Data architectures evolution: warehouse, lake, lakehouse

by Nicola Orecchini

June 2025

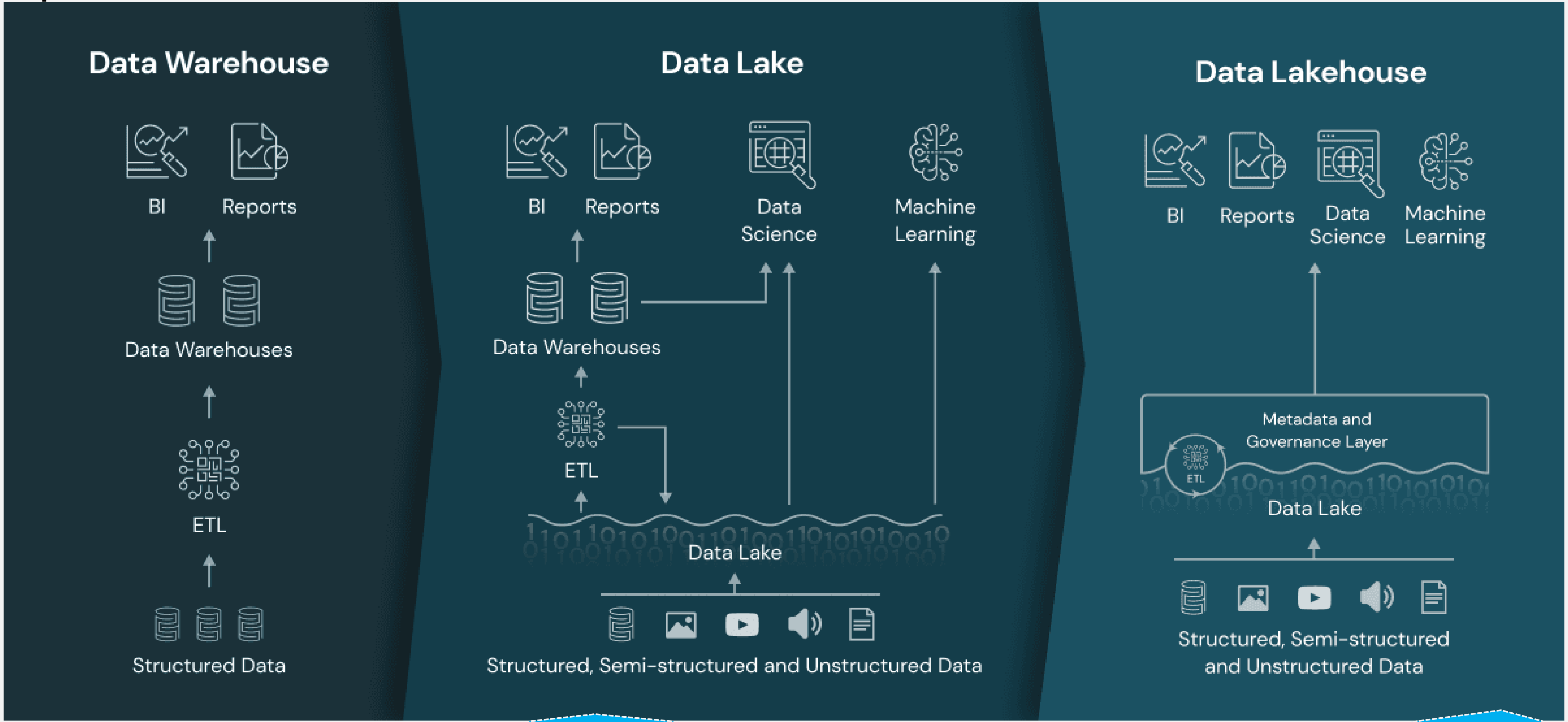
DWHs were not suited for handling big data. Data lakes emerged to handle raw data in a variety of formats on cheap storage, though lacked critical features

