Noname manuscript No.
(will be inserted by the editor)

Data lake concept and systems: a survey

Rihan Hai · Christoph Quix · Matthias Jarke

Received: date / Accepted: date

Abstract Although big data has been discussed for some years, it still has many research challenges, especially the variety of data. It poses a huge difficulty to efficiently integrate, access, and query the large volume of diverse data in information silos with the traditional 'schema-on-write' approaches such as data warehouses. *Data lakes* have been proposed as a solution to this problem. They are repositories storing raw data in its original formats and providing a common access interface. This survey reviews the development, definition, and architectures of data lakes. We provide a comprehensive overview of research questions for designing and building data lakes. We classify the existing data lake systems based on their provided functions, which makes this survey a useful technical reference for designing, implementing and applying data lakes. We hope that the thorough comparison of existing solutions and the discussion of open research challenges in this survey would motivate the future development of data lake research and practice.

Keywords Data lake · Metadata management · Data discovery

Rihan Hai
Delft University of Technology, Delft, Netherlands
E-mail: R.Hai@tudelft.nl

Christoph Quix
Hochschule Niederrhein & Fraunhofer FIT, Krefeld, Germany
E-mail: christoph.quix@hs-niederrhein.de

Matthias Jarke
RWTH Aachen University & Fraunhofer FIT, Aachen, Germany
E-mail: jarke@dbis.rwth-aachen.de

1 Introduction

Big data has undoubtedly become one of the most important challenges in database research. Unprecedented volume, large variety, and high velocity of data need to be collected, stored, and processed with high data quality (veracity) to provide value.

Especially the variety of data still poses a daunting challenge with many open issues [1]. 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. Only the combination and integrated analysis of the information across these silos achieve the required valuable insight.

Traditional *schema-on-write* approaches such as the extract, transform, load (ETL) process are too 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 for a high amount 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 with higher-level languages (e.g., Pig[1] and Hive[2]), as well as NoSQL database (e.g., MongoDB[3] and Neo4j[4]), are gaining more popularity and implemented scenarios. 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 required from data management today.

[1] `https://pig.apache.org/`
[2] `https://hive.apache.org/`
[3] `https://www.mongodb.com/`
[4] `https://neo4j.com/`

To address this gap, *data lakes* (DLs) have been proposed as big data repositories, which store raw data and provide a rich list of functionalities with the help of metadata descriptions [125, 119, 124, 107]. Various solutions and systems are proposed to address many of the data lake challenges. However, as 'data lake' is a current buzzword, there is much hype about it, without exactly knowing what it is. Moreover, most recent data lake proposals only target a specific research problem or certain types of source data. As a relatively new concept, there has been merely a survey [112] discussing certain aspects of data lakes (i.e., architecture and metadata management). A coherent complete picture of DL problems and solutions is still missing.

Contributions. Our survey provides a thorough explanation of the DL concept and categorizes the existing data lake solutions. The survey also aims at helping to build or customize a data lake, and discover open questions and future research directions about data lakes. We summarize the contributions as follows.

- We review the almost ten-year development of the data lake concept and implementations. We provide the definition of data lakes and compare it with other similar concepts.
- We categorize the necessary functions in a data lake and propose a fine-grained architecture, which clarifies the workflow and function components for building a data lake system.
- We provide a general classification of existing data lake studies according to their provided functions. We analyze each class of research problems in depth and compare the data lake solutions.
- We illuminate a few new directions, applications, and potential technologies of data lakes.

Notably, different from a recent tutorial [95] that discusses challenges and potentially useful technologies for data lakes, this survey focuses on systems that claim to be a data lake, or provide partial functions of a data lake. Moreover, the comparison in this survey will mainly cover **implemented** systems that resolve certain research problems of data lakes, rather than high-level DL system proposals or commercial DL products. Refer to [110, 65] for a more industrial point of view.

Survey outline. The rest of the survey is divided into two parts. In the first part, we first discuss the origin (Sec. 2), definition (Sec. 3), and existing architectures (Sec. 4.1) of data lakes. We propose a data lake architecture and categorization criteria for the survey in Sec. 4.2-4.3. In the second part of the survey, we mainly compare existing data lake solutions with regard to data storage (Sec. 5), ingestion (Sec. 6), maintenance (Sec. 7), and exploration (Sec. 8). In Sec. 9 we review formal schema mapping and provenance aspects of metadata management. We discuss some new directions and applications of data lakes in Sec. 10 and conclude the survey in Sec. 11.

2 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.

2.1 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 [34]. 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, and 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 [21], 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 an *action tier* that links these insights with the existing applications; (3) additional tiers *monitor and manage* the data. The company also suggested using Hadoop as the storage system of the data lake, and applying their existing products for realizing the aforementioned tiers. However, w.r.t the actual implementation of a data lake, not many details were given.

2.2 2014–2015: Criticisms and further development

In 2014, *Gartner* raised several criticisms about data lakes [49]. When ingesting disparate data into a data lake, the data lake may easily turn into an unusable "data swamp", unless it requires metadata or data governance. In particular, after ingestion, the semantics or data quality of the raw data is unknown, and the origin (provenance) of individual datasets or its possible connection to other datasets is missing. It would be difficult for users to retrieve useful information or have any possible application of such a data swamp. In addition, Gartner pointed out that the existing data lake solutions did not provide a good answer to how oversight of data security and privacy should be conducted. Such crucial criticisms had a significant influence on many data lakes studies in the following years, which we will elaborate on in Sec. 6-7.

In response, Dixon revisited the concept [35], and emphasized that a data lake should also be equipped with metadata and governance, such that even with data in its raw form instead of transformed/aggregated form, a data lake could enable ad-hoc data analytics.

Meanwhile, more DL proposals started to emerge. They brought more requirements, solutions, and challenges. *PwC* has defined a data lake as a repository of structured, semi-structured, unstructured data in heterogeneous formats [119], originating from the business transactions, sensors, or mobile/cloud-based applications. With Hadoop in the center, 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. They postulate that a data lake actively extracts metadata from the raw data and stores it; then it discovers patterns about 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.

Meanwhile, *IBM* proposed more sophisticated information governance and management [26], and defined an enhanced DL solution, i.e., *data reservoir*, and refined it by the notion of *data wrangling* [124] for data governance issues such as license checking or visualization. A data reservoir groups the diverse input datasets by their types of usage. Each group of datasets is referred to as a *zone*. It maintains the catalog with descriptive details about data. Moreover, the IBM proposal allowed multiple manners of interaction between users and the stored raw data, e.g., discover datasets with the help of catalogs, and access the data with API-based service calls or user-defined decision models.

Other proposals addressed new possibilities such as Artificial Intelligence (AI) and Crowdsourcing to facilitate data integration, access, and quality improvement in data lakes [99]. For example, AI helps extract features of data, generate tags with descriptive metadata, find related datasets, discover possible structures from schema-less data, and reduce data redundancy. Crowdsourcing can help with collectively tagging semantic knowledge about the data, and linking the possible relationships among datasets.

With a special focus on security information and event management, how to store and access the data properly is discussed in [88]. The importance of *metadata management* is emphasized in [125] with an architecture to parse, store, and query heterogeneously structured personal data. There is also a proposal [40] emphasizing the importance of Human-in-the-loop, e.g., data scientists govern the data in data lakes.

2.3 2016–present: Prosperity and diversity

Since 2016 the realization of data lakes in both industry and academic community has been booming. There are proposals about data lake architectures [115, 66, 83, 116], concept, components and challenges [92, 89, 106].

Many IT companies offer commercial tools for building data lakes, e.g., *GOODS* [63] from Google, *Azure Data Lake* [109] from Microsoft, *AWS Lake*[5] from AMAZON, *Vora* from SAP [114], IBM and Cloudera[6], Oracle[7], and *Snowflake*[8]. In particular, *Delta Lake*[9] from Databricks is open-source and offers the storage layer compatible with Apache Spark[10] APIs.

Meanwhile, DL-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 [56], data quality [41], data provenance [120], metadata enrichment [10, 59], data preparation [82], dataset organization [6, 131], modeling [108, 98, 52], data integration [60, 57] and related dataset discovery [43, 16, 131, 14, 130]. Such data lake proposals targeting specific research challenges, are also our main focus in this survey, which will be addressed in Sec. 5-10.

3 Data lake definition

In this section, we first give a general definition of data lakes, then compare data lakes with similar concepts, i.e., data warehouses and dataspace systems. We also illustrate the application of data lakes with an example from a manufacturing process.

3.1 Definition

Based on the discussion in Sec. 2, we extract the most essential aspects of a general-purpose data lake, which are also agreed upon in most of the existing literature, and define the concept of data lakes as below.

[5] `https://aws.amazon.com/lake-formation/`
[6] `https://www.ibm.com/analytics/data-lake`
[7] `https://blogs.oracle.com/bigdata/data-lake-solution-patterns-use-cases`
[8] `https://www.snowflake.com/data-lake/`
[9] `https://delta.io/`
[10] `https://spark.apache.org/`

Definition 1 (Data Lake) 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 query processing and data analytics in an on-the-fly manner.

Regarding the definition, several remarks are in order.

Data lakes store raw data. A data lake ingests raw data in its original format from heterogeneous data sources, fulfills its role as a storage repository, and allows users to query and explore the data. The ingestion of raw data may lead to the absence of schema information, constraints, and mappings, which are not defined explicitly or required initially for a data lake. However, metadata management is crucial for data reasoning, query processing, and data quality management. Without any metadata, the data lake is hardly usable as the structure and semantics of the data are not known, which turns a data lake quickly into a 'data swamp'. Therefore, it is important to extract as much metadata as possible from the data sources during the ingestion phase. If sufficient metadata cannot be extracted from the sources automatically, a human expert has to provide additional information about the data source.

A data lake is not only a storage system. The primary role of a data lake is a data repository, which can be a centralized repository or a set of distributed repositories. However, a data lake is not only a storage system, e.g., a storage repository on top of a Hadoop file system. It needs to provide a set of functions to manage and govern the data such that it can be used later. Notably, in the existing literature the term "data lake" is used in both cases: solely the storage layer (i.e., Hadoop or certain databases), or a system providing storage and further services (e.g., metadata management). In this survey, we apply the latter meaning of a data lake (system).

Data lakes support on-demand data processing and querying. One important feature of data lakes is *on-the-fly*, or *on-demand*, which indicates that schema definition, data integration, or indexing should be done only if necessary at the time of data access. This might lead to more cost or effort for accessing the data, but it also increases the flexibility for posing dynamic, ad-hoc queries or applications of data. In data lakes, the ingestion of data sources could be light weight, as it is not necessary to force schema definitions and mappings beforehand. Moreover, the metadata in data lakes matures incrementally. For instance, the data marts can be defined when a user queries the data lake. Such information is stored to help with possible future queries, e.g., joined attributes.

