Information Systems 127 (2025) 102460

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

Data Lakehouse: A survey and experimental study

Ahmed A. Harby [a,b,*], Farhana Zulkernine [a]

[a] School of Computing, Queen's University, Kingston, Ontario, Canada
[b] Computer Engineering Department, Arab Academy for Science, Technology and Maritime Transportation (AASTMT), Heliopolis, Cairo, Egypt

ARTICLE INFO

Keywords:
Big data
Data warehouse
Data lake
Data lakehouse

ABSTRACT

Efficient big data management is a dire necessity to manage the exponential growth in data generated by digital information systems to produce usable knowledge. Structured databases, data lakes, and warehouses have each provided a solution with varying degrees of success. However, a new and superior solution, the data Lakehouse, has emerged to extract actionable insights from unstructured data ingested from distributed sources. By combining the strengths of data warehouses and data lakes, the data Lakehouse can process and merge data quickly while ingesting and storing high-speed unstructured data with post-storage transformation and analytics capabilities. The Lakehouse architecture offers the necessary features for optimal functionality and has gained significant attention in the big data management research community. In this paper, we compare data lake, warehouse, and lakehouse systems, highlight their strengths and shortcomings, identify the desired features to handle the evolving challenges in big data management and analysis and propose an advanced data Lakehouse architecture. We also demonstrate the performance of three state-of-the-art data management systems namely HDFS data lake, Hive data warehouse, and Delta lakehouse in managing data for analytical query responses through an experimental study.

1. Introduction

Big data has a huge influence on our daily lives. The exponential growth in data from the Internet of Things (IoT) i.e., the connected digital artifacts, has raised the 5 V challenges of big data management namely volume, velocity, variety, veracity, and value [1,2]. Massive volumes of data are progressively being processed and stored in cloud data storage systems. Managing, analyzing, and querying data of multiple modalities (audio, video, sensor, text, hybrid) ingested at different frequencies and speeds from a wide variety of IoT, Cloud, and other digital sources serve as the key driving force behind the development of various big data processing and management architectures [3]. The value of data is measured by the number of relevant insights that can be derived from it using data management, linking, analysis, and knowledge extraction techniques. Machine Learning (ML) and Artificial Intelligence (AI) approaches are increasingly being used to correlate data features, extract and represent knowledge to improve decision-making [4,5]. Researchers have proposed a variety of hierarchical and scalable systems empowered by new technologies and expanded the traditional database systems [6] to enable storage, management, analysis, and querying of structured, unstructured, and semi-structured big and streaming data using architectures such as data warehouse (DW) [7] and data lake (DL) [8].

In the 1990s, DWs were built to perform extract, transform, and load (ETL) operations to merge data from numerous sources into a data grid that could be studied with additional analytical tools [7,9]. As the volume of high-speed hybrid unstructured data grew, DLs were proposed to address dynamic and scalable storage needs of unstructured data of different modalities such as numeric, image, text, and hybrid data [8]. DLs harvest and process data at a high speed utilizing innovative data ingestion technology that can handle batch, near real-time, and real-time data storage and processing [6]. Furthermore, unlike DWs, DLs provide metadata and knowledge extraction functionality [10,11] but lack analytical query support. In contrast to the conventional approach to developing separate DLs and DWs, researchers have recently proposed the concept of the data Lakehouse (LH) [12], a novel paradigm shift, to avoid data swamps in the traditional DL systems, and improve data organization and storage management for efficient query processing.

1.1. Contribution

In this paper, we explore the state-of-the-art (SOTA) DW, DL, and LH technology.

* Corresponding author at: School of Computing, Queen's University, Kingston, Ontario, Canada.
E-mail addresses: Ahmed.Harby@queensu.ca (A.A. Harby), farhana.zulkernine@queensu.ca (F. Zulkernine).

https://doi.org/10.1016/j.is.2024.102460
Received 10 March 2024; Received in revised form 16 July 2024; Accepted 12 September 2024
Available online 26 September 2024

- We provide a comprehensive analysis of the strengths and weaknesses of the three storage systems, focusing on their architecture, metadata storage for data organization and linking, security aspects, and data processing capabilities. Fig. 1 shows an overview and evolution of DW, DL, and LH architectures. We summarized existing work based on the features, functionality, and architecture [13]. To demonstrate the impact of the features and architectures, we provide an experimental study to evaluate specific functionalities of the three types of storage systems.
- We perform a comparative analysis of the LH service vendors based on a conceptual model proposed by Harby et al. [13], which can be effectively used as a standard for a comprehensive understanding of the architecture of various implemented LH systems and service providers. We identify the LH systems that follow this standard and evaluate them for a user portfolio to determine the most optimal fit for specific use case scenarios.
- Based on the above comparative study, we propose a baseline conceptual architecture model for the LH systems to improve metadata extraction during data ingestion, enhance storage and metadata management, and enable efficient data linking for online analytical processing (OLAP) queries. The proposed data lakehouse architecture integrates comprehensive features from data warehouses and data lakes to create a unified, efficient, and secure data management system.
- Finally, we implement and demonstrate three widely recognized SOTA DW, DL, and LH systems due to their robust feature sets, strong community support, and flexibility. These systems enable the design of comprehensive and scalable experiments involving large-scale data storage, processing, and analysis, while also providing the necessary tools to ensure data integrity, performance optimization, and real-time analytics capabilities. Additionally, we compare their performances using 4 different analytical queries on the IMDb dataset. We implement HDFS as a DL, Hive as a data DW, and Delta LH as the LH. The experimental results from our study demonstrate that Delta LH has significant benefits over the others in real-world situations, depending on the workload, data size, and query patterns.

The rest of the paper is organized as follows. DW, DL, and LH are described in Section 2 in light of the existing work in the literature which emphasizes the increasing need for real-time data lineage and analytics using ML using a LH architecture. Section 3 presents a comparison of the features and performances of existing DW and DL technology to identify the necessary features for a data lineage system. Section 4 presents a comparison of the existing LH systems based on the standard features. We illustrate our implementation of popular DL, DW, and LH systems in Section 5. In Section 6, we present an experimental study to analyze the performances of three SOTA DW, DL, and LH systems and discuss the results. Finally, Section 7 concludes the paper with a list of future research directions.

2. Overview and related work

Based on the literature, a summary of the progression of DW, DL, and LH technology is given below.

2.1. Data warehouse

Data warehouses (DW) serve as the central repository for an organization. Data from various relational databases are extracted, transformed, integrated, and then stored as a linked data cube in DWs as shown in Fig. 1(a) [14] to expedite data access and execution of analytical queries requiring data linking from multiple sources. Therefore, DWs are great for decision-making based on BI needs. Custom applications are developed to extract and transform necessary business and transactional data, which are then loaded into the DW at regular intervals [15]. In DW, data is neatly organized and customized using SQL clients and analytical tools to cater to the specific needs of business analysts, data engineers, and decision-makers [16]. However, DWs are mainly optimized for predefined analytical SQL queries and not equipped to support advanced ML or AI algorithms. The platform is commonly utilized by business users and decision-makers rather than data scientists.

DWs refer to collections of nonvolatile, integrated, time-variant, and subject-oriented data, which are essential for management to make informed decisions through data mining [17,18]. In recent years, the concept of Data Marts (DMs) for OLAP queries has emerged as a smaller and more focused subset of data than DWs [19–21]. For example, a DM may contain information about a single branch or product line, while a DW provides data about all products. However, extracting DMs from an enterprise-wide information system can be costly and cumbersome due to the existing widespread implementation of DWs. We also examined research studies that focused on modeling and categorizing DWs [22]. It can be challenging for an enterprise to find an optimal storage system that meets all of the needs as different DW vendors provide different ETL and data integration tools, and operational features for data quality improvement, master data management, metadata management, reference data management, and big data management.

2.2. Data lake

A data lake (DL) [23] is a central repository that is highly scalable and can store structured, semi-structured, or unstructured raw and granular data from multiple sources as shown in Fig. 1(b). In 2010, Dixon introduced the concept of DLs [23]. DLs support diverse user needs and avoid the costly and time-consuming preprocessing required by data warehouses, which need predefined structures and extensive data transformation. DLs enable users to perform ad-hoc analysis and gain insights from raw data without the need for rigid schemas [24].

On the other hand, data warehouses (DWs) require data to be cleaned, transformed, and loaded into a structured format before analysis, DLs allow raw data to be ingested as-is [25]. This flexibility makes DLs particularly useful for big data applications where data comes in different formats such as logs, sensor data, and social media feeds [26,27]. Additionally, the identifiers and metadata tags associated with the data stored in a DL facilitate faster data retrieval [28,29]. DLs can be configured using cost-effective and expandable hardware clusters, either on-premises or in the cloud to provide easily accessible and scalable storage for large amounts of data without concerns about storage capacity [30]. Additionally, DLs can perform advanced analytics, limited machine learning, and real-time data processing, which are essential for modern data-driven decision-making [22,31]. Existing research discusses various aspects of DL architectures, covering functions such as data storage, modeling, metadata management, and governance [32–36].

In past years, there has been a concerted effort to develop DL architectures, which can be broadly classified into pond [10] and zone [30,37,38] architectures, each with unique organizational principles. The pond architecture organizes ingested data based on its status and usage, initially storing raw data in a raw data pond. Subsequently, the data is transformed and moved to specific ponds such as analog, application, or textual data ponds, before finally being stored in an archival data pond for long-term retention. This approach ensures data is appropriately processed and prepared for analytical tasks. Conversely, the zone architecture of DLs divides the dataset lifecycle into distinct stages, including loading, quality checking, storing raw and cleaned data, and using the data for analysis [26]. This segmentation allows each stage of the data lifecycle to be managed independently, facilitating more controlled and systematic data processing. Lately, a proposed function-oriented DL architecture addresses the limitations of pond and zone architectures by integrating specific functional tiers: ingestion, maintenance, and exploration [39]. The ingestion tier handles data import from various

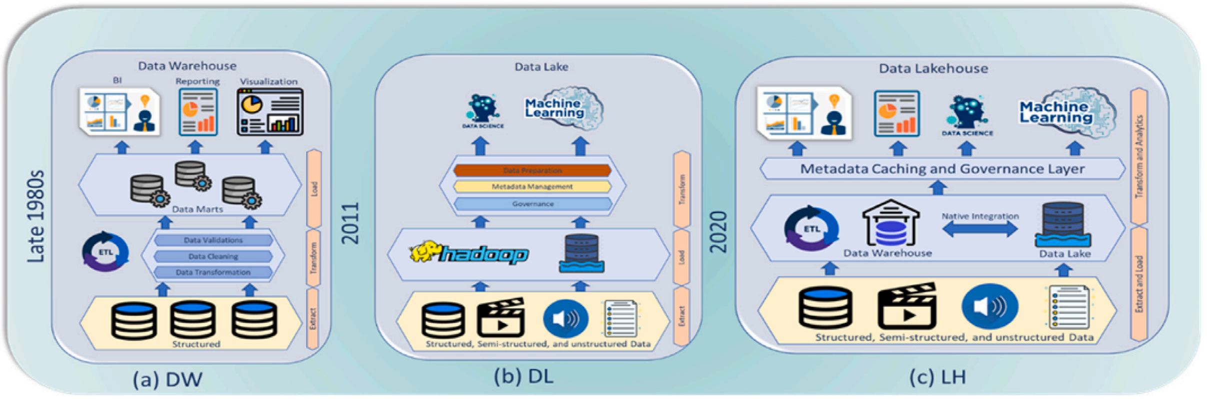


Fig. 1. (a) The DW ingests structured data in diverse formats, and performs ETL operations before storing data in Data Marts for visualization, analysis, and reporting. (b) DL ingests structured, semi-structured, and unstructured data and extracts metadata before loading it into scalable storage to later transform the same (ELT) for analysis and reporting. (c) LH integrates the features of both DW and DL to benefit from both approaches.

sources and metadata extraction. The maintenance tier manages and organizes ingested datasets, preparing them for querying and analytics. The exploration tier supports user access and querying, enabling efficient data discovery and analysis. This architecture aims to provide a comprehensive view of necessary functions in a DL, facilitating modular and reusable implementations. By categorizing functions based on their necessity at different stages of the data workflow, this architecture enhances the organization and usability of DLs, supporting a wide range of data management and analysis tasks.

DL architecture emerged as a solution to efficiently process and analyze data by utilizing a staging repository like the Hadoop Distributed File System (HDFS) in the Hortonworks Data Platform (HDP) [18]. Data can be ingested in various ways using tools like NiFi [40] and Kafka [41]. Sqoop [42] is used for transferring bulk data between Hadoop and structured data stores. The ingested content can be analyzed using big data and analytics tools such as Spark [43]. HBase and Hive are used for processing SQL queries in a Hadoop cluster [44]. Hive can be used to insert data into HBase and join data from different sources. Spark is used for high-performance analytics of both batch and streaming data [45]. Analytics results can be visualized through display tools like Zeppelin [46]. DL architecture allows for the transport of data from a traditional database to Hadoop components to address the limitations of traditional computing. In addition to incorporating significant steps to facilitate data ingestion and higher-level analytics, DLs provide a unified data management and analytics platform. The authors mentioned metadata definition as potential future work to be performed when importing relational data into HDFS using Sqoop and storing the original data in different NoSQL databases to compare computation time.

DLs, while offering powerful data storage and processing capabilities, have certain limitations. These include a lack of robust data governance, which can lead to data quality and consistency issues, security and access control risks, the potential for creating data silos, and the risk of becoming a data swamp if data is dumped without structure or governance [22]. Additionally, DLs can lead to increased storage and processing expenses, and the data needs to be cleansed, tagged, and structured before use, which can be time-consuming [47].

2.3. *Data lakehouse*

A data lakehouse (LH) is a platform that combines the best features of both DL and DW as shown in Fig. 1(c) [48]. It offers scalability, versatility, and cost-effectiveness in storing raw and unstructured data of DLs, and the data quality, governance, and efficient query performance capabilities of DWs. DL architecture aims to provide a single platform that can handle various data types, processing needs, and analytical queries while maintaining data integrity and usability.