	Data Warehouse (DWH)	Data Lake
Data Model & Schema	Strict control over schemas (rigid structure), schema-on-write	Loose/no enforcement of schema (put files in the filesystem, maintain them primitive), schema-on-read
Transactionality	Fully transactional, ACID[2] guaranteed	No transaction support, no ACID[2], hard to update data inplace. Lack of CI makes it hard to mix appends and reads, and batch and streaming jobs
Data Quality	Enforced through schema and constraints	Not enforced, must be implemented manually
Scalability	Limited; vertical scaling costly	Horizontally scalable on cheap storage
Data types supported	Structured / tabular	Unstructured data, semi-structured data
Storage & Compute	Tightly coupled (expensive to scale)	Decoupled (scale storage and compute independently)
Cost	Expensive due to compute–storage coupling, proprietary licenses, and constant resource provisioning	Lower storage costs, but costly to keep data organized, governed and functional
File formats	Row-based (e.g., CSV)	Columnar (e.g., parquet[1], ORC)
Processing paradigm	ETL: Transform data before loading (upfront modelling, tight control)	ELT: Load raw/semi-structured data first, transform later when needed (flexible, schema-on-read); ELT also supported

1. More on parquet files: https://data-mozart.com/parquet-file-format-everything-you-need-to-know/, https://data-mozart.com/inside-vertipaq-compress-for-success/; 2. Atomicity, Consistency, Isolation, Duration; details and examples in Annex

Source: *Table Formats: the Rise of Duck Lake with Antonio Murgia*, AgileLab Youtube Channel; Databricks, *What is a Data Lakehouse?*

Solution: Data Lakehouse, an open data mgmt. architecture that combines flexibility, cost-efficiency, and scale of data lakes with data mgmt. and ACID properties of DWHs



To exploit benefits from both architectures, data teams stitched DWH and DL together, creating a so-called Two-Tier Data Architecture, but resulting in duplicate data, extra infrastructure cost, security challenges, and significant operational costs. In a two-tier data architecture, data is ETLd from the operational databases into a data lake

Lakehouse architecture reduces the complexity, cost and operational overhead of Two-Tier Data A. by providing many of the reliability and performance benefits of the data warehouse tier directly on top of the DL, ultimately eliminating the DWH tier

Sources: What is a Data Lakehouse? | Databricks; *Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics*

Data Lakehouse has been enabled by 3 key technology advancements…

Metadata layers for data lakes

Storage layers that sit on top of open file formats (e.g. Parquet files) and **track** which files are part of **different table versions** to offer rich data management features like **ACID-compliant transactions**

New query engines

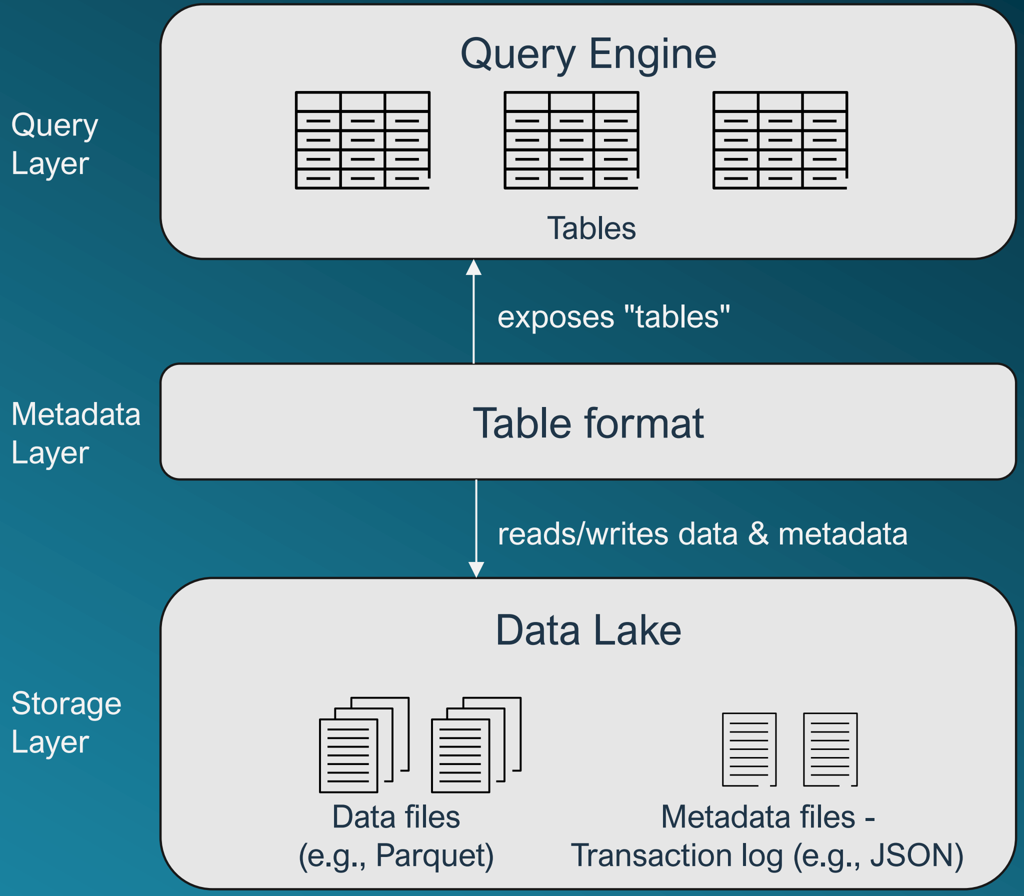
New query engine designs providing **high-performance SQL execution** on data lakes, through optimizations like:

- **caching** hot data in RAM/SSDs (possibly transcoded into more efficient formats)
- data layout optimizations **to cluster co-accessed data**
- auxiliary data structures like **statistics** and **indexes**, and **vectorized execution** on modern CPUs

Integration with other ML tools

Optimized access for **data science** and **machine learning** tools: thanks to open data formats used by data lakehouses, **data scientists and machine learning engineers** can **easily access the data in the lakehouse** and use tools popular in the DS/ML ecosystem *e.g., pandas, TensorFlow, etc.) that can already access sources like Parquet and ORC

…building layers on top of the existing Data Lake



1. For fore info on parquet files: https://data-mozart.com/parquet-file-format-everything-you-need-to-know/, https://data-mozart.com/inside-vertipaq-compress-for-success/
Source: *Table Formats: the Rise of Duck Lake with Antonio Murgia*, AgileLab Youtube Channel; Databricks, *What is a Data Lakehouse?*

If the data lake was a lake, Metadata Layers would be the **smart buoy system** that **makes fishing reliable**

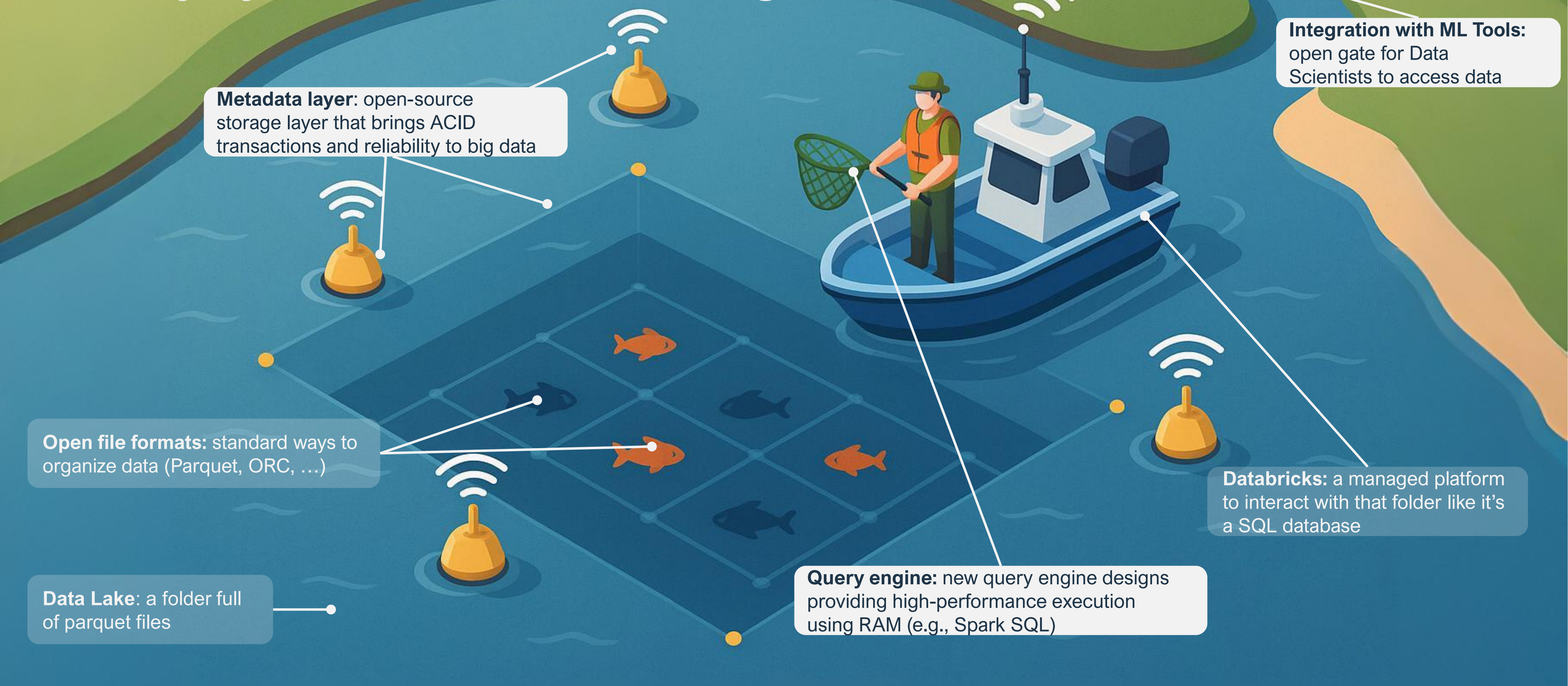
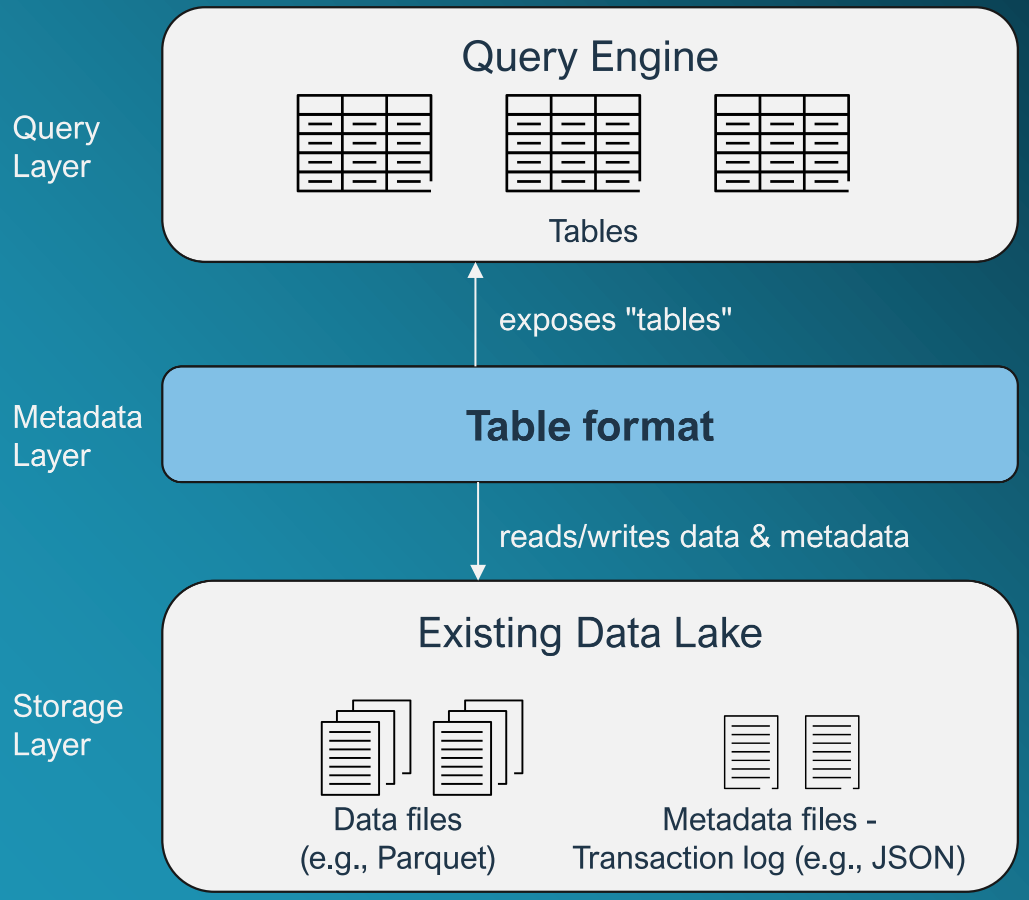


