Scientific Research Publishing

American Journal of Industrial and Business Management, 2026, 16(9), 1071-1097
https://www.scirp.org/journal/ajibm
ISSN Online: 2164-5175
ISSN Print: 2164-5167

Interoperability in Data Management: A Comparative Study of Distributed and Federated Database Systems

Waqar Ahmad

Industrial Engineering Department, King Abdulaziz University, Jeddah, Saudi Arabia
Email: wahmed@kau.edu.sa

How to cite this paper: Ahmad, W. (2026). Interoperability in Data Management: A Comparative Study of Distributed and Federated Database Systems. *American Journal of Industrial and Business Management, 16,* 1071-1097.
https://doi.org/10.4236/ajibm.2026.169054

Received: August 10, 2026
Accepted: September 15, 2026
Published: September 18, 2026

Abstract

This study compares Distributed Database Systems (DDBS) and Federated Database Systems (FDBS) in data management. Modern DDBS, including NoSQL and NewSQL platforms, offer scalability, elasticity, and operational flexibility without assuming specific workload characteristics. Their component architecture accommodates homogeneous and heterogeneous configurations for diverse requirements in large-scale information systems. An FDBS maintains a global schema, translating global queries into local queries and amalgamating local results to present a unified global view; it addresses the complexities of integrating and managing diverse data from multiple autonomous sources. The choice between these paradigms depends on specific use cases and organizational needs. Distributed databases excel in scenarios that prioritize data distribution, replication, and concurrency control, offering higher availability and performance under favorable workload and replication conditions. Federated systems, by contrast, are valuable in situations that demand seamless data exchange and pooled analyses across heterogeneous sources. They accommodate diverse data models, data definitions, and manipulation facilities, and transaction management while preserving the autonomy of participating databases. Despite their advantages, federated systems introduce complexities related to transaction determinism in specific deterministic designs, transaction aborts, and potential latency. This review elucidates the distinctive attributes of both paradigms across architecture, scalability, fault tolerance, query optimization, and security, and identifies directions for future research on interoperable data management.

Keywords

Distributed Database Systems, Federated Database Systems, Interoperability,

Schema Autonomy, Data Replication, Query Optimization, Fault Tolerance

1. Introduction

Databases are not a new terminology in the business world. They are essential to managing most jobs, governments, banks, universities, and practical business life, and their technology is developing rapidly alongside networks, communication technologies, and intelligent devices.

Databases are essential for managing diverse sectors like business, government, education, and healthcare. Advancements in networking, communication, and intelligent devices have driven the evolution of database technology (Ahmed, 2022). Two prominent paradigms, Distributed Database Systems (DDBS) and Federated Database Systems (FDBS), have emerged as robust solutions for handling growing data volumes and complexity (Özsu & Valduriez, 2020). A DDBS distributes data across multiple sites, improving communication and local control, while an FDBS maintains a global schema, translating global queries into local ones and harmonizing local results (Sheth & Larson, 1990). As the demand for interoperable databases grows, a comparative exploration of these systems is essential to understand their intricacies, scalability, fault tolerance mechanisms, optimization strategies, and security features. This review aims to elucidate the distinctive attributes of DDBS and FDBS as interoperable databases, offering insights into their roles and potential synergies in the ever-evolving data management landscape.

1.1. Database and Database Management System

A computerized database stores a huge collection of information and permits users to retrieve and update it. It consists of data, hardware, software, and users. All access to the database is handled by software called a Database Management System (DBMS).

"*A DBMS is a powerful tool for creating and managing large amounts of data efficiently and allowing it to persist over long periods of time safely.*" (Garcia-Molina et al., 2008)

A comprehensive database system consists of data, hardware, software, and users. The hardware is the computer system that houses and supports the database, including servers, storage devices, and network infrastructure. The DBMS manages the organization, storage, retrieval, and manipulation of data, while users, administrators, developers, and end-users interact with the database to perform tasks like querying, updating, and managing the data.

The DBMS acts as an intermediary between users and the database, providing essential functions such as data definition, data manipulation, data security, concurrency control, data integrity, backup and recovery, and query optimization (Silberschatz et al., 2002). It also offers various user interfaces, such as command-line interfaces, graphical user interfaces (GUIs), and application programming in-

terfaces (APIs). Users can interact with the database using a query language, such as SQL (Structured Query Language), to retrieve specific data or perform operations. This structured approach to data management is fundamental in modern businesses and other domains where effective information handling is crucial. A DBMS strongly supports databases by controlling redundancy, restricting unauthorized access, providing backup and recovery, providing multiple user interfaces, and enforcing integrity constraints (Elmasri & Navathe, 2004).

1.2. Overview of Distributed Databases

Today, the world has become a small village with the tremendous growth of communication technologies and large-scale use of the World Wide Web. In each organization, there are several computers, connected locally or geographically separated, sharing data but not memory or disks. These data are stored in the form of a database that combines with network technology to create a distributed database (DDB), with no centralization and more decentralized, autonomous processing.

"*A logically interrelated collection of shared data, and a description of this data, physically distributed over a computer network.*" (Connolly & Begg, 2002)

A DDB is a collection of sites that share a common schema and work together to make data available efficiently anywhere in the same network, exactly as if it were stored locally, while allowing each site to serve its own users (Özsu & Valduriez, 2020). Data in distributed databases can be distributed through fragmentation and replication, allowing for distribution independence: the way the data is distributed can be changed without affecting existing programs (Howe, 2001).

Modern distributed data stores, such as NoSQL and NewSQL systems, store huge amounts of data and offer advantages such as improved communication, parallel data processing, and local control over the data (Cattell, 2011; Klisarova-Belcheva, 2015). Distributed database systems rely on distributed platforms and eliminate the need for a centralized server, making them more robust and offering better availability and response times as the user pool grows (Abdelhafiz, 2020). Production-grade geo-distributed systems such as Google Spanner demonstrate that globally distributed data can be managed with strong consistency at scale (Corbett et al., 2013). Distributed architectures also have the potential to strengthen communities' ownership over their data, promoting data sovereignty (Malard-Adam et al., 2022). Overall, distributed databases have gained importance in business software development and continue to evolve in terms of architectural solutions and development trends.

1.3. Overview of Federated Database Systems

A federated database system (or multidatabase) is a type of distributed system in which the servers at each site are independent: each DBMS has its own local users, transactions, and database administrator, with a very high degree of local autonomy, while applications share a global schema (Elmasri & Navathe, 2004). Feder-

ated systems can be found in engineering design, healthcare, banking, and many other domains (Sheth & Larson, 1990); an early industrial example is the Efendi federated database system of Cadlab (Radeke et al., 1995).

Because FDBS servers may run different software, a translation mechanism is needed to handle the different query languages of each server. Heterogeneity in an FDBS arises from differences in data models, constructs, and query languages. A multidatabase system (MDBS), by contrast, has complete autonomy and no global schema; it

"*provides an environment in which new database applications can access data from a variety of preexisting databases located in various heterogeneous hardware and software environments*." (Silberschatz et al., 2002)

An FDBS maintains a global schema. It translates global queries and updates into local ones, then combines the local results to produce a final global answer. In addition, it handles commit and abort processing for global transactions, typically using synchronization protocols such as Two-Phase Commit.

Federated database systems have been developed to handle the challenges of integrating and managing complex data from multiple sources. These systems provide a unified data view and allow for seamless data exchange and pooled analyses. They can handle heterogeneous data models, data definition and manipulation facilities, and transaction management. Federated systems have been used in contexts such as scientific data analysis (Kim & Moon, 2018) and large-scale biobank-based research projects integrating phenome and genome data from 600,000 twin pairs across Europe (Muilu et al., 2007). More recently, federated query engines such as Presto provide a single SQL interface over dozens of heterogeneous data sources, from data warehouses to NoSQL and streaming systems (Sethi et al., 2019), while polystore architectures such as BigDAWG integrate multiple, disparate storage engines under one federating layer (Duggan et al., 2015). These systems offer a powerful facility for combining different types of information from multiple data sources while providing a standardized interface for end users and application developers.

