FEDERATED DATA MANAGEMENT

2026 Edition

Contents

Chapter 1

Federated Data Management

1.1 Introduction

Modern enterprises and research institutions rarely operate with a single, monolithic database. Instead, organizational data is distributed across multiple heterogeneous systems—relational databases, NoSQL stores, data lakes, cloud-hosted services, and legacy mainframe applications.

★ Definition: Federated Data Management

Federated Data Management (FDM) provides an architectural paradigm for integrating diverse, autonomous data sources into a coherent, queryable whole *without* requiring physical data migration or centralization.

This chapter provides a comprehensive treatment of federated data management: its architectural foundations, the heterogeneity challenges it addresses, query processing strategies, transaction semantics, security considerations, and real-world applications. The discussion is aimed at practitioners who must design, deploy, or maintain data integration infrastructure in complex organizational settings.

1.1.1 Distributed Data Management Overview

A **distributed database system** partitions or replicates data across multiple networked nodes. Key properties include:

- **Data Distribution:** Data is fragmented (horizontally, vertically, or by hybrid partitioning) across sites.
- **Replication:** Copies of data may exist at multiple sites for fault tolerance and reduced latency.
- **Transparency:** The distribution is ideally invisible to the end user—location transparency, replication transparency, and fragmentation transparency.
- **Coordination:** A unified DBMS layer coordinates query execution, concurrency control, and recovery across all sites.

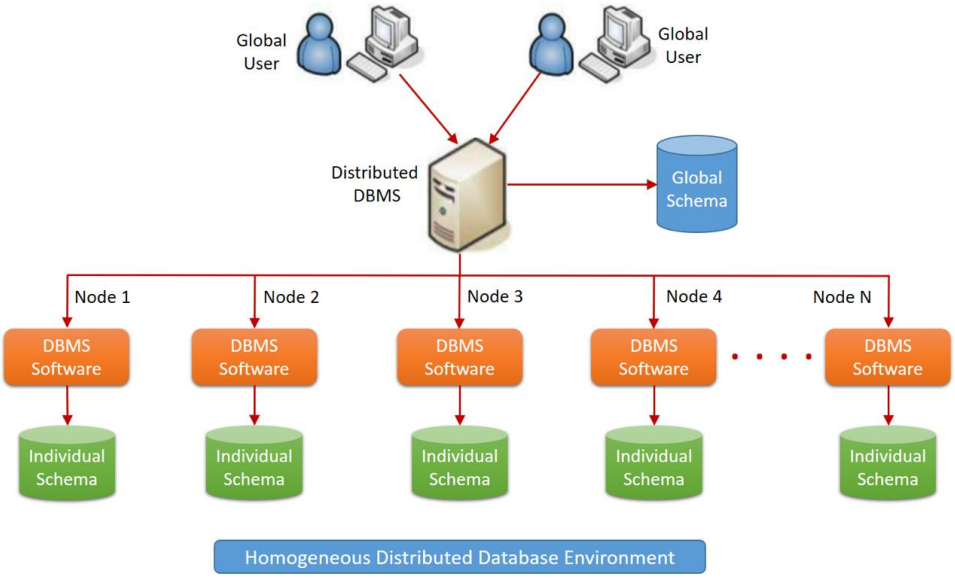


Figure 1.1: Architecture of a traditional distributed database system.

While distributed databases assume a *homogeneous* software stack (all sites run the same DBMS), real-world environments are far more heterogeneous. This gap motivates federated approaches.

1.1.2 Need for Federated Data Management

Several forces drive the adoption of federated data management:

1. **Organizational Mergers and Acquisitions:** When two companies merge, their data systems rarely share the same schema, DBMS vendor, or data model. A federated approach allows coexistence without costly migration.

2. **Legacy System Integration:** Many organizations rely on decades-old mainframe systems that cannot be easily replaced. Federation places a modern query interface atop these legacy stores.

3. **Multi-Cloud and Hybrid Environments:** With data spanning AWS, Azure, GCP, and on-premises infrastructure, a federated layer provides unified access.

4. **Regulatory Constraints:** Data sovereignty laws (e.g., GDPR, HIPAA) may prohibit moving data across jurisdictional boundaries, making physical centralization impossible.

5. **Agility and Time-to-Value:** Setting up a federated query layer can be orders of magnitude faster than a full ETL migration to a data warehouse.

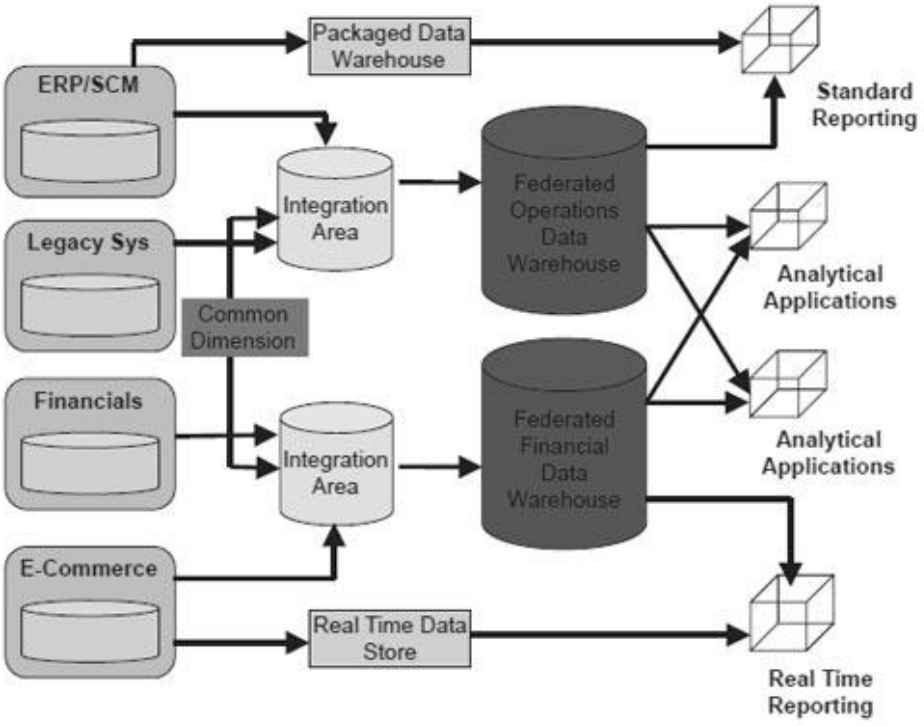


Figure 1.2: Federated Data Warehouse Architecture showing integration of ERP/SCM, Legacy Systems, Financials, and E-Commerce sources.

Table 1.1: Comparison: Centralized vs. Distributed vs. Federated Data Management

Criterion	**Centralized**	**Distributed**	**Federated**
Data Location	Single site	Multiple sites, same DBMS	Multiple sites, heterogeneous DBMS
Autonomy	None	Low	High—local systems retain control
Schema	Single global schema	Partitioned global schema	Virtual global schema over local schemas
Heterogeneity	None	Minimal	Full (data model, schema, platform)
Scalability	Vertical only	Horizontal	Horizontal + heterogeneous
Typical Use Case	Small/medium apps	Large homogeneous apps	Enterprise integration, data mesh

1.2 Federated Data System Architecture

The architecture of a federated database system (FDBS) defines how autonomous, heterogeneous data sources are interconnected and presented as a unified system. This section

examines the core concepts, coupling strategies, and the mediator-wrapper pattern.

