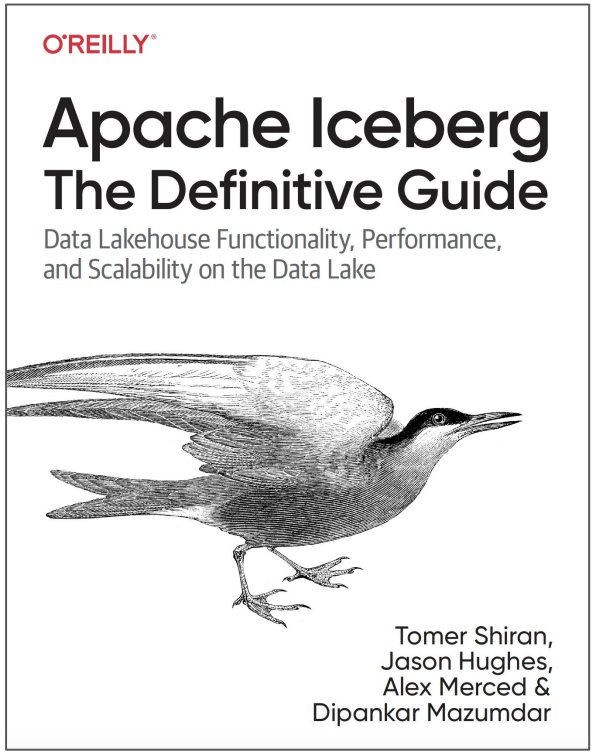
Apache Iceberg

An Architectural Look Under the Covers

Community Over Code, North America, 2023

About Me



- **Current:** Data (Eng/Sci) Advocate at Dremio
- **Open source:** Apache Iceberg, Arrow, Project Nessie
- **Past:** BI, Data Viz, ML

Key Takeaways

- Evolution of data architecture
- Data Lakehouse
- Architectural deep dive: Iceberg
- How queries work under covers?
- Design Benefits

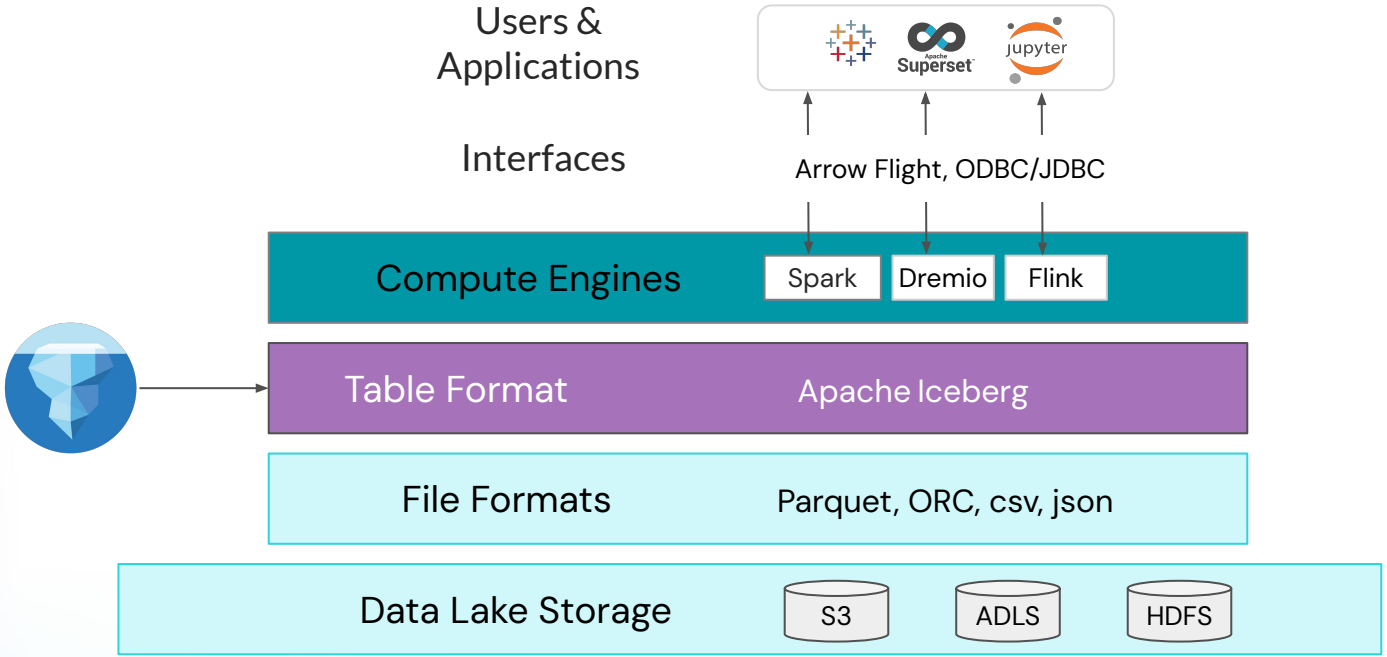
Evolution of Data Architecture

How did we get here?

