O'REILLY®

Apache Iceberg
The Definitive Guide

Data Lakehouse Functionality, Performance, and Scalability on the Data Lake

Early Release
Raw & Unedited

Compliments of

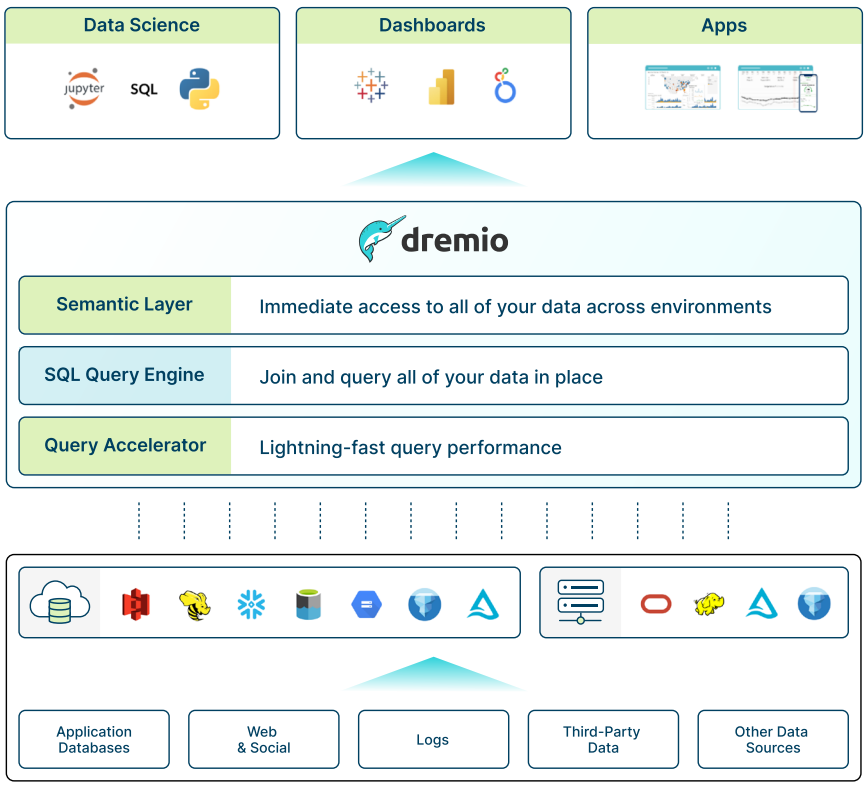
Tomer Shiran,
Jason Hughes,
Alex Merced &
Dipankar Mazumdar

The Easy and Open Data Lakehouse

Self-service analytics with data warehouse functionality and data lake flexibility across all of your data

Dremio is an open data lakehouse, providing self-service SQL analytics, data warehouse performance and functionality, and data lake flexibility across all of your data.

We believe in the future of open data, co-creating Apache Arrow and directly contributing to Apache Iceberg, the next-generation table format for data.



Try it for free at Dremio.com/get-started

Apache Iceberg: The Definitive Guide

Data Lakehouse Functionality, Performance, and Scalability on the Data Lake

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

Tomer Shiran, Jason Hughes, Alex Merced, and Dipankar Mazumdar

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Apache Iceberg: The Definitive Guide
by Tomer Shiran, Jason Hughes, Alex Merced, and Dipankar Mazumdar

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*https://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Aaron Black
Development Editor: Gary O'Brien
Production Editor: Elizabeth Faerm
Copyeditor: TO COME
Proofreader: TO COME

Indexer: TO COME
Interior Designer: David Futato
Cover Designer: Randy Comer
Illustrator: Kate Dullea

February 2024: First Edition

Revision History for the Early Release
2023-02-27: First Release

See *https://oreilly.com/catalog/errata.csp?isbn=9781098148621* for release details.

This work is part of a collaboration between O'Reilly and Dremio. See our statement of editorial independence.

978-1-098-14862-1

[LSI]

Table of Contents

CHAPTER 1
What Is a Data Lakehouse?

A Note for Early Release Readers

With Early Release ebooks, you get books in their earliest form—the authors' raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *gobrien@oreilly.com*.

Data is a primary asset for organizations to make critical business decisions. Whether it is analyzing historical trends of the annual sales of a product or making predictions about future opportunities, data shapes the direction for organizations to make reliable choices. Further, in today's day and age, data isn't just nice-to-have, but a requirement for not only winning in the market, but even competing at all. With such a massive demand for information, there has been an enormous effort to accumulate data generated by the various systems within an organization to derive insights.