LH allows for flexible and cost-efficient storage of big data with a structured organization and tools for data integration, processing, metadata management, and analytics [48–50]. Existing LH systems such as Apache Iceberg [51] and Delta LH [52] can manage data and ensure integrity and consistency. While metadata layers in these systems can improve data management, they do not always guarantee good SQL performance. By combining DL and DW, data lock-in, staleness, reliability, and cost challenges can be addressed. However, the integration of the two would increase the overall complexity of the system and data management to support different data types, schemas, and SQL dialects. DWs are incapable of handling unstructured data such as images, sensor data, and documents whereas LHs are great for storing large binary data like images, sensor data, and documents due to the inherent support for structured, semi-structured, and unstructured data. Additionally, LHs were used for biomedical research and health data analytics that support query, accessibility, interoperability, and reusability (FAIR) standards, health insurance portability and accountability (HIPAA) regulations, and institutional review boards (IRB) protocols [53]. Begoli et al. [53] proposed implementing automated tools to optimize data accessibility and retrieval during data ingestion processes, representing a potential future direction for the research field.

In recent times, there has been a gradual shift from traditional DLs to LH systems such as Apache Hudi, Apache Iceberg, and Delta LH [54,55]. These storage systems offer simplified management of large datasets and accelerated SQL workloads, along with fast file access for machine learning. A tool called LHBench [54] was proposed by Jain et al. and was used to compare the performance and features of Apache Hudi, Apache Iceberg, and Delta LH systems based on the various design aspects. It can analyze the designs and provide a detailed report to help organizations choose the best storage system based on their specific requirements. Harby et al. [13] highlighted the shortcomings of DL and DW in extracting, processing, integrating, and storing the high volume, velocity, and variety of data to facilitate efficient query performance. DLs can turn into data swamps due to the delay in processing raw data caused by evolving data format and content, inconsistency, and errors in data, and the need to maintain recency by recurring metadata extraction. The advantages of DWs, DLs, and the current LH systems are discussed below to propose a new LH architecture.

We outlined the desired features of a data LH system in our previous work as listed below [13]. The LH architecture should

- Offer accessibility, flexibility, and affordability.
- Enable fast data ingestion, knowledge extraction, and cost-effectiveness.
- Support Knowledge graphs to improve the management and linking of metadata and thereby, advance analytical capabilities by incorporating relevant information about linked data.

- Ensure accuracy and efficiency with the data undergoing a thorough transformation process involving cleaning, preprocessing, and summarizing prior to being loaded into memory or storage.

Advanced techniques should be offered for knowledge extraction and linked data exploration. Snowflake or Amazon S3 were recommended for data storage, while Egeria, Apache Flink, and Apache NiFi were recommended for data ingestion [13]. The problems of data swamps and enabling faster execution of OLAP queries can be addressed by developing:

- An advanced and intelligent data ingestion pipeline with knowledge extraction and linking capabilities.
- A data profiling method to capture significant and anomalous data to reduce storage footprint while enhancing data linking capabilities.
- A knowledge graph to augment the metadata management capabilities in LHs.

Recently, Errami et al. proposed a spatial LH for the evolution of data management systems to handle spatial big data effectively [55]. Traditional spatial DWs, which integrate spatial dimensions for decision-making, have faced limitations in scalability and supporting IoT data streams. This led to the emergence of DLs, which, although offering flexibility and cost-effective storage, encountered consistency issues. To overcome these challenges, the LH paradigm combines the reliability and governance of DW with the scalability and flexibility of DL. The paper explores the components and best practices for constructing a LH optimized for spatial big data. It discusses data ingestion, storage, analytical processing, and indexing, emphasizing the need for a robust architecture that can handle the unique requirements of spatial data. By addressing these challenges, the LH aims to provide a comprehensive solution for managing and analyzing spatial big data efficiently.

2.4. *Comparative overview of DW, DL, and LH*

Management of data using LH is rapidly progressing to become an industry standard due to the flexibility, scalability, analytic capabilities, and BI potential of both DLs and DWs. Data LHs are an attractive storage option for organizations with a growth and cost-sensitive mindset [22]. Table 1 summarizes the major differences among the three data management models namely the DW, DL, and LH, based on the most relevant criteria for meeting industry standards. The primary industrial metrics are cost and quality along with the atomicity, consistency, isolation, and durability (ACID) criteria to ensure that database transactions are completed on time. We also consider how the storage models can be customized to meet the requirements of the developers. Additionally, we identify usability problems to gather qualitative and quantitative data to provide a summary of compatible tools that are currently available for the various storage systems. By utilizing the best of both DW and DL, the LH can provide a storage system that meets most of its customer needs.

3. Taxonomy of DW, DL, & LH

We categorize and compare different architectural and functional features of DW and DL systems to identify the necessary features for our proposed LH system. DWs are commonly used for business intelligence and reporting, historical analysis, and structured data analytics. DLs are well-suited for real-time data ingestion, big data processing, and data transformation. Data scientists and analysts working with large amounts of data can make great use of flexible and scalable DLs. By combining the necessary features of DW and DL, we propose a conceptual feature model for a LH system.

3.1. *Categorization based on architecture*

The key set of functionalities demonstrated by the DW, DL, and LH includes data extraction, transformation, loading, and management (storage, indexing, querying, archival). The need for analytical functionality is increasing to generate insights for business decisions in near real-time.

Fig. 1 shows how and at what stage each of the systems implements extract, transform, and load (ETL) functionalities, and in some cases, execute analytics. Analytical capabilities are not a part of the storage system for DW and DL but are suggested to be incorporated in the LH. Each of these functionalities can have regular and advanced capabilities and that is where the systems mainly differ. Fig. 2(a) shows the different categories of DW architectures. A description and comparison of each architecture of DW is provided in Table 2 [56]. DLs have three main architectures as mentioned in Section 2.2: pond, zone, and Functional. The pond architecture organizes data by its status and use, storing it in different ponds like raw data, application data, and archival data ponds [10]. The zone architecture, on the other hand, manages data through various stages such as loading, quality checking, and analysis, ensuring each stage is handled separately for better control. Furthermore, the function-oriented architecture improves upon these by dividing the DL into ingestion, storage, processing, and access tiers [39]. The ingestion tier imports data and extracts metadata to be stored in the data storage tier, Data processing organizes data for querying, and the data access tier allows user access for analysis. This design offers a structured, modular approach, enhancing DL functionality and usability as illustrated in Fig. 2(b). A description and comparison of DL architectures are provided in Table 3.

3.2. *Proposed features*

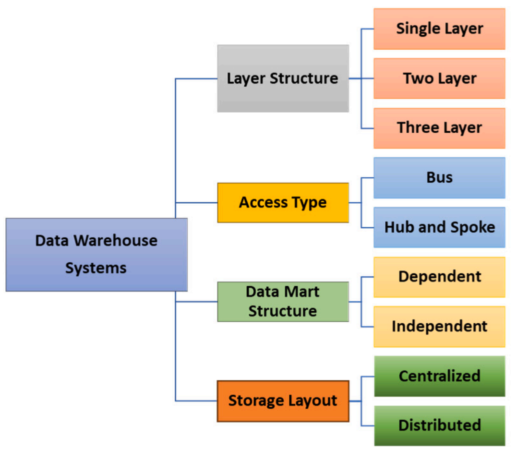
We propose a comprehensive LH architecture that offers data storage solutions with high accessibility, scalability, and cost-effective options like Amazon S3. Robust data ingestion mechanisms support batch, real-time, and streaming data, addressing challenges with tools such as Apache NiFi and Apache Flink. Efficient data transformation and loading emphasize normalization, cleaning, and validation to enhance decision-making and cost reduction. Advanced metadata management provides semantic annotation, data linking, and system comparisons to tackle data governance and quality issues. Enhanced data access integrates OLAP, OLTP, and robust analytics methods. Strengthened privacy and security measures ensure compliance, encryption, and highlight blockchain and quantum computing for improved data security. Fig. 3(a) and (b) show the necessary features of both DW and DL to be incorporated in a standard LH architecture, which is shown in Fig. 3(c). The features are described below.

- Data storage systems should have high accessibility, scalability, availability, hierarchical distribution, and comprehensive data administration.
- Data ingestion should support all types of data including batch, real-time, and streaming data.
- Transformation and loading of data are essential after the extraction of metadata. Data transformation is required to enable data normalization, cleaning, aggregation, loading with validation, and data capturing.
- Metadata management is required to offer semantic annotation for creating data ontologies, enabling data linking, and supporting data heterogeneity, versioning, and indexing.
- Data access includes both online analytical processing OLAP and online transactional processing (OLTP) queries, reporting, and getting business insights.

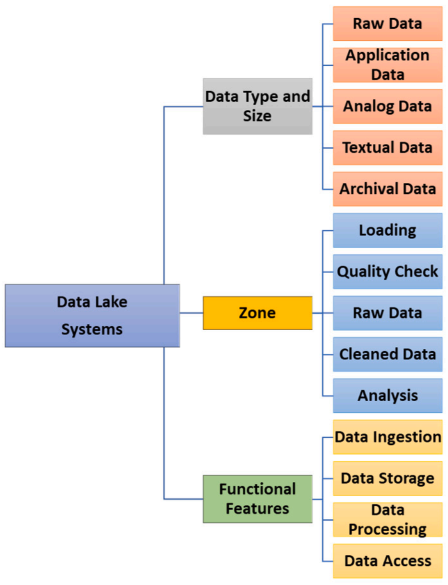
We discuss relevant tools to support these functionalities with a view to preserving coherence.

Table 1
Comparison of DW, DL, and LH.

Criteria	Data warehouse	Data lake	Data lakehouse
Key focus	Data analytics and business intelligence	Data exploration, big data analytics, and serves as single source of truth for different types of data (e.g., logs, documents, media)	Structured Data analytics and transaction management
Data type	Structured	Structured, semi-structured, and unstructured	Structured, semi-structured, and unstructured
Cost	Expensive and time-consuming	Inexpensive, quick, and adaptable	Inexpensive, quick, and adaptable
Structure	Unconfigurable	Customizable	Customizable
Schema	Defined before the data is stored (schema-on-write)	Developed after the data is saved (schema-on-read)	Developed after the data is saved (schema-on-read)
Schema enforcement	Strict	Flexible	Flexible
Usability	Users can easily access and report data	Complex to analyze vast amounts of raw data without classification and cataloging tools	Combines the structure and simplicity of a DW with the broader use cases of a DL
ACID conformity	Guarantees high levels of integrity, data is recorded in an ACID-compliant way	Non-ACID compliant, updates and deletes are complex procedures	ACID-compliant to ensure consistency during concurrent read/write operations
Quality	High	Risk of low quality (data swamps) if not managed properly	High
Data governance	Built-in governance features for data quality and integrity	Requires additional tools for effective governance	Hybrid approach, leverages data warehouse features for better governance
Scaling	Vertical scaling	Horizontally scalable	Horizontally scalable



(a) Categorization of DW systems



(b) Categorization of DL systems

Fig. 2. DW system can be categorized based on: Layer Structure, Access Type, Data Mart Structure, and Storage Layout. DL systems can be categorized based on Data Type and Size, Zone and Functional features.

3.2.1. *Data storage*

A robust storage management should support ACID (Atomicity, Consistency, Isolation, Durability) [72]. HDFS is the most used open-source framework for collecting and processing large amounts of unstructured data for DLs. There are also other cloud-based alternative systems that give greater flexibility by orders of magnitude such as the Amazon S3 storage system. Table 4 shows the most commonly used data storage systems that are compatible with DWs, DLs, and LH. Amazon S3 has the lowest cost among all, and its performance is almost double the performance of HDFS [73]. It also provides support for Databricks for data integrity. Amazon S3 is a reliable and scalable cloud-based storage service, but it has certain limitations. Storing large amounts of data can be costly and data transfer may come with additional costs. Furthermore, pricing structures can be complicated, depending on the amount of data stored. Nevertheless, organizations can overcome these limitations by paying attention to design and architecture, as well as by utilizing complementary AWS services and tools to optimize their storage options.

3.2.2. *Data ingestion*

Due to the continuing growth in the volume of data generated by numerous data sources and the IoT, extracting data for the target system cost-effectively offers a challenging problem. The key challenges in data ingestion are listed as follows.

- The increasing velocity and veracity pose greater challenges in prioritizing data processing.

Table 2
Existing categories of DW architectures.

Architecture	Category	Description
Layer	Single-Layer [57]	The intermediate layer of the DW creates a multidimensional model of operational data, effectively reducing data redundancy by storing only a minimal amount of variables.
	Two-Layer [58,59]	Retrieve and integrate data using ETL tools to load into DWs or data marts. Analyze with reporting, OLAP, and other data mining tools.
	Three-Layer [60,61]	The three-tier architecture consists of the source layer, reconciliation layer, and DW layer, which includes both data warehouses and data marts. The reconciliation layer acts as a bridge between the source and DW, establishing a consistent reference model for businesses. It also assists in distinguishing the difficulties of extracting and integrating source data from the warehouse and the advantages of refining and integrating data within the warehouse.
Access	Bus [62]	A significant contrast between an independent data mart and bus architecture is that while data marts are logically linked, they provide a more comprehensive perspective of information across the organization.
	Hub-and-Spoke [63]	In a hub-and-spoke architecture, data sources are matched with data marts. The data marts contain normalized atomic data, which then feeds a chain of multidimensional data marts. This system enables scalability, extensibility, and the retrieval of vast amounts of data.
Data mart	Dependent [64–66]	The independent data mart architecture consists of separate data marts that are planned and developed independently without integration. However, this approach may lead to inconsistencies in data descriptions.
	Independent [65–67]	The architecture of independent data marts comprises separate units that are planned and developed individually without integration. These data marts often have inconsistencies in data descriptions, as well as varying dimensions and sizes, which can pose challenges in data analysis.
Storage layout	Centralized [68]	The hub-and-spoke architecture is quite similar to this, but it does not include data marts. Instead, it comprises a single unified data warehouse that integrates both data and data marts.
	Distributed [69–71]	The architecture incorporates every data warehouse and data mart either conceptually or physically through shared keys, universal metadata, distributed queries, and other methods.

Table 3
Comparison of DL architectures.

