1. **Comparison of database architectures: data warehouse, data lake and data LAKEHOUSE**



By [Gonzalo ETSE](https://www.adaltas.com/en/author/gonzalo/)

May 17, 2022

Categories

[Big Data](https://www.adaltas.com/en/skills/big-data/)

[Data Engineering](https://www.adaltas.com/en/skills/data-engineering/)

Tags

[Data Governance](https://www.adaltas.com/en/skills/governance/)

[Infrastructure](https://www.adaltas.com/en/skills/infrastructure/)

[Iceberg](https://www.adaltas.com/en/tag/apache-iceberg/)

[Parquet](https://www.adaltas.com/en/tag/apache-parquet/)

[Spark](https://www.adaltas.com/en/tag/apache-spark/)

[Data Lake](https://www.adaltas.com/en/tag/data-lake/)

[Data LAKEHOUSE](https://www.adaltas.com/en/tag/data-lakehouse/)

[Data Warehouse](https://www.adaltas.com/en/tag/data-warehouse/)

[File Format](https://www.adaltas.com/en/tag/file-format/)

[more]

Do you like our work...[...we hire!](https://www.adaltas.com/en/careers/)

Top of Form

Email address

Bottom of Form

*Never miss our publications about Open Source, big data and distributed systems, low frequency of one email every two months.*

Database architectures have experienced constant innovation, evolving with the appearence of new use cases, technical constraints, and requirements. From the three database structures we are comparing, the first one to appear was the [Data Warehouses](https://www.adaltas.com/en/tag/data-warehouse/), introduced in the 80’s with the support of [Online Analytical Processing](https://en.wikipedia.org/wiki/Online_analytical_processing) (OLAP) systems, helping organizations face the rise of diverse applications in the 90’s by centralizing and supporting historical data to gain competitive business analytics. Later in the early 2000s [Data Lakes](https://www.adaltas.com/en/tag/data-lake/) appeared, thanks to innovations in cloud computing and storage, enabling to save an exorbitant amounts of data in different formats for future analysis.

To this day both solutions remain popular depending on different business needs. For instance, data warehouses allow for high-performance Business Analytics and fine grained [data governance](https://www.adaltas.com/en/skills/governance/). However, they lack on affordable scalability for petabytes of data. On the other side of the pitch, data lakes enable high throughput and low latency, but they have issues with data governance leading to unmanageable “data swamps”. In addition, data is considered immutable, which leads to additional integration efforts.

This is why we can find modern data lake and data warehouse ecosystems converging, both getting inspiration, borrowing concepts, and addressing use cases from each other. In this landscape we find a new architecture emerge: the **Data LAKEHOUSE**, which tries to combine the key benefits of both competing architectures, offering low-cost storage accessible by multiple data processing engines such as [Apache Spark](https://www.adaltas.com/en/tag/apache-spark/), raw access to the data, data manipulation, and extra flexibility. Let’s review each one of them in detail.

1. **Methodology of Comparison**

There are multiple indicators to consider when selecting a database architecture. To be more comprehensive, we pre-selected a set of common concerns.

* **Accessibility**  
  Ability to democratize data by allowing both technical and non-technical users to access crucial data if needed.
* **Lineage**  
  Knowledge of the data origin, how it’s modified and where it moves over time, enabling to trace errors to it’s root cause.
* **Data Types**  
  Supported data types.
* **Ingestion**  
  Sources and methods for data ingestion.
* **Governance & Security**  
  Ability to set procedures which ensure critical data is formally and securily managed through the company.
* **Scalability**  
  Ability to maintain availability and behavior when more resources are demanded.
* **Upserts & purging**  
  Ability to update and delete obsolete data.
* **Performance**  
  Efficiency to handle multiple queries concurrently, both in term of throughput and latency.
* **Reliability**  
  Data consistency and accuracy, we will also account for availability.
* **Applications**  
  Which usecases the architecture allows for.

