CSCI E-103: Reproducible Machine Learning
Lecture 05: Data Lakes, Lakehouses, and Delta Lake

Harvard Extension School

Fall 2025

■ **Course:** CSCI E-103: Reproducible Machine Learning

■ **Week:** Lecture 05

■ **Instructors:** Anindita Mahapatra & Eric Gieseke

■ **Objective:** Master the evolution from data silos to data lakehouses, understand Delta Lake's role in providing reliability, and learn job orchestration fundamentals

Contents

1 Lecture Overview

Key Summary

This lecture covers the evolution of data storage paradigms and introduces the Data Lakehouse architecture:

Core Topics:

- **Data Silos** – The problem of isolated, compartmentalized data
- **Data Lakes** – Centralized storage for all data types (Schema on Read)
- **Data Swamps** – The danger of ungoverned data lakes
- **Data Lakehouses** – Combining lake flexibility with warehouse reliability
- **Delta Lake** – The protocol that enables ACID transactions on data lakes
- **Medallion Architecture** – Bronze/Silver/Gold data organization
- **Job Orchestration** – Managing multi-task data pipelines
- **Data Mesh vs. Data Fabric** – Organizational vs. technical paradigms

The central theme of this lecture is understanding how to "hydrate" a data lake properly so it doesn't become a data swamp. This means not just pushing data in, but ensuring you can effectively pull data out with reliability and governance.

2 Key Terminology Reference

Before diving into the details, here is a comprehensive reference table of the key concepts covered in this lecture:

Term	Core Description (Analogy)	Key Characteristics	Relationship
Data Silo	Isolated "data islands" per department. (e.g., Finance's Excel, Marketing's CRM)	- Data isolation and duplication - Cross-functional analysis impossible	The **problem** Data Lake solves
Data Lake	Giant "lake" storing all data in raw format.	- All data types (structured/unstructured) - **Schema on Read** - Low storage cost	Solves silos. Can become a swamp.
Data Swamp	Ungoverned lake where nobody knows what data exists or where.	- No governance - Missing metadata - Lost data trust	**Failed** Data Lake
Data Warehouse	"Department store" for cleaned, structured data only.	- Structured data only - **Schema on Write** - BI/Reports, high performance	Complementary to Lake. (Analogy: Ferrari)
Data Lakehouse	**Hybrid** combining Lake flexibility with Warehouse reliability.	- Lake + Warehouse - Single platform for BI and AI	Evolution of Data Lake. Delta Lake enables this.
Delta Lake	"Management protocol" making Data Lakes reliable.	- Parquet-based + Transaction Log - **ACID, Time Travel** support	**Core technology** for Lakehouses
Medallion Arch.	3-tier data refinement pipeline: Bronze → Silver → Gold	- **Bronze**: Raw data - **Silver**: Cleaned, filtered - **Gold**: Aggregated, business-ready	**Methodology** for managing Lakehouse data
DAG	"Directed Acyclic Graph" – represents task dependencies.	- Direction: execution order - Acyclic: no circular dependencies	**Operating principle** of Spark and workflows
Data Mesh	**Organizational culture** where each business domain owns its data.	- Decentralized ownership - Data as a Product	**Organizational/culture** strategyapplicable with Lakehouses
Data Fabric	**Technical architecture** connecting data across environments.	- Hybrid/multi-cloud integration - Metadata-driven automation	**Technology/architecture** focused

3 Evolution of Data Storage: From Silos to Lakehouses

Data storage and utilization approaches have evolved in response to changing business requirements. Understanding this evolution helps explain why modern architectures like the Lakehouse exist.

3.1 Data Silos: Isolated Islands

Definition:

Data Silo A **Data Silo** is a state where data is isolated across different departments or systems within an organization, preventing cross-functional analysis and creating inefficiencies.

Example:

The Isolated Islands Analogy Consider Company A: The Finance team stores revenue data in their Excel files, while Marketing stores customer click data in a separate marketing platform. Because these two data sources aren't connected, answering the question "Which marketing activities actually drove sales?" becomes nearly impossible.
Each department is trapped on its own "data island" with no bridge to the others.

Why do Data Silos form?

