Data Lakes: A Survey of Functions and Systems

Rihan Hai, *Member, IEEE*, Christos Koutras, Christoph Quix, and Matthias Jarke, *Senior Member, IEEE*

(Survey Paper)

***Abstract*—Data lakes are becoming increasingly prevalent for Big Data management and data analytics. In contrast to traditional 'schema-on-write' approaches such as data warehouses, data lakes are repositories storing raw data in its original formats and providing a common access interface. Despite the strong interest raised from both academia and industry, there is a large body of ambiguity regarding the definition, functions and available technologies for data lakes. A complete, coherent picture of data lake challenges and solutions is still missing. This survey reviews the development, architectures, and systems of data lakes. We provide a comprehensive overview of research questions for designing and building data lakes. We classify the existing approaches and systems based on their provided functions for data lakes, which makes this survey a useful technical reference for designing, implementing and deploying data lakes. We hope that the thorough comparison of existing solutions and the discussion of open research challenges in this survey will motivate the future development of data lake research and practice.**

***Index Terms*—Data discovery, data lake, metadata management.**

I. Introduction

BIG data has undoubtedly become one of the most important challenges in database research. Unprecedented volume, large variety, and high velocity of data impede their collection, storage, and processing; especially the variety of data still poses a daunting challenge with many open issues [2]. Web-based business transactions, sensor networks, real-time streaming, social media, and scientific research generate a large amount of (semi-)structured and unstructured data, often stored in separate information silos. Combining and integrating the information across these silos is critical for reaching valuable insights.

Traditional *schema-on-write* approaches, like the extract, transform, load (ETL) process of data warehouses [77], are inefficient for such data management requirements. This has drawn the interest of many developers and researchers to NoSQL data management systems. NoSQL systems provide data management features tailored to high amounts of schema-less data, which enables a *schema-on-read* manner of data handling, i.e., the structure of data is not required for storing but only when further analyzing and processing the data. Open-source platforms, such as Hadoop [132] with higher-level languages (e.g., Pig and Hive), and NoSQL databases (e.g., MongoDB and Neo4j), have gained popularity. Although the current market share is still dominated by relational database systems, a one-size-fits-all Big Data system is unlikely to solve all the challenges related to data management today.

To address this gap, *data lakes* (DLs) have been proposed. In essence, a data lake is a flexible, scalable data storage and management system, which ingests and stores raw data from heterogeneous sources in their original format, and provides maintenance, query processing and data analytics in an on-the-fly manner, with the help of rich metadata [116], [138], [142], [143]. Data lakes are proposed to store and manage data in many real-life use cases: Internet of things (IoT) and smart city [99], manufacturing [112], medicine [42], [55], [114], mobility service (e.g., Uber) [50], biology [23], smart grids [20], [103], air quality control [145], flights data [96], disease control, labor markets and products [13].

A. Survey Goal and Related Work

In the past decade, various solutions and systems have been proposed to address the research challenges of data lakes. However, while 'data lake' is a current buzzword with a lot of hype surrounding it, there is a lot of ambiguity about its exact definition and functions. Moreover, most recent data lake proposals only target a specific research problem or certain types of source data. A coherent, complete picture of data lake problems and solutions is still missing.

In this survey, we provide a thorough explanation of the data lake concept, its development, more importantly, a categorization and review of existing data lake solutions. The survey also aims at helping researchers and developers to build or customize a data lake, and discover open questions and future research directions about data lakes. Earlier efforts in structuring the data lake field only provide a limited view of a subset of research problems regarding data lakes. Moreover, none of these works touch on the details of future data lake challenges such as supporting machine learning in data lakes.

Several earlier works [78], [98], [115] propose possible research topics that should be studied with respect to data lakes without reviewing existing data lake systems. These works are orthogonal to our goals in this survey. Instead of merely listing

Manuscript received 12 May 2022; revised 19 January 2023; accepted 8 April 2023. Date of publication 25 April 2023; date of current version 8 November 2023. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612. This work was supported by the European Union Horizon Programme call HORIZON-CL4-2022-DATA-01, under Grant 101093164 (ExtremeXP). Recommended for acceptance by L. Zou. *(Corresponding author: Rihan Hai.)*

Rihan Hai and Christos Koutras are with the Department of Software Technology, Delft University of Technology, 2628, CD Delft, Netherlands (e-mail: r.hai@tudelft.nl; c.koutras@tudelft.nl).

Christoph Quix is with the Hochschule Niederrhein, Krefeld, Germany and Fraunhofer FIT, Hochschule Niederrhein University of Applied Sciences, 47805 Krefeld, Germany (e-mail: christoph.quix@fit.fraunhofer.de).

Matthias Jarke is with the RWTH Aachen University, 52062 Aachen, Germany, and also with Fraunhofer FIT, 53757 Sankt Augustin, Germany (e-mail: jarke@dbis.rwth-aachen.de).

Digital Object Identifier 10.1109/TKDE.2023.3270101

© 2023 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License. For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

TABLE I
CLASSIFICATION OF DATA LAKE SOLUTIONS BASED ON FUNCTIONS

Tier	Functions	Systems
Ingestion	Metadata extraction	GEMMS [117]
		DATAMARAN [53]
		Skluma [137]
	Metadata modeling	GEMMS [64], [117]
		HANDLE [43]
		Data vault [57], [107]
		Diamantiniet al. [34], [35], [36]
		Aurum [48]
		Sawadogoet et al. [127]
Maintenance	Dataset organization	GOODS [67], [68]
		DS-Prox [3], [4], [5]
		KAYAK [90], [91]
		Nargesian et al. [104]
		Ronin [110]
		Juneau [152]
	Related dataset discovery	Aurum [48]
		Brackenbury et.al. [15]
		JOSIE [155]
		D^3L [14]
		Juneau [75], [151], [152]
		PEXESO [40]
		RNLIM [121]
		DLN [12]
	Data integration	Constance [61], [62], [63], [65]
		ALITE [82]
	Metadata enrichment	CoreDB [9], [10]
		D^4 [109]
		DomainNet [85]
		Constance [64]
		GOODS [67], [68]
	Data cleaning	CLAMS [47]
		Constance [64]
		Song et al. [138]
	Schema evolution	Klettkeet et al. [83]
	Data provenance	IBM tool [143]
		Suriarachchi et al. [141]
		GOODS [67], [68]
		CoreDB [9], [10]
		Juneau [75], [151], [152]
Exploration	Query-driven data discovery	JOSIE [155]
		D^3L [14]
		Juneau [75], [151], [152]
		Aurum [48]
	Heterogeneous data querying	Constance [61], [65]
		CoreDB [9], [10]
		Ontario [44], [80]
		Squerall [94]

potential research questions, our focus is to make a technical comparison of the existing systems for data lakes.

In a recent tutorial, Nargesian et al. [105] cover seven functions of data lakes. They have briefly discussed existing data lake solutions, together with technologies and systems potentially useful for data lakes[1]. In contrast to this tutorial, our survey provides a more holistic introduction to data lakes and discusses the required functions of data lakes in more detail (cf. Fig. 2 and Table I for our functional view of data lakes).

Couto et al. [29] compare different data lake definitions and list common open-source tools used in data lake architectures. In [147], Zagan et al. review the architectures of seven specific data lake systems. In a data lake architecture proposal [122], Ravat and Zhao propose classification criteria for metadata categories and data governance in data lakes. Giebler et al. [58] discuss some additional aspects of data lake architecture, data storage, data modeling, metadata management and data governance. Each of these works only provides a partial list of data lake functions. In this survey, we give a more comprehensive view of the current data lake landscape and have a more in-depth discussion regarding research challenges and solutions of data lakes.

In [126] Sawadogo et al. compare different data lake definitions, architectures, metadata types, metadata models, and metadata management components (e.g., semantic enrichment). They provide a high-level guide for conceptual design of data lakes. However, their discussion regarding functions to implement for a data lake system is very brief and limited to summarizing the open-source technologies and tools used in existing data lakes, e.g., Apache Spark, Drill, and Pig. A similar survey [25] also only focuses on the aspects of data lake architecture, metadata management, and open-source technologies. In this survey, we

[1]https://rjmillerlab.github.io/data-lake-tutorial-slides/

cover a wider range of topics on data lakes *beyond* architecture and metadata management. We also show how to navigate from a conceptual architecture to system functions. We propose a more fine-grained categorization of existing systems for data lakes and provide a more detailed comparison of systems in each category. For each data lake function, we also cover the state-of-the-art systems not mentioned in [25], [126].

B. Contributions and Outline

Our main contributions are summarized as follows:

- We review the more than ten-year development of the data lake concept and implementations, and discuss future directions.
- We clarify the workflow and functions for building a data lake through a fine-grained architecture.
- We provide a three-level classification of existing studies about data lakes according to their provided functions. We analyze each class of research problems in depth and compare the existing data lake approaches.

Scope of the Survey. In this survey, we focus on systems that explicitly claim to be a data lake (e.g., personal data lake [143]), or provide partial functions of a data lake (e.g., the data discovery system Aurum [48]). It is beyond the scope of **one** survey article to list all *possible* research topics and all *potential* solutions for data lake systems. Some research topics mentioned in this survey have been intensively studied in the database community, e.g., data integration, data cleaning, and data discovery. There are dedicated surveys on these topics, while in this survey we only introduce and compare systems tackling these problems within a data lake. For these topics, we will explain the lake-specific problem settings. For instance, consider the data integration problem (Section VI-C), which is to resolve heterogeneous schemata or entity values. In a data lake, for data integration we often assume more levels of heterogeneity among the data sources. Finally, for system comparison, we mainly cover **implemented** systems that resolve **research** problems of data lakes, rather than high-level DL system proposals or commercial DL products. We made this choice because such high-level proposals often lack details for meaningful comparison, and do not always reflect the feasibility of actual system implementation. For a more industrial point of view, we point the reader to papers like [59], [73], [125].

Outline. As illustrated in Fig. 1, the survey has four parts. The first part covers the fundamentals of data lakes, including the introduction (Section I) and the origin and development (Section II). Then we discuss the common aspects that almost every data lake designer needs to consider: the system architecture (Section III) and data storage (Section IV). In particular, we introduce the criteria of classifying data lake solutions in Section III-B. In the third part, we categorize existing data lake solutions by their functional tiers: ingestion (Section V), maintenance (Section VI), and exploration (Section VII). Finally, we discuss research challenges and future directions in Section VIII and conclude the survey in Section IX.

How to Use This Survey. We organize the survey in the structure of Fig. 1, such that it is self-contained and presented in a natural flow. We first explain high-level concepts and architecture, before discussing data storage options and functions. A data lake expert interested in a particular research problem, can directly go to Sections V, VI, and VII. Discussions on challenging new directions are in Section VIII.

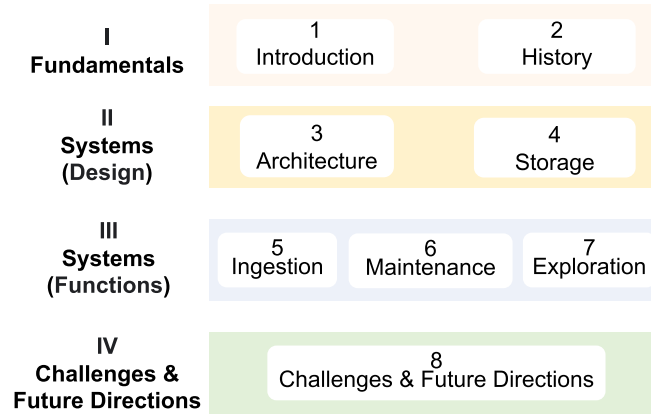

Fig. 1. Survey outline.

II. A Brief History of Data Lakes

As of this writing, the concept of data lakes is about a decade old and has significantly evolved in this period. We summarize this evolution in three stages.

A. 2010-2013: Beginnings

The concept of *data lake* was first coined in the industry. In 2010, it was first proposed by *Pentaho* CTO James Dixon, as a solution that handles raw data from **one** source and supports diverse user requirements [37]. This was seen in sharp contrast to data warehouses or data marts for which the structure and usage of the data must be predefined and fixed, while rigorous data extraction, transformation, and cleaning are necessary before entering data. By storing **raw** data in the original format, data lakes could avoid or delay this expensive standard preprocessing.

In 2013, *Pivot* proposed an architecture for a business data lake [19], which ingests **multiple** data sources in three abstract tiers: (1) an *ingestion tier* takes data in real-time/micro-batch/batch, (2) an *insight tier* analyzes data in real-time or interactive time and derives insights, and (3) an *action tier* that links insights with the existing applications; additional tiers *monitor and manage* the data. Pivot also suggested using Hadoop [132] as the storage system of a data lake, and applying its existing products to realize the previous tiers. However, not many details were given w.r.t. the actual implementation of a data lake.

B. 2014-2015: Criticisms and Further Development

In 2014, *Gartner* raised several criticisms about data lakes [54]. The main one was that ingesting disparate data might easily turn the data lake into an unusable "data swamp", unless there are metadata management and data governance. In particular, after ingestion, the semantics and data quality of the raw data are unknown, while the origin (provenance) of individual datasets and possible connections among them are missing. Indeed missing this information hinders user interaction with the data lake. In addition, Gartner pointed out that the existing data lake solutions did not provide a good answer on

how oversight of data security and privacy should be conducted. These crucial criticisms had a significant influence on many data lakes studies in the following years, which we discuss in Sections V and VI. In response, Dixon revisited the general concept [38] and emphasized that a data lake should also be equipped with metadata and governance, so that even with data in its raw form, a data lake could enable ad-hoc data analytics.

As more DL proposals started to emerge, they brought new requirements, solutions, and challenges. They significantly augmented the *possible* functions of a data lake, e.g., heterogeneous data, schema-on-read, metadata extraction/enrichment/management, applied Artificial Intelligence, and Crowdsourcing. *PwC* defined a data lake as a repository of structured, semi-structured, and unstructured data in heterogeneous formats [138], originating from the business transactions, sensors, or mobile/cloud-based applications. With Hadoop in the center, a new requirement is that a data lake should provide a low-cost data storage that is easy to access, yet in a **schema-on-read** manner, i.e., the data and metadata (e.g., semantics) can grow over time. The postulation is that a data lake actively extracts metadata from the raw data and stores it; then, it discovers patterns in the raw data. Moreover, users provide additional descriptive information of datasets (e.g., semantic annotations, domain-specific knowledge, and attribute linkages). The dynamic interaction between the data lake and users should thus continuously improve the quality and value of data.