At the same time, the rate at which the various operational and analytical systems have been generating data has also skyrocketed. While more data has presented enterprises the opportunity to make better-informed decisions, there is also a dire need to have a platform that allows storing and analyzing all of this data so it can be used to build analytical products such as Business Intelligence (BI) reports and machine learning models to support decision making. This chapter will walk us through the history and evolution of data platforms from a practical point of view and present the benefits of a lakehouse architecture with open table formats like Apache Iceberg.

How Did We Get Here? A Brief History

In terms of storage and processing systems, relational databases (RDBMS) have long been a standard option for organizations to keep a record of all of their transactional data. For example, if you are a transportation company, you would like to maintain information about any new bookings made by a customer. This new booking would be a new *row* in a relational database system. Information like this can support the day-to-day operations of a business. RDBMS systems used for these purposes support a specific data processing category called Online Transaction Processing (OLTP). Examples of these OLTP-optimized RDBMS systems are PostgreSQL, MySQL, and Microsoft SQL Server. These OLTP systems are designed and optimized for interacting with one or a few rows at a time very quickly. However, for the example above, if you want to understand the *average profit* made on all of the new bookings for the last quarter, using the data stored in an OLTP-optimized RDBMS will lead to significant performance problems when your data gets large enough.

Now, imagine that your organization has a large number of operational systems. These systems generate a vast amount of data. Your analytics teams' goal is to build dashboards that rely on aggregations of the data from these different data sources (application databases). Unfortunately, OLTP systems are not designed to deal with such complex aggregate queries involving a large number of historical records. These workloads are known as Online Analytical Processing (OLAP) workloads. To address these limitations, a different kind of system optimized for OLAP workloads was needed.

Foundational Components of a System Designed for OLAP Workloads

A system designed for OLAP workloads is composed of a set of technological components that enable supporting modern-day analytical workloads.

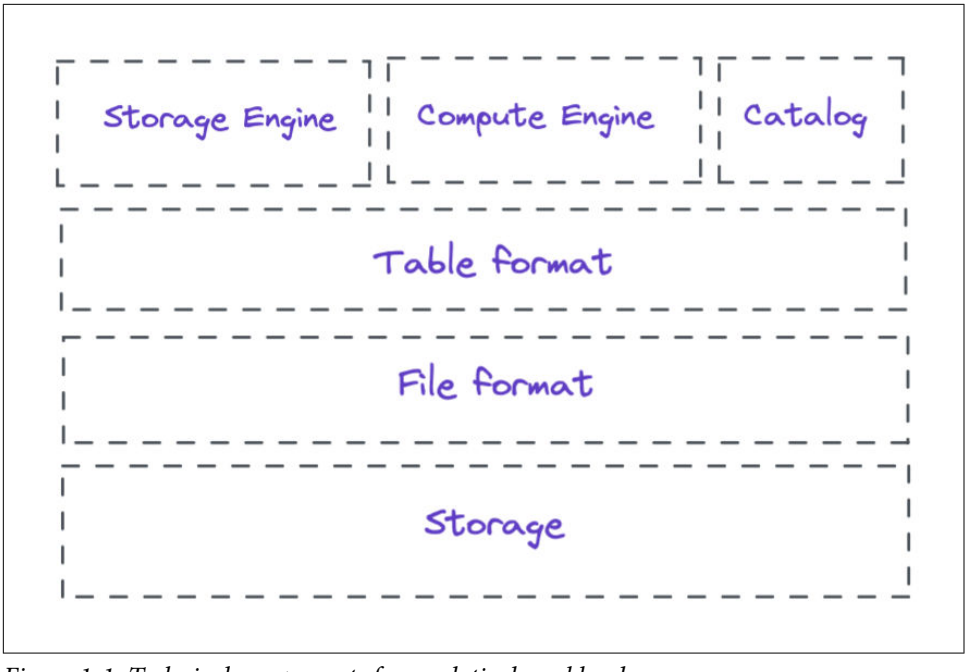


Figure 1-1. Technical components for analytical workloads

Storage

To analyze historical data coming in from a variety of sources, you need to have a system that allows you to store such huge amounts of data. Therefore, storage is the first component we would need in a system that can deal with analytical queries on large datasets. There are a few options for storage, such as a local file system on a direct-attached storage (DAS), a distributed file system on a set of nodes that you operate like Hadoop Distributed File System (HDFS), or object storage provided as a service by cloud providers like Amazon Simple Storage Service (S3).

