also secondary filters to help users to have a more precise result. For instance, when machine learning is selected, more filters will be displayed, such as type of machine learning (supervised, unsupervised, reinforcement), applied landmarkers, applied algorithms with parameters and evaluations (see Fig. 6.17). For analyses, users should be able to consult details of each single analysis as well:

- Properties: basic information of selected analysis, such as its name, description and creation date. This information helps users to have a general view of a dataset.
- Lineage: visualized data lineage that shows the dataset on which the selected analysis was performed and the study in which the analysis exists.
- Hyper-graph: details of an analysis which include the used algorithm, selected parameters and set values, output model of the analysis and the evaluation of different measures. This information helps users to know if an analysis is performed with which algorithm and what parameters.
- Features: statistics of the numeric features and nominal features. In this tab, user can also find statistics of each attribute existing in the dataset. This information helps users understand the selected features of the analysis.
- Relationship of attribute: the calculated relationships between every two attributes in the selected structured or semi-structured dataset. It is the same tab in the dataset result.

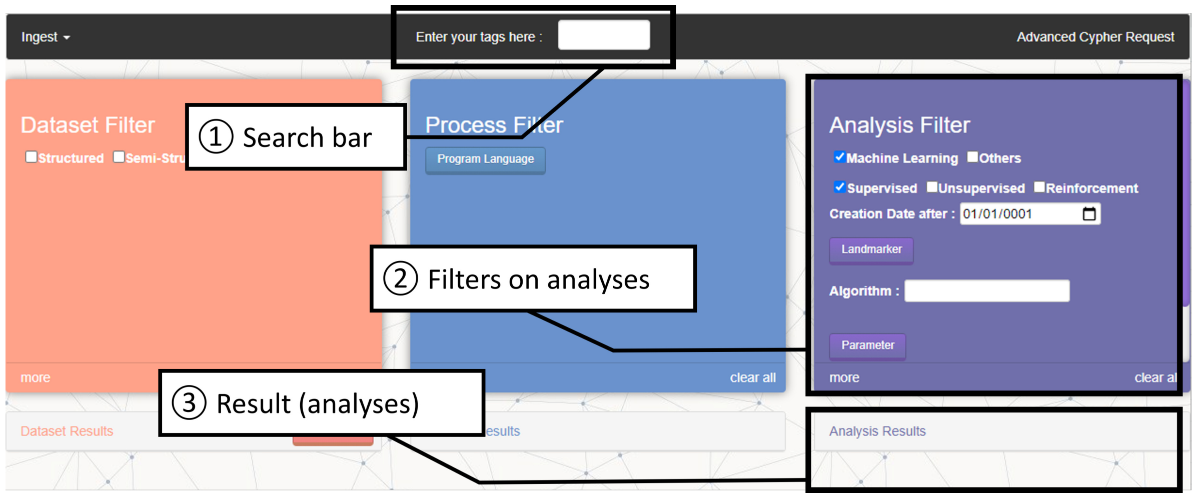
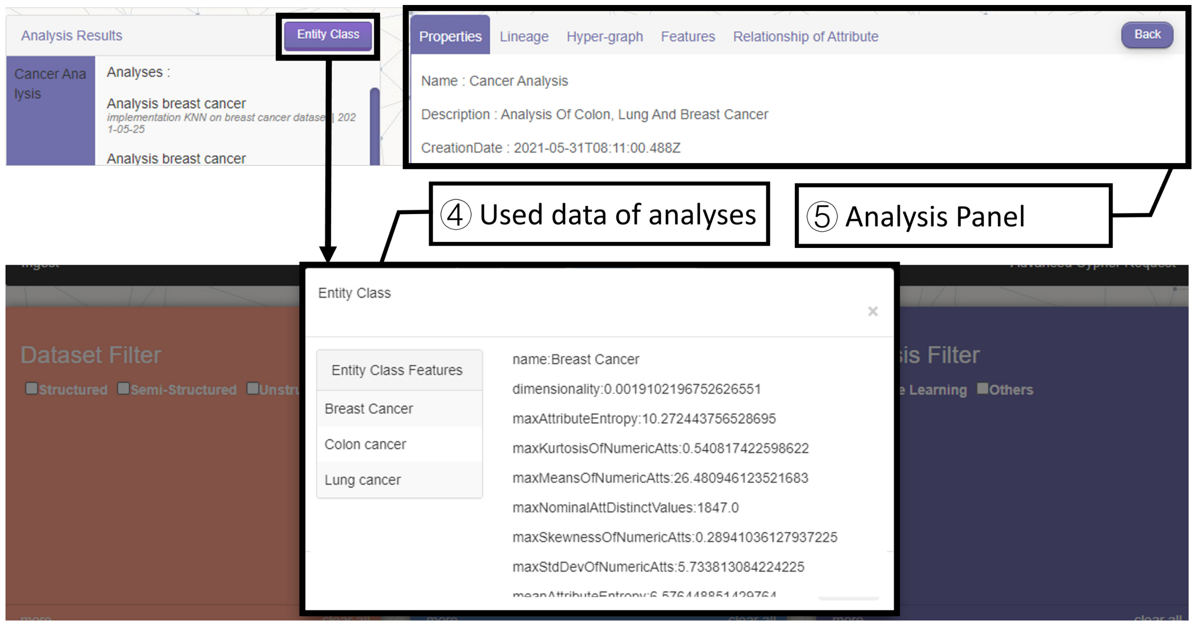


Figure 6.17: DAMMS - element search interface with filters on analyses



Name : Cancer Analysis

Description : Analysis Of Colon, Lung And Breast Cancer

CreationDate : 2021-05-31T08:11:00.488Z

Entity Class

name:Breast Cancer
dimensionality:0.0019102196752626551
maxAttributeEntropy:10.272443756528695
maxKurtosisOfNumericAtts:0.540817422598622
maxMeansOfNumericAtts:26.480946123521683
maxNominalAttDistinctValues:1847.0
maxSkewnessOfNumericAtts:0.28941036127937225
maxStdDevOfNumericAtts:5.733813084224225

Figure 6.18: DAMMS - analysis panel

Front-end

To ensure the process search, similar to other DL elements, DAMMS also should have (i) a search bar with which explorers can use keyword(s) to obtain a list of all corresponding analyses, (ii) filters that can be applied on analyses to limit the result, such as analysis type, creation date and applied algorithm and (iii) a result panel where all the analyses that correspond to the search are listed (see Fig. 6.13).

Similar to dataset and process search, for data analyses, DAMMS displays search result in a hierarchical navigation. The first level is a list of analyses that correspond to the input keywords and filters. The second level is about all used detests/tables of an analysis project (see Fig. 6.18), for each table, the statistic information of its values is displayed to help users to have a general view of its content. The third level concerns details of a selected analysis (analysis panel). Analysis details are regrouped in different tabs in the application, each tab contains different categories of metadata that are introduced in section 5.2 (see Fig. 6.18 and an example in Fig. 6.19).

Back-end

Same as searching datasets and processes, when users look for analyses, they enter keyword(s) in the search bar, DAMMS launches a query to find all the analyses whose name, description or lined tags contain the entered keyword(s) (see Listing D.1 in annex D). If any filter is added to limit the result, the above query will be modified to adapt to the new condition and be automatically launched to obtain new result (see Listing D.1).

When users select an analysis in the result list, all the metadata about this analysis will be searched and loaded in the analysis panel. Different queries that are used for this task can be found in Listing D.2 in Annex D.

6.4.3.2 Use Case 1: Dataset similarity detection

Identifying the right machine learning model for a given dataset can be a very tedious task. On this way, detecting similar datasets where previous models have already been executed is a possible solution. This would make it easier for a user to identify the type of algorithm that can be launched for their dataset. This approach is inspired by the automated machine learning field where a model is automatically proposed from a set of existing datasets and machine learning workflows (Feurer et al., 2015). Thanks to the dataset and attribute metadata, it is easier to detect stored close datasets, as long as a proximity measure has been implemented in the machine learning (*Relationship* metadata).

In our running example, a dissimilarity measure, proposed in (Raynaut et al., 2016), is used. This measure is defined on two levels. The first level estimates a dissimilarity between datasets (classes *AnalysisDSRelationship* and *RelationshipDS*) and the second level is between attribute (classes *AnalysisAttribute* and *RelationshipAtt*).

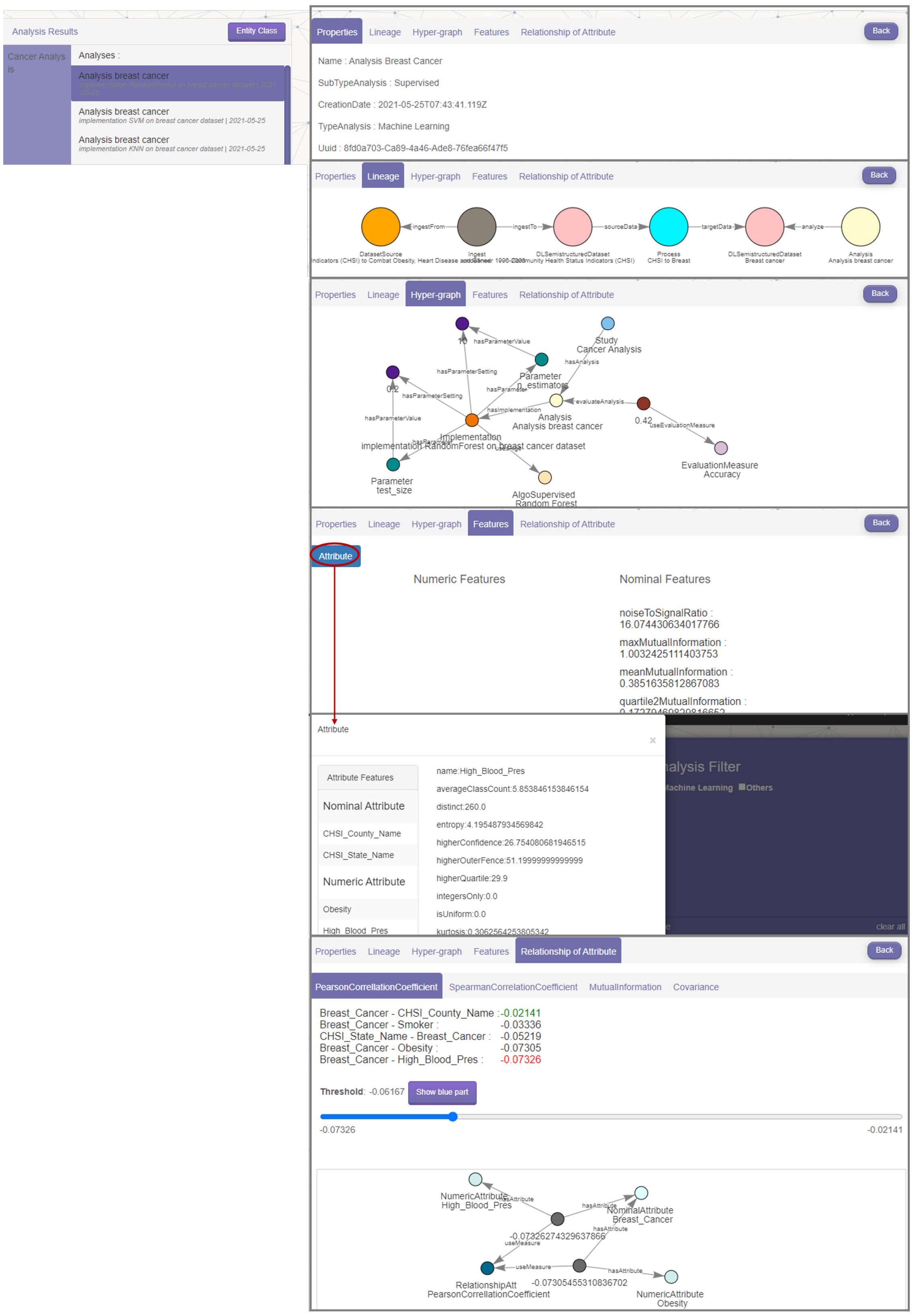
Bob wants to retrieve more datasets which are close to the colon cancer dataset that he analyzes. Thus, he uses the metadata management application to find colon cancer dataset, and goes to the *Relationship Dataset* tab to check the information that he needs (see top image in Fig. 6.20). He finds out that breast cancer dataset and lung cancer dataset are both similar to the colon cancer dataset (with the dissimilarities <0.01).

Thanks to the result, Bob decides to enrich his analysis with the breast and lung cancer datasets. Moreover, Bob can search all the analyses/algorithms that are already executed on these datasets. For each algorithm, he can get the information of used algorithm name, parameter names, parameter values and evaluation values (see bottom image in Fig. 6.20). For instance, the shown result concerns an analysis of lung cancer, for which a random forest algorithm is preformed with two parameters (n_estimators = 20, test_size = 0.2) and the result is evaluated by accuracy (0.58).

Looking at these results, Bob decided to perform a Random Forest and SVM algorithms on his dataset because they give the best results in terms of accuracy on the breast (0.48) and lung cancer (0.58) analyses.

6.4.3.3 Use Case 2: Machine learning guidance

Metadata based on landmarkers are a powerful solution to estimate which kind of machine learning model will be performed better on a dataset. By running a set of diverse simple models on the dataset, a baseline of model performances can be established. This baseline is a first indicator of possible performances for more complex versions of each model, which reduces the need for trial and error testing commonly associated with the choice of machine learning model.



Name : Analysis Breast Cancer

SubTypeAnalysis : Supervised

CreationDate : 2021-05-25T07:43:41.119Z

TypeAnalysis : Machine Learning

Uuid : 8fd0a703-Ca89-4a46-Ade8-76fea66f47f5

noiseToSignalRatio : 16.074430634017766

maxMutualInformation : 1.0032425111403753

meanMutualInformation : 0.3851635812867083

quartile2MutualInformation :

name:High_Blood_Pres
averageClassCount:5.853846153846154
distinct:260.0
entropy:4.195487934569842
higherConfidence:26.754080681946515
higherOuterFence:51.19999999999999
higherQuartile:29.9
integersOnly:0.0
isUniform:0.0
kurtosis:0.3062564253805342

Breast_Cancer - CHSI_County_Name : -0.02141
Breast_Cancer - Smoker : -0.03336
CHSI_State_Name - Breast_Cancer : -0.05219
Breast_Cancer - Obesity : -0.07305
Breast_Cancer - High_Blood_Pres : -0.07326

Threshold: -0.06167

-0.07326 -0.02141

Figure 6.19: DAMMS - analyses result

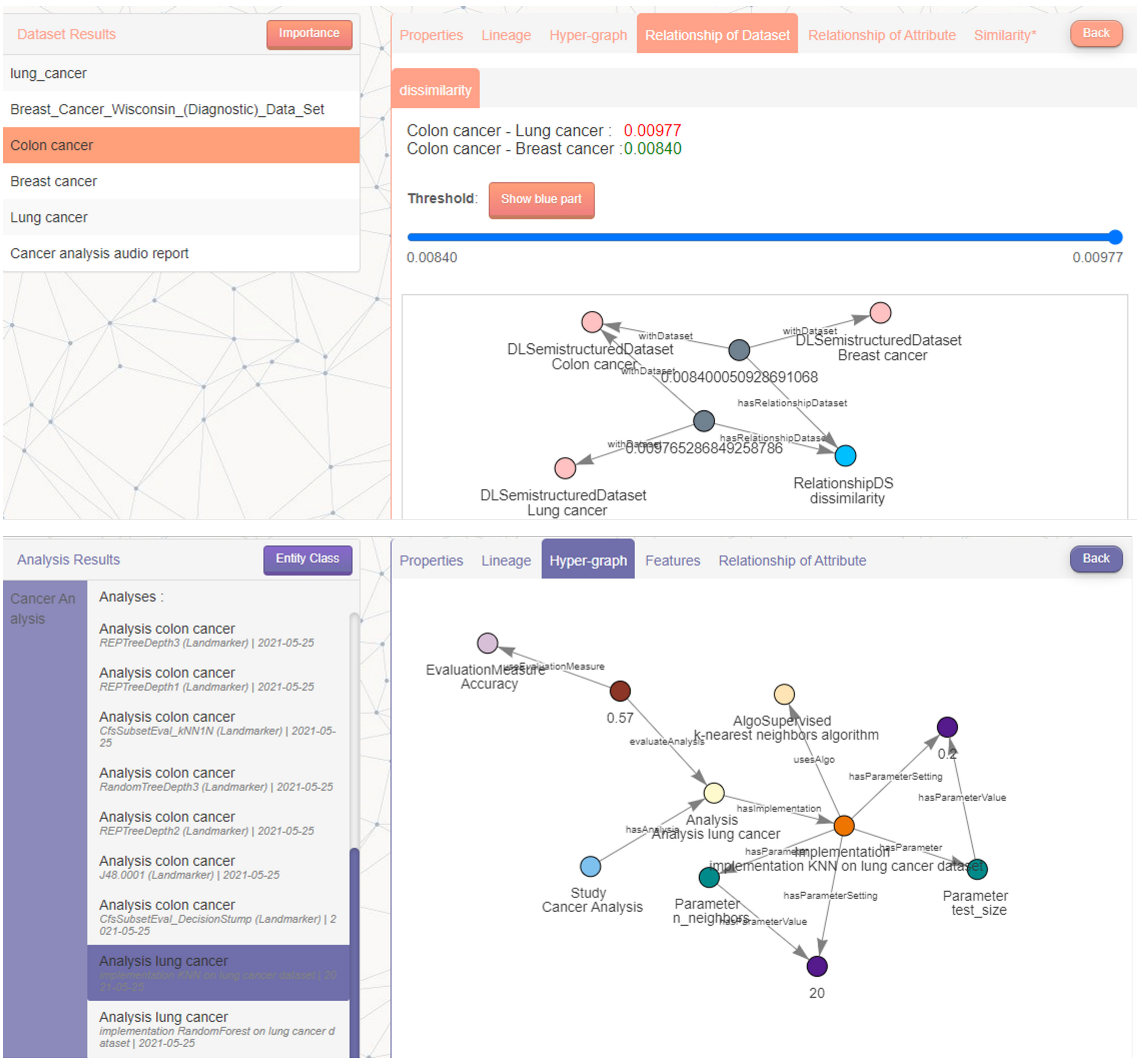


Figure 6.20: DAMMS - search analyses use case 1

In our running example, four landmarkers (and derived versions using different parameters) are available in the Data Lake. These landmarkers include: Decision Tree, Naive Bayes, KNN and Random Forest (see class *Landmarker* in Fig. 5.3). They are evaluated thanks to one measure: Error rate.

Bob wishes to execute a predictive model on the colon cancer dataset. To guide him, he wants to know the error rate of the landmarkers available in the data lake. In the application, he can check the existing landmarkers and evaluations one by one. For instance, the left image of Fig. 6.21 concerns the landmarker *RandomTreeDepth3* and its error rate is 0.4649; the right image of Fig. 6.21 concerns the landmarker *REPTreeDepth3* and its error rate is 0.5070.

After checking all the landmarkers that have been executed on the colon cancer dataset, Bob finds the lowest error is performed by *J48.0001*. Therefore, Bob can use the landmarker J48 which gives him a basis to build a better model then.