- **Structural Issues:** Systems were designed without data sharing in mind
- **Political Issues:** Departments treat data as "property" and refuse to share
- **Growth Issues:** New systems introduced during expansion don't integrate with existing ones
- **Vendor Lock-in:** SaaS solutions make data export difficult

Caution

Data silos lead to:

- Duplicate data maintenance costs
- Inconsistent "versions of truth"
- Inability to get a holistic view of customers (the "Customer 360" problem)
- Lost business opportunities due to fragmented insights

3.2 Data Lakes: The Great Unifier

Definition:

Data Lake A **Data Lake** is a centralized repository that stores all types of data (structured, semi-structured, unstructured) in their original raw format at any scale.

The Data Lake emerged to solve the silo problem with a simple philosophy: "Store everything in one place, figure out how to use it later."

Key Characteristics:

- **All Data Types:** Handles structured (tables), semi-structured (JSON, XML), and unstructured

(images, videos, logs)

- **Raw Storage:** Data is stored as-is without transformation
- **Low Cost:** Built on cheap cloud storage (S3, ADLS, GCS)
- **Schema on Read:** No schema required at write time

Very Important:

Schema on Read vs. Schema on Write	**Schema on Read** (Data Lake)	**Schema on Write** (Data
	1. **Write:** Dump anything, no schema check 2. **Read:** Define/interpret schema at query time	1. **Write:** Strict schema 2. **Read:** Fast reads schema
	Pros: Fast ingestion, flexible **Cons:** Complex reads, quality issues	**Pros:** Data quality guara **Cons:** Rigid, can't store

3.3 Data Swamps: When Lakes Go Wrong

The flexibility of Data Lakes comes with a dangerous catch: without proper governance, they become **Data Swamps**.

Definition:

Data Swamp A **Data Swamp** is a Data Lake that has devolved into an unusable state due to lack of metadata management and governance. Nobody knows what data exists, where it is, or whether it can be trusted.

Example:

Warehouse vs. Junkyard Analogy **Well-governed Data Lake** = Amazon's fulfillment center

- Every item has a label and barcode
- Exact location is tracked in the system
- Can find anything in seconds

Data Swamp = Municipal junkyard

- Items dumped randomly
- No inventory or organization
- Something valuable might be there, but good luck finding it!

Caution

The Pendulum Metaphor

Think of data management as a pendulum swinging between two extremes:

$$\textbf{Data Silos} \longleftrightarrow \textbf{Data Swamps}$$

(Too rigid, can't use data) $\longleftrightarrow$ (Too loose, can't trust data)

The goal is to find the **middle ground**: a **Governed Data Lake** that provides flexibility while maintaining quality and discoverability.

3.4 Data Warehouses: The Structured Alternative

Definition:

Data Warehouse (DW) A **Data Warehouse** is a system designed for Business Intelligence (BI) and reporting that stores only cleaned, structured, and processed data with strict schema enforcement.

Key Characteristics:

- **Schema on Write:** Data must conform to predefined structure
- **Primary Users:** Business Analysts, SQL users
- **Strengths:** Reliable data, very fast queries
- **Weaknesses:** Expensive, can't handle unstructured data, limited for ML use cases

3.5 Data Lake vs. Data Warehouse Comparison

Example:

Ferrari vs. Tractor Trailer Analogy **Data Warehouse (Ferrari):**

- Incredibly fast and sleek (high-performance queries)
- But can only carry 2 passengers (structured data only)
- No cargo space (limited flexibility)

Data Lake (Tractor Trailer):

- Not quite as fast as the Ferrari
- But can haul any type of cargo (all data types)
- Unlimited capacity (petabyte scale)

Characteristic	Data Lake	Data Warehouse
Data Types	All (structured, semi, unstructured)	Structured only
Data State	Raw and processed	Processed only
Schema	Schema on Read	Schema on Write
Process	ELT (Extract, Load → Transform)	ETL (Extract, Transform → Load)
Primary Users	Data Scientists, ML Engineers	Business Analysts, SQL users
Main Use Cases	AI/ML modeling, data exploration	BI reports, dashboards
Cost Model	Low (storage/compute separated)	High (storage/compute coupled)
Data Format	Open formats (Parquet, Delta)	Proprietary formats

3.6 Data Lakehouse: Best of Both Worlds

Historically, organizations had to maintain both a Data Warehouse (for BI) and a Data Lake (for AI/ML). This created:

- Data duplication between systems

- DW/DL silos (ironic, given lakes were meant to eliminate silos!)
- Double infrastructure costs
- Data synchronization nightmares

Definition:

Data Lakehouse A **Data Lakehouse** is a unified platform that combines the **low cost and flexibility** of Data Lakes with the **reliability, governance, and performance** of Data Warehouses.