Regarding the types of storage, you could use row-oriented databases or columnar. In recent years, columnar-oriented databases have seen a tremendous adoption rate as they have proved more efficient when dealing with vast volumes of data.

File Format

The file format is a component responsible for organizing the raw data in a particular format, which is then stored in a storage system. The choice of a file format impacts things such as compression of the files, data structure, and performance for a given workload.

File formats generally fall into three high-level categories: structured, semi-structured, and unstructured. In the structured and semi-structured categories, file

formats can be row-oriented or column-oriented (columnar). Row-oriented file formats store all columns of a given row together, while column-oriented file formats store all rows of a given column together. Two common examples of row-oriented file formats are comma-separated values (CSV) and Apache Avro. Examples of columnar file formats are Apache Parquet and Apache ORC.

Depending on the use cases, certain file formats can be more advantageous. For example, row-oriented file formats are generally better when dealing with a small number of records at a time. In comparison, columnar file formats are generally better if you are dealing with a sizable portion of records at a time.

Table Format

A table format is another critical component for a system that can support analytical workloads with aggregated queries on a vast volume of data. Table formats take the role of a metadata layer on top of the file formats described above and are responsible for specifying how the data files should be laid out on the storage.

Ultimately the goal of a table format is to abstract the complexity of the physical data structure and facilitate capabilities such as the ability to do data manipulation language (DML) operations (e.g., doing inserts, updates, deletes) and change a table's schema. Table formats also bring in the atomicity and consistency guarantees required for the safe execution of the DML operations on the data.

Storage Engine

A storage engine is the system responsible for actually doing the work of laying out the data in the form specified by the table format and keeping all the files & data structures up to date with the new data. Storage engines handle some of the critical tasks, such as physical optimization of the data, index maintenance, and getting rid of old data.

Catalog

When dealing with data from various sources and on a larger scale, it is important to identify the data you might need for your analysis quickly. A catalog's role is to tackle this problem by leveraging metadata to identify datasets. The catalog is the central location that engines and users can go to find out about the existence of a table and additional information about each table, such as table name, table schema, and where that table's data is stored on the storage system. Some catalogs are internal to a system and can only be directly interacted with via that system's engine, such as Postgres and Snowflake, while some catalogs are open for any system to use, such as Hive and Project Nessie.

Compute Engine

A compute engine is the final component needed in a system that can efficiently deal with a massive amount of data persisted in a storage system. A compute engine's role in such a system would be to run user workloads to process the data. Depending on the volume of data, computation load, and type of workload, you can utilize one or more compute engines to process the data. When dealing with a large dataset and/or heavy computational requirements, you might need to use a distributed compute engine in a processing paradigm called Massively Parallel Processing (MPP). A few examples of MPP-based compute engines are Apache Spark, Snowflake, and Dremio.

Bringing It All Together

Traditionally for OLAP workloads, these technical components have all been tightly coupled into a single system known as a data warehouse. Data warehouses allow organizations to store data coming in from a variety of sources and run analytical workloads on top of it. In the next section, we will discuss in detail the capabilities of a data warehouse, how the technical components are integrated, and the pros and cons of using such a system.

Data Warehouse

A data warehouse (DW) or OLAP database is a centralized repository that supports storing large volumes of data ingested from various sources such as operational systems, application databases, and logs.

Looking at the technical components described in the section above, this is how they get incorporated into a data warehouse. Figure 1-2 presents an architectural overview.