Image source: Chat GPT

Table formats (a.k.a. metadata layers/storage frameworks) bring properties of a Data Warehouse to a Data Lake



Note: For fore info on parquet files: https://data-mozart.com/parquet-file-format-everything-you-need-to-know/, https://data-mozart.com/inside-vertipaq-compress-for-success/
Source: *Table Formats: the Rise of Duck Lake with Antonio Murgia*, AgileLab Youtube Channel; Databricks, *What is a Data Lakehouse?*

Table formats are a **set of rules** and **metadata** that define how to treat a bunch of **data files** as a **coherent table**

They're not a processing engine or a file format by themselves: data files remain in formats like Parquet, ORC, or Avro. Table format is the software layer that **tracks those files** and **presents them as a single table**, with SQL semantics, to any execution engine

They sit **on top of Data Lake technologies**, on top of open file formats (e.g., Parquet)

They enable a **higher level of abstraction** (schema evolution, consistency, transactions) **maintaining scalability** of the underlying data lake

They achieve this by storing **metadata** (schemas, logs, versions) **alongside data,** typically in the same object storage. This enables features that were missing in basic Data Lakes but are standard in Data Warehouses:

- **ACID transactionality**
- **Time travel** to older versions
- **Schema** enforcement, evolution, and validation

Metadata files contain ordered records of **every transaction performed on the table**, including info about:

- **Operations performed** (e.g., insert, update, delete, merge) and filters/predicates used (e.g., WHERE)
- **Data files affected** (added/removed), including file paths, row counts, sizes, partition values
- **Schema version** at the time of operation
- **Timestamps** and commit info

Metadata Layers | There are many Table Formats in the market

	Widely adopted			New (specialized)	
	DELTA LAKE	ICEBERG	Apache hudi	ApachePaimon	DuckLake
Where is metadata stored?	• JSON log + Parquet checkpoint in object storage • Delta Lake Catalogs	• JSON & Avro metadata + manifest lists • Iceberg Catalog	• Object storage file-based timeline • metadata table	• File-based changelogs and snapshots • Hive/Flink metastores	• Relational DB (e.g., DuckDB or Postgres) • no separate catalog needed
How is metadata updated?	Each update involves an HTTP request via REST API: download file, modify locally, and write a new copy. With lots of transactions, latency of metadata operations degrades				SQL updates to relational db; faster but needs point-to-point integration
Use it for:	• Using Spark/Databricks, or need notebook/MLflow integration • Need strong schema evolution, rollback, branching		Streaming ingestion, frequent upserts	Data capture use cases, i.e., replicate an operational db and perform mainly updates by key	Many clients writing small amounts of data frequently
Avoid when:	Have lots of small files frequently updated				If metadata scales too much, problems of relational database (not horizontally scalable)