1.4. Aims and Objectives

This study aims to understand the fundamental concepts of Distributed and Federated Database Systems, their architectural variances, scalability, fault tolerance mechanisms, optimization strategies, and security features. It also considers each paradigm's performance characteristics (query response time, throughput, and effectiveness in handling hardware failures and network disruptions) and its ability to prevent unauthorized access to sensitive data.

The objectives of the study are:

- Define the fundamental concepts of Distributed Database Systems, including data partitioning, replication, and transaction management, and of Federated Database Systems, focusing on data autonomy and distributed query processing.

- Examine the variances in architecture between federated and distributed database systems and how each system handles processing, storing, and distributing data among several nodes or locations.
- Evaluate the scalability and performance of distributed and federated databases, focusing on their ability to handle growing datasets and user demands.
- Examine fault tolerance mechanisms in distributed and federated databases, focusing on data replication, transaction logging, and recovery procedures, and their impact on system reliability.
- Examine query optimization techniques in distributed and federated databases, including parallel processing, indexing, and query rewriting, and compare their efficiency considering response time and resource utilization.
- Explore the security features of distributed and federated databases, focusing on encryption, access controls, and authentication, and the effectiveness of security measures in federated systems.

1.5. Scope and Terminology

Before proceeding, it is useful to distinguish several related but non-identical terms used throughout this review, as these categories are sometimes conflated in the literature. A **Distributed Database System (DDBS)** is a single logical database whose data is fragmented and/or replicated across multiple sites under one administrative design; it is managed by a **Distributed Database Management System (DDBMS)**, which provides a single global schema and coordinates distributed query processing and distributed transaction management across all sites (Özsu & Valduriez, 2020). A **Federated Database System (FDBS)** is a collection of pre-existing, autonomous component databases integrated through local, component, export, and federated schema layers; in the classical sense it provides a federated schema and typically supports coordination of global transactions via mechanisms such as Two-Phase Commit, while preserving local autonomy (Sheth & Larson, 1990). A **Multidatabase System (MDBS)** differs from an FDBS in that it provides no persistent global or federated schema and no centralized transaction coordination; applications instead construct ad hoc views over multiple local databases at query time, and each local system retains complete autonomy (Silberschatz et al., 2002). **Data-federation engines**, such as Presto, provide a single query interface over heterogeneous sources without imposing a persistent global schema or coordinating distributed transactions; they federate at query time only (Sethi et al., 2019). **Polystore architectures**, such as BigDAWG, extend this idea by routing sub-queries to whichever underlying storage engine best fits a given data model, again without a single persistent global schema governing all engines (Duggan et al., 2015).

Of these categories, only DDBS/DDBMS and FDBS in the classical Sheth and Larson (1990) sense provide a genuine global or federated schema. Only DDBS/DDBMS provides native distributed transaction coordination across all participating sites; FDBS provides it only for the subset of operations exported by local

databases. MDBS, data-federation engines, and polystores provide federated query access without persistent schema integration or centralized transaction coordination. This review uses "DDBS" and "FDBS" in the senses defined above; where a cited system more precisely fits one of the other categories (MDBS, federation engine, or polystore), that distinction is noted explicitly at the relevant point in the text.

2. Methodology

2.1. Literature Survey

This study conducted a structured narrative literature survey to identify key concepts related to Distributed and Federated Database Systems. Academic databases searched were IEEE Xplore, the ACM Digital Library, SpringerLink, and PubMed, supplemented by Google Scholar for foundational textbooks and standards.

Inclusion criteria were: 1) peer-reviewed journal articles, peer-reviewed conference papers, patents, or graduate-level textbooks; 2) written in English; 3) directly addressing the architecture, scalability, fault tolerance, query processing, or security of distributed or federated database systems. Exclusion criteria were: non-peer-reviewed marketing material without accompanying technical documentation, sources not available in English, and duplicate reports of the same system or result.

2.2. Development of the Conceptual Framework

For subsequent analysis, a conceptual framework was developed to distinguish between Distributed and Federated Database Systems, based on key concepts identified in the literature review.

2.3. Architectural Variances

The study analyzed architectural differences between Distributed and Federated Database Systems, focusing on their management of data distribution, storage mechanisms, and processing models. A comparative analysis framework was developed to evaluate the impact of architectural choices on scalability, fault tolerance, and performance.

2.4. Assessment of Scalability and Performance

The study assessed the scalability and performance of both paradigms through a qualitative, narrative synthesis of the architectural characteristics and design claims reported in the included literature, focusing on how each paradigm's design addresses growing datasets and increased user loads. Because the included studies vary widely in benchmark workload, hardware, and measurement methodology, and because few report results collected under a common protocol, this study does not extract or statistically aggregate query response time or throughput values across studies into a single quantitative comparison. Instead, Section 7 summarizes what the literature reports about each paradigm's scalability and per-

formance characteristics in qualitative terms, and separately notes, without cross-study statistical comparison, where individual systems report their own quantitative results.

2.5. Analysis of Fault Tolerance Mechanisms

The study analyzed fault tolerance mechanisms in Distributed and Federated Database Systems, focusing on replication, logging, and recovery procedures. It aimed to compare and contrast the effectiveness of these mechanisms in ensuring data integrity and availability in challenging scenarios.

2.6. Optimization of Query Processing

The study analyzed query optimization techniques in Distributed and Federated Database Systems, including parallel processing, indexing, and query rewriting, and assessed their impact on system performance considering query response time and resource utilization.

2.7. Review of Security Features

Finally, the study reviewed security mechanisms in both paradigms, covering encryption, authentication, access control, and the specific security and privacy issues that arise when autonomous systems federate and share data across administrative domains.

3. Distributed Databases

3.1. Fundamental Principles of DDB

The fundamental principles of distributed databases include distribution, heterogeneity, and autonomy of data sources (Özsu & Valduriez, 2020). These principles are based on the idea that data is divided and stored across multiple computers connected via a communication network. Distributed databases aim to provide scalability, elasticity, and operational flexibility without making assumptions about the workload (Loesing et al., 2015), and they can strengthen communities' ownership over their data (Malard-Adam et al., 2022). A fundamental constraint on their design is captured by the CAP theorem, which shows that a distributed system cannot simultaneously guarantee consistency, availability, and partition tolerance (Gilbert & Lynch, 2002).

It is important to distinguish the availability guaranteed by the CAP theorem from operational high availability, as the two are frequently conflated in comparative claims about distributed and federated systems. CAP availability describes whether every non-failing node returns a response to every request while a network partition is occurring; it says nothing about how quickly that response arrives, how stale it may be, or how the system performs once the partition heals. Operational high availability, by contrast, refers to a system's uptime under normal operating conditions and specific failure scenarios, and is typically expressed as a percentage of time the system is reachable and serving requests correctly. A

system can be CAP-available during a partition while still exhibiting poor operational availability if its chosen consistency model, replica-recovery procedure, or reconciliation process after the partition heals is costly (Gilbert & Lynch, 2002). Comparative claims in this review that refer to "availability" specify which of these two senses is meant.

The emergence of new distributed data management techniques and systems, such as MapReduce and Hadoop, has raised questions about the need for new foundations and a generic architectural model to explain these principles (Valduriez, 2011). While these newer solutions may be effective for specific applications, they may also pose challenges regarding data interoperability.

The classical objectives of a distributed database are summarized in Date's twelve rules (Date, 2004):

1) Local autonomy. Autonomy means "the distribution of control, not of data" (Özsu & Valduriez, 2020).

2) No reliance on a central site.

3) Continuous operation.

4) Location independence.

5) Fragmentation independence.

6) Replication independence.

7) Distributed query processing.

8) Distributed transaction management.

9) Hardware independence.

10) Operating system independence.

11) Network independence.

12) DBMS independence.

Each database is controlled and maintained by software that manages the DDB, called a Distributed Database Management System (DDBMS). It handles functions such as data distribution, database recovery, and security.