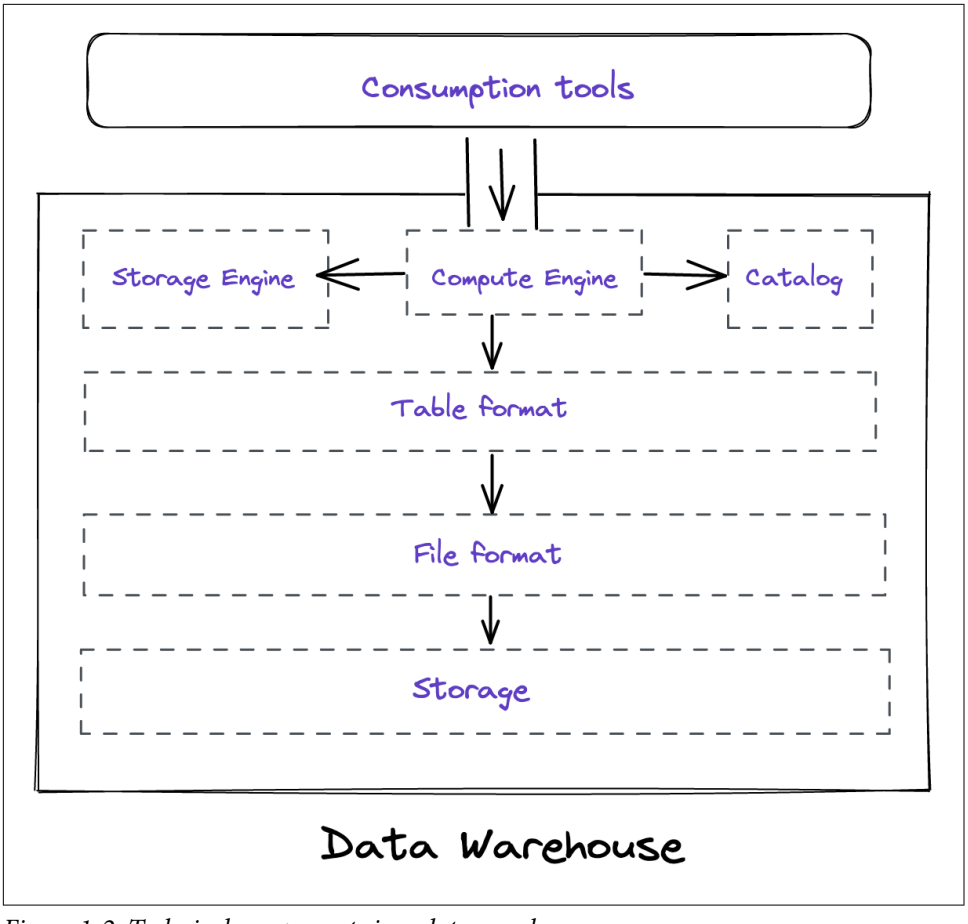


Figure 1-2. Technical components in a data warehouse

A data warehouse owns all the technical components in a single system. So, all the data stored in a DW system is stored on the DW's storage in the DW's proprietary file format in the DW's proprietary table format. This data is then managed exclusively by the DW's storage engine, registered in the DW's catalog, and can only be accessed by the user or analytical engines through the DW's compute engine.

Up until about 2015, the majority of DWs had the storage and compute components tightly coupled together on the same nodes, since most DWs were designed and run on-premises. However, this resulted in a lot of problems. Because datasets grew in volume faster and faster, as well as the number and intensity of workloads (i.e., compute tasks running on the warehouse), scaling became a big issue. Specifically, there was no way to independently increase the compute and storage resources depending on your tasks. If your storage needs grew faster than your compute needs,

it didn't matter – you still needed to pay for additional compute even though you didn't need it.

This led to the next generation of data warehouses being built with a big focus on the cloud. The next generation of data warehouses was built starting in around 2015 as cloud-native, allowing you to separate these two components and scale compute and storage resources as you would like for your tasks, as well as even shut down compute when you weren't using it and not lose your storage.

Pros and Cons of a Data Warehouse

While data warehouses, whether on-premises or cloud-based, make it easy for enterprises to make sense of all their historical data quickly, there are also certain areas where a warehouse still causes issues. We list the pros and cons of a data warehouse in Table 1-1.

Table 1-1. Pros and cons of a data warehouse

Pros	Cons
A data warehouse serves as the single source of truth as it allows storing & querying data from various sources.	Data in a warehouse is locked into a vendor-specific system that only the warehouse's compute engine can use, thereby locking the data.
Supports querying vast amounts of historical data enabling analytical workloads to run quickly.	Expensive in terms of both storage and computation. As the workload increases, the cost becomes hard to manage.
Provides effective data governance policies to ensure data is available, usable and aligned with the security policies.	Mainly supports structured data.
Organizes the data for you, ensuring it's optimized for querying	Organizations cannot run advanced analytical workloads such as machine learning natively in a data warehouse.
Ensures data written to a table conforms to the technical schema	

Data warehouses act as a centralized repository for organizations to store all their data coming in from a multitude of sources, allowing data consumers such as analysts and BI engineers to access data easily and quickly from one single source to start their analysis. In addition, the technological components powering data warehouses enable accessing vast volumes of data while supporting workloads such as business intelligence to run on top of it.

Although data warehouses have been elemental in the democratization of data and allowed businesses to derive historical insights from varied data sources, they are primarily limited to relational workloads. For example, if you go back to the transportation company example from earlier and say now, you want to derive insights into *how much total sales you will make in the next quarter.* In this case, you will need to build a forecasting model using historical data. However, you cannot achieve this capability natively with a data warehouse as the compute engine & the other technical components are not designed for machine learning-based tasks. So, the only

viable option is moving or exporting the data from the warehouse to other platforms supporting it. This means you will have data in multiple copies, which can lead to critical issues such as data drift, model decay, etc.