3.2 Data lakes and data warehouses

A *data warehouse* (DW) manages, integrates, and aggregates data from multiple sources, and provides data analytics for decision making via multi-dimensional data cubes in data marts [67, 2]. The focus is on high data quality, usage of compact historical data, and interactive user support using pre-defined multi-dimensional user views or data marts [69]. Based on [31, 55] and the data lake definition given in Sec. 3.1, we summarize key differences between DW and data lakes in Table 1.

In data warehouses, heterogeneous source datasets need to first go through the Extract/transform/load (ETL) process. While enabling compact continuous embedding of new experiences with historical information, ETL is time-consuming and requires careful design, thus precluding real-time analytics. DW data is stored and handled as structured data in relational databases or cube structures. In contrast, in data lakes the transformation step is delayed and data is loaded in its original structure (i.e., load-as-is) to reduce upfront cleaning and integration effort and to make full source data available for later data analysis (i.e., pay-as-you-go). Data lakes aim to process more heterogeneous sources including semi-structured and unstructured sources, and to manage these different data models efficiently using multiple dedicated kinds of storage (cf. Sec. 5). To access the data, a DW usually applies (multi-dimensional extensions of) SQL queries, while a data lake may need to support different query languages (e.g., SQL, Cypher[11], JSONiq[12]), and programming languages (e.g., Python, Java, R). Further differences in system design and implementation between data lakes and data warehouses can be derived from these key differences.

Some industrial proposals also suggest to use data lakes to simply store massive raw data, or when the usage of data is not clear; later they store a subset of cleaned, transformed, nonvolatile data in data warehouses, or – in the so-called Lambda architecture – mix clean integrated warehouse data and current raw data streams in data lakes to combine real-time capability with data warehouse quality.

3.3 Data lakes and dataspaces

Before the emergence of data lakes, a similar concept was proposed by Microsoft Research initially for better personal data management, i.e., *dataspaces* [45, 61]. These early dataspace prototypes tackled the problems of how to organize, integrate, discover, and query the

[11] `https://neo4j.com/docs/cypher-manual/current/`
[12] `http://www.jsoniq.org/`

Table 1: Key differences between data warehouse and data lakes

Criteria	**Data Warehouses**	**Data Lakes**
Data ingestion	ETL	Load-as-is
Ingested data format	Structured	Heterogeneous (structured, semi-structured, and unstructured)
Data storage	Relational databases	Hadoop, Relational databases, NoSQL data stores, etc
Data access	SQL queries (OLTP, OLAP)	Different query languages (e.g., SQL, Cypher), programming languages (e.g., Java, Python, R)

data from several loosely interrelated data sources. They already mentioned a *pay-as-you-go* style of data integration to reduce the overhead, and suggested multiple services such as metadata catalog, storage, indexing, event detection, monitoring, support for complex workflows, effectively reusing human input, and handling data uncertainty and inconsistency.

Similar to data lake, dataspace proposals of the 2000s face several levels of variety problems. The data sources may have structured (e.g., relational databases), semi-structured (e.g., XML) or unstructured (e.g., text) formats. An early proposal was to have multiple data models with a hierarchy of expressive power, starting at the top with only basic metadata and a data source as a named Resource Description Framework (RDF) alike resource with basic information indicating its type and size; next, a bag-of-words data model which supports keyword queries; third, a labeled-graph data model, supporting simple path or containment queries, or complex queries based on the semi-structured data model; and finally, individual local data models like relational, XML, RDF, Web Ontology Language (OWL), etc. Query processing solutions in such a dataspace should be able to describe a collection of various relationships among participating sources, and provide multiple ways of data exploration such as keyword search, structured queries, result ranking, and source/participant discovery. We will see in Sec. 7 and 8 that similar research questions are also addressed in data lakes.

Stimulated by the growing concerns of losing data sovereignty in the presence of dominating Internet platforms, the *industrial data space* initiative extends these issues by a strong emphasis on sovereign data exchange among local data spaces/data lakes of small and medium IT users or vendor organizations [70]. Additional requirements collected from a broad range of industries and public organizations [101] include, e.g., trusted connectors controlling the import and export of data among dataspace partners, as well as usage control policies and services, in addition to what is needed for data lakes. Starting in 2015, the *International Data Space Association*[13] produced a reference architecture for the concept, and contributed many of them to GAIA-X [53], a central component of the Data Strategy of the European Union. In summary, the dataspace concept can be considered a complement to the data lake approach supporting sovereign inter-organizational cooperation.

[13] `https://www.internationaldataspaces.org/`

3.4 Use cases of data lakes

Data lakes have been proposed to store and manage data in many real-life use cases, e.g., Internet of things (IoT) and smart city [90], smart grids [93], air quality control [126], flights data [87], disease control, labor markets and products [13]. For a better understanding of DL definition and applications, we showcase real-life use cases of applying data lakes in manufacturing as illustrated in Fig. 1. It is abstracted and simplified from an Industrial 4.0 project *Internet of Production*[14].

Example 31 (**Multiple heterogeneous raw data inputs**). *Alice is a scientist who studies the milling process, and she has two problems with the data.*

First, vast and heterogeneous data is generated, which she needs to store and organize. Such production data includes binary image files from cameras, CSV and JSON files from different sensors, and ontologies enriched with her own annotation for the milling process. She hopes to store the heterogeneous data in raw formats, as transformation would be time-consuming, and might lose certain information and jeopardize her future research.

Second, instead of browsing through massive, diverse data files, she hopes to have a data management system, which provides her easy access to the raw data. A data lake would be a good solution to solve Alice's problems.

Example 32 (**Data integration and transformation**). *Bob is an industrial consultant, who has similar production data but in different formats, structures, and terminologies from Alice. Charlie is also a milling scientist but with a different research goal. He has another set of milling machines, sensors, and engineering models. For data analysis, he uses machine learning (ML) and runs Python scripts instead of merely queries.*

Both Bob and Charlie want to integrate and compare their data and results with Alice. The three users hope to have a data lake, which can combine these independent data sources, help them easily find the datasets relevant for their own use cases, transform the data flexibly, and provide query answering and data analytics.

[14] `https://www.iop.rwth-aachen.de/`

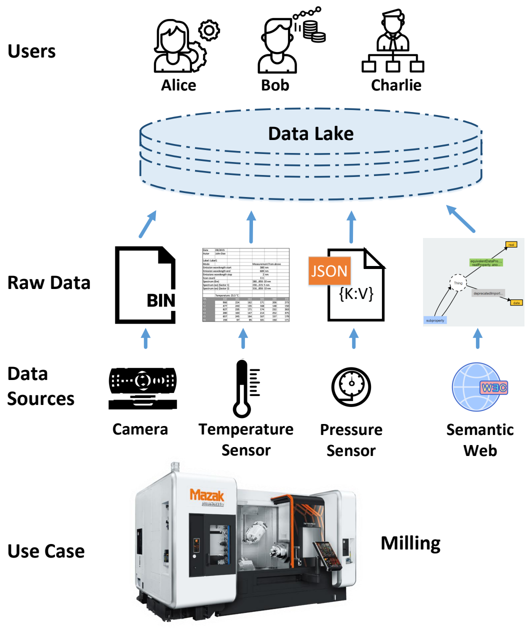

Fig. 1: Use case example: data lakes for manufacturing

Example 33 (**On-demand data processing and querying**). *As a researcher, Alice has her research questions and solutions evolving with time. She may introduce a new sensor, which has different machine-generated data formats compared to the existing datasets in the data lake. She may have new queries or want to use the raw data in a different manner. In addition, in the beginning the usage of some raw datasets was unclear, which she just stored without using. Now she has created an engineering model for the unused data. Similar dynamic analytical requirements also apply to the use cases of Bob and Charlie. Therefore, they would like to have a data lake, which supports data processing, integration, and querying in an on-the-fly manner rather than being fixed from the beginning.*

4 Data lake architecture

The architecture of a data lake describes the structure and components of the system, indicating how to store, organize and use the data. Rather than repeating a recent survey on data lake architecture [112], we briefly review two high-level data lake philosophies, but then only present an integrative function-oriented architecture used to structure this survey. Additionally, we propose classification criteria for data lake studies, and discuss the different kinds of data lake users.

4.1 Pond and zone architectures

The *pond architecture* [66] partitions ingested data by their status and usage. 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 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, 134, 115, 102], 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.

4.2 Our proposed architecture

High-level architectural philosophies, such as pond or zone architecture, often lack technical details about **functions**, which hampers modular and repeatable implementations. Therefore, we propose in Fig. 2 an architecture based on [56]. Besides the data storage systems, it divides the tasks of the whole workflow into three functional tiers, along with the workflow *when* the task should be performed. There are certain tasks conducted during or right after data is ingested (ingestion). Other tasks are triggered by queries as shown on the right side of Fig. 2 (exploration). Tasks in the middle (maintenance) conduct the general management and organization of the ingested datasets, but can also be considered as the preparation for querying. This architecture can also be seen as an abstraction of earlier layer-based data lake proposals [120, 115, 70, 38]. But here we define relevant components, and specify their functions.

Our main goals of proposing such an architecture are for the discussion of necessary functions in the whole workflow of a data lake, and for comparing existing technologies for each function in Sec. 6-8. This provides a much more comprehensive and complete view compared to earlier DL architecture proposals, and permits a fine-grained categorization of existing DL research and practice.

4.3 Classification criteria

Many DL research works focus on one specific task in data lakes. We classify them by their functions in the architecture of Fig. 2. Of course, some proposals provide more than one function. We will give a detailed explanation of each functional criterion in Sec.6-8, and group existing data lake solutions as listed in Table 2.