3.2. Component Architecture for a DDBMS

A DDBMS is typically decomposed into four principal components: 1) the local DBMS at each site, 2) a data communication component, 3) a global system catalog, and 4) the distributed DBMS layer that coordinates the sites. Research on componentized architectures has extended this model in several directions. Component-based DBMS architectures have been proposed for nomadic and mobile computing, providing lightweight, customizable systems that can adapt to a changing environment (McCann & Crane, 1998). In multilevel secure distributed database management systems (MLS/DDBMS), secure transaction management protocols and concurrency control mechanisms have been proposed to prevent covert channels and starvation of high-security-level transactions (Kaur et al., 2007). A four-layer enterprise architecture model has been proposed for service management, with a component model addressing functional requirements for integrated service management functions (Chen et al., 2011). Microsoft's OLE DB

approach aims to provide uniform access to data stored in diverse DBMS and non-DBMS information containers through a collection of interfaces that encapsulate DBMS functionality (Blakeley, 1996). More recently, the disaggregation trend separates compute from storage so that each can scale independently and efficiently based on workload characteristics (Ghandeharizadeh et al., 2023).

3.3. Advantages and Disadvantages of DDBMS

3.3.1. Advantages

Distributed Database Management Systems offer several advantages. Under replication schemes with sufficient redundancy and a consistency model suited to the workload, they can provide higher operational availability than centralized systems, remaining operational even in the presence of certain site failures (Fior et al., 2013). They can also offer higher performance and throughput than centralized database systems, under workloads that permit effective parallelism across sites (Özsu & Valduriez, 2020). Modern cloud-native and NewSQL systems demonstrate these benefits at scale under their respective workloads: Amazon Aurora reports high throughput for cloud-native relational workloads by redesigning the storage layer for distribution (Verbitski et al., 2017), while geo-distributed SQL systems such as CockroachDB report resilient, serializable transactions across regions (Taft et al., 2020). It is important to note, however, that there may be a trade-off between security level and system throughput in a DDBMS (Lamba & Sharma, 2011). Overall, the adoption of a DDBMS can enhance a company's profitability, but it may also require changes in organizational structure and culture (Gordon & Gordon, 1992).

3.3.2. Disadvantages

Distributed database management systems also have disadvantages. One major challenge is the complex management and implementation of security in heterogeneous environments, including distributed authentication services, encryption of network data, and secure remote database administration (Harris & Sidwell, 1994). Another disadvantage is the difficulty of designing an optimally distributed database given a large number of geographically distributed sites and database relations. Fragmentation, data allocation, and replication are the main design techniques, but they are often treated separately and rarely processed together, leading to limitations in achieving an optimal design (Yoshida et al., 1985). Scaling the software architecture itself is a further challenge that has motivated redesigned, more scalable system structures (Zhuang et al., 2015).

3.4. Homogeneous and Heterogeneous DDBMSs

Distributed systems may be homogeneous:

"*where all sites have common shared data and database system code, or heterogeneous, where schema and system code may differ.*" (Silberschatz et al., 2002)

All server nodes and clients use an identical DBMS in a homogeneous system; otherwise, the system is classified as heterogeneous. In addition, the degree of lo-

cal autonomy is related to the degree of homogeneity:

"*If there is no provision for the local site to function as a stand-alone DBMS, then the system has no local autonomy. On the other hand, if direct access by local transactions to a server is permitted, the system has some degree of local autonomy.*" (Elmasri & Navathe, 2004)

Both homogeneous and heterogeneous distributed database management systems are used in modern large-scale information systems. Homogeneous systems are those in which all databases share the same data model and query language, whereas heterogeneous systems involve databases with different data models and query languages. The uniform integration of data from different databases is a central problem in data integration, and various methods have been proposed to address structural, syntactic, and semantic heterogeneity between relational and NoSQL databases and among different NoSQL databases (Stojanovic et al., 2022). Additionally, the disaggregation trend in cloud database management systems allows processing to be separated from storage, enabling each to scale independently based on workload characteristics (Ghandeharizadeh et al., 2023).

Different hardware and different DBMS products between sites constitute heterogeneity, which requires a translation mechanism between the sites. This mechanism uses a canonical system language to translate queries into each server's language for implementing DDBMS queries and processing.

4. Federated Database System (FDBS)

4.1. FDBS Architecture

The reference architecture for federated database systems distinguishes local, component, export, and federated schemas, with the federation layer mediating between autonomous component databases (Sheth & Larson, 1990). A universal relation approach has also been proposed to simplify federated database management by shielding users from schema diversity (Zhao et al., 1995). A federated data warehouse (FDW) provides a unified perspective on a set of independent data warehouses: it creates a virtual global schema from partial schemas of participating warehouses, allowing the end user to view them as a single "super" data warehouse (Kern et al., 2020). Hybrid SQL/NoSQL databases have emerged as a solution for organizations that use different database types in parallel, integrating SQL and NoSQL components to provide the advantages of both (Bjeladinović et al., 2020). Data federation systems aim to easily access and semantically connect heterogeneous data stored in various sources; they create a virtual database to integrate data from multiple sources and use techniques such as natural language processing and association rule analysis for data retrieval and analytics (Vu et al., 2019). Federated identity management enables the management of identity processes and policies among collaborating entities without centralized control; analyses of federated identity approaches include the European eID solution, which aims to serve millions of people while maintaining strict security constraints (Carretero et al., 2018). At the systems level, HRDBMS is a distributed shared-nothing

relational database designed to improve the scalability of OLAP queries; it combines techniques from relational and big data systems and achieves scalability comparable to Hive and Spark SQL while competing with MPP databases on per-node performance (Arnold et al., 2019). Industrial federated engines such as Presto (Sethi et al., 2019) and polystores such as BigDAWG (Duggan et al., 2015) show that federation over heterogeneous engines is now a mainstream architectural pattern.

4.2. Advantages and Disadvantages of FDBS

4.2.1. Advantages

Federated Database Systems offer several advantages. They provide a unified perspective on independent data warehouses, allowing end users to access and query data from multiple sources as if it were a single database (Kern et al., 2020). An FDBS can increase the timely throughput of real-time data services by federating a set of databases and efficiently sharing the workload among them (Zhou & Kang, 2012). In the genomics domain, integrating non-relational stores into federated workflows offers sublinear query time and comprehensive management of different data formats, enabling efficient storage and retrieval of genomic and phenotypic information (Souza et al., 2021). Federated systems also facilitate the integration of complex data from different sources, providing secure and standardized data exchange for large-scale research projects (Muilu et al., 2007). It is worth noting that determinism, a characteristic of some federated and replicated designs, has both advantages and disadvantages in terms of replication, performance, and transaction processing (Ren et al., 2014).

4.2.2. Disadvantages

Federated Database Systems also have disadvantages, though several of the most frequently cited disadvantages in the literature are properties of deterministic replicated database designs specifically, rather than of FDBS as a general category. In their evaluation of deterministic database systems, Ren et al. (2014) identify two such costs: additional overhead from processing transactions for which it is not known in advance what data will be accessed, and an inability to abort transactions arbitrarily, for example, in the case of database or partition overload. These properties arise from the requirement, specific to deterministic designs, that every replica receives and processes the same input in the same order; they are not general properties of FDBS, which in the classical Sheth and Larson (1990) sense need not be deterministic or replicated at all. Deterministic designs additionally require a preprocessing layer that ensures the same input is sent to every replica, which increases latency (Pereira et al., 2016). For general federated-transaction coordination, in which autonomous participants need not be deterministic or replicated, the relevant costs instead relate to the coordination protocol itself: Two-Phase Commit and comparable global-transaction protocols introduce coordination latency, and because participants retain local autonomy, the federation layer cannot guarantee in advance that every local database will honor a prepare-to-commit

request, a general challenge in coordinating autonomous participants rather than a property specific to deterministic designs (Sheth & Larson, 1990; Elmasri & Navathe, 2004). Furthermore, sharing data across domains in federated identity management blurs security boundaries and potentially creates privacy risks, although these threats can be addressed by combining technical and legal/policy tools (Landau et al., 2009; Carretero et al., 2018).