How is this achieved? Through technologies like **Delta Lake** that add data warehouse capabilities on top of data lake storage.

Key Information

Why Lakehouses Matter:

- Single platform for both BI and AI workloads
- No data duplication or sync issues
- Reduced infrastructure costs
- Open formats prevent vendor lock-in
- ACID transactions ensure reliability

4 Delta Lake: The Foundation of Reliable Lakehouses

4.1 What is Delta Lake? (A Protocol, Not a Format)

Very Important:

Common Misconception **Delta Lake is NOT a file format!**
Delta Lake is a **storage protocol** or **management layer**. The actual data files are still stored as efficient **Parquet** files.
Delta Lake adds a **_delta_log** folder containing transaction logs on top of Parquet files, enabling powerful capabilities that raw Parquet cannot provide.

Why was Delta Lake needed?

Traditional Data Lake files (Parquet, CSV in Hadoop/S3) were immutable and had critical limitations:

- **No Updates/Deletes:** Modifying a single row required rewriting entire files
- **No Transaction Safety:** Failed jobs could corrupt data (duplicates, partial writes)
- **No Concurrency:** Multiple readers/writers caused inconsistency
- **No Quality Guarantees:** No schema enforcement after initial write

Delta Lake solves all these problems, making data lakes as reliable as databases.

Key Information

Delta Lake Alternatives:
Delta Lake is not the only solution. Similar technologies include:

- **Apache Iceberg** – Created by Netflix, now widely adopted
- **Apache Hudi** – Created by Uber

All three provide similar ACID guarantees. Delta and Iceberg are working on interoperability.

4.2 Delta Lake Core Capabilities

1. **ACID Transactions:** Like traditional databases, Delta Lake guarantees:
   - **Atomicity:** Operations either complete fully or not at all
   - **Consistency:** Data remains valid before and after transactions
   - **Isolation:** Concurrent operations don't interfere with each other
   - **Durability:** Committed changes persist
2. **Schema Management:**
   - **Schema Enforcement:** Rejects data that doesn't match table schema
   - **Schema Evolution:** Allows intentional schema changes (adding columns)
3. **Time Travel:** Every change is logged in _delta_log, enabling queries like:

```
-- Query data as it was at version 5
SELECT * FROM my_table VERSION AS OF 5;

-- Query data as it was at a specific time
```

```
SELECT * FROM my_table TIMESTAMP AS OF '2025-10-26 03:00:00';
```

4. **DML Operations:** Direct SQL operations on lake files:

```
UPDATE my_table SET column = 'value' WHERE condition;
DELETE FROM my_table WHERE condition;
MERGE INTO target USING source ON condition
  WHEN MATCHED THEN UPDATE ...
  WHEN NOT MATCHED THEN INSERT ...;
```

5. **Performance Optimization:**
   - **Data Skipping:** Statistics-based file filtering
   - **Z-Ordering:** Multi-dimensional data clustering
   - **OPTIMIZE:** Compacts small files into larger ones

5 Data Architecture Principles

5.1 Medallion Architecture

The Medallion Architecture is the most widely used pattern for organizing data in a Lakehouse. It separates data into three layers based on quality and refinement level.

Example:

Ore Refinery Analogy Think of data processing like refining ore:
Bronze (Raw Ore): Fresh from the mine, unprocessed. Contains impurities but preserves everything.
Silver (Refined Metal): Impurities removed, standardized form. Ready for manufacturing.
Gold (Finished Product): Shaped into specific products for end consumers.

Layer	Bronze	Silver	Gold
Data State	Raw (as received)	Cleaned, filtered	Aggregated, business-ready
Key Operations	- Ingest all source data- Preserve original format- Add ingestion metadata	- Remove nulls/duplicates- Standardize types- Join multiple sources- Convert to Delta format	- Business aggregations- KPI calculations- ML feature tables- Dashboard-ready views
Data Structure	Same as source system	3NF or similar	Denormalized, Star Schema
Primary Users	Data Engineers	Data Engineers, Scientists	Business Analysts, Scientists
Update Frequency	Real-time or batch	Batch (Bronze → Silver)	Batch (Silver → Gold)

Key Information

Why Medallion Architecture Works:

- **Reprocessing:** If transformation logic changes, reprocess from Bronze
- **Debugging:** Compare Bronze vs. Silver to trace data issues
- **Multiple Uses:** One Silver table can feed multiple Gold tables
- **Clear Ownership:** Each layer has defined responsibilities