Feature	Pond architecture	Zone architecture	Function-oriented architecture
Data partitioning	By status and usage	By lifecycle stages	By functional tiers
Stages/Components	Raw data pond, analog pond, application pond, archival pond	Loading zone, quality check zone, raw data zone, cleaned data zone, analysis zone	Ingestion tier, maintenance tier, exploration tier
Data processing	Sequential, status-based transformation	Stage-specific processing	Tier-specific functions for ingestion, storage, processing and access
Flexibility	Moderate	High	Very high
Focus	Status and usage	Lifecycle management	Functionality and user access
Advantages	Structured transformation pipeline	Clear separation of lifecycle stages	Comprehensive function integration, modularity
Challenges	Managing transitions between ponds	Ensuring consistent data quality across zones	Implementing detailed functional tiers

Table 4
Comparison of data storage systems.

Criteria	Amazon S3	Apache hive	Google cloud store	HDFS
Elasticity	Yes	No	Yes	No
Cost-effective	Low	High	High	High
Availability	High	High	High	High
Durability	High	High	High	High
Data integrity	Yes with databricks	Yes	Yes	Yes
Fault detection & recovery	Yes	Yes	Yes	Yes
Access control	Full	Full	Full	Not fully managed
Compatibility	DW, DL, LH	DW	DW, DL, LH	DL, DW, LH
Performance	High for low cost	High for high cost	High for high cost	High for high cost

Table 5
Data ingestion tools.

Tool	Batch	Real-time	Near real-time
Apache Kafka [74]		✓	
Apache NiFi [75]	✓	✓	✓
Amazon Kinesis [76]		✓	✓
Apache Flink [72]	✓	✓	
Apache Spark [77]	✓		✓
Apache Sqoop [78]	✓		
Apache Storm [72]		✓	
Apache Gobblin [72]	✓	✓	

- Variety in data structure and content from numerous data sources including IoT, requires continuous adaptation of data ingestion algorithms and automatic detection of data content and structure.
- Real-time BI requires low-latency data ingestion and fast learning of important information including semantic information processing for real-time BI.
- The syntax and semantics of data continue to change over time to incorporate more information, but the data ingestion applications are often not notified about the changes in the data.

Table 5 compares the data import technologies used most often based on the type of data ingested. Apache NiFi and Apache Flink handle both batch and real-time operations. Our objective is to identify

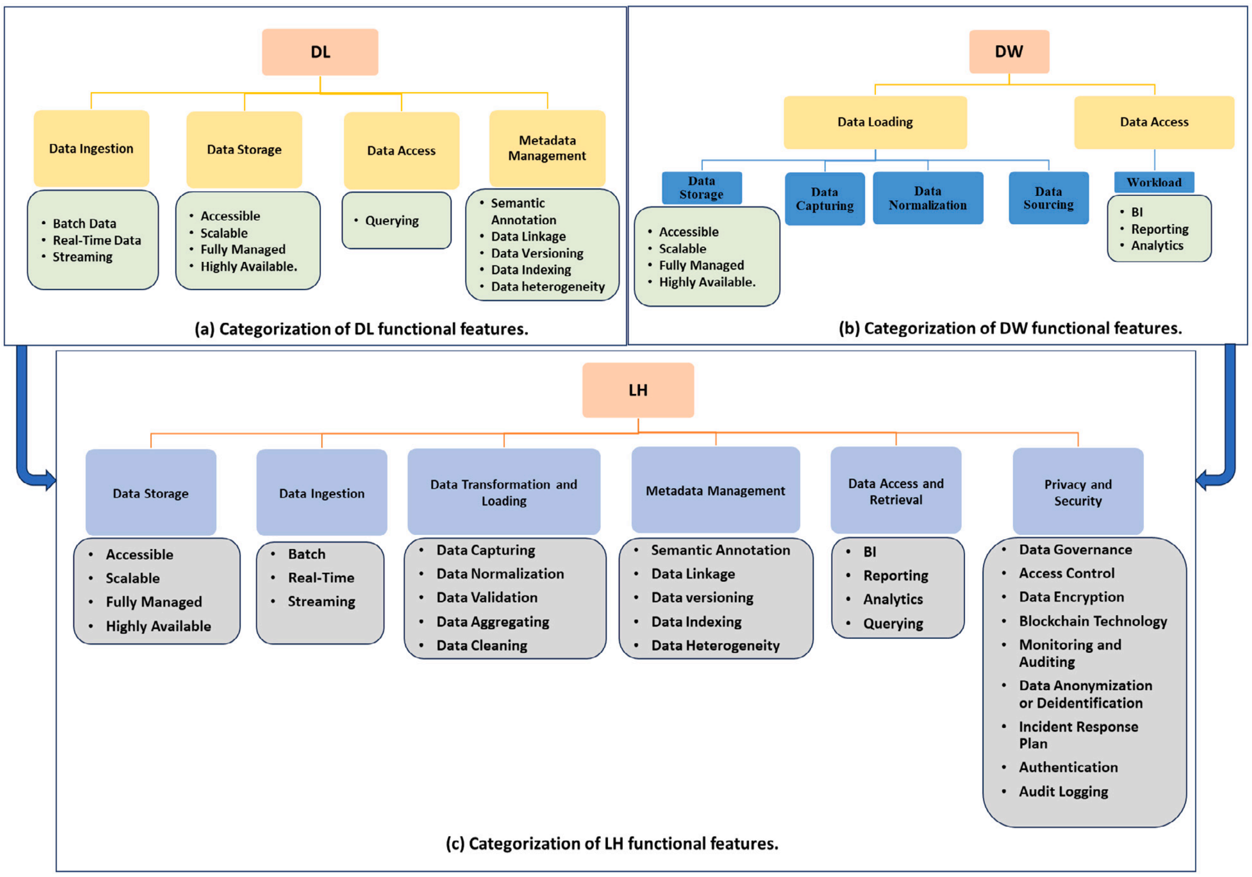


Fig. 3. The key features offered by DL systems and DW are shown in (a) and (b) respectively. We propose a LH standardized model to establish standardized features for LH systems.

Table 6
Comparison of metadata systems.

Category	System	SE	DL	DH	DV	DI	Year of publication
General metadata management	CODAL [79]	✓					2019
	Constance [80,81]	✓				✓	2016, 2018
	GEMMS [82]	✓					2016
	KAYAK [83]	✓				✓	2018
	DM [84]	✓			✓		2021
	CoreDB [85]				✓	✓	2017
	DomainNet [86]	✓	✓				2021
Domain specific applications	CLAMS [87]	✓					2019
	Ground [88]	✓			✓	✓	2017
	DLDS [89]	✓		✓			2022
	CEBA [90]	✓			✓		2022
Comprehensive metadata systems	GOODS [91]	✓	✓		✓	✓	2016
	Medal [92]	✓	✓	✓	✓	✓	2020
	GoldMEDAL [93]	✓	✓	✓	✓	✓	2021
	HANDLE [94]	✓	✓			✓	2020
	CoreKG [95]	✓	✓	✓		✓	2018
	Egeria [96]	✓	✓	✓	✓	✓	2021
	EMEMODL [97]	✓	✓	✓	✓	✓	2023

the ideal solution for sustaining high-speed processing, adjusting to data changes, and ingesting diverse types of data.

3.2.3. Metadata management

Metadata in a DL is written in a machine-readable format that follows the Common Data Model (CDM) standard. Many difficulties exist with metadata management, some of which are listed below.

- Accelerating data transformation such as digitalization and modernization is becoming increasingly important.
- Data governance, compliance requirements, and data accessibility are becoming highly significant.
- Data is getting increasingly more complex with new data sources being added to the existing sources.
- Greater emphasis is being placed on data quality and analytics that deliver business value.
- Usability is getting more emphasis to accommodate additional data interaction.

Table 6 compares several metadata management systems that are generally used in DL research and can be used in data LH as well.

Furthermore, we incorporated comparison criteria such as semantic annotation (SE), data linkage (DL), data heterogeneity (DH), versioning(DV), and data indexing(DI) that are used by most of the existing metadata management tools. Our objective is to pick a metadata model that addresses the issues while also providing a variety of features to ensure the stability of our model. In Table 6, we divide metadata models into three groups to help understand their functions. The first category, "General Metadata Management", encompasses systems designed for broad and versatile metadata management across various domains. These systems focus on providing solutions for organizing, annotating, and indexing metadata in a generic manner suitable for diverse data types and applications. The second category, "Domain-Specific Applications", includes systems tailored to specific domains or industries. These applications are designed with specialized features and functionalities to address the unique metadata management requirements of particular fields, such as scientific research, healthcare, or finance. By categorizing the metadata systems into these two groups, the classification aims to provide a clear understanding of their intended use cases and functionalities. The third category, "Comprehensive Metadata Systems", covers platforms with many features for managing metadata across various areas.

3.2.4. Data transformation and loading

The process of data transformation can be represented as data separation and data cleansing. Once businesses fail to set limits on the amount of data to be collected, what was once a well-organized DL can turn into a data swamp drowned under information they may never need. Organizations can also benefit from data preparation and cleansing by doing the following.

- Identify and fix data issues that may go undetected.
- Cut the cost of data management and analytics.
- Prepare data and eliminate duplicates for using data across different applications to optimize the Return on Investment (RoI) for BI and analytics activities.
- Help business executives and operational workers make better decisions.
- Apply intelligent data cleansing and data preparation methods by linking these into data LHs, which might assist data scientists in saving time and making better judgments according to the organization's objectives.

3.2.5. Data access and retrieval

To facilitate efficient data access in a data LH, knowledge extraction and presentation methods should be integrated into the architecture. Knowledge extraction can be achieved by correlating and connecting data within a data LH. In order to provide more robust analytics and overcome the scarcity of data, knowledge extraction should provide the following functions including visualization and communication tools.

- Classification, clustering, and other statistical data mining and deep learning libraries can be used to create data exploration and knowledge extraction workflows.
- Linking and correlation of data from multiple distributed storage can empower knowledge discovery, and metadata management systems can leverage this function. Recent knowledge graph (KG) methods [98] can be implemented to facilitate data management, linking, and metadata discovery that can in turn empower knowledge extraction.
- Summarization, reporting, and visualization tools must be incorporated into the architecture to enable user-friendly and comprehensible methods of knowledge representation and query response. Recent voice-enabled technology [99] can advance LH architectures with leading-edge communication and query technology.
- Deviation detection models will allow the system to understand changes in the incoming data and apply adaptations to data ingestion, metadata management, and knowledge extraction methods.

3.2.6. Privacy and security

Privacy Issues and Challenges: LHs contain a diverse range of data, encompassing both sensitive and non-sensitive information, such as Personally Identifiable Information (PII) and proprietary business data. However, without adequate data governance measures in place, enforcing privacy policies becomes challenging, increasing the risk of potential misuse or unauthorized access to sensitive data [100].

To enhance the security in LHs, we should take the following into consideration:

- **Data Governance:** It is crucial to ensure proper anonymization or pseudonymization of data where necessary, aligning with privacy regulations such as General Data Protection Regulation (GDPR) [101,102] or California Consumer Privacy Act (CCPA) [103,104] to safeguard individual privacy and maintain data integrity within the LH environment.
- **Access Control:** Implement strict access controls to ensure only authorized individuals can access sensitive data. This includes Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) [105].
- **Data Encryption:** Encrypt data at rest and in transit to protect it from unauthorized access [18,22,106].
- **Blockchain Technology:** Utilize blockchain technology for enhanced data security and integrity. Using the hybrid blockchain [107] can help ensure that data remains unaltered and secure, providing a reliable audit trail.
- **Monitoring and Auditing:** Regularly monitor and audit activities within the LH to detect any suspicious activities early [55].
- **Data Anonymization or Deidentification:** Anonymize sensitive data to protect privacy. This can be particularly important when dealing with PII [108,109].
- **Incident Response Plan:** Have a robust incident response plan in place to quickly address any security breaches or data privacy incidents [52]. The use of quantum computing has been recently proposed to minimize incident response time in real-world scenarios [110].
- **Authentication:** Strong authentication methods, such as Multi-Factor Authentication (MFA) [111], alongside integration with identity LH management systems, enhance security by verifying users' identities.
- **Audit Logging:** Detailed audit logging is necessary to track data access and modifications, and aid in the detection and response to suspicious activities [52].

These security measures collectively fortify the LH environment, ensuring data confidentiality, integrity, and availability.

4. Review, standardization, and comparison of LH systems

In this section, we review the existing LH systems and compare the systems based on a standard conceptual architecture. Our conceptual model helps to evaluate if the existing LH systems, both open-source and commercial, support the necessary functionality. Based on the proposed standard, we identify the main architectural and functional components of the LH systems and give an overview of the various tools and technologies that are compatible with the LH models. Our proposed conceptual standard architecture of an LH system integrates elements from both DLs and DWs, to provide a unified platform for storing, managing, and analyzing diverse data types.

4.1. Delta LH

Delta LH [112] is an open-source storage layer built on top of Apache Spark that provides ACID properties to data stored in cloud object stores such as Amazon S3 and Azure DL Storage [113]. It uses a compacted transaction log in Apache Parquet format to ensure data consistency and reliability, making it an excellent choice for most DH applications.

The Delta LH architecture consists of table metadata, data files, a partitioning scheme, a transaction log (Delta Log), checkpoints, and tombstones. The transaction log records all changes to the table, including new data insertions, schema updates, and table modifications. Checkpoints summarize the current state of the transaction log and improve metadata operations performance. Tombstones record data file deletions in the transaction log.

However, querying large-scale datasets in a LH system presents challenges. Photon, a fast query engine for Delta LH systems, is built on top of Apache Spark and operates on a columnar storage model with Adaptive Indexing [114]. Adaptive Indexing allows for faster query performance and improves scalability while dynamically determining the most efficient index based on the query workload. Photon's features such as query optimization, predicate pushdown, and projection pruning help with LH queries. Photon can easily integrate with Delta LH, providing a complete end-to-end data and analytics solution. The built-in notebook feature available in Databricks, a data analytics platform, is a collaborative workspace for data science and engineering teams.

4.2. Apache Iceberg LH

Apache Iceberg LH is a cutting-edge open-source table format system that is specifically designed to streamline the creation of a LH architecture, which is comparable to Delta LH [112]. This innovative system was created by Netflix to address the shortcomings and consistency issues that Apache Hive encounters when dealing with extensive tables. A notable distinction between Apache Iceberg and Hive is that Iceberg offers transactional and consistent data via metadata collection in manifest lists and manifest files that pertain to snapshots of tables. This enables developers to effortlessly maintain and manage vast data sets with ease.