If Bob does not want to click on every landmarker to look for the lowest error rate, he can always do advanced research by writing query by himself. We provide a function of free consultation in the web-application which requires Cypher (Neo4j querying language)

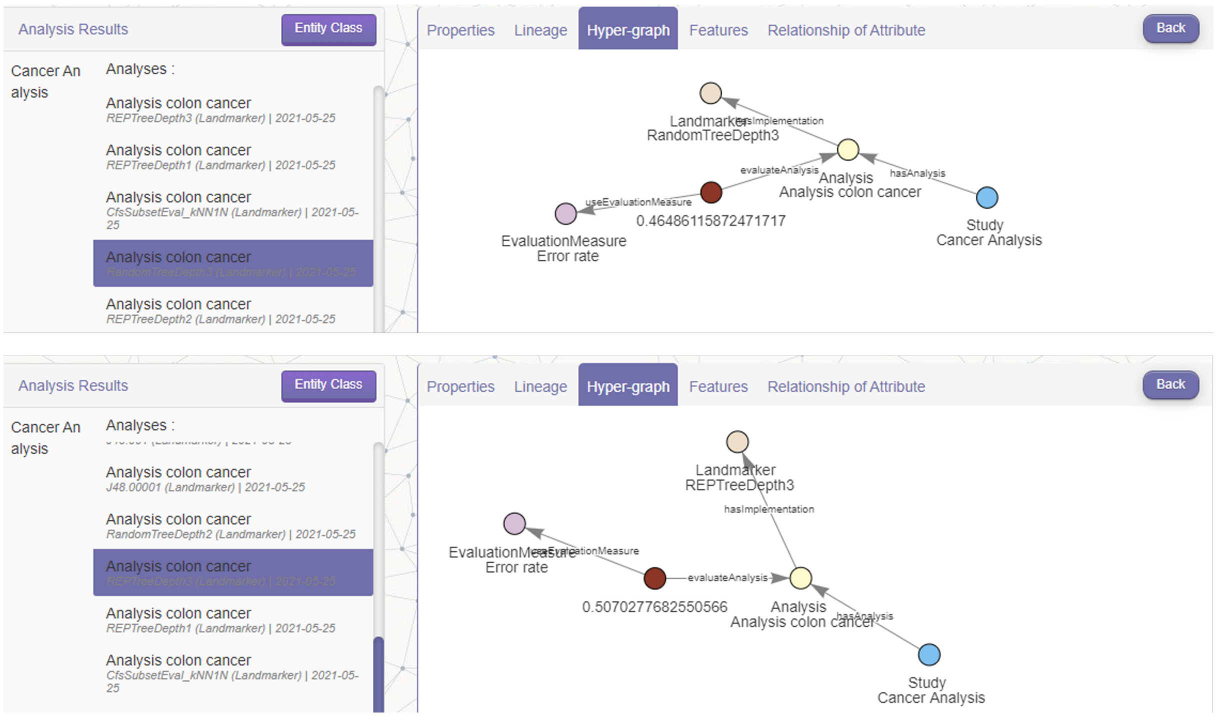


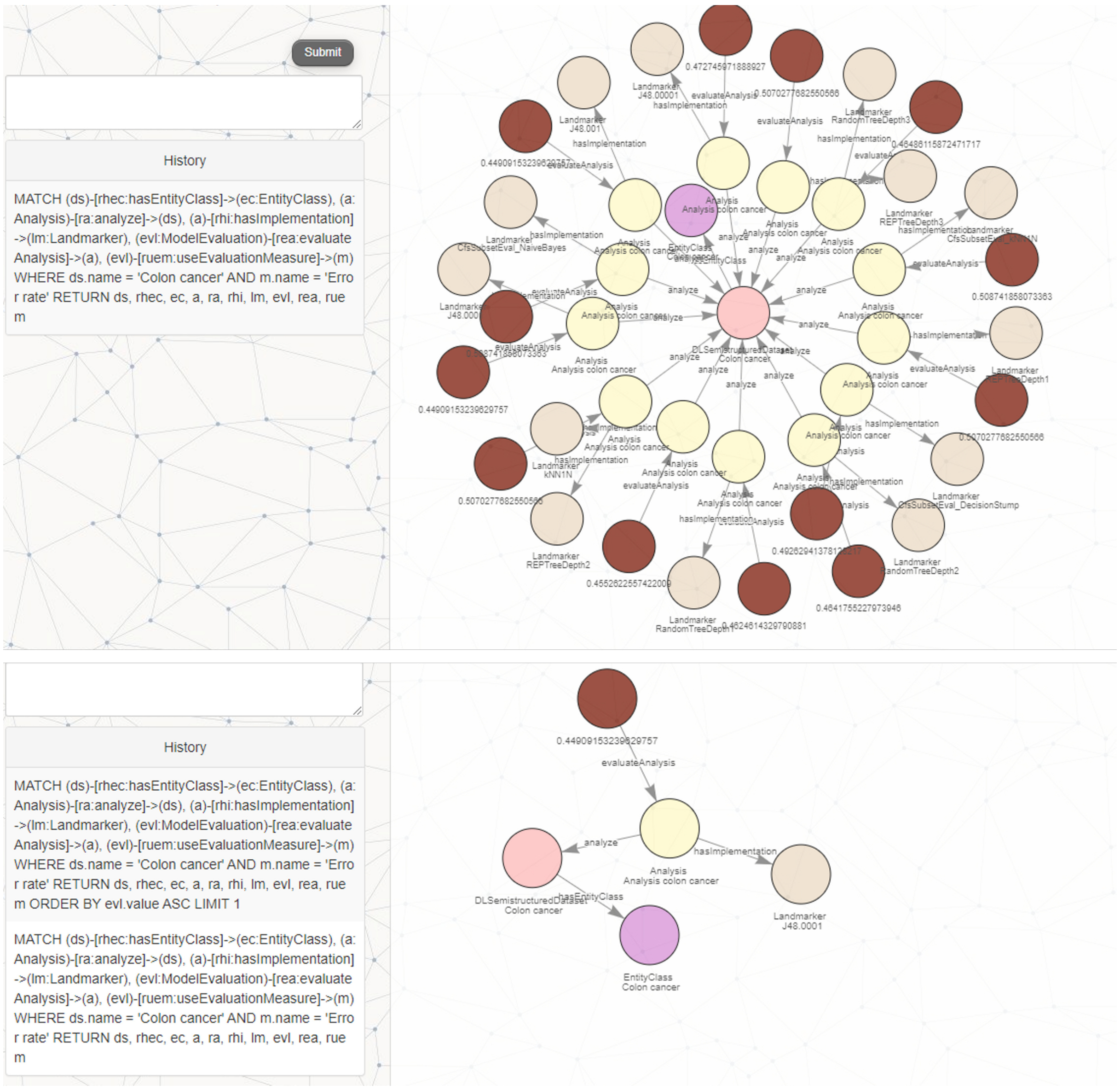
Figure 6.21: DAMMS - search analyses use case 2

skill. So that Bob can use the Chyper query below to find all the landmarkers of colon cancer dataset (top result in Fig. 6.22).

```
    MATCH
      (ds)-[:hasEntityClass]->(ec:EntityClass),
      (a:Analysis)-[:analyze]->(ec),
      (a)-[:hasImplementation]->(lm:Landmarker),
      (evl:ModelEvaluation)-[:evaluateAnalysis]->(a),
      (evl)-[:useEvaluationMeasure]->(m)
    WHERE
       ds.name = 'Colon cancer'
       AND m.name = 'Error rate'
    RETURN ds, ec, a, lm, evl, m
```

By adding one condition on the query, Bob can get the lowest error rate landmarker directly (bottom result in Fig. 6.22).

```
    ORDER BY evl.value ASC LIMIT 1
```



```
MATCH (ds)-[rhec:hasEntityClass]->(ec:EntityClass), (a:Analysis)-[ra:analyze]->(ds), (a)-[rhi:hasImplementation]->(lm:Landmarker), (evl:ModelEvaluation)-[rea:evaluateAnalysis]->(a), (evl)-[ruem:useEvaluationMeasure]->(m) WHERE ds.name = 'Colon cancer' AND m.name = 'Error rate' RETURN ds, rhec, ec, a, ra, rhi, lm, evl, rea, ruem
```

```
MATCH (ds)-[rhec:hasEntityClass]->(ec:EntityClass), (a:Analysis)-[ra:analyze]->(ds), (a)-[rhi:hasImplementation]->(lm:Landmarker), (evl:ModelEvaluation)-[rea:evaluateAnalysis]->(a), (evl)-[ruem:useEvaluationMeasure]->(m) WHERE ds.name = 'Colon cancer' AND m.name = 'Error rate' RETURN ds, rhec, ec, a, ra, rhi, lm, evl, rea, ruem ORDER BY evl.value ASC LIMIT 1
```

```
MATCH (ds)-[rhec:hasEntityClass]->(ec:EntityClass), (a:Analysis)-[ra:analyze]->(ds), (a)-[rhi:hasImplementation]->(lm:Landmarker), (evl:ModelEvaluation)-[rea:evaluateAnalysis]->(a), (evl)-[ruem:useEvaluationMeasure]->(m) WHERE ds.name = 'Colon cancer' AND m.name = 'Error rate' RETURN ds, rhec, ec, a, ra, rhi, lm, evl, rea, ruem
```

Figure 6.22: DAMMS - free query

6.5 Conclusion

Metadata management is essential for preventing a data lake turning into a data swamp which is invisible, inaccessible and incomprehensible. To ensure that users are able to find, access, interoperate and reuse existing datasets, processes and analyses to take advantage of all the capitalized experiences, we proposed different categories of metadata in chapters 3-5. And in this chapter, we introduced a developed DAda lake Metadata Management System (DAMMS) which can generate the proposed metadata and allows users to search all the metadata of stored elements in the data lake. The system has ergonomic interfaces which visualize different types of information to allow users to search and understand different elements easily.

Chapter 7

Experimental Assessments

Contents

7.1 Introduction

Metadata of different elements in a data lake can be generated by DAMMS and stored in a graph database (Neo4j). Moreover, the stored metadata can be explored and discovered by users through ergonomic interfaces of DAMMS. In the previous chapter, we explained the two functions provided by the DAMMS tool that can meet the FAIR requirements.

Beyond the functionalities of DAMMS, it is necessary to analyze the user's appropriation of this tool. This study must take into account the two main functionalities of the tool: (i) the generation of metadata and (ii) the exploration of the DL elements.

Regarding the generation of metadata, the major problem that may encounter users is the difficulty in integrating different types of elements and the processing time for the generation of metadata. From the exploration point of view, it is necessary to check whether it is easy to find different elements and if the results correspond to the users' expectations.

To answer these problems, we propose two types of analysis: the first one analyzes the metadata generation performance, the second one is oriented to user experiences for the exploration of the data lake elements.

In this chapter, we will present the experiments that have been conducted to evaluate DAMMS in the two aspects. We have jointly conducted a performance study of metadata ingestion in the section 7.2 and a study analyzing the user experience of DAMMS in the section 7.3.

7.2 Quantitative Evaluation

To carry out the evaluation of the metadata generation and to ensure its completeness, we propose experiments with different parameters of the input type and analysis metrics.

Regarding the input of experiments, datasets vary in type and size which can be used evaluate the scalability of our system. We address the different types : structured, semi-structured and unstructured. To be more precise, structured datasets can be stored in different DBMS (database management system, e.g. ORACLE, PostgreSQL). The semi-structured datasets can have different forms (e.g. XML, JSON, NoSQL) and unstructured datasets designate different formats (e.g. image, text, video).

Regarding the analysis metrics, we choose to evaluate the system according to the (i) time spent for the generation of metadata and (ii) the size of the generated metadata. These metrics allow us to evaluate the efficiency and discuss the amount of metadata stored in the DL compared to the ingested data.

Regarding the infrastructure of the experiments, all the results are performed on a computer with a Intel(R) Xeon(R) E3-1230 v3 3.30GHz processor and 16GB of RAM

In rest of the section, we regroup different experiments according to the structural type of input. For each type of input, we analyze the result with different metrics. In section 7.2.1, we evaluate metadata generation of structured datasets. In section 7.2.2, we worked on semi-structured datasets. And in section 7.2.3, we compare the metadata generation of different unstructred datasets.

7.2.1 Metadata generation from structured data sources

Input

To vary the size of data sources, we use three relational databases. From the volume point of vue, the DB may have from 1 to 15 tables and from 2981 to 460268 instances. Besides the data source size, DAMMS supports the metadata generation from two different DBMS: ORACLE and PostgreSQL. To evaluate metadata generation of datasets of different sizes and stored in different DBMS, we implemented the four databases in ORACLE and in PostgreSQL (see table 7.1). The three databases, ranked according the volume of their instances, are the following:

- OT[1], a database storing fictitious computer selling information, which contains 12 tables, 54 columns and 2981 instances in total;
- LCD (Leading causes of death)[2], a table that summarizes death reasons of each of the US counties, which contains 235 columns and 1315 instances in total;
- Rent DVD[3], a database for DVD renting, which contains 15 tables, 86 columns and 44820 instances in total;
- MIMIC[4], an open-source medical database (only 6 tables are imported for the experiment due to disk limit), which contains 6 tables, 82 columns and 460269 instances.

Database Name	Database type	Number of tables	Number of columns	Number of instances	Database size (MB)
OT	ORACLE	12	54	2981	0.75
	PostgreSQL	12	54	2981	8.813
LCD	ORACLE	1	235	1315	1.91
	PostgreSQL	1	235	1315	12
Rent DVD	ORACLE	15	86	44820	8
	PostgreSQL	15	86	44820	15
MIMIC(part)	ORACLE	6	82	460268	41
	PostgreSQL	6	82	460268	62

Table 7.1: Data sources (structured datasets)

Result

After the metadata generation, 72-243 nodes and 71-242 relationships are created in the Neo4j database (see table 7.2).

Regarding the size of metadata, the created metadata are around 1MB for the eight datasets. Regarding the time spent for metadata generation, as shown in Fig. 7.1, the generation time of ORACLE databases is globally stable. However, the generation time of PostgreSQL increases as the number of instances or columns of database increases. Moreover, in general, DAMMS spends more time on generating metadata from PostgreSQL than ORACLE databases.

[1]`https://www.oracletutorial.com/getting-started/oracle-sample-database/`
[2]`https://data.world/us-hhs-gov/fcdb091a-3d47-4f43-a99c-19c9e95c8ca9`
[3]`https://www.postgresqltutorial.com/postgresql-sample-database/`
[4]`https://mimic.mit.edu/`

	Database type	nodes	relationships	metadata size (MB)	% of dataset size
OT	ORACLE	72	71	0.914	122%
	PostgreSQL	72	71	0.914	10%
LCD	ORACLE	243	242	0.929	51%
	PostgreSQL	243	242	0.929	8%
Rent DVD	ORACLE	108	107	0.914	11%
	PostgreSQL	108	107	0.914	6%
MIMIC(part)	ORACLE	94	93	0.914	2%
	PostgreSQL	94	93	0.914	1%

Table 7.2: Metadata generation result (structured datasets)

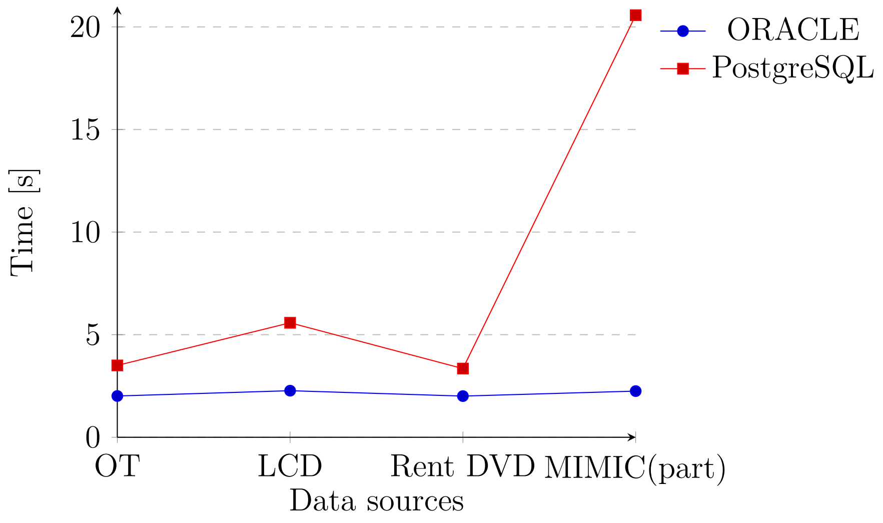


Figure 7.1: Metadata generation time of structured data sources

Analysis

Regarding the generated metadata volume, we can see that the stored metadata have the same size (0.914MB) for the OT, Rent DVD and MIMIC(part) datasets regardless of the DBMS and the size of dataset source. It is because metadata are stored with almost the same number of nodes and relationships for the three datasets which are related to the number of tables and columns of datasets. The metadata of LCD dataset have a bigger size than the other datasets because more nodes and relationships were created. The volume of metadata is not necessarily linked to the size of datasets but the number of tables and columns of structured datasets. For the datasets who have nearly the same number of tables and columns, the larger the dataset (more instances), the lower the proportion of metadata stored.

Regarding the time spent on metadata generation, in general, the consumed time is very short (of the order of a few seconds) for the eight datasets. By extrapolating these values on large volume of data, especially for PostgreSQL, it remains reasonable in the order of a few minutes. Metadata generation from LCD dataset takes lightly more time because this dataset has more columns. The reason why it takes more time to generate metadata from PostgreSQL than ORACLE databases when there are more tables/columns/instances is that DAMMS uses different functions to extract metadata:

- for PostgreSQL databases, we use the "*pg*" API and extract the information such as database name, size and schematic directly by SQL queries. However, regarding statistical information, such as max, min value and missing value count, we collect firstly the values of different columns and then calculate them by the functions that we defined.

- for ORACLE databases, we use the "*oracledb*" API and extract metadata from the views of "DBA_OBJECTS" and "DBA_TAB_COLUMNS" from the instance of database directly. Therefore, the increase database size (number of instances) does not influence a lot the metadata generation time.

7.2.2 Metadata generation from semi-structured data sources

Input

In DAMMS, the generation of metadata from semi-structured datasets is currently supported for CSV files. To ensure that the datasets used for the evaluation vary in size, we ingested different CSV files from 1MB to 500MB (see table 7.3).

Name	Rows	Columns	Size (MB)
Country GDP	42451	4	0.995
Leading causes of death	3142	235	3.86
Videos	12919	9	5.17
Clients	50000	9	9.95
Train splits	1019926	8	100
DATETIMEEVENTS (from mimic database)	1048576	14	501.43

Table 7.3: Data sources (semi-structured datasets)

Result

After the generation of metadata, 9-19 nodes and 8-18 relationships are created in the Neo4j database (see table 7.4).

Regarding the size of stored metadata, it remains the same (≈ 0.9 MB) regardless of the size of the five CSV files. Regarding the time spent on generating the metadata, we can see in Fig. 7.2 that the execution time increases as the size of dataset increases.

Name	nodes	relationships	metadata size (MB)	% of dataset size
Country GDP	9	8	0.891	89.50%
Leading causes of death	240	239	0.953	24.69%
Videos	15	14	0.891	17.22%
Clients	14	13	0.891	8.95%
Train splits	13	12	0.891	0.89%
DATETIMEEVENTS (from mimic database)	19	18	0.898	0.18%

Table 7.4: Metadata generation result (semi-structured datasets)

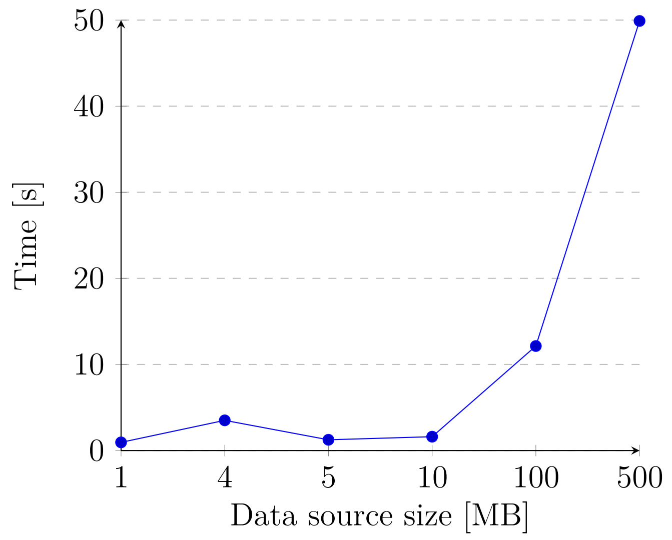


Figure 7.2: Metadata generation time of semi-structured data sources (CSV)

Analysis

Regarding the generated metadata volume, we note the same observation as the previous sub-section, the volume of stored metadata is not necessarily linked to the size of datasets but the number of columns.

Regarding the time spent on metadata generation, we notice also that the execution time increases as the number of rows and columns increases. The exponential growth of time is related to the process performed on csv files: we collect firstly the file content (all its values), then we extract basic information (e.g. name, size, data of creation), schematic information (e.g. columns name, type) and statistical information (e.g. min, max, missing value count). As the metadata feeding for semi-structured datasets is evolving exponentially, we plan to implement mechanisms for parallelizing calculations (such as MapReduce) on the implemented algorithms in order to reduce execution times.

7.2.3 Metadata generation from unstructured data sources

Input

In DAMMS, the generation of metadata from unstructured datasets is supported. To evaluate different data sources, we carry out the experiments with datasets that vary in format (image, music, video and text) and size. (from 1MB to 100MB). For each format of dataset, we prepared 3 or 4 datasets (see table 7.5).

Format	Size
images	1MB, 5MB, 10MB
texts	1MB, 5MB, 10MB, 100MB
music	1MB, 5MB, 10MB
videos	1MB, 5MB, 10MB, 100MB

Table 7.5: Data sources (unstructured datasets)

Result

After the generation of metadata, 4 nodes and 3 relationships are created in the Neo4j database for each of the datasets (see table 7.6).

Regarding metadata size, the same volume (0.882MB) is stored for all the datasets. Regarding the time of metadata generation, it is globally stable (≈ 0.6s) regardless of the dataset format or size (see Fig. 7.3).

	nodes	relationships	metadata size (MB)
image/music/video/text	4	3	0.882

Table 7.6: Metadata generation result (unstructured datasets)

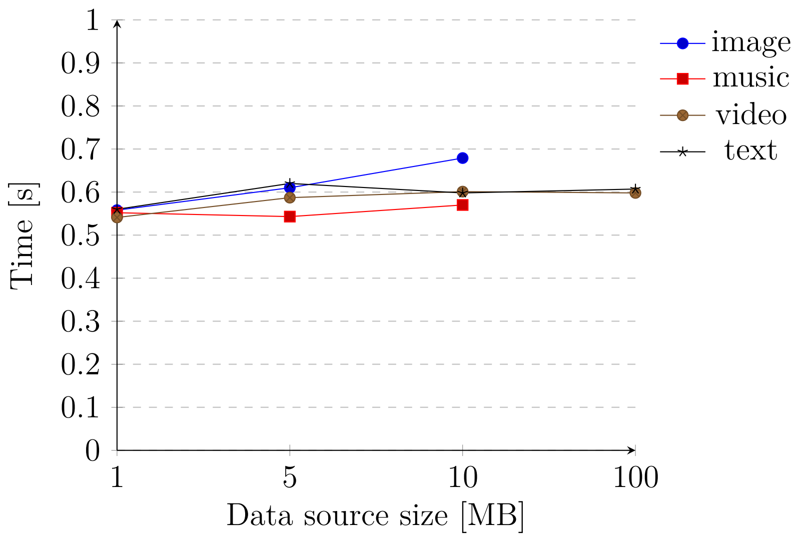


Figure 7.3: Metadata generation time of unstructured data sources

Analysis

Regarding the metadata storage size, it is small and stays stable as we only store the information of *name*, *location*, *ingestion mode*, *file name extension*, *creation date*, *size*, *format*, *owner* and *location* with 4 nodes and 3 relationships in the Neo4j database regardless of the dataset size and format.