5.2 Six Guiding Principles for Lakehouses

1. **Treat Data as Products:** Don't just accumulate data—curate it into trusted, well-documented "data products" that internal customers can rely on.
2. **Eliminate Data Silos, Minimize Data Movement:** Keep data in one place (the Lake) and let systems access it directly. Every copy creates sync issues and stale data risks.
3. **Democratize Value Creation Through Self-Service:** Enable business users to access and analyze data themselves, under proper governance. Don't bottleneck everything through the data team.
4. **Adopt Organization-Wide Data Governance:** Implement centralized access control, quality management, and cataloging (e.g., Unity Catalog). Without governance, you get a swamp.
5. **Use Open Interfaces and Formats:** Avoid vendor lock-in by using open formats like Parquet and Delta Lake. This preserves flexibility and enables ecosystem tools.
6. **Build for Scale, Optimize for Performance and Cost:** Design for growth from the start. Balance performance needs against cost constraints—don't over-provision, but don't bottleneck either.

5.3 Data Mesh vs. Data Fabric

Two concepts frequently discussed alongside Lakehouses are Data Mesh and Data Fabric. Despite similar-sounding names, they address different concerns.

Very Important:

Core Difference: Organization vs. Technology **Data Mesh:** An **organizational/cultural** approach. "Decentralize data ownership to domain teams who understand their data best."
Data Fabric: A **technological/architectural** approach. "Create a unified layer that connects data across all environments (cloud, on-prem, hybrid)."

Dimension	Data Mesh	Data Fabric
Core Idea	Decentralization	Integration and connection
Primary Focus	Organization, people, process	Technology, architecture, automation
Data Ownership	Domain teams (business units)	Central IT team (or delegated)
Governance Model	Federated (local rules under global standards)	Centralized (embedded via metadata)
Data Treatment	Data = Product (discoverable, trustworthy)	Data = Accessible Asset
Best Fit	Large enterprises with complex org structure	Enterprises with hybrid/multi-cloud complexity
Analogy	Federation of cities, each managing its own infrastructure	Highway system connecting all cities

Four Pillars of Data Mesh:

1. **Domain Ownership:** Each business domain owns and manages its data
2. **Data as a Product:** Data must be discoverable, addressable, and trustworthy
3. **Self-Service Infrastructure:** Teams can create resources on demand
4. **Federated Computational Governance:** Global standards, local implementation

6 Data Pipeline Operations and Debugging

6.1 Data Consolidation Tools

Getting data into the Data Lake requires ingestion tools. Here are the primary methods in Databricks:

1. **AutoLoader:**
   - Purpose: Continuous, incremental ingestion
   - Monitors cloud storage folders for new files
   - Automatically detects and processes new arrivals
   - Uses `cloudFiles` format (streaming-based)
   - Scales to billions of files
2. **COPY INTO:**
   - Purpose: One-time bulk batch ingestion
   - SQL command for loading data
   - **Idempotent:** Running multiple times doesn't create duplicates
   - Simpler than AutoLoader for single-load scenarios
3. **Delta Live Tables (DLT):**
   - Next-generation declarative ETL
   - Define "what" you want, system handles "how"
   - Built-in data quality constraints
   - Automatic error handling and recovery
4. **Local File Upload:**
   - Upload small files directly via Databricks UI
   - Limit: 10 files, 2GB total
   - Supports: CSV, TSV, JSON, Avro, Parquet, TXT, XML

6.2 Job Orchestration

Complex data pipelines involve multiple interconnected tasks. Managing these is called **Job Orchestration**.

Definition:

Workflow/Job A **Workflow** (or Job) is a collection of tasks with defined dependencies, schedules, and failure handling. Tasks can be notebooks, SQL scripts, Python files, or other executables.

Key Orchestration Concepts:

- **Dependencies (DAG):** Tasks form a Directed Acyclic Graph:
  - **Parallel:** Independent tasks run simultaneously
  - **Sequential:** Dependent tasks wait for predecessors
- **Scheduling:**
  - Cron expressions for time-based triggers ("every day at 3 AM")

  - Event triggers ("when new file arrives")
  - Continuous mode (always running)
- **Failure Handling:**
  - Automatic retries for transient failures
  - Timeouts for hung jobs
  - Alert notifications to pipeline owners
- **Repair and Run:** If a 10-task pipeline fails at task 8, you can "repair" from task 8 onwards, reusing results from tasks 1-7. This saves time and compute costs.