Apache Iceberg is a cloud-based table format system that can store extremely large tables exceeding a petabyte. It was developed to overcome performance and data consistency issues that are commonly faced by Hive [54]. This architecture is based on the concept of snapshot metadata files, manifest lists, manifest files, and data files. All metadata about the table is stored in JSON format, and snapshots represent the current state of the table at a specific point in time. Manifest files provide a detailed description of a group of data files within the table, while data files store the actual table data in Apache Parquet format. Iceberg maintains a transaction log to track changes made to the table's metadata, and by using a snapshot, it ensures that the reading and writing of data are isolated. Apache Iceberg is a data management system that offers independent schema changes with no side effects [55]. By using hidden partitioning, it allows for an evolution of the partitioning specification, enabling a change in the column split without breaking the table. Moreover, Apache Iceberg is integrated with Apache Spark, enabling users to read and write data using Spark Data Frames.

4.3. Apache Hudi LH

Apache Hudi is a distributed data management framework for large-scale data storage in distributed file systems [55]. It provides fast upserts (update and insert), deletes, and incremental processing with a directory-based data management structure [115]. It offers two types of table storage, ACID compliance, fast upsert and delete functionality, optimistic concurrency control, and data layout optimization through file-level stats, rollback support, and lineage tracking.

Apache Hudi LH utilizes a directory-based structure to efficiently organize tables [54]. Each table is stored in a directory with subdirectories representing partitions. These partitions are then further divided into file groups consisting of a base file and log files. Hudi tables store metadata about each commit, enabling time travel queries, incremental data processing, and rollback support. The framework uses an index to facilitate efficient record-level operations, such as upserts and deletes. To make changes to a table, Apache Hudi employs two methods: copy-on-write or merge-on-read. Copy-on-write saves data in a Parquet file format, generating a new version of the file with each update [52]. Merge-on-read saves data using a combination of the Parquet and Avro file formats, with updates logged in delta files. Depending on the workload, either copy-on-write or merge-on-read may be the optimal choice. Copy-on-write is best for read-intensive batch downloads, while merge-on-read is better suited for streaming write-intensive workloads. Overall, Apache Hudi offers a robust set of features for managing tables, with efficient indexing and flexible options for data management.

4.4. Dremio LH

Dremio is a LH platform that provides a unified platform to simplify and accelerate data analytics [116]. It bridges the gap between DLs and DWs and allows users to query data across different sources without the need for data movement.

Dremio uses a distributed SQL engine to process queries across a cluster of nodes, enabling parallel processing and scalability for handling large datasets [117]. It also includes a virtualization layer that supports various data formats and sources, including DLs, relational databases, and cloud storage. Dremio incorporates an acceleration engine that utilizes technologies like Apache Arrow to improve query performance and reduce latency. The metadata catalog in Dremio helps organize and manage metadata related to the datasets, including schema information, data location, and optimization statistics. Dremio's user interface is designed for self-service analytics, facilitating data exploration, query building, and visualization for both technical and non-technical users. Some pros of Dremio include unified data access, performance acceleration, and compatibility with various data sources and formats.

4.5. Cloudera

Cloudera is a leading provider of LH architecture. They have developed an open LH that comprises multiple data services, including Cloudera Data Engineering (CDE), Cloudera Data Warehouse (CDW), and Cloudera Machine Learning (CML) [118]. The foundation of this LH is the Cloudera Data Platform (CDP), which provides a unified data management platform, while the Shared Data Experience (SDX) ensures consistent security, governance, and metadata management. Additionally, the Data Catalog manages metadata, and the Management Console simplifies administration.

CDP is a platform that supports multiple storage solutions, like cloud-based object storage and on-premises Hadoop Distributed File System (HDFS) and includes processing engines for distributed data processing, transformation, analytics, and machine learning. It also incorporates metadata management capabilities for cataloging and organizing metadata related to datasets, security, and governance features to ensure data privacy and compliance with regulatory requirements, and integration with a variety of data sources and tools It is built on open standards, fostering interoperability, and leverages cloud-based object storage for infinite data growth. The virtualized computing power takes advantage of cloud computing resources. However, designing and maintaining the LH can be challenging due to its distributed architecture using multiple open-source tools. Furthermore, universal designs may have lower functionality compared to specialized solutions, which is a potential downside of this approach.

4.6. AWS LH

The LH reference architecture utilizes Amazon Redshift and Amazon S3 as a unified storage layer [119]. While Redshift is designed to store structured data in dimensional schemas, S3 provides storage for structured, semi-structured, and unstructured data. With Redshift Spectrum, compressed data in various formats can be queried, and the AWS LH Formation catalog stores dataset schemas. This enables building pipelines that ingest hot data into Redshift and store historical data in S3.

For interactive queries and fast BI dashboards, Amazon Redshift is ideal for highly structured data. On the other hand, Amazon S3 is better suited for semi-structured and unstructured data that drives ML, data science, and big data processing use cases. Data can be delivered from structured sources to either the S3 DL or Amazon Redshift DW using AWS DMS and Amazon AppFlow. DataSync brings data into Amazon S3, and the processing layer components can access data in the unified LH storage layer.

In the S3 LH [120], data is ingested from the source into the landing zone, validated and stored in the raw zone, transformed, partitioned, and stored in the trusted zone, and modeled and joined with other datasets to create trusted zone datasets that are stored in the curated layer. Datasets from the curated layer are ingested into Amazon Redshift DW to serve use cases that need very low latency access or need to run complex SQL queries. Amazon Redshift is a comprehensive data warehousing service that facilitates the management of vast amounts of structured data. Its exceptional scalability enables it to handle petabyte-scale data while offering the flexibility of designing dimensional or denormalized schemas. With its capabilities, Amazon Redshift is well-suited for use in business and academic settings where large-scale data management and analysis is a priority.

4.7. Snowflake LH

Snowflake LH is a fully managed, cloud-based data platform that offers ease of use, fast, cost-effective performance, global connectivity, and fine-grained governance [121]. With its architecture, Snowflake ensures consistent governance, security, and control across all data types, whether it is structured tables or raw files, providing unified management.

Snowflake separates storage from computing, making it cost-effective and flexible, and allows users to use different languages for processing data. It provides users with near-instant, elastic scaling to support massive data volumes and concurrent users with Python, SQL, Java, and Scala. It efficiently handles multiple queries, providing high concurrency support while maintaining cost-effectiveness. One of the key benefits of Snowflake LH is its unified approach, which allows data ingestion into a managed repository while enabling read-and-write operations directly on cloud object storage. The platform also supports the Apache Iceberg table format, making it easier to manage large datasets.

However, compatibility challenges may arise with different tool versions when using an open table format. Managing billing across disparate tools can be complex, and variations in performance may occur due to the mixed workload nature of a LH. It is important to consider specific workload requirements while using Snowflake LH.

4.8. Oracle LH

The Oracle LH System [122] is a powerful combination of DL and DW capabilities that enables the processing of streaming and other types of data from diverse enterprise resources. It offers several key benefits, such as seamless data usage without the need for data replication across the DL and DW, enhanced multimodal and polyglot architecture for diverse data support, and a zero-trust security model for effective governance and security. Furthermore, it ensures efficient resource utilization by decoupling storage and computing and supports multiple compute engines for processing data for different use cases. The system is also natively managed by Oracle Cloud Infrastructure (OCI) services, which helps reduce operational overhead.

The key technologies that power the data LH on OCI include Big Data Service, Data Flow, Autonomous DW, MySQL Heatwave, Data Catalog, Data Integration, GoldenGate, Data Integrator, Streaming, and Object Storage. These services are built on high-scale, low-cost OCI Object Stores, which are integrated with OCI Data Catalog for simplified data management. They also provide scalable data ingestion and movement capabilities using Oracle Data Integration.

4.9. Google LH

The LH pattern from Google Cloud is a smart solution for common data management issues [123]. It combines the benefits of DLs and DWs, providing low-cost storage options that can be accessed by different processing engines. Dataplex offers powerful management and optimization features to manage distributed data assets, ensuring secure accessibility for analytics tools. The ultimate goal is to streamline the infrastructure and enable teams to deliver value to the business. Google Cloud has brought together the key capabilities for enterprise data operations, DLs, and DWs, simplifying management tasks and increasing value. BigQuery serves as the central hub of this ecosystem, providing a LH architecture that combines the best of both the DL and the DW without their overhead. Dataplex and Analytics Hub in Google LH unlocks the full potential of data and ensures unified governance.

The Google LH architecture reduces operational costs, simplifies transformation processes, and enhances governance. By centralizing and unifying disparate data sources and engineering efforts, it provides a more cohesive dataset for all users. Additionally, it takes advantage of BigQuery's robust storage and computing power to use views instead of materialized tables, minimizing data replication.

Table 7 compares the selected SOTA LH systems considering the following features, which are included in our proposed conceptual standard feature model: data ingestion and integration, storage options, governance and security, metadata management, query and analytics capabilities, scalability and performance, data processing and transformation, cost and pricing model, community and ecosystem, vendor support, best practices, data lineage and versioning, user-friendliness, vendor viability, performance benchmarks, and feedback from peers and industry analysts. When designing or selecting one of the above LH systems for deployment, the above aspects should be prioritized based on the desired goals and evaluated thoroughly to make an informed decision. Designing LH systems can be difficult because they use low-cost data storage systems that offer weaker guarantees [54]. These systems need to support various workloads, from large-scale DLs to interactive DWs, and be easily accessible. The protocols used to access data must support ACID transactions and high performance, which can be a challenge when running over a high-latency object-store. Standardizing LH systems is critical for seamless data management. To achieve this, data formats, metadata management, data governance, and security measures must also be standardized. It is also necessary to implement unified data catalog and discovery mechanisms, versioning, and time travel practices; ensure compatibility among different query engines; establish consistent data transformation and ETL processes and standardize monitoring and performance management. Additionally, the compatibility and integration capabilities must be evaluated when selecting vendors or tools, data quality, and validation processes need to be established, and data lifecycles and retention policies must be defined. Standardization leads to a cohesive and well-managed LH architecture that enables seamless data integration, analysis, and decision-making across different systems and data sources.

Table 7
Comparison of existing LH systems.

LH vendors	Data storage	Data ingestion	Data transformation	Metadata management	Data access & retrieval
Delta LH [52]	• Optimized for structured and semi-structured data and ACID transactions. • Compatible with AWS S3, Microsoft Azure, and Google Cloud Storage	• Supports batch and streaming ingestion • Compatible with tools such as Spark, Flink	• It supports Spark for ETL processes • Integrated Data Compaction and Data Deduplication	Supports Schema evaluation, transaction Log, serializability, data Skipping, and data versioning	SQL queries via Spark and supports Delta Table for efficient data retrieval
Apache Iceberg [98]	• Supports Structured and semi-structured data • Compatible with AWS S3	• Supports batch and streaming ingestion. • Compatible with tools such as Spark, Flink	Integrated features such as partitioning and clustering	It uses transactional metadata, schema evaluation, hierarchical file distribution, snapshot Isolation and serializability	SQL queries engines such as Presto, spark
Apache Hudi [54]	• Supports large-scale of data and efficient for structured and semi-structured data. • Compatible with AWS S3	• Supports batch and near real-time data ingestion • Compatible with tools such as Kafka, Spark, Flink	Integrated features such as Data compaction, cleaning,	Schema evaluation, Linear clustering, and index management	SQL queries using engines such as Hive, Spark, Presto, etc.
Dremio [116]	Supports various storage systems including Hadoop, S3, ADLS	• Supports batch and near real-time data ingestion • Compatible with tools such as Spark and Flink	Data Compaction	Schema evolution, data lineage, and Impact Analysis	SQL queries using Dremio's query engine
Cloud-era [118]	AWS S3, Microsoft Azure, Cloudera Operational Database (COD), and Google Cloud Storage	• Supports batch and streaming data ingestion. • Compatible with tools such as Spark	Utilizes Apache Spark for ETL processes	Centralized metadata management (Iceberg Catalog)	SQL queries using tools like Impala, Hive
AWS [119]	AWS S3	• AWS Glue used batch processing (ETL) • Amazon Kinesis for real-time data streaming and processing	Supports Spark and AWS Glue for ETL processes.	AWS Glue Data Catalog for metadata	SQL queries using Athena and other analytics such as Amazon QuickSight, Amazon Redshift, and Amazon Athena
Snowflake [121]	• Optimized cloud storage with automatic scaling • AWS S3, Microsoft Azure, or Google Cloud Platform (GCP)	• Bulk loading, streaming, and Snowpipe for real-time data • Compatible with tools such as Spark, Flink	Native support for SQL-based transformations	Metadata categorization, resource management, data governance, transactions	SQL queries via Snowflake's query engine(Single relational SQL-based query engine)
Oracle [122]	• Supports large-scale of data and efficient for structured and semi-structured data. • Uses Oracle Cloud Infrastructure (OCI)	• Supports both batch and streaming data processing offered by Oracle Cloud Infrastructure (OCI) • Supports real-time by Oracle GoldenGate	Uses Oracle Data Integrator and SQL-based transformations	Oracle Data Catalog for metadata	SQL queries using Oracle SQL and tools such as Apache Zeppelin, Jupyter
Google BigLake [123]	• Supports large-scale data and efficient for structured and semi-structured data. • Uses Google Cloud Platform (GCP)	• Supports batch and streaming ingestion. • Compatible with tools such as Spark	Uses SQL-based transformations and machine learning integration	Uses Google Cloud Data Catalog which allows quick discovery, understanding, and managing data from a single interface	SQL queries using BigQuery's query engine

5. Experimental study

In this section, we demonstrate the features and functionality of 3 different DL, DW, and LH from the ones discussed above and thereby illustrate the competitive advantages of the LH system.