Centralized, reliable data platform
Democratize data

Data Warehouse -> Data Lakes -> Lakehouse

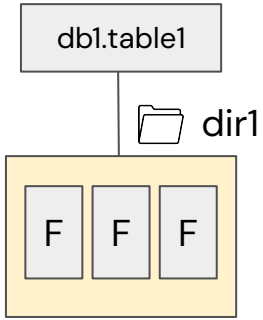
Iceberg in a Data Lakehouse



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?"

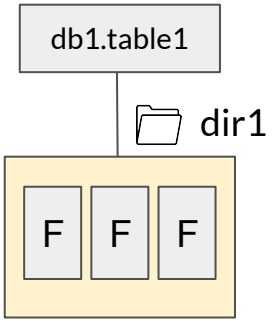
Old way (Hive)



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

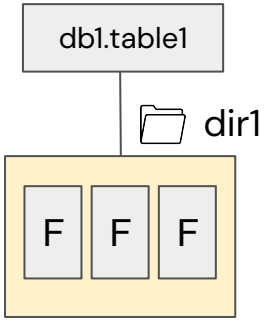
Cons

- 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?

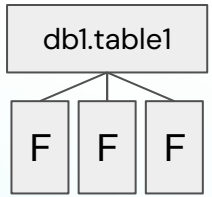
→ **We need a new table format**

Old way (Hive)



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

New way



A table is a **canonical list of files**

NETFLIX 's goals

- Table correctness/consistency
- Faster query planning and execution
- Allow users to not worry about the physical layout of the data
- Table evolution
- Accomplish all of these at scale

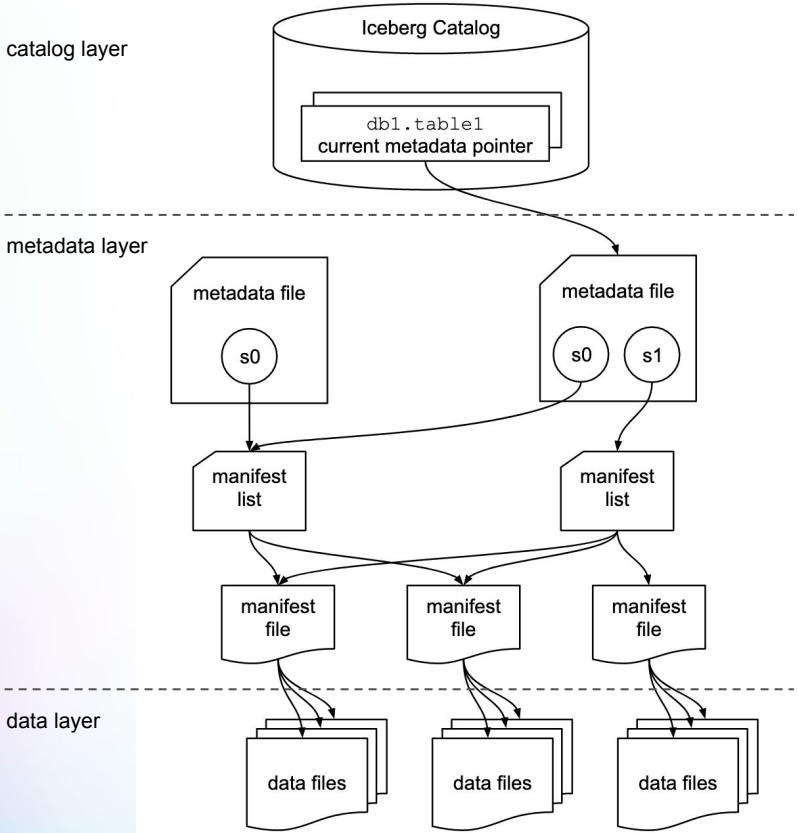
What Iceberg is and isn't

- Table format specification
- A set of APIs and libraries for interaction with that specification
  - These libraries are leveraged in other engines and tools that allow them to interact with Iceberg tables