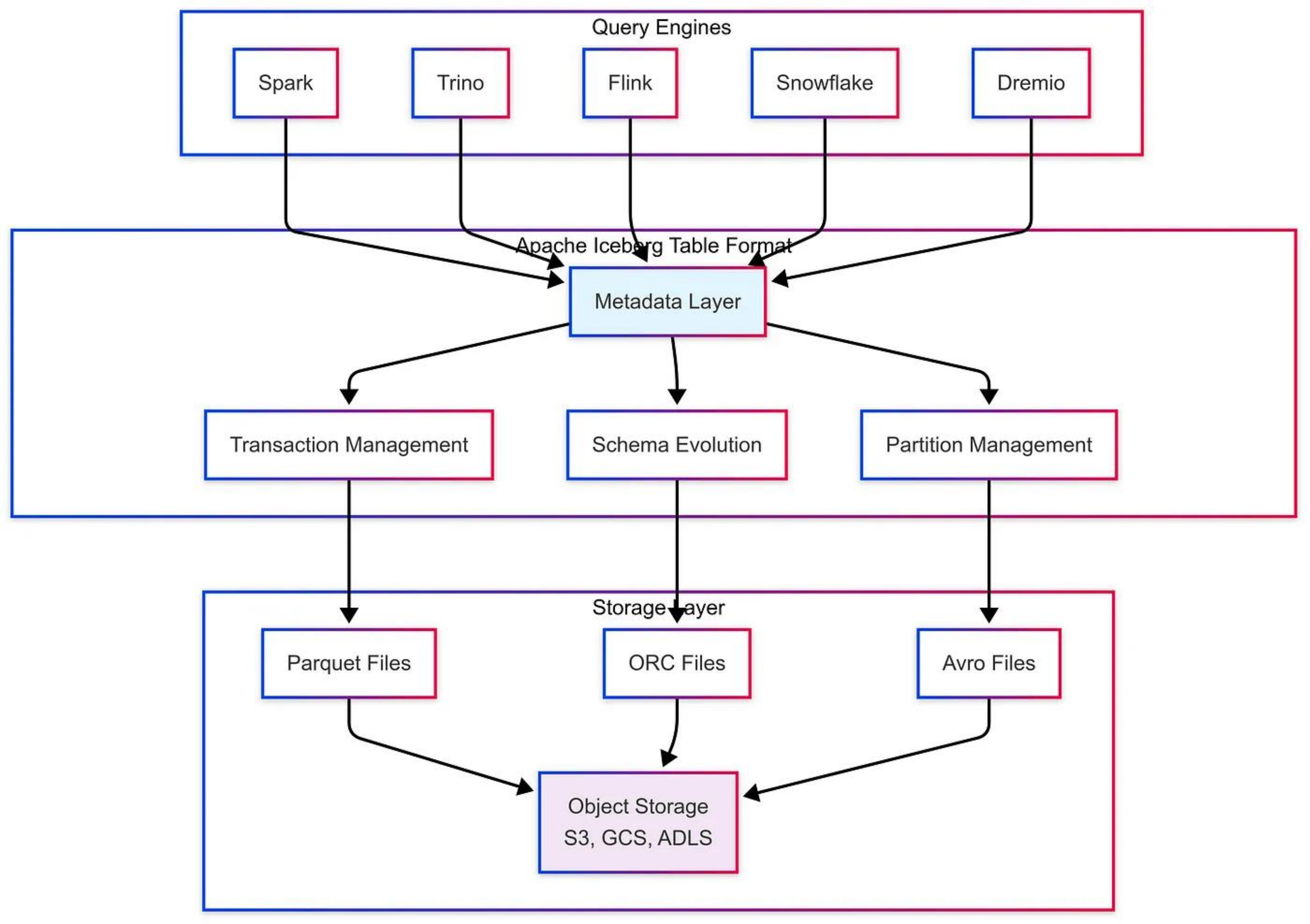
• To achieve transactionality of updates (i.e., if 2 clients try to write in the same table conflicting data, 1 will take over the other and the other will fail), a catalogue (backed by a database) is also used in the mix

1. For fore info on parquet files: https://data-mozart.com/parquet-file-format-everything-you-need-to-know/, https://data-mozart.com/inside-vertipaq-compress-for-success/

Source: *Table Formats: the Rise of Duck Lake with Antonio Murgia*, AgileLab Youtube Channel; Databricks, *What is a Data Lakehouse?*

Metadata Layers | Iceberg Architecture

ICEBERG



Source: *How Apache Iceberg Actually Works,* Thomas F McGeehan V, *https://medium.com/@tfmv/how-apache-iceberg-actually-works-64f97fb13c45*

Overview of most popular query engines

	Apache Spark	Apache Flink	Trino/Presto	Hive	Impala
Best For	Batch & stream ETL, ML, SQL	Low-latency stream processing	Interactive SQL on lake data	Legacy Hadoop-based batch	Fast interactive SQL on Hadoop
Strengths	Large community, rich ecosystem, ML support	True streaming (event-time), exactly-once	Fast federated querying, lightweight	Mature, integrates with HDFS, wide ecosystem	Low latency SQL, optimized for HDFS
Weaknesses	High memory needs, JVM overhead	Complex setup, mostly JVM-centric	Limited data mutation support	Slower, older architecture	Stronger ties to Cloudera stack
Typical Use Case	ETL pipelines, ML workloads, Delta Lake	Realtime analytics with Hudi or Paimon	Ad-hoc SQL queries on S3/Parquet	Long-running batch jobs, historical systems	OLAP-style workloads on Hive tables

Nicola Orecchini

nicola.orecchini@gmail.com