Regarding the metadata generation time, we note that no matter the size and the format of datasets, the time of generation is around 0.6 second. The reason why the result is stable is that we extract the same metadata from unstructured datasets without reading its content. This information can be extracted directly.

7.3 Usability Evaluation

Usability, according to the Merriam-Webster dictionary[5], means the quality or state of being usable: ease of use. Evaluating the usability of the DAMMS when exploring different elements in the data lake is the issue of this section. Indeed, the exploration of metadata and the functions of DAMMS requires an interaction with the users which it is necessary

[5]https://www.merriam-webster.com/

to validate. To address this issue, we designed a controlled experimental, with different users, in which we analyze different aspects of the usability of our system. To measure the usability of a system in general, ISO 9241-11:2018 (Ergonomics of human-system interaction - Part 11: Usability) emphasized three aspects (Bevan et al., 2015; ISO, 2018):

- *Effectiveness*: accuracy and completeness with which users achieve specified goals.
- *Efficiency*: resources used in relation to the results achieved (typical resources include time, human effort, costs and materials).
- *Satisfaction*: extent to which the user's physical, cognitive and emotional responses that result from the use of a system.

We will use these aspects to evaluate the usability of DAMMS. This section will be divided into four parts. Firstly, we introduce the experimental assessment protocol, then we analyze the experiment results in the three aspects: effectiveness, efficiency and satisfaction.

7.3.1 Experimental Assessment Protocol

Protocol

The controlled experiment consisted of two sessions realized by the same participants:

- The first session consists of two parts:
  - A quick introduction (around 2 minutes) of a simulated data lake which concerns a folder that contains 9 sub-folders (see table 7.7). Each sub-folder concerns a project which contains different files (datasets, data preparation and analysis scripts) (see an example in Fig. 7.4).
  - Participants need to answer 6 questions (2 questions for each type of the data lake elements: datasets, data preparation and analyses) (see appendix E) by searching useful information in the folder. Note that the use of the search bar of the file system is authorized.
- The second session, a self-control experiment using DAMMS, consists of two parts: (i) a short introduction (around 5 minutes) of DAMMS. The metadata of all the projects used in the first session are generated and stored in the system. (ii) participants need to answer the same 6 questions by using the DAMMS system.

To prepare the efficiency analysis, during the two sessions, we will record the consumed time and the number of clicks for each question that participants answer.

Questionnaire Design

When participants finished the two sessions, they need to answer a questionnaire which contains 24 questions following three aspects (see appendix F):

- General: questions about user data lake or data lake metadata management system experiences.

Projects/Folders	Sub-folders	Files
Argos	10	90
Breast Cancer Wisconsin (Diagnostic) Data Set		1
Castor	6	59
Chest X-Ray Images (Pneumonia)	30	17591
CHSI		8
COVID-19 World Vaccination Progress		2
Fetal Health Classification		1
Lung Cancer DataSet		1
OMOP	86	8110

Table 7.7: Projects in the simulated data lake

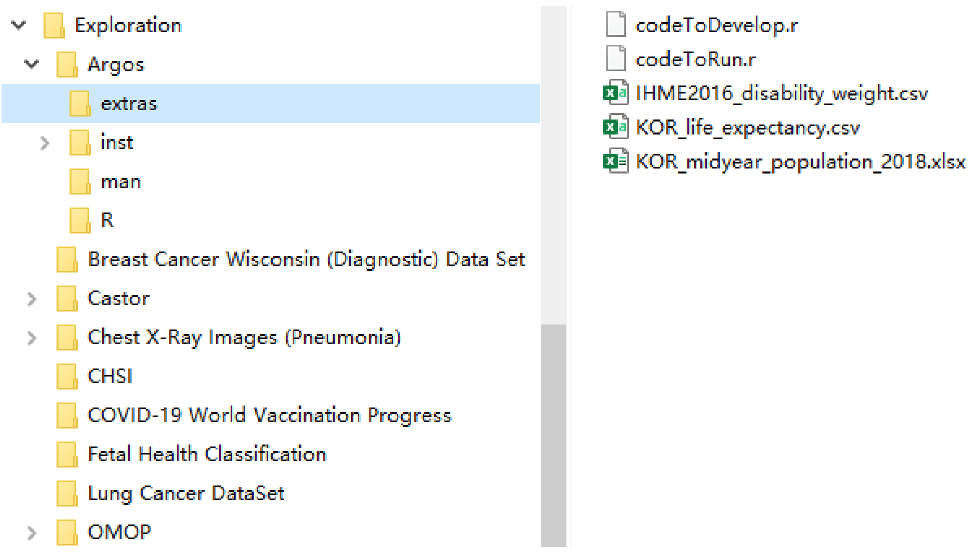

Exploration
Argos
extras
inst
man
R
Breast Cancer Wisconsin (Diagnostic) Data Set
Castor
Chest X-Ray Images (Pneumonia)
CHSI
COVID-19 World Vaccination Progress
Fetal Health Classification
Lung Cancer DataSet
OMOP
codeToDevelop.r
codeToRun.r
IHME2016_disability_weight.csv
KOR_life_expectancy.csv
KOR_midyear_population_2018.xlsx

Figure 7.4: Screen shot of a project in the simulated data lake folder

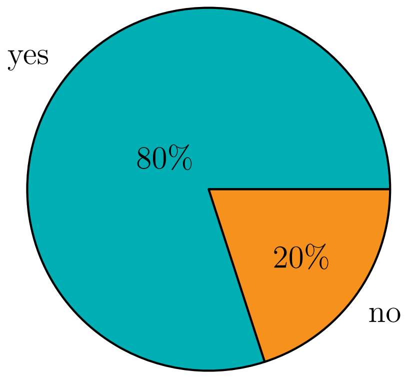
- Usability: (i) Questions of the System Usability Scale (SUS) method (Brooke, 1995), overview and suggestion of the usability of the system. The method SUS is one of the most popular system usability evaluation methods (Lewis and Sauro, 2018; Lewis, 2018). It consists of 10 questions to answer and it yields a single score on a scale of 0–100. (ii) Questions about the interfaces including the clearness, smoothness and stability of layout, navigation and information presentation.

- Practicality: questions of the layout and the navigation plan of the system.

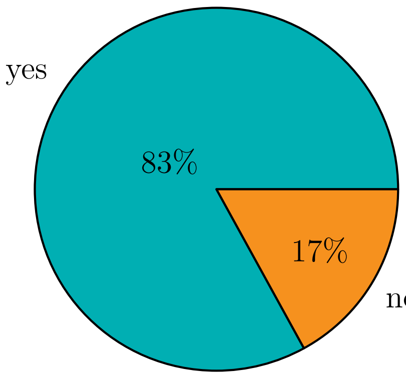
Participants

To ensure the comprehensiveness of the experiment, participants need to have profiles that vary in (i) their experience of data analysis, (ii) knowledge of the concept of data lake and (iii) hands-on experience of data lake system.

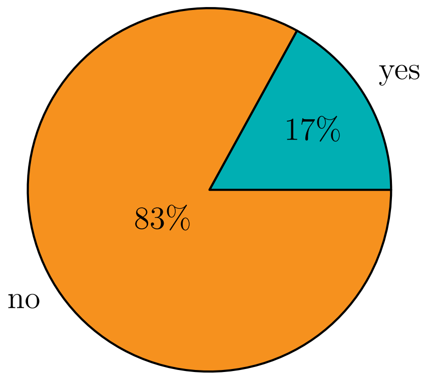
15 participants joint the controlled experiments. 80% of them have done data analysis before, 20% never did it (1 participant works in the field of law about information technology, 2 work on decision support systems) (see Fig.. 7.5a). Among the participants with data analysis experience, 83% have heard of the data lake concept (see Fig. 7.5b) whereas only 2 of them already used a data lake system (see Fig. 7.5c). Globally, all the participants are interested in data lake metadata management system (see Fig. 7.5d).



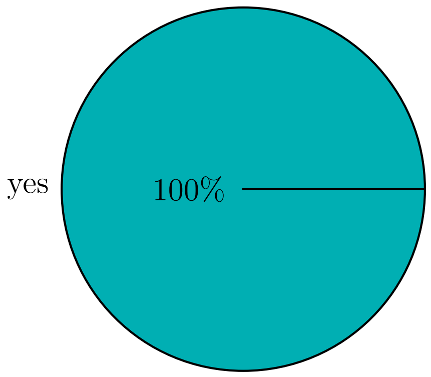
(a) Do you have experience in data analysis (machine learning, data mining, BI etc.)?



(b) Have you heard of the concept of data lake? (for participants who have experience in data analysis)



(c) Have you ever used a data lake system? (for participants who have experience in data analysis)



(d) Do you plan to use the data lake in your future career or recommend others to use it?

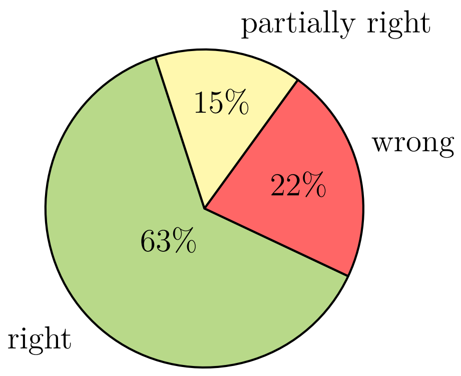
Figure 7.5: Experiment participants distribution

In the following sections, we will present (i) the results of the 6 questions that participants needed to answer to analyze the effectiveness, (ii) the time and number of clicks that participants used to answer the question manually and with the help of DAMMS to analyze the efficiency and (iii) the questionnaire results to analyze the user satisfaction.

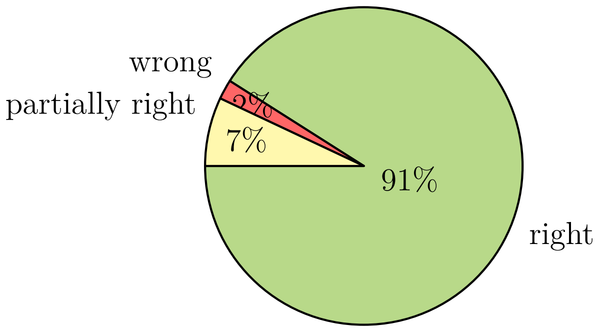
7.3.2 Effectiveness of DAMMS

To evaluate the effectiveness of DAMMS, we introduce and analyze the accuracy and completeness of the answers of the 6 questions. Note that if a participant cannot find an answer for a question, he/she can skip it, in this case, the result of the question will be considered as *wrong*.

In general, when searching manually, the ratio of correct answers to the six questions is 85% of which 63% are entirely correct and 12% are partially correct (see Fig. 7.6a). When researching with DAMMS, the ratio of correct answers increases to 98% of which 91% are entirely correct (see Fig. 7.6b). With the help of DAMMS, the effectiveness of data lake exploration is improved.



(a) Question results - searching manually



(b) Question results - searching with DAMMS

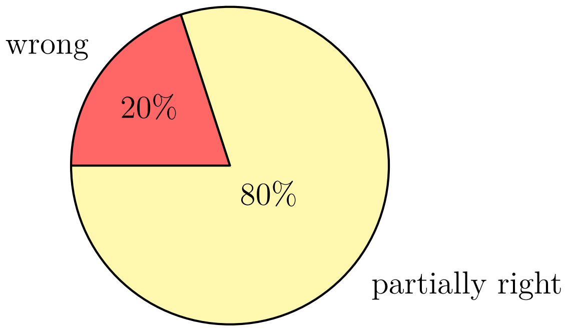
Figure 7.6: Question results

7.3.2.1 Question 1

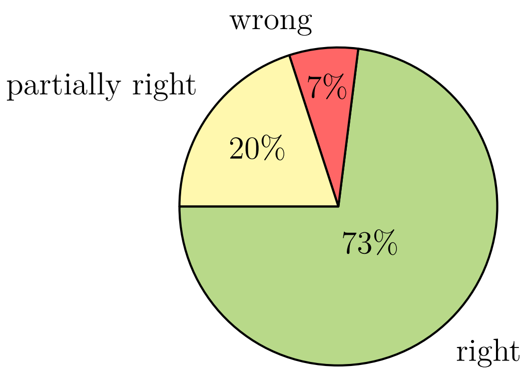
Question 1 is about letting participants to find datasets about a subject/tag: *"There are different medical datasets, can you try to find how many of them are about "breast" cancer?"*

Result

The result of the question 1 is shown in Fig. 7.7a-7.7b. From Fig. 7.7a, we can see that nobody found the right answer, 80% of participants found some relative datasets but they did not find all the relative datasets, 20% of participants found wrong datasets. From Fig. 7.7b, we can see that with the help of DAMMS, 93% of participants found relative datasets in which 73% of them found all the relative datasets. 20% of participants submitted partially correct answer and 7% found the wrong answer.



(a) Question 1 result - searching manually



(b) Question 1 result - searching with DAMMS

Figure 7.7: Question 1 result

Analysis

For this question, we can see that DAMMS greatly improved the effectiveness of answer with a contribution of 100% of entirely correct answers.

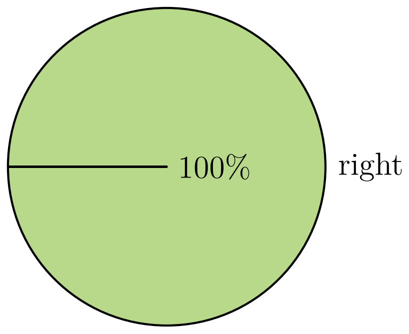
When searching with DAMMS, 73% of participants found the entirely correct answer as in DAMMS, all the datasets whose title, description, tags containing the keyword are displayed in the result panel. 20% of participants did not answer the question with all relative datasets because although they saw three datasets in the result panel, they did not trust it and they still chose the datasets whose name contains "*breast*". There is 7% of participants answered incorrectly, because he searched two keywords "*breast*" and "*cancer*", and he found all the datasets about cancer (not only breast cancer).

7.3.2.2 Question 2

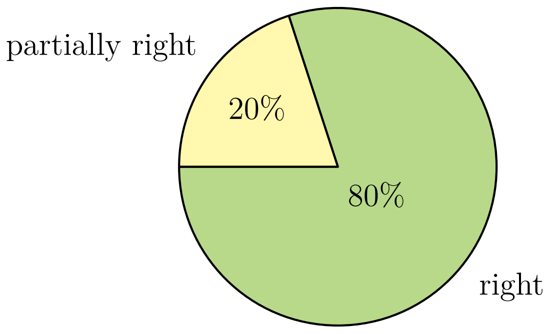
Question 2 is about letting participants to find data provenance information: *"The dataset "cancer_breast.csv" is stored in "\Exploration\ CHSI\ cancer_ breast. csv ", there are two other datasets "colon cancer" and "lung cancer" stored in the same folder, can you find out if the three datasets are similar or not?"*

Result

The result of the question 2 is shown in Fig. 7.8a - 7.8b. From Fig. 7.8a, we can see that all the participants found the entire correct answer when researching manually. From Fig. 7.8b, all the participants found the right answer in DAMMS in which 20% submitted partially correct answer.



(a) Question 2 result - searching manually



(b) Question 2 result - searching with DAMMS

Figure 7.8: Question 2 result

Analysis

From this result, we can see that regarding correctness, user manual work is more effective.

When searching manually, all participants opened the three datasets to check if they have common or similar columns (name, type) and/or if they have the same or almost the same dimensions (number of columns and instances). One participant found the creation scripts of the three datasets and found out directly that they have common columns.

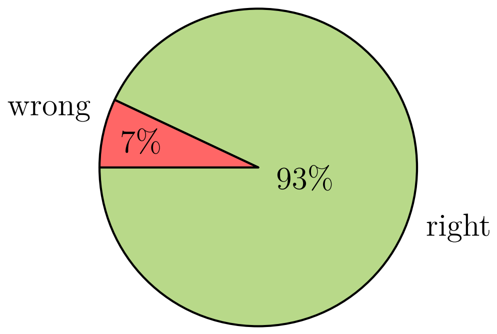
When searching with DAMMS, participants used different functions in the system to check the similarity between datasets, such as "*similarity*", "*relationship of datasets - dissimilarity*", "*lineage*". 80% of participants who found the right tabs and understood the result graphs submitted the right answer. However, 20% participants indicated that two of the three datasets are similar. Two of them (67%) did not submit the full right answer because they did not check the tabs "*similarity*", "*relationship of datasets - dissimilarity*" the first time, when they finally clicked on these tabs, they only read the fist line of result table. One participant (33%), besides the dataset relationship tab, also checked the tab of *relationship of attribute* and he answered the question with irrelevant information.

7.3.2.3 Question 3

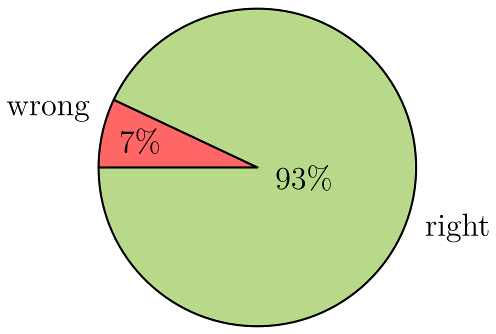
Question 3 is about letting users to find a data preparation process (lineage) when the result dataset is indicated: *"The dataset "cancer_breast.csv" was obtained through a transformation process, can you identify the* ***data source file*** *of this transformation?"*

Result

The result of the question 3 is shown in Fig. 7.9a-7.9b. From the two figures, we can see that the accuracy of the results is the same with or without DAMMS: 93% of participants found the entirely correct answer and 7% submitted the wrong answer.



(a) Question 3 result - searching manually



(b) Question 3 result - searching with DAMMS

Figure 7.9: Question 3 result

Analysis

Comparing the accuracy of results, there is no difference. However, the accessibility of the information is different from the two experiences.

When searching manually, 73% of participants found the script of data preparation, they read the script in details to find the answer. 27% of participants chose to look for the source dataset directly by opening different datasets and try to find common columns. 7% of participant found a dataset which has a column with the same name of one column in the dataset "*cancer_breast.csv*" in a another project/folder, so he submitted the wrong answer.

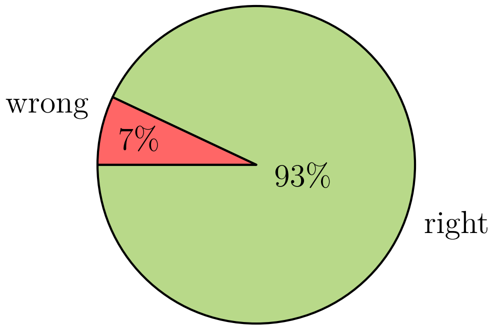
When searching with DAMSS, all participants checked the lineage tab of breast cancer dataset and found the right answer. 7% participant found the right tab, he saw the lineage of the dataset, but he did not read labels of nodes with attention. He answered the question with the process node instead of the data source of the process.

7.3.2.4 Question 4

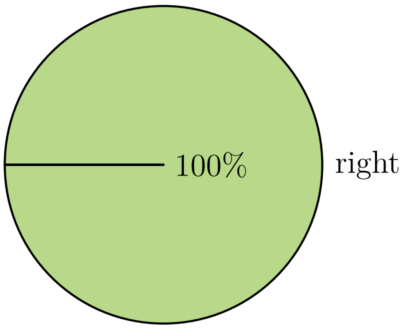
Question 4 is about checking if users can understand the main operations of a data preparation process: *"Do you understand the* ***main operations*** *(such as cleaning, aggregation, merging, etc.) used in the transformation script?"*

Result

The result of the question 4 is shown in Fig. 7.10a-7.10b. From Fig. 7.10a, we can see that 93% of participants found the entirely correct answer whereas 7% of them submitted the wrong answer. From Fig. 7.10b, 100% of participants submitted the entirely correctly answer.