1.2.1 Federated Database Concepts

A federated database system is formally defined as a collection of cooperating but autonomous component database systems (CDBSs). The key conceptual elements are:

- **Component Database Systems (CDBS):** The individual, pre-existing databases that participate in the federation. Each CDBS has its own data model, schema, query language, and transaction manager.
- **Federation Layer:** The middleware that provides integrated access. It includes components for schema mapping, query decomposition, result integration, and transaction coordination.
- **Global (Federated) Schema:** A virtual, unified schema that users and applications interact with. It abstracts away the differences among component schemas.
- **Local Export Schemas:** A subset of each component's local schema that is "exported" for federation access. Not all local data need be exposed.

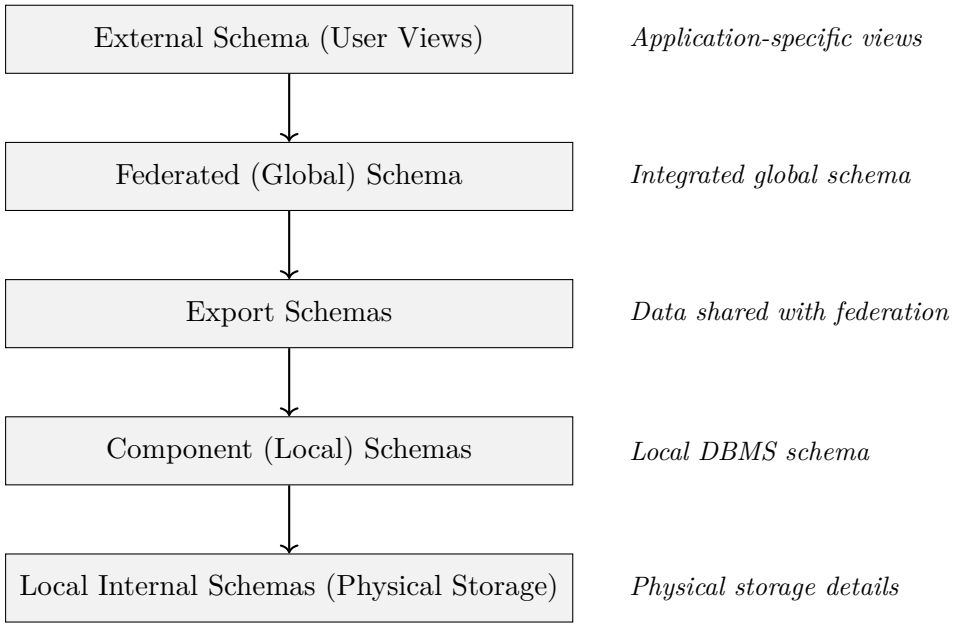


Figure 1.3: Five-Level Schema Architecture for Federated Database Systems (after Sheth & Larson, 1990).

Loosely Coupled Federation

In a **loosely coupled** federation, there is no predefined global schema. Instead, users construct ad hoc integrations by directly referencing multiple export schemas. Characteristics include:

- No central schema administration.
- Users bear the burden of schema mapping.

- Greater flexibility and autonomy for component systems.

- Weaker consistency; suitable for exploratory or read-mostly workloads.

Table 1.2: Tightly Coupled vs. Loosely Coupled Federated Systems

Feature	**Tightly Coupled**	**Loosely Coupled**
Global Schema	Predefined by administrator	None; ad hoc integration
Administration	Centralized	Decentralized
Schema Evolution	Complex to manage	Flexible but user-burdened
Consistency	Strong	Eventual or best-effort
Autonomy	Moderate	High
Best For	Stable enterprise integration	Research, data exploration

1.2.2 Mediator–Wrapper Architecture

The **mediator–wrapper** architecture (Wiederhold, 1992) is the most widely adopted pattern for building federated data systems. It cleanly separates the concerns of heterogeneity resolution from query integration.

- **Wrappers (Adapters):** Each component data source is fronted by a wrapper that translates the source's native interface (query language, data model, access protocol) into a common canonical form. The wrapper hides source-specific details from the rest of the federation.

- **Mediator:** The mediator sits above the wrappers and performs schema integration, query decomposition, sub-query routing, and result merging. It operates on the canonical data model and presents the federated schema to external applications.

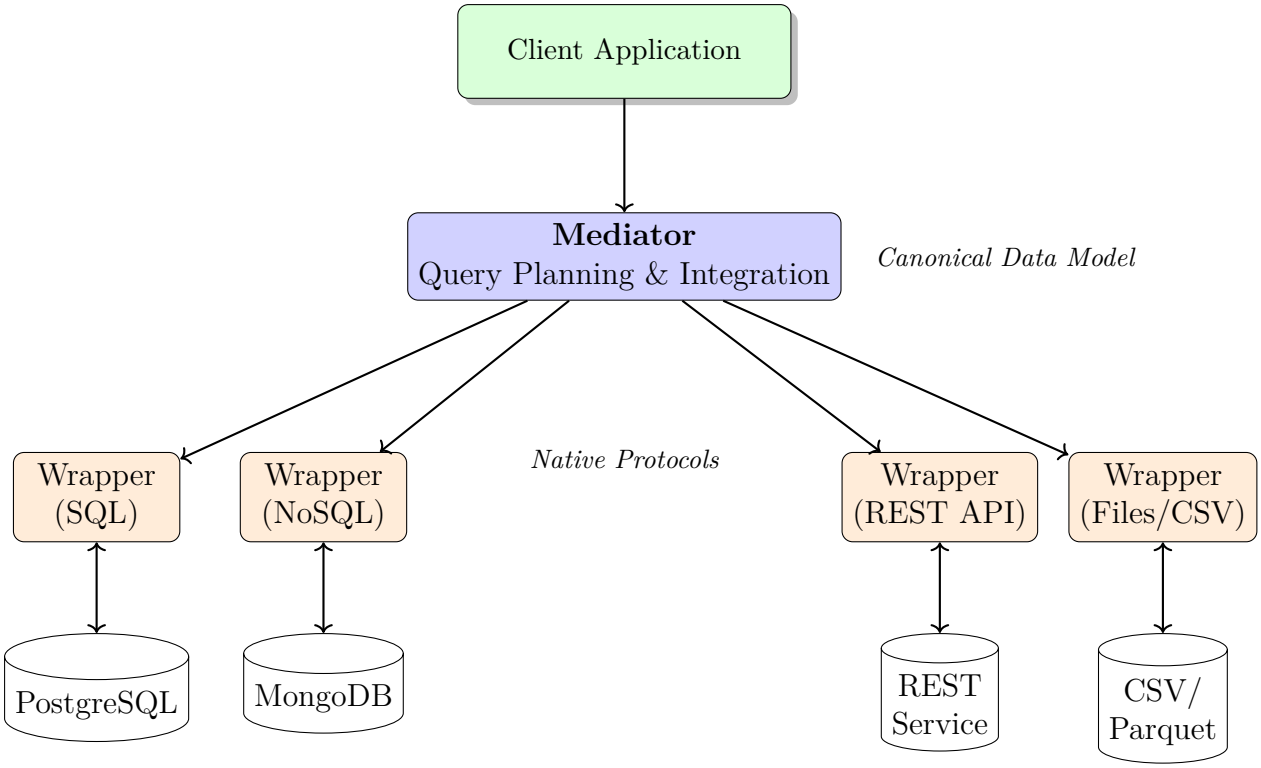


Figure 1.4: The Mediator–Wrapper architecture for federated data access.