Other proposals addressed new possibilities such as Artificial Intelligence (AI) and Crowdsourcing to facilitate data integration, access, and quality improvement in data lakes [108]. For example, AI helps with extracting features of data, generating tags with descriptive metadata, finding related datasets, discovering possible structures from schema-less data, and avoiding data redundancy. Crowdsourcing can help with collectively tagging semantic knowledge about the data, and linking possible relationships among datasets.

With a special focus on security information and event management, In [97] Marty discussed how to properly store and access the data. The importance of *metadata management* is emphasized in [143] with an architecture to parse, store, and query diversely structured personal data. Another proposal [46] emphasized the importance of Human-in-the-loop, e.g., data scientists govern the data in data lakes.

C. 2016-present: Prosperity and Diversity

Since 2016 the realization of data lakes in industry and research has been booming. There are proposals about data lake architectures [74], [92], [131], [134], concept, components and challenges [98], [101], [115].

Many IT companies offer commercial tools for building data lakes, e.g., *GOODS* [68] from Google, *Azure Data Lake* [120] from Microsoft, *AWS Lake*[2] from AMAZON, *Vora* from SAP [130], IBM and Cloudera[3], Oracle[4], and *Snowflake*[5]. *Delta Lake*[6] from Databricks is an open-source project that offers a storage tier compatible with Spark[7] APIs.

[2]https://aws.amazon.com/lake-formation/
[3]https://www.ibm.com/analytics/data-lake
[4]https://www.oracle.com/data-lakehouse/
[5]https://www.snowflake.com/data-lake/
[6]https://delta.io/
[7]https://spark.apache.org/

Meanwhile, data lake related research problems are raising massive attention associated with the implementation of DL prototypes. A large range of challenges are discussed, such as metadata management [61], data quality [47], data provenance [140], metadata enrichment [9], [64], dataset organization [5], [154], modeling [57], [107], [117], data integration [62], [65] and related dataset discovery [14], [15], [48], [151], [154]. Such systems, targeting specific research challenges of data lakes, are also our main focus in this survey. We address them in Sections IV, V, and VII.

III. Data Lake Architecture and Proposed Categorization Criteria

The *architecture* of a data lake describes the structure and components of the system, indicating how to store, organize and use the data. A recent survey [126] elaborated on the categorization of existing data lake architectures, while a methodology for designing data lake architecture is discussed in [56]. In existing data lake architecture proposals, there are mainly two high-level data lake philosophies, *pond* and *zone* architectures. Rather than repeating similar content as in [56], [126], we briefly review pond and zone architectures in Section III-A. We mainly focus on presenting an integrative, function-oriented architecture and classification criteria for data lake studies in Section III-B. Additionally, we discuss the different kinds of data lake users in Section III-C.

A. Pond and Zone Architectures

The *pond architecture* [74] partitions ingested data by their status and usage. In specific, ingested data is first stored in the *raw data pond*, then transformed and moved to the *analog data pond*, *application data pond*, or *textual data pond* if possible. Associated processes are created to prepare the data for future analytical processing. Later on, valuable data is secured long-term in an *archival data pond*. For instance, analog data generated by an automated device is moved to the analog data pond followed by data reduction to a feasible data volume. In contrast, the *zone architecture* [26], [111], [122], [131], [156], separates the life cycle of each dataset into different stages. For instance, there could be individual zones for loading data and checking data quality, storing raw data, storing cleaned and validated data, discovering and exploring the data, or using the data for business/research analysis.

B. Proposed Architecture and Categorization Criteria

High-level architectural philosophies, such as pond or zone architecture, often lack technical details about **functions**, which hampers modular and reusable implementations. Therefore, we propose an architecture based on our previous works [61], [78], as in Fig. 2. This architecture can also be seen as an abstraction of earlier tier-based data lake proposals [44], [131], [140]. Notably, here we define specific functions in each tier, which are not covered in these proposals. The goal of proposing such an architecture is two-fold. First, it clarifies the necessary functions in the whole workflow of a data lake, and provides a more comprehensive view compared to earlier DL

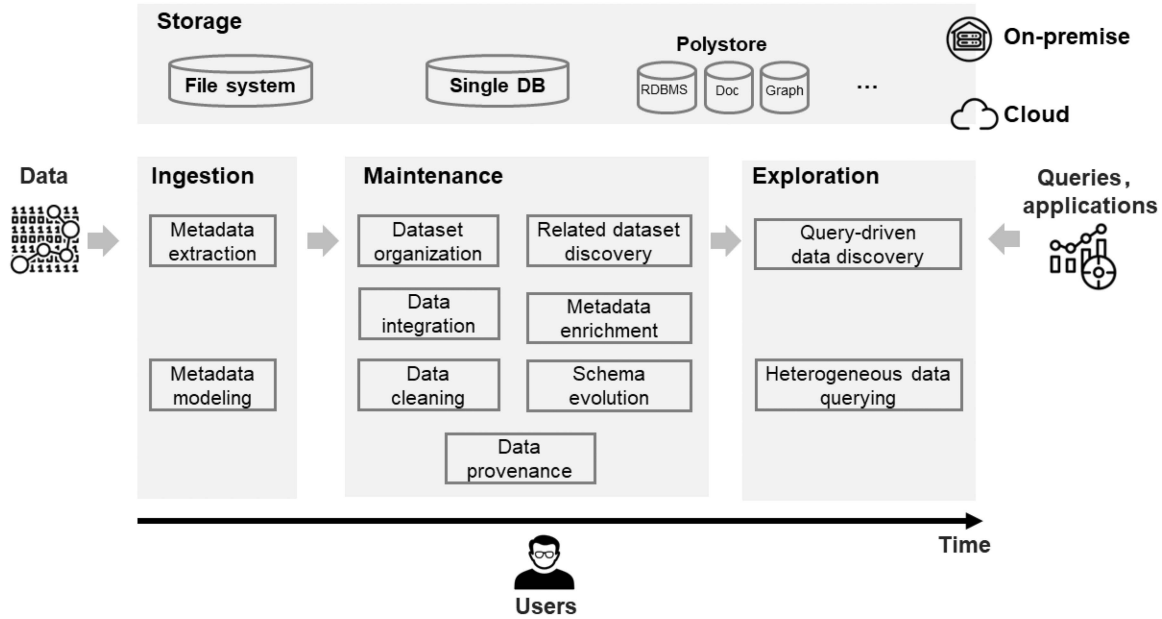

Fig. 2. Proposed architecture for data lake solution categorization.

architecture proposals. Second, it enables a fine-grained categorization and in-depth comparison of existing systems for each function.

Proposed Architecture. The data lake architecture in Fig. 2 provides a three-tier function-oriented linkage between data lake users, and a storage tier encompassing potentially multiple different technologies. In Section IV we review storage strategies applied in existing data lake solutions, which could be on-premise or cloud, single or multiple data storage systems, relational or NoSQL databases.

The architecture divides functions of the whole workflow into three tiers, according to *when* the functions are needed. There are certain functions during or right after data is ingested, which are in the *ingestion tier*. The ingestion tier is responsible for importing data from heterogeneous sources into the data lake system. The main challenges are about extracting and modeling metadata (Section V).

Functions in the middle (i.e., the *maintenance tier*) are for general management and organization of ingested datasets, which can also be considered as the preparation for querying. To prepare ingested raw data for querying or analytics, a data lake needs a set of operations in the maintenance tier to organize, discover, or integrate datasets (Section VI).

Some functions are triggered by user queries or applications of a data lake, as shown on the right side of Fig. 2, i.e., the *exploration tier*. These functions mainly contribute to allowing users to access the data lake. We observe two manners of exploring in existing works: query-driven data discovery and heterogeneous data querying, which we discuss in Section VII. They also form the basis for external application tiers on top of these three tiers, e.g., for visualization [99] or machine learning.

Three-Level Classification Criteria. As one of our main contributions, here we explain our classification criteria applied in this survey, which is based on our proposed data lake architecture in Fig. 2. Many existing research works focus on one specific function inside a data lake. Thus, we classify them by their functions in the architecture in Fig. 2. Table I shows such a categorization: we first group the existing systems by the tier and then the exact provided function. Of course, some systems provide more than one function. For instance, some data discovery systems in Table I also have components for metadata extraction and querying. We will give a detailed explanation of each functional criterion in Sections V, VI, and VII. For some extensively studied functions, e.g., metadata modeling, related dataset discovery, we further categorize the systems based on their methods. As emphasized in Section I, in this survey we mainly compare existing systems developed for data lakes. The 11 functions in Table I are summarized based on existing data lake related studies. For more potentially useful functions in a data lake architecture, yet not studied, see [115]. In summary, we follow a three-level categorization of existing data lake solutions: tiers (*when* the function is needed), functions (*what* the function is), and methods (*how* the function is achieved).

Lake-Specific Research Perspectives. Some functions in Table I are challenges almost every data management system with heterogeneous data needs to face, e.g., related dataset discovery, metadata extraction and enrichment, data cleaning, and data provenance. However, as discussed in Section I, data lakes require a novel, flexible manner of data management, which also leads to new research challenges. By ingesting the raw data as it is, we cannot simply refer to an existing schema. Instead, we need to actively discover related datasets partially at ingestion time, but mostly during maintenance or even at querying time. Therefore, we discuss related dataset discovery methods from both maintenance (Section VI-B) and exploration (Section VII-A) perspectives. Moreover, a data lake often needs to ingest a large volume of data, possibly also at a high velocity or even as continuous data streams, which cannot be stored in full in the data lake. Not all metadata can be extracted at ingestion time (Section V-A), but we need to continue enrichment during later

phases as well (Section VI-D). For similar reasons, we assign the continuous activities of data cleaning (Section VI-E) and data provenance (Section VI-G) in the maintenance tier.

C. Data Lake Users

As shown in Fig. 2, the complete picture of a data lake also includes human users. Users often interact with a data lake in different roles. According to [26], a business data lake scenario typically includes: (1) *data scientists and business analysts* who build and apply analytics models over the data lake, (2) *information curators* who define new data sources, organize and maintain the metadata in the catalog of existing data sources, (3) the *governance, risk, and compliance team* who ensures that the organizational regulations and business policies are followed (e.g., an auditor), and (4) the *operations team* that maintains the data lake (e.g., data quality analysts, integration developers). Such users can help improve enriching the semantics of data lakes over time, by adding metadata tags and linkage information based on conceptual models or standard vocabularies with respect to ontologies, such as schema.org [78], [138]. Moreover, it is not an easy task to design a data lake that has effective control of data security over diverse users and heterogeneous data stores in data lakes. CoreDB [9], [10] creates different users or roles for access control, and enables authentication and data encryption. A few tools are mentioned in [29] for system authentication, authorization, and data encryption based on the Hadoop platform, e.g., Apache Ranger[8].

In this survey, our focus is to conduct a technical comparison of systems, instead of human-system interaction. In the following, we do not emphasize these different kinds of users and generally call them data lake users.

IV. Storage

An important aspect of a data lake's architecture is the *storage tier*, which specifies the technology used for storing data. In what follows, we show that some approaches rely on the common relational or NoSQL databases while others have developed new storage systems, or combinations (polystores); the upper part of Fig. 2 depicts such diverse choices, which could be operated on-premise or in the cloud. We classify the existing data storage solutions for data lakes by how the ingested data is stored in the lake: as files (Section IV-A), in a single database (Section IV-B), or using polystores (Section IV-C). We also briefly mention industrial solutions that build data lakes on cloud platforms (Section IV-D).

A. File-Based Storage Systems

The Hadoop Distributed File System (HDFS) is one of the most frequently mentioned data storage systems for data lakes [13], [19], [138]. HDFS supports a wide range of files [60]. Besides text (e.g., CSV, XML, JSON) and binary files (e.g., images), it supports certain formats for data compression, e.g., Snappy[9], Gzip[10]. It also allows columnar storage formats such as Parquet[11] and row-based storage format Avro[12] that enable easy schema management. As one of the most common data storage options for data lake, file systems such as Hadoop are widely used in practice. In this survey, we list a few representative systems as follows.

Hadoop alone usually does not fulfill the goals of a data lake. Microsoft's *Azure data lake store* [120] offers a hierarchical, multi-tier file-based storage system[13]. It applies *Azure Blob storage*, which is a cloud storage solution optimized for large unstructured object data. It also supports HDFS and the Hadoop ecosystem, e.g., Spark, Sqoop[14]. Azure also has an indexing subsystem *Hyperspace* [1]. Another system built upon HDFS is *CLAMS* [47]. CLAMS is a prototype system that stores the ingested dataset in HDFS, and allows users to register the datasets for constraint discovery and data cleaning for data lakes.

B. Single Data Store

Some DL systems aim at specific types of data and employ a single database system at their storage tier. As an example, the *personal data lake* [143] applies a graph-based data model (i.e., property graphs), and stores data in Neo4j. The proposed data lake has a special focus on user data. Such data is usually relatively small in size compared to business scenarios, but imposes higher requirements regarding data privacy. Heterogeneous personal data fragments generated from user-web interaction (structured, semi-structured, unstructured) are serialized to specifically defined JSON objects. These are flattened to Neo4j graph structures with extensible metadata management in the data lake, categorizing for kinds of data: raw data, metadata, additional semantics, and the data fragment identifiers.

Potential Techniques. Going further, *multi-model databases* support multiple data models and formats in a single database (for a survey, cf. [88]). However, before choosing a multi-model database in a data lake, one should also check the underlying storage strategy, i.e., native storage support for different data models or merely different interfaces to the same storage strategy. If not, polystores should be considered, as discussed next.