- A storage engine
- An execution engine
- A service

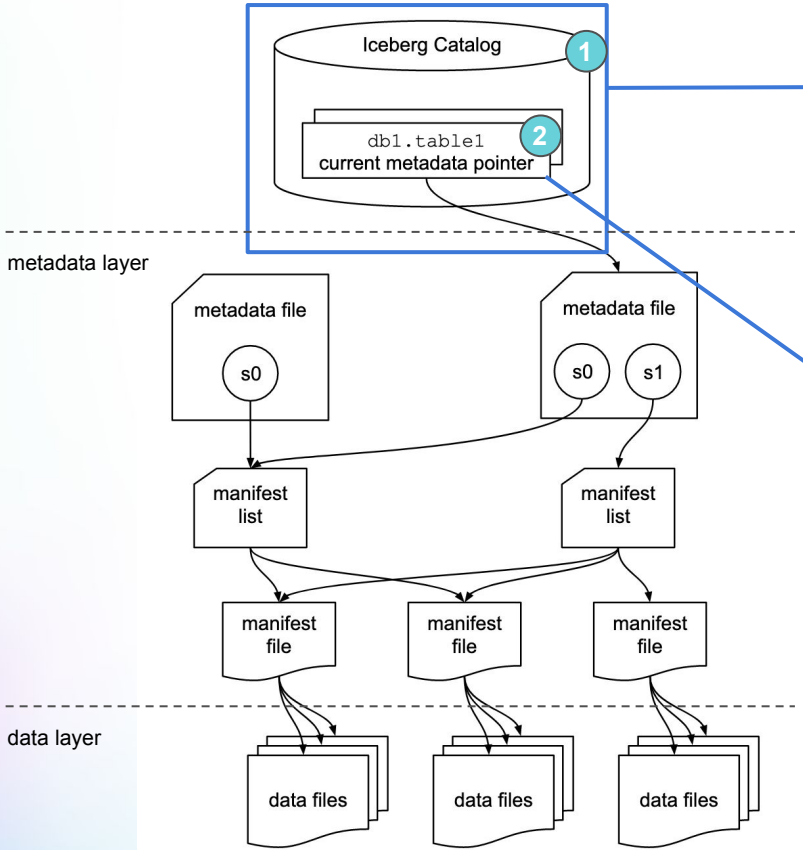
Architectural Deep Dive

Iceberg table format



- Overview of the components
- Summary of the read path (SELECT)

Iceberg components: **Catalog**



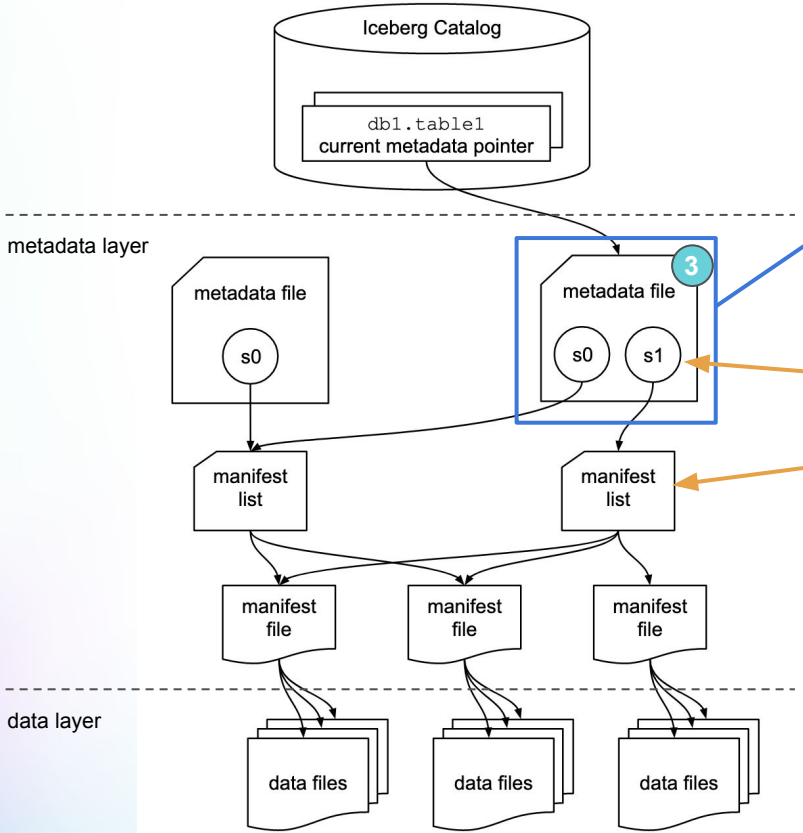
Iceberg Catalog

- A store that houses the current metadata pointer for Iceberg tables
- Must support atomic operations for updating the current metadata pointer (e.g. HDFS, HMS, Nessie)