★ Engineering Advantages: Mediator–Wrapper Pattern

The mediator–wrapper pattern offers three key engineering advantages:

1. Adding a new data source requires implementing only a new *wrapper* — the mediator remains unchanged.
2. The mediator can be enhanced with caching, access control, and logging without modifying any wrapper.
3. The architecture aligns naturally with microservice deployment patterns.

1.3 Data and Schema Heterogeneity

Heterogeneity is the defining challenge of federated data management. When data sources are designed independently—by different teams, at different times, using different technologies—mismatches arise at every level: data models, schemas, data types, naming conventions, units of measurement, and semantics.

1.3.1 Types of Heterogeneity

Heterogeneity in federated systems can be classified along several dimensions:

System-Level Heterogeneity

Differences in the underlying DBMS platforms:

- **Data Model Heterogeneity:** Relational vs. document-oriented vs. graph vs. key-value stores.
- **Query Language Heterogeneity:** SQL vs. MongoDB Query Language vs. Cypher vs. SPARQL.
- **Platform Heterogeneity:** Differences in operating systems, hardware, network protocols.

Schema-Level Heterogeneity

Even when two sources use the same data model (e.g., both are relational), their schemas often differ:

- **Naming Conflicts:** The same real-world concept is represented by different attribute names (e.g., `cust_id` vs. `customer_number`).
- **Structural Conflicts:** Information is organized differently (e.g., a single table in one source vs. a normalized two-table design in another).
- **Data Type Conflicts:** The same attribute uses different types (e.g., `VARCHAR(10)` for zip codes vs. `INTEGER`).

Semantic Heterogeneity

The most difficult form—same terms have different meanings, or different terms refer to the same concept:

- `"revenue"` might mean gross revenue in one system and net revenue in another.
- Units of measurement differ (metric vs. imperial; USD vs. EUR).
- Temporal semantics vary (fiscal year start date, time zones).

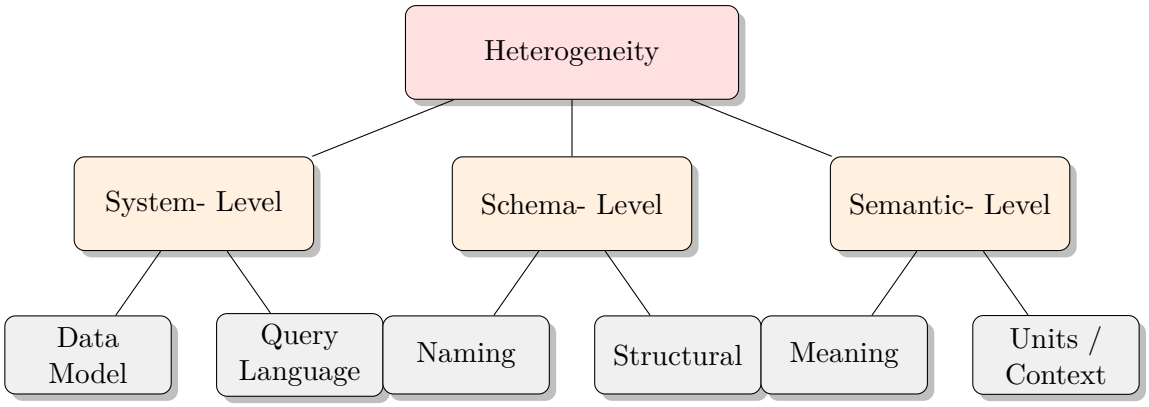


Figure 1.5: Taxonomy of heterogeneity types in federated data systems.

1.3.2 Schema Conflicts

Schema conflicts are concrete manifestations of heterogeneity that must be resolved during schema integration. The following table categorizes common conflict types with examples.

Table 1.3: Common Schema Conflicts and Resolution Strategies

Conflict Type	Category	Example	Resolution Strategy
Synonyms	Naming	`emp_name` vs. `employee_name`	Canonical naming via mapping table
Homonyms	Naming	`address` (home) vs. `address` (office)	Disambiguation via context qualifiers
Table vs. Attribute	Structural	`Phone` table vs. `phone_number` column	Schema restructuring in global view
Missing Attributes	Structural	Source A has `middle_name`; Source B does not	NULL imputation or derived defaults
Data Type Mismatch	Type	`INT` vs. `VARCHAR` for zip codes	Type casting functions in wrappers
Unit Mismatch	Semantic	Price in USD vs. EUR	Conversion functions with metadata

★ Schema Conflict Resolution

Resolving schema conflicts requires a combination of **automated schema matching algorithms**, **domain expert review**, and **well-maintained metadata catalogues**. No single technique is sufficient in isolation.

1.4 Global Schema and Metadata Management

The global schema is the cornerstone of a tightly coupled federated system. It provides the unified logical view through which users interact with all federated data. Designing, maintaining, and evolving this schema is one of the most challenging aspects of federated data management.

1.4.1 Global Schema Design

Global schema design follows a multi-step integration process:

1. **Pre-integration:** Analyze each component schema—understand its entities, relationships, constraints, and semantics. Document assumptions and domain context.
2. **Correspondence Identification:** Identify which elements across schemas refer to the same real-world entities or relationships. This step relies on schema matching algorithms and domain expertise.

3. **Conflict Resolution:** Resolve naming, structural, semantic, and type conflicts identified between corresponding elements.

4. **Merging and Restructuring:** Combine the resolved schemas into a single, coherent global schema. This may involve introducing supertype/subtype hierarchies, union views, or virtual join tables.

5. **Validation:** Verify that the global schema correctly represents all component data, supports required queries, and preserves semantic integrity.

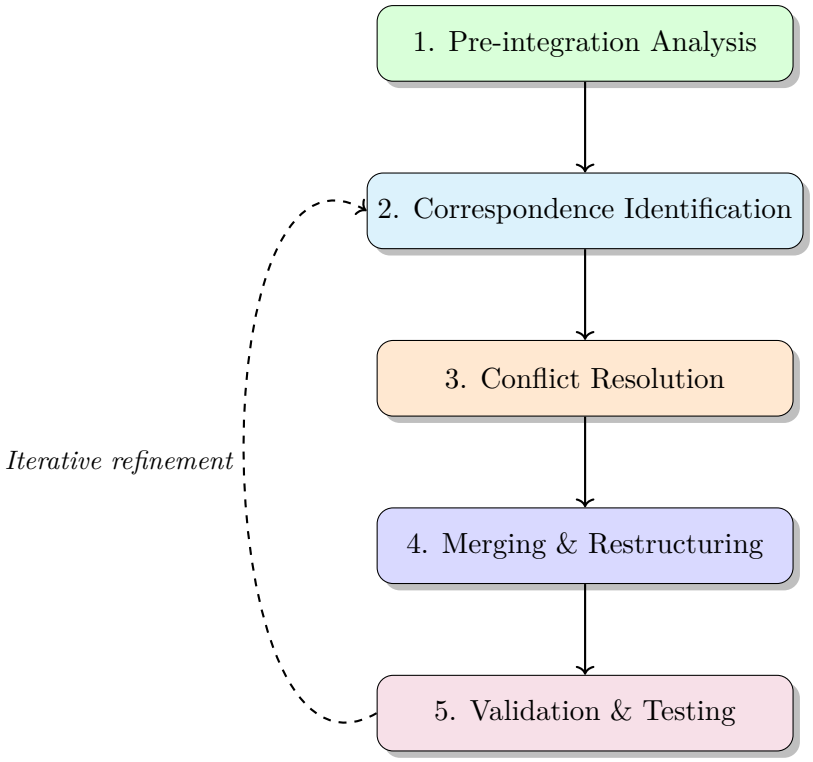


Figure 1.6: Global schema design process.