1. **What is a data warehouse?**

A data warehouse is a centralized system designed to store present and historical data. Their focus is to provide readily available data for advanced querying and analysis. The main motivation for the emergence of data warehouses relied on solving the inconsistent data of RDBMS by transforming data from operational systems to analytical-processing support systems. The data warehouse is considered as the primary source of truth on business operations.

A diagram of data storage

Description automatically generated

A standard data warehouse architecture (image above) uses Extract, Transform and Load (ETL) for data transit through three different layers, data staging, data core, and data access. The first layer sees raw data format transition to a fully transformed set. Then data schema-on-write is used to optimize the data model for downstream BI consumption. Finally, the access layer lets users retrieve the translated and organized data through SQL queries.

Data warehouse powers reports, dashboards, and analysis tools by storing data efficiently. They minimize input and output (I/O), allowing query results to be delivered faster and to multiple users simultaneously. Additionally, subsets of the data warehouse, called data marts, can be provided to answer specialized analytical needs.

Let’s review data warehouses by our selected indicators:

* **Accessibility**  
  Allows end users to easily access all the processed data on a centralized database through SQL. Data warehouse often provides data analysis and visualization tools with multiple features to aid non-technical users access data easily.
* **Lineage**  
  Only access to processed data. Upstreaming to their raw origin is difficult as it’s not accesible to end users.
* **Data Types**  
  Support of structured data, with limited support of semi-structured data.
* **Ingestion**  
  Require ETL pipelines to transform the data before its use. Data warehouses use both the data staging and core layers for batch processing. The staging layer stores raw data from different sources, such as transactional systems, relational databases, and other sources. The core layer integrates the data by translating, normalizing, historicizing, and moving it to an operational data store. This data then moves to the data warehouse, organizing it into hierarchical groups called *dimensions*, *facts*, and *aggregate facts*. The process schema-on-write is used, optimizing the data for fast query performance. For streaming data, an alternative solution is their support of *micro-batching* to collect data in small increments.
* **Governance & Security**  
  Fine-grained security and governance at row/columnar level for tables.
* **Scalability**  
  Data warehouses have tightly coupled compute and storage, making vertical scalability very expensive. Plus, on-premise data warehouses requires the client to pay for the peak load of users and data under management, since they must purchase, deploy, and maintain all hardware and software. It is important to note that new cloud-based virtualized warehouses allow for multi-cluster, decoupled compute and storage, and better scaling options.
* **Upserts & purging**  
  Data warehouses implement a Slowly Changing Dimension (SDC) policy. Updates only occur with SCD1, otherwise a new row is inserted along with tracking data. Purging is done by manipulating the partition of the database. The valid data is migrated to a new partition and then the database switch to the new partition. Purging is recommended to reduce the cost of storage and improve performance, unless previous data versions are required for analysis.
* **Performance**  
  Data warehouses accept SQL and are able to optimize everything under the hood, including proprietary storage formats.
* **Reliability**  
  High quality, reliable data with ACID transactions. Data warehouses are built to work with SQL analytics and BI, with schemas, indexes, and caching capabilities. On top of that, they allow for time-traveling and versioning.
* **Applications**  
  BI and SQL applications readily for use with the support of historical analysis and automated reporting to inform decision-making across an organization’s lines of business.

As a result, data warehouses excel at providing quality and consistency of the data; they allow for performant data analytics and BI thanks to their ACID compliance.

The limitations of data warehouses start when there is the need to use varied types of data like textual, streaming IoT or multimedia. In addition, the rise of machine learning and AI requires iterative computations that need direct raw data access.

1. **What is a data lake?**

Data lakes first appeared thanks to the inception of Hadoop which allows for multiple machines to work as one, allowing compute and storage to scale horizontally. Additionally, Hadoop allows data to be ‘thrown in’ without worrying about its structure. This works thanks to a schema-on-read process, where data is ingested in its raw format and only transformed when needed for analysis, enabling for semi-structured and unstructured data support. Lastly, data lake architecture relies on a metadata catalog (e.g., Hive Metastore, Apache Glue). Without the metadata catalog, data lakes quickly become unmanageable and drift into data swamps.