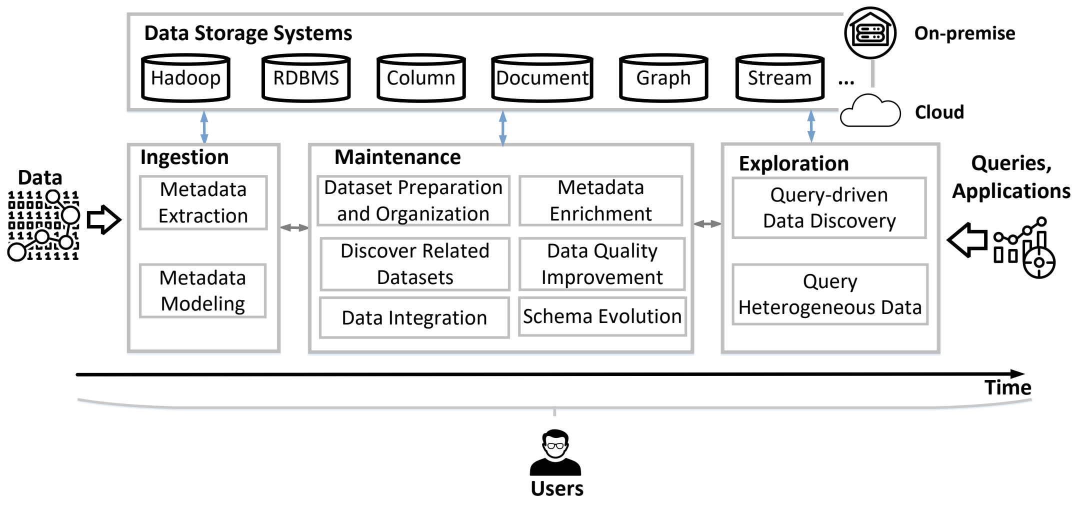

Fig. 2: Proposed architecture for existing DL solution categorization

In Sec. 5 we introduce the data storage strategies applied in existing data lake solutions, because the specific choice of storage strategy often shapes the required functions. The *ingestion* layer is responsible for importing data from heterogeneous sources into the data lake system. The main challenges are about extracting and modeling metadata (Sec. 6). To prepare the ingested raw data for querying or analytics, a data lake needs a set of operations in the *maintenance* layer to organize, discover, integrate or transform the datasets (Sec. 7). The functions in the *exploration* tier mainly contribute to allowing users to access the data lake (Sec. 8).

Depending on the purpose of the data lakes, there could be an external application layer on top of these three tiers e.g., for visualization [90]. The design choices of the application layer are myriad in practice. Thus, we leave them out of this survey.

4.4 Data lake users

As indicated in Fig. 2, human users 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); (4) *operations team* who maintains the data lake (e.g., data quality analysts, integration developers). Such users can help improve the semantics of data lakes over time, by adding metadata tags and linkage based on conceptual models or standard vocabularies resp. ontologies, such as schema.org [119, 70]. In the sequel, we do not emphasize the different kinds and generally talk about "data lake users".

It is not an easy task to design a data lake that has effective control of **data security** over heterogeneous data stores in data lakes. Only few data lake systems have targeted this problem, often based on their data stores. A few tools are mentioned in [29] for system authentication, authorization, and data encryption based on the Hadoop platform, e.g., Apache Ranger[15]. Among the full DL prototypes, CoreDB [10, 11] creates different users or roles for access control, and enables authentication and data encryption.

5 Storage

The data lake storage problem asks which data storage systems should be used to preserve the ingested datasets. Some approaches rely on the common relational or NoSQL databases while others have developed new storage systems, or combinations (Polystore); the upper part of Fig. 2 depicts diverse choices which could be operated on-premise or in clouds. We classify solutions by how the ingested data is stored in the data lake: as files, in a single database format, or using polystores. We also briefly mention industrial solutions that build data lakes on cloud platforms.

5.1 File-based storage systems

The Hadoop Distributed File System (HDFS) is one of the most frequently mentioned data storage systems for data lakes [21, 119, 13]. HDFS supports a wide range of

[15] `https://ranger.apache.org/`

Table 2: Classification of data lake solutions based on functions

Layer	**Function**	**Systems**
Ingestion	Metadata extraction	GEMMS [108]
		DATAMARAN [48]
		Skluma [118]
	Metadata modeling	Generic metadata model (GEMMS [108], Constance [59])
		Data vault [98, 52]
		Graph-based metadata model (Diamantiniet al. [32, 33], Aurum [43], Sawadogoet et al. [113])
Maintenance	Dataset preparation and organization	KAYAK [81, 82]
		GOODS [62, 63]
		DS-Prox [6], DS-kNN [4]
		Nargesian et al. [94]
		Juneau [130]
	Discover related datasets	Aurum [43]
		Brackenbury et.al. [16]
		JOSIE [131]
		D^3L [14]
		Juneau [129, 68, 130]
	Data integration	Constance [56, 60, 58, 57]
	Metadata enrichment	CoreDB [10, 11]
		GOODS [62, 63]
		Constance [59]
	Data quality improvement	CLAMS [41]
		Constance [59]
	Schema evolution	Klettkeet et al. [71]
Exploration	Query-driven data discovery	Aurum [43]
		JOSIE [131]
		D^3L[14]
		Juneau [129, 68, 130]
	Query heterogeneous data	Constance [56, 60]
		CoreDB [10, 11]
		Ontario [38] Squerall [84]

files [54]. Besides text (e.g., CSV, XML, JSON) and binary files (e.g., images), it supports certain formats for data compression, e.g., Snappy[16], Gzip[17]. It also allows columnar storage formats such as Parquet[18] and row-based storage format Avro[19] that enable easy schema management.

As discussed in Sec. 3, Hadoop alone usually does not fulfill the goals of a data lake. Microsoft's *Azure data lake store* [109] offers a hierarchical, multi-tier file-based storage system.[20] 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, YARN[21], Sqoop[22]).

CLAMS [41] uses RDF as its data model. It can directly ingest RDF data; for other formats, the ingestion must be extended by information extraction tools (e.g., Open Calais[23]) for extracting RDF triples, which could be linked to existing ontologies. CLAMS stores the ingested dataset in HDFS, and allows users to register the datasets for constraint discovery and data cleaning.

5.2 Single data store

Some DL systems aim at specific types of data and employ a single database system as the storage system.

As one example, *Personal data lake* [125] applies a graph-based data model (e.g., property graphs), and stores data in Neo4j. The proposed data lake has a special application focus on user data of usually relatively small size compared to business scenarios, but higher requirements regarding data privacy. Inputs of the personal data lake, heterogeneous personal data fragments

[16] `https://github.com/google/snappy`
[17] `https://www.gzip.org/`
[18] `https://parquet.apache.org/`
[19] `https://avro.apache.org/`
[20] The latest system is known as *Azure Data Lake Storage Gen2*: `https://docs.databricks.com/data/data-sources/azure/azure-datalake-gen2.html`
[21] `https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`
[22] `https://sqoop.apache.org/`
[23] `https://www.refinitiv.com/en/products/intelligent-tagging-text-analytics`

generated from user-web interaction (structured, semi-structured, unstructured), are serialized to specifically defined JSON objects, which 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.

Going further, *multi-model databases* support multiple data models and formats in a single database (for a survey, cf. [80]). 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.

5.3 Polystore systems

By definition, a data lake supports heterogeneous data in raw format, e.g. for zone architectures [134]. *Polystore* (or *multistore*) systems implement *polyglot persistence*, i.e., with integrated access to a configuration of multiple data stores for heterogeneous data. For example, Constance [56, 60] ingests raw data depending on their original format to 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 stream data source of large binary image files, which requires parallel data compression. If these defaults seem inadequate, users can specify the data store via the UI. Google's data lake, *Google Dataset Search (GOODS)* [62, 63] supports heterogeneous data storage used in Google's key-value stores *Bigtable* [23], file systems, and *Spanner* [28]; see also Sec. 5.4. Another data lake allowing raw data stored in both relational and NoSQL is *CoreDB* [10, 11]. To store diverse data from web applications, besides relational databases (e.g., MySQL, PostgreSQL, Oracle) it supports multiple NoSQL systems, i.e., MongoDB, HBase[24], and HIVE. JSON is used as a unified format to represent entities.

As a data lake designed for supporting data science tasks, *Juneau* [130] 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 polystore has been booming with regard to systems (e.g., *Polybase* [64], *BigDAWG* [37]) and open-source tools (e.g., Drill[25], Spark) – see [121] for a survey. Although they are not claimed as data lakes, we discuss in Sec. 8 their role as part of a data lake architecture in terms of how to store and query diverse source data.

5.4 Data lakes on clouds

In the aforementioned data lake systems, most are on-premises, except for a few real-life applications. For instance, the data lake in [93] uses Google Infrastructure as a Service (IaaS) cloud computing platform.

For commercial data lakes, it is a more common practice to build data lakes on clouds due to their large scale of data. 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)[26], Azure Data Lake Store[27], Google Cloud Platform (GCP)[28], Alibaba Cloud[29], and the Data Cloud from Snowflake[30].

We take GCP as an example.[31] The data lake serves as a raw data repository for heterogeneous data. GCP provides multiple data stores for different types of data and usages: *Cloud Storage*[32] for unstructured data, *Cloud SQL*[33] for structured data and transactional workload, *Cloud Spanner*[34] for horizontal scalability, etc. Then the user can build a pipeline (e.g., an ETL pipeline) to prepare the data, before loading it into a data warehouse (e.g., *BigQuery*[35]).

Cloud platforms have several prominent advantages for building data lakes. In such a cloud data lake, one

[24] `https://hbase.apache.org/`

[25] `https://drill.apache.org/`

[26] `https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/`

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

[28] `https://cloud.google.com/solutions/build-a-data-lake-on-gcp`

[29] `https://www.alibabacloud.com/product/data-lake-analytics`

[30] `https://www.snowflake.com/workloads/data-lake/`

[31] Considering the rapid evolution of industrial solutions, we encourage readers to check the latest information for the content of this subsection.

[32] `https://cloud.google.com/storage`

[33] `https://cloud.google.com/sql-server`

[34] `https://cloud.google.com/spanner`

[35] `https://cloud.google.com/bigquery`

can scale the storage space and computation power dynamically, and in many cases the prices of resources are more economic than on-premises. Moreover, the 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 practical functions on top of 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. In conclusion, it is interesting to see more data lake solutions researched and developed over cloud platforms.

6 Ingestion

Ingestion components load data into the lake, and store data into databases or file systems. The required throughput and synchronization of multiple incoming data streams may require sophisticated scheduling approaches that are typically associated with the storage systems. In this section, however, we focus on the question of metadata modeling and extraction during or shortly after ingestion, which is essential to prevent turning data lakes into incomprehensible data swamps.

6.1 Metadata extraction

The process of *metadata extraction* discovers metadata information that is essential for accessing a dataset, e.g., its name, extension, structure. In Sec. 7.4 we further address detecting more *hidden* metadata such as functional dependencies.