C. Polystore Systems

Polystore (or *multistore*) systems provide an integrated access to a hybrid of multiple data stores for heterogeneous data. By definition, a data lake supports heterogeneous data in raw format, e.g., for zone architectures [156]. Thus, polystores are a feasible choice when a data lake is diverse.

Constance [61], [65] applies polystores, and stores the diverse raw data according to its original format: relational (e.g., MySQL), document-based (e.g., MongoDB), and graph databases (e.g., Neo4j). For instance, a JSON file will be stored in MongoDB. If an input dataset cannot be directly stored in a relational or NoSQL store, or considering, e.g., scalability for distributed computing, data can also be stored in HDFS. An example would be a data source producing streams of large binary image files, which requires parallel data compression. If

[8]https://ranger.apache.org/
[9]https://github.com/google/snappy
[10]https://www.gzip.org/
[11]https://parquet.apache.org/
[12]https://avro.apache.org/
[13]The latest system is known as *Azure Data Lake Storage Gen2*.
[14]https://sqoop.apache.org/

these defaults seem inadequate, users can specify the data store via the user interface.

Google Dataset Search (GOODS) [67], [68] supports heterogeneous data storage used in Google's key-value stores *Bigtable* [22], file systems, and *Spanner* [28] (see also Section IV-D).

Another data lake service allowing raw data stored in both relational and NoSQL is *CoreDB* [9], [10]. To store diverse data from web applications, besides relational databases (e.g., MySQL, PostgreSQL, Oracle), it supports multiple NoSQL systems, i.e., MongoDB, HBase[15], and HIVE. JSON is used as a unified format to represent entities.

Juneau [151] supports data science tasks in data lakes. It mainly focuses on tabular data or nested data that can be easily unnested into relations. Besides data processed by the notebook kernel, Juneau also handles (Jupyter, JupyterLab, Apache Zeppelin, or RStudio) notebooks, workflows in a notebook, and cells that constitute a workflow. Moreover, it needs to generate and store the relationships of these data objects as graphs. Thus, it applies both a relational database (PostgreSQL) and a graph database (Neo4j).

Potential Techniques. Since 2015, the study of polystores has been booming with regard to systems (e.g., *Polybase* [69], *BigDAWG* [41]) and open-source tools (e.g., Drill[16], Spark) – see [141] for a survey. Although they are not claimed as data lakes, they are potentially useful techniques to be part of a data lake architecture. In Section VII-B we continue with how to query a polystore.

D. Cloud Data Lakes

Recently, it is becoming a common practice to build large-scale commercial data lakes on cloud infrastructure [144], [148], [152]. Cloud-based storage choices include single-cloud, multi-cloud and a hybrid of cloud and on-premise platforms [148]. Several major cloud database vendors are promoting server-less data analytics and native cloud platform for building data lakes, most prominently Amazon Web Services (AWS)[17], Azure Data Lake Store[18], Google Cloud Platform (GCP)[19], Alibaba Cloud[20], and the Data Cloud from Snowflake[21]. Cloud platforms have several prominent advantages for data lakes. In a cloud data lake, one can scale storage space and computation power dynamically, and in many cases, the prices of resources are more economical than on-premises. Moreover, major cloud vendors provide many additional analytics tools in their product portfolio, e.g., AI services and data visualization tools, which make it convenient for developing different applications on top of data lakes. There are proposals [1], [144] on building indexing structures for cloud data lakes. Nevertheless, relying on a cloud platform also implies risks and challenges in some aspects such as data security, data provenance, and fault tolerance.

[15] https://hbase.apache.org/
[16] https://drill.apache.org/
[17] https://aws.amazon.com/big-data/datalakes-and-analytics/
[18] https://azure.microsoft.com/en-us/solutions/data-lake/
[19] https://cloud.google.com/solutions/build-a-data-lake-on-gcp
[20] https://www.alibabacloud.com/product/data-lake-analytics
[21] https://www.snowflake.com/workloads/data-lake/

E. Summary and Discussion

Choosing the *right* data storage system is one of the most important parts of architecting a data lake. A data lake designer needs to factor in not only the raw data itself, but also how the data will be used. We have shown that the choices are diverse: file systems or databases (relational or NoSQL), single or hybrid systems, on-premise or cloud, etc. The specific choice of storage strategy often shapes the required functions, which we introduce next.

V. Ingestion

During the ingestion phase, a data lake loads raw data. Without any additional information or insights, i.e., *metadata*, a data lake is hardly usable as the structure and semantics of the data are not known; this could potentially turn the data lake into a 'data swamp'. Therefore, it is crucial to acquire as much metadata as possible from the data sources. There are different types of metadata: schemata which preserve the structure of the dataset, semantic metadata, constraints, and other descriptive information, etc. During or shortly after data ingestion, existing solutions mainly extract metadata from the input datasets and model them. Thus, in this section, we focus on metadata extraction (Section V-A) and modeling (Section V-B).

A. Metadata Extraction

Metadata extraction is the process of discovering metadata information of a dataset. Often, structural metadata is extracted in the first place, but also semantical information and relationships to other datasets. In a data lake, metadata extraction is essential for accessing datasets in a later phase. In Section VI-D, we further address extracting *hidden* metadata such as functional dependencies. Given semi-structured or unstructured data, existing approaches [53], [61], [117] extract primarily structural metadata, while the one in [136] extracts metadata related to content and context.

The *Generic and Extensible Metadata Management System* (*GEMMS*) for data lakes [117] is a framework for extracting metadata from heterogeneous sources, which is then stored in an extensible metamodel. Since the data sources and schemata may change over time, it is important that the data lake has a flexible and extensible manner of metadata extraction. For each input file, GEMMS first detects its format, then initiates a corresponding parser to obtain the structural metadata (e.g., trees, tables, and graphs) and metadata properties (e.g., header information implying the content of the file). A tree structure inference algorithm is implemented for structural metadata extraction, which iterates semi-structured data in a breadth-first manner, and detects the tree structure. A follow-up work, Constance [61], can also extract structural metadata, i.e., schemata from semi-structured files such as XML and JSON.

DATAMARAN [53] provides a three-step algorithmic approach to extract structures from semi-structured log files. The records of its input log files, span multiple lines, with record types and boundaries. It first generates candidate *structure templates*, which use regular expressions [135] to express the record structure while allowing minor variations. The structure templates are stored in hash-tables, and only the ones satisfying

a coverage threshold assumption are kept. Next, redundant structure templates are pruned based on a specially designed score function, and finally further optimized using two refinement techniques over the pruned structure templates. The process of DATAMARAN does not require human supervision and provides a high extraction accuracy compared to existing works. In the experiments, the authors crawled 100 datasets with large log files from GitHub to mimic a real data lake.

Skluma [136] extracts metadata regarding content and context from scientific data files. The types of input files are diverse, e.g., JSON, CSV, unstructured texts, and images. It first finds the name, path, size, and extension of the files; then it infers file types and adds specific extractors accordingly to process tabular data, free texts or null values, etc. With the growing importance of Research Data Management for replicability in scientific studies, we expect such approaches to grow significantly over the following years.

B. Metadata Modeling

Metadata modeling answers the question of how to structure and organize the metadata. Notably, it is a necessary step to make the content of a data lake findable, accessible, interoperable, and reusable (FAIR Principles [146]). The majority of proposed models are either logic-based or graph-structured. Here we categorize the existing solutions based on the *types of metadata models:* generic models, data vault, and graph-based models.

1) Generic Metadata Models: The logic-based metadata model of GEMMS [117] has different model elements and allows the separation of metadata containing information about the content, semantics, and structure. It captures the general metadata properties in the form of key-value pairs, as well as structural metadata as trees and matrices to assist querying. Moreover, domain-specific ontology terms can be attached to metadata elements as semantic metadata. In [64], this metadata model is extended for representing individual schemata for relational tables, JSON, and labeled property graphs of Neo4j.

Another generic metadata model is *HANDLE* [43]. It has three abstract entities: data, metadata, and property. HANDLE enables flexibility with fine-grained levels, and it adapts the zone architecture mentioned in Section III-A. The elements of the GEMMS model can also be mapped to HANDLE. Finally, HANDLE can be used for linked data and can be implemented in Neo4j.

2) Data Vault: For structured or semi-structured data in typical business scenarios, a promising conceptual modeling environment is *data vault* [57], [107]. It has three main types of elements: *hubs* representing business concepts, *links* indicating the many-to-many relationships among hubs, and *satellites* with descriptive properties of hubs and links [86], [87]. Nogueira et al. [107] show how their conceptual model based on data vault can be transformed into relational and document-oriented logical models, and further to physical models (PostgreSQL and MongoDB, respectively). Giebler et al. [57] have reported their experience with applying data vaults for data lakes in the domains of manufacturing, finance, and customer service. They also pointed out practical obstacles, such as inconsistencies among data sources.

3) Graph-Based Metadata Models: Adapting ideas about knowledge graphs from the linked data and Semantic Web communities, several network- or hypergraph-based metamodels have been proposed for data lakes. In the business context, Diamantini et al. [34], [35], [36] propose a network-based metadata model, focusing on business names, data field descriptions, and rules, in addition to data formats and schemata. It creates a graph-based representation with XML/JSON nodes and labeled arcs indicating their relationship. Nodes can be merged based on lexical and string similarities, and linked to semantic knowledge (e.g., from DBpedia). The authors suggest extracting thematic views of interest to the business, similar to data marts in data warehouses. In [36] the proposed model can support unstructured data, besides (semi-)structured ones.

To efficiently discover relevant datasets from massive data sources, *Aurum* [48] devises an *enterprise knowledge graph (EKG)* to capture and query relationships among datasets. An EKG is a hypergraph with three elements: nodes, weighted edges, and hyperedges. Nodes represent dataset attributes, which are connected by edges when there is a relationship among them; hyperedges represent different granularities among arbitrary numbers of nodes, e.g., connecting attributes and tables. Aurum builds the EKG, maintains it upon data changes and allows users to query it with a graph query language based on discovery primitives.

Sawadogoet et al. [127] emphasize six evolution-oriented features of metadata management: semantic enrichment, data indexing, link generation and conservation (discover hidden similarities or integrate existing links among datasets), data polymorphism (preserve multiple transformed forms of the same dataset), data versioning, and usage tracking. Taking these features into consideration, their metadata model encompasses the notions of hypergraph, nested graph, and attributed graph. In terms of content, it can describe attributes, objects, datasets, historical versions, (similarity or parent-child) relationships, logs, and indexes.

4) Summary and Discussion: Metadata extraction and semantically rich modeling of metadata are crucial issues for the ingestion phase of a data lake. The data vault model has been developed for more structured environments like data warehouses and does not seem to fully fit the requirements for unstructured data and flexibility in data lakes. Early lake-specific approaches like GEMMS aimed at structured metamodels (i.e., formally expressed as a UML class diagram), carefully designed to cover all the information possibly relevant for metadata management in a data lake. These approaches also have their limitations in flexibility: the metadata model might be easily extensible, but the management of the metadata (i.e., its storage, user interfaces for creation and manipulation, etc.) is much more challenging. Recent graph-based models are more promising for flexibility if the graph-based storage systems are applied for the management of metadata. Furthermore, as the linking of information in organizations and knowledge graphs become more important, graph-based metamodels are a better fit for these techniques.

VI. Maintenance

After ingesting heterogeneous raw data from diverse sources, a data lake is a vast collection of unrelated data, for which we have limited information. To make the data usable, the data lake needs to further *process and maintain* the raw data, e.g., find

more metadata, discover hidden relationships, and perform data integration, transformation or cleaning if necessary. As shown in Fig. 2, in this section we categorize the maintenance-related functions into seven groups and discuss the corresponding data lake solutions.

A. Dataset Organization

The *dataset organization* problem studies how to structure and navigate the massive heterogeneous datasets in data lakes. Existing solutions for this problem, define new *structures* to group and organize datasets for better understanding and accessing a data lake. We categorize them based on their *underlying technologies* to construct the structure for a data lake: catalogs, classification models, and directed acyclic graphs. Although the following systems provide means to organize a data lake, their exact goals differ. For example, KAYAK [90], [91] and Juneau [151] facilitate data science applications, while DS-Prox [3], [4], [5] is a pre-filtering step of schema matching [118].

1) Catalog-Based Organization: GOODS [67], [68] allows datasets to be created, stored, and modified first, before conducting metadata collection. For each dataset, it collects various metadata and adds it as one entry in the *GOODS catalog*, which is stored in Bigtable. To organize, profile and search datasets (e.g., cluster different versions of the same dataset), the metadata is classified into six categories, including basic, content-based, provenance, user-supplied, team, project, and temporal metadata. This categorization of metadata is closely related to Google's specific information retrieval requirements. If a data lake developer applies similar catalog-based organization strategy, she should be encouraged to customize the metadata catalog to her own needs.

2) Classification Model Based Organization: DS-Prox [5] and a later version *DS-kNN* [3] consider the dataset organization problem as a classification problem. In specific, DS-kNN incrementally adds every dataset into a new or existing category by applying k-nearest-neighbour (k-NN) search. Before the step of classification, DS-kNN first conducts data preparation by feature extraction. For each attribute, depending on whether its values are continuous or discrete, DS-kNN extracts statistical or distribution-based features respectively, e.g., average numeric mean, or the average number of values. Such data-based features are added to each dataset, together with other features based on extracted metadata, e.g., the number of attributes, and types of each attribute. Using these features, DS-kNN computes dataset similarity by employing Levenshtein distance [95]. Next, given a new dataset, the proposed classification-based algorithm returns top-k neighbors (classified datasets), from which DS-kNN chooses the most frequently appeared category, then assigns the current dataset to this category. If none of the existing datasets are found, the new dataset is assigned to a new category. Finally, the datasets in the lake can be visualized as a graph: each node is a dataset, and edges between two nodes are labeled with the similarity of the two datasets. A later work [4] uses supervised ensemble models to obtain the similarity values between dataset pairs.

3) DAG-Based Organization: In what follows, we introduce the dataset organization solutions that apply *directed acyclic graphs (DAGs)*. Although they all apply DAGs to organize and navigate a data lake, the exact functions and the definitions of DAGs differ, as listed in Table II.