table1's current metadata pointer

- Mapping of table name to the location of current metadata file

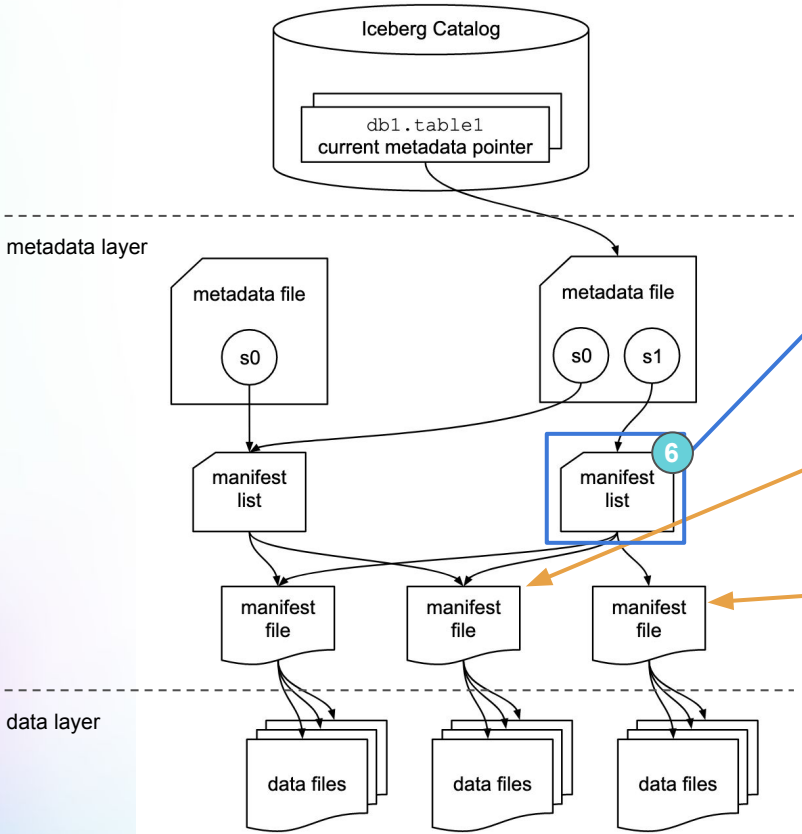
Iceberg components: **Metadata File**



Metadata file – stores metadata about a table at a certain point in time

```
{
  "table-uuid" : "<uuid>",
  "location" : "/path/to/table/dir",
  "schema": {...},
  "partition-spec": [ {<partition-details>}, ...],
4 "current-snapshot-id": <snapshot-id>,
  "snapshots": [ {
    "snapshot-id": <snapshot-id>
5   "manifest-list": "/path/to/manifest/list.avro"
  }, ...],
  ...
}
```