5.1. Methodology

We implement a DL, DW, and a LH storage system using available popular open-source systems as discussed above. HDFS, Hive, and Delta Lake are highly suitable for experimental purposes due to the ease of integration with the Apache Hadoop and Spark ecosystems, ensuring seamless compatibility with various big data tools. This allows researchers to efficiently combine data storage, processing, and analysis in their experimental setups. Strong community support and extensive documentation make it easy for researchers to find resources, troubleshoot issues, and stay updated with the latest developments. These systems are continually evolving to make community contributions. On the other hand, HDFS offers flexible storage for diverse data types, Hive provides a familiar SQL interface for traditional data warehousing, and Delta Lake integrates features of both DLs and DWs, supporting transactional integrity and real-time processing. Being open-source, these systems are cost-effective, easy to deploy and scale without significant licensing costs, and accessible to institutions with limited budgets. Furthermore, we compare their performances in storing a real-world dataset and running both simple and complex queries against the data. In this section, we describe each of the storage systems, the analytic software used to process and ingest data to store in these systems, and the queries formulated for the experimentation. We analyze various quantitative and qualitative aspects of the query executions to evaluate the 3 storage systems and use metrics such as query response time to measure the performance of the different systems. To ensure a more accurate performance evaluation, we conduct identical queries on each system with randomized execution. This allows us to record critical performance metrics such as response times, memory consumption, and CPU utilization. These metrics are then analyzed to provide a comprehensive assessment of the system's performance.

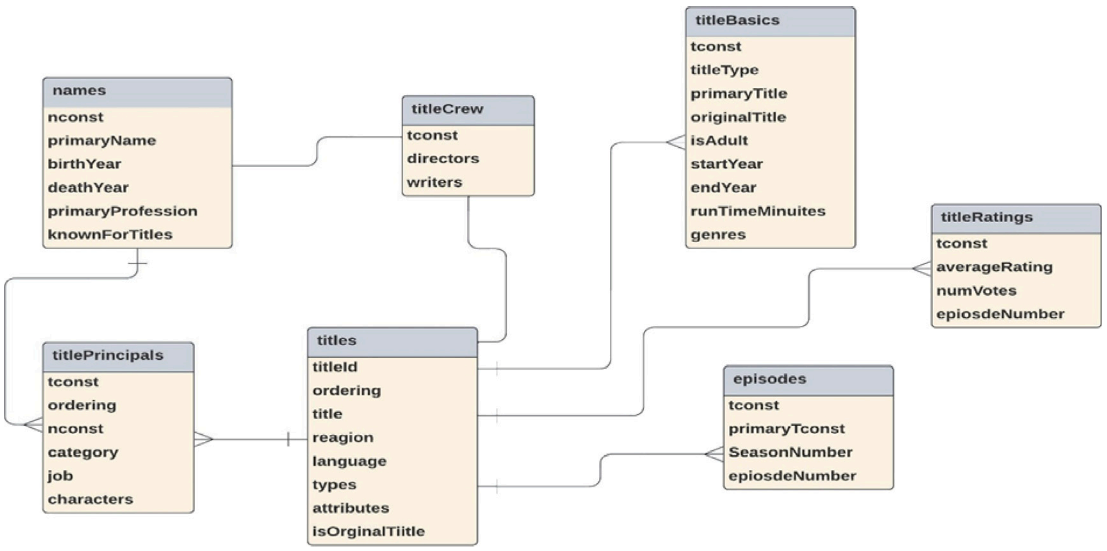


Fig. 4. IMDb ER diagram contains 7 entities, including names, titles, titlesRatings, episodes, titlesBasics, titlesCrew, and titlesPrinciples.

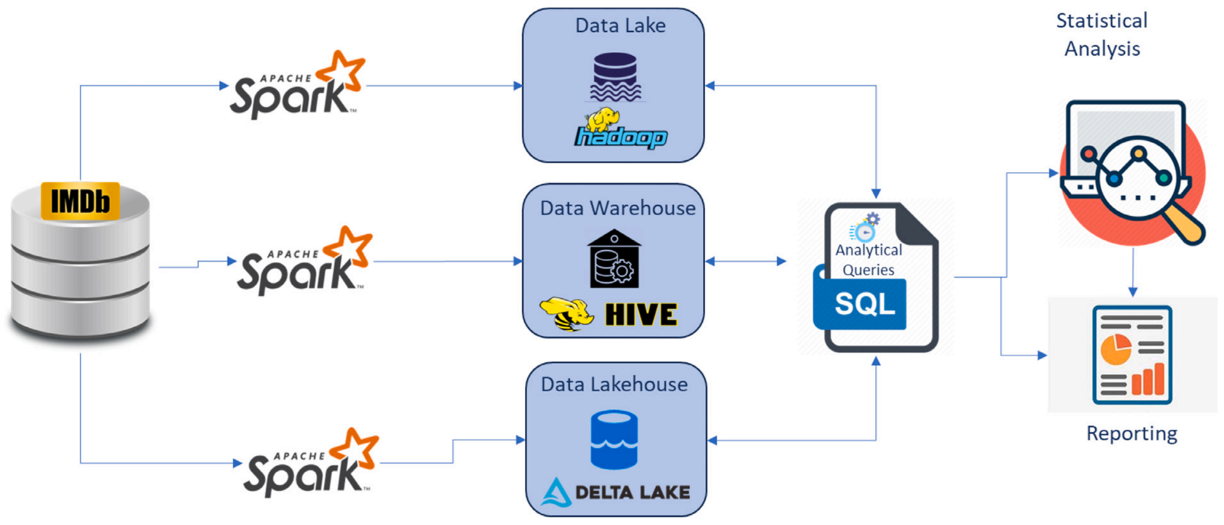


Fig. 5. Pipelines built using 3-storage systems Delta Lakehouse, Hive Warehouse, and HDFS Data Lake.

5.2. Storage systems

Considering the data storage and machine learning needs of big data, we have chosen three open-source systems: Apache Hive for DW [124], Hadoop Distributed File System (HDFS) for DL [6], and Delta LH [52] as described below.

5.2.1. Apache Hive DW

Apache Hive [124] is a DW that allows for analytics on a massive scale. It can manage and analyze petabytes of data using SQL and is built on top of Apache Hadoop. Hive's SQL-like interface, called HiveQL, is designed for non-programmers and is highly efficient in handling massive amounts of data. It seamlessly converts HiveQL queries into MapReduce or Tez jobs. These jobs are then executed on Apache Hadoop's distributed job scheduling framework. Hive's metadata is stored in a megastore and is managed by HCatalog, a layer that manages tables and storage, enabling effortless integration with Apache Pig and MapReduce.

5.2.2. HDFS DL

HDFS DLs [6] are ideal for storing and managing large amounts of data in distributed environments. They offer distributed storage, replication for fault tolerance, schema-on-read for flexibility, and can handle various data types. HDFS is cost-effective and scalable and allows easy integration with processing engines like Apache Spark, Hive, and Hadoop MapReduce for efficient big data processing and analytics.

5.2.3. Delta LH

Delta LH [52] is a storage layer in the Databricks LH Platform. It uses a file-based transaction log for ACID transactions and metadata handling. It works seamlessly with structured data streaming for batch and streaming operations as we mentioned previously in Section 4.

5.3. Analytic software for data ingestion

Our ingestion pipeline is built using Apache Spark [43]. Apache Spark [43] is a powerful open-source distributed computing system for processing large-scale data. It can extract data from a variety of sources, transform and clean it, and handle real-time data ingestion. Spark supports data serialization and deserialization in various formats and can load data into target data stores. It can also perform data quality checks and optimize data storage using partitioning and bucketing

Table 8
Sample of the queries used in the experiments.

NO.	Query

Q1	SELECT *
	 FROM titleBasics as b
	 WHERE genres LIKE '%Fantasy%' and startYear = 2022 group by
	 b.genres, b.isAdult, b.tconst, b.endYear, b.startYear, b.titleType,
	 b.originalTitle, b.primaryTitle, b.runtimeMinutes
	 order by originalTitle

Q2	SELECT originalTitle,startYear,runtimeMinutes,genres,averageRating FROM titleBasics b,titleRatings r where b.tconst = r.tconst and startYear = 2020 and averageRating > 8.0 and titleType LIKE 'movie' and runtimeMinutes > 0 group by originalTitle,startYear,runtimeMinutes,genres,averageRating order by originalTitle
Q11	SELECT b.genres,b.originalTitle, r.averageRating, tp.category,ak.title, n.primaryName,b.titleType From name As n Left join titlePrincipals As tp on tp.nconst = n.nconst Left join titleCrew As tc on tc.tconst = tp.tconst Left join akas As ak on tp.tconst = ak.titleId Left join titleBasics As b ON b.tconst = tp.tconst Left join titleRatings As r ON r.tconst = tp.tconst where b.genres LIKE '%Action%' and n.primaryProfession IS NOT NULL and tp.category IS NOT NULL and r.averageRating > 8.0 and r.numVotes > 1000 and b.titleType = 'short' and tp.category = 'actor' group by b.genres,b.originalTitle, r.averageRating, tp.category,ak.title, n.primaryName,b.titleType order by b.genres
Q14	SELECT distinct b.genres,b.originalTitle,b.startYear, r.averageRating, tp.category,ak.title, n.primaryName,b.titleType, E.seasonNumber, E.episodeNumber From name As n Left join titlePrincipals As tp on tp.nconst = n.nconst Left join titleCrew As tc on tc.tconst = tp.tconst Left join akas As ak on tp.tconst = ak.titleId Left join titleBasics As b ON b.tconst = tp.tconst Left join titleRatings As r ON r.tconst = tp.tconst Left join titleEpisode As E ON tp.tconst = E.tconst group by b.genres,b.originalTitle,b.startYear, r.averageRating, tp.category, ak.title, n.primaryName,b.titleType, E.seasonNumber, E.episodeNumber order by b.originalTitle

techniques. Additionally, Spark integrates well with the components of other big data ecosystems.

5.4. Dataset

To test and validate different models, we have chosen to use the Internet Movie Database (IMDb) [125–127] as our benchmark dataset for real-world scenarios. Standard benchmarks like TPC-H and TPC-DS are not ideal for evaluating query optimizers and estimating cardinality. Their data generators use simplified assumptions that real-world data sets lack, which makes cardinality estimation more challenging. IMDb can present challenges that synthetic data may not capture. For example, it may include skewed data distribution, outliers, and evolving data patterns. Testing query optimizers with real-world data can help identify how well they handle these challenges, potentially leading to more realistic performance assessments.

IMDb contains extensive information on movies, actors, directors, and production companies [125]. The dataset includes 50,000 movies. The complexity of the dataset poses a challenge to its processing and analysis. The ER schema diagram of the dataset utilized in our experimental research is portrayed in Fig. 4. The diagram provides an overview of the data model that relates to the database tables of the IMDb dataset, and the relationships between them.

5.5. Queries

We defined 15 OLAP queries[1] of varying degrees of complexity. In Table 8, we select four queries (Q1, Q2, Q11, Q14) based on the IMDb dataset [125]. These queries have been designed to introduce and test the performance of the DL, DW, and LH. We conducted our experiments using 3 pipelines as shown in Fig. 5. When it comes to database benchmarking, selecting the right queries is essential to effectively evaluate the performance of a database system. These queries should represent real-world scenarios and test various aspects of the database engine, such as indexing, join operations, aggregation, and data retrieval. We choose specific fields when running queries, as the performance can vary greatly depending on the selected fields. By specifying certain fields, we demonstrate how well the storage systems perform with regard to data retrieval and processing of the requested data. This also helps simulate real-world situations where all data is not necessary for a given operation.

6. Validation

We use various metrics to demonstrate the query performance including response time, execution time, CPU and memory utilization, and the ability of the selected DL, DW, and LH systems to handle multiple requests simultaneously. The goal is to measure how efficiently the different systems can process queries while maintaining acceptable response times under different workloads. To measure the performance of complex analytical operations involving complex joins across multiple dimensions on the IMDb dataset, we execute each OLAP query five times and calculate the average response time. Our experiments are capable of extracting an extensive array of fields, which can be utilized for reporting and in-depth analysis. In the second part of our experiments, we select some of the features that are offered by Delta LH to demonstrate their effect on query performances compared to the other systems.

6.1. Experimental setup

The vSphere cluster used for the experiments had 16 CPU cores and 32 GB of RAM and was equipped with a NVIDIA V100 GPU. We have executed 15 queries for each system and recorded end-to-end response time along with the memory consumption and CPU utilization.

6.2. Results

Figs. 6 and 7 show the comparative performances of the 3 storage systems for the 15 OLAP queries. Fig. 6 shows the comparative query response times while Fig. 7(a) shows the comparative mean and standard deviations of the response times. A comparison of the average consumption of CPU time and memory are presented in Fig. 7(b) and (c) respectively. Results of our experimentation on schema evolution and time travel features of Delta LH are illustrated in Table 10. The observations from the study and an analysis of the results are given below.

6.3. Discussion

Table 9 presents a detailed comparison of HDFS (Data Lake), Hive (Data Warehouse), and Delta Lakehouse across various criteria, including integration and ecosystem compatibility, community support, flexibility, cost-effectiveness, data storage, data ingestion, ETL processes, metadata management, data access, privacy, security, performance, and experimental findings. This comparison aims to highlight the strengths and weaknesses of each system to guide researchers and practitioners in

[1] https://github.com/HarbyElectro/SQL_Queries_IMDb.git.