KAYAK [90], [91] aims to support data science pipelines in data lakes, and it organizes the dataset relationships and operations. It defines a data lake as a collection of datasets, and manages the operations on the datasets, which are the basic blocks of data preparation pipelines. *Data preparation* refers to the processing of raw data, so that it can be used in downstream tasks, e.g., analytics. KAYAK first defines atomic tasks such as basic profiling and dataset joinability computation. Then a sequence of such atomic tasks further builds up a specific operation for data preparation, referred to as a *primitive*, e.g., insert a dataset. Table II shows two different usages and definitions of DAGs in KAYAK, pipeline and task dependency. To represent data preparation pipelines, it uses a DAG with primitives as nodes and their dependencies (based on execution order) as edges. To manage dependencies among tasks and execute the atomic tasks of a primitive in parallel, KAYAK defines the second type of DAG for task dependency as shown in Table II. Here each node represents an atomic task, and the directed edges indicate the execution order of two tasks. Such a DAG helps to identify which tasks can be parallelized during execution.

Nargesian et al. [104] define the *data lake organization* problem as discovering the optimal structure to effectively find the desired dataset in a data lake. Such a structure for navigating data lakes is referred to as an *organization:*

- As listed in Table II, a DAG-based organization in [104], has sets of attributes as nodes. The leaf nodes are attributes of input tables, while non-leaf nodes have a topic label that summarizes the set of attributes or topics represented by its child nodes. The edges represent containment relationships between the set of attributes represented by the nodes.
- To measure semantic similarities among attributes, attribute values are associated with n-dimensional representations [106], which enable the use of cosine similarity. The process of navigation is formalized as a *Markov model*, where the states are the nodes (i.e., sets of attributes) and transitions are the edges, i.e., future states depend only on the current state, not on all the historical states. Thus, given a query asking about a topic (e.g., searching keyword is $food$), the transition probability depends only on the current node in the DAG and the similarities between its child nodes and the given topic. The proposed algorithms in [104] try to find the organization structure that achieves the maximum probability for all the attributes of tables to be found.

A more recent system *RONIN* [110], combines navigation using the above DAG-based structure [104], with metadata keyword search and joinable dataset search in a data lake.

In Section IV-C we have mentioned that *Juneau* [151] handles computational notebooks, workflows, and cells, from which it builds graphs for data management. A *workflow graph* is a directed bipartite graph with two types of nodes: *data object nodes* which represent input/output files or formatted text cells, and *computational module nodes* representing code cells in a Jupyter notebook. If the data object is the input or output of the computational module, there is a directed edge connecting their nodes. Moreover, as shown in Table II, Juneau also has a DAG for

TABLE II
COMPARISON OF DAG-BASED DATASET ORGANIZATION APPROACHES IN SECTION VI-A3

System	*KAYAK [90], [91] (pipeline)*	*KAYAK [90], [91] (task dependency)*	*Nargesian et al. [104]*	*Juneau [152] (variable dependency)*
Function	Represent the primitives of a data preparation pipeline	Enforce correct execution sequence of tasks while parallelization	Semantic navigation	Measure table relatedness w.r.t. notebook workflow
Node	Primitives	Atomic tasks for data preparation operations	Sets of attributes	Notebook variables
Edge	Sequential execution order of two primitives	Sequential execution order of two tasks	Containment relationships	Notebook functions (as edge labels)
Edge direction	From the previous primitive to the subsequent primitive	From the previous task to the subsequent task	From the superset to the subset	From the input variable of the function to the output variable

managing the relationships of variables in notebooks, referred to as *variable dependency graphs*. In a variable dependency graph, nodes represent the variables, and the labeled, directed edges indicate that one variable is computed using another variable through a function. Via subgraph isomorphism, Juneau is able to discover tables sharing similar workflows of notebooks (similar sequences/patterns of variables and functions).

4) Summary and Discussion: In this subsection, we have discussed various methods on organizing datasets. Yet, each of these methods comes with its own use cases, merits and limitations.

In particular, while GOODS [67], [68] presents an innovative way to build a dataset metadata catalog, it is heavily tailored to datasets and practices used to produce them in Google; the majority of metadata crawled is connected to standardized processes used inside the company to create, maintain and store datasets. Methods based on classification models, like DS-Prox [5] and DS-kNN [3], are ideal when the goal is to group datasets sharing some kind of relatedness. Yet, there are types of relatedness among datasets that might not be covered by such simple metadata based on data instances, e.g., semantics-aware dataset unionability [106].

Finally, DAG-based organization methods vary considerably in terms of functionality. KAYAK [90], [91] computes inter-dataset metadata regarding only equi-joins among tabular datasets. Moreover, Nargesian et al. [104] and RONIN [110] focus on organizing attributes as DAGs to maximize the probability of users finding relevant tables with respect to their needs. Importantly, this organization is based on containment similarities among the attributes, which means that fuzzy-kind similarities are not supported. On the other hand, Juneau [151] exploits workbook metadata, which presents a promising signal of inter-dataset similarity.

B. Related Dataset Discovery

In data lakes, the process of *related dataset discovery*, also referred to as *data discovery*, tries to find a subset of relevant datasets that are similar or complementary to a given dataset in a certain way, e.g., with similar attribute names or overlapping instance values. Since a data lake stores and manages a large number of datasets, it is neither realistic nor necessary to query or integrate all of them. Therefore, it can be more beneficial to first discover datasets that are useful for a specific purpose. Moreover, the relatedness discovered among datasets is also a valuable and essential type of metadata for exploring a data lake and preventing a data swamp, e.g., for enabling entity resolution or resolving inconsistency across datasets.

As shown in Table III, we present the solutions that address the related dataset discovery problem in data lakes. The systems in this group mainly handle tabular data, or hierarchical data that can be transformed into tabular data (not necessarily relational data, i.e., some may even violate the first normal form). We categorize these systems primarily based on the *types of relatedness* they use: joinable tables [14], [15], [48], [154] (Section VI-B1), tables related for data science tasks [75], [150], [151] (Section VI-B2), and tables with semantic relationships [40], [121] (Section VI-B3). The fourth group of approaches focuses on the *scalability* issue during the discovery process [12] (Section VI-B4). However, note that solutions belonging to one category might also be applicable to other cases: joinable tables can be used for data science, and semantically related tables could also be joinable, etc.

1) Discovery of Joinable Datasets: Aurum [48] enables the discovery of joinable datasets by building a hypergraph (i.e., EKG) which stores information on how columns of different tabular datasets might be related. To construct it, Aurum first profiles each table column by adding *signatures*, i.e., information extracted from column values such as cardinality, data distribution, and a representation of data values (i.e., MinHash). Then, it indexes these signatures using locality-sensitive hashing (LSH).

When two columns have their signatures indexed into the same bucket after hashing, an edge is created between corresponding nodes, and their similarity score is stored as the edge weight. Aurum also detects primary-foreign key relationships between columns by first inferring approximate key attributes. A highlight of Aurum is its efficiency of computing set similarities. More specifically, given the total number n of attributes of all datasets, instead of conducting an all-pair comparison of $\mathcal{O}(n^2)$ complexity, it profiles columns with signatures and stores them in an LSH-index; then, by using approximate nearest neighbor search, it reduces to linear complexity. When changes occur in the data, Aurum does not re-read it from scratch. Only if the difference compared to the original values is above a threshold, it updates column signatures and the hypergraph.

Brackenbury et al. [15] provide a high-level data lake proposal, which shares a similar idea to Aurum, in terms of using multiple criteria to measure dataset similarities. The difference is that when the algorithms alone cannot provide reliable suggestions, it also includes humans in the loop. To find joinable datasets, it measures the similarity of files (e.g., HTML tables), and considers approximate matches in terms of data values, schemata and descriptive metadata (the source of data, information added by users, etc.). For measuring the similarity of the files and clustering them, it computes the Jaccard similarity between file paths using MinHash and LSH.

TABLE III
COMPARISON OF RELATED DATASET DISCOVERY APPROACHES IN DATA LAKES

Systems	Relatedness criteria	Similarity metrics	Applied technique
Aurum [48]	Instance value overlap Attribute name PK-FK candidate	Jaccard similarity (MinHash) Cosine similarity (TF-IDF)	Hypergraph
Brackenbury et.al. [15]	Instance value overlap Attribute name Semantics Descriptive metadata	Jaccard similarity (MinHash)	-
JOSIE [154]	Instance value overlap	Intersection size of sets	Inverted Index
D^3L [14]	Instance value overlap Attribute name Semantics Data value representation pattern (Numerical) data distribution	Jaccard similarity (MinHash) Cosine similarity (Random projections)	5–dim Euclidean space
Juneau [75], [150], [151]	Instance value overlap Domain overlap Attribute name Key constraint New attributes rate New instance rate Variable dependency Descriptive metadata Null Values	Jaccard similarity	Workflow graph Variable dependency graph
PEXESO [40]	(Textual) instance values	Any similarity function in a metric space	High-dimensional vectors Hierarchical grids Inverted Index
RNLIM [121]	Table name Attribute name Attribute data type Attribute value domain	–	BERT [33]
DLN [12]	Attribute name Instance values	Jaccard similarity Cosine similarity	Classification models

JOSIE [154] handles data lakes with tabular data, e.g., a corpus of web tables. It addresses two challenges with regard to applying existing overlap set similarity search solutions in data lakes: *i*) a data lake may contain a large number of tables; hence the number of columns and distinct values could also be large, and *ii*) it could be difficult for a human user to directly give an appropriate threshold value θ for the intersection value. Thus, an exact top-k overlap set similarity search approach is proposed in [154], which enables *i*) scaling to large sets (with size over 1 K and maximum size in the millions) and *ii*) returning top-k results without the need of human-defined threshold value θ.

Given a table T in the data lake, and one specific column C from T, JOSIE can return tables in the data lake that could be joined with T on C. The task is formalized as the problem of *overlap set similarity*, which considers the table columns as sets, and the same tuple values as the set intersection. Each table in the output contains a column that has an overlap with C, and the intersection value is larger than a given threshold θ. Then naturally, the problem of joinable table discovery is transformed into the problem of finding the exact top-k overlap set similarity search. The measurement used in JOSIE is the *intersection size* of the sets, also referred to as *overlap similarity*. For returning top-k sets JOSIE has applied *inverted indexes*, which map between the sets and their distinct values and make JOSIE scalable with a large number of tables. JOSIE employs a cost model to eliminate unqualified candidates effectively. Such a method makes the performance robust to different data distributions.

D^3L [14] also incorporates multiple criteria to decide whether a dataset is relevant to another. In particular, it regards five signals of dataset similarity: *i*) attribute name similarity, *ii*) instance value overlaps between columns, *iii*) embedding similarity of columns, *iv*) format similarity of instance values, and *v*) distribution similarity of numerical attributes. Therefore, given table attributes, D^3L first transforms schemata and data instances to intermediate representations of q-grams, TF/IDF tokens, regular expressions, word-embeddings [79], and the Kolmogorov-Smirnov statistic [27]. Based on these five features, D^3L transforms the problem of finding the relatedness between tables to the calculation of weighted euclidean distance in a 5-dimensional space. In doing so, the weight of each feature (i.e., feature coefficients) indicates its significance for the combined distance. To tune the feature weights, D^3L trains a **binary classifier** over a training dataset with relatedness ground truth, and applies the coefficients of the trained model as the weight of features for distance calculation. Similar to Aurum, D^3L builds LSH to index the features and maps them to the distance space, where two items are considered to be similar if they are hashed into the same bucket. An interesting finding is that using LSH to discover joining paths leads to accurate discovery of more related tables (and attributes).

2) Discovery of Task-Specific Datasets for Data Science: Juneau [75], [150], [151] provides searching over related tables from a different perspective. It extends computational notebooks (e.g., Jupyter) and supports common data science tasks, such as finding additional data for training or validation, and feature engineering. When users specify the desired target table, the

system can automatically return a ranked list of tables, which might be relevant to the given table. Specifically, as shown in Table III, Juneau extends the notion of "relatedness" with the following signals.

1) In addition to the instance value overlap and similar attribute names, it considers pairwise matched attributes that share similar domains, and matched candidate key pairs. The proposed similarity metrics are based on Jaccard similarity; sketches and LSH-based approximation [49], [155] are mentioned as alternatives for scalability.
2) To augment user queries, it may suggest data instances or attributes in the candidate tables, which do not exist in the target table.
3) Based on the variable dependency graphs (Section VI-A), it defines the *provenance similarity* of two tables based on the graph similarity of their variables. This measurement aims to help connect variables and tables via user-defined workflow operations. This allows finding new tables that are related to the current table via workflows.
4) Juneau also identifies similar tables with regard to descriptive metadata (e.g., information about the data science task), and the number of null values (e.g., fill missing values in a data cleaning task).

For a specific data science task, Juneau picks a subset of relatedness features and computes similarities based on them. For instance, when searching tables for a data cleaning task, it considers the instance value overlap, schema overlap, provenance similarity, and null value differences.

3) Discovery of Semantically Related Datasets: PEXESO [40] tackles the problem of finding semantically joinable tables when considering only textual attributes. Towards this direction, it transforms textual values into high-dimensional vectors, and computes their vector similarities. For efficient similarity computation among such representation vectors, it utilizes an inverted index, and a *hierarchical grid* which is used for partitioning the space.

The *Relational Natural Language Inference Model* (RNLIM) [121] is a framework that transforms related attribute discovery to unsupervised natural language inference [89], which determines whether a hypothesis can be inferred from the given premise texts. Contrary to other data discovery systems, it focuses on specifying semantic relationships between the tables. That is, given a pair of attributes, RNLIM optimizes a neural network for labeling their relatedness. More specifically, RNLIM considers four signals and separates them into two groups: table and attribute names, attribute data types and attribute value domains. For each such group, it uses multiple matching methods. For instance, to perform the domain match between numerical attributes, it uses the Kolmogorov-Smirnov statistic, which is similar to D^3L [14]. Using pre-trained language representation models from BERT [33], RNLIM generates similarity-preserving representations from these two groups of signals, which enable the training of a classification model.