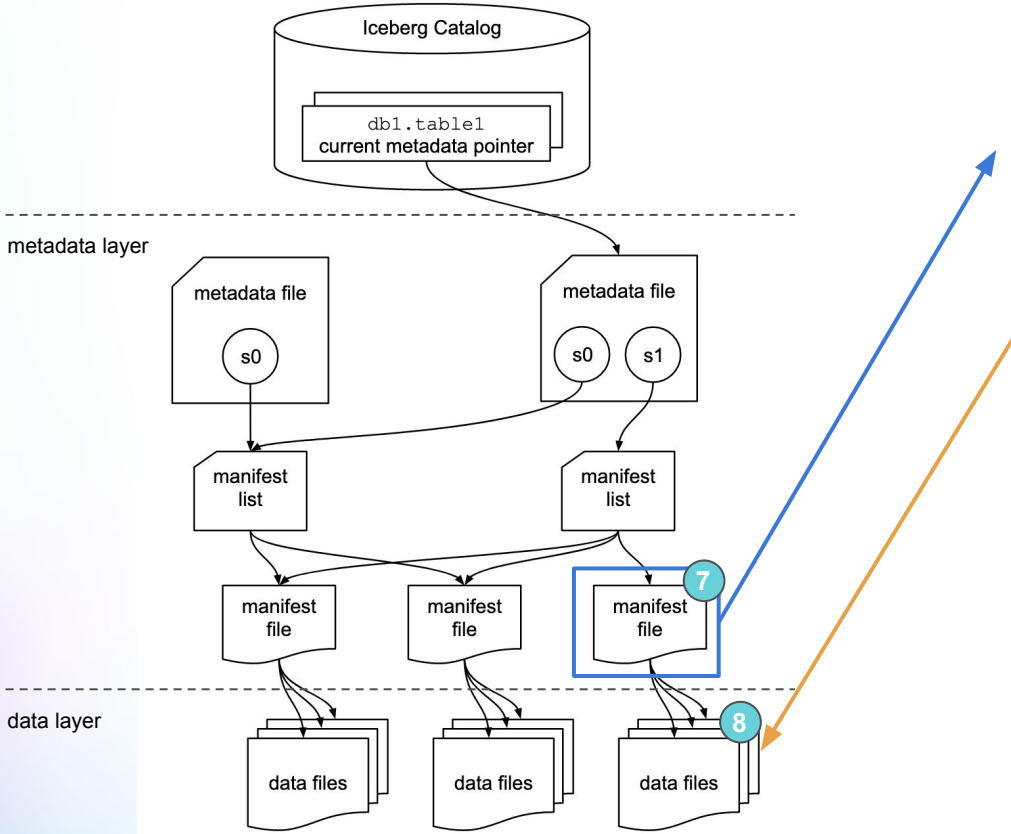
Iceberg components: **Manifest List**



Manifest list file – a list of manifest files

```
{
  "manifest-path" : "/path/to/manifest/file.avro",
  "added-snapshot-id": <snapshot-id>,
  "partition-spec-id": <partition-spec-id>,
  "partitions": [ {partition-info}, ...],
  ...
}
{
  "manifest-path" : "/path/to/manifest/file2.avro",
  "added-snapshot-id": <snapshot-id>,
  "partition-spec-id": <partition-spec-id>,
  "partitions": [ {partition-info}, ...],
  ...
}
```

Iceberg components: **Manifest file**



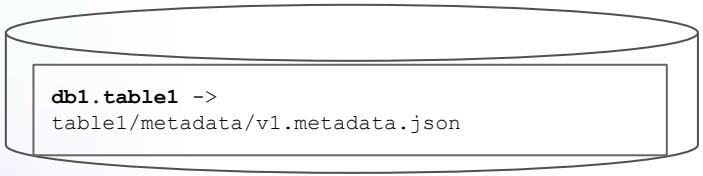
Manifest file – a list of data files, along with details and stats about each data file

```
{
  "data-file": {
    "file-path": "/path/to/data/file.parquet",
    "file-format": "PARQUET",
    "partition": {"<part-field>":{"<data-type>":<value>}},
    "record-count": <num-records>,
    "null-value-counts": [{
      "column-index": "1", "value": 4
    }, ...],
    "lower-bounds": [{
      "column-index": "1", "value": "aaa"
    }, ...],
    "upper-bounds": [{
      "column-index": "1", "value": "eee"
    }, ...],
  }
  ...
}
{
 ...
}
```

