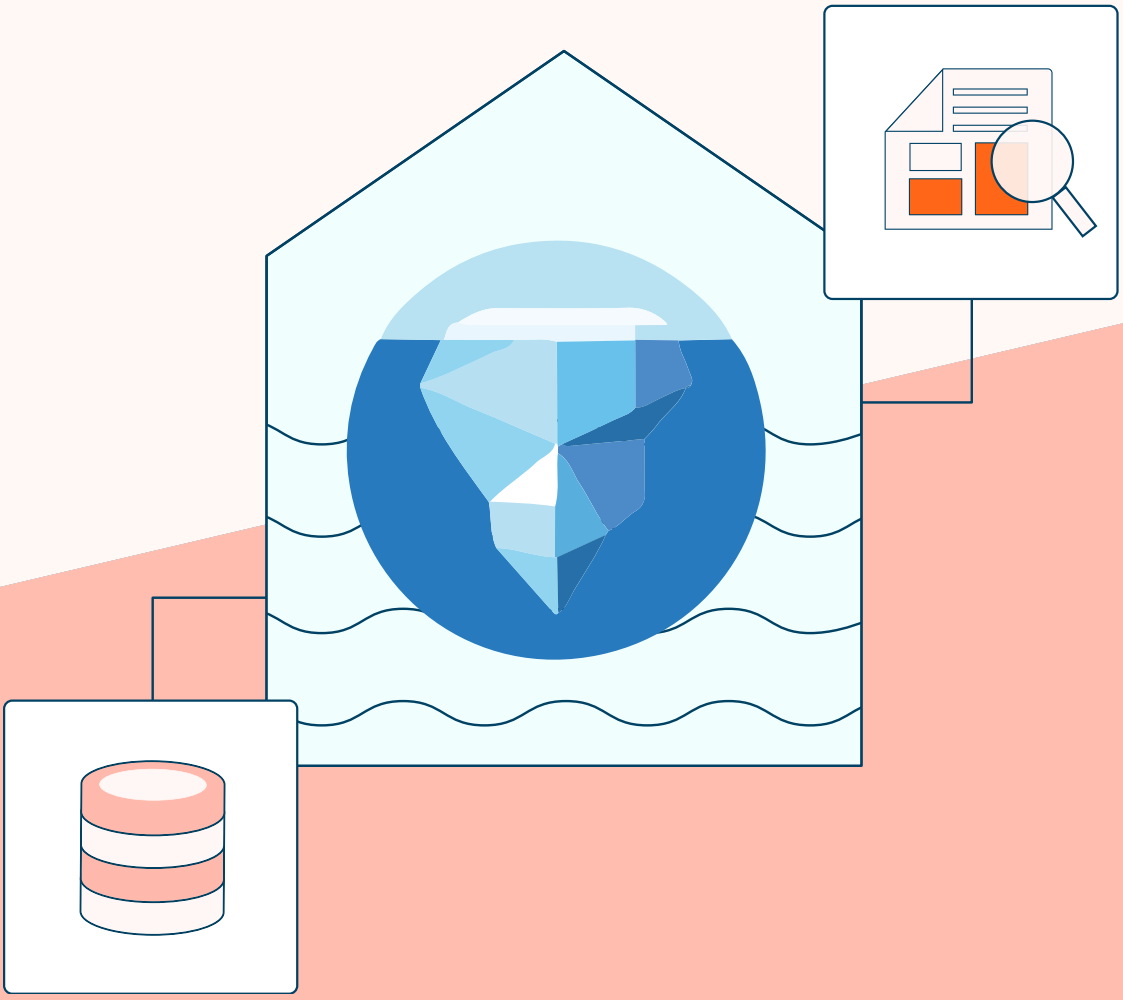
dremio

WHITEPAPER

The Apache Iceberg Advantage



What Is a Data Lake Table Format?

Table formats have evolved to address the need to efficiently organize and leverage the growing volumes of data stored in a data lake. Data lakes, initially designed primarily for cost-effective and scalable storage, presented a new challenge for data teams: how to provide streamlined access to the raw data they store.

File formats such as Apache Parquet represented the first attempt to provide some structure for the data lake, delivering significant compression and performance benefits over formats like comma-separated values (CSV). Table formats build on the advantages of file formats, providing a structured blueprint that transforms disparate data files into a cohesive structure that's easy to work with for users — a table. It encompasses essential elements like schema definition, partitioning strategies, and metadata, giving raw data a sense of order and accessibility.