4) Scalable Related Dataset Discovery: Data Lake Navigator (DLN) [12] has a different focus compared to the aforementioned data discovery systems, which often require processing of all the available data, and hence hinder scalability. DLN tackles the problem of handling large-volume data at the enterprise level, e.g., a data lake with petabytes or even exabytes of data. Consider a data lake with stream data. DLN discovers related columns in the streams with respect to a given column. The core solution of DLN is building random-forest classification models. In specific, DLN considers textual and numerical attributes, and extracts two types of features from them: metadata features, including attribute names and uniqueness, and data-based features. Accordingly, it builds two classifiers. The first classifier uses only metadata features. The second classifier is an ensemble model, which only uses metadata features for numeric attributes, and both metadata features and data features for textual attributes. Notably, for learning classification models DLN needs labeled samples. In essence, it labels the attribute-pairs in the JOIN clauses of queries as positive samples (related columns), whereas it samples negative examples of attribute pairs that never appear in any JOIN clause (non-related columns).

5) Summary and Discussion: Related dataset discovery is a well-researched topic with respect to data lakes. Yet, among all the different methods that have been proposed, one can identify a standard procedure that they follow: the first step is to define and extract relatedness signals from tables w.r.t. data (e.g., value overlaps, data distribution patterns), schemata (e.g., attribute names, key constraints), semantics, and descriptive metadata. The next step is to compute multi-dimensional similarities between attributes (e.g., based on Jaccard similarity or cosine similarity), and aggregate them to an overall similarity between tabular datasets. The LSH index and its extensions (e.g., LSH Forest [8]) are often used to index and map feature values to boost performance or increase the accuracy of relatedness. Another important part of data discovery is querying the data lake, which is discussed in Section VII-A.

Nonetheless, we also find that data discovery solutions may differ in their focus. Aurum is fast and robust against data value changes and offers a graph-based structure, whereas JOSIE shows high performance. D^3L improves the accuracy of discovered related tables by employing multiple similarity signals. Juneau emphasizes workflows for multiple data science tasks. To obtain semantic relatedness, PEXESO uses high-dimension vectors, while RNLIM relies on BERT. DLN addresses the challenge of related dataset discovery for exabyte-scale data lakes. Therefore, as shown in Table III, their individual relatedness criteria, similarity measures, and applied techniques vary significantly. Recent system demonstration proposals also indicate the possibilities of applying knowledge graphs [72] and example-based interaction [123] for data discovery in data lakes.

Data discovery has been intensively studied beyond the scope of data lakes. We refer the reader to recent tutorials [105] and surveys [149] for exploring more **potential** solutions for data discovery in data lakes. It is our firm belief that data discovery solutions for data lakes will continue being introduced, due to the value and insights they bring to businesses and organizations.

C. Data Integration

Data integration (DI) studies the problem of combining multiple heterogeneous data sources and providing unified data access for users [39]. Given a large scale of sources in a data lake, users might need to first discover a subset of relevant datasets, before resolving the heterogeneities of sources with regard to

data models and schemata. Thus, in some literature [100], the related dataset discovery (cf. Section VI-B) is also considered as part of data integration. Here we consider the fundamental data integration steps including schema matching [118], schema mapping [45], entity linkage [16], query reformulation [66], etc.

Few data lake proposals provide an end-to-end data integration pipeline. Constance [61] uses the generic metadata model for extracted schemata of relational, JSON documents and graph data (see Section V-B1). For data integration Constance first performs schema matching, which finds semantically related attributes. Users can select a subset of data sources and schema elements via the user interface, and the system generates an *integrated schema* for *partial* integration. Next, Constance generates schema mappings, which preserve the relationships between the source schemata and integrated schema [63]. With schema mappings Constance performs query rewriting and data transformation in a **polystore**-based setting [65]. It rewrites the input user query (against the integrated schema) to subqueries (against source schemata), executes the generated subqueries in the query languages of each data store (e.g., MySQL, MongoDB, Neo4j), and retrieves the subquery results. For the final integrated results it further resolves the data type and value conflicts while merging the subquery results. It also pushes down selection predicates to the data sources to optimize query execution and reduce the amount of data to be loaded.

ALITE [82] deals with the problem of integrating related tables in data lakes that have been obtained from dataset discovery tasks (Section VI-B). Particularly, the method gathers results from top-k unionable and joinable queries on datasets and applies holistic schema matching. To do so, it leverages embeddings on language models, namely distributed vector representations with the following property: datasets that are similar to each other are embedded close in the dimensional space (based on the distributional hypothesis [71]). Specifically, ALITE embeds columns by using state-of-the-art techniques such as TURL [32], and then applies hierarchical clustering in order to obtain sets of columns that are related. Finally, based on the aligned columns, it computes the Full Disjunction (FD) [52] among discovered datasets in an optimized way.

D. Metadata Enrichment

In this survey, we refer to *metadata enrichment* as the process of creating implicit metadata from raw data in the data lake, which often requires intensive computation or human effort. Notably, in Section V-A we have discussed systems *extracting* embedded metadata. Here we discuss the necessity to compute and extract "more hidden" information from the data, which helps to better understand and explore datasets in the data lake. Such a process is time-costly, and sometimes impossible or unnecessary to be conducted during data ingestion. The metadata discussed in this section is more relevant to the functions in the maintenance tier. For example, the semantic metadata enriched by CoreDB can be used for data provenance. The semantic information of domains extracted by D^4 and DomainNet could be used to improve the process of related dataset discovery. Structural metadata discovered by Constance can be used for data cleaning, while descriptive metadata enriched by GOODS can be used for dataset organization and data provenance.

Next, we discuss systems fulfilling such a goal here, categorizing them based on the *types of metadata* that they discover: semantic, structural, or descriptive metadata.

1) Semantic Metadata Enrichment: CoreDB [9], [10] is a data lake service that aims at extracting insights from raw data. It first extracts essential information representative of the original raw data, referred to as features, e.g., keywords and named entities. Then it provides services that add synonyms and stems to such features, while it connects them to open knowledge bases such as Google Knowledge Graph[22], Wikidata[23]. CoreDB also annotates and groups the data sources in the data lake.

D^4 [109] tackles the problem of *semantic type detection*, also known as *domain discovery*. That is, given a set of input tables, D^4 discovers their semantic domains and represents each domain with a set of terms. For instance, if there are several color-related attributes, e.g., $vehicle_color$, $building_color$, $cloth_color$, then one of the output domains of D^4 is $color$, and it is represented by terms $\{red, white, black, green, \ldots\}$. The complete list of the terms of a domain, may come from multiple attributes, while an attribute may contain terms for several different domains. D^4 applies a data-driven approach, i.e., it processes all the data in the given set of datasets. Additionally, the approaches applied in D^4 allow it to cope with a large number of tables and attributes, and ambiguous terms (e.g., *Apple* can be a type of fruit or a brand name).

DomainNet [85] tackles a similar problem as D^4. It also discovers hidden semantics, and handles the ambiguity and incompleteness in data values. For instance, when the value *Apple* appears in multiple tables of a data lake, DomainNet tries to find out if it represents the semantics of one domain (fruit or brand), or both. As an approach developed for data lakes, it assumes that a priori knowledge about datasets could be missing, like the types of entities in a table. Its proposed approach includes building a network graph using data values and attribute names, followed by applying community detection over such a network.

2) Structural Metadata Enrichment: Constance [64] enriches the metadata in the data lake by discovering *relaxed functional dependencies (RFD)* [21]. The relaxed functional dependencies are relaxed in the sense that they do not apply to all tuples of a relation, or that similar attribute values are also considered to be matched. Such dependencies provide insights that specific attributes functionally depend on some other attributes in a loose manner, which apply to the ingested datasets even though they have a certain percentage of inconsistent tuples.

3) Descriptive Metadata Enrichment: In order to obtain metadata that describes dataset origin, ownership, and its possible usage, it is often beneficial to keep human experts in the loop. Google's data lake GOODS [67], [68] stores metadata of its datasets in the catalog, and it applies *crowdsourcing* for metadata enrichment. For instance, it allows adding descriptive metadata of datasets, marking datasets worth additional security attention, such that people from different teams of the organization (e.g., data owners, auditors, users) can exchange and communicate about the information of the datasets.

[22]https://developers.google.com/knowledge-graph/
[23]https://www.wikidata.org/wiki/Wikidata:Main_Page

E. Data Cleaning

Data cleaning is the process of discovering and fixing data quality problems. The data quality problems may reside in one or multiple sources, at the schema level or the instance level [119]. For example, a dataset may have missing values, misspellings and redundancies in its instances. When we talk about data cleaning in data lakes, we refer to dealing with data quality issues residing in the ingested raw data. Given the volume and variety of the data in a lake, it is ideal that the data cleaning approach can work with heterogeneous raw data, and reduce human effort. Thus, there are certain proposals about how to obtain hidden "rules" from the data in the data lakes, and then use them to improve the data quality. We divide the systems in this group based on the methods applied: constraint inference or validation rule inference.

1) Data Cleaning by Constraint Inference: CLAMS [47] uses *conditional denial constraints* to detect the potentially erroneous data. Given the RDF triples, a conditional denial constraint specifies a set of negation conditions about the tuples. The proposed approach automatically detects such constraints by discovering possible schemata from RDF data, and corresponding constraints. It examines the triples violating the obtained constraints and uses them to build a hypergraph, which indicates the number of constraints violated by each triple. Then, it accordingly ranks the RDF triples and asks the user to validate whether such a candidate dirty triple should be removed.

Constance [64] also uses discovered dependencies for data cleaning, whereas it applies *relaxed functional dependencies*. These dependencies are especially useful in cases where the source data has lower quality with inconsistencies and incorrect values. By using relaxed functional dependencies, Constance identifies the data objects violating the detected dependencies, which could be potentially erroneous data.

2) Data Cleaning by Validation Rule Inference: In [137], Song et al. have tackled a specific data cleaning problem, i.e., data validation. In a large enterprise data lake with terabytes of data, the data may change with time. The data validation rules indicate whether the changes are significant enough, and will affect the downstream applications. The approach in [137] tries to automatically derive such rules from the machine-generated, string-valued data, rather than inferred by human experts. In principle, it formulates the rule inference problem as an optimization problem, which balances between false-positive-rate minimization and quality issue preserving.

F. Schema Evolution

Schema evolution requires handling the changes of schemata and integrity constraints [31]. In data lakes, the possible challenges of schema evolution could be the heterogeneity of the schemata and the frequency of the changes. While data warehouses have a relational schema that is usually not updated very often, data lakes are more agile systems in which data and metadata can be updated very frequently.

Klettke et al. [83] address the problem of how to construct the whole evolving history of schemata given data stored in NoSQL databases, e.g., JSON stored in MongoDB. Instead of table schemata in relational databases, they consider the structure of persisted objects in NoSQL databases, referred to as *entity types*. The proposed approach first extracts each entity type from loaded datasets, with assigned timestamps that indicate its residing time interval. Then from different structure versions of the entity types, it detects the possible operations between two consecutive versions. In the case of multiple alternative operations, users will make the final validation. In addition, to detect certain schema changes, it is often useful to detect integrity constraints, e.g., inclusion dependencies. The assumption in [83] is that in NoSQL databases often schemata are "less" normalized, which leads to the inclusion dependencies involving multiple attributes rather than a single attribute as in relational databases. In [83] an algorithm is proposed to detect such k-ary inclusion dependencies.

G. Data Provenance

Data provenance (also known as *data lineage*) refers to meta information of data records, which indicates their origin, usage, status in the life cycle, etc. The provenance information can be seen as a special type of metadata, which tells how a dataset is obtained from original sources and helps to make proper access to datasets [133]. Such information could be extracted during data ingestion, and later enriched during maintenance or exploration, possibly with human input.

In [142], a governance tool from IBM is presented, which can manage the requests for ingesting new data sources or using already ingested datasets in a data lake. Suriarachchi et al. [140] propose an abstract architecture that provides integrated provenance (information of activities) given multiple data processing and analytics systems (e.g., Hadoop, Storm[24], and Spark), as these systems populate provenance events in different standards and apply various storage manners. They also study a use case, in which data from Twitter is collected and processed (e.g., count hashtags, aggregate data by each category) by Apache Flume[25], Hadoop jobs, and Spark jobs.

GOODS [67], [68], CoreDB [9], [10], and Juneau [75], [150], [151] all preserve the provenance information as graphs. As mentioned in Section VI-A1, GOODS stores provenance information in its metadata catalog, as one of the six metadata groups. It builds provenance graphs and visualizes them to users, such that users can keep track of the usage and transformation of the data. It exports the provenance metadata in the catalog as subject-predicate-object triples into a graph-based system, then generates the provenance graphs for visualization and path-based querying. CoreDB uses the descriptive, administrative and temporal metadata to build DAG-based provenance graphs [11], which helps answer questions such as who queried a specific entity. Juneau [151] generates graphs with variables as nodes, and connects two variable nodes in the same function. Given a variable v in the notebook, one can find all other variables affecting v via some functions, and the relationships between these variables and v.

VII. Exploration

It is important that useful information can be retrieved from data lakes. However, this is often a challenging task due to

[24]https://storm.apache.org/
[25]https://flume.apache.org/

the large number of ingested sources, and the heterogeneity of data. A user may have knowledge of one or a few data sources, but rarely, if not never, all the datasets. Thus, the existing solutions mainly solve the querying problem in data lakes in the following two directions: explore the data lakes based on the relatedness of datasets (Section VII-A), or provide a unified query interface for heterogeneous data sources (Section VII-B).

A. Query-Driven Data Discovery

Query-driven data discovery [100] refers to searching a data lake based on the measured relatedness (e.g., joinable) among datasets as introduced in Section VI-B. With input queries specifying a given dataset (usually tabular data), the system returns the top-k most related datasets.

Exploration Input/Output. There are three ways of exploration. We denote the set of datasets in a data lake as $\mathbf{S}$.