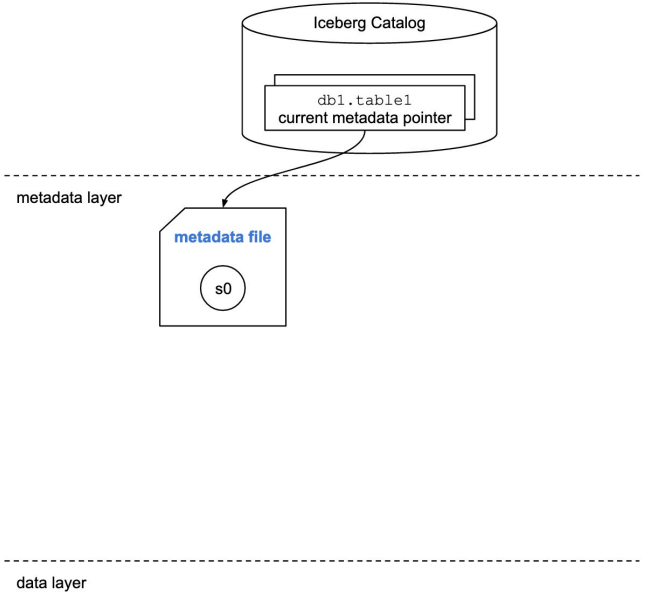
Let’s Look Under the Covers

CREATE TABLE

```
CREATE TABLE db1.table1 (
      order_id bigint,
      customer_id bigint,
      order_amount DECIMAL(10, 2),
      order_ts TIMESTAMP
)
USING iceberg
PARTITIONED BY ( hour(order_ts) );
```



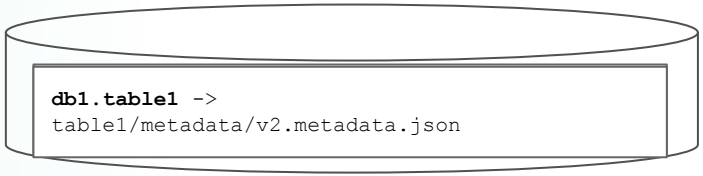

```
table1/
 |- metadata/
 |    |- v1.metadata.json
 |- data/
```



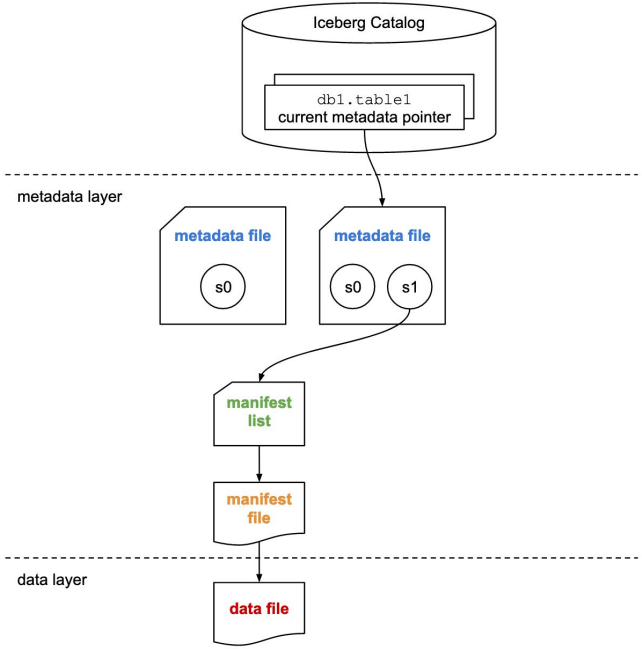
dremio

INSERT

```
INSERT INTO db1.table1 VALUES (
      123,
      456,
      36.17,
      '2021-01-26 08:10:23'
);
```



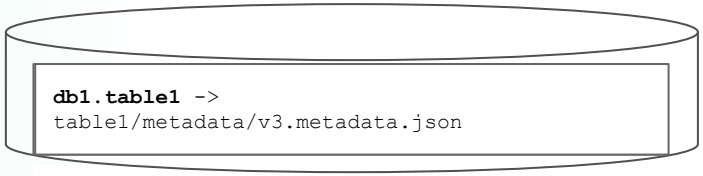

```
table1/
 |- metadata/
 |    |- v1.metadata.json
 |    |- v2.metadata.json
 |    |- snap-2938-1-4103.avro
 |    |- d8f9-ad19-4e.avro
 |- data/
 |    |- order_ts_hour=2021-01-26-08/
 |    |    |- 00000-5-cae2d.parquet
```