6.3 The 4 S's of Job Performance Issues

When Spark jobs run slowly, investigate these four common causes:

Example:

Understanding Performance Issues Through Analogies

1. **Spill:**
   - **Symptom:** Memory (RAM) insufficient, data written to disk temporarily
   - **Analogy:** Cooking on a tiny counter—you have to put ingredients on the floor and keep picking them up
   - **Solution:** Use nodes with more memory, increase T-shirt size (serverless)
2. **Shuffle:**
   - **Symptom:** Large data exchange between nodes during sorts or joins
   - **Analogy:** Team project where everyone needs materials from each other, spending all time sending Slack messages
   - **Solution:** Optimize partitioning strategy to minimize cross-node movement
3. **Skew/Stragglers:**
   - **Symptom:** Data distributed unevenly; one node does most work while others idle
   - **Analogy:** 5-person team project where 1 person gets 80% of the work—everyone waits for that one person to finish
   - **Solution:** Change partition keys for even distribution (e.g., "country + city" instead of just "country")
4. **Small Files:**
   - **Symptom:** Millions of tiny files create more I/O overhead than actual processing
   - **Analogy:** Reading a book split into 1 million single-page files—opening/closing files takes longer than reading
   - **Solution:** Use OPTIMIZE command to compact small files into larger ones

6.4 Monitoring Approaches

Traditional Compute (Ganglia Metrics):

- Visual dashboards for CPU, memory, network usage

- Click on individual nodes to see detailed graphs
- Useful for diagnosing spill (high memory + disk I/O) or I/O-bound operations

Serverless:

- No Ganglia-level metrics available
- Uses T-shirt sizing (Small, Medium, Large, X-Large)
- Simplified management but less fine-grained tuning
- Check query history for statistics and performance data

7 Lab: Delta Lake Features

This section covers hands-on Delta Lake operations from the lab.

7.1 Converting Parquet to Delta

Converting existing Parquet data to Delta Lake format is straightforward:

```sql
-- Create a Delta table from existing Parquet data
CREATE TABLE IF NOT EXISTS loans_by_state_delta
USING delta
LOCATION '/path/to/delta/table'
AS SELECT * FROM parquet_table_view;

-- Verify the table format
DESCRIBE DETAIL loans_by_state_delta;
-- Result shows: format: delta
```

Listing 1: Creating a Delta table from Parquet data

7.2 Concurrent Streaming and Batch Operations

One of Delta Lake's powerful features is allowing simultaneous streaming reads and batch writes:

```python
# Start a streaming read from the Delta table
streaming_query = (
    spark.readStream
        .format("delta")
        .load("/path/to/delta/table")
        .writeStream
        .format("console")
        .start()
)
```

Listing 2: Reading Delta table as a stream

```sql
-- Insert batch data while streaming is running
INSERT INTO loans_by_state_delta VALUES ('IA', 450);
INSERT INTO loans_by_state_delta VALUES ('IA', 450);
-- ... repeat for more inserts
-- The streaming query will automatically pick up these changes
```

Listing 3: Batch inserts while streaming is active

7.3 DML Operations (DELETE, UPDATE)

Operations impossible on raw Parquet files work seamlessly on Delta tables:

```sql
-- Delete specific rows (impossible on plain Parquet!)
DELETE FROM loans_by_state_delta WHERE state = 'IA';
```

Listing 4: Row-level delete operation

7.4 MERGE (Upsert)

The MERGE command provides atomic "upsert" (update or insert) functionality:

```
MERGE INTO loans_by_state_delta AS target
USING new_updates_table AS source
ON target.state = source.state

WHEN MATCHED THEN
  -- State exists: update the count
  UPDATE SET target.count = source.new_count

WHEN NOT MATCHED THEN
  -- State doesn't exist: insert new row
  INSERT (state, count) VALUES (source.state, source.new_count);
```

Listing 5: MERGE operation for upserting data

7.5 Schema Evolution

Delta Lake enforces schema by default but allows intentional evolution:

```
# Without mergeSchema, this fails if 'amount' column is new
df.write \
    .format("delta") \
    .mode("append") \
    .option("mergeSchema", "true") \  # Allow new columns
    .save("/path/to/delta/table")
```

Listing 6: Enabling schema evolution during write

7.6 Time Travel

Every change is recorded, enabling historical queries:

```
-- View all changes to the table
DESCRIBE HISTORY loans_by_state_delta;
-- Shows: version, timestamp, operation, operationParameters

-- Query specific version
SELECT * FROM loans_by_state_delta VERSION AS OF 0;
-- Returns data as it was initially created (Iowa missing)

SELECT * FROM loans_by_state_delta VERSION AS OF 9;
-- Returns data after all modifications (Iowa = 10)