The *Generic and Extensible Metadata Management System* (*GEMMS*) for data lakes [108] is a framework that extracts metadata from heterogeneous sources, and stores the metadata in an extensible metamodel. Since the data sources and schemas may change over time and in some cases unknown in advance, 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. Then it transforms and stores the metadata in an extensible metamodel, such that new types of sources could be easily integrated. It also provides basic querying support. The following work Constance [56] can also extract structural metadata, i.e., schemas from semi-structured files such as XML and JSON.

DATAMARAN [48] provides an approach to extract the structures from semi-structured log files whose records span multiple lines, with record types and boundaries. DATAMARAN follows a three-step algorithmic approach. It first generates candidate *structure templates*, which use regular expressions [117] 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 (Assumption 1, [48]) 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 DATAMARAN process does not require human supervision and provides a high extraction accuracy compared to existing works. To mimic a real data lake, the authors crawled 100 datasets with large log files from GitHub.

For scientific data files, *Skluma* [118] extracts JSON metadata for content and context of scientific data files. 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, this kind of approaches can be expected to grow significantly over the next years.

6.2 Metadata modeling

To make DL content findable, accessible, integratable, and reusable (FAIR Principle [127]), a natural question is how to structure and organize the metadata in a formal way, i.e., *metadata modeling*. The majority of such models are either logic-based or graph-structured with more or less formal semantics.

6.2.1 Generic metadata model

The logic-based metadata model of GEMMS [108] 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 referred by Uniform Resource Identifier (URI) can be attached to metadata elements as semantic metadata. The relationships between the model elements are “has-a” relationships or aggregations. In [59], this metadata model is extended for representing individual schemas for relational tables, JSON, and labeled property graphs of Neo4j.

6.2.2 Data vault

For structured or semi-structured data in typical business scenarios, a promising conceptual modeling environment is *data vault* [98, 52]. It has three main elements types: *hubs* representing business concepts, *links* indicating the many-to-many relationships among hubs, and *satellites* with descriptive properties of hubs and links [78, 79]. Nogueira et al. [98] 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. [52] report experience with applying data vault for data lakes in the domains of manufacturing, finance, and customer service, pointing out practical obstacles such as inconsistencies among data sources.

6.2.3 Graph-based metadata model

Adapting ideas about knowledge graphs as pursued in the linked data and semantic web community, several network- or hypergraph-based metamodels have been proposed for data lakes.

Again in the business context, a proposed network metadata model [32, 33], focuses on business names, data field descriptions, and rules, in addition to data formats and schemas. It creates a graph-based representation with XML/JSON nodes and labeled arcs indicating their relationship. Nodes can be merged nodes based on lexical and string similarities, and linked to semantic knowledge (e.g., from DBpedia). Moreover, it suggests extracting thematic views of interest to the business, similar to data marts in DW.

To efficiently discover relevant datasets from massive data sources, Stonebraker and colleagues proposed *Aurum* [43] with an *enterprise knowledge graph (EKG)* to capture and query the relationships among datasets. An EKG is a hypergraph with three elements: nodes, weighted edges, and hyperedges. Nodes represent dataset attributes; an edge between two attribute nodes represents their relationship; and hyperedges represent different granularities among arbitrary numbers of nodes, e.g., connecting attributes and tables. Aurum builds the corresponding EKG, maintains it upon data changes and allows users to query EKG with a graph query language based on discovery primitives.

In [113], six evolution-oriented features of metadata management in data lakes are emphasized: 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 comprises 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.

7 Maintenance

After ingesting heterogeneous raw data sources, a data lake is a vast, unrelated collection with initially little metadata. 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, we categorize the maintenance-related functions into six groups and discuss corresponding data lake solutions in what follows.

7.1 Dataset preparation and organization

The *dataset organization* problem studies how to structure and navigate the massive heterogeneous datasets in data lakes. Often, it requires *preparing* and *profiling* the datasets with their metadata.

7.1.1 Dataset preparation

KAYAK [81, 82] focuses on just-in-time data preparation in data lakes. It automatically collects and stores in catalogs *intra-dataset* metadata describing individual datasets, and *inter-dataset* metadata for relationships among the attributes of different datasets. Inter-dataset metadata includes integrity constraints, semantic connection [111] (calculated based on external domain knowledge such as ontologies) and data instance overlap. Overall, the metadata form a graph where nodes are the datasets, and weighted edges indicate joinability by similarity values based on inter-dataset metadata.

For dataset preparation KAYAK first defines basic tasks such as basic profiling, compute joinability. 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. To represent data preparation pipelines it uses a *directed acyclic graph (DAG)* with primitives as nodes and their dependencies (based on execution order) as edges. For instance, an edge $a \rightarrow b$ indicates that the steps of primitive a (e.g., insert a dataset) needs to be executed before the primitive `b` (e.g., use the dataset to train an ML model). In Table 3 we compare different usages and definitions of DAGs in KAYAK and other systems to be discussed.

To manage dependencies among tasks and execute the atomic tasks of a primitive in parallel, KAYAK has a second type of DAG in Table 3. Here each node represents an atomic task, and the directed edges indicate the execution order of two tasks. Such a graph shows which tasks can be parallelized during execution.

7.1.2 Data lake organization

The GOODS data lake [62, 63] allows datasets to be created, stored, and modified in the data lake first, then conducts 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 and project, and temporal metadata. This categorization of metadata is closely related to Google's specific information retrieval requirements, but also a generally nice example of metadata management in data lakes. Data lake developers should customize the metadata catalog and algorithms to their own usage scenarios.

DS-Prox [6] and a later version *DS-kNN* [4] consider the dataset organization problem as a classification problem. That is, in an incremental manner, for each dataset DS-kNN adds it to a new category or an existing category that consists of already classified datasets. It has applied the k-nearest-neighbour (k-NN) for grouping the datasets. Before the step of classification, DS-kNN first conducts data preparation by feature extraction. The features are extracted through a metadata management process [3]. 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 the metadata, e.g., number of attributes, type of each attribute. With features of two datasets, DS-kNN then computes their similarity with string comparison metric in Levenshtein distance [85]. 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 with 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. In a later work [5], it uses supervised ensemble models to obtain the similarity values between dataset pairs. If the similarity value of two datasets reaches the threshold value, then they are proposed as candidates for schema matching.

Nargesian et al. [94] defined the *data lake organization* problem as discovering the optimal structure for users 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 3, a **DAG**-based organization has attributes of tables as leaf nodes; non-leaf nodes are assigned 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. For instance, a parent non-leaf node $food$ represents a set of food-related attributes, and has three child non-leaf nodes with $meat$, $vegetables$, and $fruit$, which are the subsets of attributes of $food$.
- To measure semantic similarities among attributes, the attribute values are represented as word embeddings [96], and cosine similarity is applied. 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 $food$), the transition probability is only relevant with the current node in DAG and the similarities between its child nodes and the given topic. The proposed algorithms in [94] try to find the organization structure that achieves the maximum probability for all the attributes of tables to be found.

In Sec. 5.3 we mentioned that *Juneau* [130] is a data lake handling computational notebooks, workflows, and cells, from which Juneau builds graphs for data management. From the workflows, it builds *workflow graphs*. 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. If there are program variables used for the data object and computational module, the variables are added as the edge label. Moreover, another type of graph in Juneau is for managing the relationships of variables in notebooks, referred to as *variable*

Table 3: Comparison of DAG-based systems for dataset preparation and organization

System	*KAYAK [81, 82] (pipeline)*	*KAYAK [81, 82] (task dependency)*	*Nargesian et al. [94]*	*Juneau [130] (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
Nodes	Primitives	Atomic tasks for data preparation operations	Sets of attributes	Notebook variables
Edges	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

dependency graphs, which is defined as a DAG-based model as listed in Table 3. In a variable dependency graph, nodes represent the variables, and the labeled, directed edges indicate one variable is computed using another variable through a function. For instance, the edge $data \xrightarrow{train} var$ indicates that the variable `data` is the input of function `train`, whose output variable is `var`. Then by examining the subgraph isomorphism Juneau is able to discover tables sharing similar workflows of notebooks (similar sequences/patterns of variables and functions).

7.2 Discover related datasets

When a data lake is hosting a large number of datasets, it is neither realistic nor necessary to query or integrate all the datasets. It might be more beneficial that data lakes first discover useful datasets for the current purpose. Moreover, the relatedness discovered among datasets is also a valuable type of metadata for exploring a data lake and preventing a data swamp, e.g. for entity resolution or resolving inconsistency across datasets. 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 attributes names or overlapped instance values.

As shown in Table 4, we now 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).

7.2.1 Aurum

Aurum [43] employs hypergraphs (i.e., EKG) to find similarity-based relationships and primary-foreign key candidates among tabular datasets. To construct the hypergraph introduced in Sec. 6.2.3, it first profiles each table column by adding *signatures*, i.e. information extracted from column values such as cardinality, data distribution, a representation of data values (i.e., MinHash). It indexes these signatures in locality-sensitive hashing (LSH) with Jaccard similarity using the MinHash signature, or cosine similarity with TF-IDF signature. If two columns have their signatures indexed into the same bucket after hashing, an edge is created between the two column nodes, and the similarity value is stored as the weight of the edge. 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. Given the total number n of attributes of all datasets, instead of conducting an all-pair comparison whose complexity is $\mathcal{O}(n^2)$, it profiles columns with signatures and indexes them in LSH, searches the approximate nearest neighbors, and only needs $\mathcal{O}(n)$. By using data parallelism, the process of profile construction can scale to a considerable number of data sources in the data lake. When source data is changed, Aurum does not re-read all the data, but first computes a magnitude of the Jaccard similarity. Only if the difference compared to the original value is above a threshold, it updates the signature and hypergraph.

7.2.2 Human-in-the-loop data discovery

The high-level data lake proposal in [16] shares a similar idea for using multiple criteria to measure dataset similarities, but includes humans in the loop to make similarity decisions if the algorithms alone cannot provide reliable suggestions. To find joinable datasets, it measures the similarity of files (e.g., HTML tables), and considers approximate matches in terms of data values, schemas and descriptive metadata (the source of data, information added by users, etc). For measuring the similarity of the files and group them, it computes the Jaccard similarity between file paths using MinHash and LSH.

7.2.3 JOSIE

Given a data lake with tabular data (e.g., a corpus of web tables), *JOSIE* [131] addressed two challenges with regard to applying existing overlap set similarity search

Table 4: Comparison of related dataset discovery approaches in data lakes

Systems	**Relatedness criteria**	**Similarity metrics**	**Auxiliary structure**
Aurum [43]	Instance value overlap Attribute name Primary-key/foreign-key candidate	Jaccard similarity (MinHash) Cosine similarity (TF-IDF)	Hypergraph
Brackenbury et.al. [16]	Instance value overlap Attribute name Semantics Descriptive metadata	Jaccard similarity (MinHash)	-
JOSIE [131]	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–dimensional Euclidean space
Juneau [129, 68, 130]	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

solutions in data lakes. A data lake may contain a large number of tables, thus, the number of columns and the distinct values from a column could also be large; and it could be difficult for a human user to directly give an appropriate threshold value θ for the intersection value.