(a) Question 4 result - searching manually



(b) Question 4 result - searching with DAMMS

Figure 7.10: Question 4 result

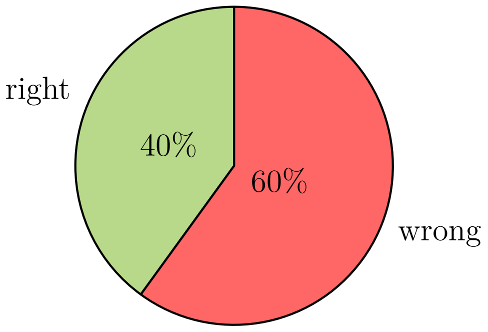
Analysis

From the result, we can see that DAMMS improves the preparation process understanding. A participant who never programmed could not understand the python script of transformation but she found the main steps of data transformation with DAMMS.

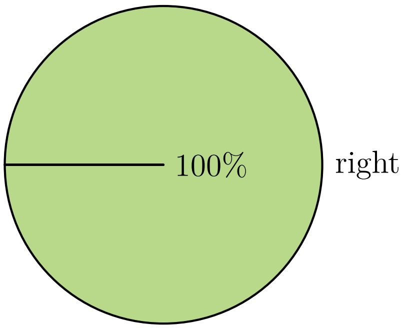
Note that in order to ensure that the experiment will not be too long, the script that we prepared for the experiment is short and easy to understand. With longer or more complicated script, the advantages of the system will be more obvious.

7.3.2.5 Question 5

Question 5 is about the missing value count to prepare an analysis: *"To analyze breast cancer, "obesity" is a feature in the dataset, can you find there are* ***how many missing values*** *in the "obesity" column?"*



(a) Question 5 result - searching manually



(b) Question 5 result - searching with DAMMS

Figure 7.11: Question 5 result

Result

The result of the question 5 is shown in Fig. 7.11a-7.11b. From Fig. 7.11a, we can see that only 40% of participants found the right answer while manually searching whereas 60% participants did not. From Fig. 7.11a, we note that all of the participants submitted the right answer with the help of DAMMS.

Analysis

From the result, we can see that DAMMS improved the fulfillment ratio from 40% to 100%.

When searching manually, 80% of participants converted the CSV file to EXCEL whereas 20% of them did not know how to process CSV files. Among the 80% of participants who finally worked on the EXCEL file, 50% of them found the right answer by using the function *COUNTBLANK* or sorting data then count or selecting the column then check the automatically counted information; the other 50% of participants did not find the right answer even they worked on the EXCEL file because they did not know the function or they used wrong functions. The 20% of participants who did not convert the CSV file gave up either directly or after trying by counting manually.

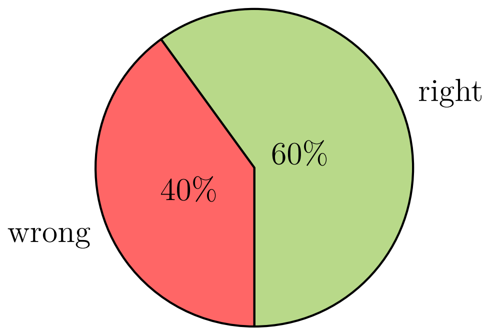
When searching with DAMMS, 100% of participants found the correct answer by checking the statistics metadata of the attributes in the dataset.

7.3.2.6 Question 6

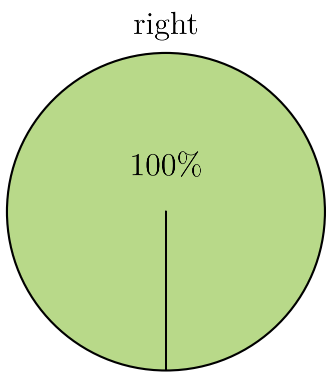
Question 6 is about finding the analysis basic information, in particular, its type: *"The dataset "CHSI" was already used for a machine learning analysis, can you find out the* ***type of the analysis*** *(such as descriptive, predictive)?"*

Result

The result of the question 6 is shown in Fig. 7.12a-7.12b. From Fig. 7.12a, we can see that, when searching manually, 60% of participants submitted the right answer whereas 40% of them did not. From Fig. 7.10b, we note that, with the help of DAMMS, all the participants submitted the right answer.



(a) Question 6 result - searching manually



(b) Question 6 result - searching with DAMMS

Figure 7.12: Question 6 result

Analysis

To find information of performed analyses, DAMMS can improve the effectiveness of searching.

When searching manually, participants had to locate the script of data analysis, read the code in detail and try to understand the code to find out the type of analysis. All the participants found the script but only 60% of them understand to code.

When searching with DAMMS, participants did not have to read the script line by line to try to understand the analysis. They can see the basic information of analysis directly, therefore, the ratio of correctness is 100%.

7.3.3 Efficiency of DAMMS

To analyse the efficiency of DAMMS, we observed the time consumed for both experiences and the number of clicks to access the information.

Time spent

From the time consumed by users with and without DAMMS (see Fig.7.13), we notice a significant gain of time (saved 46% of time in general), especially to access to information on datasets and analyzes.

Regarding questions about datasets (Q1 and Q2), a large margin is observed using DAMMS (saved 57% of time for the two questions) while stressing that the results are correct by referring to figure 7.7a-7.7b. For Q1, 78% of time was saved by using DAMMS. The average time spent on searching manually is 2 to 3 times longer than searching with DAMMS. In particular, the max time of searching manually is about five time longer than the max time of searching with DAMMS. For Q2, the difference of time spent on searching with or without DAMMS is not as significant as Q1 (24% of time was saved), but we also note that the time spent on searching manually is longer than when using DAMMS.

Regarding questions about processes (Q3 and Q4), we can always mark the shortening of time when using DAMMS (9% of time was saved). Especially, for Q3 and Q4, the max time spent on searching manually are outliers (plotted as individual points).

Regarding questions about analyses (Q5 and Q6), we underline a significant saving of time when using DAMMS (saved 54% of time). Note that for the question 5, 20% of participants gave up on finding the number of missing values for a column after a short try, if users are obliged to find the right answer, the gain of time will be more important.

Number of clicks

By observing the results, the number of clicks are positively correlated to the time spent on information search. In general, 55% of clicks were saved.

We notice that depending on the difficulty of the questions the number of clicks can explode by doing the exploration manually. For instance, for the Q1, the number of clicks when searching manually is about 14 times bigger than searching with DAMMS (93% of clicks are saved). Note that we only prepared a small simulated data lake which contains 9 projects (141 folders and 85863 files), considering the sacalability of a data lake in the real life, the difference will be more extensive.

Knowing that in the data analysis process, the search of datasets, transformation processes and analyses are essential tasks to help users to access and reuse existing elements, search with metadata is highly useful for data lake users. Moreover, DAMMS offers rapid access.

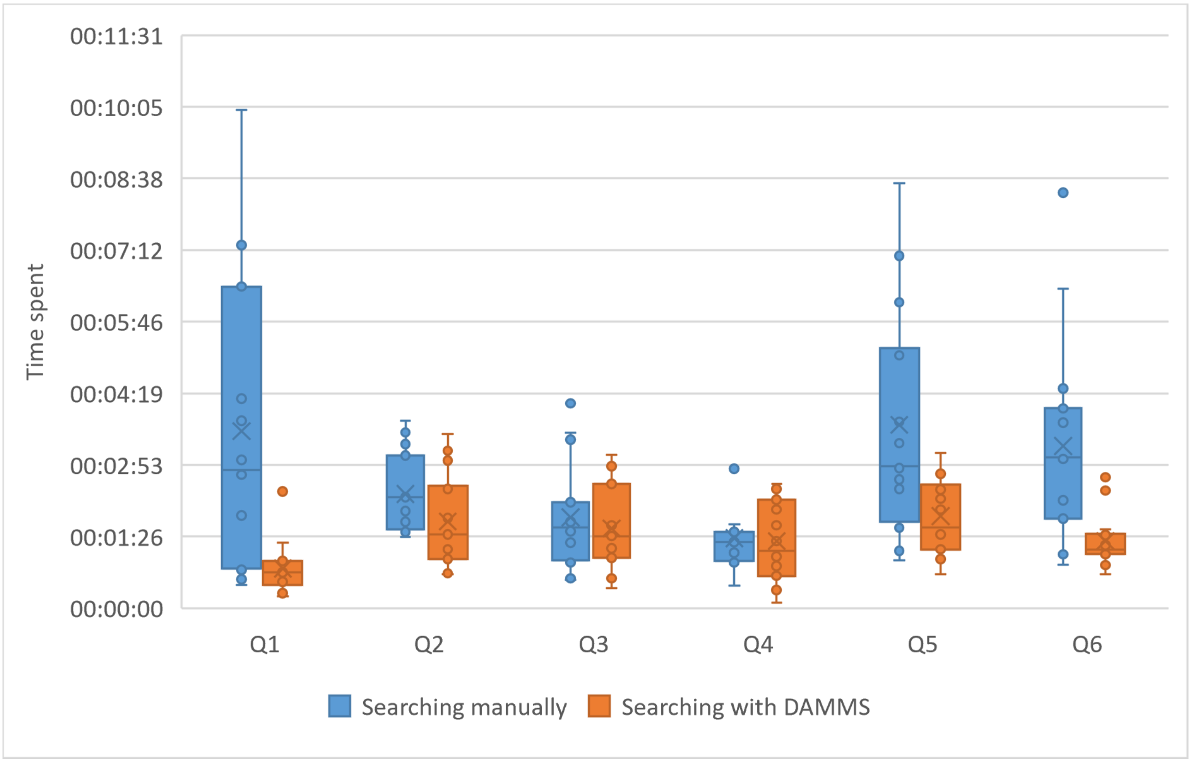


Figure 7.13: Time spent on answering different questions

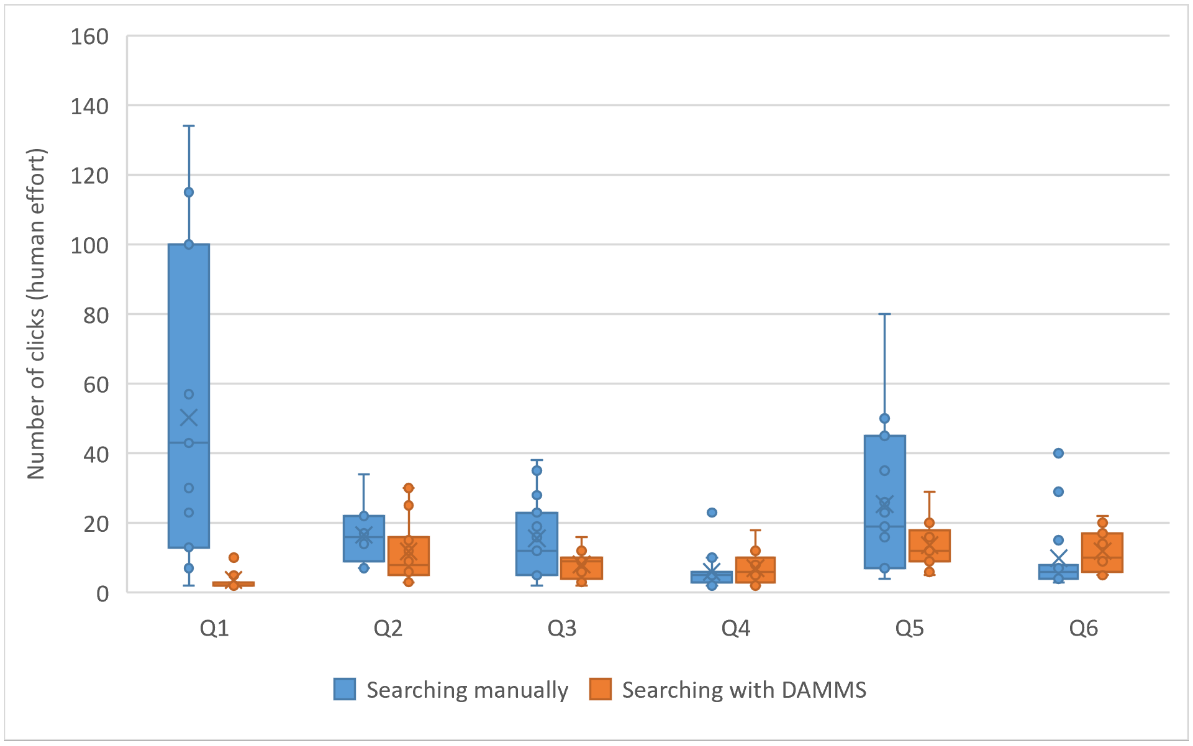


Figure 7.14: Number of clicked used on answering different questions

7.3.4 Satisfaction of DAMMS

Regarding the satisfaction of DAMMS, we measure it in two aspects: (i) SUS score and (ii) practicality of interfaces.

System Usability Scale

The SUS consists of ten questions that collect the subjective point of view of users on a system. In the second part *(part B. usability)* of the questionnaire, the questions from Q8 to Q17 are the 10 questions of SUS.

SUS gives a single score representing a composite measure of the general usability of the system studied. We followed the following steps to calculate the SUS score (Brooke, 1995):

- For odd-numbered questions (questions 1, 3, 5, 7, 9), with positive consonance, we subtract 1 from the result given by the respondent. For instance, if the respondent answers 4, the corresponding score is 3 (4-1).
- For even-numbered questions (questions 2, 4, 6, 8, 10), with negative consonance, the score is equal to 5 minus the score given by the respondent. For instance, if the respondent answers 3, the score is 2 (5-3).
- We add up all the scores and we multiply the total score by 2.5. This gives us the SUS score between 0 and 100.

The SUS score of DAMMS is **77.33** out of 100, according to (Bangor et al., 2008), DAMMS is at the 3rd quartile ranges, it is acceptable and can be classified as a good system (see Fig.7.15).

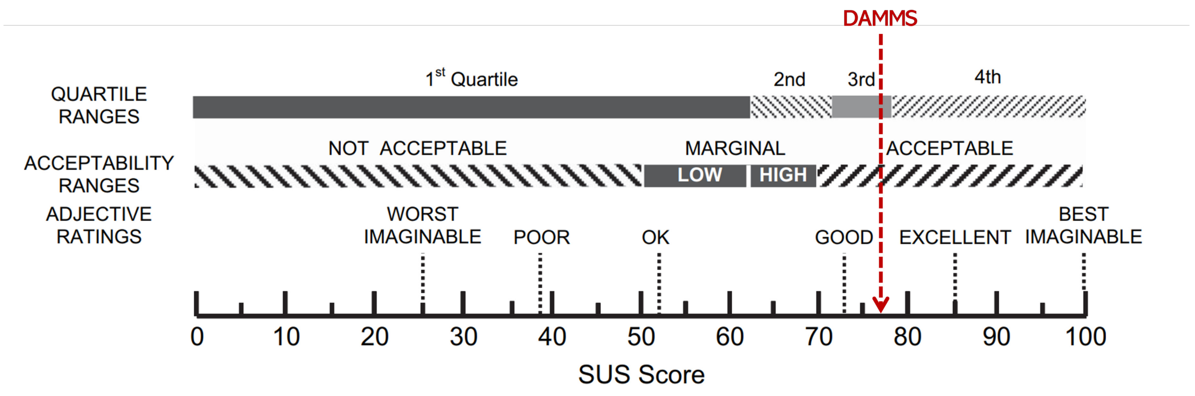


Figure 7.15: SUS score of DAMMS

System Interfaces

Besides the SUS score, we also want to check if the designed and developed interfaces conform to the usage habits of users. Therefore, we prepared 5 questions in the questionnaire (Q19 to Q23) to ask users opinions in different aspects about interface design.

The result is shown in Fig. 7.16, globally, participants are satisfied with the interface with the score of 4.19 out of 5. We note that 93% of participants think that the system is fast and stable to use. 87% of participants think that the system has beautiful and clear interface and layout and 80% of participants think that the information presentation is full and comprehensive. However, participants think that the navigation classification is less clear and the system is not perfectly easy and smooth to use.

Regarding the navigation of the system, some participants think that the operation interface is too streamlined, many functions can only be used in the second-level or third-level directory and cannot be found on the homepage at the first time.

Regarding the smoothness of use, some participants think that the interactivity should be improved, for instance, in the result panel, when a node in the graph is clicked, the system should present all the information of the node directly.

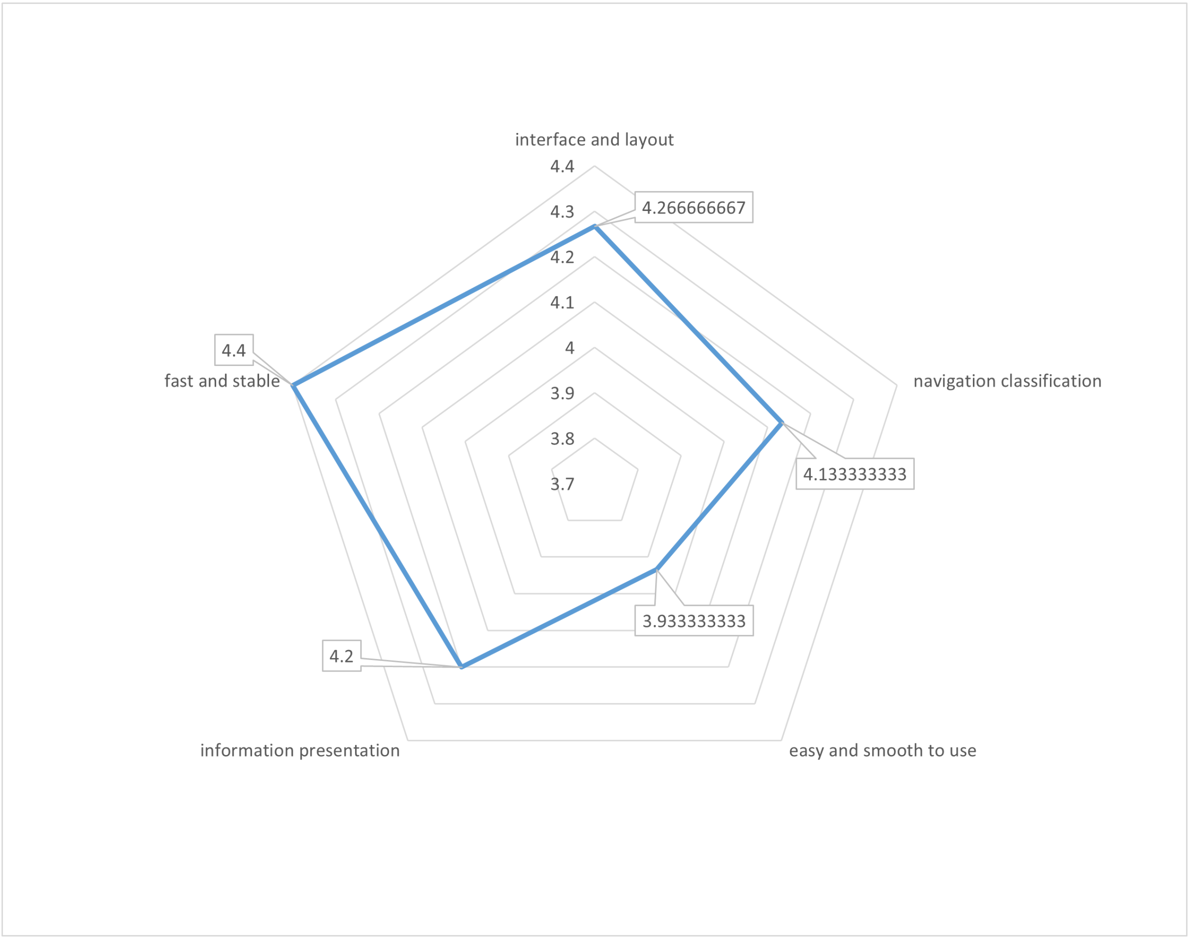


Figure 7.16: Evaluation of interfaces of DAMMS

7.4 Conclusion

In this chapter, we evaluated our system DAMMS according to quantitative and usability aspects.

Regarding the quantitative evaluation, we studied the performance of metadata generation in DAMMS. The carried out experiments were done with data sources that vary in type and size. Moreover, we evaluate the experiment results according to the size of generated metadata and time spent generation metrics. For structured datasets, we conclude that the volume of generated metadata is not necessarily linked to the size of datasets but the number of tables and columns. The time spent on metadata generation increases when the number of tables/columns increases for ORACLE databases while it increases when the number of tables/columns and instances increases for PostgreSQL databases. For semi-structured datasets, the similar conclusion can be summarized. For unstructured datasets, the metadata generation is quick and the size of generated metadata is small regardless of the dataset format and size. From the point of view of volume of data sources, we did not focus on very large volumes of data but we studied medium volumes of data.

Regarding the usability evaluation, we analyzed user experiences of data lake exploration with DAMMS. The participants of the experiments vary in experience of data analysis, knowledge of the concept of DL and hands-on experience of DL system. The results of experiment were analyzed in three aspects: effectiveness, efficiency and satisfaction. Regarding the effectiveness, DAMMS improved the accuracy and completeness ratio (31%) of data lake exploration result. Regarding the efficiency, DAMMS can help users to have a significant gain of time consumption (46%) and human effort (55%). Regarding the satisfaction, the system usability scale of DAMMS is 77.33 out of 100 which means users think the system is acceptable and is in the good range. Moreover, users think the system have globally clear and comprehensive interfaces to use with an obtained note of 4.19 out of 5.