```

```
-- Query by timestamp
SELECT * FROM loans_by_state_delta
TIMESTAMP AS OF '2025-10-26 03:00:00';
```

Listing 7: Viewing table history and time travel

8 Lab: Databricks Workflows

8.1 Creating a Job

Steps to create a workflow:

1. Navigate to **Jobs & Pipelines** in Databricks
2. Click **Create Job**
3. Add tasks (notebooks, SQL, Python files, etc.)
4. Configure dependencies between tasks
5. Set scheduling (cron, file arrival, continuous)
6. Configure alerts and retry policies

8.2 Task Dependencies

- Tasks with **no dependencies** run in parallel
- Tasks with **dependencies** wait for predecessors to complete
- Removing a dependency makes tasks parallel
- The visual DAG editor shows the execution flow

8.3 Parameters

Workflows support two types of parameters:

Job Parameters: Global to the entire workflow

```
# Access job parameter in notebook
job_param = dbutils.widgets.get("job_param1")
```

Task Parameters: Specific to individual tasks

```
# Access task parameter in notebook
task_param = dbutils.widgets.get("param1")
```

8.4 Scheduling Options

- **Scheduled:** Use cron syntax (e.g., "0 9 * * 1" = every Monday at 9 AM)
- **File Arrival:** Trigger on new files in a location
- **Continuous:** Always running (expensive, use for real-time needs)

9 Industry Adoption and Maturity

9.1 Lakehouse in the Gartner Hype Cycle

According to Gartner's analysis:

2021: Lakehouses were on the "Innovation Trigger" upslope—gaining attention but not yet proven.

2025: Lakehouses have passed through the "Trough of Disillusionment" and are climbing the "Slope of Enlightenment." Expected to reach the "Plateau of Productivity" within 2 years.

Key Information

MIT Report Finding:
Approximately 70% of CIOs surveyed reported that their organizations have adopted Lakehouse architecture. This indicates strong enterprise acceptance of the paradigm.

9.2 Why Lakehouses Are Succeeding

Before Lakehouses, organizations maintained separate platforms for:

- Streaming (Kafka, Flink)
- Batch processing (Spark)
- Machine Learning (separate ML platforms)
- BI/Analytics (Data Warehouse)
- Data Storage (Data Lake)

Keeping 4-5 systems synchronized, reconciled, and operational was a nightmare.

The Lakehouse Promise:

"Data gravity is the most important thing. Once you've massaged, cleansed, curated, and governed your data in one place—that's it. All use cases should run from there because specialized engines can access it."

Key enabling technologies:

1. **Delta Lake:** ACID transactions, time travel, DML operations
2. **Unity Catalog:** Centralized governance and metadata
3. **Photon Engine:** Rewritten Spark for warehouse-level performance
4. **Serverless Compute:** Simplified infrastructure management

10 One-Page Summary

Evolution of Data Storage

1. **Data Silos (Problem):** Isolated data islands. Departments can't share or integrate data.

↓

2. **Data Lake (Solution):** All data in one place, raw format, Schema on Read.

↓

3. **Data Swamp (Risk):** Lake without governance becomes unusable mess.

↓

4. **Data Lakehouse (Evolution):** Lake flexibility + Warehouse reliability in one platform.

Delta Lake: The Foundation

What it is: A protocol (not format!) adding transaction logs to Parquet files.
Key Capabilities:

- **ACID Transactions:** Reliable reads/writes
- **Time Travel:** Query historical data
- **DML Support:** UPDATE, DELETE, MERGE on lake files
- **Schema Evolution:** Controlled schema changes

Medallion Architecture

Bronze → Silver → Gold

- **Bronze:** Raw data as ingested (preserve everything)
- **Silver:** Cleaned, validated, joined (the work happens here)
- **Gold:** Aggregated, business-ready (what users consume)

The 4 S's of Performance Issues

- **Spill:** Memory overflow → disk usage (slow I/O)
- **Shuffle:** Cross-node data exchange (network bottleneck)
- **Skew:** Uneven data distribution → straggler nodes
- **Small Files:** Too many tiny files → I/O overhead

Data Mesh vs. Data Fabric

Data Mesh: Organizational culture. Decentralize data ownership to domain teams.
Data Fabric: Technical architecture. Unified access layer across all environments.
Both can be built on top of a Lakehouse platform!