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. JOSIE formulates the problem of joinable table discovery as the problem of finding 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. JOSIE returns *top-k* results and scalable with a large number of tables. It employs a cost model to eliminate the unqualified candidates effectively. Such a method makes the performance robust to different data distributions.

7.2.4 D^3L

D^3L [14] broadened the criteria of "relatedness" that make a dataset relevant. It computes five features for multiple similarities, including attribute names, instance value overlaps, representation patterns of instance values, semantics for textual attributes, and data distribution for numerical attributes. Given attributes of tables, D^3L first transforms schemas and data instances to intermediate representations with features of q-grams, TF/IDF tokens, regular expressions, word-embeddings, and the Kolmogorov-Smirnov statistic [27]. Similar to previous data discovery systems, D^3L first computes the pair-wise similarities between attributes based on Jaccard similarity (MinHash [17]), and high cosine similarity (random projections [24]). From the similarity values of five features, it transforms the problem of finding the relatedness between tables to the calculation of weighted Euclidean distance in the 5–dimensional Euclidean space. In such a distance calculation, the weight of each feature (i.e., feature coefficients) indicates its significance for the combined distance.

To optimize the weights of the five features, D^3L trains a **binary classifier**, i.e., logistic regression, over a training dataset with relatedness ground truth, and applies the coefficients of the trained model as the weight of features for distance calculation. D^3L also builds LSH to index the features and map them to the distance space. It considers two items similar if they are hashed into the same buckets. One interesting finding in this work is that using LSH to discover joining paths leads to finding more related tables (and attributes), also with higher accuracy.

7.2.5 Juneau

Juneau [129, 68, 130] 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 a desired target table via the user interface, the system can automatically return a ranked list of tables, which might be relevant to the given table. Specifically, as shown in Table 4, Juneau extends the notion of "relatedness" with the following signals.

1. In addition to value overlap and similar attribute names, it considers pairwise matched attributes that share similar attribute value domains, and matched candidate key pairs. The proposed similarity metrics are based on Jaccard similarity; sketches and LSH-based approximation [44, 132] 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 introduced in Sec. 7.1.2, it defines the *provenance similarity* of two tables based on the graph similarity of their variables. This measurement aims to help connecting 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 tables with similar descriptive metadata (e.g., information about the data science task), and number difference of null values (e.g., fill missing values in a data cleaning task).

For a specific data science task, Juneau picks a subset of aforementioned relatedness features, measures and combines their similarity values. For instance, when searching tables for a data cleaning task, it considers the instance value overlap, schema overlap, provenance similarity, and null value differences.

7.2.6 Discussion

In this subsection, we have reviewed one of the hottest topics in data lakes, i.e., data discovery.[36]

We can observe a standard procedure here: the first step is to define and extract relatedness signals from tables w.r.t. data (e.g., value overlaps, data distribution patterns), schemas (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 [9]) 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 to query the data lake, which is discussed in Section 8.1.

However, even though all data lakes introduced here tackle the data discovery problem, they have slightly different focuses. Aurum is fast and robust against data value changes and offers a graph-based structure; JOSIE shows a high performance; D^3L improves the accuracy of discovered related tables by dimensions of similarity. Juneau emphasizes workflows for multiple data science tasks. Therefore, their individual similarity measures, applied data structures and proposed algorithms vary significantly. We can expect more novel perspectives and techniques to be brought to the landscape of data discovery in data lakes.

[36] Refer to a recent tutorial [95] for more potential solutions for data discovery in data lakes.

7.3 Data integration

Data integration (DI) studies the problem of combining multiple heterogeneous data sources and providing unified data access for users [36]. Given a large scale of sources in a data lake, users might need to first discover a subset of relevant datasets (the related dataset discovery process in Sec. 7.2), before resolving the heterogeneities of sources with regard to data models and schemas.

The fundamental data integration techniques include schema matching, schema mapping, query reformulation, entity linkage, etc. Few data lake proposals provide an end-to-end data integration pipeline. Constance [56] has the extracted schemas of relational, JSON documents and graph data modeled in the generic metadata model (see Sec. 6.2.1). For data integration Constance first performs schema matching, which finds the similar attributes representing the same real-world entity. Instead of creating a global schema covering the information of all the datasets in the data lake, users can select a subset of data sources and the system generates an *integrated schema* for *partial* integration, or users can choose a set of relevant schema elements via the user interface to form the desired integrated schema. Next Constance generates schema mappings, which preserve the relationships between the source schemas and integrated schema [58]. With schema mappings Constance performs query rewriting and data transformation in a **polystore**-based setting [60]. It rewrites the input user query (against the integrated schema) to subqueries (against source schemas), 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 optimizes query execution and reduce the amount of data to be loaded.

7.4 Metadata enrichment

As discussed in Section 6.1, in order to access and use the data, it is necessary to extract certain metadata from the raw ingested data in a data lake, e.g., schemas. To further understand and explore a dataset, sometimes it is necessary to perform additional steps to discover more metadata of a dataset, such as semantic knowledge, its relationships to other datasets, and constraints that could also be helpful for data cleaning. We introduce the following data lake systems providing such enrichment of metadata.

CoreDB [10, 11] is developed given that more insights should be extracted and enriched from the 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; it connects the features to open knowledge bases such as Google Knowledge Graph[37], Wikidata[38]; it further annotates and groups the data sources in the data lake.

In order to obtain metadata of a dataset that precisely describes its origin, ownership, and possible usage, it is often beneficial to keep human experts in the loop. Google's data lake GOODS [62, 63] 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.

Constance [59] can also enrich the metadata in the data lake by discovering dependencies, more specifically, relaxed functional dependencies (RFD) [22]. 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 certain 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.

[37] `https://developers.google.com/knowledge-graph/`
[38] `https://www.wikidata.org/wiki/Wikidata:Main_Page`

7.5 Data quality improvement

A data lake provides means for data quality management, which guarantees the quality and efficiency of the user query results. The ingested raw data sometimes has inadequate data quality. Thus, there are certain proposals about how to obtain dependencies from the data in the data lakes, and then use them to **improve** the data quality.

CLAMS [41] 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 schemas 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 accordingly it ranks the RDF triples and asks the user to validate whether such a candidate dirty triple should be removed.

Constance [59] 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.

7.6 Schema evolution

Schema evolution requires handling the changes of schemas and integrity constraints [30]. In data lakes, the possible challenge of schema evolution could be the heterogeneity of the schemas 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.

In [71] it addresses the problem of how to construct the whole evolving history of schemas given data stored in NoSQL databases, e.g., JSON stored in MongoDB. Instead of table schemas in relational databases, it considers the structure persisted object in NoSQL database, 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 [71] is that in NoSQL databases often schemas are "less" normalized, which leads to the inclusion dependencies involving multiple attributes rather than a single attribute as in relational databases. In [71] an algorithm is proposed to detect such k-ary inclusion dependencies.

8 Exploration

It is important that useful information can be retrieved from data lakes. However, this is often a challenging task due to a large number of ingested sources, and the heterogeneity of data. A user may have knowledge of one or a few data sources, but rarely all the datasets. Thus, the existing solutions mainly solve the querying problem in data lakes in the following two directions: discover the data lakes based on the relatedness of datasets (Sec. 8.1), or provide a unified query interface for heterogeneous data sources (Sec. 8.2).

8.1 Query-driven data discovery

Query-driven data discovery [91] refers to searching a data lake based on the measured relatedness (e.g., joinable, unionable) among datasets (e.g. web tables) as introduced in Sec. 7.2. With input queries specifying a given dataset (usually tabular data), the system returns the top-k most related datasets.

8.1.1 Exploration methods

There are roughly three ways of exploration. We denote the set of datasets in the 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 with T, e.g., JOSIE [131].
2. Given the user-specified 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 include 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* [130].

Notably, in this group of studies the challenge of exploring datasets is a "search" problem rather than the query reformulation problem in data integration. It can also be seen as a step prior to data integration or data science tasks [14, 130].

8.1.2 Querying methods and indexing

Given an input query table, the data lakes in Table 4 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 set. In addition, Aurum [43] 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 schemas 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., users can obtain ranking results of different criteria without re-running the query. In Juneau [130] a query is a cell output table picked by the user. The user also chooses the type of 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 and tables connected by workflow steps, and *pruning* tables under a threshold of schema-level overlap.

8.1.3 Remarks on future directions

DL exploration could benefit greatly by taking into account recent results on web table exploration [76, 19, 105], data wrangling [124, 46], or external applications upon the data lake. With the existing works mainly focus 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.

8.2 Query heterogeneous data

For accessing a data lake, the second group of studies focuses on providing a unified querying interface for heterogeneous data.

Constance [56, 60] provides an incremental manner for users to explore the data lake. A user can first browse the existing data sources, including their description, statistics, and schema; then she can write a query (SQL or JSONiq) for a single dataset, or use the user interface to make a keyword search over the schema or the data. Alternatively, with certain knowledge of the datasets, which could be learned through the previous exploration processes, she can choose to integrate and query a subset of datasets as introduced in Sec. 7.3. 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 [50].

CoreDB [10, 11] 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[39] for the underlying full-text search, SQL queries for relational database systems, and SPARQL queries for knowledge graphs.

Ontario [38] and *Squerall* [84] 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[40]), 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, the data lake provide 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. 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, with the mapping the relevant data entities are retrieved from data sources, which are joined and transformed to form the final query results. Squerall features distributed query processing and is implemented with two versions with different data connectors: Spark and Presto[41].

Potentially useful systems. In Sec. 5.3 we have discussed that polystore systems enable multiple data models and diverse query languages. We divide their querying solutions into three groups.