5. Conceptual Framework

Based on the literature survey, this study organizes the comparison of Distributed and Federated Database Systems around five analytical dimensions: 1) architecture, covering data distribution, storage mechanisms, processing models, schema autonomy, and query distribution; 2) scalability and performance; 3) fault tolerance; 4) query processing optimization; and 5) security. The framework treats a DDBS as a single logical database whose fragments and replicas are placed across sites under one administrative design (Özsu & Valduriez, 2020), and an FDBS as a federation of pre-existing, autonomous databases coordinated through export and federated schemas (Sheth & Larson, 1990). The two paradigms therefore differ most fundamentally in where design authority resides: top-down in a DDBS, and bottom-up, negotiated among autonomous participants, in an FDBS. Each of the following sections applies one dimension of this framework to both paradigms.

6. Architecture Variances

6.1. Data Distribution

Distributed and federated database systems differ architecturally in how they manage data distribution. In distributed database systems, the focus is on replication, concurrency, storage, and sharding, with design choices driven by performance goals (Ruan et al., 2021). Federated database systems, on the other hand, emphasize autonomy and cooperation between autonomous and heterogeneous databases (Bonifati et al., 2008). They allow local databases to maintain a high degree of autonomy while enabling global transactions to access and manipulate data from multiple local databases (Deacon et al., 1994). Federated systems often restrict global transactions to executing only high-level operations exported by local databases (Sheth & Larson, 1990). These architectural differences reflect the different goals and requirements of distributed and federated database systems in managing data distribution.

6.2. Storage Mechanism

Distributed database systems focus on performance and scalability (examples include CockroachDB and TiDB) and employ replication, concurrency control, and sharding techniques to ensure efficient data storage and retrieval (Taft et al., 2020; Huang et al., 2020; Samaraweera & Chang, 2021). Federated database systems, such as the multilevel federated architecture proposed by Deacon et al. (1994),

aim instead to maintain the autonomy of local databases in a heterogeneous environment. They allow local and global transactions to coexist, with global transactions accessing and manipulating data from multiple local databases through a multilevel nested transaction model and a practical scheduling mechanism that ensures correct concurrent execution. In short, distributed systems prioritize performance and scalability, while federated systems focus on maintaining autonomy and enabling cooperation among heterogeneous databases (Sheth & Larson, 1990; Ruan et al., 2021).

6.3. Processing Models

Distributed database systems are designed to handle large volumes of data and complex analysis tasks. They use client-server architectures to enable interactive analysis and visualization, focusing on responsiveness and prompt construction of visualizations to uncover data patterns (Starič et al., 2015). Federated database systems allow existing local databases to maintain a high degree of autonomy in a heterogeneous environment; they model transaction executions using a multilevel nested transaction model, which increases concurrency by ignoring pseudo-conflicts and offers greater performance and flexibility (Deacon et al., 1994). The shift towards distributed data management has also inspired micro-distributed database management systems, which address the limitations of centralized architectures and aim for CPU-efficient distributed algorithms (Pirk, 2015). Overall, distributed systems emphasize interactive analysis at scale, while federated systems prioritize autonomy and concurrency in transaction execution.

6.4. Schema Autonomy

Schema autonomy is a particularly clear point of architectural difference. In a classical distributed database system, the sites are coordinated under a single logical schema: the system defines a reference architecture from system and schema viewpoints, and the global design is developed top-down (Özsu & Valduriez, 2020). Federated database systems, by contrast, enable existing local databases in a heterogeneous environment to maintain a high degree of autonomy. These systems support integrating existing data into virtual databases, allowing databases to remain under the control of their respective owners while still supporting data integration and site autonomy requirements (Sheth & Larson, 1990; Deacon et al., 1994). The coexistence of local and global transactions is a key problem in federated systems, and schema integration across heterogeneous sources remains a central research issue (Parent & Spaccapietra, 1998).

6.5. Query Distribution

In distributed database systems, data is partitioned across multiple instances, and query processing and transaction management can be decoupled from data storage. This allows scalability, elasticity, and operational flexibility without assuming a partition-friendly workload (Loesing et al., 2015), and partitioning can be tuned

to the query workload itself (Rabl & Jacobsen, 2017). Federated systems instead integrate heterogeneous data sources behind a single query interface: federation layers such as FedX enable efficient query processing over distributed Linked Open Data sources, optimizing query computation when a query can be answered by multiple sources (Schwarte et al., 2011), and grid middleware such as OGSA-DAI DQP integrates distributed data sources through distributed query processors and views (Dobrzelecki et al., 2010). Scalable distributed indexing further supports query processing over linked, heterogeneous data (Karnstedt et al., 2012). Distributed systems thus focus on scalability and flexibility, while federated systems prioritize integration and optimization for distributed query processing.

7. Scalability and Performance

7.1. Scalability Metrics

Distributed and federated databases have been compared with respect to scalability metrics such as data volume and load. A comparison of three common relational multi-database approaches (federated, gateway, and middleware) suggests that the middleware approach provides better or comparable performance to the other two, making it the most cost-effective solution from a global query perspective (Chen et al., 1998). Designing massively scalable big data systems requires selecting distributed database platforms that can satisfy application quality and cost requirements; a detailed feature taxonomy has been created to enable rigorous comparison and evaluation of distributed database platforms in support of architects building big data systems (Gorton et al., 2015). P2P database systems have been compared with established decentralized models (distributed, federated, and multidatabases) to provide insight into their features and research directions (Bonifati et al., 2008). A reference architecture for federated database systems and a methodology for developing such systems have been defined, highlighting critical issues in their development and operation (Sheth & Larson, 1990). SQL and NoSQL data stores designed for scaling simple OLTP-style application loads have been examined and compared with respect to data model, consistency mechanisms, storage mechanisms, durability guarantees, availability, and query support (Cattell, 2011). More recent NewSQL systems demonstrate that horizontal scalability and strong transactional guarantees can be combined: Spanner scales globally with external consistency (Corbett et al., 2013), TiDB unifies transactional and analytical processing over a Raft-replicated store (Huang et al., 2020), CockroachDB provides resilient geo-distributed SQL (Taft et al., 2020), and Aurora shows how separating compute from a distributed, log-structured storage service improves cloud throughput (Verbitski et al., 2017).

7.2. Performance Metrics

The broader literature on computer performance evaluation cautions that naive comparisons of point performance estimates can be misleading, given the variability inherent in computer performance measurement. Alternative methods for

performance evaluation include resampling methods such as randomization tests and bootstrapping confidence estimation (Irving et al., 2020) and non-parametric Hierarchical Performance Testing frameworks for computer architecture research (Chen et al., 2012). Related methodological work covers interlaboratory comparisons and statistical analysis for calibration and testing competence (Acko et al., 2014) and the selection of appropriate classification performance metrics (Israel, 2006). Within the database community specifically, benchmarking frameworks such as DIAMetrics aim to enable query engines to be compared at scale and support reproducible experimental research (Boncz, 2021).

8. Fault Tolerance Mechanisms

The comparison in this section draws on a small number of individually published designs rather than a single formal specification, and the mechanisms described below are not uniform requirements of every DDBS or FDBS. Unless otherwise noted, the distributed-side discussion assumes a crash-only failure model (sites fail by stopping rather than behaving arbitrarily), replicas owned and administered under a single distributed design, and continuous availability of all non-failed replicated sites during recovery. The federated-side discussion draws primarily on two specific published designs a fault-tolerant federated distributed database using local and global agreement algorithms over a temporal data model (Newman, 2019), and a RAID-based recovery scheme for distributed database management systems (Pareek et al., 2019), each evaluated under its own stated failure models, replica-ownership assumptions, and transaction protocols. These mechanisms should be read as properties of those particular designs rather than as mechanisms inherent to all federated database systems. General FDBS fault tolerance instead depends on the availability and autonomy guarantees each component database chooses to offer, and on whether the federation layer coordinates global transactions via a protocol such as Two-Phase Commit (Sheth & Larson, 1990); it does not presuppose local/global agreement algorithms or RAID-style recovery.