Another hindrance to running advanced analytical workloads on top of a data warehouse is that it only has support for structured data. But, the rapid generation and availability of other types of data, such as semi-structured and unstructured data (JSON, images, texts, etc.), have allowed machine learning models to bring out interesting insights. For our example, this could be understanding the *sentiments of all the new booking reviews* made in the last quarter. This ultimately impacts an organization's ability to make future-oriented decisions.

There are also specific design challenges in a data warehouse. If you go back to the diagram (Figure 1-2) above, you can see that all six technical components are tightly coupled in a data warehouse. Before you understand what that implies, an essential thing to observe is that both the file and the table formats are *internal* to a particular data warehouse. This design pattern leads to a *closed form* of data architecture. It means that the actual data is accessible only using the data warehouse's compute engine, which is specifically designed to interact with the warehouse's table and file formats. This type of architecture leaves organizations with a massive concern about locked-in data. With the increase in workloads and the vast volumes of data ingested to a warehouse over time, you are bound to that particular platform. And that means your analytical workloads, such as BI and any future tools you plan to onboard, have to run specifically on top of this particular data warehouse only. This also prevents you from migrating to another data platform that can cater specifically to your requirements.

Additionally, a significant cost factor is associated with storing data in a data warehouse and using the compute engines to process that data. This cost only increases with time as you increase the number of workloads in your environment, thereby invoking more compute resources. Other than the monetary costs, there are additional overheads, such as the need for engineering teams to build and manage numerous ETL (extract, transform, load) pipelines to move data from operational systems, delayed time-to-insight on the part of the data consumers, etc. These challenges have led organizations to seek alternative data platforms that allow data to be within their control and stored in open file formats, thereby allowing downstream applications such as BI and machine learning to run parallelly with much-reduced costs. It led to the emergence of Data Lakes.

Data Lake

While a data warehouse provides a mechanism for running analytics on structured data, it still had several issues that left a need for different solutions:

- A data warehouse could only store structured data
- Storage in a data warehouse is generally more expensive than on-prem Hadoop clusters or cloud object storage.

To address these issues the goal was to have an alternative storage solution that was cheaper and could store all our data. This is what's called the data lake.

Originally, you'd use a Hadoop to allow you to use a cluster of inexpensive computers to store large amounts of structured and unstructured data. Although it wasn't enough to just be able to store all this data. You'd want to run analytics on it too.

The Hadoop ecosystem included MapReduce, an analytics framework from which you'd write analytics jobs in Java and run them on the cluster. Many analysts are more comfortable writing SQL than Java, so Hive was created to convert SQL statements into MapReduce jobs.

To write SQL, a mechanism to distinguish which files in our storage are part of the dataset or table we want to run the SQL against was needed. This resulted in the birth of the Hive table format which recognized a directory and the files inside it as a table.

Over time. people moved away from using Hadoop clusters to using Cloud Object storage as it was easier to manage and cheaper to use. MapReduce also fell out of favor for other distributed query engines like Apache Spark, Presto, and Dremio. What did stick around was the Hive table format which became the standard in the space for recognizing files in your storage as singular tables on which you can run analytics.

A distinguishing feature of the data lake as compared to the data warehouse is the ability to leverage different compute engines for different workloads. This is important because there's never been a silver bullet of a compute engine that is best for every workload. This is just inherent to the nature of computing since there are always tradeoffs, and what you decide to tradeoff determines what a given system is good for and what it is not as well suited for.

Note that in data lakes, there isn't really any service that fulfills the needs of the storage engine function. Generally the compute engine decides how to write the data, then the data is usually never revisited and optimized, unless rewriting entire tables or partitions which is usually done on an ad-hoc basis. Refer to Figure 1-3 to see how the components of a data lake interact with one another.