1.4.2 Local Schemas

Each component database maintains its own **local schema**, which is the native schema as defined within its DBMS. The federation does not modify local schemas. Instead, it operates through two key abstractions:

- **Component Schema:** A canonical representation of the local schema translated into the federation's common data model.

- **Export Schema:** The subset of the component schema that the local data owner chooses to make available to the federation. This mechanism enforces local autonomy and data governance—owners control exactly what is shared.

1.4.3 Metadata and Catalog Management

A federated system requires a rich **metadata catalog** that stores:

- **Schema Mappings:** Correspondences between global schema elements and local/-export schema elements, including transformation functions.
- **Data Source Registry:** Connection parameters, capabilities (supported query operators, join types), and availability status of each component data source.
- **Statistics:** Cardinality estimates, data distributions, and access costs per source—essential for the federated query optimizer.
- **Lineage and Provenance:** Tracking which local sources contributed to each global result, enabling auditability and trust.
- **Semantic Annotations:** Ontology mappings, business glossary terms, and data quality indicators.

Table 1.4: Key Metadata Catalog Components

Catalog Component	Description	Used By
Schema Mappings	Global-to-local attribute correspondences	Query Translator
Source Capabilities	Supported operators, join types per source	Query Optimizer
Statistics	Row counts, selectivity, I/O cost estimates	Query Optimizer
Access Control Policies	User permissions mapped to export schemas	Security Module
Data Lineage Records	Origin tracking per result tuple	Audit / Compliance

1.5 Query Processing in Federated Systems

Query processing in a federated environment is significantly more complex than in a single-site or homogeneous distributed database. The federated query engine must decompose a global query into sub-queries targeting individual data sources, translate each sub-query into the native language of the target, execute them in parallel where possible, and integrate the results.

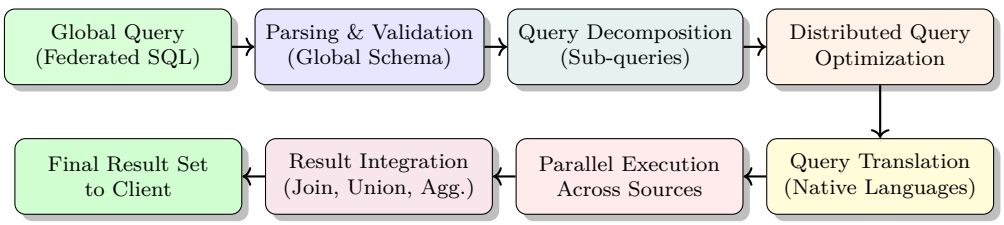


Figure 1.7: Federated query processing pipeline (8 stages).

1.5.1 Query Decomposition

Query decomposition transforms a query expressed against the *global schema* into a set of sub-queries against *component schemas*. The decomposition process involves:

1. **Schema Lookup:** Identify which global schema elements are referenced and map them to their corresponding local sources via the metadata catalog.
2. **Predicate Pushdown:** Push selection predicates (`WHERE` clauses) as close to the data sources as possible. This is critical for minimizing data transfer across the network.
3. **Projection Pushdown:** Request only the attributes needed, rather than full rows, from each source.
4. **Join Decomposition:** If a join spans multiple sources, determine which portions can be evaluated locally (intra-source joins) and which must be evaluated at the mediator (inter-source joins).

Example: Consider a federated query that joins `Customer` data from a PostgreSQL database with `Order` data from a MongoDB collection:

```
SELECT c.name, o.total FROM Customer c JOIN Orders o ON c.id = o.cust_id WHERE o.total
                                    > 1000
```

The decomposer generates: (a) a sub-query to PostgreSQL for `Customer` (`id`, `name`), and (b) a sub-query to MongoDB for `Orders` where `total > 1000` (`cust_id`, `total`). The join is performed at the mediator.

1.5.2 Query Translation

After decomposition, each sub-query is in the federation's canonical query representation. The **query translator** converts it into the native query language of the target data source:

- PostgreSQL sub-query → standard SQL.
- MongoDB sub-query → MongoDB aggregation pipeline (BSON).
- REST API sub-query → HTTP GET with query parameters.
- File-based sub-query → File reader with filter/projection logic.

Table 1.5: Query Translation Examples Across Source Types

Source Type	Native Interface	Translated Query Form
PostgreSQL	SQL	`SELECT name FROM customer WHERE id IN (...)`
MongoDB	Aggregation Pipeline	`db.orders.aggregate([{$match: {total: {$gt: 1000}}}])`
REST API	HTTP + JSON	`GET /api/orders?min_total=1000`
CSV/Parquet	File Reader	Scan with predicate filter on column `total`

1.5.3 Distributed Query Optimization

The federated query optimizer faces a search space far larger than a single-site optimizer due to:

- **Source Selection:** When the same data is available from multiple sources (e.g., replicas), the optimizer must choose the source with the lowest expected cost.
- **Join Ordering:** Determining the optimal order to join results from different sources. In cross-source joins, the cost is dominated by network transfer and intermediate result sizes.
- **Data Shipping vs. Query Shipping:** Should the mediator pull raw data and perform the join centrally (data shipping), or push the join to a capable source (query shipping)?
- **Parallelism:** Exploiting independent sub-queries that can execute concurrently at different sources.

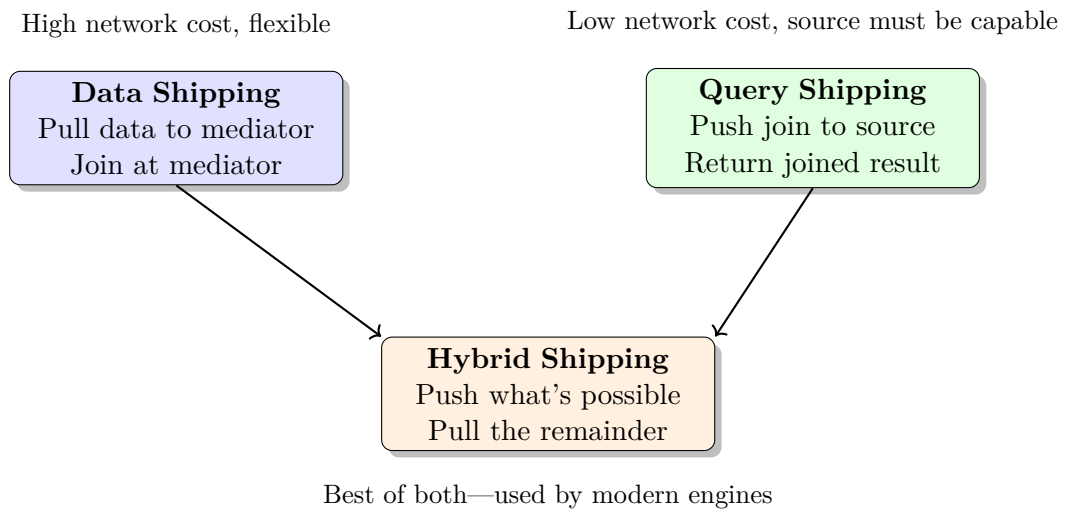


Figure 1.8: Query execution strategies: data shipping vs. query shipping vs. hybrid.

★ Modern Federated Query Engines

Modern federated engines such as **Presto/Trino**, **Dremio**, and **Denodo** employ *cost-based optimization* using statistics from the metadata catalog. They estimate network bytes transferred, CPU time, and I/O cost for each candidate plan — selecting the plan with the globally lowest estimated cost.

