Table Format Comparison

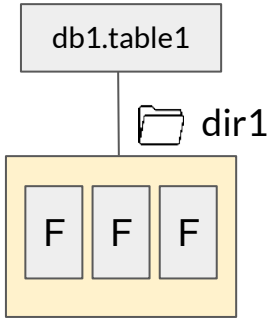
Agenda

- What's a table format?
- Why do we need a new one?
- The Main Three Table Formats
- Feature by Feature Comparison

What is a table format?

- A way to organize a dataset's files to present them as a single "table"
- A way to answer the question "what data is in this table?"

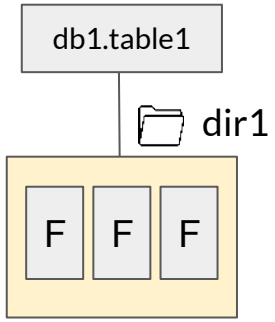
Old way (Hive)



A table's contents is all files in that table's directories

Hive table format

- A table's contents is all files in that table's directories
- The old de-facto standard



Pros

- Works with basically every engine since it's been the de-facto standard for so long
- More efficient access patterns than full-table scans for every query
- File format agnostic
- Atomically update a whole partition
- Single, central answer to "what data is in this table" for the whole ecosystem

Cons

- Smaller updates are very inefficient
- No way to change data in multiple partitions safely
- In practice, multiple jobs modifying the same dataset don't do so safely
- All of the directory listings needed for large tables take a *long* time
- Users have to know the physical layout of the table
- Hive table statistics are often stale

How can we resolve these issues?

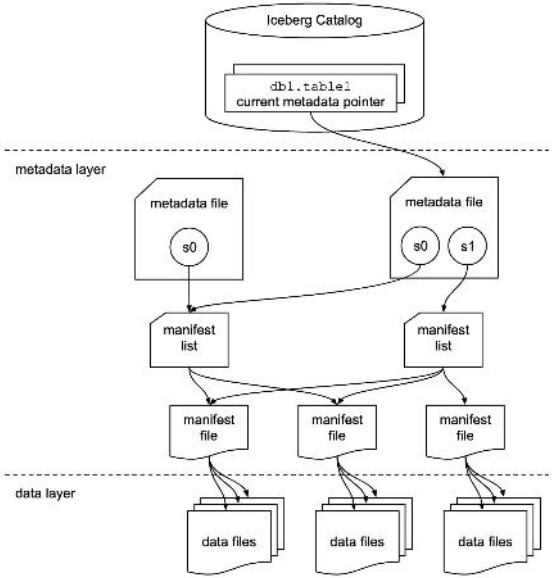
Apache Iceberg

Apache Hudi

Delta Lake

Overview of Architecture

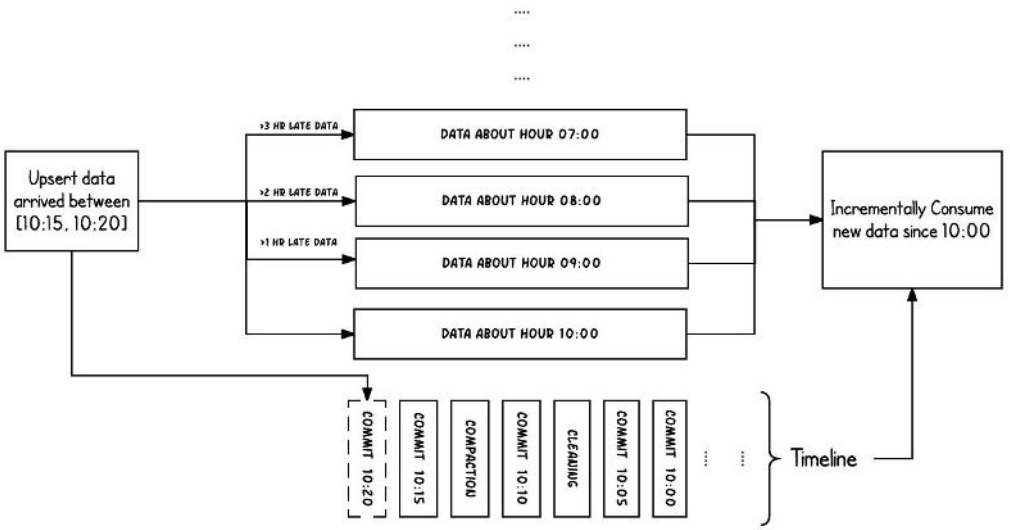
Apache Iceberg



Apache Iceberg's approach is to define the table through three layers of metadata. These categories are:

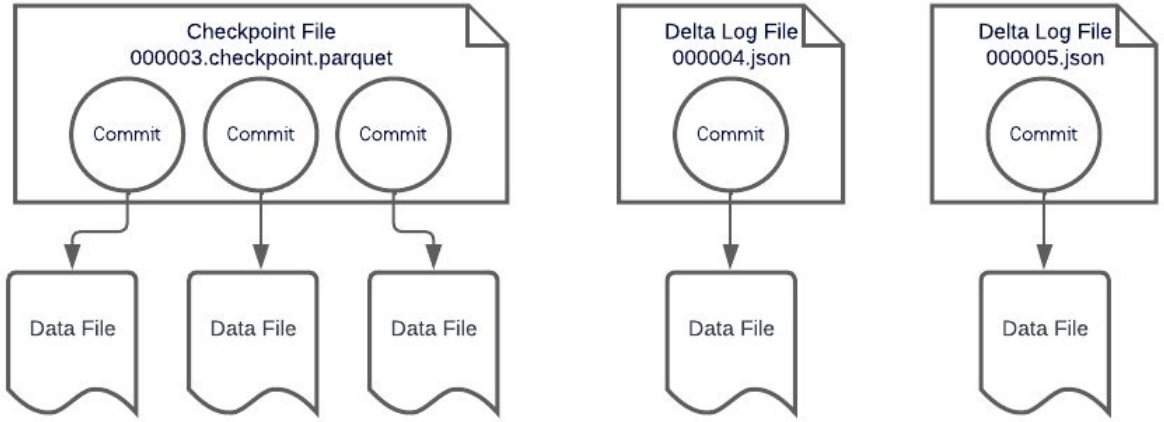
- **metadata files** that define the table
- **manifest lists** that define a snapshot of the table, with a list of manifests that make up the snapshot and metadata about their data
- **manifests** is a list of data files along with metadata on those data files for file pruning

Apache Hudi



- Apache Hudi's approach is to group all transactions into different types of actions that occur along a timeline.
- Directory-based approach with timestamped files and log files that track changes.
- An optional metadata table for additional file pruning.

Delta Lake



Delta Lake's approach is to track metadata in two types of files:

- Delta Logs sequentially track changes to the table.
- Checkpoints summarize all changes to the table up to that point.

In these files are indexes of columns used for file pruning

Feature Overview

Feature Overview

	ICEBERG	Apache hudi	DELTA LAKE
ACID Transactions	✔	✔	✔
Partition Evolution	✔	✘	✘
Schema Evolution (later chart with more detail)	✔	partial	partial
Time-travel	✔	✔	✔
File Format Support	Parquet, ORC, AVRO	Parquet, ORC	Parquet

Schema Evolution

Schema Evolution

	ICEBERG	Apache hudi	DELTA LAKE
Add Column	✔	✔	✔
Drop Column	✔	✔*	✘
Rename Column	✔	✔*	✔
Update Column	✔	✔*	✔
Reorder Column	✔	✔*	✔

** Only supported in Spark*

Partitioning

- Can evolve partitions without rewriting table
- Can specify column transforms (hour, day, month, year, truncate, bucket) when declaring partition columns
- Filtering by transformed columns will take advantage of partitioning (Hidden Partitioning)

- Cannot evolve partitions without rewriting table
- Can specify transforms in queries which will be used for data skipping based on column stats index
- You must filter by partitioned columns to take advantage of partitioning

- Cannot evolve partitions without rewriting table
- Can specify generated columns who are automatically calculated (like month, day, hour columns) then specify these columns as the partitioning columns
- If you filter by a generated partition columns source column, the additional partition filters will automatically be generated.

	ICEBERG	Apache hudi	DELTA LAKE
Change partitioning scheme without rewriting table	✔	✘	✘
Use transforms of existing columns to specify partitions	✔	✘	Can use transforms to define generated columns, then partition based on those columns.
Consumers don't need to understand the table's partitioning to benefit from it	✔	✘	✔
File Pruning	Min/Max filtering based on individual columns using metadata	Data Skipping using Column Stats Index used to prune files (can use transforms in filters)	Maintain Indexes on set number of columns used for Data Skipping

Query Engine Support

READ SUPPORT

	ICEBERG	Apache hudi	DELTA LAKE
Apache Flink	✅	✅	✅
Apache Beam	❌	❌	✅
Apache Drill	✅	❌	❌
Apache Hive	✅	✅	✅
Apache Impala	✅	✅	❌
Apache Spark	✅	✅	✅
Athena	✅	✅	✅
BigQuery	❌	✅	❌
Databricks Spark	✅	✅	✅
Databricks SQL Analytics	❌	❌	✅
Dremio Sonar	✅	❌	✅
Presto	✅	✅	✅
Redshift	❌	✅	✅
Snowflake	✅	❌	✅
Trino	✅	✅	✅
TOTAL	**11/15**	**10/15**	**12/15**