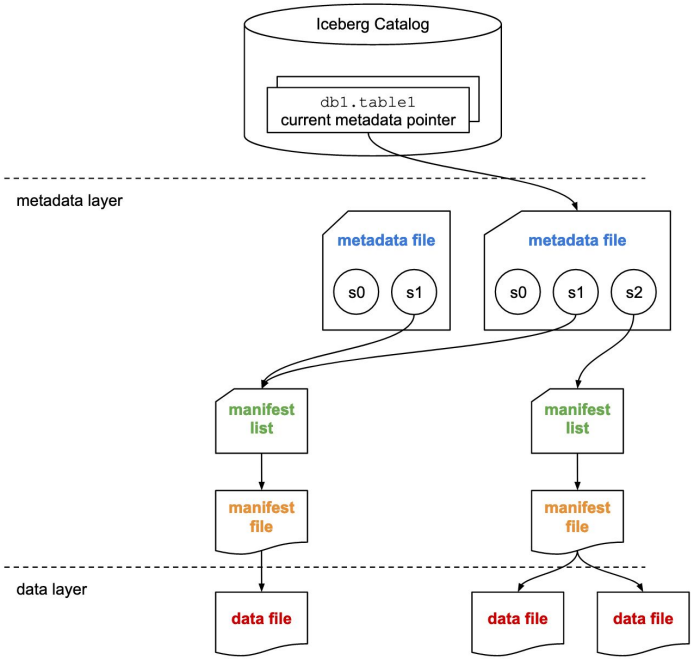
dremio

UPSERT

```
MERGE INTO db1.table1
USING ( SELECT * FROM table1_stage ) s
ON table1.order_id = s.order_id
WHEN MATCHED THEN UPDATE table1.order_amount = s.order_amount
WHEN NOT MATCHED THEN INSERT *
```



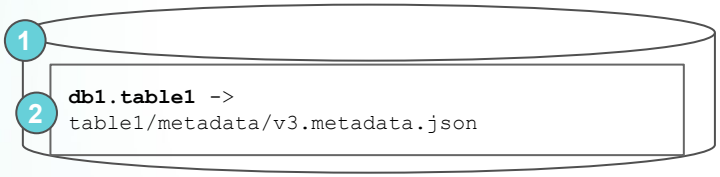

```
table1/
 |- metadata/
 |    |- v1.metadata.json
 |    |- v2.metadata.json
 |    |- v3.metadata.json
 |    |- snap-29c8-1-b103.avro
 |    |- snap-9fa1-3-16c3.avro
 |    |- d8f9-ad19-4e.avro
 |    |- 0d9a-98fa-77.avro
 |- data/
 |    |- order_ts_hour=2021-01-26-08/
 |    |    |- 00000-5-cae2d.parquet
 |    |    |- 00000-1-aef71.parquet
 |    |- order_ts_hour=2021-01-27-10/
 |    |    |- 00000-3-0fa3a.parquet
```



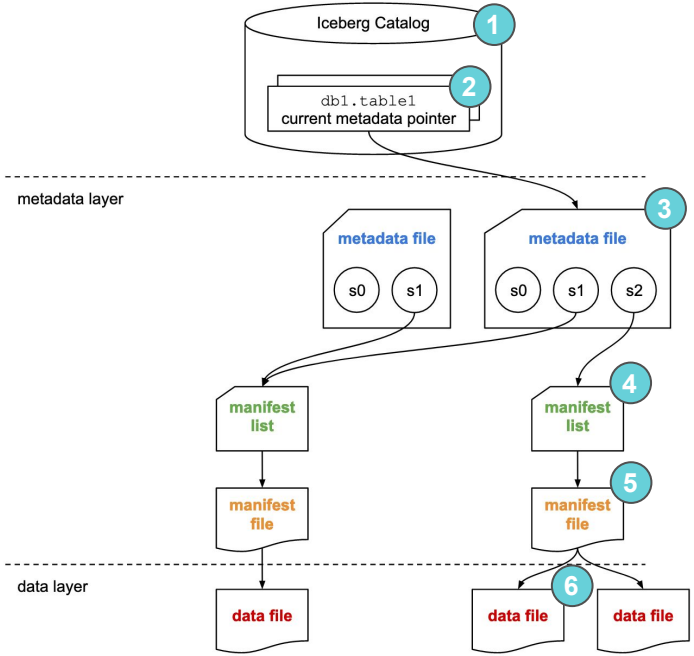
dremio

READ

```
SELECT *
FROM db1.table1
WHERE order_ts = DATE '2021-01-26'
```



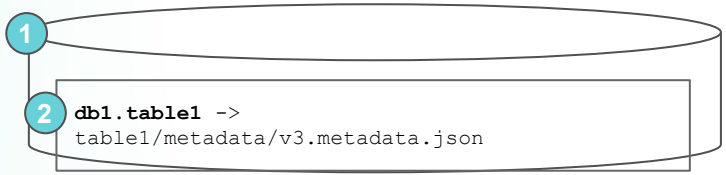

```
table1/
 |- metadata/
 |    |- v1.metadata.json
 |    |- v2.metadata.json
 |    |- v3.metadata.json
 |    |- snap-29c8-1-b103.avro
 |    |- snap-9fa1-3-16c3.avro
 |    |- d8f9-ad19-4e.avro
 |    |- 0d9a-98fa-77.avro
 |- data/
 |    |- order_ts_hour=2021-01-26-08/
 |    |    |- 00000-5-cae2d.parquet
 |    |    |- 00000-1-aef71.parquet
 |    |- order_ts_hour=2021-01-27-10/
 |    |    |- 00000-3-0fa3a.parquet
```