1.6 Transaction and Concurrency Management

Ensuring data integrity during concurrent access and system failures is paramount. In a federated environment, the federation layer does not own or fully control the local transaction managers. Each component database independently manages its own ACID properties, and the federated system must coordinate across these autonomous managers.

1.6.1 Distributed Transactions

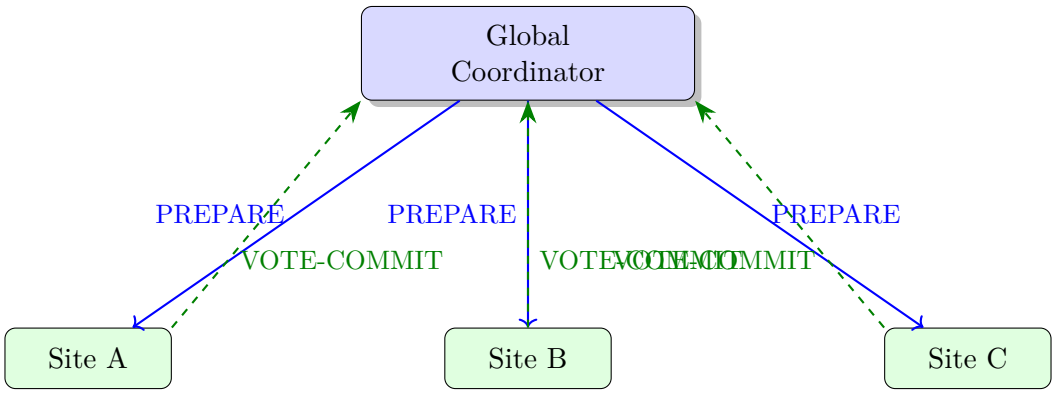
A **global transaction** spans multiple component databases. The key challenges include:

- **Atomicity:** Either all sub-transactions at all participating sites commit, or all abort.
- **Durability:** Once a global transaction commits, its effects must persist at all sites.
- **Coordination Overhead:** Communication between the global coordinator and local sites incurs latency and is vulnerable to network partitions.

Two-Phase Commit Protocol (2PC)

The **Two-Phase Commit (2PC)** protocol is the classical solution for distributed atomic commit:

1. **Phase 1 — Prepare (Voting):** The coordinator sends `PREPARE` to all participating sites. Each site responds with `VOTE-COMMIT` or `VOTE-ABORT`.
2. **Phase 2 — Commit/Abort (Decision):** If all sites vote to commit, the coordinator broadcasts `COMMIT`. If any site votes to abort, it broadcasts `ABORT`.



Phase 2: Coordinator broadcasts COMMIT or ABORT

Figure 1.9: Two-Phase Commit (2PC) protocol flow.

Limitations of 2PC in federated systems:

- **Blocking:** If the coordinator fails after `PREPARE` but before the decision, participants hold locks indefinitely.
- **Autonomy Violation:** Strict 2PC requires local sites to relinquish commit control to the global coordinator.
- **Performance:** The protocol requires $O(3n)$ messages for n participants.

Saga Pattern

For long-running transactions, the **Saga pattern** decomposes a global transaction into a sequence of local transactions, each with a corresponding **compensating transaction** that logically undoes its effects if a subsequent step fails.

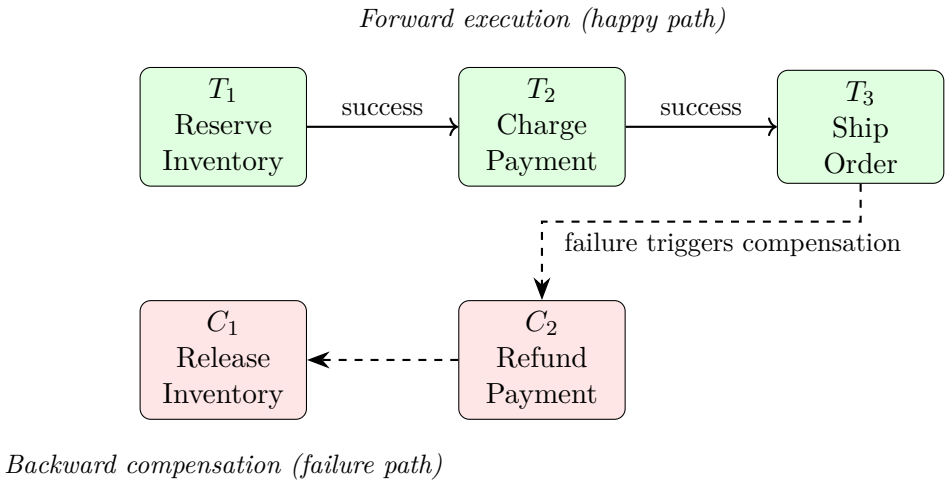


Figure 1.10: The Saga pattern: forward execution and compensating rollback.

1.6.2 Concurrency Controls

Local vs. Global Serializability

★ Local vs. Global Serializability

Each component DBMS independently enforces **local serializability** using its own mechanism (2PL, MVCC, OCC). However, locally serializable schedules do *not* automatically guarantee **global serializability**. Cross-site serialization anomalies can arise when global transactions interleave differently at different sites — making global concurrency control an open challenge in federated systems.

Table 1.6: Concurrency Control Mechanisms in Federated Systems

Mechanism	Description	Federated Applicability
Two-Phase Locking (2PL)	Acquire all locks before release	Requires global lock manager; high overhead
MVCC	Multi-version reads avoid blocking	Local only; global consistency needs extra coordination
Optimistic CC	Validate at commit; abort on conflict	Low overhead; high abort rate under contention
Ticket-Based	Global tickets enforce serialization order	Effective for tightly coupled federations

1.7 Security and Privacy Issues

Security in a federated system is inherently more complex than in a centralized database. The integration of varied autonomous domains means that security policies, authentication mechanisms, and data classification schemas must be harmonized without compromising the security posture of any single participant.

1.7.1 Access Control

Authentication and Identity Federation

- **Single Sign-On (SSO):** Users authenticate once at the federation level; credentials are propagated via OAuth 2.0, SAML, or OpenID Connect.
- **Credential Mapping:** The federation maintains a secure mapping from global user identities to local-source credentials.

Authorization Models

- **Role-Based Access Control (RBAC):** Global roles (e.g., `Analyst`, `Auditor`) are mapped to specific permissions on export schemas.
- **Attribute-Based Access Control (ABAC):** Policies based on user attributes, data sensitivity, and environmental conditions.
- **Policy Enforcement Point:** The mediator evaluates access policies before routing sub-queries.

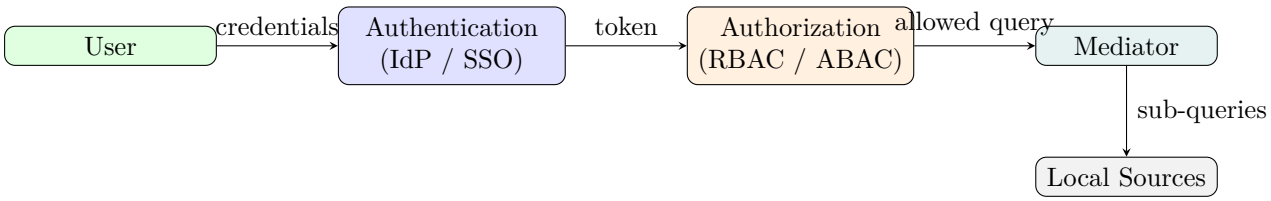


Figure 1.11: Access control flow in a federated data system.