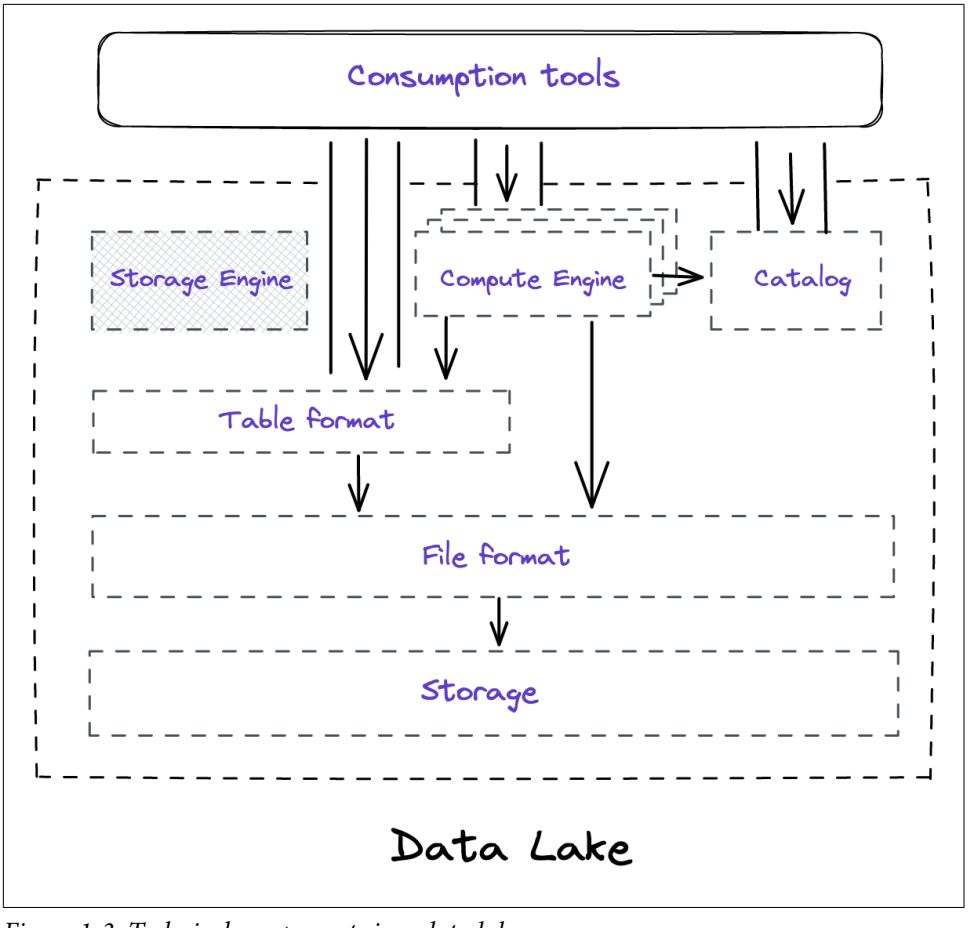


Figure 1-3. Technical components in a data lake

Pros and Cons of a Data Lake

No architectural pattern is perfect and that applies to data lakes. While data lakes have a lot of benefits like Lower Costs, the ability to store in open formats and handle unstructured data; data lakes also have several disadvantages such as performance issues, lack of ACID guarantees and lots of configuration. You can see a summary of these pros and cons in Table 1-2.

Pro: Lower Cost

The costs of storing data and executing queries on a data lake are much lower than in a data warehouse. This makes a data lake particularly useful for enabling analytics on data that isn't high enough priority to justify the cost of a data warehouse enabling a wider analytical reach.

Pro: Store Data in Open Formats
: In a data lake you can store the data in any file format you like unlike data warehouses where you have no say in how the data is stored, which would typically be a proprietary format built for that particular data warehouse. This allows you to have more control over the data and consume the data in a greater variety of tools that can support these open formats.

Pro: Handle Unstructured Data
: Data warehouses can't handle unstructured data, so if you wanted to run analytics on unstructured data the data lake was the only option.

Con: Performance
: Since each component of a data lake is decoupled, many of the optimizations that can exist in tightly coupled systems are absent. While they can be recreated, it requires a lot of effort and engineering performant to cobble the components (storage, file format, table format, engines) in a way to give you the comparable performance of a data warehouse. This made data lakes undesirable for high priority data analytics where performance and time mattered.

Con: Lots of Configuration
: As previously mentioned, creating a tighter coupling of your chosen components with the level of optimizations you'd expect from a data warehouse would require significant engineering. This would result in a need for lots of data engineers to configure all these tools, which can also be costly.

Table 1-2. Pros and cons of a data lake

Pros	Cons
• Lower Cost • Store Data in Open Formats • Handle unstructured data	• Performance • Lack of ACID Guarantees • Lots of Configuration

Should I Run Analytics on the Data Lake or Data Warehouse?

While Data Lakes provided a great place to land all your structured and unstructured data, there were still imperfections. After running ETL to land your data in your data lake you'd generally take one of two tracks when running analytics.