1) Given the user-specified table T and a column c of T, the system returns top-k tables that are most related to T, e.g., JOSIE [154].
2) Given a table T, the system returns top-k tables (referred to as $\mathbf{S}^k$) that contain relevant attributes for populating T, e.g., D^3L [14]. In addition, if a table S_i is not in the top-k result set (i.e., $S_i \in (\mathbf{S} - \mathbf{S}^k)$), yet it can be joined with some table(s) in $\mathbf{S}^k$ and improve the attribute coverage of T, D^3L also includes S_i in the result.
3) Given the user-specified table T and the search type τ for external applications (e.g., a data science task), the system returns top-k tables that are most relevant to T based on the relatedness measurements associated to τ, e.g., *Juneau* [151].

Notably, in this group of studies, the challenge of exploring datasets is a search problem rather than a query reformulation problem in data integration. It can also be seen as a step prior to data integration or data science tasks [14], [151].

Querying Methods and Indexing. Given an input query table, the systems in Table III often rely on similarity estimation using indexes (e.g., aforementioned LSH indexes, inverted indexes). They rank candidate tables, and include the top-k tables in the result. In addition, Aurum [48] applies a graph index to accelerate expensive queries containing discovery path queries for searching its hypergraphs. In its primitive-based query language, an Aurum user can compose queries to search schemata or data values with keywords to find specific columns, tables, or paths. Users can specify criteria and obtain ranked querying results in a flexible manner, i.e., they can obtain the ranking results of different criteria without re-running the query. In Juneau [151], a query is a cell output table picked by the user. The user also chooses the type of the search, e.g., find tables for data cleaning. Then Juneau uses the corresponding relatedness measurements to perform the top-k search. It speeds up the search with strategies such as *indexing* columns profiled in the same domain or tables connected by workflow steps, and *pruning* tables under a threshold of schema-level overlap.

Remarks on Future Directions. Data lake exploration could benefit greatly by taking into account recent results on web table exploration [18], [84], [113], data wrangling [51], [142], or external applications upon the data lake. With the existing works mainly focusing on evaluating the accuracy of similarity computation, or the performance (query processing time), deep analysis and further improvement on the accuracy and completeness of the top-k result set are still rare. Finally, the existing solutions in this group mostly study tabular data. In what follows, we discuss the data lakes that explore datasets with diverse data models.

B. Heterogeneous Data Querying

In this survey, by *querying heterogeneous data*, we indicate the systems providing a unified querying interface to access heterogeneously structured data. Next, we introduce studies that tackle such a research problem.

Constance [61], [65] provides an incremental manner for users to explore the data lake. Via the user interface, a user can first browse the existing data sources, including their description, statistics, and schema; then she can write a query (SQL or JSONiq[26]) for a single dataset. She can also make a keyword search over the schemata or the data. Alternatively, with certain knowledge of the datasets, which can be developed through the previous exploration processes, she can choose to integrate and query a subset of datasets as introduced in Section VI-C. In addition, users can transform data in the data lake to their desired structure and format. The information retrieval requirements of external applications are supported via RESTful APIs [55].

CoreDB [9], [10] provides users with a unified interface, i.e., through a REST API for querying data or performing *Create, Read, Update and Delete* (CRUD) operations. It applies Elasticsearch[27] for the underlying full-text search, SQL queries for relational database systems, and SPARQL queries for knowledge graphs.

Ontario [44], [80] and *Squerall* [94] both enable a federated query processing over a semantic data lake and apply SPARQL to query the heterogeneous data lake. Ontario supports heterogeneous data, e.g., RDF (stored in Virtuoso[28]), local JSON files, TSV files (in HDFS), XML files (in MySQL). It profiles each dataset with its metadata and additional information, e.g., the types of the source, or the web API for querying this type of source. For instance, for TSV files stored in HDFS, it provides Spark-based services which translate the SPARQL queries to SQL. Given an input SPARQL query, Ontario first decomposes the query. Then it uses the profiles to generate subqueries for each dataset with a set of proposed rules. Using metadata, it also tries to generate optimized query plans. In [124], the general guidance of query optimization on top of Ontario is proposed. Similar to Ontario, Squerall also supports querying diverse data sources, including files (i.e., CSV, Parquet) and relational (i.e., MySQL) and NoSQL databases (i.e., Cassandra, MongoDB). The schemata of the sources are mapped to a mediator, which consists of high-level ontologies. Given SPARQL queries against the mediator, relevant data entities are retrieved from data sources, which are joined and transformed to form the final query results. Squerall enables distributed query processing and

[26]https://www.jsoniq.org/
[27]https://www.elastic.co/
[28]https://virtuoso.openlinksw.com/

is implemented with two versions with different data connectors: Spark and Presto[29].

VIII. Challenges and Future Directions

We have addressed specific technical aspects of managing data lakes so far. Next, we discuss the challenges of applying data lakes in broad technological application domains in Section VIII-A, and two main future research directions of data lakes in Sections VIII-B and VIII-C.

A. Data Lakes in Digital Business Transformation

The organizational perspective is gaining relevance and places new requirements on future data lakes. Many large traditional firms, often incumbent market leaders in their (non-IT) business, pursue a digital transformation strategy in order to be able to compete with purely digital newcomers more effectively. In this context, the classical model of using internal IT departments or external software houses via service contracts is increasingly abolished in favor of integrative product-oriented teams in which developers, domain experts, sales and purchase representatives closely work together in an agile manner within a longer-term business roadmap and architecture. Moreover, the digitization process involves frequent mergers, acquisitions, and re-organizations of the business. In many cases, traditional data and systems integration methods, together with different organizational philosophies, have led to the failure of such processes, in some cases costing billions of dollars.

Executives have begun to perceive the idea of data lakes as a design pattern to deal with this organizational volatility. In this pattern, the data lake (at its core a store of mildly cleaned raw data) serves as a mediating element between the evolving set of internal and external transaction and monitoring streams, and the equally evolving set of business analytics and decision support tasks of the above-mentioned teams. In the ideal case, integrating a new company would simply mean adding their raw data to the lake, and using methods discussed in our survey to link them up to the existing data lake content. Existing analytics solutions could automatically include the new data, and new business teams could use analytics toolkits plus specialist expertise for their current challenges, without waiting for resources in the IT department. In this way, executives are trying to convert the IT provisioning from a cost and reimbursement factor to a continuously value-creating investment [102].

Achieving such a setting can be mission-critical for many organizations in traditional businesses. However, its realization is by no means trivial, both on the organizational and on the technical side. The following technical challenges concern both the input side and the analytics side of the data lake pattern.

First, nowadays, data is largely consumed by machine learning and data science applications everywhere. However, existing data lakes lack *matured* functions to meet such a requirement. Second, the lack of traditional analytical data management such as transaction management, indexing, and caching, makes data lakes less adequate for complex analytical workloads in the industry. Tackling these limitations, we discuss the exciting new challenges next.

[29]https://prestodb.io/

B. Data Lakes Meet Machine Learning

Recent advances of *DBML* [93], [128], [153], i.e., in-database machine learning or applying machine learning for data management, mainly consider a relational database instead of data lakes. Below we focus on the *lake-specific* challenges.

Training Data Heterogeneity. The systems and methods covered in this survey mainly support tabular data, JSON/XML, graphs, and texts. However, ML training data may also include other common types such as images, audio, and videos. The challenge of data multi-modality is non-trivial, and stretches beyond simply utilizing technologies such as multimedia databases [139] and polystores. The key question is how to design abstractions for heterogeneous data in data lakes. With the rapid development of ML models, e.g., BERT [33], GPT-3 [17], possible abstractions for multi-modal data are embeddings [30]. Such new options invoke more challenges. How to design the data abstraction to represent and connect multi-modal training data? How to design the data representation for a specific function in Table I, e.g., related dataset discovery, data integration? Moreover, ML models , in particular, deep neural networks require intensive tensor-based computations such as matrix multiplication. With the recent advances in hardware, e.g., Tensor Processing Unit (TPU), and tensor runtimes such as ONNX[30], it is an interesting direction to explore tensor-based intermediate representations (IRs) of data lakes, w.r.t. both data management and machine learning operations. It leads to more questions, e.g., how to redesign data lake architectures based on such new intermediate representation possibilities?

In-Lake Machine Learning. Following the research line of in-database machine learning [128], one of the most exciting challenges is to support machine learning training and inference in data lakes. First, the existing in-database machine learning studies mainly focus on structured data, i.e., relational tables. It is a rich area regarding how to extend these in-DB ML problems (e.g., factorization through joins [24], [129]) over heterogeneous, schema-less data in a data lake. Second, new APIs and systems are needed to connect existing data lakes with ML platforms such as MLflow[31], Amazon SageMaker[32], AzureML[33], or *model zoos* (repositories of pre-trained models) such as HuggingFace[34], Tensorflow Hub[35], and PyTorch Hub[36]. A more ambitious design alternative is a tighter integration of data lakes and machine learning, i.e., *ML-aware data lakes*, which are built upon the requirements of downstream ML applications, and bring more optimization opportunities. We need new data lakes providing functions such as preparing, labeling, and cleaning the raw, heterogeneous data for downstream ML applications. To build ML-aware data lakes, it also calls for novel data lake architecture, storage, and function redesign. These considerations lead to the following research questions. How to discover related datasets to augment the existing training dataset and improve ML model accuracy and fairness? How to effectively clean the raw, heterogeneous datasets in data lakes

[30]https://onnxruntime.ai/
[31]https://mlflow.org/
[32]https://aws.amazon.com/sagemaker/
[33]https://azure.microsoft.com/en-us/products/machine-learning
[34]https://huggingface.co/
[35]https://www.tensorflow.org/
[36]https://pytorch.org/hub/

to improve the effectiveness of ML models? How to combine and optimize the whole pipeline of data management and ML life cycle in data lakes?

ML Workflow Optimization. Towards designing ML-aware data lakes, one of the main goals is to improve ML models in terms of effectiveness (e.g., model accuracy) and efficiency (e.g., training time). Besides the functional and system-level redesign, another possibility is to utilize the metadata. One of the most intensively studied DBML problems is optimizing ML workflows and programs over relational databases; surveys like [93], [153] have elaborated on such studies. For instance, by utilizing the primary key-foreign key relationships and join dependencies [24], or functional dependencies [81], the runtime of model training can be significantly reduced. However, one of the key differences between data lakes from databases is the lack of metadata. Instead of predefined join dependencies or FDs, in a data lake we might need to discover the joinability between datasets (SectionVI-B1) or relaxed functional dependencies (Section VI-D2), which are probabilistic. Thus, one of the open challenges of optimizing ML operations in data lakes, is to utilize such fuzzy, discovered metadata about data.

ML-Driven Metadata Management. Besides the metadata of data, we need to cover also the metadata of ML models. The life cycle of an ML model contains multiple steps, including model training, hyperparameter tuning, debugging, deployment, etc. Accordingly, we need new metadata extraction, modeling, and enrichment methods for the relevant metadata about the ML life circle and the datasets involved in each step, which also calls for new data provenance methods.

C. Advanced Analytics and Transaction Management

Another future direction is to bring well-studied database and data warehouse functions, such as transaction management and query optimization, into data lakes for business intelligence. Towards this direction, the new paradigm of *Lakehouse* [6], [7], [70], [76] has emerged. Earlier, a common industrial practice was to apply data lakes (e.g., Amazon S3, GCP) as a cheap storage of large-scale raw data, before the datasets are selected and transformed for data warehouses (e.g., Snowflake, BigQuery). The overhead and complexity of maintaining two systems, a data lake and a data warehouse, have led to Lakehouses, e.g., Delta lake [6], Apache Hudi[37], and Apache Iceberg[38]. A Lakehouse inherits data lakes' role for storing large-scale raw data, i.e., supporting open formats such as Parquet and ORC over cloud storage, and data warehouses' analytics capabilities, e.g., transaction management, indexing, caching, and metadata management [6].

Following the path of Lakehouse development, many challenges emerge regarding transaction management, storage, indexing, metadata management, and machine learning. How to design cloud-native storage for read-write workloads with low-latency transaction guarantees? How to design auxiliary structures such as indexes over open data formats for efficient query processing? Moreover, recent Lakehouses provide open interfaces for ML workloads to query the data [76], or tensor-based IR for deep learning models [70]. A deeper integration between Lakehouses and ML, similar to the discussion in Section VIII-B, will bring more optimization opportunities, which calls for more research effort.

[37]https://hudi.apache.org/

[38]https://iceberg.apache.org/

IX. Conclusion and Outlook

In the first decade of their existence, data lakes have been receiving increasing interest from both academia and industry. In this survey, we have looked back at the origin and development of data lakes in the past decade. Besides offering a fine-grained data lake architecture and discussing storage systems, we have provided a comprehensive review of existing data lake methods based on their specific functions. We have used a three-level categorization, which facilitates a deep analysis of the corresponding research questions. To bring forth new challenges, we have also discussed potential technologies and future directions.

Without any doubt, the research, engineering, and application challenges are there, waiting for novel data lakes to be developed together with cutting-edge technologies of machine learning and cloud computing. Some well-studied research problems (e.g., data integration, data cleaning, schema evolution) need new perspectives and methods to address the specific characteristics of data lakes. The concept of data lakes is complex and still evolving, not limited to the problems addressed in this survey. We foresee the explosive development of data lake applications in the coming years. The golden age of data lakes is yet to come.

References

[1] R. Potharaju et al., "Hyperspace: The indexing subsystem of azure synapse," *Int. J. Very Large Data Bases*, vol. 14, no. 12, pp. 3043–3055, Jul. 2021.

[2] D. Abadi et al., "The beckman report on database research," *Commun. ACM*, vol. 59, no. 2, pp. 92–99, 2016.

[3] A. Alserafi, A. Abelló, O. Romero, and T. Calders, "Keeping the data lake in form: DS-KNN datasets categorization using proximity mining," in *Proc. Int. Conf. Model Data Eng.*, 2019, pp. 35–49.

[4] A. Alserafi, A. Abelló, O. Romero, and T. Calders, "Keeping the data lake in form: Proximity mining for pre-filtering schema matching," *ACM Trans. Inf. Syst.*, vol. 38, no. 3, pp. 26:1–26:30, 2020.