1.7.2 Secure Data Sharing

- **Encryption in Transit:** All communication uses TLS 1.3 or equivalent.
- **Encryption at Rest:** Cached or materialized data at the mediator is encrypted (AES-256).
- **Data Masking and Anonymization:** Wrappers dynamically mask PII per GDPR/HIPAA requirements before results enter the federation.
- **Differential Privacy:** Calibrated noise added to aggregate results prevents inference of individual records.
- **Audit Logging:** Every data access is logged with user identity, query text, sources accessed, and timestamps.

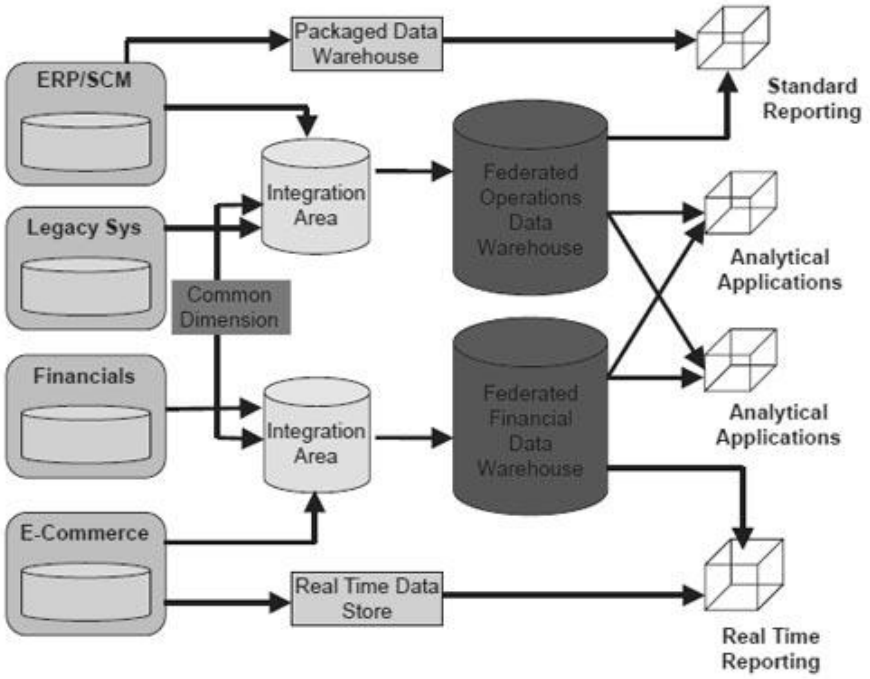


Figure 1.12: Security and privacy considerations in federated systems.

1.8 Applications of Federated Data Management

Federated data management enables organizations to achieve a unified view of their data without the extreme cost and risk of physically migrating petabytes of information to a centralized warehouse.

1.8.1 Healthcare Systems

Healthcare domains operate with disjointed systems: Electronic Health Records (EHR), laboratory information systems (LIS), radiology archives (PACS), pharmacy and billing platforms.

- **Health Information Exchange (HIE):** Federated architecture enables regional or national HIEs where patient data remains at the source hospital but can be queried by authorized practitioners across institutions.

- **Clinical Research:** Multicenter clinical trials query patient cohorts across hospitals without centralizing protected health information (PHI).

- **Epidemiological Surveillance:** During public health emergencies, federated queries rapidly aggregate case data from distributed hospital systems.

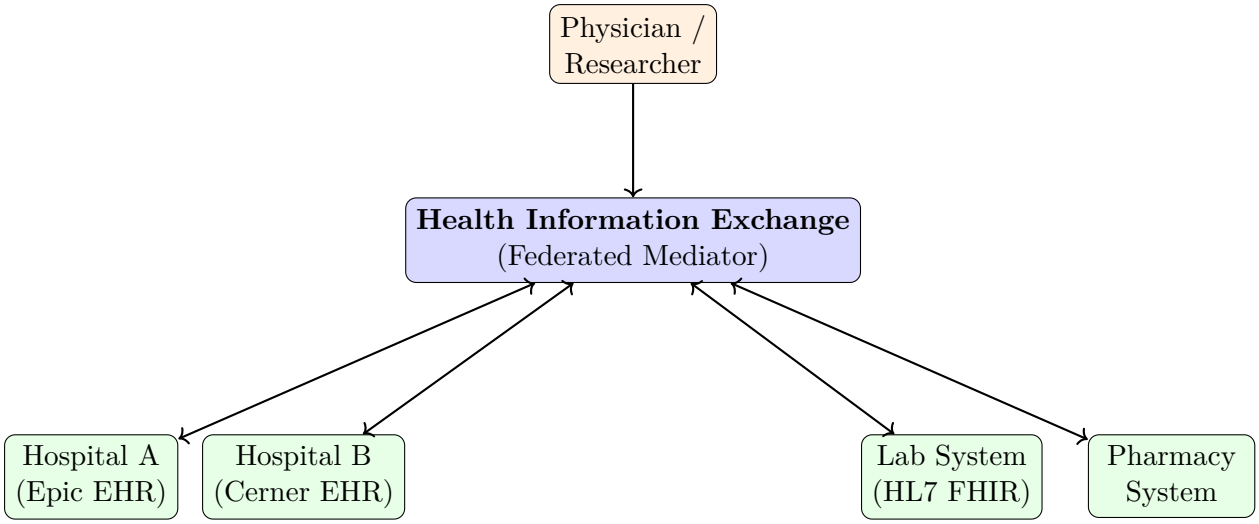


Figure 1.13: Federated data architecture for healthcare information exchange.

1.8.2 Financial and Enterprise Systems

Large financial institutions often have siloed data systems due to mergers, acquisitions, or distinct business units.

- **Enterprise Risk Management:** Federated queries across trade execution systems, derivatives platforms, and loan portfolios calculate global risk exposure in near real time.
- **Regulatory Reporting:** Compliance frameworks (Basel III, SOX) require cross-system data aggregation without the latency of daily batch ETL.
- **Customer 360:** Unifying customer data across CRM, transaction systems, and support platforms for a holistic view.

Table 1.7: Application Domains and Key Federated Use Cases

Domain	Example Use Case	Key Benefit
Healthcare	Cross-hospital patient record query	Data stays at source; HIPAA compliance
Finance	Real-time enterprise risk calculation	Avoids ETL latency; near real-time insight
Government	Cross-agency citizen services	Data sovereignty; inter-agency cooperation
Scientific Research	Multi-institution genomic analysis	Federate globally; protect sensitive data
E-Commerce	Unified product + logistics view	Multi-vendor integration without migration
Multi-Cloud	Cross-cloud analytical queries	Avoid cloud vendor lock-in

1.8.3 Case Study I: Federated Energy Management in Smart Cities

Modern smart cities integrate massive IoT sensor networks to manage energy distribution and consumption. However, centralized collection of high-frequency household energy data raises significant privacy concerns. *Dwarampudi and Yogi (2024)* [1] demonstrate a federated approach to energy management where individual smart meters perform local computation to optimize grid load.

In this use case, the federated mediator coordinates multiple residential micro-grids. Instead of uploading raw power usage patterns, each local controller (wrapper) processes the data in-situ. The result is an optimized global energy distribution model that improves grid reliability while ensuring that personal consumption habits remain strictly at the household level [2].