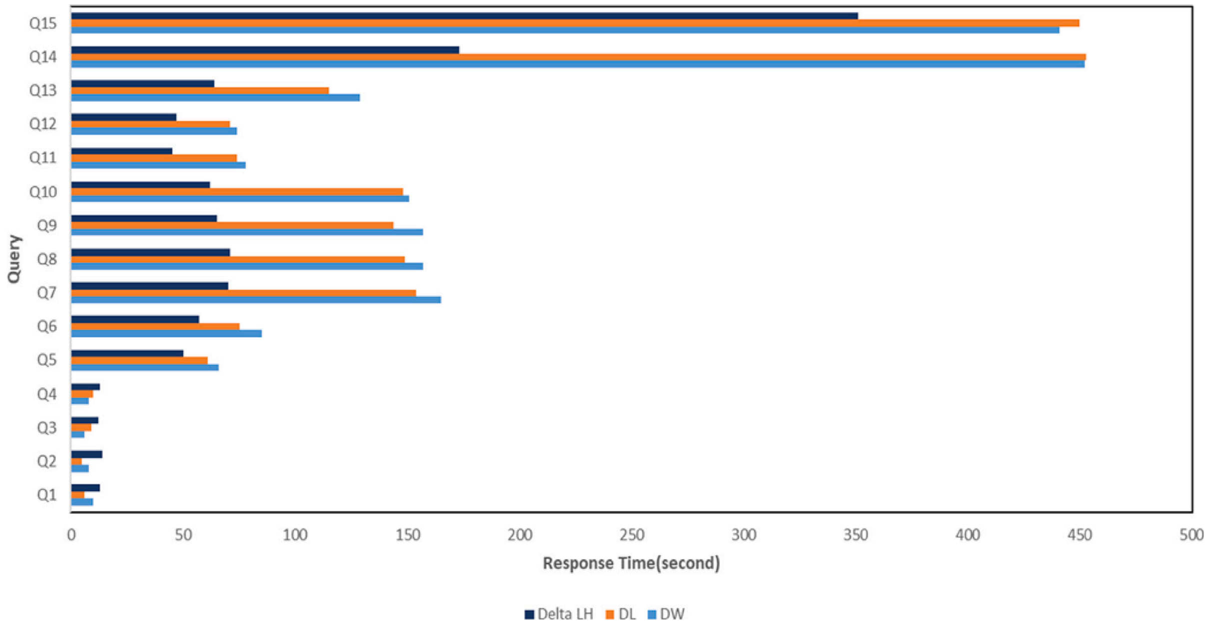


Fig. 6. Comparative review of response times of the queries applied on the 3 data storage systems Delta LH, Hive DW, and HDFS DL.

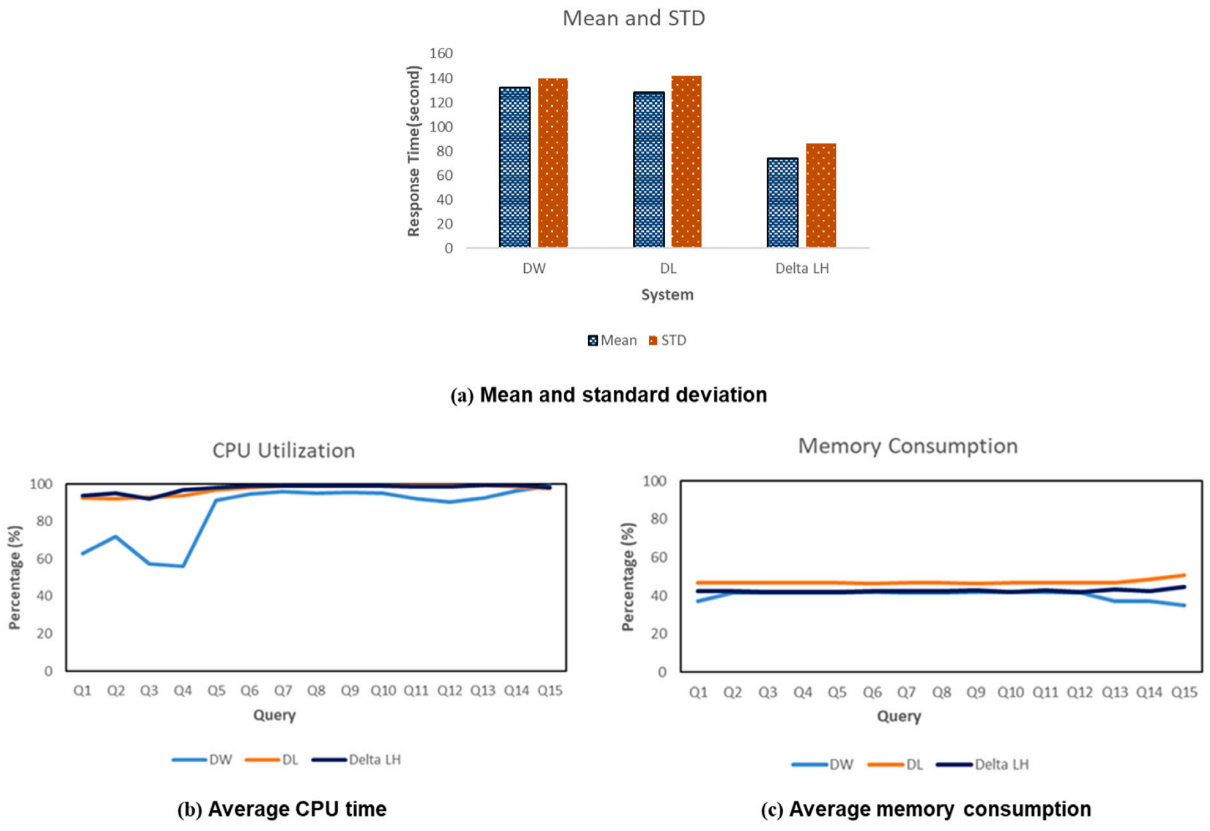


Fig. 7. Mean, standard deviation, CPU time, and memory consumption.

selecting the most suitable architecture for their specific experimental setups. By evaluating these systems on multiple aspects, we ensure a comprehensive understanding of their capabilities and limitations, facilitating informed decision-making for designing robust and efficient big data solutions. Additionally, we discuss the experimental findings in detail as follows:

Response Time: As shown in Fig. 6, Delta LH exhibits a higher response time for certain queries including Q1, Q2, Q3, Q4, and a lower response time for Q5 to Q15 queries. The combination of Delta LH tables and LH architecture offers several benefits for data management, reliability, and performance. However, these features also add some overhead to the response time due to metadata management, data integrity maintenance, schema evolution, and indexing and processing of streaming/batch data. Although for complex queries Q5 to Q15, Delta LH's adaptive indexing strategy allows for faster query performance and better scalability by dynamically determining the most efficient index

Table 9
Comparison of Data Lake (HDFS), Data Warehouse (Hive), and Delta LH.

Criteria	HDFS (DL)	Hive (DW)	Delta Lake (LH)
Integration & Ecosystem compatibility	• Part of the Apache Hadoop ecosystem. • Integrates well with Hadoop tools like Spark, MapReduce, etc.	• Built on Hadoop. • Integrates with tools like Pig and MapReduce.	• Part of the Apache Spark ecosystem. • Integrates well with Spark and other big data tools.
Community support & Documentation	• Strong community support • Extensive documentation available.	• Strong community support • Extensive documentation available.	• Strong community support. • Extensive documentation available.
Flexibility & Extensibility	• Schema-on-read • Can handle various data types.	• Schema-on-write. • Optimized for SQL queries.	• Combines schema-on-read and schema-on-write. • Supports ACID transactions and real-time data processing.
Cost-Effectiveness	• Open-source • Cost-effective with no significant licensing fees.	• Open-source. • Cost-effective with no significant licensing fees.	• Open-source • Cost-effective with no significant licensing fees.
Data storage	• Distributed storage • Replication for fault tolerance.	• Stores metadata in a metastore • Built on HDFS for storage.	• Optimized for structured and semi-structured data; supports ACID transactions.
Data ingestion	• Supports batch ingestion.	• Supports batch ingestion • Converts HiveQL into MapReduce jobs.	• Supports both batch and streaming ingestion • Compatible with tools like Spark and Flink.
Data transformation & Loading (ETL)	• Flexible • Supports schema-on-read.	• Supports ETL through HiveQL.	• Integrated with Spark for ETL processes • Supports data compaction and deduplication.
Metadata management	• Limited metadata support.	• Stores metadata in a megastore managed by HCatalog.	• Advanced metadata management with schema evolution, transaction log, and data versioning.
Data access & Retrieval	• Limited query optimization. • Not designed for interactive queries.	• SQL-like interface (HiveQL). • Optimized for batch processing.	• Supports SQL queries via Spark. • Efficient data retrieval with Delta Table.
Privacy & Security	• Basic security features. • Relies on Hadoop's security mechanisms.	• Supports access control through HCatalog.	• Advanced security features. • Supports ACID transactions and data versioning.
Performance	• Good for large-scale storage but less efficient for complex queries.	• Efficient for batch processing but not for real-time queries.	• High performance for both batch and real-time queries. • Supports ACID transactions.
Experimental findings	• Suitable for storing diverse datasets without schema enforcement.	• Suitable for traditional data warehousing tasks with large-scale analytics.	• Combines the best of data lakes and warehouses. • Offers transactional integrity and real-time data processing.

based on the query workload, for simple queries the extra work makes LH response time greater compared to the other systems as shown in Fig. 6. This is evident in the superior performance of more complex queries, Q5 to Q15. According to the results displayed in Fig. 7, Delta LH exhibits better performance than the other tested systems in relation to the average and variability of response times for queries. Therefore, Delta LH's advantages can outweigh the added data management cost for complex queries depending on the data size, workload, and query patterns, making LH a better choice for real-world data analysis and informed decision-making.

CPU Time: Delta LH also shows a higher average CPU time than Hive as shown in Fig. 7(b). Delta LH requires an automated metadata management layer that stores critical information, including the schema, transaction logs, and data versions resulting in a higher CPU time. This management layer guarantees that the Delta LH follows ACID principles and allows data versioning. Modifications to a Delta table call APIs to commit changes in the data tables and update the metadata including version information.

Memory Consumption: Fig. 7(c) shows that the memory consumption of Hive is lower compared to the other systems. On the other hand, Delta LH shows lower memory consumption than HDFS DL. Delta employs Optimistic Control to handle concurrency when data is being appended to time-ordered partitions. This concurrency model is designed to minimize conflicts and ensure reliable data operations. With Optimistic Control, Delta LH allows multiple writers to append data concurrently to the same time-ordered partition without creating conflicts. When a writer appends data to a partition, Delta LH creates a unique transaction identifier for that write operation allowing low memory usage. If two or more writers try to append data to the same partition at the same time, Delta LH compares the transaction identifiers and allows the write to the process that has the highest identifier while blocking the others. This ensures that data is written in a consistent and reliable manner without any conflicts or data loss. Optimistic Control is a critical component of Delta LH's data management approach, providing users with a powerful and flexible tool for handling concurrency in large-scale data operations.

Data Versioning in LH: To test the data versioning features of Delta LH, we conducted a test on Delta LH using the IMDb dataset to investigate its capabilities to handle schema evolution and time travel. First, we made changes to the Delta tables' schema, such as adding, dropping, or renaming columns, or changing data types. To enable schema evolution when writing new data to the Delta tables, we used the schema merging option. As part of our testing process, we modified the schema of selected tables in Delta LH to assess Delta LH's capabilities in comparison to HDFS and Hive. We (a) introduced a new column called "releaseDate" in Table 8, and (b) renamed the "primaryTitle" column to "title" in Table 10. We then used the time travel feature to query the Delta tables and compared the responses at different points in time including querying the "titleBasics" table version 0 as shown in Table 10.

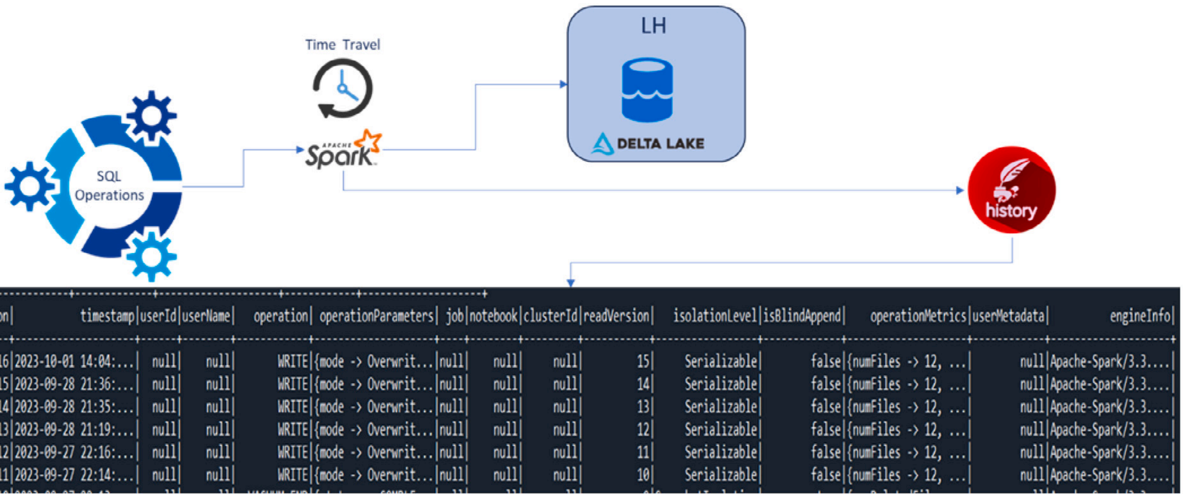
Table 10
List of different operations along with the executed queries for delta LH.