Later on, cloud services, AWS being the first one, introduced capabilities to decouple compute and storage, which mean they could scale independently. Moreover, many upgrades were possible thanks to Hadoop ecosystem being open-source. This include big data frameworks (e.g., Hortonworks, Cloudera, mapR) which aid on Hadoop components handling, as well as the addition of new open-source tools like Apache Spark, which started to be used as the processing engine, enabling performant ETL to transform raw data into refined and structured data for different uses cases.

Nevertheless, data lakes have difficulties providing data management, guaranteeing quality, security, and data governance. On top of that, if you need high performant SQL queries on petabytes of data and return complex analytical results fast, or the use of BI tools and features such as schema enforcement data and versioning, then data lakes are not enough. That’s why the current use of data lakes has evolved to a two-tier architecture alongside data warehouses.

A diagram of data processing

Description automatically generated

In this two-tier architecture, structured data is traditionally stored in raw format in the data lake, but is later on processed and stored in tabular format in the data warehouses, as we can see in the image above. The data stored in data warehouses are used for data analytics and Business Intelligence, while semi-structured and unstructured data is stored in the data lake and used for Data Science and Machine Learning.

Let’s review the two-tier architecture with the preselected indicators:

* **Accessibility**  
  BI analysts have limited access to data warehouses in the two-tier architectures, relying on data engineers to structure data into the warehouse. Lastly, data lakes often become data swamps when there is incorrectly cataloged metadata, making that data unmanageable.
* **Lineage**  
  Raw data is accessible through data lakes, but often BI analysts will only have access to the data warehouse where engineers load selected data.
* **Data Types**  
  Structured, semi-structured and unstructured support.
* **Ingestion**  
  Data lakes perform ELT (Extract, Load, Transform), meaning they don’t transform data before loading it, as they do not implement a schema for the data to fit. Instead, the transformation and schema are verified when a query is required. This is the process previously mentioned as a schema-on-read approach.
* **Governance & Security**  
  All the data is stored and managed as files. This doesn’t provide fine-grained access control on the contents of files, but only coarse-grained access control.
* **Upserts & purging**  
  Any data manipulation language (DML) operation on a data lake results in a file modification. A new file is created, and additional metadata operations must be made to keep track of the changes. Managing and deleting file versions is a complex task in a data lake.
* **Scalability**  
  Thanks to decoupled compute and storage, data lakes can scale independently. Cloud-based data lakes offer both storage and computing, making them quite scalable in comparison to Data Warehouses. Additionally, distributed file systems allow for increased storage capacity. On the negative side, due to the nature of the two-tier architecture, the scalability issues of data warehouses arise. On top of that, data is constantly transformed and processed towards data warehouses, which adds further costs and duplication of the data in both data lakes and data warehouses.
* **Performance**  
  Data lakes can’t match the performances of data warehouses even with engines like Spark. Additionally, the two-tier architecture is highly complex for users because data goes first into data lakes and then into data warehouses, creating complexities, delays, and new failure modes. This brings considerable performance issues when compared to plain data warehouses.
* **Reliability**  
  Since data lakes lack fine-grained governance and ACID compliance, there are potential issues with the consistency of the data. This is especially an issue when there are multiple readers and writers. There’s also the complexity of schema mismatch due to the lack of schema enforcement in an environment that relies on batch and streaming data from multiple heterogeneous sources. Additionally, in the case of two-tier architecture, continuous engineering is required to ETL data between data warehouses and data lakes. Each transformation incurs risks of failures that reduce data quality. This also results in stale data in the data warehouse, as transformations from data lakes may take days to load.
* **Applications**  
  ML & DS applications, thanks to open data format (such as parquet and orc) and engines like Spark, are directly accessible to a wide range of other analytics engines, such as machine learning systems. For BI and high performant SQL applications, it is required to do ETL pipelines towards a Data warehouse.

As a result, data lakes bring efficient capabilities to the openness of the data and the cost of data storage. Additionally, they are suitable for machine learning and artificial intelligence algorithms, thanks to their support of various processing frameworks (allowing the use of python libraries) and give access to large amounts of raw data.