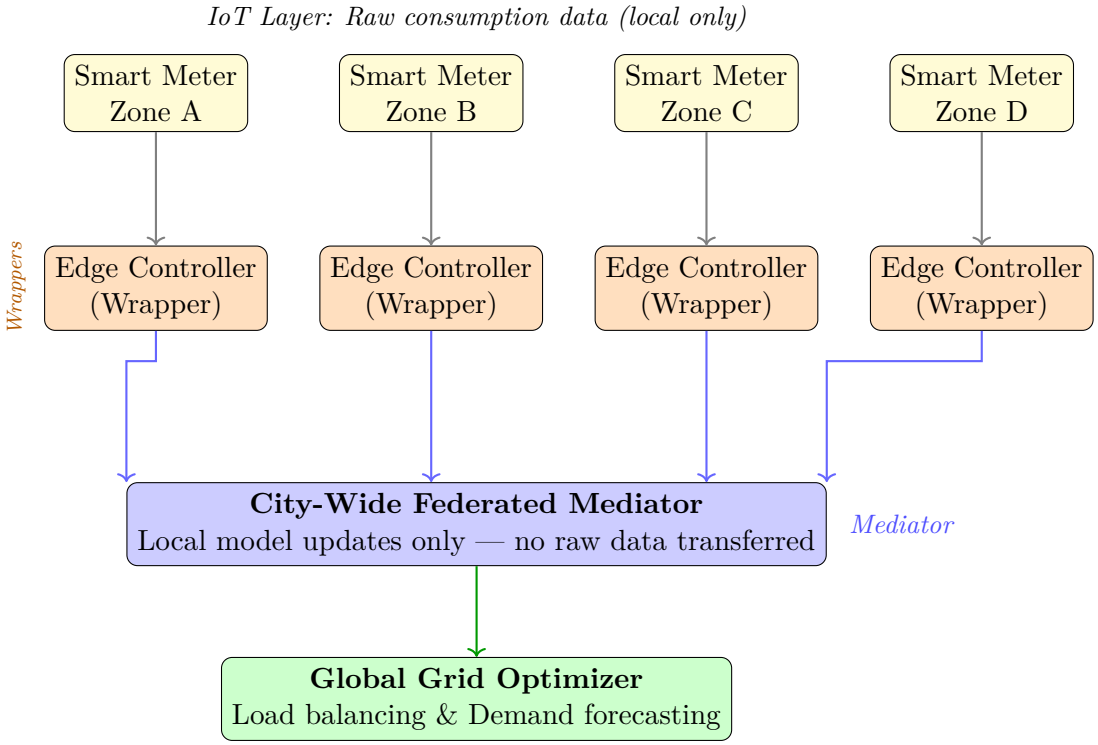


Figure 1.14: Federated energy management architecture for smart cities. Raw consumption data stays within each zone; only aggregated model updates flow to the central mediator.

1.8.4 Case Study II: Privacy-Preserving Medical Image Analysis

Collaborative research in oncology and neurology requires access to diverse, high-quality medical imaging datasets (MRI, CT scans) distributed across multiple hospitals. Data sovereignty laws and Patient Health Information (PHI) protections often prohibit the physical movement of these datasets to a central repository. *Guan et al. (2024)* [3] explore a federated data management framework that allows multiple diagnostic centers to collaboratively train deep learning models for early cancer detection. The mediator manages a virtual global schema of medical metadata, while the raw imaging data never

leaves the hospital's secure on-premises storage. This enables a "shared intelligence" model where hospitals benefit from a globally trained diagnostic tool without compromising patient confidentiality [4].

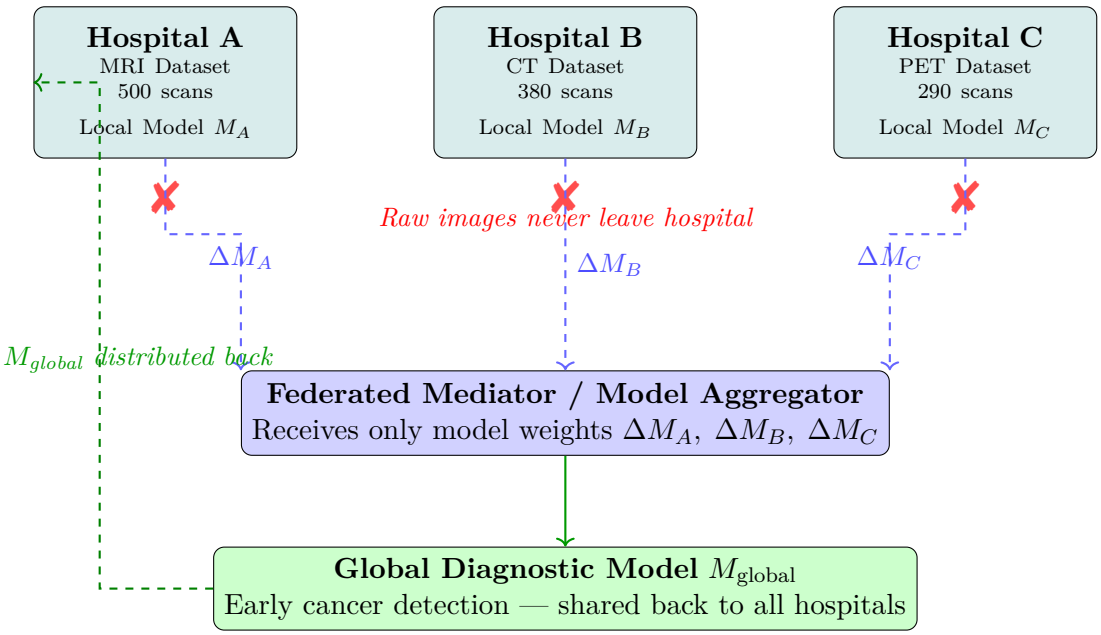


Figure 1.15: Privacy-preserving federated medical image analysis. Hospitals train local diagnostic models on their own data. Only model weight updates (ΔM) are sent to the mediator—raw patient images are never shared.

Table 1.8: Comparative Analysis of Federated Case Studies

Feature	Smart City Case	Healthcare Case
Data Type	High-frequency IoT streams	High-resolution medical images
Primary Constraint	Network latency & Power	Data sovereignty & Privacy
Privacy Strategy	On-device local computation	On-premises secure storage
Participation	Massive (thousands of sensors)	Moderate (multi-center hospitals)

★ Key Insight

Comparative Analysis of Case Studies. A comparative analysis of these two case studies reveals the versatility of federated data management across diverse domains. While the smart city application focuses on high-velocity IoT streams at the edge, the healthcare application addresses high-sensitivity medical records in institutional settings. Both cases, however, rely on the same fundamental property: the data remains at its source, and only the *"intelligence"* or *"models"* are federated globally. This dual-purpose utility makes FDM an essential architecture for future data-driven ecosystems.

1.9 Challenges and Future Directions

While federated data management effectively addresses data silos, the field continues to evolve to overcome fundamental limitations.

1.9.1 System Autonomy vs. Consistency

The classic tension is described by two foundational theorems:

- **CAP Theorem** (Brewer, 2000): A distributed system can provide at most two of Consistency, Availability, and Partition Tolerance simultaneously.
- **PACELC Theorem:** Even without partitions, there is a trade-off between Latency and Consistency.

Practical design implications:

- **Strong Consistency (CP):** For transactional workloads (e.g., financial trading). Requires 2PC; increases latency.
- **Eventual Consistency (AP):** For analytical workloads where slight staleness is acceptable.
- **Tunable Consistency:** Modern systems allow per-query consistency levels.