No.	Operation	Queries

1	Adding new column called “releaseDate”	df = df.withColumn(“releaseDate”,col(“startYear”)+“-”+col(“endYear”))
		df.write.format(“delta”).mode(“overwrite”).option(“overwriteSchema”, “true”).save(“IMDb/titleBasics”)
		schema = spark.read.format(“delta”).load(“IMDb/titleBasics”)
		schema.printSchema()

2	Renaming the “primaryTitle” column to “title”	spark.read.format(“delta”).load(“IMDb/titleBasics”)
		.withColumnRenamed(“primaryTitle”, “title”)
		.write
		.format(“delta”)
		.mode(“overwrite”)
		.option(“overwriteSchema”, “true”)
		.save(“IMDb/titleBasics”)

3	Time travel at version 0 for titleBasics” table while also querying the data at different timestamps	spark.read.format(”delta“).option(”versionAsOf",0)
		.load((“IMDb/titleBasics”)).show()
		spark.read.format(“delta”).option(“timestampAsOf”,“2023-09-28 00:00”).load((“IMDb/titleBasics”)).show()



```
+-------+--------------------+------+--------+-----------+--------------------+----+--------+---------+-----------+--------------+-------------+--------------------+------------+--------------------+
|version|           timestamp|userId|userName|  operation| operationParameters| job|notebook|clusterId|readVersion|isolationLevel|isBlindAppend|    operationMetrics|userMetadata|          engineInfo|
+-------+--------------------+------+--------+-----------+--------------------+----+--------+---------+-----------+--------------+-------------+--------------------+------------+--------------------+
|     16|2023-10-01 14:04:...|  null|    null|      WRITE|{mode -> Overwrit...|null|    null|     null|         15|  Serializable|        false|{numFiles -> 12, ...|        null|Apache-Spark/3.3....|
|     15|2023-09-28 21:36:...|  null|    null|      WRITE|{mode -> Overwrit...|null|    null|     null|         14|  Serializable|        false|{numFiles -> 12, ...|        null|Apache-Spark/3.3....|
|     14|2023-09-28 21:35:...|  null|    null|      WRITE|{mode -> Overwrit...|null|    null|     null|         13|  Serializable|        false|{numFiles -> 12, ...|        null|Apache-Spark/3.3....|
|     13|2023-09-28 21:19:...|  null|    null|      WRITE|{mode -> Overwrit...|null|    null|     null|         12|  Serializable|        false|{numFiles -> 12, ...|        null|Apache-Spark/3.3....|
|     12|2023-09-27 22:16:...|  null|    null|      WRITE|{mode -> Overwrit...|null|    null|     null|         11|  Serializable|        false|{numFiles -> 12, ...|        null|Apache-Spark/3.3....|
|     11|2023-09-27 22:14:...|  null|    null|      WRITE|{mode -> Overwrit...|null|    null|     null|         10|  Serializable|        false|{numFiles -> 12, ...|        null|Apache-Spark/3.3....|
```

Fig. 8. History table provided by Delta LH of different schema versions of “titleBasics” table during the experiments. The table displays version numbers recorded along with the timestamps, user info, operation type, metrics, and spark engines used.

We were able to query the Delta tables from their state at a specific point in time or version using time travel as shown in Table 10. Fig. 8 displays a screenshot of the modified Delta LH table history during the experiments. While performing identical operations on HDFS or Hive, we observed that the two systems cannot register various versions when overwriting the schema of the “titleBasics” table.

7. Conclusion and future work

Data management technologies like DLs and DWs face challenges with high-speed data. A new technology called data LH has emerged to meet today’s needs. We compared DWs, DLs, and LH systems to propose new standardized desired features of LH architecture based on an extensive review of the SOTA LH systems.

We also perform a review of data ingestion, metadata management, and storage systems to suggest a list of associated tools to use to design a LH system to meet the specific needs of user applications and future improvements. Finally, we present an experimental study to analyze the performance of 3 open source DL, DW, and LH systems namely HDFS DL, Hive DW, and Delta LH. The experiments demonstrated that although Delta LH had a higher response time for simple queries, it performed much better for complex queries with real-world data than the DL and DW systems. Furthermore, it can support time travel and schema evolution to produce correct query results at different time points which the other systems cannot.

For future work, we intend to explore data integration tools to build an intelligent data ingestion pipeline for LH systems to enable knowledge extraction and linking capabilities. Knowledge extracted during ingestion can help design advanced storage policy and metadata management systems to facilitate data organization and smart linking. It will thereby, enable faster execution of advanced OLAP queries across multiple storage systems. By subjecting these systems to higher workloads and complex queries, we can validate the robustness and credibility of such advanced LH systems.

CRediT authorship contribution statement

Ahmed A. Harby: Writing – review & editing, Writing – original draft, Visualization, Validation, Resources, Methodology, Formal analysis, Conceptualization. **Farhana Zulkernine:** Writing – review & editing, Supervision, Investigation, Funding acquisition.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ahmed Harby reports financial support was provided by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery. Ahmed Harby reports financial support was provided by Research Tools and Instruments (RTI). Ahmed Harby reports financial support was provided by Canada Foundations for Innovation (CFI) grants. Farhana Zulkernine reports a relationship with Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery that includes: funding grants. Farhana Zulkernine reports a relationship with

Research Tools and Instruments (RTI) that includes: funding grants. Farhana Zulkernine reports a relationship with Canada Founda- tons for Innovation (CFI) grants that includes: funding grants. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have included a link to the code and queries used in the experiments on GitHub in the manuscript. Also, the IMDb dataset used in the experiments is publicly available for download.

Acknowledgments

The research was supported by NSERC Discovery, Research Tools and Instruments (RTI), and Canada Foundation for Innovation (CFI) grants.

References

[1] S.A. El-Seoud, H.F. El-Sofany, M. Abdelfattah, R. Mohamed, Big data and cloud computing: Trends and challenges, Int. J. Interact. Mob. Technol. 11 (2) (2017).
[2] H.E. Miller, Big-data in cloud computing: a taxonomy of risks, 2013.
[3] I. Khan, S.K. Naqvi, M. Alam, S.A. Rizvi, Data model for big data in cloud environment, in: 2015 2nd International Conference on Computing for Sustainable Global Development, INDIACom, IEEE, 2015, pp. 582–585.
[4] I. Lee, Big data: Dimensions, evolution, impacts, and challenges, Bus. Horiz. 60 (3) (2017) 293–303.
[5] S. Tonidandel, E.B. King, J.M. Cortina, Big data methods: Leveraging modern data analytic techniques to build organizational science, Organ. Res. Methods 21 (3) (2018) 525–547.
[6] C. Mathis, Data lakes, Datenbank-Spektrum 17 (3) (2017) 289–293.
[7] W.H. Inmon, The data warehouse and data mining, Commun. ACM 39 (11) (1996) 49–51.
[8] N. Miloslavskaya, A. Tolstoy, Big data, fast data and data lake concepts, Procedia Comput. Sci. 88 (2016) 300–305.
[9] S.R. Gardner, Building the data warehouse, Commun. ACM 41 (9) (1998) 52–60.
[10] B. Inmon, Data Lake Architecture: Designing the Data Lake and Avoiding the Garbage Dump, Technics publications, 2016.
[11] F. Ravat, Y. Zhao, Metadata management for data lakes, in: European Conference on Advances in Databases and Information Systems, Springer, Cham, 2019, pp. 37–44.
[12] S. Taktak, J. Feki, Toward propagating the evolution of data warehouse on data marts, in: International Conference on Model and Data Engineering, Springer, Berlin, Heidelberg, 2012, pp. 178–185.
[13] A.A. Harby, F. Zulkernine, From data warehouse to lakehouse: A comparative review, in: 2022 IEEE International Conference on Big Data, Big Data, IEEE, 2022, pp. 389–395.
[14] N.H.Z. Abai, J.H. Yahaya, A. Deraman, User requirement analysis in data warehouse design: a review, Proc. Technol. 11 (2013) 801–806.
[15] J.C. Nwokeji, F. Aqlan, A. Anugu, A. Olagunju, Big data ETL implementation approaches: A systematic literature review (P), in: SEKE, 2018, pp. 713–714.
[16] W.H. Inmon, What is a data warehouse, Prism Tech. Top. 1 (1) (1995) 1–5.
[17] Q. Yang, M. Ge, M. Helfert, Analysis of data warehouse architectures: Modeling and classification, in: ICEIS (2), 2019, pp. 604–611.
[18] R. Liu, H. Isah, F. Zulkernine, A Big Data Lake for Multilevel Streaming Analytics, IBDAP, IEEE, 2020, pp. 1–6.
[19] A. Nordeen, Learn Data Warehousing in 24 Hours, Guru99, 2020.
[20] A.K. Hamoud, M.Abd. Ulkareem, H.N. Hussain, Z.A. Mohammed, G.M. Salih, Improve HR decision-making based on data mart and OLAP, in: Journal of Physics: Conference Series, Vol. 1530, IOP Publishing, 2020, 012058, (1).
[21] I.A. Najm, J.M. Dahr, A.K. Hamoud, A.S. Alasady, W.A. Awadh, M.B. Kamel, A.M. Humadi, OLAP mining with educational data mart to predict students' performance, Informatica (Ljubl.) 46 (5) (2022).
[22] A. Nambiar, D. Mundra, An overview of data warehouse and data lake in modern enterprise data management, Big Data Cognit. Comput. 6 (4) (2022) 132.
[23] J. Dixon, Pentaho, hadoop, and data lakes, 2010, https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/.
[24] A. Gorelik, The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science, O'Reilly Media, 2019.
[25] I.G. Terrizzano, P.M. Schwarz, M. Roth, J.E. Colino, Data wrangling: The challenging yourney from the wild to the lake, in: CIDR, 2015.
[26] P. Pasupuleti, B.S. Purra, Data Lake Development with Big Data, Packt Publishing Ltd, 2015.
[27] A. Mohamed, M.K. Najafabadi, Y.B. Wah, E.A.K. Zaman, R. Maskat, The state of the art and taxonomy of big data analytics: view from new big data framework, Artif. Intell. Rev. 53 (2020) 989–1037.
[28] I.D. Nogueira, M. Romdhane, J. Darmont, Modeling data lake metadata with a data vault, in: Proceedings of the 22nd International Database Engineering & Applications Symposium, 2018, pp. 253–261.
[29] A.M. Olawoyin, C.K. Leung, A. Cuzzocrea, Open data lake to support machine learning on arctic big data, in: 2021 IEEE International Conference on Big Data, Big Data, IEEE, 2021, pp. 5215–5224.
[30] S. Sharma, Expanded cloud plumes hiding big data ecosystem, Future Gener. Comput. Syst. 59 (2016) 63–92.
[31] A. Cuzzocrea, Big data lakes: models, frameworks, and techniques, in: 2021 IEEE International Conference on Big Data and Smart Computing, BigComp, IEEE, 2021, pp. 1–4.
[32] F. Nargesian, E. Zhu, R.J. Miller, K.Q. Pu, P.C. Arocena, Data lake management: challenges and opportunities, Proc. VLDB Endow. 12 (12) (2019) 1986–1989.
[33] F. Ravat, Y. Zhao, Data lakes: Trends and perspectives, in: Database and Expert Systems Applications: 30th International Conference, DEXA 2019, Linz, Austria, August 26–29, 2019, Proceedings, Part I 30, Springer International Publishing, 2019, pp. 304–313.
[34] J. Couto, O.T. Borges, D.D. Ruiz, S. Marczak, R. Prikladnicki, A mapping study about data lakes: An improved definition and possible architectures, in: SEKE, 2019, pp. 453–578.
[35] E. Zagan, M. Danubianu, Data lake approaches: A survey, in: 2020 International Conference on Development and Application Systems, DAS, IEEE, 2020, pp. 189–193.
[36] C. Giebler, C. Gröger, E. Hoos, H. Schwarz, B. Mitschang, Leveraging the data lake: Current state and challenges, in: Big Data Analytics and Knowledge Discovery: 21st International Conference, DaWaK 2019, Linz, Austria, August 26–29, 2019, Proceedings 21, Springer International Publishing, 2019, pp. 179–188.
[37] M. Chessell, F. Scheepers, N. Nguyen, R. van Kessel, R. van der Starre, Governing and managing big data for analytics and decision makers, IBM Redguides Bus. Lead. 252 (2014).
[38] P. Patel, G. Wood, A. Diaz, Data lake governance best practices, in: The DZone Guide to Big Data-Data Science & Advanced Analytics, Vol. 4, 2017, pp. 6–7.
[39] R. Hai, C. Koutras, C. Quix, M. Jarke, Data lakes: A survey of functions and systems, IEEE Trans. Knowl. Data Eng. (2023).
[40] S. Chatti, Using spark, kafka and NIFI for future generation of ETL in IT industry, J. Innov. Inf. Technol. 3 (2) (2019) 11–14.
[41] J. Kreps, N. Narkhede, J. Rao, Kafka: A distributed messaging system for log processing, in: Proceedings of the NetDB, Vol. 11, 2011, pp. 1–7, No. 2011.
[42] K. Ting, J.J. Cecho, Apache Sqoop Cookbook: Unlocking Hadoop for Your Relational Database, O'Reilly Media, Inc, 2013.
[43] S. Salloum, R. Dautov, X. Chen, P.X. Peng, J.Z. Huang, Big data analytics on apache spark, Int. J. Data Sci. Anal. 1 (2016) 145–164.
[44] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, . . ., R. Murthy, Hive-a petabyte scale data warehouse using hadoop, in: 2010 IEEE 26th International Conference on Data Engineering, ICDE 2010, IEEE, 2010, pp. 996–1005.
[45] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, . . ., M. Zaharia, Structured streaming: A declarative api for real-time applications in apache spark, in: Proceedings of the 2018 International Conference on Management of Data, 2018, pp. 601–613.
[46] Y. Cheng, F.C. Liu, S. Jing, W. Xu, D.H. Chau, Building big data processing and visualization pipeline through apache zeppelin, in: Proceedings of the Practice and Experience on Advanced Research Computing, 2018, pp. 1–7.
[47] A.B. Rashid, M. Ahmed, A.B. Ullah, Data lakes: a panacea for big data problems, cyber safety issues, and enterprise security, in: Next-Generation Enterprise Security and Governance, CRC Press, 2022, pp. 135–162.
[48] M. Armbrust, A. Ghodsi, R. Xin, M. Zaharia, LH: a new generation of open platforms that unify data warehousing and advanced analytics, in: Proceedings of CIDR, 2021.
[49] B. Shiyal, Modern data warehouses and data LHs, in: Beginning Azure Synapse Analytics, A Press, Berkeley, CA, 2021, pp. 21–48.
[50] S. Vakharia, P. Li, W. Liu, S. Narayanan, Shared foundations: Modernizing meta's data lakehouse, in: The Conference on Innovative Data Systems Research, CIDR, 2023.
[51] Apache iceberg, 2023, Available at: https://iceberg.apache.org/. (Accessed 24 September 2023).
[52] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, . . ., M. Zaharia, Delta lake: high-performance ACID table storage over cloud object stores, Proc. VLDB Endow. 13 (12) (2020) 3411–3424.
[53] E. Begoli, I. Goethert, K. Knight, A lakehouse architecture for the management and analysis of heterogeneous data for biomedical research and mega-biobanks, in: 2021 IEEE International Conference on Big Data, Big Data, IEEE, 2021, pp. 4643–4651.
[54] P. Jain, P. Kraft, C. Power, T. Das, I. Stoica, M. Zaharia, Analyzing and comparing lakehouse storage systems, in: CIDR, 2023.