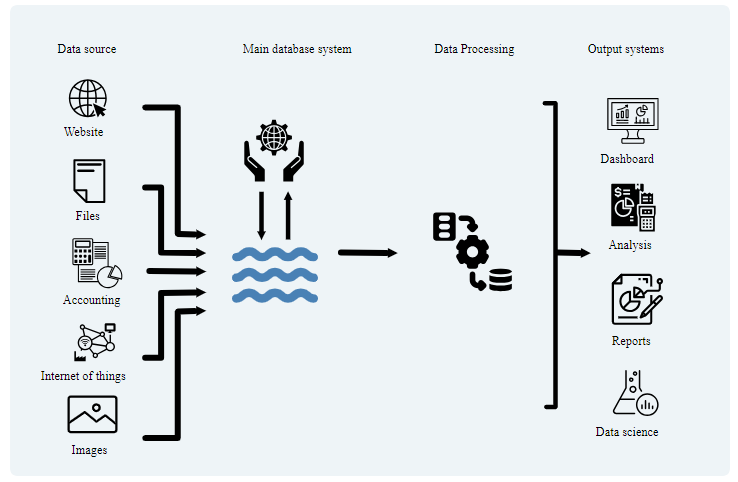
On the downside, the two-tier architecture results in complex ETL pipelines due to the significant movement, processing, and duplication of data towards data warehouses. Operationalizing and governance of this data architecture also become a challenge due to cost and complexity. All these results in data swamps and stale data.

1. **What is a data LAKEHOUSE?**

In 2019 Databricks released the paper [Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores](https://databricks.com/wp-content/uploads/2020/08/p975-armbrust.pdf) introducing the concept of data LAKEHOUSE and [Delta Tables](https://github.com/delta-io/delta). They intended to add a storage layer alongside Apache Spark, allowing for transactions and enforcing schema-on-write in an object-store. On the side, Netflix and Uber had released similar capabilities through [Apache Iceberg](https://www.adaltas.com/en/tag/apache-iceberg/) and [Apache Hudi](https://hudi.apache.org/), respectively. This would make the use of data warehouses redundant on top of a data lake.

In its architecture, a data LAKEHOUSE aims to provide data governance capabilities to a data lake while also reducing the operational costs of the aforementioned two-tier architecture. To achieve this, two features become vital. The first is the use of open [file formats](https://www.adaltas.com/en/tag/file-format/), such as [Parquet](https://www.adaltas.com/en/tag/apache-parquet/) and [ORC](https://www.adaltas.com/en/tag/apache-orc/), to facilitate essential statistics and allow for predefined data schemas. The second is the low-cost data storage system of a data lake, as decoupled compute and storage will allow for multiple processing engines.

But this doesn’t provide warehouse capabilities such as in-depth data management, versioning, or schema enforcement. ACID transactions are required, which were previously impossible in a distributed system where everything is on object storage.



LAKEHOUSE architecture (image above) embraces this ACID paradigm by leveraging a metadata layer (e.g., Hive metastore, HDFS) and, more precisely, a storage abstraction framework (Apache Iceberg, Apache Hudi, Delta Tables). These open table formats will allow the metadata layer to register changes as transactions while handling concurrency.

Let’s dive into each point of comparison:

* **Accessibility**  
  Allows for direct access to files with SQL, R, Python, Spark, and other languages. With both Iceberg and Delta based LAKEHOUSEs you can easily query all types of data through different platforms, or work on top of the raw data through a notebook.
* **Lineage**  
  The architecture allows for an open API to give direct access to raw data without the need for proprietary engines and vendor lock-in.
* **Data Types**  
  Structured, semi-structured and unstructured support.
* **Ingestion**  
  Ingestion is handled similarly to a data lake through ELT with the added complexity of updating the metadata layer through Spark or Hive. On the positive, data LAKEHOUSE allows for unifying batches and streaming data processing capabilities. For example, Delta Lake, alongside Structured Streaming, makes it possible to analyze streaming and historical data, or different data sources together at high speeds.
* **Governance & Security**  
  LAKEHOUSE ACID properties allow for schema enforcements, audits, and fine-grained data governance as they supply complete RBAC for clusters, pools, jobs and table-level. Moreover, new tools like Apache Nessie and Dremio Arctic (through Apache Iceberg) allow for git-like data management features on top of a LAKEHOUSE.
* **Scalability**  
  They are quite scalable thanks to decoupled processing and storage without requiring to load data into a warehouse for additional BI and governance capabilities. They also offer scalable metadata API. For example, with iceberg, when reading using a snapshot, the Iceberg API will perform the filtering and get the necessary data to scan. It can make the read effort planning without affecting the data.
* **Upserts & purging**  
  A data LAKEHOUSE has an integrated transaction tracking system, stored as metadata and transaction log. On DML operations, a new file is created and tracked by the metadata store. Purging can be done by analyzing the metadata and deleting inactive file versions.
* **Performance**  
  Provides high SQL performance on top of an object store. Additionally, data LAKEHOUSE optimizes size of objects without impacting running queries.
* **Reliability**  
  High-quality, reliable data thanks to ACID transactions and reduced ETL jobs. ACID-properties are achieved thanks to Optimistic Concurrency Control and Serializable isolation levels ensure that readers do not work with inconsistent data.
* **Applications**  
  Allows for BI, SQL, as well as Machine Learning and Data Science applications. Additionally it allows access to raw data in an object store directly via the DataFrames API (with Delta Tables). This architecture also works well with leading machine learning systems like TensorFlow, PyTorch and XGBoost (with Delta Tables, Iceberg is working on these features).

This architecture allows key warehouse capabilities to exist on a data lake. Furthermore, LAKEHOUSE solutions implement other optimizations on the engine layer (through Spark or Flink) to optimize query performance, such as caching, auxiliary data structures (indexes and statistics), and data layout optimizations. In comparison to data lakes, they reduced data redundancy and data staleness due to a single all-purpose data storage, reduced the so-called data swamps as data is now versioned, and added governance and security layers on top of it.

On the negative side, data LAKEHOUSE architecture is relatively new and immature, and some added features are still on a to-do list. There are also complaints on different topics, like Delta Lake’s reliance on querying uniquely through Delta Lake tables and not external tables or the complexities of using notebooks compared to the simple interface of modern Data warehouses.

1. **Which architecture to use?**

The convergence of data warehouses and lakes towards each other has brought us the new LAKEHOUSE architecture, but let’s summarize how each one stacks against the data LAKEHOUSE:

* If you need high-performance Business Analytics while having access to fine-grained data governance, data warehouses are your choice. The high performance of a warehouse is unmatched by the others. Nevertheless, they are challenging and expensive to scale and lack the flexibility to efficiently handle all data types. If high transaction throughput and different data types are a requirement, LAKEHOUSE architecture can be a solution. The main argument against this transition is the complex migration of such different architectures.
* If you want to scale and process petabytes of data affordably, having storage while preserving computing and provide high throughput (in opposition to low latency access), data lakes are your choice. On the other hand, data lakes do not provide fine-grained access control or high-performance business analytics. In case you need this, a transition to data LAKEHOUSE might be possible and easier to perform due to these architecture reliance on similar distributed technology.

We also want to briefly mention modern cloud data warehouses (such as Snowflakes, Clickhouse, Azure Synapse) as they provide comparable solutions to LAKEHOUSEs. Nevertheless, they are slightly different, as they behave more as a data warehouse trying to adopt data lake properties like decoupled compute and storage.

1. **Sources**

* [Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores](https://databricks.com/wp-content/uploads/2020/08/p975-armbrust.pdf)
* [Apache Iceberg](https://iceberg.apache.org/)
* [Apache Hudi](https://hudi.apache.org/)
* [Delta Tables](https://github.com/delta-io/delta)
* [Data Warehouse](https://en.wikipedia.org/wiki/Data_warehouse)
* [Data Lake](https://en.wikipedia.org/wiki/Data_lake)