1. A straightforward strategy is to transform the data in heterogeneous NoSQL stores into relational tables and uses an existing relational database to process the data. For example, Argo [25] studies enabling key-value and document-oriented systems in relational systems. Though such solutions offer an easy adaptation to existing relational systems, they come with a high cost of data transformation, especially with large datasets, and might not be scalable for big data settings.
2. Another group of approaches focuses on multistore systems providing a SQL-like query language to query NoSQL systems. For instance, CloudMdsQL [74] supports relational, document-oriented, and graph-based databases. BigIntegrator [133] is a system that enables querying over both relational databases and cloud-based data stores (e.g., Google's Bigtable). MuSQLE [51] performs optimization of queries across multiple SQL engines. FORWARD [100] provides a powerful query language SQL++ for both structured and semi-structured data. Although these approaches perform optimizations such as semi-join rewriting, they do not consider schema mappings during the rewriting steps.
3. The third category covers the approaches which provide more efficient query processing techniques by applying a middle-ware to access the multiple NoSQL stores, such as Estocada [18] and Polybase [64], BigDAWG [37], and MISO [75].

9 Composite metadata management in DLs

In Sec. 6-8 we have discussed individual functions provided by existing data lake systems. To prevent a data lake turn into a "data swamp", *metadata management* is crucial. As surveyed in [112], there are different types of metadata: schemas that preserve the structure of the dataset, semantic metadata, constraints, and other descriptive information. Metadata management is therefore necessary throughout all aspects of data lakes. It starts from extracting metadata such as schemas, semantics, provenance information from ingested datasets (Sec. 6.1); then it needs to model the metadata in a formal manner (Sec. 6.2); it is possible to enrich the metadata, e.g., relatedness signals extracted from datasets and the computed relatedness between datasets (Sec. 7.2), schema mappings (Sec. 7.3), semantics, functional dependencies, or human added annotations (Sec. 7.4); finally, the metadata is used for facilitating querying a data lake (Sec. 8), or improving the data quality (Sec. 7.5).

In the sequel, we look at two important integrative aspects, i.e. schema mapping and provenance of data.

9.1 Schema mapping formalisms

Schema mappings are also metadata; thus, metadata management includes studying their *algorithmic prop-*

[39] `https://www.elastic.co/`
[40] `https://virtuoso.openlinksw.com/`
[41] `https://prestodb.io/`

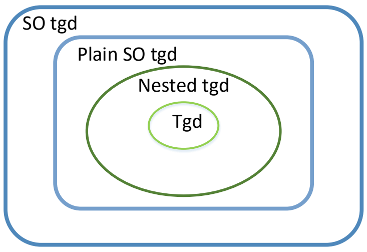

Fig. 3: Expressive power comparison

Table 5: Comparison of structural and reasoning properties

Structural/reasoning properties	**Tgds**	**Nested tgds**	**Plain SO tgds**	**SO tgds**
Closure under target homomorphisms	Yes	Yes	Yes	No
Admitting universal solutions	Yes	Yes	Yes	Yes
Allowing for conjunctive query rewriting	Yes	Yes	Yes	Yes
Decidable for implication problem	Yes	Yes	Unknown	No
Decidable for logical equivalence problem	Yes	Yes	Unknown	No

erties (computational efficiency) [72]. Schema mappings are often expressed as logical sentences, e.g., source-to-target tuple generating dependencies (s-t tgd), which are also known as Global-Local-as-View (GLAV) assertions [77]. In what follows, we discuss the expressive power and properties of mapping languages that are relevant to the metadata-related challenges in data lakes.

Expressive power. A data lake may have not only relational data but also hierarchically structured data such as XML and JSON. The ideal mapping language should be expressive enough to describe a wide variety of data integration or data exchange scenarios in data lakes. Regarding expressive power, Fig. 3 shows the strict inclusion hierarchy between the four most intensively studied schema mapping languages [7, 73]. For hierarchically structured data nested mappings [47] are proven to be more accurate for describing the relationship between source and target schemas, and also more efficient for data transformation than the basic mapping expressed in tgds. Nested mappings can be expressed by nested tuple-generating dependencies (nested tgds) [47, 122]. The second-order mapping formalism, i.e., second-order tuple-generating dependencies (SO tgds) [39] and plain second-order tuple-generating dependencies (plain SO tgds), have stronger expressive power than tgds and nested tgds, which means that they can describe more mapping scenarios. In particular, SO tgds and plain SO tgds are rules with Skolem functions, which are used for the creation of object identifiers. Many existing data exchange and integration applications [128, 47, 86, 15] generate mappings with Skolem functions as grouping keys for set elements, and their schema mappings can be expressed in plain SO tgds.

Structural properties and decidability of reasoning tasks. Table 5 summarizes the structural properties and decidability of implication and logical equivalence problems of these four mapping languages based on the results from [123, 7, 73, 42]. Here we discuss the three arguably most important structural properties of schema mapping languages: the existence of universal solutions, closure under target homomorphism, and allowing conjunctive query rewriting [123]. The first two properties are crucial for data exchange while the third property plays a fundamental role in query reformulation for data integration. When a mapping allows conjunctive query rewriting, it enables rewriting a union of conjunctive queries over the global/target schema to a union of conjunctive queries against the source schema, which corresponds to the polynomial time complexity in terms of data complexity. In Table 5 we can see that all three structural characteristics are satisfied except the case that SO tgds do not hold with the closure under target homomorphism. To the best of our knowledge, the two first-order mapping languages, i.e., tgds and nested tgds, are decidable for the logical equivalence problem and implication problem, which is not the case for plain SO tgds and SO tgds [73, 42].

It is a complicated problem to choose the *right* mapping language for a large-scale, heterogeneous data lake. In addition to the comparison in Fig. 3 and Table 5, in practice SO tgds and plain SO tgds are less desirable specifications compared to dependencies with FO semantics. To begin with, FO and SO logics correspond to algorithmic behaviors with different complexities [73]. The data complexity of the model checking problem for plain SO tgds is NP-complete while the same problem is merely LOGSPACE for nested tgds [104, 73]. FO tgds are more user-friendly and can be easily translated to SQL [97, 73]. Thus, there are studies [8, 57] transforming schema mappings in plain SO tgds to first-order mappings (i.e., tgds and nested tgds). Furthermore, to embrace ontologies in data lakes, one possible path is to apply description logic [20] that can express ontology language, and support query answering and verifying knowledge base satisfiability.

9.2 Data provenance

Data provenance (also known as *data lineage*) refers to the information of data records, which indicates their origin, usage, status in the whole life circle, etc. The provenance information tells how a dataset is obtained and helps to make proper access to the datasets. Such information could be extracted during data ingestion, later enriched during maintenance or exploration phases, possibly with human input.

In [124] a governance tool from IBM is developed, which can manage the requests for ingesting new data sources or using already ingested datasets in a data lake. Suriarachchi et al. [120] have proposed an abstracted architecture that provides integrated provenance (information of activities) given multiple data processing and analytics systems (e.g., Hadoop, Storm[42], and Spark), as these systems populate provenance events in different standards and apply various storage manners. They have also studied a use case, in which data from Twitter is collected and processed (e.g., count hashtags, aggregate data by each category) by Apache Flume[43], Hadoop jobs, and Spark jobs.

CoreDB [10, 11] and Juneau [129, 68, 130] both preserve the provenance information as graphs. CoreDB uses the descriptive, administrative and temporal metadata to build DAG-based provenance graphs [12]. In such a provenance graph model, nodes represent entities, and users/roles; edges represent CRUD (Create, Read, Update and Delete) or querying operations. For example, by querying the provenance graph one can answer the question: who queried this entity, and when? In Sec. 7.1.2 we have mentioned that Juneau [130] 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 among these variables and v.

10 New directions of data lakes

In this section, we list a few directions that are not well-studied in existing literature, yet indicate promising future for applications and development of data lakes.

10.1 Machine learning in data lakes

In recent years, machine learning has shown many new possibilities in information systems. With a large amount of data, a data lake could be applicable for certain machine learning or deep learning models.

ML models have been applied in data lakes for handling the problems of dataset organization and discovery. The related dataset discovery problem requires finding "similar" attributes and datasets from a large number of candidates, and often a set of features that can be extracted from the data, metadata, etc. Thus, the problem can be coined as a clustering or a classification (with category labels) problem. Supervised ML models are applied to assign datasets to categories [6, 4], i.e., KNN classifier, random forest ensemble. To optimize feature coefficients, D^3L trains a logistic regression model over a training dataset with relatedness ground truth. It is interesting to see whether there are more sophisticated solutions w.r.t. ML models, distance metric calculation, and feature selection to further improve the accuracy of related dataset discovery.

[42] `https://storm.apache.org/`
[43] `https://flume.apache.org/`

10.2 Data lakes for data science

One important application of data lakes is data science [91]. If a data lake is used for data science tasks, we call it a *scientific data lake*. A straightforward solution to build a scientific data lake, is to apply the open-source tools that support data preparation pipeline and ML-related tasks (e.g., model debugging). For instance, the data lake proposal in [90] uses Spark to build the logistic regression model for sentiment analysis and Python libraries (e.g., Matplotlib) for visualization.

A more sophisticated manner is to integrate the specific data science tasks into the design of data lake functions. First, when profiling the datasets, a scientific data lake might need to extract information relevant to the data science tasks [130, 16], e.g., manage notebook variables, scientific topics of the experimental data. KAYAK [82] divides the process of data preparation into atomic operations, then allows data scientists to optimize the whole pipeline of data preparation. Juneau [130] allows the training/validation data augmentation, linking data, features extraction, and data cleaning. For each data science task, the corresponding similarity measures can be assembled based on the basic similarity measures discussed in Sec. 7.2. For example, for data augmentation it also tries to find tables that can bring new data instances to improve the information gain. In this way, the accuracy of classification models could be improved.

Furthermore, given a scientific data lake with massive datasets, it is important to efficiently discover relevant data for data analysis [91]. The related dataset discovery systems introduced in Sec. 7.2 and Sec. 8.1 are contributing to this promising direction.

10.3 Stream data lakes

With the recent advances of Industrial 4.0, smart city, and e-commerce websites, a large volume of stream data in high speed needs to be selectively stored and analyzed in near real-time. Data lakes can store raw data and support heterogeneous data structure and formats, which makes data lakes a prominent technology in these scenarios, even though the concept of data stream management is based exactly on the assumption that there is not enough storage to keep all of them. Therefore, a core challenge in bringing both ideas together in *stream data lakes* will be to find an optimal storage mix of storing raw data in limited time windows, reducing them by data analytics on the fly, and selective storage of aggregated time series.

In [103], it is proposed to apply data lakes to store, integrate, and collectively analyze vast amounts of heterogeneous data streams generated from sensors and machines in the production environment. Built upon Apache Kafka[44] and Flink[45], it supports ingesting stream data into Hadoop, and other DBMSs with the support of Kubernetes[46]. In a smart city use case, [90] proposes to apply Apache Flume and Spark for ingesting and process stream data.