Chapter 8

Conclusion

Contents

8.1 Assessments and contributions

In this work, we focused on one of the most popular Big Data analytics solutions: **Data Lake**. Data lakes are essential for the use as well as the sustainability of analyzes. Data lake governance is the core to ensure the industrialization of data science. The concept of data lake was brought up in 2010 (Dixon, 2010) and has been studied in depth in the last seven years. To contribute to such a relatively new concept, we have made various proposals.

8.1.1 Contributions on Data Lake Concept

At the beginning of the thesis, the concept of data lake was rarely recognized by the research community; it had no standard definition nor acknowledged architecture. Therefore, to clearly define the concept of data lake, we proposed a more generic definition and a more complete functional architecture than the state of the art.

Data Lake Definition

We emphasized that a data lake is not just a simple data storage space but a big data analytics solution which is in charge of all the different phases of data life-cycle (ingestion, preparation, analysis). Our definition has the advantages of being more complete than the literature by indicating that different types of users (data scientists, data analysts, BI professionals, etc.) can carry out different activities (ingest, process, analyze, govern) for different structural types of datasets (structured, semi-structured and unstructured).

Data Lake Functional Architecture Definition

In the extension of the previous definition, we have formalized a complete and generic data lake functional architecture. Regarding the completeness, we proposed 3+1 zones: the three zones (ingestion, preparation, analysis) are defined to manage the data life-cycle for decision making and the one zone (data governance) is defined to manage the entire data lake. Regarding the generalization, contrary to the solutions of the state of the art, we have defined an architecture that can meet the expectations of different users for doing different activities (ingest, process, analyze, govern). Moreover, each zone has standard functionalities (processing and storage).

8.1.2 Contributions on Metadata Management of Data Lake

To ensure data lake governance, we proposed a solution of metadata management which includes a metadata model and the processes of metadata generation.

Metadata Model

Our metadata model is formalized and it can be adapted to different types of data sources and all phases of data life-cycle (ingestion, preparation and analysis). This solution has the following characteristics:

- Completeness of the data life-cycle (not limited to the data catalog). We have defined a complete metadata model which covers all the stages of the data life-cycle in a data lake (ingestion, preparation and analysis).

  - Ingestion is the phase during which data are moved from one or more sources to a destination. For this phase, we proposed ingestion metadata to ensure the findability and reuse of datasets. The proposed metadata can be adapted to various types of data sources (e.g. IoT objects, databases, files), different types of datasets (structured, semi-structured and unstructured) and ingestion modes (batch and real-time). These metadata can not only describe each single DL element (e.g a dataset or an ingestion process) but the relationships between different elements.
  - Preparation is a set of acts that transform raw data into a form which is appropriate to be analyzed. It is commonly agreed that up to 80% of a data wrangler time can be spent on data preparation. For this phase, we proposed process metadata to help users to easily find, understand and reuse existing processes. The proposed metadata can be adapted to different types of data processes (transformation process of ETL used in BI, data mining, machine learning) and ingestion modes (batch and real-time). Moreover, these metadata cover different categories of process information (technical, business, operation). In particular, the operation metadata is based on a predefined list of coarse-grained operations which are formalized by a controlled language.
  - Analysis is the activity during which different data are recovered and examined to help on decision-making. For this phase, we proposed analysis metadata to help users to search, analyze, reuse existing datasets and analyses. The proposed metadata capitalize all relevant experiences including not only descriptive information on datasets and attributes, but also analytical information on performed analyses of these datasets. With the help of these metadata, users are able to select features, choose algorithms, analysis models or establish reports based on existing experiences.

- Complete vision of metadata. The proposed metadata contains three levels of information : intra-, inter- and global. Intra-metadata describe each single DL element to help users to understand different elements easier and faster. Inter-metadata describe the relationships between different DL elements (e.g. provenance, containment, similarity) to improve the efficacy of element finding, version tracing and analysis enrichment. Global metadata are applied to the entire data lake instead of specific DL element, such as common knowledge and DL logs to reduces user learning costs.

- A formalized metadata structure. We formalized the proposed metadata by UML class schema to clearly present different types (classes) of metadata and the details (attributes) of each type. The formalized and detailed model is more comprehensive than the state of the art and can facilitate the findability, accessibility, interoperability and reuse (FAIR) of data and metadata. Moreover, we proposed a solution to translate UML class diagram to Neo4j Cypher language to ensure the implementation of the proposed metadata model.

Metadata Generation Processes

Besides the metadata model, we also proposed formalized metadata generation processes. To ensure a semi-automatic extraction of metadata, we formalized the processes of metadata generation of different phases (ingestion, preparation and analysis) through BPMN schemas and algorithms.

8.1.3 Contributions on Metadata Management System of Data Lake

Implemented Data Lake Metadata Management System (DAMMS)

We implemented DAMMS (DAta lake Metadata Management System) to validate our proposed metadata model and metadata generation processes. DAMMS allows different types of users to (i) generate different types of metadata and (ii) explore metadata to have access to different elements (datasets, processes and analyses) stored in the data lake. DAMMS has the following advantages:

- It helps users to generate metadata of different structural types of datasets in a semi-automatic way. Users do not have to enter tedious information to manage or maintain metadata.
- It has ergonomic interfaces dedicated to metadata exploration and consultation that can be used by different types of users (e.g. data scientists, analysts, BI professionals). The search results are displayed through a hierarchical navigation to help users to easily understand the results.
- It is accessible for users having advanced Cypher skills to query metadata with more complex scenarios.

Evaluation of DAMMS

We assessed DAMMS according to quantitative and usability point of views. Regarding the quantitative evaluation, we analyzed the performance of metadata generation from different types of datasets and different sizes. Regarding the usability evaluation, we analyzed user experiences when exploring data lake in three aspects: effectiveness, efficiency and satisfaction. In general, when exploring data lake with DAMMS, the effectiveness was improved 31%, the efficiency was improved 50% (46% of time consumption and 55% of human effort (clicks) are saved). Moreover, DAMMS is noted 77.33 out of 100 for the system usability scale score and 4.19 out of 5 for the system interfaces.

8.2 Future Work

During this thesis, we have made several contributions regarding the concept of data lake, its metadata management as well as its metadata management system. However, new directions of research are opened up according to our work. In this section, we introduce future research perspectives in short, mid and long terms.

8.2.1 Short-term plan

Extension of DAMMS

In the short term, we will extend DAMMS with more user functionalities and a better performance in the generation of all types of metadata from all type of datasets.

Regrading user functionalities, we will complement DAMMS (i) to allow users to integrate other types of datasets (e.g. MySQL, NoSQL, NewSQL, XML, JSON, etc.) and (ii) to ingest other types of DL elements (processes and analyses) and generate automatically as many corresponding metadata as possible.

Regarding the performance of metadata generation, currently, for CSV files and PostgreSQL, the execution time of metadata generation evolves exponentially with the volume of raw data in DAMMS. To address this deficiency, we plan to modify the program scripts or apply new APIs to improve the performance.

Continuation of DAMMS assessment

We also plan to pursue the assessment of DAMMS in the aspects of quantitative and usability. Regarding quantitative evaluation, from the point of view of volume of data sources, we will test larger volumes with a wider scope on the types of data sources. Regarding usability evaluation, we will engage more participants to join the experimentation with a more sophisticated protocol. Their feedback and suggestions can help us to improve DAMMS.

8.2.2 Mid-term plan

Application of DAMMS on real world projects

So far, our system is validated on open-source data for reasons of data confidentiality. In the mid-term, we plan to set up the system DAMMS on real projects and for real users. The University Hospital of Toulouse (CHU of Toulouse) is an environment to be developed. For the existing experimental assessments, we ingested the MIMIC dataset and transformed it to the OMOP CDM as a similar project at the CHU of Toulouse which aims to transform a local Electronic Health Record (EHR) database to OMOP CDM database (Zhao et al., 2021a). In addition, we integrated the project EBERS which aims to extract and analyse different medical reports from the EHR database (Megdiche et al., 2021). In future, we plan to apply our solution to more complete and larger projects for the CHU of Toulouse and other organisations over a longer period and involving more users. The results of the experimental protocol will be compared with the results obtained by the current evaluations.

Extension of Data Security and Quality Metadata Management

Our metadata model, for data ingestion, includes security and quality information. These two aspects will be further enriched, particularly with regard to the question of the Big Data veracity.

Data security, data confidentiality, integrity and availability, also known as the *CIA triad*, should be ensured (Covert et al., 2020). To guarantee data security, data lake can integrate functions such as privacy control and fully supervised access control (Bertino and Ferrari, 2018; Zhang, 2018). Regarding data quality in a data lake, a quality metrics that can be applied on different types of datasets is required (Gómez et al., 2018; Gyulgyulyan et al., 2019) to ensure the quality of various types of datasets.

8.2.3 Long-term plan

Our proposed metadata model in this thesis is generic and can adapt to different data lakes or data management systems. In a long term, we plan to set up the metadata model on open systems or multi-data lake systems (Gorelik, 2020) to ensure interoperability of different systems. These system can be internal, such as multiple data lakes, data warehouses and data marts, or external, such as open systems from which we do not ingest data but we need to link to their data with metadata.

Annexes

Appendix A

Scripts to Generate Metadata

```
module.exports.ingestFromOracle = (datasetSource, datasetDatalake, eC, attribute, tags) => {
  var t1 = Date.now()
  var session = driver.session();
  query = ‘MERGE (dsource:DatasetSource {name:’‘ + datasetSource[0].name + ‘’,owner:’‘ + datasetSource[0].owner + ‘’ ,type:’‘ + datasetSource[0].type + ‘’ , uuid: apoc.create.uuid()})
      MERGE (dsDl:DLStructuredDataset {name:’‘ + datasetSource[0].name + ‘’,size:’‘ + datasetSource[0].size + ‘’ ,type:’‘ + datasetSource[0].type + ‘’ , uuid: apoc.create.uuid(), creationDate: datetime()})
      CREATE (ingest:Ingest {ingestionMode:’Batch’, ingestionStartTime : datetime()})
      CREATE (dsource)<-[:ingestFrom]-(ingest)-[:ingestTo]->(dsDl)
      MERGE (rds:RelationshipDS {name:’Contains’, description:’The database is part of a bigger database’})‘
  for (var t = 0; t < tags.length; t++) {
    query += ‘MERGE (t‘ + t + ‘:Tag {name:’‘ + tags[t].name + ‘’}) MERGE (t‘+ t + ‘)<-[:hasTag]-(dsDl)‘
  }
  for (var i = 0; i < datasetDatalake.length; i++) {
    query += ‘MERGE (dsDl‘ + i + ‘:DLStructuredDataset {name: ’‘ + datasetDatalake[i].name + ‘’, uuid: apoc.create.uuid()})
                CREATE (aDSR‘+ i + ‘:AnalysisDSRelationship {name:’contains’})
                CREATE (aDSR‘+ i + ‘)-[:hasRelationshipDataset]->(rds)
                CREATE (dsDl)<-[:withDataset]-(aDSR‘+ i + ‘)-[:withDataset]->(dsDl‘ + i + ‘)‘
    for (var j = 0; j < eC.length; j++) {
    if (eC[j][0] == datasetDatalake[i].name) {
      query += ‘CREATE (ec‘ + i + ‘0‘ +j + ‘:EntityCLass {name:’‘ + eC[j][1].name + ‘’,numberOfAttributes:’‘ + eC[j][1].numberOfAttributes + ‘’})
                    CREATE (ec‘ + i + ‘0‘ +j + ‘)<-[:hasEntityClass]-(dsDl‘ + i + ‘)‘
      for (var k = 0; k < attribute.length; k++) {
        if (eC[j][1].name == attribute[k][0]) {
          if (attribute[k][1].type == ’Numeric’){
            query += ‘CREATE (att‘ + i + ‘0‘ +j + ‘0‘ +k + ‘:NumericAttribute {name:’‘ + attribute[k][1].name + ‘’, type: ’‘ + attribute[k][1].type + ‘’ , min: ’‘ + attribute[k][1].min +‘’ , max: ’‘ + attribute[k][1].max + ‘’, missingValue: ’‘ + attribute[k][1].missingValue + ‘’})CREATE (att‘ + i + ‘0‘ +j + ‘0‘ +k + ‘)<-[:
```

```
    hasAttribute]-(ec‘ + i + ‘0‘ +j + ‘)‘
          } else {
            query += ‘CREATE (att‘ + i + ‘0‘ +j + ‘0‘ +k + ‘:
    NominalAttribute {name:’‘ + attribute[k][1].name + ‘’, type: ’‘ +
    attribute[k][1].type + ‘’})CREATE (att‘ + i + ‘0‘ +j + ‘0‘ +k + ‘)
    <-[:hasAttribute]-(ec‘ + i + ‘0‘ +j + ‘)‘
          }
        }
      }
   }
   }
}
```

Listing A.1: Script of ORACLE database metadata storing

```
MERGE (dsource:DatasetSource {name:'mimic',owner:'MIT Laboratory
    computational ' ,type:'structured dataset' , uuid: apoc.create.uuid()
    })
MERGE (dsDl:DLStructuredDataset {name:'mimic',size:'1.9000000000000001
    Go' ,type:'structured dataset' , uuid: apoc.create.uuid(),
    creationDate: datetime()})
CREATE (ingest:Ingest {ingestionMode:'Batch', ingestionStartTime :
    datetime()})
CREATE (dsource)<-[:ingestFrom]-(ingest)-[:ingestTo]->(dsDl)
MERGE (rds:RelationshipDS {name:'Contains', description:'The database is
     part of a bigger database'})MERGE (t0:Tag {name:'opensource'})
MERGE (t0)<-[:hasTag]-(dsDl)MERGE (t1:Tag {name:'healthcare'})
MERGE (t1)<-[:hasTag]-(dsDl)MERGE (t2:Tag {name:'laboratory'})
MERGE (t2)<-[:hasTag]-(dsDl)MERGE (dsDl0:DLStructuredDataset {name: '
    INTRO_USER', uuid: apoc.create.uuid()})
CREATE (aDSR0:AnalysisDSRelationship {name:'contains'})
CREATE (aDSR0)-[:hasRelationshipDataset]->(rds)
CREATE (dsDl)<-[:withDataset]-(aDSR0)-[:withDataset]->(dsDl0)CREATE (
    ec000:EntityCLass {name:'LABEVENTS',numberOfAttributes:'9'})
CREATE (ec000)<-[:hasEntityClass]-(dsDl0)CREATE (att0000196:
    NominalAttribute {name:'FLAG', type: 'VARCHAR2'})
CREATE (att0000196)<-[:hasAttribute]-(ec000)CREATE (att0000197:
    NominalAttribute {name:'VALUEUOM', type: 'VARCHAR2'})
CREATE (att0000197)<-[:hasAttribute]-(ec000)CREATE (att0000198:
    NumericAttribute {name:'VALUENUM', type: 'Numeric' , min: 'null' ,
    max: 'null' , missingValue: '0' })
CREATE (att0000198)<-[:hasAttribute]-(ec000)CREATE (att0000199:
    NominalAttribute {name:'VALUE', type: 'VARCHAR2'})
```

Listing A.2: Excerpt of script of OT database metadata storing

Appendix B

Scripts to Explore Datasets

```

//Function to search dataset metadata, with parameters for each filter.
module.exports.getDatabases = (tags, type = 'defaultValue', creationdate = '0001-01-01T00:00:00Z', quality = [], sensitivity = 0, entityAttributeNames = "") => {
  var session = driver.session();
  //Cypher query with ifs to have the dataset type filter used.
  var query = "MATCH (ds),(a),(e:EntityClass) WHERE ("; //,(q:QualityMetric),(s:SensitivityMark), (sv:SensitivityValue)
  if (!type.includes("Structured") && !type.includes("Semi-Structured") && !type.includes("Unstructured")) {
    query += " ds:DLStructuredDataset OR ds:DLSemistructuredDataset OR ds:DLUnstructuredDataset ";
  } else {
    if (type.includes("Structured") && type.includes("Semi-Structured") && type.includes("Unstructured")) {
      query += " ds:DLStructuredDataset OR ds:DLSemistructuredDataset OR ds:DLUnstructuredDataset ";
    } else {
      if (type.includes("Structured") && type.includes("Semi-Structured")) {
        query += " ds:DLStructuredDataset OR ds:DLSemistructuredDataset ";
      } else {
        if (type.includes("Structured") && type.includes("Unstructured")) {
          query += " ds:DLStructuredDataset OR ds:DLUnstructuredDataset ";
        } else {
          if (type.includes("Semi-Structured") && type.includes("Unstructured")) {
            query += " ds:DLSemistructuredDataset OR ds:DLUnstructuredDataset ";
          } else {
            if (type.includes("Semi-Structured")) {
              query = query + "ds:DLSemistructuredDataset";
            } else {
              if (type.includes("Unstructured")) {
                query = query + "ds:DLUnstructuredDataset ";
              } else {
                if (type.includes("Structured")) {
                  query = query + "ds:DLStructuredDataset";
                }
              }
            }
          }
        }
      }
    }
  }
  query = query + ") AND (";
  for (var i = 0; i < tags.length; i++) {
    if (i != tags.length - 1) {
      query = query + "toLower(ds.name) CONTAINS toLower('" + tags[i] + "') OR toLower(ds.description) CONTAINS toLower('" + tags[i] + "') OR "
    }
    else {
      query = query + "toLower(ds.name) CONTAINS toLower('" + tags[i] + "') OR toLower(ds.description) CONTAINS toLower('" + tags[i] + "')"
    }
  }
  //Cypher query for dates filter
  query = query + ' ) AND (datetime(ds.creationDate) >= datetime("' + creationdate + '")) RETURN distinct ds '

  //Cypher query for the quality filter
  if(quality.lenght>0){
    query += "AND (n)-[qv:qualityValue]-(q) AND ("
    for( var i=0; i<quality.length; i++){
      if(i!=quality.length -1){
        query += "toLower(q.name) CONTAINS toLower("+ quality[i][0] +") AND qv.value >= "+ quality[i][1] +" OR" ;
      }else{
        query += "toLower(q.name) CONTAINS toLower("+ quality[i][0] +") AND qv.value >= "+ quality[i][1] +" )  "
      }
    }
  }

  //Cyper query for the sensitivity filter
  if(sensitivity != 0){
    query += "(n)-[:hasSensitivity]-(s)-[:hasValue]-(sv) AND (sv.value >= "+ sensitivity +" )";
  }

  //Cypher query for the entity class filter
  if (entityAttributeNames.length > 0) {
    query += "AND (a:NominalAttribute OR a:NumericAttribute) AND ((ds)-[:hasEntityClass]->(e:EntityClass)-[:hasAttribute]->(a)) AND ( toLower(e.name) CONTAINS toLower('" + entityAttributeNames + "') OR toLower(a.name) CONTAINS toLower('" + entityAttributeNames + "'))"
  }

  //Cypher query that allow a dataset to not have a Tag, else it is not taken in account
  query = query + ' union MATCH (ds)-[:hasTag]->(t:Tag) WHERE ( '
  for (var i = 0; i < tags.length; i++) {
    if (i != tags.length - 1) {
        query = query + "toLower(t.name) CONTAINS toLower('" + tags[i] + "') OR "
      }
      else {
        query = query + "toLower(t.name) CONTAINS toLower('" + tags[i] + "')"
      }
  }

  query = query + ") RETURN distinct ds"
}
```

Listing B.1: Script to find datasets with keywords and filters

```


//Get information of lineage

query = `MATCH path = allshortestpaths ((ds:DatasetSource)-[*]-(d))
         WHERE NONE(n IN nodes(path) WHERE n:Tag OR n:Operation OR n:AnalysisDSRelationship) AND (d:DLStructuredDataset OR d:DLSemistructuredDataset OR d:UnstructuredDataset) AND NONE(x IN nodes(path) where exists(()-[:hasSubprocess]->(x:Process))) AND d.uuid = '` + $(this).attr('id').split('$')[2] + `'
         RETURN path
         UNION ALL
         MATCH path = allshortestpaths ((sos:SourceOfSteam)-[*]-(d))
         WHERE NONE(n IN nodes(path) WHERE n:Tag OR n:Operation OR n:AnalysisDSRelationship) AND (d:DLStructuredDataset OR d:DLSemistructuredDataset OR d:UnstructuredDataset) AND NONE(x IN nodes(path) where exists(()-[:hasSubprocess]->(x:Process))) AND d.uuid = '` + $(this).attr('id').split('$')[2] + `'
         RETURN path`

//Get information of Hyper Graph

query = `MATCH (d)
          WHERE (d:DLStructuredDataset OR d:DLSemistructuredDataset OR d:DLUnstructuredDataset) AND d.uuid = "` + $(this).attr('id').split('$')[2] + `"
          OPTIONAL MATCH (d)-[r:sourceData]->(p:Process)
          WHERE NOT (p)<-[:hasSubprocess]-()
          OPTIONAL MATCH (d)<-[s:targetData]-(p1:Process)
          WHERE NOT (p1)<-[:hasSubprocess]-()
          with d,p,p1,r,s
          RETURN d,p,p1,r,s
          UNION ALL
          MATCH (d)
          WHERE (d:DLStructuredDataset OR d:DLSemistructuredDataset OR d:DLUnstructuredDataset) AND d.uuid = "` + $(this).attr('id').split('$')[2] + `"
          OPTIONAL MATCH (d)-[r:sourceData]->(p:Process)
          OPTIONAL MATCH (d)<-[s:targetData]-(p1:Process)
          with d,p,p1,r,s
          WHERE NOT exists((p)<-[:hasSubprocess]-(:Process)-[]-(d)) AND NOT exists((p1)<-[:hasSubprocess]-(:Process)-[]-(d))
          RETURN d,p,p1,r,s`