8.1. Replication

Replication is central to fault tolerance in both paradigms. In a distributed database system, multiple copies of data are stored on multiple servers, providing fault tolerance and data availability, and formal methods have been used to develop replication-based fault tolerance rigorously (Katta et al., 2021). Modern commercial database systems are evolving into hybrid distributed systems, where a primary database host system utilizes a secondary distributed system for specific tasks; these systems aim to recover from node failures with minimal disruption to the workload, achieving quick recovery, reduced downtime, and improved execution transparency (Pasupuleti et al., 2022). In a fault-tolerant federated distributed database, data is stored in computing nodes according to a temporal data model, and updates are performed simultaneously using local agreement algorithms; the

updated blocks are then combined and agreed upon using a global distributed agreement algorithm (Newman, 2019). Such mechanisms (replication, high redundancy, and high availability) prevent cascading failures and ensure the availability of distributed services (Sari & Akkaya, 2015).

8.2. Logging

Logging-based mechanisms periodically record the system state, allowing recovery to a consistent state in the event of a failure. A logging layer can even be placed over deterministic systems and can recover the underlying system to a consistent state after a server crash, including actions interrupted mid-way (Pereira et al., 2016). Different methods of storing logging information can be employed, and a fault-tolerant master-slave cluster can be deployed for replicating that information (Katta et al., 2021). In a federated distributed database, a global distributed agreement algorithm can additionally be used to agree upon key edits based on a predefined time range and key set (Newman, 2019).

8.3. Recovery Procedures

Recovery procedures aim to minimize disruption to the workload and improve system availability. One approach uses a hybrid distributed model in which a primary database host enlists a secondary system acting as an accelerator, with query fault tolerance ensuring transparent re-execution after failures (Pasupuleti et al., 2022). Data replication across servers likewise enhances availability and fault tolerance (Katta et al., 2021). Fault-tolerant federated designs use local agreement groups and computing nodes to store and update data, ensuring fault tolerance through local and global agreement algorithms (Newman, 2019). RAID-based redundancy can further enhance fault tolerance in distributed database management systems, providing high recoverability in cases of single and double site failures (Pareek et al., 2019).

8.4. State Maintenance

Beyond replication and logging, fault tolerance also depends on maintaining consistent system state. In federated distributed designs, local agreement algorithms update the state of keys simultaneously within each globally distributed local agreement group, and the updated blocks are combined into an agreed block of key edits (Newman, 2019). In distributed systems generally, fault tolerance can be achieved by persisting the state related to requested operations in a data store and restarting components from the stored state after a failure (Sari & Akkaya, 2015). Formal development methods provide additional assurance that replicated state remains consistent under failure (Katta et al., 2021).

9. Query Processing Optimization

9.1. Parallel Processing

Parallel processing has long been recognized as the foundation of high-perfor-

mance database systems (DeWitt & Gray, 1992). Scalable query processing and query engines over cloud databases have emerged as important topics in academic and industrial research (Cuzzocrea, 2021). Techniques based on sharing data and computation among queries have been formalized in query batching optimization, which significantly improves retrieval time (Eslami et al., 2020). Query-centric partitioning and allocation strategies for partially replicated systems have also been shown to reduce network overhead during parallel query execution (Rabl & Jacobsen, 2017). Recent algorithmic developments cover data processing in large distributed clusters, including multiway join queries, sorting, and matrix multiplication (Koutris et al., 2018).

9.2. Indexing

Indexing-focused optimization has been the subject of extensive research. Scalable distributed indexing approaches for Linked Data show how index structures can be built and queried efficiently across distributed nodes to support fast, distributed access (Karnstedt et al., 2012). Cloud-oriented strategies for distributed query optimization similarly reduce unnecessary operations and improve response times (Kaseb et al., 2021). Query batching using mixed binary quadratic programming (MBQP) can also exploit indexes to significantly improve retrieval time (Eslami et al., 2020). These studies highlight the importance of indexing in improving the efficiency of distributed and federated database systems.

9.3. Query Rewriting

Query rewriting techniques restructure queries before execution. Heuristic rewriting algorithms for multi-database systems built on distributed databases optimize the evaluation of queries while considering evaluation and data shipment costs (Ding, 2022). In federated RDF systems, multi-query optimization rewrites and groups SPARQL queries to minimize total evaluation cost (Peng et al., 2021). Machine-learning-based query optimization for federated databases generates predictive data movement instructions based on predefined efficiency and capacity criteria and produces execution plans by estimating costs across potential target data sources (Creedon & Roche, 2021). Query batching formulations complement rewriting by merging compatible queries into shared execution units (Eslami et al., 2020).

9.4. Federated Query Optimization

In federated databases, where data is spread across multiple autonomous sources, query optimization becomes more challenging: the optimizer must identify relevant data sources and find efficient execution plans across them. Source selection and join ordering strategies of federation layers such as FedX demonstrate how such plans can be computed for distributed Linked Data (Schwarte et al., 2011). Advanced metadata including functional, uniqueness, order, and inclusion dependencies can be leveraged for query optimization in distributed settings

(Kossmann et al., 2021). Cost-based multi-query evaluation further reduces overall processing effort in federated RDF systems (Peng et al., 2021). Industrial engines such as Presto embody these principles at scale, applying predicate pushdown, connector-aware planning, and adaptive scheduling across dozens of heterogeneous sources (Sethi et al., 2019), while polystores such as BigDAWG route sub-queries to the storage engine best suited to each data model (Duggan et al., 2015).

10. Security Features

Security cuts across both paradigms but takes different forms in each. In distributed databases, the principal concerns are distributed authentication, encryption of data in transit between sites, and secure remote administration in heterogeneous environments (Harris & Sidwell, 1994). Secure concurrency control protocols must additionally balance the security level against system throughput (Lamba & Sharma, 2011), and multilevel secure DDBMS designs employ dedicated transaction management protocols to prevent covert channels (Kaur et al., 2007). Surveys of the big data era show that distribution and outsourcing of storage substantially widen the attack surface, making encryption at rest and fine-grained access control essential (Samaraweera & Chang, 2021). In federated systems, security must also be negotiated across administrative domains: federated identity management enables collaborating entities to share authentication and authorization without centralized control, but blurring domain boundaries creates privacy risks that must be addressed through both technical and policy measures (Landau et al., 2009; Carretero et al., 2018).

11. Results and Discussion

This study explored the concepts and architectural nuances of Distributed Database Systems and Federated Database Systems in the context of data management. A DDBS optimizes data distribution across multiple instances, focusing on replication, concurrency control, storage efficiency, and sharding; this approach enhances system performance and efficiency by decoupling query processing and transaction management from data storage.

An FDBS, on the other hand, prioritizes maintaining the autonomy of local databases within a heterogeneous environment, allowing local and global transactions to coexist. Its architecture incorporates a multilevel nested transaction approach, promoting concurrency in transaction execution. This design choice preserves the independence of local databases while fostering cooperation among diverse data sources.

The architectural variances extend beyond data distribution to storage mechanisms, processing models, schema autonomy, and query distribution. Understanding these distinctions is crucial for comprehending the dynamics of distributed and federated database systems in the evolving data management landscape. **Table 1** summarizes the comparison; consistent with the qualitative scope de-

scribed in Sections 2.4 and 7.2, this is an architectural summary rather than a quantitative comparison.

Table 1. Comparative overview of distributed DBS and federated DBS.

Aspect	Distributed DBS	Federated DBS
Data Distribution	Focus on replication, concurrency, storage, and sharding	Emphasizes autonomy and cooperation between heterogeneous DBs
Storage Mechanism	Emphasizes performance and scalability	Focuses on maintaining autonomy among heterogeneous databases
Processing Models	Prioritizes interactive analysis and visualizations	Prioritizes autonomy and concurrency in transaction executions
Schema Autonomy	Single global design; sites autonomous but not heterogeneous	Enables autonomy in a heterogeneous environment
Query Distribution	Data partitioning and decoupled processing	Integrates heterogeneous sources behind a single query interface

The study used several methodological lenses to evaluate scalability, performance, and fault tolerance. Volume and load metrics provide insight into a system's ability to handle increasing data and computational demand, while query response time and throughput, interpreted with due regard for performance variability, allow system design characteristics to be discussed qualitatively across systems. Robust strategies such as replication, logging, and state maintenance fortify system reliability, ensuring data availability, resilience, minimal disruption, and enhanced availability.