One of the hallmark features of table formats is their ability to facilitate ACID-compliant interactions. ACID compliance — ensuring atomicity, consistency, isolation, and durability of data operations — enhances reliability and integrity. ACID compliance is a foundational requirement of data warehouses (DW) and relational database management systems (RDBMS). By adhering to ACID principles, table formats enable multiple users to concurrently engage with the data without compromising data integrity. Table formats act as the bridge between the raw, unorganized data of data lakes and the user-friendly, structured data that data consumers can seamlessly explore and leverage for a wide range of purposes.

Hive Tables: The First Table Format for Data Lakes

The Hive table format was created in 2009 (and introduced via just **three short bullet points in a whitepaper**) to allow Facebook employees to write SQL against Facebook's internal Hadoop clusters, and it became the de facto standard within the Hadoop ecosystem.

In the Hive table format, a table is defined as the entire contents of one or more directories. For non-partitioned tables, this is a single directory. For partitioned tables, which are much more common in the real world, the table is composed of many directories — one directory per partition. This mapping of table name to directory/directories is housed in a metastore such as the Hive metastore or AWS Glue Catalog.

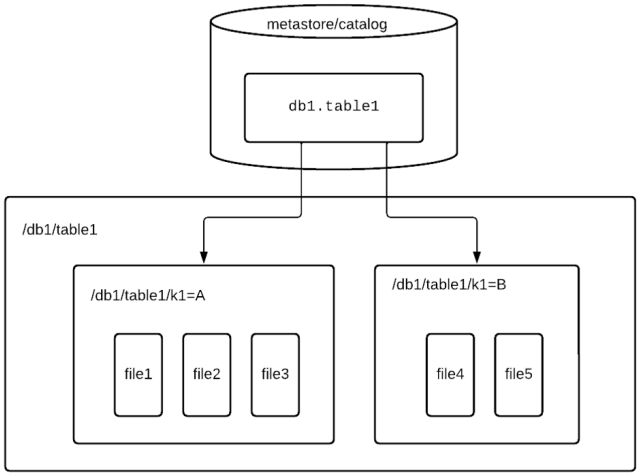


Figure 1: The structure of the traditional de facto standard for data lakes, the Hive table format

While its shortcomings started to arise early on, workarounds were continually used to get around them, rather than addressing the core problem: data for a given table was defined as a set of *one or more directories*, rather than tracking the *individual files* like Apache Iceberg.

Let's review the challenges faced when using the Hive table format, rather than a modern table format like Apache Iceberg.

Challenges of Using the Hive Table Format

1. **Changes to the data are inefficient**
   Tracking data in a table is done at the directory level, so users cannot change data at a granularity lower than a whole directory in a transactionally consistent way. To solve this, users replicate a whole partition to a new location behind the scenes, making updates or deletes during replication, then updating the partition's location in the metastore to the new location. This workaround is inefficient, especially with large partitions and small and/or frequent changes to the data, all of which are common in modern datasets and use cases.

2. **There's no way to safely change data in multiple partitions as part of one operation**
   Because the only transactionally consistent operation users can do to change data is to swap a single partition, they can't change data in multiple partitions at the same time in a consistent/atomic way, even something as simple as adding a file to two partitions.

3. **In practice, multiple jobs can't safely modify the same dataset**
   There isn't a well-adopted method in the Hive table format to deal with more than one process or person updating the data at the same time. There is one method, but it's restrictive and causes issues so that only Hive jobs can utilize it. To overcome this, organizations need to define and coordinate strict controls on who can write and when or risk data loss due to concurrent changes in the data.

4. **The directory listings required for large tables take a long time**
   Tools such as query engines need to get a list of what files are in all of the partition directories at runtime, and getting a response generally takes a long time. In one notable instance, Ryan Blue, the co-creator of Iceberg at Netflix, **shared an example** where it took over 9 minutes just to plan a query due to the directory listings. High-performance query engines such as Dremio cache these directory listings, and the system refreshes these caches regularly to ensure that queries can process the latest data. That said, the cost and administrative burden of the metadata cache are a challenge for some organizations as well.

5. **Because file maintenance is inefficient and difficult, costs can be much higher than needed**
   Trade-offs are inherent to data processing, and many of these trade-offs can lead to suboptimal file sizes for tables. This is known as the small files problem, where the cost to read a set amount of data grows as you increase the number of files the data is stored in, both in terms of performance and monetary cost in the cloud. Data teams can correct file sizes after the fact, but because changes in Hive are inefficient and difficult, they often are not made. In **one example**, Insider was able to reduce overall costs by 90% with Apache Iceberg by addressing the small files problem.