```

```
//Get information of Relationship of Dataset

query = ‘MATCH (dl)<-[r1:withDataset]-(a)-[r2:hasRelationshipDataset]->(rDS:RelationshipDS),(a)-[r3:withDataset]->(dl2)
        WHERE dl.name CONTAINS ’‘ + datasetChosed[0] + ‘’ and dl.uuid = ’‘ + datasetChosed[1] + ‘’
        AND
        (dl:DLStructuredDataset OR dl:DLSemistructuredDataset OR dl:DLUnstructuredDataset)
        AND
        (dl2:DLStructuredDataset OR dl2:DLSemistructuredDataset OR dl2:DLUnstructuredDataset)
        and rDS.name=’‘+ relationDS + ‘’
        RETURN DISTINCT dl,dl2,rDS,a,r1,r2,r3‘

//Get information of Reationship of Attributes

query = ‘MATCH (dl)-[]-(e:EntityClass)-[]-(a),(a)-[r1:hasAttribute]-(AA:AnalysisAttribute)-[r2:useMeasure]-(RA:RelationshipAtt),(AA)-[r3:hasAttribute]-(a2)
         WHERE dl.uuid = ’‘ + trans + ‘’
         AND
         (a:NominalAttribute OR a:NumericAttribute OR a:Attribute) and RA.name=’‘ + relationAtt + ‘’
         RETURN DISTINCT a,r1,AA,r2,RA,a2,r3 union all MATCH (dl)-[]-()-[]-(e:EntityClass)-[]-(a),(a)-[r1:hasAttribute]-(AA:AnalysisAttribute)-[r2:useMeasure]-(RA:RelationshipAtt),(AA)-[r3:hasAttribute]-(a2)
         WHERE dl.uuid = ’‘ + trans + ‘’
         AND
         (a:NominalAttribute OR a:NumericAttribute OR a:Attribute) and RA.name=’‘ + relationAtt + ‘’
         RETURN DISTINCT a,r1,AA,r2,RA,a2,r3‘

//Get information of Similarity

query = ‘CALL gds.nodeSimilarity.stream(’graph-DDDT’)
  YIELD node1, node2, similarity
  RETURN gds.util.asNode(node1).name AS Person1, gds.util.asNode(node2).name AS Person2, similarity
  ORDER BY similarity DESCENDING, Person1, Person2‘
```

Listing B.2: Queries used to find details about a dataset

Appendix C

Scripts to Explore Processes

```
//Function to search processus metadata with parameters to apply filter.
//Attributed values are default value if no parameter is given.
module.exports.getProcesses = (tags, language = "", date = "0001-01-01",
    typeOpe = [], exeEnv = []) => {
  var session = driver.session();
  //Classic query without filter, search with name, description and tag
   name.
  var query = "MATCH (p:Process) OPTIONAL MATCH (p)-[r:hasTag]->(t :Tag)
    OPTIONAL MATCH (o:Operation)-[:isUsedBy]->(:OperationOfProcess)<-[:
   containsOp]-(p) WITH p,t,o WHERE ("
  for (var i = 0; i < tags.length; i++) {
    if (i != tags.length - 1) {
      query = query + "toLower(t.name) CONTAINS toLower('" + tags[i] + "
   ') OR toLower(p.name) CONTAINS toLower('" + tags[i] + "') OR toLower(
   p.description) CONTAINS toLower('" + tags[i] + "') OR toLower(o.name)
    CONTAINS toLower('" + tags[i] + "') OR "
    }
    else {
      query = query + "toLower(t.name) CONTAINS toLower('" + tags[i] + "
   ') OR toLower(p.name) CONTAINS toLower('" + tags[i] + "') OR toLower(
   p.description) CONTAINS toLower('" + tags[i] + "') OR toLower(o.name)
    CONTAINS toLower('" + tags[i] + "') )"
    }
  }
  //Cypher query for language filter
  if (language.length > 0) {
    query += " AND ("
    for (var i = 0; i < language.length; i++) {
      if (i != language.length - 1) {
        query += " p.programLanguage CONTAINS ('" + language[i] + "') OR
    "
      } else {
        query += " p.programLanguage CONTAINS ('" + language[i] + "') )
   "
      }
    }
  }
  //Cypher query for execution environment
  if (exeEnv.length > 0) {
    query += " AND ("
    for (var i = 0; i < exeEnv.length; i++) {
      if (i != exeEnv.length - 1) {
        query += " p.executionEnvironment CONTAINS ('" + exeEnv[i] + "') OR "
      } else {
        query += " p.executionEnvironment CONTAINS ('" + exeEnv[i] + "') ) "
      }
    }
  }
  //Cypher query for dates filter
  query = query + ' AND (datetime(p.creationDate) >= datetime("' + date + '"))'
  //Cypher query for used operation filter
  if (typeOpe.length > 0) {
    query += " AND (p)-[]-()-[]-(o:Operation) AND ("
    for (var i = 0; i < typeOpe.length; i++) {
      if (i != typeOpe.length - 1) {
        query += " o.name CONTAINS ('" + typeOpe[i] + "') OR "
      } else {
        query += " o.name CONTAINS ('" + typeOpe[i] + "') )"
      }
    }
  }
  query = query + " RETURN distinct p"
}
```

Listing C.1: Script to find processes with keywords and filters

```
//Get information of lineage

query = `MATCH path =(m)<-[:targetData]-(c:Process {uuid:'` + $(this).attr('id').split('$')[1] + `'})
      OPTIONAL MATCH (c)<-[q:hasSubprocess]-(w: Process)
      RETURN path,w,q
      UNION ALL
      MATCH path3 =(c:Process {uuid:'` + $(this).attr('id').split('$')[1] + `'})<-[:sourceData]-(d)
      OPTIONAL MATCH (c)<-[q:hasSubprocess]-(w: Process)
      RETURN path3 AS path,w,q
      UNION ALL
      MATCH path2=((dl)-[]-(i:Ingest)-[]-(p:Process {uuid:'`+ $(this).attr('id').split('$')[1] + `'})-[]-(d:DatasetSource)-[]-(sos:SourceOfSteam))
      WHERE (dl:DLStructuredDataset OR dl:DLSemistructuredDataset OR dl:DLUnstructuredDataset)
      RETURN path2 AS path, null as w, null as q`;

//Get information of hyper graph

query = "MATCH path= (p:Process {name:'" + $(this).text() + "'})-[:hasSubprocess]-(t:Process) RETURN path"

//Get information of operation

query = `MATCH (p:Process {name:"` + $(this).text() + `"})
      OPTIONAL MATCH (p)-[r3:containsOp]->(c:OperationOfProcess)
      OPTIONAL MATCH (p)-[r5:hasSubprocess]->(p1:Process)-[r1:containsOp]->(c1:OperationOfProcess)
      OPTIONAL MATCH (c)-[f:followedBy]-()
```

```
    RETURN c,c1,f, null as p, null as p1, null as r1, null as r5
    UNION ALL
    MATCH (p:Process {name:"‘ + $(this).text() + ‘"})-[r5:
hasSubprocess]->(p1:Process)-[r1:containsOp]->(c1:OperationOfProcess)
    OPTIONAL MATCH (c1)-[f:followedBy]-()
    RETURN p,p1,r1,c1,r5,f, null as c‘
```

Listing C.2: Queries to find details about a process

Appendix D

Scripts to Explore Analyses

```
//Function to search study metadata
module.exports.getStudies = (tags, type, creationdate = '0001-01-01',
   landmarker, algoNames, parameter = [], evaluation = [],omNames) => {
  var session = driver.session();
  let typeRech = Object.values(type);
  console.log('Algorithm names : ' + algoNames)
  var query = "MATCH (s:Study)-[:hasAnalysis]->(a:Analysis),(l:
   Landmarker),(al)"
  //Classic cypher query to search for study without filter.
  if(parameter.length > 0){
    query+= ',(p)'
  }
  if(evaluation.length > 0){
    query+= ',(e)'
  }
  query += "WHERE ("
  for (var i = 0; i < tags.length; i++) {
    if (i != tags.length - 1) {
      query = query + "toLower(s.name) CONTAINS toLower('" + tags[i] + "
   ') OR toLower(s.description) CONTAINS toLower('" + tags[i] + "') OR
   toLower(a.name) CONTAINS toLower('" + tags[i] + "') OR "
    }
    else {
      query = query + "toLower(s.name) CONTAINS toLower('" + tags[i] + "
   ') OR toLower(s.description) CONTAINS toLower('" + tags[i] + "') OR
   toLower(a.name) CONTAINS toLower('" + tags[i] + "') )"
    }
  }
  if (typeRech.indexOf('machineLearning') != -1 && typeRech.indexOf('
   otherAnalysis') == -1) {
    query = query + ' AND toLower(a.typeAnalysis) CONTAINS toLower("
   Machine Learning")'
  }else if (typeRech.indexOf('machineLearning') == -1 && typeRech.
   indexOf('otherAnalysis') != -1) {
    query = query + ' AND toLower(a.typeAnalysis) CONTAINS toLower("
   Other Analysis")'
  }
  if (typeRech.indexOf('machineLearning') != -1 ) {
    typeRech.splice(typeRech.indexOf('machineLearning'), 1)
  }
  if (typeRech.indexOf('otherAnalysis') != -1) {
    typeRech.splice(typeRech.indexOf('otherAnalysis'), 1)
  }
  //Cypher query for analysis type filter
  if (typeRech.length > 0) {
    query += ' AND ('
    for (var i = 0; i < typeRech.length; i++) {
      if (i != typeRech.length - 1) {
        query += ' toLower(a.subTypeAnalysis) CONTAINS toLower("' + typeRech[i] + '") OR '
      } else {
        query += ' toLower(a.subTypeAnalysis) CONTAINS toLower("' + typeRech[i] + '")  )'
      }
    }
  }
  //Cypher query for landmarkers query
  if (landmarker.length > 0) {
    query += ' AND (s)-[:hasAnalysis]->(a)-[:hasImplementation]->(l) AND ('
    for (var i = 0; i < landmarker.length; i++) {
      if (i != landmarker.length - 1) {
        query += ' toLower(l.name) CONTAINS toLower("' + landmarker[i] + '") OR toLower(l.description) CONTAINS toLower("' + landmarker[i] + '") OR '
      } else {
        query += ' toLower(l.name) CONTAINS toLower("' + landmarker[i] + '") OR toLower(l.description) CONTAINS toLower("' + landmarker[i] + '") )'
      }
    }
  }

  if(evaluation.length > 0){
    query += ' AND (s)-[:hasAnalysis]->(a)-[:evaluateAnalysis]-()-[]-(e:EvaluationMeasure) AND ( '
    for (var i = 0; i < evaluation.length; i++) {
      if (i != evaluation.length - 1) {
        query += ' toLower(e.name) CONTAINS toLower("' + evaluation[i] + '") OR '
      } else {
        query += ' toLower(e.name) CONTAINS toLower("' + evaluation[i] + '") )'
      }
    }
  }

  if(parameter.length > 0){
    query += ' AND (s)-[:hasAnalysis]->(a)-[:hasImplementation]->()-[:hasParameter]-(p:Parameter) AND ( '
    for (var i = 0; i < parameter.length; i++) {
      if (i != parameter.length - 1) {
        query += ' toLower(p.name) CONTAINS toLower("' + parameter[i] + '") OR '
      } else {
        query += ' toLower(p.name) CONTAINS toLower("' + parameter[i] + '") )'
      }
    }
  }
```

```

//Cypher query for algo filter. The database does not have all the algo type implemented so this part of query is commented.
if (algoNames.length > 0 || type.includes('algosupervised') || type.includes('algoUnsupervised') || type.includes('AlgoReinforcement')) {
  query += ' AND (s)-[:hasAnalysis]->(a)-[:hasImplementation]->()-[:usesAlgo]->(al) '
  if (type.includes('algosupervised') || type.includes('algoUnsupervised') || type.includes('AlgoReinforcement')) {
    query += 'AND ('
    if (!type.includes("algosupervised") && !type.includes("algoUnsupervised") && !type.includes("AlgoReinforcement")) {
      query += " al:AlgoSupervised OR al:AlgoUnsupervised OR al:AlgoReinforcement ";
    } else {
      if (type.includes("algosupervised") && type.includes("algoUnsupervised") && type.includes("AlgoReinforcement")) {
        query += " al:AlgoSupervised OR al:AlgoUnsupervised OR al:AlgoReinforcement ";
      } else {
        if (type.includes("algosupervised") && type.includes("algoUnsupervised")) {
          query += " al:AlgoSupervised OR al:AlgoUnsupervised ";
        } else {
          if (type.includes("algosupervised") && type.includes("AlgoReinforcement")) {
            query += " al:AlgoSupervised OR al:AlgoReinforcement ";
          } else {
            if (type.includes("algoUnsupervised") && type.includes("AlgoReinforcement")) {
              query += " al:AlgoUnsupervised OR al:AlgoReinforcement ";
            } else {
              if (type.includes("algoUnsupervised")) {
                query = query + "al:AlgoUnsupervised";
              } else {
                if (type.includes("AlgoReinforcement")) {
                  query = query + "al:AlgoReinforcement ";
                } else {
                  if (type.includes("algosupervised")) {
                    query = query + "al:AlgoSupervised";
                  }
                }
              }
            }
          }
        }
      }
    }
    query += ') '
  }
  //Cypher query to search a particular algo names.
  if (algoNames.length > 0) {
    query += ' AND (toLower(al.name) CONTAINS toLower("' + algoNames + '") OR toLower(al.description) CONTAINS toLower("' + algoNames + '")) '
  }
}
```

```

  query = query + ' AND (datetime(s.creationDate) >= datetime("' + creationdate + '"))'
  query = query + " RETURN DISTINCT s"

}

//module.exports.to get entity class by dataset
module.exports.getEntityClassByDataset = (datasetName, datasetId, typeDS) => {
  var session = driver.session();
  query = 'MATCH (e:EntityClass)<-[:hasEntityClass]-(a) WHERE '
  if (typeDS.includes("Semi-Structured")) {
    query = query + "a:DLSemistructuredDataset";
  } else {
    if (typeDS.includes("Unstructured")) {
      query = query + "a:DLUnstructuredDataset ";
    } else {
      if (typeDS.includes("Structured")) {
        query = query + "a:DLStructuredDataset ";
      }
    }
  }
  query += 'a.name = "' + datasetName + '" AND a.uuid = "' + datasetId + '" RETURN DISTINCT e';
}
```

Listing D.1: Script to find analyses with keywords and filters

```

//Get information of lineage

query = `MATCH path = shortestPath ((d:DatasetSource)-[*]-(u:Analysis {uuid:"` + $(this).attr('id').split('$')[1] + `"}))
            WHERE NONE(n IN nodes(path) WHERE n:Tag OR n:Operation OR n:AnalysisDSRelationship OR n:Study)
            RETURN path
            UNION ALL
            MATCH path = shortestPath ((d)-[*..1]-(u:Analysis {uuid:"` + $(this).attr('id').split('$')[1] + `"}))
            WHERE NONE(n IN nodes(path) WHERE n:Tag OR n:Operation OR n:AnalysisDSRelationship OR n:Study)
            AND (d:DLStructuredDataset OR d:DLSemistructuredDataset OR d:DLUnstructuredDataset)
            RETURN path`

//Get information of hyper graph

query = `MATCH (a:Analysis)
            MATCH (a)<-[r1:hasAnalysis]-(s:Study)
            MATCH (a)<-[r2:evaluateAnalysis]-(me:ModelEvaluation)-[r3:useEvaluationMeasure]->(em:EvaluationMeasure)
            WHERE
            toLower(a.name) CONTAINS toLower('`+ $(this).attr('id').split('$')[0] + `') AND a.uuid = '` + $(this).attr('id').split('$')[1] + `'
            WITH a,s,em,me,r1,r2,r3
            OPTIONAL MATCH (a)-[r4:hasImplementation]-(ld:Landmarker)
            WITH a,s,em,me,ld,r1,r2,r3,r4
            OPTIONAL MATCH (a)-[r5:hasImplementation]->(i:Implementation
    )-[r6:usesAlgo]->(al:AlgoSupervised),(i)-[r7:hasParameterSetting]->(
    ps:ParameterSetting)<-[r8:hasParameterValue]-(p:Parameter)<-[r9:
    hasParameter]-(i)
            RETURN a,s,me,em,ld,i,al,p,ps,r1,r2,r3,r4,r5,r6,r7,r8,r9`

//Get information of features

//Numeric attribute use the same query but replace all the nominal with
    numeric

query = 'Match (na:NominalAttribute),(nf:AnalysisNominalFeatures),(f:
    AnalysisFeatures),(a:Analysis),(ta:AnalysisTarget) WHERE a.uuid = "'
    + analyseId + '"  AND ((a)-[:hasFeaturesAnalysis]->(f)-[:
    hasNominalFeaturesAnalysis]->(nf)-[:hasFeatures]->(na) OR (a)-[:
    hasTargetAnalysis]->(ta)-[:hasTarget]->(na)) RETURN DISTINCT na'

//Bouton attribute
//Numeric attribute use the same query but replace all the nominal with
    numeric
query = 'Match (na:NominalAttribute),(nf:AnalysisNominalFeatures),(f:
    AnalysisFeatures),(a:Analysis),(ta:AnalysisTarget) WHERE a.uuid = "'
    + analyseId + '"  AND na.name= "' + name + '" AND ((a)-[:
    hasFeaturesAnalysis]->(f)-[:hasNominalFeaturesAnalysis]->(nf)-[:
    hasFeatures]->(na) OR (a)-[:hasTargetAnalysis]->(ta)-[:hasTarget]->(
    na)) RETURN DISTINCT na'

//Get information of relationship of attributes is the same query with
    Datasets

//Get information of entity class
```

Listing D.2: Queries used to find details about a analysis

Appendix E

Experiment Protocol

Thank you for participating in this experiment which is divided into three parts:

- Firstly, we'll show you a folder in which you can find different projects. In each project file, you may find datasets, transformation scripts and/or analysis scripts related to project. You need to answer 6 questions by looking through different projects manually. We will record the number of clicks and time that you spend on each question.
- Secondly, after a quick presentation, you will answer the same questions with the help of DAMMS (data lake metadata management system). We will record the number of clicks and time that you spend on each question.
- Finally, Finally, you can give us feedback on your experience in the form of a questionnaire.

<u>**Questions:**</u>

About datasets

1. There are different medical datasets, can you try to find how many of them are about '**breast**' cancer?
2. The dataset "*cancer_breast.csv*" is stored in "`\Exploration\CHSI\cancer_breast.csv`", in the same folder, there are two other datasets "*colon cancer*" and "*lung cancer*" stored in the same folder, can you find out if the three datasets are **similar or not**?

About processes

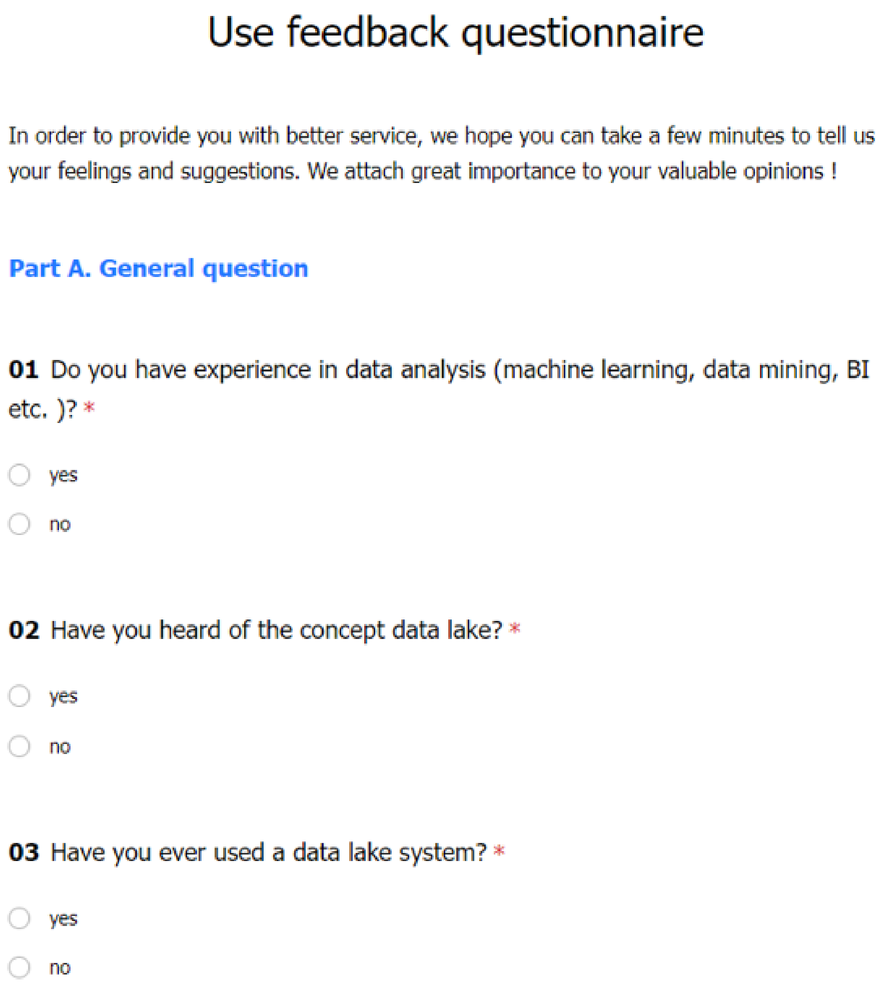
3. The dataset "*cancer_breast.csv*" was obtained through a transformation process, can you identify the **data source file** of this transformation?
4. Do you understand the **main operations** (such as cleaning, aggregation, merging, etc.) used in the transformation script?