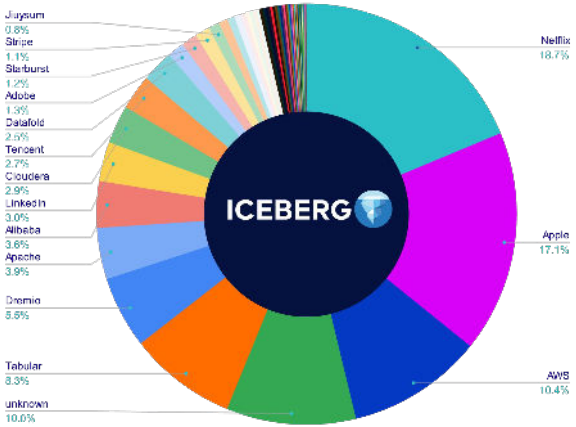
WRITE SUPPORT

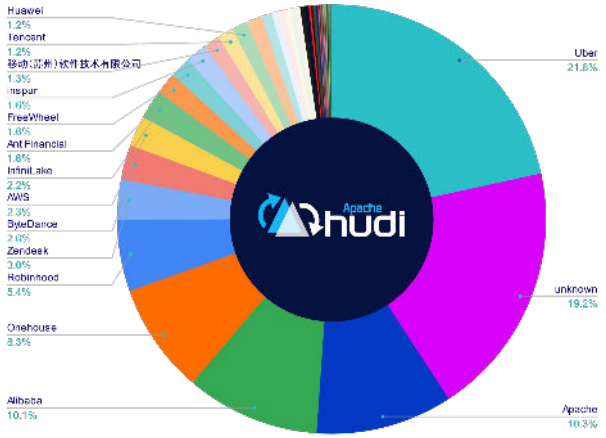
	ICEBERG	Apache hudi	DELTA LAKE
Apache Flink	✅	✅	✅
Apache Impala	✅	✅	❌
Apache Spark	✅	✅	✅
Athena	✅	❌	❌
Databricks Spark	✅	✅	✅
Databricks Photon	❌	❌	✅
Dremio Sonar	✅	❌	❌
Presto	✅	❌	❌
Trino	✅	❌	✅
TOTAL	**8/9**	**4/9**	**4/9**

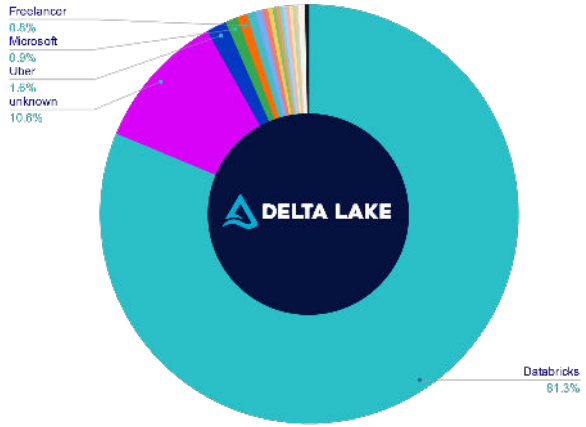
Project Community & Governance

	ICEBERG	Apache hudi	DELTA LAKE
Project Governance	Apache Project with a diverse PMC (top-level project)	Apache Project with a diverse PMC (top-level project)	Linux Foundation Project with an all-Databricks TSC
Community Contributions **(stats as of 3/28/22)**	240 contributors, 2241 merged PRs, 275 open PRs	252 contributors, 2880 merged PRs, 160 open PRs	145 contributors, 16 merged PRs, 43 open PRs

Diversity of Contributions by Company







Conclusion

Which is the right format for you?

- What tools do I use today and does the format allow me to read and write using that tool?
- Do I think I may change tools or introduce new tools in the future? How important is broad tool compatibility for me?
- How often will my schema evolve? Does this table format allow me to evolve my schema in the way I need?
- Might I need to evolve a table's partitioning at some point? If I do, do I want to avoid a rewrite of the table?
- Does the format enable an intuitive and easy to use SQL syntax for creating and querying tables to avoid unnecessary full table scans?
- Is there a large and diverse developer community behind the project to avoid vendor lock-in and a potential future lack of support for tools and use cases that don't suit dominant contributors?

Q&A