Consistent with the qualitative scope described in Sections 2.4 and 7.2, this comparison is narrative and architectural rather than a quantitative meta-analysis: no effect estimates, confidence intervals, or statistical significance tests are reported. Table 2 summarizes representative fault-tolerance mechanisms drawn from the specific published designs discussed in Section 8; per the assumptions stated there, the Federated DBS column reflects particular published designs (Newman, 2019; Pareek et al., 2019) evaluated under their own stated failure models, rather than mechanisms inherent to all federated database systems.

Table 2. Fault tolerance mechanisms.

Mechanism	Distributed DBS	Federated DBS (representative published designs)
Replication	Enhances data availability and fault tolerance through replicas	Utilizes local and global agreement algorithms for fault tolerance (Newman, 2019)
Logging	Periodically logs system state for recovery after failure	Uses logging-based mechanisms for recovery and consistency
Recovery Procedures	Hybrid model with primary and secondary (accelerator) systems	Local and global agreement groups (Newman, 2019); RAID-style redundancy in a specific published design (Pareek et al., 2019)

Query optimization enhances system efficiency through complementary techniques. Parallel processing accelerates query execution, improving analytical tasks and responsiveness. Indexing optimizes data access with strategically crafted indexes, reducing computational overhead. Query rewriting uses heuristic and learning-based methods to streamline the evaluation of multiple queries and reduce costs. Federated query optimization identifies relevant data sources and creates efficient execution plans, navigating the complexity of large-scale distributed environments. Table 3 summarizes these techniques.

Table 3. Query optimization techniques.

Technique	Description
Parallel Processing	Scalable query processing and engines over cloud databases
Indexing	Generation and merging of indexes for efficient database access
Query Rewriting	Heuristic and machine-learning-based approaches to optimizing query evaluation
Federated Query Optimization	Source selection, cost-based planning, and scalability handling in federated DBS

The architectural differences between distributed and federated databases significantly shape their operational dynamics and their response to workload demands. The tables above provide a concise overview of these variances, offering a nuanced understanding of the diverse approaches adopted by the two paradigms.

12. Recommendations for Future Research

Future research should explore the evolving landscape of database systems, focusing on advanced technologies and emerging paradigms. This includes:

- Integrating blockchain technology into distributed and federated systems to improve data security and decentralization, building on the emerging dichotomy and fusion between blockchains and distributed databases (Ruan et al., 2021).
- Exploring edge computing to address challenges and opportunities in localized data processing.
- Developing adaptive query optimization strategies that dynamically adjust to changing workloads.
- Ensuring ethical data practices within distributed and federated environments, and investigating quantum computing for unprecedented computational efficiency.
- Addressing standardization and interoperability challenges to facilitate seamless communication and data exchange.
- Considering the environmental impact of database systems, with a focus on optimizing energy consumption and resource utilization.
- Investigating human-computer interaction aspects to ensure user-friendly technologies that meet diverse stakeholder needs.

During the preparation of this work, the author used AI-based language tools solely to improve the clarity, grammar, and readability of the text. The author re-

viewed and edited the output as needed.

13. Conclusion

The number of organizations using database systems continues to grow, along with their expectation of operational integration, shareability, and controlled data access. Their desire to gain the effective functions of both DDBS and FDBS has become a necessity.

This study compared Distributed Database Systems and Federated Database Systems as interoperable databases, examining their fundamental concepts, architectural variances, and performance characteristics. Both paradigms offer unique advantages in data management, each with specific challenges. Distributed databases including NoSQL and NewSQL systems are robust in distributing data across multiple sites, enhancing communication, processing capability, and local control. They provide scalability, elasticity, and operational flexibility without making assumptions about the workload.

Federated Database Systems maintain a global schema, translating global queries into local queries and combining local results into a unified global view. FDBS has been instrumental in handling the challenges of integrating and managing complex data from multiple sources, offering a unified perspective on independent data warehouses and demonstrating, particularly in hybrid SQL/NoSQL settings, the advantages of using different database types in parallel.

The choice between the paradigms hinges on specific use cases and organizational requirements. Distributed databases excel where data distribution, replication, and concurrency control are paramount; under workloads and replication schemes that support effective parallelism, they can offer higher availability and performance than centralized systems, but they bring challenges including complex security management in heterogeneous environments and the difficulty of optimal distributed design. Federated databases prove valuable where seamless data exchange and pooled analyses from heterogeneous sources are demanded; they handle heterogeneous data models, data definition and manipulation facilities, and transaction management, but introduce complexities related to transaction determinism in specific deterministic designs, transaction aborts, and potential latency.

The architectural variances between the two paradigms emphasize their distinctive goals: distributed databases prioritize replication, concurrency, storage, and sharding for performance-driven design, whereas federated databases emphasize autonomy and cooperation among autonomous, heterogeneous participants. Their processing models reflect the same philosophies: large-volume interactive analysis on the distributed side, and multilevel nested transactions preserving autonomy on the federated side. Schema autonomy emerges as the pivotal difference: distributed systems partition data under a single top-down design, decoupling query processing and transaction management from storage, while federated systems integrate heterogeneous sources into a virtual repository, optimizing

query computation across autonomous domains. As hybrid, polystore, and cloud-native architectures mature, the boundary between the two paradigms is likely to blur further, making a firm grasp of their respective principles all the more valuable.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

Abdelhafiz, B. M. (2020). Distributed Database Using Sharding Database Architecture. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (pp. 1-17). IEEE. https://doi.org/10.1109/csde50874.2020.9411547

Acko, B., Sluban, B., Tasič, T., & Brezovnik, S. (2014). Performance Metrics for Testing Statistical Calculations in Interlaboratory Comparisons. *Advances in Production Engineering & Management, 9,* 44-52. https://doi.org/10.14743/apem2014.1.175

Ahmed, M. H. (2022). *Distributed Databases: A Review.* ScienceOpen Preprints. https://doi.org/10.14293/S2199-1006.1.SOR-.PPGGLZW.v1

Arnold, J., Glavic, B., & Raicu, I. (2019). A High-Performance Distributed Relational Database System for Scalable OLAP Processing. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (pp. 738-748). IEEE. https://doi.org/10.1109/ipdps.2019.00083

Bjeladinović, S., Marjanovic, Z., & Babarogic, S. (2020). A Proposal of Architecture for Integration and Uniform Use of Hybrid SQL/NoSQL Database Components. *Journal of Systems and Software, 168,* Article ID: 110633. https://doi.org/10.1016/j.jss.2020.110633

Blakeley, J. A. (1996). OLE DB: A Component DBMS Architecture. In *Proceedings of the Twelfth International Conference on Data Engineering* (pp. 203-204). IEEE Computer Society Press. https://doi.org/10.1109/icde.1996.492108

Boncz, P. (2021). Technical Perspective DIAMetrics: Benchmarking Query Engines at Scale. *ACM SIGMOD Record, 50,* 23-23. https://doi.org/10.1145/3471485.3471491

Bonifati, A., Chrysanthis, P. K., Ouksel, A. M., & Sattler, K. (2008). Distributed Databases and Peer-to-Peer Databases: Past and Present. *ACM SIGMOD Record, 37,* 5-11. https://doi.org/10.1145/1374780.1374781

Carretero, J., Izquierdo-Moreno, G., Vasile-Cabezas, M., & Garcia-Blas, J. (2018). Federated Identity Architecture of the European Eid System. *IEEE Access, 6,* 75302-75326. https://doi.org/10.1109/access.2018.2882870

Cattell, R. (2011). Scalable SQL and NoSQL Data Stores. *ACM SIGMOD Record, 39,* 12-27. https://doi.org/10.1145/1978915.1978919

Chen, C. M., Sun, W., & Rishe, N. (1998). Performance Comparison of Three Alternatives of Distributed Multidatabase Systems: A Global Query Perspective. In *Proceedings of the IEEE International Conference on Performance, Computing and Communications* (pp. 53-59). IEEE.

Chen, J., Childress, R., McIntosh, I., Africa, G., & Sitaramayya, A. (2011). A Service Management Architecture Component Model. In *2011 7th International Conference on Network and Service Management* (pp. 1-4). IEEE.