About analyses

5. To analyze breast cancer, "*obesity*" is a feature in the dataset, can you find there are **how many missing values** in the "*obesity*" column?
6. The dataset "*chsi*" was already used for a machine learning analysis, can you find out the **type of the analysis** (such as descriptive, predictive)?

Appendix F

Questionnaire



Use feedback questionnaire

In order to provide you with better service, we hope you can take a few minutes to tell us your feelings and suggestions. We attach great importance to your valuable opinions !

Part A. General question

01 Do you have experience in data analysis (machine learning, data mining, BI etc.)? *

○ yes

○ no

02 Have you heard of the concept data lake? *

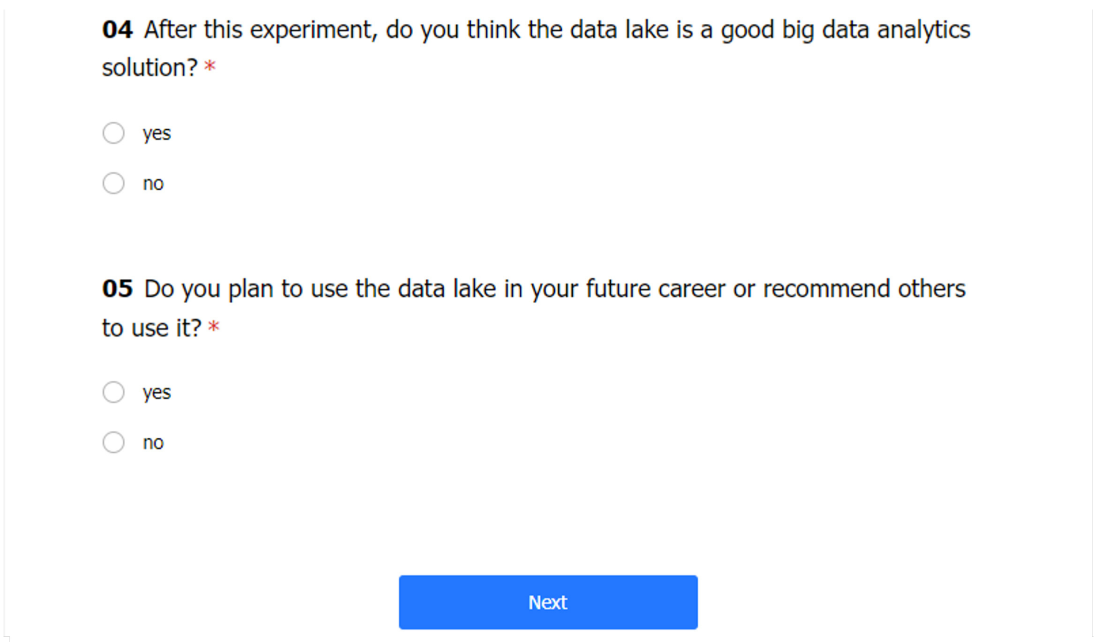
○ yes

○ no

03 Have you ever used a data lake system? *

○ yes

○ no

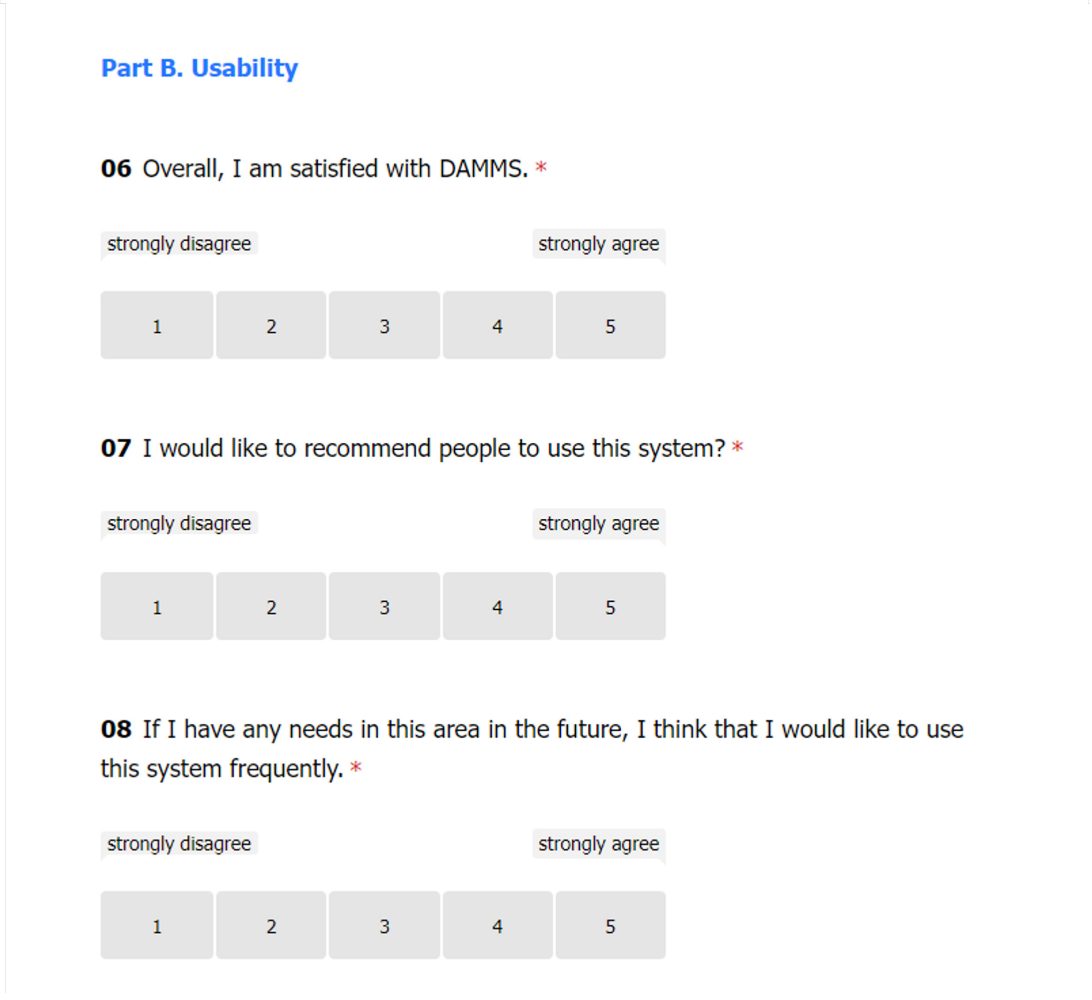


04 After this experiment, do you think the data lake is a good big data analytics solution? *

- ○ yes
- ○ no

05 Do you plan to use the data lake in your future career or recommend others to use it? *

- ○ yes
- ○ no



Part B. Usability

06 Overall, I am satisfied with DAMMS. *

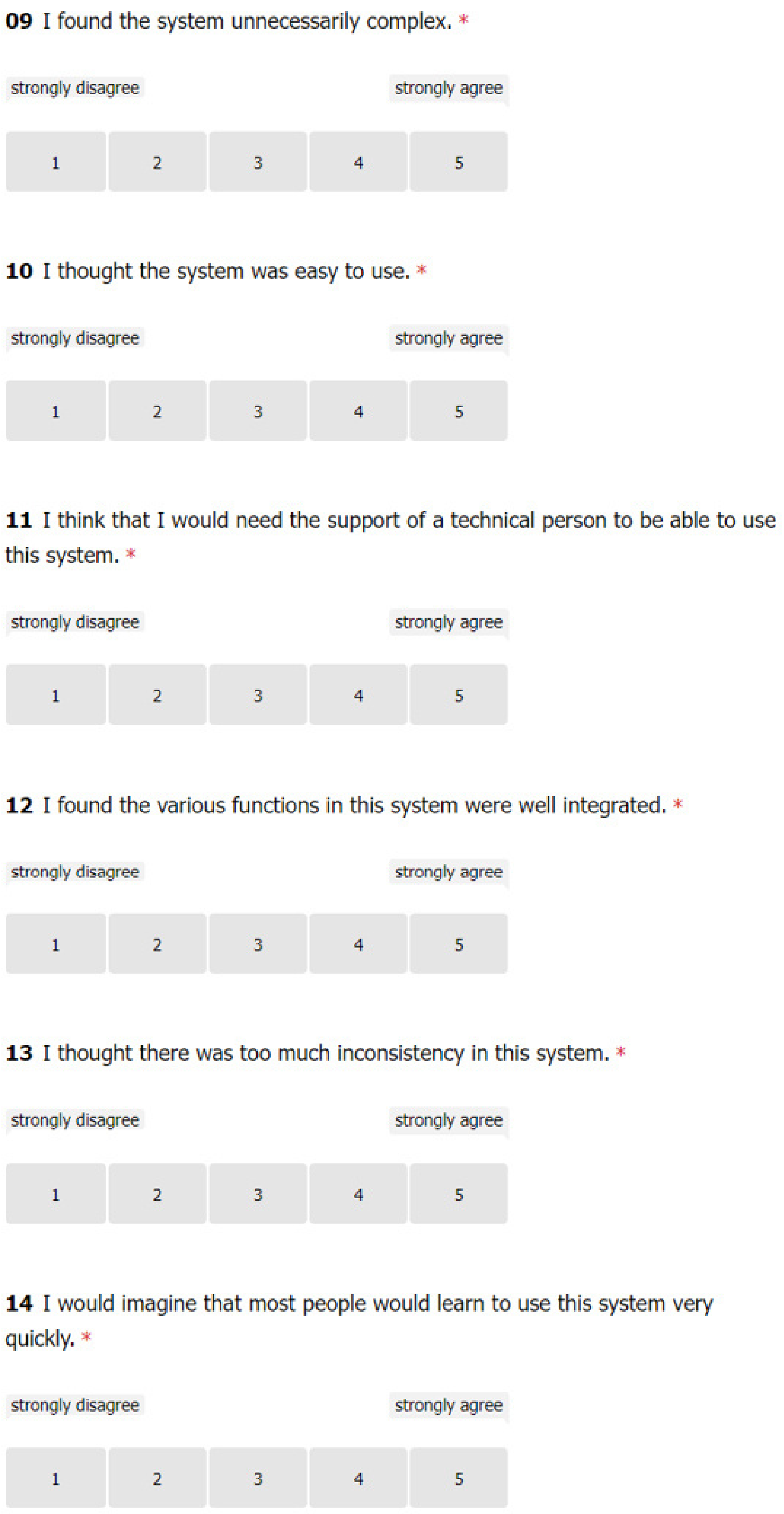
strongly disagree				strongly agree
1	2	3	4	5

07 I would like to recommend people to use this system? *

strongly disagree				strongly agree
1	2	3	4	5

08 If I have any needs in this area in the future, I think that I would like to use this system frequently. *

strongly disagree				strongly agree
1	2	3	4	5



09 I found the system unnecessarily complex. *

strongly disagree strongly agree

1 2 3 4 5

10 I thought the system was easy to use. *

strongly disagree strongly agree

1 2 3 4 5

11 I think that I would need the support of a technical person to be able to use this system. *

strongly disagree strongly agree

1 2 3 4 5

12 I found the various functions in this system were well integrated. *

strongly disagree strongly agree

1 2 3 4 5

13 I thought there was too much inconsistency in this system. *

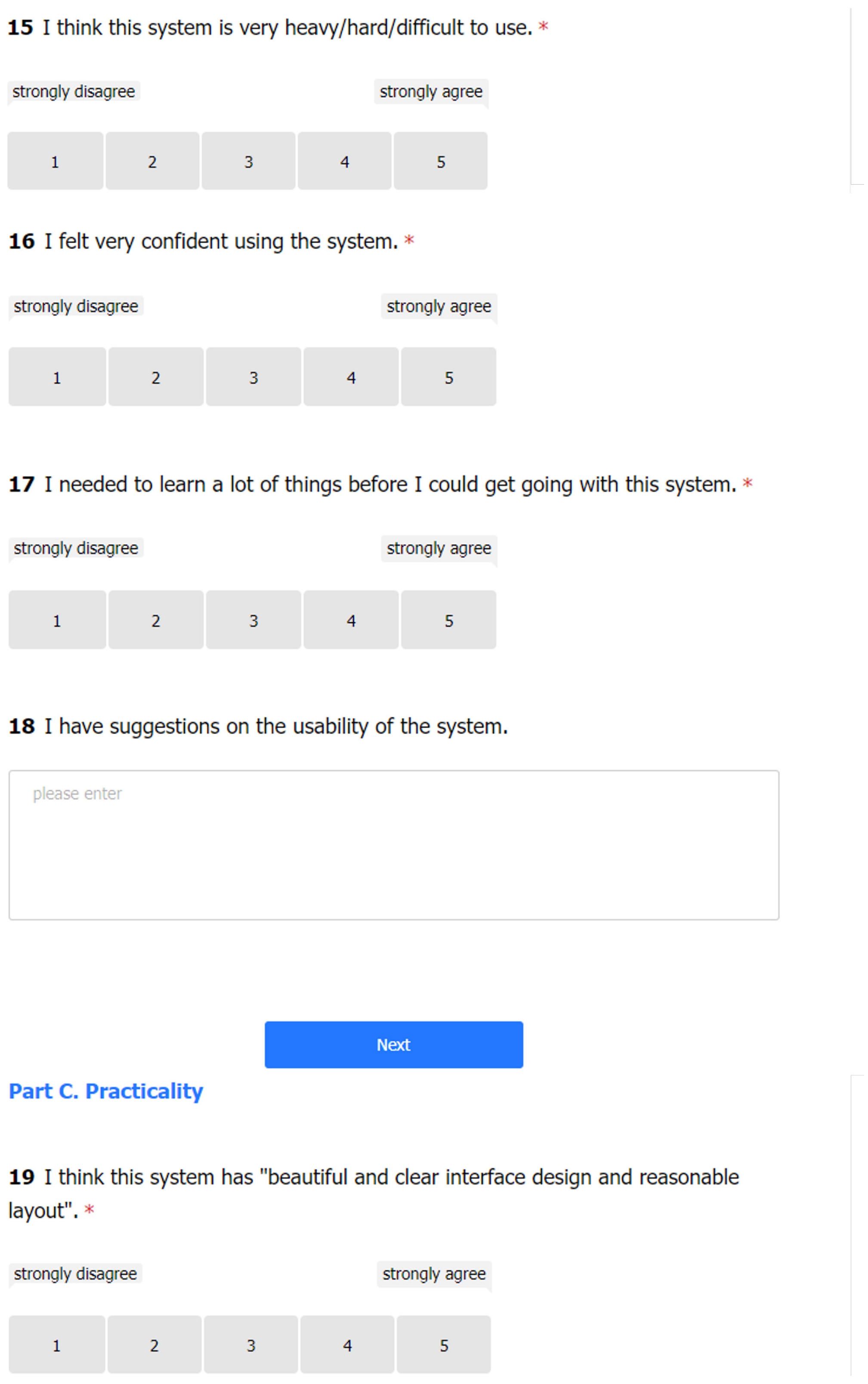
strongly disagree strongly agree

1 2 3 4 5

14 I would imagine that most people would learn to use this system very quickly. *

strongly disagree strongly agree

1 2 3 4 5



15 I think this system is very heavy/hard/difficult to use. *

strongly disagree — strongly agree

1	2	3	4	5

16 I felt very confident using the system. *

strongly disagree — strongly agree

1	2	3	4	5

17 I needed to learn a lot of things before I could get going with this system. *

strongly disagree — strongly agree

1	2	3	4	5

18 I have suggestions on the usability of the system.

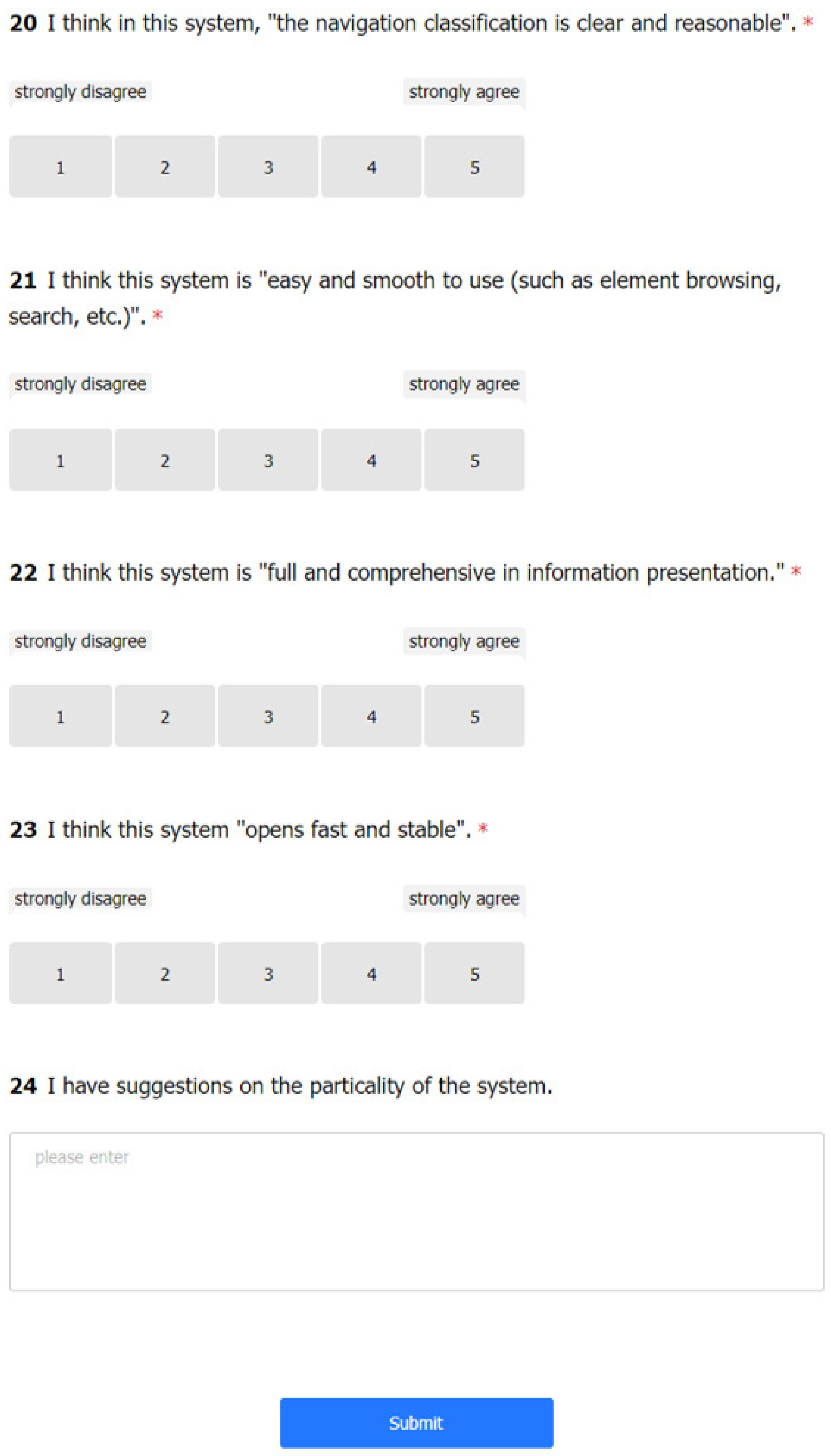
please enter

Part C. Practicality

19 I think this system has "beautiful and clear interface design and reasonable layout". *

strongly disagree — strongly agree

1	2	3	4	5



20 I think in this system, "the navigation classification is clear and reasonable". *

strongly disagree strongly agree

1	2	3	4	5

21 I think this system is "easy and smooth to use (such as element browsing, search, etc.)". *

strongly disagree strongly agree

1	2	3	4	5

22 I think this system is "full and comprehensive in information presentation." *

strongly disagree strongly agree

1	2	3	4	5

23 I think this system "opens fast and stable". *

strongly disagree strongly agree

1	2	3	4	5

24 I have suggestions on the particality of the system.

please enter

Submit

Bibliography

(2016). Fair principles. https://www.go-fair.org/fair-principles/.