To build a stream data lake, besides applying the existing open-source data stream libraries (e.g., Kafka, Flume, Storm, and Spark), another challenge is how to apply existing stream techniques (e.g., Lambda architecture) in a data lake setting. For instance, the architecture proposed in [93] replaces the maintenance layer in Fig. 2 with the three parts of a Lambda architecture. Its batch layer serves all the data stored Hadoop in batches and precomputes batch views; the speed layer handles streams ingested by Flume in real-time and computes real-time views; the serving layer integrates the batch views and real-time views respectively such that results can be passed to the exploration layer of the system for data querying and analytics.

11 Summary and outlook

In the first decade of their existence, data lakes have been receiving increasing interest from both academia and industry. However, as a comprehensive concept, the term “data lake” still contains a large body of unclarity. Therefore, in this survey, we have started by providing an introduction of the definition of data lakes, then categorize and discuss existing data lake systems. With the existing data lake technologies grouped by their provided functions, it is easier to have a deeper analysis of the corresponding research questions. Note that even to design a data lake focusing on a specific function, it might still be necessary to complete the workflow with components from other tiers (e.g., for data discovery it also needs metadata extraction and querying) and set up the data storage systems.

The concept of “data lakes” is complex and evolving, not limited to the problems addressed in this survey. Some well-studied problems (e.g., data integration, schema evolution, metadata modeling) need new perspectives and methods in data lakes; while many blank spaces (e.g., stream data lakes, integrate data lakes with machine learning and data science) also call for novel solutions.

Acknowledgements This work was supported in part by Deutsche Forschungsgemeinschaft (DFG) under Germany’s Excellence Strategy – EXC-2023 Internet of Production – 390621612, and by the German Federal Ministry of Research (BMBF) in the IndaSpacePlus project (grant Id. 01IS17031).

[44] `https://kafka.apache.org/`
[45] `https://flink.apache.org/`
[46] `https://kubernetes.io/`

References

1. D. Abadi et al. The Beckman report on database research. *Commun. ACM*, 59(2):92–99, 2016.
2. S. M. F. Ali and R. Wrembel. From conceptual design to performance optimization of ETL workflows: current state of research and open problems. *The VLDB Journal*, 26(6):777–801, 2017.
3. A. Alserafi, A. Abelló, O. Romero, and T. Calders. Towards Information Profiling: Data Lake Content Metadata Management. In *ICDMW*, pages 178–185, 2016.
4. A. Alserafi, A. Abelló, O. Romero, and T. Calders. Keeping the Data Lake in Form: DS-kNN Datasets Categorization Using Proximity Mining. In *MEDI*, pages 35–49, 2019.
5. 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.*, 38(3):26:1–26:30, 2020.
6. A. Alserafi, T. Calders, A. Abelló, and O. Romero. DS-Prox: Dataset Proximity Mining for Governing the Data Lake. In *SISAP*, pages 284–299, 2017.
7. M. Arenas, J. Pérez, J. Reutter, and C. Riveros. The language of plain SO-tgds: Composition, inversion and structural properties. *Journal of Computer and System Sciences*, 79(6):763–784, 2013.
8. P. C. Arocena, B. Glavic, and R. J. Miller. Value invention in data exchange. In *SIGMOD*, pages 157–168. ACM, 2013.
9. M. Bawa, T. Condie, and P. Ganesan. LSH forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

10. A. Beheshti, B. Benatallah, R. Nouri, V. M. Chhieng, H. Xiong, and X. Zhao. CoreDB: a Data Lake Service. In *CIKM*, pages 2451–2454, 2017.
11. A. Beheshti, B. Benatallah, R. Nouri, and A. Tabebordbar. CoreKG: a Knowledge Lake Service. *PVLDB*, 11(12):1942–1945, 2018.
12. S.-M.-R. Beheshti, H. R. Motahari-Nezhad, and B. Benatallah. Temporal provenance model (TPM): model and query language. *arXiv preprint arXiv:1211.5009*, 2012.
13. E. Boci and S. Thistlethwaite. A novel big data architecture in support of ADS-B data analytic. In *ICNS*, pages C11–C18, 2015.
14. A. Bogatu, A. A. A. Fernandes, N. W. Paton, and N. Konstantinou. Dataset Discovery in Data Lakes. In *ICDE*, pages 709–720. IEEE, 2020.
15. A. Bonifati, E. Chang, T. Ho, L. V. Lakshmanan, R. Pottinger, and Y. Chung. Schema mapping and query translation in heterogeneous P2P XML databases. *VLDBJ*, 19(2):231–256, 2010.
16. W. Brackenbury, R. Liu, M. Mondal, A. J. Elmore, B. Ur, K. Chard, and M. J. Franklin. Draining the data swamp: A similarity-based approach. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 13:1–13:7, 2018.
17. A. Z. Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES*, pages 21–29. IEEE, 1997.
18. F. Bugiotti, D. Bursztyn, A. Deutsch, I. Ileana, and I. Manolescu. Invisible glue: scalable self-tuning multi-stores. In *CIDR*, 2015.
19. M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *PVLDB*, 2(1):1090–1101, 2009.
20. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Dl-lite: Tractable description logics for ontologies. In *AAAI*, volume 5, pages 602–607, 2005.
21. Capgemini SE and Software Pivotal. The Technology of the Business Data Lake Table. `https://www.capgemini.com/wp-content/uploads/2017/07/pivotal-business-data-lake-technical_brochure_web.pdf`, 2013.
22. L. Caruccio, V. Deufemia, and G. Polese. Relaxed functional dependencies - A survey of approaches. *IEEE Trans. Knowl. Data Eng.*, 28(1):147–165, 2016.
23. F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems*, 26(2):4:1–4:26, 2008.
24. M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.
25. C. Chasseur, Y. Li, and J. M. Patel. Enabling JSON Document Stores in Relational Systems. In *WebDB*, pages 1–6, 2013.
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 for Business Leaders*, 2014.
27. W. J. Conover. *Practical nonparametric statistics*, volume 350. John Wiley & Sons, 1998.
28. J. C. Corbett et al. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems*, 31(3):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 *SEKE*, pages 453–578, 2019.
30. C. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo. Automating the database schema evolution process. *The VLDB Journal*, 22(1):73–98, 2013.
31. P. T. Demirkan and Haluk. Data lakes: the biggest big data challenges. *Analytics*, 9(6):56–63, 2016.
32. 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 *M2U@ADBIS*, pages 165–177, 2018.
33. 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 *SEBD*, 2018.
34. J. Dixon. Pentaho, Hadoop, and Data Lakes — James Dixon's Blog. `https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/`, 2010.
35. J. Dixon. Data Lakes Revisited. `https://jamesdixon.wordpress.com/2014/09/25/data-lakes-revisited/`, 2014.
36. A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Elsevier, 2012.
37. J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. B. Zdonik. The BigDAWG Polystore System. *SIGMOD Record*, 44(2):11–16, 2015.
38. K. M. Endris, P. D. Rohde, M.-E. Vidal, and S. Auer. Ontario: Federated Query Processing Against a Semantic Data Lake. In *DEXA*, pages 379–395, 2019.

39. R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems (TODS)*, 30(4):994–1055, 2005.
40. 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 *CYBER*, pages 820–824. IEEE, 2015.
41. M. Farid, A. Roatis, I. F. Ilyas, H.-F. Hoffmann, and X. Chu. CLAMS: Bringing Quality to Data Lakes. In *SIGMOD*, pages 2089–2092, 2016.
42. I. Feinerer, R. Pichler, E. Sallinger, and V. Savenkov. On the undecidability of the equivalence of second-order tuple generating dependencies. *Information Systems*, 48:113–129, 2015.
43. R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A Data Discovery System. In *ICDE*, pages 1001–1012, 2018.
44. R. C. Fernandez, J. Min, D. Nava, and S. Madden. Lazo: A Cardinality-Based Method for Coupled Estimation of Jaccard Similarity and Containment. In *ICDE*, pages 1190–1201, 2019.
45. M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.
46. T. Furche, G. Gottlob, L. Libkin, G. Orsi, and N. W. Paton. Data Wrangling for Big Data: Challenges and Opportunities. In *EDBT*, volume 16, pages 473–478, 2016.
47. A. Fuxman, M. A. Hernandez, H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested mappings: schema mapping reloaded. In *VLDB*, pages 67–78, 2006.
48. Y. Gao, S. Huang, and A. Parameswaran. Navigating the Data Lake with DATAMARAN. In *SIGMOD*, pages 943–958, 2018.
49. I. Gartner. Gartner Says Beware of the Data Lake Fallacy. `https://www.gartner.com/newsroom/id/2809117`, 2014.
50. S. Geisler, C. Quix, R. Hai, and S. Alekh. An Integrated Ontology-Based Approach for Patent Classification in Medical Engineering. In *DILS*, pages 38–52, 2017.
51. V. Giannakouris, N. Papailiou, D. Tsoumakos, and N. Koziris. MuSQLE: Distributed SQL query execution over multiple engine environments. In *BigData*, pages 452–461, 2016.
52. C. Giebler, C. Gröger, E. Hoos, and B. Mitschang. Modeling Data Lakes with Data Vault: Practical Experiences, Assessment, and Lessons Learned. In *ER*, pages 63–77, 2019.
53. G.-X. W. Groups. *Reference Architecture Document, Release 21.03.* GAIA-X, European Association for Data and Cloud AISBL, 2021.
54. M. Grover, T. Malaska, J. Seidman, and G. Shapira. *Hadoop Application Architectures: Designing Real-World Big Data Applications.* O'Reilly Media, Inc., 2015.
55. W. Gryncewicz, M. Sitarska-Buba, and R. Zygała. *Agile Approach to Develop Data Lake Based Systems*, pages 201–216. Springer International Publishing, Cham, 2020.
56. R. Hai, S. Geisler, and C. Quix. Constance: An Intelligent Data Lake System. In *SIGMOD*, pages 2097–2100. ACM, 2016.
57. R. Hai and C. Quix. Rewriting of plain so tgds into nested tgds. *Proceedings of the VLDB Endowment*, 12(11):1526–1538, 2019.
58. R. Hai, C. Quix, and D. Kensche. Nested Schema Mappings for Integrating JSON. In *ER*, pages 397–405, 2018.
59. R. Hai, C. Quix, and D. Wang. Relaxed Functional Dependency Discovery in Heterogeneous Data Lakes. In *ER*, pages 225–239, 2019.
60. R. Hai, C. Quix, and C. Zhou. Query rewriting for heterogeneous data lakes. In *ADBIS*, pages 35–49, 2018.
61. A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.
62. A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing Google's Datasets. In *SIGMOD*, pages 795–806, 2016.
63. A. Y. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Managing Google's data lake: an overview of the Goods system. *IEEE Data Eng. Bull.*, 39(3):5–14, 2016.
64. D. Halverson, A. J DeWitt, R. Nehme, S. Shankar, J. Aguilar-Saborit, A. Avanes, M. Flasza, and J. Gramling. Split query processing in polybase. In *SIGMOD*, pages 1255–1266, jun 2013.
65. T. S. Hukkeri, V. Kanoria, and J. Shetty. A study of enterprise data lake solutions. *IRJET*, 2020.
66. B. Inmon. *Data Lake Architecture: Designing the Data Lake and avoiding the garbage dump.* Technics publications, 2016.
67. W. H. Inmon. *Building the data warehouse.* John wiley & sons, 2005.
68. Z. Ives, Y. Zhang, S. Han, and N. Zheng. Dataset Relationship Management. In *CIDR*, 2019.
69. M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. *Foundations of Data Warehouses.* Springer, 2nd edition, 2003.