A subset of data goes to the data warehouse
: You'd set up an additional ETL pipeline to create a copy of a curated subset of data that is for high priority for analytics and store it in the warehouse to get the performance and flexibility of the data warehouse.

  This results in several issues:

- Additional costs in the compute for the additional ETL work and the cost for storing a copy of data you are already storing in a data warehouse where the storage costs are often greater.
- Additional copies of the data may be needed to populate data marts for different business lines and even more copies as analysts create physical copies of data subsets in the form of BI extracts to speed up dashboards. Leading to a web of data copies that are hard to govern, track and keep in sync.

You run analytics directly on the data lake
: You'd use query engines that support data lake workloads like Dremio, Presto, Apache Spark, Trino, Apache Impala and more to execute queries on the data lake. These engines are generally well suited for read-only workloads. However, due to the limitations of the Hive table format, they ran into complexity when trying to update the data safely from the data lake.

So the data lake and data warehouse each have their unique benefits and unique cons. It would be to our advantage to develop a new architecture that brings together all these benefits while minimizing all their faults, and that architecture is called a data lakehouse.

Enter Data Lakehouse

While using a data warehouse gave us performance and ease of use, analytics on data lakes gave us lower costs and reduced data drift from a complex web of data copies. The desire to thread the needle leads to great strides and innovation leading to what we now know as the data lakehouse.

What makes a data lakehouse truly unique are data lake table formats that eliminate all the previous issues with the Hive table format. You store the data in the same places you would with a data lake, you use the query engines you would use with a data lake, your data is stored in the same formats it would be on a data lake, what truly transforms your world from a "read only" data to a "center of my data world" data lakehouse is the table format (refer to Figure 1-4). Table formats enabled better consistency, performance and ACID guarantees when working with data directly on your data lake storage leading to several value propositions.