Al-Tashi, Q., Abdulkadir, S. J., Rais, H. M., Mirjalili, S., and Alhussian, H. (2020). Approaches to multi-objective feature selection: A systematic literature review. *IEEE Access*, 8:125076–125096.

Alasadi, S. A. and Bhaya, W. S. (2017). Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, 12:4102–4107.

Alexandropoulos, S.-A. N., Kotsiantis, S. B., and Vrahatis, M. N. (2019/ed). Data preprocessing in predictive data mining. *The Knowledge Engineering Review*, 34.

Alserafi, A., Abelló, A., Romero, O., and Calders, T. (2016). Towards information profiling: Data lake content metadata management. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 178–185.

Amazon (2016). Data lake — microsoft azure.

Arthur, C. and editor, t. (2013). Tech giants may be huge, but nothing matches big data. *The Guardian*.

Azure (2016). What is a data lake?

Bangor, A., Kortum, P. T., and Miller, J. T. (2008). An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, 24(6):574–594.

Bansal, S. K. (2014). Towards a semantic extract-transform-load (etl) framework for big data integration. In *2014 IEEE International Congress on Big Data*, pages 522–529.

Bansal, S. K. and Kagemann, S. (2015). Integrating big data: A semantic extract-transform-load framework. *Computer*, 48(3):42–50.

Berti-Equille, L. (2019). Learn2clean: Optimizing the sequence of tasks for web data preparation. In *The World Wide Web Conference*, WWW '19, pages 2580–2586, New York, NY, USA. Association for Computing Machinery.

Bertino, E. and Ferrari, E. (2018). Big data security and privacy. In Flesca, S., Greco, S., Masciari, E., and Saccà, D., editors, *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, Studies in Big Data, pages 425–439. Springer International Publishing, Cham.

Bevan, N., Carter, J., and Harker, S. (2015). Iso 9241-11 revised: What have we learnt about usability since 1998? In Kurosu, M., editor, *Human-Computer Interaction: Design and Evaluation*, Lecture Notes in Computer Science, pages 143–151, Cham. Springer International Publishing.

Bilalli, B., Abelló, A., Aluja-Banet, T., and Wrembel, R. (2016). Towards intelligent data analysis: The metadata challenge:. In *Proceedings of the International Conference on Internet of Things and Big Data*, pages 331–338, Rome, Italy. SCITEPRESS - Science and and Technology Publications.

Brooke, J. (1995). Sus: A quick and dirty usability scale. *Usability Eval. Ind.*, 189.

Brownlee, J. (2020). *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery.

Cai, L. and Zhu, Y. (2015). The challenges of data quality and data quality assessment in the big data era. *Data Science Journal*, 14(0):2.

Cavanillas, J. M., Curry, E., and Wahlster, W. (2016). The big data value opportunity. In Cavanillas, J. M., Curry, E., and Wahlster, W., editors, *New Horizons for a Data-Driven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe*, pages 3–11. Springer International Publishing, Cham.

Coussement, K., Lessmann, S., and Verstraeten, G. (2017). A comparative analysis of data preparation algorithms for customer churn prediction: A case study in the telecommunication industry. *Decision Support Systems*, 95:27–36.

Couto, J., Borges, O., Ruiz, D. D., Marczak, S., and Prikladnicki, R. (2019). A mapping study about data lakes: An improved definition and possible architectures. In *The 31st International Conference on Software Engineering and Knowledge Engineering*, pages 453–458.

Covert, Q., Steinhagen, D., Francis, M., and Streff, K. (2020). Towards a triad for data privacy.

Delfosse, V., Billen, R., and Leclercq, P. (2012). Uml as a schema candidate for graph databases. *NoSql Matters 2012*.

Demchenko, Y., Grosso, P., de Laat, C., and Membrey, P. (2013). Addressing big data issues in scientific data infrastructure. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, pages 48–55, San Diego, CA, USA. IEEE.

Deng, K., Li, X., Lu, J., and Zhou, X. (2015). Best keyword cover search. *IEEE Transactions on Knowledge and Data Engineering*, 27(1):61–73.

Derakhshannia, M., Gervet, C., Hajj-Hassan, H., Laurent, A., and Martin, A. (2019). Life and death of data in data lakes: Preserving data usability and responsible governance. In El Yacoubi, S., Bagnoli, F., and Pacini, G., editors, *Internet Science*, Lecture Notes in Computer Science, pages 302–309, Cham. Springer International Publishing.

Derakhshannia, M., Gervet, C., Hajj-Hassan, H., Laurent, A., and Martin, A. (2020). Data lake governance: Towards a systemic and natural ecosystem analogy. *Future Internet*, 12(8):126.

Diamantini, C., Giudice, P. L., Musarella, L., Potena, D., Storti, E., and Ursino, D. (2018). A new metadata model to uniformly handle heterogeneous data lake sources. In Benczúr, A., Thalheim, B., Horváth, T., Chiusano, S., Cerquitelli, T., Sidló, C., and Revesz, P. Z., editors, *New Trends in Databases and Information Systems*, Communications in Computer and Information Science, pages 165–177, Cham. Springer International Publishing.

Dixon, J. (2010). Pentaho, hadoop, and data lakes. https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/.

Dorleon, G., Bricon-Souf, N., Megdiche, I., and Teste, O. (2021). Qualification du biais de données dans le processus de la science des données. *Revue des Nouvelles Technologies de l'Information*, Extraction et Gestion des Connaissances, RNTI-E-37:515–516.

Eichler, R., Giebler, C., Gröger, C., Schwarz, H., and Mitschang, B. (2020). Handle - a generic metadata model for data lakes. In Song, M., Song, I.-Y., Kotsis, G., Tjoa, A. M., and Khalil, I., editors, *Big Data Analytics and Knowledge Discovery*, Lecture Notes in Computer Science, pages 73–88, Cham. Springer International Publishing.

El Akkaoui, Z. and Zimanyi, E. (2009). Defining etl worfklows using bpmn and bpel. In *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*, DOLAP '09, pages 41–48, New York, NY, USA. Association for Computing Machinery.

Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., and Vanschoren, J. (2016). Ml schema core specification. http://htmlpreview.github.io/?https://github.com/ML-Schema/documentation/blob/gh-pages/ML%20Schema.html.

Fang, H. (2015). Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem. In *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 820–824.

Farid, M., Roatis, A., Ilyas, I. F., Hoffmann, H.-F., and Chu, X. (2016). Clams: Bringing quality to data lakes. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2089–2092, New York, NY, USA. Association for Computing Machinery.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2755–2763, Cambridge, MA, USA. MIT Press.

García, S., Luengo, J., and Herrera, F. (2015). *Data Preprocessing in Data Mining*, volume 72 of *Intelligent Systems Reference Library*. Springer International Publishing, Cham.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., and Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1):9.

Giebler, C., Gröger, C., Hoos, E., Eichler, R., Schwarz, H., and Mitschang, B. (2021). *The Data Lake Architecture Framework*. Gesellschaft für Informatik, Bonn.

Gómez, P., Roncancio, C., and Casallas, R. (2018). Towards quality analysis for document oriented bases. In Trujillo, J. C., Davis, K. C., Du, X., Li, Z., Ling, T. W., Li, G., and Lee, M. L., editors, *Conceptual Modeling*, Lecture Notes in Computer Science, pages 200–216, Cham. Springer International Publishing.

Gorelik, A. (2020). *What Is a Data Lake?* O'Reilly Media, Inc., S.l.

Gupta, S. and Giri, V. (2018). *Practical Enterprise Data Lake Insights*. Apress, Berkeley, CA.

Gyulgyulyan, E., Aligon, J., Ravat, F., and Astsatryan, H. (2019). Data quality alerting model for big data analytics. In Welzer, T., Eder, J., Podgorelec, V., Wrembel, R., Ivanović, M., Gamper, J., Morzy, M., Tzouramanis, T., Darmont, J., and Kamišalić Latifić, A., editors, *New Trends in Databases and Information Systems*, Communications in Computer and Information Science, pages 489–500, Cham. Springer International Publishing.

Hai, R., Geisler, S., and Quix, C. (2016). Constance: An intelligent data lake system. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2097–2100, New York, NY, USA. Association for Computing Machinery.

Halevy, A., Korn, F., Noy, N. F., Olston, C., Polyzotis, N., Roy, S., and Whang, S. E. (2016a). Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 795–806, New York, NY, USA. Association for Computing Machinery.

Halevy, A., Korn, F., Noy, N. F., Olston, C., Polyzotis, N., Roy, S., and Whang, S. E. (2016b). Managing google's data lake: an overview of the goods system. {*IEEE*} *Data Eng. Bull.*, 39(3):5–14.

Hidalgo, M., Menasalvas, E., and Eibe, S. (2009). Definition of a metadata schema for describing data preparation tasks. pages 64–75.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Automated Machine Learning: Methods, Systems, Challenges.* The Springer Series on Challenges in Machine Learning. Springer International Publishing.

Iliou, T., Konstantopoulou, G., Stephanakis, I., Anastasopoulos, K., Lymberopoulos, D., and Anastassopoulos, G. (2018). Iliou machine learning data preprocessing method for stress level prediction. In Iliadis, L., Maglogiannis, I., and Plagianakos, V., editors, *Artificial Intelligence Applications and Innovations*, IFIP Advances in Information and Communication Technology, pages 351–361, Cham. Springer International Publishing.

Inmon, B. (2016). *Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump.* Technics Publications.

ISO (2018). Iso 9241-11:2018(en), ergonomics of human-system interaction — part 11: Usability: Definitions and concepts.

Jin, Z., Anderson, M. R., Cafarella, M., and Jagadish, H. V. (2017). Foofah: Transforming data by example. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 683–698, New York, NY, USA. Association for Computing Machinery.

John, T. and Misra, P. (2017). *Data lake for enterprises: leveraging Lambda architecture for building enterprise data lake.* Packt, Birmingham, UK.

Judd, C. M., McClelland, G. H., and Ryan, C. S. (2011). *Data Analysis: A Model Comparison Approach, Second Edition.* Routledge.

Keet, C. M., Ławrynowicz, A., d'Amato, C., Kalousis, A., Nguyen, P., Palma, R., Stevens, R., and Hilario, M. (2015). The data mining optimization ontology. *Journal of Web Semantics*, 32:43–53.

Kredpattanakul, K. and Limpiyakorn, Y. (2019). Transforming javascript-based web application to cross-platform desktop with electron. In Kim, K. J. and Baek, N., editors, *Information Science and Applications 2018*, Lecture Notes in Electrical Engineering, pages 571–579, Singapore. Springer.

Kwon, O., Lee, N., and Shin, B. (2014). Data quality management, data usage experience and acquisition intention of big data analytics. *International Journal of Information Management*, 34(3):387–394.

Lal, M. (2015). *Neo4j Graph Data Modeling.* Packt Publishing.

LaPlante, A. (2016). *Architecting data lakes: data management architectures for advanced business use cases.* O'Reilly Media, Sebastopol, CA, first edition. edition.

Lenzerini, M. (2002). Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 233–246, New York, NY, USA. Association for Computing Machinery.

Lewis, J. R. (2018). The system usability scale: Past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590.

Lewis, J. R. and Sauro, J. (2018). Item benchmarks for the system usability scale. *Journal of Usability Studies*, 13(3).

Liu, P., Loudcher, S., Darmont, J., and Noûs, C. (2021). Archaeodal: A data lake for archaeological data management and analytics. *25th International Database Engineering & Applications Symposium*, pages 252–262.

Maccioni, A. and Torlone, R. (2018). Kayak: A framework for just-in-time data preparation in a data lake. In Krogstie, J. and Reijers, H. A., editors, *Advanced Information Systems Engineering*, Lecture Notes in Computer Science, pages 474–489, Cham. Springer International Publishing.

Madera, C. (2018). *L'évolution des systèmes et architectures d'information sous l'influence des données massives : les lacs de données.* Theses, Université Montpellier.

Madera, C. and Laurent, A. (2016). The next information architecture evolution: the data lake wave. In *Proceedings of the 8th International Conference on Management of Digital EcoSystems*, MEDES, pages 174–180, New York, NY, USA. Association for Computing Machinery.

Meehan, J., Aslantas, C., Zdonik, S., Tatbul, N., and Du, J. (2017). Data ingestion for the connected world. In *CIDR*.

Megdiche, I., Ravat, F., and Zhao, Y. (2021). Metadata management on data processing in data lakes. In Bureš, T., Dondi, R., Gamper, J., Guerrini, G., Jurdziński, T., Pahl, C., Sikora, F., and Wong, P. W., editors, *SOFSEM 2021: Theory and Practice of Computer Science*, Lecture Notes in Computer Science, pages 553–562, Cham. Springer International Publishing.

Mehmood, H., Gilman, E., Cortes, M., Kostakos, P., Byrne, A., Valta, K., Tekes, S., and Riekki, J. (2019). Implementing big data lake for heterogeneous data sources. In *2019 IEEE 35th International Conference on Data Engineering Workshops (ICDEW)*, pages 37–44.

Miloslavskaya, N. and Tolstoy, A. (2016). Big data, fast data and data lake concepts. *Procedia Computer Science*, 88:300–305.

Munshi, A. A. and Mohamed, Y. A.-R. I. (2018). Data lake lambda architecture for smart grids big data analytics. *IEEE Access*, 6:40463–40471.

Nadipalli, R. (2017). *Effective business intelligence with QuickSight: from data to actionable business insights using Amazon QuickSight!* Packt Publishing, Birmingham, UK.

NISO (2017). *Understanding metadata: what is metadata, and what is it for?*

O'Leary, D. E. (2014). Embedding ai and crowdsourcing in the big data lake. *IEEE Intelligent Systems*, 29(5):70–73.

Oliveira, B. and Belo, O. (2012). Bpmn patterns for etl conceptual modelling and validation. In Chen, L., Felfernig, A., Liu, J., and Raś, Z. W., editors, *Foundations of Intelligent Systems*, Lecture Notes in Computer Science, pages 445–454, Berlin, Heidelberg. Springer.

O'Neil, C. and Schutt, R. (2013). *Doing Data Science: Straight Talk from the Frontline.* "O'Reilly Media, Inc.".

Palmer, M. (2006). Data is the new oil.

Panov, P., Soldatova, L., and Džeroski, S. (2014). Ontology of core data mining entities. *Data Mining and Knowledge Discovery*, 28(5):1222–1265.

Panwar, A. and Bhatnagar, V. (2020). Data lake architecture: A new repository for data engineer. *International Journal of Organizational and Collective Intelligence (IJOCI)*, 10(1):63–75.

Paschalidi, C. (2015). *Data Governance : A conceptual framework in order to prevent your Data Lake from becoming a Data Swamp.*

Peguero, K. and Cheng, X. (2021). Electrolint and security of electron applications. *High-Confidence Computing*, 1(2):100032.

Quix, C., Hai, R., and Vatov, I. (2016). Metadata extraction and management in data lakes with gemms. *Complex Systems Informatics and Modeling Quarterly*, 0(9):67–83.

Ravat, F. and Zhao, Y. (2019a). Data lakes: Trends and perspectives. In Hartmann, S., Küng, J., Chakravarthy, S., Anderst-Kotsis, G., Tjoa, A. M., and Khalil, I., editors, *Database and Expert Systems Applications*, Lecture Notes in Computer Science, pages 304–313, Cham. Springer International Publishing.

Ravat, F. and Zhao, Y. (2019b). Metadata management for data lakes. In Welzer, T., Eder, J., Podgorelec, V., Wrembel, R., Ivanović, M., Gamper, J., Morzy, M., Tzouramanis, T., Darmont, J., and Kamišalić Latifić, A., editors, *New Trends in Databases and Information Systems*, Communications in Computer and Information Science, pages 37–44, Cham. Springer International Publishing.

Raynaut, W., Soule-Dupuy, C., and Valles-Parlangeau, N. (2016). Meta-mining evaluation framework: A large scale proof of concept on meta-learning. In Kang, B. H. and Bai, Q., editors, *AI 2016: Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 215–228, Cham. Springer International Publishing.

Reinsel, D., Gantz, J., and Rydning, J. (2017). Data age 2025: The evolution of data to life-critical. *Don't Focus on Big Data*, 2.

Rezig, E. K., Cao, L., Stonebraker, M., Simonini, G., Tao, W., Madden, S., Ouzzani, M., Tang, N., and Elmagarmid, A. K. (2019). Data civilizer 2.0: a holistic framework for data preparation and analytics. *Proceedings of the VLDB Endowment*, 12(12):1954–1957.

Rivolli, A., Garcia, L. P. F., Soares, C., Vanschoren, J., and Carvalho, A. (2018). Towards reproducible empirical research in meta-learning. *ArXiv*.

Sawadogo, P. and Darmont, J. (2021). On data lake architectures and metadata management. *Journal of Intelligent Information Systems*, 56(1):97–120.

Sawadogo, P. N., Scholly, É., Favre, C., Ferey, É., Loudcher, S., and Darmont, J. (2019). Metadata systems for data lakes: Models and features. In Welzer, T., Eder, J., Podgorelec, V., Wrembel, R., Ivanović, M., Gamper, J., Morzy, M., Tzouramanis, T., Darmont, J., and Kamišalić Latifić, A., editors, *New Trends in Databases and Information Systems*, Communications in Computer and Information Science, pages 440–451, Cham. Springer International Publishing.

Scholly, E., Sawadogo, P. N., Liu, P., Alfonso Espinosa-Oviedo, J., Favre, C., Loudcher, S., Darmont, J., and Noûs, C. (2021). Coining goldmedal: A new contribution to data lake generic metadata modeling. In *23rd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP@EDBT/ICDT 2021)*, volume 2840 of *CEUR*, pages 31–40, Nicosia, Cyprus.

Shimazu, K., Arisawa, T., and Saito, I. (2006). Interdisciplinary contents management using 5w1h interface for metadata. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pages 909–912.

Simitsis, A., Vassiliadis, P., Dayal, U., Karagiannis, A., and Tziovara, V. (2009). Benchmarking etl workflows. In Nambiar, R. and Poess, M., editors, *Performance Evaluation and Benchmarking*, Lecture Notes in Computer Science, pages 199–220, Berlin, Heidelberg. Springer.

Solorio-Fernández, S., Carrasco-Ochoa, J. A., and Martínez-Trinidad, J. F. (2020). A review of unsupervised feature selection methods. *Artificial Intelligence Review*, 53(2):907–948.

Sulyman, S. (2014). Client-server model. *IOSR Journal of Computer Engineering*, 16:57–71.

Suriarachchi, I. and Plale, B. (2016a). Crossing analytics systems: A case for integrated provenance in data lakes. In *2016 IEEE 12th International Conference on e-Science (e-Science)*, pages 349–354.

Suriarachchi, I. and Plale, B. (2016b). Provenance as essential infrastructure for data lakes. In Mattoso, M. and Glavic, B., editors, *Provenance and Annotation of Data and Processes*, Lecture Notes in Computer Science, pages 178–182, Cham. Springer International Publishing.

Syed, A. (2020). The challenge of building effective, enterprise-scale data lakes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 803, New York, NY, USA. Association for Computing Machinery.

Terrizzano, I., Schwarz, P., Roth, M., and Colino, J. E. (2015). Data wrangling: The challenging yourney from the wild to the lake. In *CIDR*.

van Vlymen, J. and de Lusignan, S. (2005). A system of metadata to control the process of query, aggregating, cleaning and analysing large datasets of primary care data. *Informatics in Primary Care*, 13(4):281–291.

Vassiliadis, P. (2009). A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(3):1–27.

Walker, C. and Alrehamy, H. (2015). Personal data lake with data gravity pull. In *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, pages 160–167.

whatls (2021). What is metadata management? - definition from whatis.com.

Wirth, R. and Hipp, J. (2000). Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, volume 1. Springer-Verlag London, UK.

Zgolli, A., Collet, C., and Madera, C. (2020). Metadata in data lake ecosystems. In *Data Lakes*, chapter 4, pages 57–96. John Wiley & Sons, Ltd.

Zhang, D. (2018). Big data security and privacy protection. In *8th International Conference on Management and Computer Science (ICMCS 2018)*, pages 275–278. Atlantis Press.

Zhang, Y. and Ives, Z. G. (2020). Finding related tables in data lakes for interactive data science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pages 1951–1966, New York, NY, USA. Association for Computing Machinery.

Zhao, Y., Megdiche, I., and Ravat, F. (2021a). Data lake ingestion management. *arXiv:2107.02885 [cs]*.

Zhao, Y., Megdiche, I., Ravat, F., and Dang, V.-n. (2021b). A zone-based data lake architecture for iot, small and big data. In *25th International Database Engineering & Applications Symposium*, IDEAS 2021, pages 94–102, New York, NY, USA. Association for Computing Machinery.

Zhao, Y., Ravat, F., Aligon, J., Soule-dupuy, C., Ferrettini, G., and Megdiche, I. (2021c). Analysis-oriented metadata for data lakes. In *25th International Database Engineering & Applications Symposium*, IDEAS 2021, pages 194–203, New York, NY, USA. Association for Computing Machinery.

Zhou, L., Pan, S., Wang, J., and Vasilakos, A. (2017). Machine learning on big data: Opportunities and challenges. *Neurocomputing*.