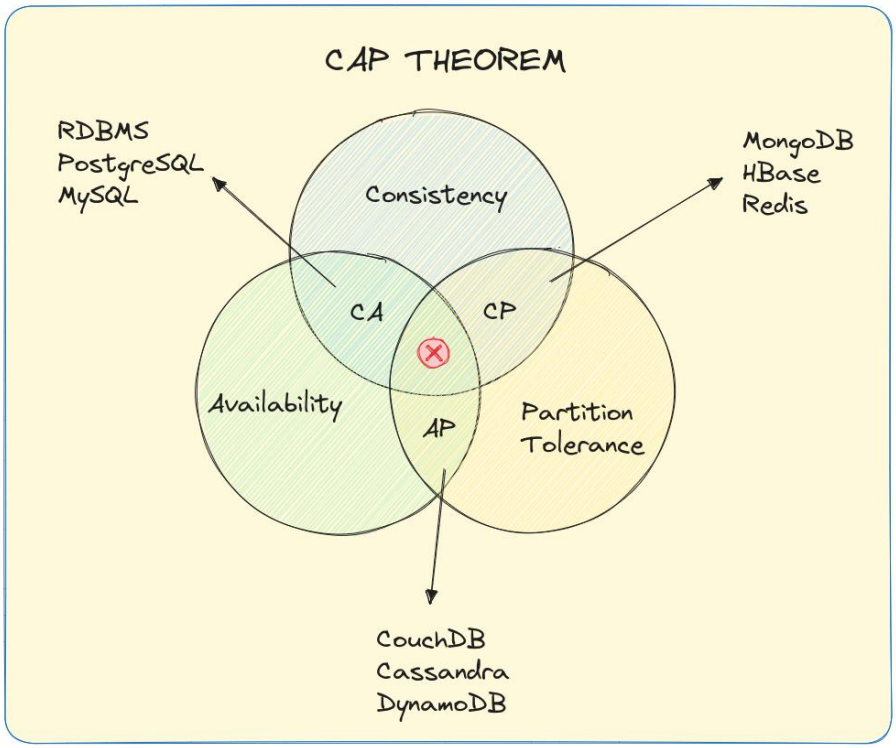


Figure 1.16: The CAP Theorem: a fundamental constraint on federated system design.

1.9.2 Emerging Research Trends

Table 1.9 summarizes the five most active research directions that are reshaping federated data management. These include AI-driven query optimization, the Data Mesh organizational paradigm [5], privacy-preserving federated learning [6], blockchain-based metadata management, and knowledge-graph-driven semantic integration.

Table 1.9: Emerging Trends and Their Impact on Federated Data Management

Trend	**Description**	**Expected Impact**
AI-Driven Optimization	ML models predict execution costs	2–5× query performance improvement
Data Mesh	Domain-oriented federated ownership	Organizational scalability
Federated Learning	In-situ model training	Privacy-preserving analytics
Blockchain Metadata	Immutable catalog on DLT	Trust in adversarial environments
Knowledge Graphs	Ontology-based semantic integration	Automated schema mapping

Summary

Federated Data Management provides a principled architectural approach to integrating heterogeneous, autonomous, and distributed data sources. Key takeaways from this chapter:

1. The **mediator–wrapper architecture** cleanly separates heterogeneity resolution (wrappers) from query integration (mediator).
2. **Schema heterogeneity**—naming, structural, and semantic conflicts—remains the most persistent challenge and requires both automated tools and expert curation.
3. **Query processing** involves decomposition, translation, optimization, and result integration, with cost models that account for network latency and source capabilities.
4. **Transaction management** must balance global consistency with local autonomy, using protocols such as 2PC or the Saga pattern.
5. **Security** demands layered enforcement: federated authentication, role/attribute-based authorization, encryption, and privacy-preserving techniques.
6. **Performance and scalability** are addressed through caching, pushdown optimization, parallel execution, and horizontally scalable mediator clusters.
7. Real-world applications span **healthcare**, **finance**, **scientific research**, and **multi-cloud environments**.
8. The field is advancing toward **AI-driven optimization**, **data mesh architectures**, **federated learning**, and **knowledge graph–based semantic integration**.

Bibliography

[1] A. Dwarampudi and M. K. Yogi, "Federated learning for energy management in next generation smart cities," *Journal of Communication Engineering & Systems*, vol. 14, no. 1, pp. 19–27, 2024.

[2] S. Pandya *et al.*, "Federated learning for smart cities: A comprehensive survey," *Sustainable Energy Technologies and Assessments*, vol. 55, p. 102987, 2023.

[3] H. Guan, P.-T. Yap, A. Bozoki, and M. Liu, "Federated learning for medical image analysis: A survey," *Pattern Recognition*, vol. 151, p. 110424, 2024.

[4] T. Q. Dang *et al.*, "A review of federated learning for smart healthcare," *IEEE Reviews in Biomedical Engineering*, vol. 15, pp. 286–301, 2022.

[5] Z. Dehghani, "How to move beyond a monolithic data lake to a distributed data mesh," 2019, martin Fowler Blog.

[6] P. Kairouz *et al.*, "Advances and open problems in federated learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1-2, pp. 1–210, 2021.

[7] O. Wahab *et al.*, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2273–2298, 2021.

[8] T. Li *et al.*, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[9] Q. Yang *et al.*, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[10] V. Mothukuri *et al.*, "A survey on security and privacy of federated learning," *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[11] D. C. Nguyen *et al.*, "Federated learning for smart city applications: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2396–2431, 2021.

[12] Z. Zheng *et al.*, "Applications of federated learning in smart cities: Recent advances, taxonomy, and open challenges," *Applied Sciences*, vol. 11, no. 18, p. 8407, 2021.

[13] A. Imteaj *et al.*, "A survey on federated learning in edge computing: Algorithms and applications," *IEEE Internet of Things Journal*, vol. 8, no. 22, pp. 16 327–16 344, 2021.

[14] M. Aledhari *et al.*, "Federated learning: A survey on enabling technologies, protocols, and applications," *IEEE Access*, vol. 8, pp. 140 699–140 725, 2020.

[15] F. Antunes *et al.*, "Federated learning for healthcare: Systematic review and future directions," *PeerJ Computer Science*, vol. 8, p. e831, 2022.

[16] Y. Y. Ghadi *et al.*, "Integration of federated learning with iot for smart cities applications, challenges, and solutions," *PeerJ Computer Science*, vol. 9, p. e1657, 2023.

[17] L. U. Khan *et al.*, "Federated learning for edge networks: Convergence analysis and resource management," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1758–1793, 2021.

[18] S. K. Lo *et al.*, "Architectural patterns for federated learning systems," *IEEE Software*, vol. 38, no. 4, pp. 31–37, 2021.

[19] M. J. Sheller *et al.*, "Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data," *Scientific Reports*, vol. 10, no. 1, pp. 1–12, 2020.

[20] C. Zhang *et al.*, "A survey on federated learning: The progress, challenges, and applications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1711–1738, 2021.

[21] L. Li *et al.*, "A survey on federated learning: The concept, application, and future direction," *Satellite Communications and Networking*, 2021.

[22] F. Sattler *et al.*, "Robust and communication-efficient federated learning from non-i.i.d. data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2019.

[23] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Computing Surveys*, vol. 22, no. 3, pp. 183–236, 1990.

[24] G. Wiederhold, "Mediators in the architecture of future information systems," *IEEE Computer*, vol. 25, no. 3, pp. 38–49, 1992.

[25] E. A. Brewer, "Towards robust distributed systems," in *ACM PODC Keynote*, 2000.