70. M. Jarke and C. Quix. On Warehouses, Lakes, and Spaces: The Changing Role of Conceptual Modeling for Data Integration. In *Conceptual Modeling Perspectives.*, pages 231–245. Springer, 2017.
71. M. Klettke, H. Awolin, U. Störl, D. Müller, and S. Scherzinger. Uncovering the evolution history of data lakes. In *BigData*, pages 2462–2471, 2017.
72. P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In C. Li, editor, *PODS*, pages 61–75, Baltimore, MD,USA, 2005. ACM.
73. P. G. Kolaitis, R. Pichler, E. Sallinger, and V. Savenkov. Nested dependencies: structure and reasoning. In *PODS*, pages 176–187. ACM, 2014.
74. B. Kolev, P. Valduriez, C. Bondiombouy, R. Jiménez-Peris, R. Pau, and J. Pereira. Cloud-MdsQL: querying heterogeneous cloud data stores with a common language. *Distributed and Parallel Databases*, 34(4):463–503, 2016.
75. J. LeFevre, J. Sankaranarayanan, H. Hacigümüs, J. Tatemura, N. Polyzotis, and M. J. Carey. MISO: souping up big data query processing with a multistore system. In *SIGMOD*, pages 1591–1602, 2014.
76. O. Lehmberg and C. Bizer. Stitching web tables for improving matching quality. *PVLDB*, 10(11):1502–1513, 2017.
77. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246. ACM, 2002.
78. D. Lindstedt and K. Graziano. *Super charge your data warehouse: invaluable data modeling rules to implement your data vault.* CreateSpace, 2011.
79. D. Lindstedt and M. Olschimke. *Building a scalable data warehouse with data vault 2.0.* Morgan Kaufmann, 2015.
80. J. Lu and I. Holubová. Multi-model databases: a new journey to handle the variety of data. *ACM Computing Surveys (CSUR)*, 52(3):1–38, 2019.
81. A. Maccioni and R. Torlone. Crossing the finish line faster when paddling the data lake with kayak. *PVLDB*, 10(12):1853–1856, 2017.
82. A. Maccioni and R. Torlone. KAYAK: a framework for just-in-time data preparation in a data lake. In *CAiSE*, pages 474–489. Springer, 2018.
83. C. Madera and A. Laurent. The next information architecture evolution: the data lake wave. In *MEDES*, pages 174–180, 2016.
84. M. N. Mami, D. Graux, S. Scerri, H. Jabeen, and S. Auer. Querying data lakes using spark and presto. In *WWW*, pages 3574–3578, 2019.
85. C. D. Manning, P. Raghavan, and H. Schütze. Probabilistic information retrieval. *Introduction to Information Retrieval*, pages 220–235, 2009.
86. B. Marnette, G. Mecca, and P. Papotti. Scalable Data Exchange with Functional Dependencies. *PVLDB*, 3(1):105–116, 2010.
87. 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 *DASC*, volume 2017-Septe, pages 1–10. IEEE, 2017.
88. R. Marty. *The Security Data Lake.* 2015.
89. C. Mathis. Data Lakes. *Datenbank-Spektrum*, 17(3):289–293, 2017.
90. H. Mehmood, E. Gilman, M. Cortés, P. Kostakos, A. Byrne, K. Valta, S. Tekes, and J. Riekki. Implementing Big Data Lake for Heterogeneous Data Sources. In *ICDE Workshops*, pages 37–44, 2019.
91. R. J. Miller. Open Data Integration. *PVLDB*, 11(12):2130–2139, 2018.
92. N. Miloslavskaya and A. Tolstoy. Big data, fast data and data lake concepts. *Procedia Computer Science*, 88:300–305, 2016.
93. A. A. Munshi and Y. A.-R. I. Mohamed. Data Lake Lambda Architecture for Smart Grids Big Data Analytics. *IEEE Access*, 6:40463–40471, 2018.
94. F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost, and R. J. Miller. Organizing Data Lakes for Navigation. In *SIGMOD*, pages 1939–1950, 2020.
95. F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena. Data Lake Management: Challenges and Opportunities. *PVLDB*, 12(12), 2019.
96. F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.
97. A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. *TODS*, 32(1):4, 2007.
98. I. D. Nogueira, M. Romdhane, and J. Darmont. Modeling Data Lake Metadata with a Data Vault. In *IDEAS*, pages 253–261, 2018.
99. D. E. O'Leary. Embedding AI and Crowdsourcing in the Big Data Lake. *IEEE Intelligent Systems*, 29(5):70–73, 2014.
100. K. W. Ong, Y. Papakonstantinou, and R. Vernoux. The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases. *CoRR, abs/1405.3631*, 2014.
101. B. Otto and M. Jarke. Designing a multi-sided data platform: findings from the international data spaces case. *Electronic Markets*, 29(4):561–580, 2019.

102. P. Patel, G. Wood, and A. Diaz. Data Lake Governance Best Practices. *The DZone Guide to Big Data - Data Science & Advanced Analytics*, 4:6–7, 2017.
103. J. Pennekamp, Jan Pennekamp, R. Glebke, M. Henze, T. Meisen, C. Quix, et al. Towards an Infrastructure Enabling the Internet of Production. In *ICPS*, pages 31–37, 2019.
104. R. Pichler and S. Skritek. The complexity of evaluating tuple generating dependencies. In *ICDT*, pages 244–255. ACM, 2011.
105. R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. *PVLDB*, 5(10):908–919, 2012.
106. C. Quix and R. Hai. *Data Lake*, pages 1–8. Springer, Cham, 2018.
107. C. Quix, R. Hai, and I. Vatov. GEMMS: A Generic and Extensible Metadata Management System for Data Lakes. In *CAiSE forum*, 2016.
108. C. Quix, R. Hai, and I. Vatov. Metadata Extraction and Management in Data Lakes With GEMMS. *CSIMQ*, (9):67–83, 2016.
109. R. Ramakrishnan, B. Sridharan, et al. Azure Data Lake Store: A Hyperscale Distributed File Service for Big Data Analytics. In *SIGMOD*, pages 51–63, 2017.
110. P. Russom. Data lakes: Purposes, practices, patterns, and platforms. *TDWI white paper*, 2017.
111. A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, pages 817–828, 2012.
112. P. Sawadogo and J. Darmont. On data lake architectures and metadata management. *Journal of Intelligent Information Systems*, pages 1–24, 2020.
113. 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*, pages 440–451. Springer, 2019.
114. C. Sengstock and C. Mathis. SAP HANA Vora: A Distributed Computing Platform for Enterprise Data Lakes. In *BTW*, page 521, 2017.
115. B. Sharma and A. LaPlante. *Architecting Data Lakes*. O'Reilly Media, Inc., 2016.
116. A. Singh. Architecture of Data Lake. *IJSRCSE*, 5(2):411–414, 2019.
117. M. Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.
118. T. J. Skluzacek, R. Kumar, R. Chard, G. Harrison, P. Beckman, K. Chard, and I. Foster. Skluma: An extensible metadata extraction pipeline for disorganized data. In *e-Science*, pages 256–266. IEEE, 2018.
119. B. Stein and A. Morrison. The enterprise data lake: Better integration and deeper analytics. *PwC Technology Forecast: Rethinking integration*, 1(1-9):18, 2014.
120. I. Suriarachchi and B. Plale. Crossing analytics systems: A case for integrated provenance in data lakes. In *e-Science*, pages 349–354, 2016.
121. R. Tan, R. Chirkova, V. Gadepally, and T. G. Mattson. Enabling query processing across heterogeneous data models: A survey. In *BigData*, pages 3211–3220. IEEE, 2017.
122. B. Ten Cate and P. G. Kolaitis. Structural characterizations of schema-mapping languages. In *ICDT*, pages 63–72, 2009.
123. B. Ten Cate and P. G. Kolaitis. Structural characterizations of schema-mapping languages. *Communications of the ACM*, 53(1):101–110, 2010.
124. I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino. Data Wrangling: The Challenging Yourney from the Wild to the Lake. In *CIDR*, 2015.
125. C. Walker and H. H. Alrehamy. Personal Data Lake with Data Gravity Pull. In *BDCloud*, pages 160–167, 2015.
126. M. Wibowo, S. Sulaiman, and S. M. Shamsuddin. Machine Learning in Data Lake for Combining Data Silos. In *DMBD*, volume 10387, pages 294–306. Springer, 2017.
127. M. e. a. Wilkonson. Commentary: The FAIR Guiding Principles for scientific data management and stewardship. *NATURE Scientific Data*, 3(160018), 2016.
128. C. Yu and L. Popa. Constraint-based XML query rewriting for data integration. In *SIGMOD*, pages 371–382. ACM, 2004.
129. Y. Zhang and Z. G. Ives. Juneau: Data Lake Management for Jupyter. *PVLDB*, 12(12):1902–1905, 2019.
130. Y. Zhang and Z. G. Ives. Finding Related Tables in Data Lakes for Interactive Data Science. In *SIGMOD*, pages 1951–1966, 2020.
131. E. Zhu, D. Deng, F. Nargesian, and R. J. Miller. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD*, pages 847–864, 2019.
132. E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH ensemble: internet-scale domain search. *PVLDB*, 9(12):1185–1196, 2016.
133. M. Zhu and T. Risch. Querying combined cloud-based and relational databases. In *CSC*, 2011.
134. P. Zikopoulos, D. DeRoos, C. Bienko, R. Buglio, and M. Andrews. *Big Data Beyond the Hype: A Guide to Conversations for Todays Data Center*. McGraw-Hill Education, 2014.