[55] S.A. Errami, H. Hajji, K.A. El Kadi, H. Badir, Spatial big data architecture: From data warehouses and data lakes to the LakeHouse, J. Parallel Distrib. Comput. 176 (2023) 70–79.
[56] K. Jameel, A. Adil, M. Bahjat, Analyses the performance of data warehouse architecture types, J. Soft Comput. Data Min. 3 (1) (2022) 45–57.
[57] H. Lv, L. Zhou, Y. Zhao, Classification of data granularity in data warehouse, in: 2017 9th International Conference on Intelligent Human–Machine Systems and Cybernetics, IHMSC, Vol. 2, IEEE, 2017, pp. 118–122.
[58] S.H.A. El-Sappagh, A.M.A. Hendawi, A.H. El Bastawissy, A proposed model for data warehouse ETL processes, J. King Saud Univ. Comput. Inf. Sci. 23 (2) (2011) 91–104.
[59] Z. El Akkaoui, E. Zimányi, J.N. Mazón, J. Trujillo, A model-driven framework for ETL process development, in: Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, 2011, pp. 45–52.
[60] S. Mu, Q. Zhu, Y. Zhang, Y. An, Data warehouse dimensional modeling for customer service business, in: MATEC Web of Conferences, Vol. 309, EDP Sciences, 2020, p. 05010.
[61] A. Sebaa, et al., Medical big data warehouse: Architecture and system design, a case study: Improving healthcare resources distribution, J. Med. Syst. 42 (4) (2018) 59.
[62] Q. Yang, M. Ge, M. Helfert, Developing reliable taxonomic features for data warehouse architectures, in: 2020 IEEE 22nd Conference on Business Informatics, CBI, Vol. 1, IEEE, 2020, pp. 241–249.
[63] Q. Yang, M. Ge, M. Helfert, Analysis of data warehouse architectures: modeling and classification, 2019.
[64] G. Agapito, C. Zucco, M. Cannataro, COVID-warehouse: A data warehouse of Italian COVID-19, pollution, and climate data, Int. J. Environ. Res. Public Health 17 (15) (2020) 5596.
[65] A. Venditti, F. Fasano, A systematic approach to choose the data warehouse architecture, in: ICISSP, 2019, pp. 711–718.
[66] C.E. Poenaru, D. Merezeanu, R. Dobrescu, E. Posdarascu, Advanced solutions for medical information storing: Clinical data warehouse, in: 2017 e-Health and Bioengineering Conference, EHB, IEEE, 2017, pp. 37–40.
[67] A.K. Hamoud, et al., Implementing data-driven decision support system based on independent educational data mart, Int. J. Electr. Comput. Eng. (IJECE) 11 (6) (2021).
[68] A. Erraissi, A. Belangour, A. Tragha, Digging into hadoop-based big data architectures, Int. J. Comput. Sci. Issues (IJCSI) 14 (6) (2017) 52–59.
[69] A. Hamoud, A.S. Hashim, W.A. Awadh, Clinical data warehouse: a review, Iraqi J. Comput. Inform. 44 (2) (2018).
[70] D. Solodovnikova, L. Niedrite, Towards a data warehouse architecture for managing big data evolution, in: DATA, 2018, pp. 63–70.
[71] P. Tiwari, S. Kumar, A.C. Mishra, V. Kumar, B. Terfa, Improved performance of data warehouse, in: 2017 International Conference on Inventive Communication and Computational Technologies, ICICCT, IEEE, 2017, pp. 94–104.
[72] G. Sharma, V. Tripathi, A. Srivastava, Recent trends in big data ingestion tools: A study, in: Research in Intelligent and Computing in Engineering, Springer, Singapore, 2021, pp. 873–881.
[73] C. Mathis, Data lakes, Datenbank-Spektrum 17 (3) (2017) 289–293.
[74] H. Wu, Z. Shang, K. Wolter, Learning to reliably deliver streaming data with apache kafka, in: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, IEEE, 2020, pp. 564–571.
[75] A. Ahmet, T. Abdullah, Real-time social media analytics with deep transformer language models: A big data approach, in: 2020 IEEE 14th International Conference on Big Data Science and Engineering, BigDataSE, IEEE, 2020, pp. 41–48.
[76] J. Evermann, J.R. Rehse, P. Fettke, Process discovery from event stream data in the cloud-a scalable, distributed implementation of the flexible heuristics miner on the amazon kinesis cloud infrastructure, in: 2016 IEEE International Conference on Cloud Computing Technology and Science, CloudCom, IEEE, 2016, pp. 645–652.
[77] H. Lv, T. Zhang, Z. Zhao, J. Xu, T. He, The development of real-time large data processing platform based on reactive micro-service architecture, in: 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference, ITNEC, Vol. 1, IEEE, 2020, pp. 2003–2006.
[78] C. Inibhunu, R. Jalali, I. Doyle, A. Gates, J. Madill, C. McGregor, Adaptive API for real-time streaming analytics as a service, in: 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC, IEEE, 2019, pp. 3472–3477.
[79] P.N. Sawadogo, E. Scholly, C. Favre, E. Ferey, S. Loudcher, J. Darmont, Metadata systems for data lakes: models and features, in: New Trends in Databases and Information Systems: ADBIS 2019 Short Papers, Workshops BBIGAP, QAUCA, SemBDM, SIMPDA, M2P, MADEISD, and Doctoral Consortium, Bled, Slovenia, September 8–11, 2019, Proceedings 23, Springer International Publishing, 2019, pp. 440–451.
[80] R. Hai, S. Geisler, C. Quix, Constance: An intelligent data lake system, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 2097–2100.
[81] R. Hai, C. Quix, C. Zhou, Query rewriting for heterogeneous data lakes, in: Advances in Databases and Information Systems: 22nd European Conference, ADBIS 2018, Budapest, Hungary, September 2–5, 2018, Proceedings 22, Springer International Publishing, 2018, pp. 35–49.
[82] C. Quix, R. Hai, I. Vatov, Metadata extraction and management in data lakes with GEMMS, Complex Syst. Inform. Model. Q. (9) (2016) 67–83.
[83] A. Maccioni, R. Torlone, KAYAK: a framework for just-in-time data preparation in a data lake, in: Advanced Information Systems Engineering: 30th International Conference, CAiSE 2018, Tallinn, Estonia, June 11-15, 2018, Proceedings 30, Springer International Publishing, 2018, pp. 474–489.
[84] L. Yin, L. Wang, Y. Zhang, Y. Peng, MapperX: Adaptive metadata maintenance for fast crash recovery of DM-Cache based hybrid storage devices, in: 2021 USENIX Annual Technical Conference, USENIX ATC 21, 2021, pp. 705–713.
[85] A. Beheshti, B. Benatallah, R. Nouri, V.M. Chhieng, H. Xiong, X. Zhao, Coredb: a data lake service, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 2451–2454.
[86] A. Leventidis, L. Di Rocco, W. Gatterbauer, R.J. Miller, M. Riedewald, DomainNet: Homograph detection for data lake disambiguation, 2021, arXiv preprint arXiv:2103.09940.
[87] M. Farid, A. Roatis, I.F. Ilyas, H.F. Hoffmann, X. Chu, CLAMS: bringing quality to data lakes, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 2089–2092.
[88] J.M. Hellerstein, V. Sreekanti, J.E. Gonzalez, J. Dalton, A. Dey, S. Nag, . . ., E. Sun, Ground: A data context service, in: CIDR, 2017.
[89] M. Cherradi, A. El Haddadi, H. Routaib, Data lake management based on dlds approach, in: Networking, Intelligent Systems and Security: Proceedings of NISS 2021, Springer Singapore, 2022, pp. 679–690.
[90] D. Sarramia, A. Claude, F. Ogereau, J. Mezhoud, G. Mailhot, CEBA: A data lake for data sharing and environmental monitoring, Sensors 22 (7) (2022) 2733.
[91] A. Halevy, F. Korn, N.F. Noy, C. Olston, N. Polyzotis, S. Roy, S.E. Whang, Goods: Organizing google's datasets, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 795–806.
[92] P.N. Sawadogo, E. Scholly, C. Favre, E. Ferey, S. Loudcher, J. Darmont, Metadata systems for data lakes: models and features, in: New Trends in Databases and Information Systems: ADBIS 2019 Short Papers, Workshops BBIGAP, QAUCA, SemBDM, SIMPDA, M2P, MADEISD, and Doctoral Consortium, Bled, Slovenia, September 8–11, 2019, Proceedings 23, Springer, 2019, pp. 440–451.
[93] E. Scholly, P. Sawadogo, P. Liu, J.A. Espinosa-Oviedo, C. Favre, S. Loudcher, . . ., C. Noûs, Coining goldmedal: A new contribution to data lake generic metadata modeling, 2021, arXiv preprint arXiv:2103.13155.
[94] R. Eichler, C. Giebler, C. Gröger, H. Schwarz, B. Mitschang, Handle-a generic metadata model for data lakes, in: Big Data Analytics and Knowledge Discovery: 22nd International Conference, DaWaK 2020, Bratislava, Slovakia, September 14–17, 2020, Proceedings 22, Springer International Publishing, 2020, pp. 73–88.
[95] A. Beheshti, B. Benatallah, R. Nouri, A. Tabebordbar, CoreKG: a knowledge lake service, Proc. VLDB Endow. 11 (12) (2018) 1942–1945.
[96] Egeria Metadata Management Tool, Egeria, egeria-project.org, 2018, (Accessed 5 August 2020).
[97] M. Cherradi, A.El. Haddadi, EMEMODL: Extensible metadata model for big data lakes, Int. J. Intell. Eng. Syst. 16 (2023).
[98] P. Sawadogo, J. Darmont, On data lake architectures and metadata management, J. Intell. Inf. Syst. 56 (1) (2021) 97–120.
[99] P.P. Ray, ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope, Internet Things Cyber-Phys. Syst. (2023).
[100] P. Ghavami, Big Data Management: Data Governance Principles for Big Data Analytics, Walter de Gruyter GmbH & Co KG, 2020.
[101] D. Oreščanin, T. Hlupić, B. Vrdoljak, Managing personal identifiable information in data lakes, IEEE Access (2024).
[102] K.A. Andresen, Data privacy implications of assessment technology, in: Talent Assessment: Embracing Innovation and Mitigating Risk in the Digital Age, 2023, p. 234.
[103] N. Henein, B. Willemsen, B. Woo, The State of Privacy and Personal Data Protection, 2020–2022, Gartner Report, 2020.
[104] J.M. Larson, Snowf lake Access Control.
[105] M. Penelova, Access control models, Cybern. Inf. Technol. 21 (4) (2021) 77–104.
[106] X. Zhao, C. Zhang, S. Guan, A data lake-based security transmission and storage scheme for streaming big data, Cluster Comput. (2023) 1–15.
[107] A. Panwar, V. Bhatnagar, M. Khari, A.W. Salehi, G. Gupta, A blockchain framework to secure personal health record (PHR) in IBM cloud-based data lake, Comput. Intell. Neurosci. 2022 (2022).
[108] J. Mesterhazy, G. Olson, S. Datta, High performance on-demand de-identification of a petabyte-scale medical imaging data lake, 2020, arXiv preprint arXiv:2008.01827.
[109] L. Arbuckle, K. El Emam, Building an Anonymization Pipeline: Creating Safe Data, O'Reilly Media, 2020.

[110] M.A. Serrano, L.E. Sánchez, A. Santos-Olmo, D. García-Rosado, C. Blanco, V.S. Barletta, . . ., E. Fernández-Medina, Minimizing incident response time in real-world scenarios using quantum computing, Softw. Qual. J. 32 (1) (2024) 163–192.
[111] E. Bulut, Lakehouse architecture in public cloud, 2024.
[112] I.A. Machado, C. Costa, M.Y. Santos, Data mesh: concepts and principles of a paradigm shift in data architectures, Procedia Comput. Sci. 196 (2022) 263–271.
[113] S. Kutay, Data warehouse vs. Data lake vs. Data lakehouse: An overview, 2021, Available online: https://www.striim.com/blog/data-warehouse-vs-data-lakevs-data-lakehouse-an-overview/.
[114] A. Behm, S. Palkar, Photon: A High-Performance Query Engine for the Lakehouse, CIDR, 2022, www.cidrdb.org. http://cidrdborg/cidr2022/papers/a100-behm.pdf.
[115] V. Belov, E. Nikulchev, Analysis of big data storage tools for data lakes based on apache hadoop platform, Int. J. Adv. Comput. Sci. Appl. 12 (8) (2021).
[116] Dremio, 2023, https://www.dremio.com/.
[117] J. Schneider, H. Schwarz, B. Mitschang, Assessing the lakehouse: Analysis, requirements and definition, in: ICEIS, 2023, pp. 44–56, (1).
[118] Cloudera, The Hybrid Data Company, https://www.cloudera.com/.
[119] S. Engdahl, Blogs. Amazon, 2008, https://aws.amazon.com/blogs/big-data/build-a-lake-house-architecture-on-aws/.
[120] G. Eagar, Data Engineering with AWS: Learn how to Design and Build Cloud-Based Data Transformation Pipelines using AWS, Packt Publishing Ltd, 2021.
[121] What is a Data Lakehouse? Snowflake, https://www.snowflake.com/guides/what-data-lakehouse.
[122] Data Platform - Data Lakehouse, Oracle Help Center, 2023, https://docs.oracle.com/en/solutions/data-platform-lakehouse/index.html#GUID-A328ACEF-30B8-4595-B86F-F27B512744DF.
[123] Google, BigLake: Unify Data Lakes & Data Warehouses & Nbsp — Nbsp; Google Cloud, Google, https://cloud.google.com/biglake.
[124] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E.N. Hanson, O. O'Malley, . . ., X. Zhang, Major technical advancements in apache hive, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, 2014, pp. 1235–1246.
[125] IMDb, IMDb.com, https://developer.imdb.com/non-commercial-datasets/.
[126] Saeed Mian Qaisar, Sentiment analysis of IMDb movie reviews using long short-term memory, in: 2020 2nd International Conference on Computer and Information Sciences, ICCIS, IEEE, 2020.
[127] S. Tripathi, R. Mehrotra, V. Bansal, S. Upadhyay, Analyzing sentiment using IMDb dataset, in: 2020 12th International Conference on Computational Intelligence and Communication Networks, CICN, IEEE, 2020, pp. 30–33.