[5] A. Alserafi, T. Calders, A. Abelló, and O. Romero, "DS-Prox: Dataset proximity mining for governing the data lake," in *Proc. Similarity Search Appl.: 10th Int. Conf.*, 2017, pp. 284–299.

[6] M. Armbrust et al., "Delta lake: High-performance acid table storage over cloud object stores," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3411–3424, 2020.

[7] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia, "Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics," in *Proc. 11th Conf. Innov. Data Syst. Res.*, 2021.

[8] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: Self-tuning indexes for similarity search," in *Proc. 24th Int. Conf. World Wide Web*, 2005, pp. 651–660.

[9] A. Beheshti, B. Benatallah, R. Nouri, V. M. Chhieng, H. Xiong, and X. Zhao, "CoreDB: A data lake service," in *Proc. Conf. Inf. Knowl. Manage.*, 2017, pp. 2451–2454.

[10] A. Beheshti, B. Benatallah, R. Nouri, and A. Tabebordbar, "CoreKG: A knowledge lake service," *Int. J. Very Large Data Bases*, vol. 11, no. 12, pp. 1942–1945, 2018.

[11] S.-M.-R. Beheshti, H. R. Motahari-Nezhad, and B. Benatallah, "Temporal provenance model (TPM): Model and query language," 2010, *arXiv:1211.5009*.

[12] S. Bharadwaj, P. Gupta, R. Bhagwan, and S. Guha, "Discovering related data at scale," *Int. J. Very Large Data Bases*, vol. 14, no. 8, pp. 1392–1400, Apr. 2021.

[13] E. Boci and S. Thistlethwaite, "A novel big data architecture in support of ADS-B data analytic," in *Proc. IEEE Int. Conf. Neutron Scattering*, 2015, pp. C11–C18.

[14] A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou, "Dataset discovery in data lakes," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 709–720.

[15] W. Brackenbury et al., "Draining the data swamp: A similarity-based approach," in *Proc. Workshop Human-in-the-Loop Data Analytics*, 2018, pp. 13:1–13:7.

[16] D. G. Brizan and A. U. Tansel, "A survey of entity resolution and record linkage methodologies," *Commun. IIMA*, vol. 6, no. 3, 2006, Art. no. 5.

[17] T. Brown et al., "Language models are few-shot learners," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1877–1901.

[18] M. J. Cafarella, A. Halevy, and N. Khoussainova, "Data integration for the relational web," *Int. J. Very Large Data Bases*, vol. 2, no. 1, pp. 1090–1101, 2009.

[19] Capgemini SE and Software Pivotal, "The technology of the business data lake table," 2023. https://www.capgemini.com/wp-content/uploads/2017/07/pivotal-business-data-lake-technical_brochure_web.pdf

[20] B. B. Cardoso et al., "Data lake architecture for distribution system operator," in *Proc. IEEE Power Energy Soc. Innov. Smart Grid Technol. Conf.*, 2021, pp. 1–5.

[21] L. Caruccio, V. Deufemia, and G. Polese, "Relaxed functional dependencies - A survey of approaches," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 1, pp. 147–165, Jan. 2016.

[22] F. Chang et al., "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, 2008.

[23] H. Che and Y. Duan, "On the logical design of a prototypical data lake system for biological resources," *Front. Bioeng. Biotechnol.*, vol. 8, 2020, Art. no. 1105.

[24] L. Chen, A. Kumar, J. Naughton, and J. M. Patel, "Towards linear algebra over normalized data," *Int. J. Very Large Data Bases*, vol. 10, no. 11, pp. 1214–1225, 2017.

[25] M. Cherradi and A. EL Haddadi, "Data lakes: A survey paper," in *Proc. Int. Conf. Smart City Appl.*, 2021, pp. 823–835.

[26] M. Chessell, F. Scheepers, N. Nguyen, R. van Kessel, and R. van der Starre, "Governing and managing big data for analytics and decision makers," IBM Redguides Business Leaders, Aug. 26, 2014. [Online]. Available: https://www.redbooks.ibm.com/redpapers/pdfs/redp5120.pdf

[27] W. J. Conover, *Practical Nonparametric Statistics*, vol. 350. Hoboken, NJ, USA: Wiley, 1998.

[28] J. C. Corbett et al., "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, pp. 8:1–8:22, 2013.

[29] J. Couto, O. T. Borges, D. D. Ruiz, S. Marczak, and R. Prikladnicki, "A mapping study about data lakes: An improved definition and possible architectures," in *Proc. 31st Int. Conf. Softw. Eng. Knowl. Eng.*, 2019, pp. 453–578.

[30] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, May 2019.

[31] C. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo, "Automating the database schema evolution process," *Proc. Int. J. Very Large Data Bases*, vol. 22, no. 1, pp. 73–98, 2013.

[32] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "TURL: Table understanding through representation learning," *ACM SIGMOD Rec.*, vol. 51, no. 1, pp. 33–40, 2022.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics Hum. Lang. Technol.*, 2019, pp. 4171–4186.

[34] C. Diamantini, P. L. Giudice, L. Musarella, D. Potena, E. Storti, and D. Ursino, "A new metadata model to uniformly handle heterogeneous data lake sources," in *Proc. Eur. Conf. Adv. Databases Inf. Syst.*, 2018, pp. 165–177.

[35] C. Diamantini, P. L. Giudice, L. Musarella, D. Potena, E. Storti, and D. Ursino, "An approach to extracting thematic views from highly heterogeneous sources of a data lake," in *Proc. 26th Italian Symp. Adv. Database Syst.*, 2018.

[36] C. Diamantini, P. L. Giudice, D. Potena, E. Storti, and D. Ursino, "An approach to extracting topic-guided views from the sources of a data lake," *Inf. Syst. Front.*, vol. 23, pp. 243–262, 2021.

[37] J. Dixon, "Pentaho, hadoop, and data lakes | james dixon's blog," 2010. [Online]. Available: https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/

[38] J. Dixon, Data lakes revisited, 2014. [Online]. Available: https://jamesdixon.wordpress.com/2014/09/25/data-lakes-revisited/

[39] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*. New York, NY, USA: Elsevier, 2012.

[40] Y. Dong, K. Takeoka, C. Xiao, and M. Oyamada, "Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 456–467.

[41] J. Duggan et al., "The BigDAWG polystore system," *SIGMOD Rec.*, vol. 44, no. 2, pp. 11–16, 2015.

[42] J. Eder and V. A. Shekhovtsov, "Data quality for federated medical data lakes," *Int. J. Web Inf. Syst.*, vol. 17, no. 5, pp. 407–426, 2021.

[43] R. Eichler, C. Giebler, C. Gröger, H. Schwarz, and B. Mitschang, "HANDLE - A generic metadata model for data lakes," in *Big Data Analytics and Knowledge Discovery*. Berlin, Germany: Springer, 2020, pp. 73–88.

[44] K. M. Endris, P. D. Rohde, M.-E. Vidal, and S. Auer, "Ontario: Federated query processing against a semantic data lake," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2019, pp. 379–395.

[45] R. Fagin, L. M. Haas, M. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis, *Clio: Schema Mapping Creation and Data Exchange*. Berlin, Germany: Springer, 2009, pp. 198–236.

[46] H. Fang, "Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem," in *Proc. IEEE Int. Conf. Cyber Technol. Automat. Control Intell. Syst.*, 2015, pp. 820–824.

[47] M. Farid, A. Roatis, I. F. Ilyas, H.-F. Hoffmann, and X. Chu, "CLAMS: Bringing quality to data lakes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 2089–2092.

[48] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A data discovery system," in *Proc. IEEE Int. Conf. Data Eng.*, 2018, pp. 1001–1012.

[49] R. C. Fernandez, J. Min, D. Nava, and S. Madden, "Lazo: A cardinality-based method for coupled estimation of jaccard similarity and containment," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1190–1201.

[50] Y. Fu and C. Soman, "Real-time data infrastructure at uber," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2021, pp. 2503–2516.

[51] T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. W. Paton, "Data wrangling for Big Data: Challenges and opportunities," in *Proc. Int. Conf. Extending Database Technol.*, 2016, pp. 473–478.

[52] C. A. Galindo-Legaria, "Outerjoins as disjunctions," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1994, pp. 348–358.

[53] Y. Gao, S. Huang, and A. Parameswaran, "Navigating the data lake with DATAMARAN," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2018, pp. 943–958.

[54] I. Gartner, "Gartner says beware of the data lake fallacy," 2014. https://www.gartner.com/newsroom/id/2809117

[55] S. Geisler, C. Quix, R. Hai, and S. Alekh, "An integrated ontology-based approach for patent classification in medical engineering," in *Proc. Int. Conf. Data Integration Life Sci.*, 2017, pp. 38–52.

[56] C. Giebler, C. Gröger, E. Hoos, R. Eichler, H. Schwarz, and B. Mitschang, "The data lake architecture framework," in *Proc. BTW2021- Datenbanksysteme für Bus. Technol. Web., Gesellschaft für Informatik*, Bonn, Germany, 2021, pp. 351–370, doi: 10.18420/btw2021-19.

[57] C. Giebler, C. Gröger, E. Hoos, and B. Mitschang, "Modeling data lakes with data vault: Practical experiences, assessment, and lessons learned," in *Proc. Int. Conf. Concept. Model.*, 2019, pp. 63–77.

[58] C. Giebler, C. Gröger, E. Hoos, H. Schwarz, and B. Mitschang, "Leveraging the data lake - current state and challenges," in *Proc. Int. Conf. Big Data Analytics Knowl. Discov.*, 2019, pp. 179–188.

[59] A. Gorelik, *The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[60] M. Grover, T. Malaska, J. Seidman, and G. Shapira, *Hadoop Application Architectures: Designing Real-World Big Data Applications*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.

[61] R. Hai, S. Geisler, and C. Quix, "Constance: An intelligent data lake system," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 2097–2100.

[62] R. Hai and C. Quix, "Rewriting of plain SO tgds into nested tgds," *Int. J. Very Large Data Bases*, vol. 12, no. 11, pp. 1526–1538, 2019.

[63] R. Hai, C. Quix, and D. Kensche, "Nested schema mappings for integrating JSON," in *Proc. Int. Conf. Conceptual Model.*, 2018, pp. 397–405.

[64] R. Hai, C. Quix, and D. Wang, "Relaxed functional dependency discovery in heterogeneous data lakes," in *Proc. Int. Conf. Conceptual Model.*, 2019, pp. 225–239.

[65] R. Hai, C. Quix, and C. Zhou, "Query rewriting for heterogeneous data lakes," in *Proc. Eur. Conf. Adv. Databases Inf. Syst.*, 2018, pp. 35–49.

[66] A. Y. Halevy, "Answering queries using views: A survey," *VLDB J.*, vol. 10, no. 4, pp. 270–294, Dec. 2001.

[67] A. Y. Halevy et al., "Goods: Organizing Google's datasets," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 795–806.

[68] A. Y. Halevy et al., "Managing Google's data lake: An overview of the goods system," *IEEE Data Eng. Bull.*, vol. 39, no. 3, pp. 5–14, Mar. 2016.

[69] D. Halverson et al., "Split query processing in polybase," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 1255–1266.

[70] S. Hambardzumyan et al., "Deep lake: A lakehouse for deep learning," in *Proc. Conf. Innov. Data Syst. Res.*, 2023.

[71] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2/3, pp. 146–162, 1954.

[72] A. Helal, M. Helali, K. Ammar, and E. Mansour, "A demonstration of KGLac: A data discovery and enrichment platform for data science," *Int. J. Very Large Data Bases*, vol. 14, no. 12, pp. 2675–2678, Jul. 2021.

[73] T. S. Hukkeri, V. Kanoria, and J. Shetty, "A study of enterprise data lake solutions," *Int. Res. J. Eng. Technol.*, vol. 7, pp. 1924–1929, 2020.

[74] B. Inmon, *Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump*. Basking Ridge, NJ, USA: Technics Publications, 2016.

[75] Z. Ives, Y. Zhang, S. Han, and N. Zheng, "Dataset relationship management," in *Proc. Conf. Innov. Data Syst. Res.*, 2019.

[76] P. Jain, P. Kraft, C. Power, T. Das, I. Stoica, and M. Zaharia, "Analyzing and comparing lakehouse storage systems," in *Proc. Conf. Innov. Data Syst. Res.* 2023.

[77] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis, *Foundations of Data Warehouses*, 2nd ed., Berlin, Germany: Springer, 2003.

[78] M. Jarke and C. Quix, "On warehouses, lakes, and spaces: The changing role of conceptual modeling for data integration," in *Conceptual Modeling Perspectives*. Berlin, Germany: Springer, 2017, pp. 231–245.

[79] A. Joulin, E. Grave, and P. B. T. Mikolov, "Bag of tricks for efficient text classification," in *Proc. 13th Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2017, Art. no. 427.

[80] K. M. Endris, "Federated query processing over heterogeneous data sources in a semantic data lake," PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, May 2020.

[81] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich, "Learning models over relational data using sparse tensors and functional dependencies," *ACM Trans. Database Syst.*, vol. 45, no. 2, pp. 1–66, 2020.

[82] A. Khatiwada, R. Shraga, W. Gatterbauer, and R. J. Miller, "Integrating data lake tables," *Int. J. Very Large Data Bases*, vol. 16, no. 4, pp. 932–945, 2022.

[83] M. Klettke, H. Awolin, U. Störl, D. Müller, and S. Scherzinger, "Uncovering the evolution history of data lakes," in *Proc. IEEE Int. Conf. Big Data*, 2017, pp. 2462–2471.

[84] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," *Int. J. Very Large Data Bases*, vol. 10, no. 11, pp. 1502–1513, 2017.

[85] A. Leventidis, L. D. Rocco, R. J. Miller, M. Riedewald, and W. Gatterbauer, "DomainNet: Homograph detection for data lake disambiguation," in *Proc. Int. Conf. Extending Database Technol.*, 2021, pp. 13–24.

[86] D. Lindstedt and K. Graziano, *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. Scotts Valley, CA, US: CreateSpace, 2011.