6. **Users have to know the physical layout of the table**
   If a table is partitioned by when an event occurred, it is often done via multi-level partitioning: first by the event's year, then by the event's month, then by the event's day, and sometimes even finer granularity. But when a user is presented with events, the intuitive way to query the events after a certain point in time looks like: WHERE event_ts >= '2023-05-10 12:00:00'. In this situation, the query engine does a full table scan, which takes much longer than if the available partition pruning was done to limit the data.

   This full-table scan happens because there is no mapping from the event's timestamp familiar to the user (2023-05-10 12:00:00) to the physical partitioning scheme (year=2023, then month=05, then day=10).

   Instead, all users need to be aware of the partitioning scheme and write their query as: WHERE event_ts >= '2023-05-10 12:00:00' AND event_year >= '2023' AND event_month >= '05' AND (event_day >= '10' OR event_month >= '06'). (Note: this partition-pruning query gets even more complicated if you were to look at events after May of 2022 instead.)

7. **Hive table statistics are usually stale**
   Table statistics are gathered in an asynchronous periodic read job; therefore, the statistics are often out of date. Furthermore, gathering these statistics requires an expensive read job that results in significant scanning and computation, so these jobs run infrequently, if ever. Consequently, some engines disregard stats for Hive tables altogether.

8. **The filesystem layout has poor performance on cloud object storage**
   Any time a user needs to read a lot of data, cloud object storage (e.g., S3, GCS) architecture dictates the reads should have as many different prefixes as possible, so they get handled by different nodes in the cloud object storage backend. However, since all data in a partition in the Hive table format has the same prefix and users generally read all of the data in a partition (or at least all of the Parquet/ORC footers in a partition), all of the read requests hit the same cloud object storage node, reducing the performance of the read operation.

Enter Apache Iceberg

Netflix ran into several of these challenges with the Hive table format and, after implementing all the usual workarounds, realized that the core issue was the Hive table format itself. The fundamental problem came down to the fact that the Hive table format tracks what data is in a table at the folder level. If they tracked what data was in a table at the file level instead, knowing exactly which files made up a table, it would resolve many of the challenges and lay the foundation for additional useful capabilities.

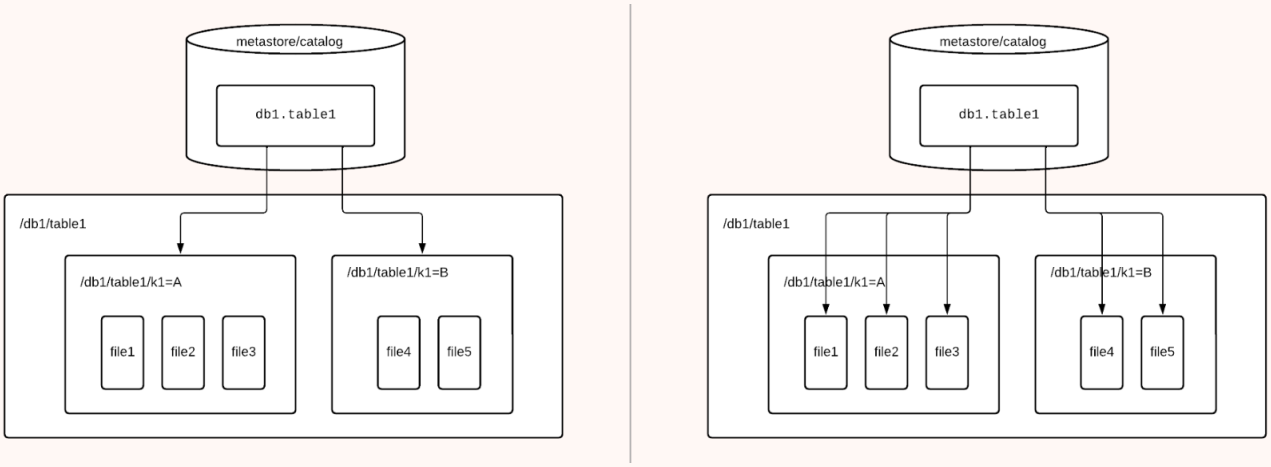


Figure 2: On the left, the Hive table format tracks what directories make up a table. On the right, the Apache Iceberg table format tracks what files make up a table. Diagram is simplified.