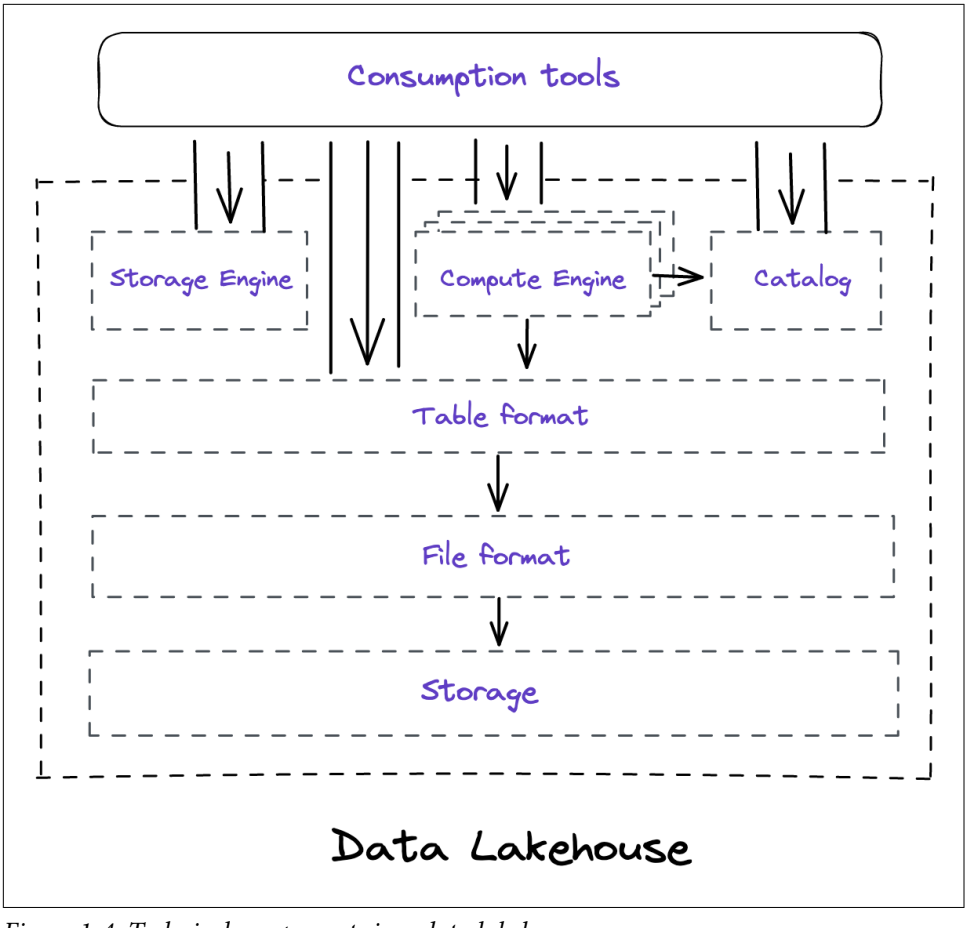


Figure 1-4. Technical components in a data lakehouse

Fewer Copies, Less Drift
: With ACID guarantees and better performance you can now move workloads typically saved for the data warehouse like updates and other data manipulation. If you don't have to move your data to the lakehouse you can have a more streamlined architecture with fewer copies. Fewer copies mean less storage costs, less compute costs from moving data to a data warehouse, and better governance of your data to maintain compliance with regulations and internal controls.

Faster Queries, Fast Insights
: The end goal is always to get business value from quality insights from our data, everything is else just steps to that end. If you can get faster queries that means you can get insights faster. Data Lakehouses enable faster performing queries by using optimizations at the query engine, table format and file format.

Mistakes Don't Have to Hurt

Data Lakehouse table formats enable the possibility to undo mistakes by using snapshot isolation, allowing you to revert the table back to prior snapshots. You can work with your data but not have to be up at night wondering if a mistake will lead to hours of auditing, repairing then backfilling.

Affordable Architecture is Business Value

There are two ways to increase profits, increase revenue and lower costs, and data lakehouses not only help you get business insights to drive up revenue but can also help you lower costs. Reduce storage costs from avoiding duplication of your data, avoid additional compute costs from additional ETL work to move data and enjoy lowers prices for the storage and compute you are using relative to typical data warehouse rates.

Open Architecture, Peace of Mind

Data Lakehouses are built on open formats such Apache Iceberg as a table format and Apache Parquet as a file format. Many tools can read and write to these formats which allows you avoid vendor lock-in which results in cost creep and prevents tool lock-out where you data sits in formats in which tools that could be great solutions can't access. Use open formats, you can rest easy that your data won't be siloed into a narrow set of tools.

The Key to the Puzzle

So, with modern innovations from the open standards previously discussed, the best of all worlds can exist by operating strictly on the data lake, and this architectural pattern is the data lakehouse. The key component that makes all this possible is the table format that enables engines to have the guarantees and performance when working with your data that just didn't exist before, now let's get started with the Apache Iceberg table format.

About the Authors

Tomer Shiran is the Founder and Chief Product Officer of Dremio, an open data lakehouse platform that enables companies to run analytics in the cloud without the cost, complexity and lock-in of data warehouses. As the company's founding CEO, Tomer built a world-class organization that has raised over $400M and now serves hundreds of the world's largest enterprises, including 3 of the Fortune 5. Prior to Dremio, Tomer was the 4th employee and VP Product of MapR, a Big Data analytics pioneer. He also held numerous product management and engineering roles at Microsoft and IBM Research, founded several websites that have served millions of users and hundreds of thousands of paying customers, and is a successful author and presenter on a wide range of industry topics. He holds an MS in Computer Engineering from Carnegie Mellon University and a BS in Computer Science from Technion - Israel Institute of Technology.

Jason Hughes is the Director of Technical Advocacy at Dremio. Previously at Dremio, he's been a Product Director, Technical Director and a Senior Solutions Architect. He's been working in technology and data for over a decade, including roles as tech lead for the field at Dremio, the pre-sales and post-sales lead for Presto and QueryGrid for the Americas at Teradata, and leading the development, deployment, and management of a custom CRM system for multiple auto dealerships. He is passionate about making customers and individuals successful and self-sufficient. When he's not working, he's usually taking his dog to the dog park, playing hockey, or cooking (when he feels like it). He lives in San Diego, California.

Alex Merced is a developer advocate for Dremio and has worked as a developer and instructor for companies like GenEd Systems, Crossfield Digital, CampusGuard and General Assembly. Alex is passionate about technology and has put out tech content on outlets such as blogs, videos and his podcasts Datanation and Web Dev 101. Alex Merced has contributed a variety of libraries in the Javascript and Python worlds including SencilloDB, CoquitoJS, dremio-simple-query and more.

Dipankar Mazumdar is currently a Data Eng/Science Advocate at Dremio where his primary focus is advocating data practitioners on Dremio's open lakehouse platform and various open-sourced projects, such as Apache Iceberg. Dipankar is also interested in Visual Analytics research, and his latest work was on "Explainability of ensemble models" using multidimensional projection techniques.