[87] D. Lindstedt and M. Olschimke, *Building a Scalable Data Warehouse With Data Vault 2.0*. San Mateo, CA, USA: Morgan Kaufmann, 2015.

[88] J. Lu and I. Holubová, "Multi-model databases: A new journey to handle the variety of data," *ACM Comput. Surveys*, vol. 52, no. 3, pp. 1–38, 2019.

[89] B. MacCartney, M. Galley, and C. D. Manning, "A phrase-based alignment model for natural language inference," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2008, pp. 802–811.

[90] A. Maccioni and R. Torlone, "Crossing the finish line faster when paddling the data lake with KAYAK," *Int. J. Very Large Data Bases*, vol. 10, no. 12, pp. 1853–1856, 2017.

[91] A. Maccioni and R. Torlone, "KAYAK: A framework for just-in-time data preparation in a data lake," in *CAiSE*. Berlin, Germany: Springer, 2018, pp. 474–489.

[92] C. Madera and A. Laurent, "The next information architecture evolution: The data lake wave," in *Proc. 8th Int. Conf. Manage. Digit. Eco Syst.*, 2016, pp. 174–180.

[93] N. Makrynioti and V. Vassalos, "Declarative data analytics: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 6, pp. 2392–2411, Jun. 2021.

[94] M. N. Mami, D. Graux, S. Scerri, H. Jabeen, and S. Auer, "Querying data lakes using spark and presto," in *Proc. 24th Int. Conf. World Wide Web*, 2019, pp. 3574–3578.

[95] C. D. Manning, *An Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[96] M. A. Martínez-Prieto, A. Bregon, I. García-Miranda, P. C. Álvarez-Esteban, F. Díaz, and D. Scarlatti, "Integrating flight-related information into a (Big) data lake," in *Proc. IEEE/AIAA 36th Digit. Avionics Syst. Conf.*, 2017, pp. 1–10.

[97] R. Marty, "The security data lake," *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud Workshops*, 2015, pp. 148–153.

[98] C. Mathis, "Data Lakes," *Datenbank-Spektrum*, vol. 17, no. 3, pp. 289–293, 2017.

[99] H. Mehmood et al., "Implementing Big Data lake for heterogeneous data sources," in *Proc. Int. Conf. Data Eng. Workshops*, 2019, pp. 37–44.

[100] R. J. Miller, "Open data integration," *Int. J. Very Large Data Bases*, vol. 11, no. 12, pp. 2130–2139, 2018.

[101] N. Miloslavskaya and A. Tolstoy, "Big data, fast data and data lake concepts," *Procedia Comput. Sci.*, vol. 88, pp. 300–305, 2016.

[102] M. Mueller-Wuensch, A. Fogelgren, J. Peppard, and R. Winter, "The real questions of enterprise transformation," *Panel, ICIS*, 2022. [Online]. Available: https://icis2022.aisconferences.org/cio-forum/

[103] A. A. Munshi and Y. A.-R. I. Mohamed, "Data lake lambda architecture for smart grids Big Data analytics," *IEEE Access*, vol. 6, pp. 40463–40471, 2018.

[104] F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost, and R. J. Miller, "Organizing data lakes for navigation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1939–1950.

[105] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: Challenges and opportunities," *Int. J. Very Large Data Bases*, vol. 12, no. 12, pp. 1986–1989, 2019.

[106] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *Int. J. Very Large Data Bases*, vol. 11, no. 7, pp. 813–825, 2018.

[107] I. D. Nogueira, M. Romdhane, and J. Darmont, "Modeling data lake metadata with a data vault," in *Proc. 22nd Int. Database Eng. Appl. Symp.*, 2018, pp. 253–261.

[108] D. E. O'Leary, "Embedding AI and crowdsourcing in the Big Data lake," *IEEE Intell. Syst.*, vol. 29, no. 5, pp. 70–73, Sep./Oct. 2014.

[109] M. Ota, H. Müller, J. Freire, and D. Srivastava, "Data-driven domain discovery for structured datasets," *Int. J. Very Large Data Bases*, vol. 13, no. 7, pp. 953–967, Mar. 2020.

[110] P. Ouellette et al., "RONIN: Data lake exploration," *Int. J. Very Large Data Bases*, vol. 14, no. 12, pp. 2863–2866, Jul. 2021.

[111] P. Patel, G. Wood, and A. Diaz, "Data lake governance best practices," *DZone Guide Big Data - Data Sci. Adv. Analytics*, vol. 4, pp. 6–7, 2017.

[112] J. Pennekamp et al., "Towards an infrastructure enabling the internet of production," in *Proc. IEEE Int. Conf. Ind. Cyber Phys. Syst.*, 2019, pp. 31–37.

[113] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *Int. J. Very Large Data Bases*, vol. 5, no. 10, pp. 908–919, 2012.

[114] C. Quix, S. Geisler, R. Hai, and S. Alekh, "Ontology matching for patent classification," in *Proc. 13th Int. Workshop Ontology Matching Co-Located 17th Int. Semantic Web Conf.*, 2017, pp. 37–48.

[115] C. Quix and R. Hai, *Data Lake*. Berlin, Germany: Springer, 2018, pp. 1–8.

[116] C. Quix, R. Hai, and I. Vatov, "GEMMS: A generic and extensible metadata management system for data lakes," in *Proc. 28th Int. Conf. Adv. Inf. Syst. Eng.*, 2016, pp. 129–136.

[117] C. Quix, R. Hai, and I. Vatov, "Metadata extraction and management in data lakes with GEMMS," *Complex Syst. Inf. Model. Quart.*, vol. 9, pp. 67–83, 2016.

[118] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[119] E. Rahm and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, 2000.

[120] R. Ramakrishnan et al., "Azure data lake store: A hyperscale distributed file service for big data analytics," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2017, pp. 51–63.

[121] M. Ramirez, A. Bogatu, N. W. Paton, and A. Freitas, "Natural language inference over tables: Enabling explainable data exploration on data lakes," in *Proc. Eur. Semantic Web Conf.*, 2021, pp. 304–320.

[122] F. Ravat and Y. Zhao, "Data lakes: Trends and perspectives," in *Database and Expert Systems Applications*. Berlin, Germany: Springer, 2019, pp. 304–313.

[123] E. K. Rezig et al., "DICE: Data discovery by example," *Int. J. Very Large Data Bases*, vol. 14, no. 12, pp. 2819–2822, Jul. 2021.

[124] P. D. Rohde and M.-E. Vidal, "Optimizing federated queries based on the physical design of a data lake," in *Proc. Workshops EDBT/ICDT Joint Conf.*, 2020.

[125] P. Russom, "Data lakes: Purposes, practices, patterns, and platforms," *TDWI*, White Paper 2017. [Online]. Available: https://info.talend.com/rs/talend/images/WP_EN_BD_TDWI_DataLakes.pdf

[126] P. Sawadogo and J. Darmont, "On data lake architectures and metadata management," *J. Intell. Inf. Syst.*, vol. 56, pp. 1–24, 2020.

[127] P. N. Sawadogo, É. Scholly, C. Favre, É. Ferey, S. Loudcher, and J. Darmont, "Metadata systems for data lakes: Models and features," in *BBIGAP@ADBIS*, Berlin, Germany: Springer, 2019, pp. 440–451.
[128] M. Schleich, D. Olteanu, M. Abo-Khamis, H. Q. Ngo, and X. Nguyen, "Learning models over relational data: A brief tutorial," in *Proc. Int. Conf. Scalable Uncertainty Manage.*, 2019, pp. 423–432.
[129] M. Schleich, D. Olteanu, and R. Ciucanu, "Learning linear regression models over factorized joins," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2016, pp. 3–18.
[130] C. Sengstock and C. Mathis, "SAP HANA Vora: A distributed computing platform for enterprise data lakes," in *Proc. Datenbanksysteme Bus. Technol. Web.*, 2017, pp. 521–522.
[131] B. Sharma and A. LaPlante, *Architecting Data Lakes*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2016.
[132] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10.
[133] Y. L. Simmhan et al., "A survey of data provenance techniques," *Comput. Sci. Dept. Indiana Univ. Bloomington*, vol. 69, 2005, Art. no. 47405.
[134] A. Singh, "Architecture of data lake," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 5, no. 2, pp. 411–414, 2019.
[135] M. Sipser, "Introduction to the theory of computation," vol. 2. Thomson Course Technology Boston, Boston, MA, USA, 2006.
[136] T. J. Skluzacek et al., "Skluma: An extensible metadata extraction pipeline for disorganized data," in *Proc. IEEE 14th Int. Conf. E-Sci.*, 2018, pp. 256–266.
[137] J. Song and Y. He, "Auto-validate: Unsupervised data validation using data-domain patterns inferred from data lakes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2021, pp. 1678–1691.
[138] B. Stein and A. Morrison, "The enterprise data lake: Better integration and deeper analytics," *PwC Technol. Forecast: Rethinking Integration*, vol. 1, no. 1/9, 2014, Art. no. 18.
[139] V. Subrahmanian and S. Jajodia, "Multimedia database systems: Issues and research directions," 2012.
[140] I. Suriarachchi and B. Plale, "Crossing analytics systems: A case for integrated provenance in data lakes," in *Proc. IEEE 12th Int. Conf. E-Sci.*, 2016, pp. 349–354.
[141] R. Tan, R. Chirkova, V. Gadepally, and T. G. Mattson, "Enabling query processing across heterogeneous data models: A survey," in *Proc. IEEE Int. Conf. Big Data*, 2017, pp. 3211–3220.
[142] I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino, "Data wrangling: The challenging yourney from the wild to the lake," in *Proc. Conf. Innov. Data Syst. Res.*, 2015.
[143] C. Walker and H. H. Alrehamy, "Personal data lake with data gravity pull," in *Proc. IEEE 5th Int. Conf. Big Data Cloud Comput.*, 2015, pp. 160–167.
[144] G. Weintraub, E. Gudes, and S. Dolev, "Needle in a haystack queries in cloud data lakes," in *Proc. EDBT/ICDT Workshops*, 2021.
[145] M. Wibowo, S. Sulaiman, and S. M. Shamsuddin, "Machine learning in data lake for combining data silos," in *Proc. Int. Conf. Data Mining Big Data*, 2017, pp. 294–306.
[146] M. E. A. Wilkonson, "Commentary: The FAIR guiding principles for scientific data management and stewardship," *Nature Sci. Data*, vol. 3, 2016, Art. no. 160018.
[147] E. Zagan and M. Danubianu, "Data lake approaches: A survey," in *Proc. IEEE Int. Conf. Develop. Appl. Syst.*, 2020, pp. 189–193.
[148] E. Zagan and M. Danubianu, "Cloud DATA LAKE: The new trend of data storage," in *Proc. IEEE 3rd Int. Congr. Human-Comput. Interact. Optim. Robot. Appl.*, 2021, pp. 1–4.
[149] S. Zhang and K. Balog, "Web table extraction, retrieval, and augmentation: A survey," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 2, pp. 1–35, 2020.
[150] Y. Zhang and Z. G. Ives, "Juneau: Data lake management for jupyter," *Int. J. Very Large Data Bases*, vol. 12, no. 12, pp. 1902–1905, 2019.
[151] Y. Zhang and Z. G. Ives, "Finding related tables in data lakes for interactive data science," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2020, pp. 1951–1966.
[152] J. K. Zhi Kang, G. S. Y. Tan, F. Cheng, S. Sun, and B. He, "Efficient deep learning pipelines for accurate cost estimations over large scale query workload," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2021, pp. 1014–1022.
[153] X. Zhou, C. Chai, G. Li, and J. Sun, "Database meets artificial intelligence: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 3, pp. 1096–1116, Mar. 2022.
[154] E. Zhu, D. Deng, F. Nargesian, and R. J. Miller, "JOSIE: Overlap set similarity search for finding joinable tables in data lakes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2019, pp. 847–864.
[155] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller, "LSH ensemble: Internet-scale domain search," *Int. J. Very Large Data Bases*, vol. 9, no. 12, pp. 1185–1196, 2016.
[156] P. Zikopoulos, D. DeRoos, C. Bienko, R. Buglio, and M. Andrews, *Big Data Beyond the Hype: A Guide to Conversations for Todays Data Center*. New York, NY, USA: McGraw-Hill, 2014.

Rihan Hai (Member, IEEE) received the PhD degree from RWTH Aachen University, Germany. She is an assistant professor in the Web Information Systems group, Delft University of Technology, Netherlands. Her research focuses on data lakes, data integration, and data management for machine learning. She has served as a program committee member of database conferences such as VLDB, ICDE and EDBT, and a journal reviewer for TKDE, SIGMOD Record, JMLR and TPDS.

Christos Koutras received the five-year diploma in electrical and computer engineering from NTUA, Greece, and the MPhil degree from the Department of Computer Science and Engineering, HKUST, Hong Kong. He is currently working toward the PhD degree in the Department of Software Technology, Delft University of Technology, Netherlands. His research interests include data integration, schema matching and related dataset discovery.

Christoph Quix is professor for Information Systems and Data Science with the Niederrhein University. Previously, he held a deputy professorship for data science, RWTH Aachen University. His research focuses on data integration, data science, management of large, heterogeneous data sets, and metadata management. His research is application-oriented, e.g., in chemistry 4.0 or industry 4.0. He has more than 100 publications in international journals and conferences.

Matthias Jarke (Senior Member, IEEE) is a professor Emeritus of Databases and Information Systems, RWTH Aachen University, Germany, and past director of the Fraunhofer FIT Institute for Applied IT. His research is focused on information systems support for cooperative tasks in engineering, business, and culture. Major contributions include query optimization, conceptual modeling, and requirements management. Matthias has served as chief editor of Information Systems, and on the editorial board of numerous journals including IEEE TSE, as well as program chair of prestigious database conferences such as VLDB, EDBT, CAiSE, and SSDBM. He is lifetime ACM Fellow, GI Fellow, recipient of the 2020 Peter Chen Award, and member of Germany's acatech National Academy of Science and Engineering.