Chen, T., Chen, Y., Guo, Q., Temam, O., Wu, Y., & Hu, W. (2012). Statistical Performance Comparisons of Computers. In *IEEE International Symposium on High-Performance Comp Architecture* (pp. 1-12). IEEE. https://doi.org/10.1109/hpca.2012.6169043

Connolly, T., & Begg, C. (2002). *Database Systems: A Practical Approach to Design, Implementation, and Management* (3rd ed.). Addison-Wesley.

Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J. et al. (2013). Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems, 31,* 1-22. https://doi.org/10.1145/2491245

Creedon, S., & Roche, I. G. (2021). *Machine Learning-Based Query Optimization for Federated Databases* (U.S. Patent No. 10,896,176). U.S. Patent and Trademark Office. https://patents.google.com/patent/US10896176

Cuzzocrea, A. (2021). Scalable Query Processing and Query Engines over Cloud Databases: Models, Paradigms, Techniques, Future Challenges. In *33rd International Conference on Scientific and Statistical Database Management* (pp. 272-275). ACM. https://doi.org/10.1145/3468791.3472264

Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Pearson.

Deacon, A., Schek, H. J., & Weikum, G. (1994). Semantics-Based Multilevel Transaction Management in Federated Systems. In *Proceedings of 1994 IEEE 10th International Conference on Data Engineering* (pp. 452-461). IEEE. https://doi.org/10.1109/icde.1994.283066

DeWitt, D., & Gray, J. (1992). Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM, 35,* 85-98. https://doi.org/10.1145/129888.129894

Ding, H. (2022). Research on Query Optimization Algorithm of Multi Database System Based on Distributed Database. In *2022 3rd Asia-Pacific Conference on Image Processing, Electronics and Computers* (pp. 946-949). ACM. https://doi.org/10.1145/3544109.3544387

Dobrzelecki, B., Krause, A., Hume, A. C., Grant, A., Antonioletti, M., Alemu, T. Y. et al. (2010). Integrating Distributed Data Sources with OGSA-DAI DQP and Views. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 368,* 4133-4145. https://doi.org/10.1098/rsta.2010.0166

Duggan, J., Elmore, A. J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J. et al. (2015). The BigDAWG Polystore System. *ACM SIGMOD Record, 44,* 11-16. https://doi.org/10.1145/2814710.2814713

Elmasri, R., & Navathe, S. B. (2004). *Fundamentals of Database Systems* (4th ed.). Addison-Wesley.

Eslami, M., Mahmoodian, V., Dayarian, I., Charkhgard, H., & Tu, Y. (2020). Query Batching Optimization in Database Systems. *Computers & Operations Research, 121,* Article ID: 104983. https://doi.org/10.1016/j.cor.2020.104983

Fior, A. G., Meira, J. A., de Almeida, E. C., Coelho, R. G., Del Fabro, M. D., & Le Traon, Y. (2013). Under Pressure Benchmark for DDBMS Availability. *Journal of Information and Data Management, 4,* 266-278.

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). *Database Systems: The Complete Book* (2nd ed.). Pearson Prentice Hall.

Ghandeharizadeh, S., Bernstein, P. A., Borthakur, D., Huang, H., Menon, J., & Puri, S. (2023). Disaggregated Database Management Systems. In R. Nambiar, & M. Poess (Eds.), *Performance Evaluation and Benchmarking. TPCTC 2022* (pp. 33-48). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-29576-8_3

Gilbert, S., & Lynch, N. (2002). Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News, 33,* 51-59. https://doi.org/10.1145/564585.564601

Gordon, S. R., & Gordon, J. R. (1992). Organizational Hurdles to Distributed Database Management Systems (DDBMS) Adoption. *Information & Management, 22,* 333-345. https://doi.org/10.1016/0378-7206(92)90029-f

Gorton, I., Klein, J., & Nurgaliev, A. (2015). Architecture Knowledge for Evaluating Scalable Databases. In *2015 12th Working IEEE/IFIP Conference on Software Architecture* (pp. 95-104). IEEE. https://doi.org/10.1109/wicsa.2015.26

Harris, D., & Sidwell, D. (1994). Distributed Database Security. *Computers & Security, 13,* 547-557. https://doi.org/10.1016/0167-4048(94)90003-5

Howe, D. (2001). *Data Analysis for Database Design* (3rd ed.). Butterworth-Heinemann.

Huang, D., Liu, Q., Cui, Q., Fang, Z., Ma, X., Xu, F. et al. (2020). TiDB: A Raft-Based HTAP Database. *Proceedings of the VLDB Endowment, 13,* 3072-3084. https://doi.org/10.14778/3415478.3415535

Irving, S., Li, B., Chen, S., Peng, L., Zhang, W., & Duan, L. (2020). Computer Comparisons in the Presence of Performance Variation. *Frontiers of Computer Science, 14,* 21-41. https://doi.org/10.1007/s11704-018-7319-2

Israel, S. A. (2006). Performance Metrics: How and When. *Geocarto International, 21,* 23-32. https://doi.org/10.1080/10106040608542380

Karnstedt, M., Sattler, K., & Hauswirth, M. (2012). Scalable Distributed Indexing and Query Processing over Linked Data. *Journal of Web Semantics, 10,* 3-32. https://doi.org/10.1016/j.websem.2011.11.010

Kaseb, M. R., Haytamy, S. S., & Badry, R. M. (2021). Distributed Query Optimization Strategies for Cloud Environment. *Journal of Data, Information and Management, 3,* 271-279. https://doi.org/10.1007/s42488-021-00057-z

Katta, R., Ali, A. A., Chramcov, B., Krayem, S., & Jasek, R. (2021). Formal Development of Fault Tolerance by Replication of Distributed Database Systems. In R. Silhavy (ed.), *Software Engineering and Algorithms* (pp. 293-306). Springer International Publishing. https://doi.org/10.1007/978-3-030-77442-4_25

Kaur, N., Singh, R., Misra, M., & Sarje, A. K. (2007). Secure Transaction Management Protocols for MLS/DDBMS. In P. McDaniel, & S. K. Gupta (Eds.), *Information Systems Security* (pp. 219-233). Springer. https://doi.org/10.1007/978-3-540-77086-2_17

Kern, R., Kozierkiewicz, A., & Pietranik, M. (2020). The Data Richness Estimation Framework for Federated Data Warehouse Integration. *Information Sciences, 513,* 397-411. https://doi.org/10.1016/j.ins.2019.10.046

Kim, S., & Moon, B. (2018). Federated Database System for Scientific Data. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management* (pp. 1-4). ACM. https://doi.org/10.1145/3221269.3222332

Klisarova-Belcheva, S. (2015). Distributed Databases—Some Approaches, Models and Current Trends. *Trakia Journal of Science, 13,* 414-419. https://doi.org/10.15547/tjs.2015.s.01.071

Kossmann, J., Papenbrock, T., & Naumann, F. (2021). Data Dependencies for Query Optimization: A Survey. *The VLDB Journal, 31,* 1-22. https://doi.org/10.1007/s00778-021-00676-3

Koutris, P., Salihoglu, S., & Suciu, D. (2018). Algorithmic Aspects of Parallel Query Processing. In *Proceedings of the 2018 International Conference on Management of Data* (pp. 1659-1664). ACM. https://doi.org/10.1145/3183713.3197388

Lamba, A., & Sharma, G. (2011). The Secure Concurrency Control Protocol for DDBMS. *International Journal of Computer Science and Technology, 2,* 347-350.