Apache Iceberg Capabilities

Let's review the core capabilities Iceberg provides:

1. **ACID transactions**
   Apache Iceberg leverages **optimistic concurrency control** to provide ACID guarantees for transactions across multiple readers and writers. This ensures that transactions either fully succeed or fail. Apache Iceberg's snapshot-based ACID compliance enables atomic transactions on tables, safeguarding against potential data loss and ensuring readers see a completely consistent view of the table.

2. **Time-travel**
   Apache Iceberg offers immutable snapshots, preserving the historical state of tables and enabling what is often referred to as time-travel. This feature allows you to query the table's state at specific points in the past, aiding tasks such as end-of-quarter reporting or re-creating the output of a machine learning model at a certain moment in time. With this capability, there's no need to duplicate the table's data to a separate location, enhancing both efficiency and accuracy.

3. **Version rollback**
   Apache Iceberg's version rollback capabilities provide a flexible and resilient approach to managing table states. If an error or undesired change occurs, you can effortlessly roll back the table to a prior state. This "undo" functionality simplifies error recovery and enhances the robustness of data management, making it a valuable feature for maintaining data integrity and consistency.

4. **Easy and safe schema evolution**
   Apache Iceberg's schema evolution capabilities allow for seamless and consistent alterations to the data structure. You can make changes to the table schema without affecting the existing data or requiring a complete rewrite of the table. This flexibility facilitates tasks such as adding new fields or modifying existing ones to keep pace with changing business requirements. Unlike traditional methods that can lead to inconsistencies or substantial downtime, Iceberg ensures that schema modifications are implemented efficiently and without disruption, maintaining the integrity and usability of the data.

5. **Partition evolution**
   Apache Iceberg enables simple and easy management of partitioning changes, addressing a previously complex and costly process. In the past, altering partitioning could necessitate rewriting the entire table, but Iceberg allows updates to the partitioning method without this rewrite. If desired, parts of the table can still be rewritten to a new partitioning scheme, but the process is far simpler. Since these changes are closely tied to metadata, adjustments to the table's structure are both efficient and economical. This capability streamlines the task of keeping partitioning aligned with evolving requirements, reducing both time and expense.

6. **Hidden partitioning**
   Apache Iceberg's hidden partitioning feature addresses a common issue where users don't know how a table is physically partitioned. Traditional partitioning in the Hive table format, as described earlier, is based on the different parts of a timestamp and often leads to queries that accidentally result in full table scans. This drastically reduces the query's performance and increases the query's cost. Iceberg's approach to partitioning resolves this by allowing the definition of the column on which physical partitioning is based, along with an optional transformation, such as bucket, truncate, year, month, day, or hour. As a result, users can write more intuitive queries that naturally benefit from partitioning without needing to add extra filter predicates on specific partitioning columns. This approach simplifies the querying process and improves overall efficiency.

7. **Object storage layout**
   Iceberg addresses the performance problems of the file layout on object storage by abandoning the traditional file layout approach to instead rely on a list of files within its metadata structure. This means the actual data files themselves don't need to be in any particular location, as long as the manifest files list the right location for that file. This allows Apache Iceberg to use traditional file layouts on systems like HDFS and use alternative file layouts like including randomized prefixes on object storage for better scaling and performance.

8. **Data optimization**
   Iceberg provides data optimization capabilities to improve the performance of operations that work with the data.

   One example is compaction, where data files can be rewritten to an optimal size while still providing consistency and without interrupting usage of the table. This is a simple solution to the small files problem.

   Another example is **optimizing the layout of the data**. This can be done via a simple process of sorting the data at the time of writing or a more complex sorting process called z-order, which helps optimize queries even when only the second field, in a table sorted by multiple fields, is used in the query.

9. **Multiple ways to update and delete data**
   Apache Iceberg provides three ways to update and delete data: copy-on-write, merge-on-read with positional deletes, and merge-on-read with equality deletes. There are trade-offs for each strategy, so selection depends on the situation. There's a default that works well, but you can easily change the strategy. You can learn more about these strategies and when to use which **here**.

Apache Iceberg Benefits

Iceberg's capabilities bring many benefits to your data ecosystem:

1. **Cost reduction**
   With Iceberg, it's much easier, and therefore more common, to be able to lower both your object storage read costs and your compute costs, thanks to Iceberg making file organization and optimization much easier. For example, after migrating from the Hive table format to Apache Iceberg, **Insider revealed** they were able to cut their S3 costs by 90% and their EC2 and EMR costs by 20%.