Landau, S., Le Van Gong, H., & Wilton, R. (2009). Achieving Privacy in a Federated Iden-

tity Management System. In R. Dingledine, & P. Golle (Eds.), *Financial Cryptography and Data Security* (pp. 51-70). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03549-4_4

Loesing, S., Pilman, M., Etter, T., & Kossmann, D. (2015). On the Design and Scalability of Distributed Shared-Data Databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (pp. 663-676). ACM. https://doi.org/10.1145/2723372.2751519

Malard-Adam, J., Harms, J., & Medema, W. (2022). Distributed Databases for Citizen Science. In *2022 EGU General Assembly Conference Abstracts* (pp. EGU22-5147). Copernicus GmbH. https://doi.org/10.5194/egusphere-egu22-5147

McCann, J. A., & Crane, J. S. (1998). Component DBMS Architecture for Nomadic Computing. In S. M. Embury, N. J. Fiddian, W. A. Gray, & A. C. Jones (Eds.), *Advances in Databases* (pp. 175-176). Springer Berlin Heidelberg. https://doi.org/10.1007/bfb0053484

Muilu, J., Peltonen, L., & Litton, J. (2007). The Federated Database—A Basis for Biobank-Based Post-Genome Studies, Integrating Phenome and Genome Data from 600 000 Twin Pairs in Europe. *European Journal of Human Genetics, 15,* 718-723. https://doi.org/10.1038/sj.ejhg.5201850

Newman, R. A. (2019). *Fault-Tolerant Federated Distributed Database* (U.S. Patent No. 10,275,400). U.S. Patent and Trademark Office. https://patents.google.com/patent/US10275400

Özsu, M. T., & Valduriez, P. (2020). *Principles of Distributed Database Systems* (4th ed.). Springer. https://doi.org/10.1007/978-3-030-26253-2

Pareek, S., Sharma, N., & Mary A., G. (2019). Fault Tolerance in Distributed Database Management Systems—Improving Reliability with Raid. In *2019 Innovations in Power and Advanced Computing Technologies (i-PACT)* (pp. 1-5). IEEE. https://doi.org/10.1109/i-pact44901.2019.8960197

Parent, C., & Spaccapietra, S. (1998). Issues and Approaches of Database Integration. *Communications of the ACM, 41,* 166-178. https://doi.org/10.1145/276404.276408

Pasupuleti, K. K., Klots, B., Nagarajan, V., Kandukuri, A., & Agarwal, N. (2022). High Availability Framework and Query Fault Tolerance for Hybrid Distributed Database Systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (pp. 3451-3460). ACM. https://doi.org/10.1145/3511808.3557086

Peng, P., Ge, Q., Zou, L., Ozsu, M. T., Xu, Z., & Zhao, D. (2021). Optimizing Multi-Query Evaluation in Federated RDF Systems. *IEEE Transactions on Knowledge and Data Engineering, 33,* 1692-1707. https://doi.org/10.1109/tkde.2019.2947050

Pereira, Ó. M., Simões, D., & Aguiar, R. L. (2016). Fault Tolerance Logging-Based Model for Deterministic Systems. In *Proceedings of the 5th International Conference on Data Management Technologies and Applications* (pp. 119-126). SCITEPRESS—Science and Technology Publications. https://doi.org/10.5220/0005979101190126

Pirk, H. (2015). ...Like Commanding an Anthill: A Case for Micro-Distributed (Data) Management Systems. *ACM SIGMOD Record, 44,* 53-58. https://doi.org/10.1145/2814710.2814720

Rabl, T., & Jacobsen, H. (2017). Query Centric Partitioning and Allocation for Partially Replicated Database Systems. In *Proceedings of the 2017 ACM International Conference on Management of Data* (pp. 315-330). ACM. https://doi.org/10.1145/3035918.3064052

Radeke, E., Böttger, R., Burkert, B., Engel, Y., Kachel, G., Kolmschlag, S. et al. (1995).

Efendi: Federated Database System of Cadlab. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data—SIGMOD '95* (pp. 481). ACM Press. https://doi.org/10.1145/223784.223894

Ren, K., Thomson, A., & Abadi, D. J. (2014). An Evaluation of the Advantages and Disadvantages of Deterministic Database Systems. *Proceedings of the VLDB Endowment, 7,* 821-832. https://doi.org/10.14778/2732951.2732955

Ruan, P., Dinh, T. T. A., Loghin, D., Zhang, M., Chen, G., Lin, Q. et al. (2021). Blockchains Vs. Distributed Databases. In *Proceedings of the 2021 International Conference on Management of Data* (pp. 1504-1517). ACM. https://doi.org/10.1145/3448016.3452789

Samaraweera, G. D., & Chang, J. M. (2021). Security and Privacy Implications on Database Systems in Big Data Era: A Survey. *IEEE Transactions on Knowledge and Data Engineering, 33,* 239-258. https://doi.org/10.1109/tkde.2019.2929794

Sari, A., & Akkaya, M. (2015). Fault Tolerance Mechanisms in Distributed Systems. *International Journal of Communications, Network and System Sciences, 8,* 471-482. https://doi.org/10.4236/ijcns.2015.812042

Schwarte, A., Haase, P., Hose, K., Schenkel, R., & Schmidt, M. (2011). FedX: A Federation Layer for Distributed Query Processing on Linked Open Data. In G. Antoniou et al. (Eds.), *The Semanic Web: Research and Applications* (pp. 481-486). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-21064-8_39

Sethi, R., Traverso, M., Sundstrom, D., Phillips, D., Xie, W., Sun, Y. et al. (2019). Presto: SQL on Everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)* (pp. 1802-1813). IEEE. https://doi.org/10.1109/icde.2019.00196

Sheth, A. P., & Larson, J. A. (1990). Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys, 22,* 183-236. https://doi.org/10.1145/96602.96604

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2002). *Database System Concepts* (4th ed.). McGraw-Hill.

Souza, A. M., Weigert, R. d. A. S., Machado de Sousa, E. P., Tassoni Andrietta, L., & Ventura, R. V. (2021). Practical Implications of Using Non-Relational Databases to Store Large Genomic Data Files and Novel Phenotypes. *Journal of Animal Breeding and Genetics, 139,* 100-112. https://doi.org/10.1111/jbg.12644

Starič, A., Demšar, J., & Zupan, B. (2015). Concurrent Software Architectures for Exploratory Data Analysis. *WIREs Data Mining and Knowledge Discovery, 5,* 165-180. https://doi.org/10.1002/widm.1155

Stojanovic, A., Horvat, M., & Kovacevic, Z. (2022). An Overview of Data Integration Principles for Heterogeneous Databases. In *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)* (pp. 1111-1116). IEEE. https://doi.org/10.23919/mipro55190.2022.9803579

Taft, R., Sharif, I., Matei, A., VanBenschoten, N., Lewis, J., Grieger, T. et al. (2020). CockroachDB: The Resilient Geo-Distributed SQL Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (pp. 1493-1509). ACM. https://doi.org/10.1145/3318464.3386134

Valduriez, P. (2011). Principles of Distributed Data Management in 2020? In A. Hameurlain, S. W. Liddle, K. D. Schewe, & X. Zhou (Eds.), *Database and Expert Systems Applications* (pp. 1-11). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23088-2_1

Verbitski, A., Gupta, A., Saha, D., Brahmadesam, M., Gupta, K., Mittal, R. et al. (2017). Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational

Databases. In *Proceedings of the 2017 ACM International Conference on Management of Data* (pp. 1041-1052). ACM. https://doi.org/10.1145/3035918.3056101

Vu, X. S., Ait-Mlouk, A., Elmroth, E., & Jiang, L. (2019). Graph-Based Interactive Data Federation System for Heterogeneous Data Retrieval and Analytics. In *The World Wide Web Conference* (pp. 3595-3599). ACM. https://doi.org/10.1145/3308558.3314138

Yoshida, M., Mizumachi, K., Wakino, A., Oyake, I., & Matsushita, Y. (1985). Time and Cost Evaluation Schemes of Multiple Copies of Data in Distributed Database Systems. *IEEE Transactions on Software Engineering, 11,* 954-959. https://doi.org/10.1109/tse.1985.232829

Zhao, J. L., Segev, A., & Chatterjee, A. (1995). A Universal Relation Approach to Federated Database Management. In *Proceedings of the Eleventh International Conference on Data Engineering* (pp. 261-270). IEEE Computer Society Press. https://doi.org/10.1109/icde.1995.380385

Zhou, Y., & Kang, K. (2012). A Federated Approach for Increasing the Timely Throughput of Real-Time Data Services. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium* (pp. 163-172). IEEE. https://doi.org/10.1109/rtas.2012.8

Zhuang, H., Lu, K., Li, C., Sun, M., Chen, H., & Zhou, X. (2015). Design of a More Scalable Database System. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 1213-1216). IEEE. https://doi.org/10.1109/ccgrid.2015.70