2. **Improved performance and efficiency**
   Apache Iceberg makes many data operations more performant and efficient through use of its capabilities:

   - Updating data is more performant and efficient

from a compute, people, process, and time perspective because you no longer need to duplicate an entire partition to change a subset of the data in it thanks to Iceberg's lightweight ACID transaction capability.

- Taking a snapshot of a table is much more efficient, thanks to Iceberg's time-travel and tagging capabilities.
- Fixing mistakes made to a table is much more efficient, thanks to Iceberg's time-travel and version rollback capabilities.
- Querying data is more performant and efficient from a compute and time perspective since their queries:
  - Leverage partitioning where possible, without them having to know how the table is physically structured, thanks to Iceberg's hidden partitioning
  - Read less data when a table has multiple fields get filtered on independently, thanks to Iceberg's z-order sorting
  - Don't need to do a listing of the folders to get the list of files at runtime, which can be slow especially at scale – rather the engine can read a comparatively small set of files it knows the path for to retrieve the full list of files – thanks to Iceberg tracking the individual files
  - Don't hit object storage rate limiting when reading large tables and/or partitions, thanks to Iceberg's object storage layout

3. **Guaranteed data consistency**
When modifying data in a table, any changes that are made follow ACID properties, even when multiple people from different engines are making changes to the same table at the same time. Snapshot isolation enables users to read a table while someone else makes changes, and neither will see the other's changes if a commit isn't made by the time a read is initiated.

4. **Iceberg makes it easy for tables to respond to business changes**
Object storage layout and partition evolution eliminate the risk of performance degradation as data volumes grow. Schema evolution dramatically reduces the risk and impact of schema changes. Finally, Iceberg enjoys broad ecosystem support for both reads and writes (Dremio, Spark, Flink, Trino, Athena, BigQuery, and many others), so data teams who choose Iceberg can leverage the broadest set of data tools.

Real-World Usage of Iceberg

Apache Iceberg is rapidly becoming the go-to solution for large-scale data management and analytics across industries. Leveraging its flexibility, scalability, and performance, companies are finding useful ways to streamline their data workflows and enhance analytic capabilities. Here are some resources from organizations that have shared their experiences and insights on how they have integrated Iceberg into their systems to meet their specific needs:

- **Bilibili** – How Bilibili Builds OLAP Data Lakehouse with Apache Iceberg
- **Pinterest** – Scaling Row-Level Deletions at Pinterest
- **Insider** – How We Migrated Our Data Lake to Apache Iceberg
- **Apple** – Lakehouse: Smart Iceberg Table Optimizer
- **SK Telecom** – Journey to Iceberg with Trino
- **Orca Security** – Orca Security's Journey to a Petabyte-Scale Data Lake with Apache Iceberg and AWS Analytics

Summary

As you've seen, the transition from Hive table format to Apache Iceberg is not just an upgrade; it's a paradigm shift that addresses the core inefficiencies and limitations inherent in traditional data lake table formats. Iceberg offers a robust set of capabilities — from ACID transactions and time-travel to hidden partitioning and optimized object storage layout — that not only resolve the challenges posed by Hive tables but also introduce new functionalities that are critical for modern data operations. Organizations that have made the switch report substantial benefits, including reductions in storage and compute costs, improved performance, improved data consistency, and the flexibility to adapt to changing business requirements.

As data continues to grow in volume, variety, and velocity, the need for a scalable, efficient, and reliable table format becomes increasingly urgent. Apache Iceberg is not just an alternative to Hive; it's the next evolutionary step for data lakes. Adopting Iceberg is not merely a technical decision but a strategic move that will future-proof your data infrastructure, optimize costs, and unlock new capabilities for data-driven decision-making.

dremio

ABOUT DREMIO

Dremio is the easy and open data lakehouse, providing self-service analytics with data warehouse functionality and data lake flexibility across all of your data. Use Dremio's lightning-fast SQL query service and any other processing engine on the same data. Dremio increases agility with a revolutionary data-as-code approach that enables Git-like data experimentation, version control, and governance. In addition, Dremio eliminates data silos by enabling queries across data lakes, databases, and data warehouses, and by simplifying ingestion into the lakehouse. Dremio's fully managed service helps organizations get started with analytics in minutes, and automatically optimizes data for every workload. As the original creator of Apache Arrow and committed to Arrow and Iceberg's community-driven standards, Dremio is on a mission to reinvent SQL for data lakes and meet customers where they are on their lakehouse journey.