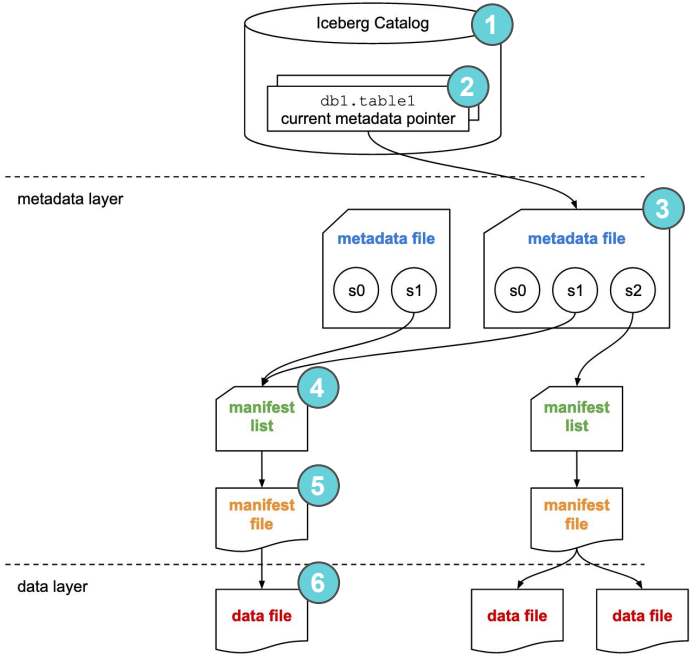
dremio

TIME TRAVEL

```
SELECT *
FROM db1.table1 AS OF '2021-05-26 09:30:00'
 -- (timestamp is from before MERGE INTO operation)
```




```
table1/
 |- metadata/
 |    |- v1.metadata.json
 |    |- v2.metadata.json
 |    |- v3.metadata.json
 |    |- snap-29c8-1-b103.avro
 |    |- snap-9fa1-3-16c3.avro
 |    |- d8f9-ad19-4e.avro
 |    |- 0d9a-98fa-77.avro
 |- data/
 |    |- order_ts_hour=2021-01-26-08/
 |    |    |- 00000-5-cae2d.parquet
 |    |    |- 00000-1-aef71.parquet
 |    |- order_ts_hour=2021-01-27-10/
 |    |    |- 00000-3-0fa3a.parquet
```



dremio

Case Study: Atlas

- Historical Atlas data:
  - Time-series metrics from Netflix runtime systems
  - 1 month: 2.7 million files in 2,688 partitions
  - **Problem: cannot process more than a few days of data**

- Sample query:

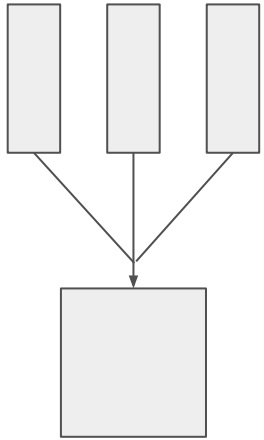
```
select distinct tags['type'] as type
from iceberg.atlas
where
  name = 'metric-name' and
  date > 20180222 and date <= 20180228
order by type;
```

Case Study: Atlas Performance

- Hive table – with Parquet filters:
  - 400k+ splits per day, not combined
  - EXPLAIN query: **9.6 min** (planning wall time)
- Iceberg table – partition data filtering:
  - 15,218 splits, combined
  - **13 min** (wall time) / 61.5 hr (task time) / **10 sec** (planning)
- Iceberg table – partition and min/max filtering:
  - 412 splits
  - **42 sec** (wall time) / 22 min (task time) / 25 sec (planning)

Enabled by this design: Compaction

- Asynchronously compact small files into fewer larger files
- It being asynchronous helps balance the write-side and read-side trade-offs
- Input and output of compaction jobs can be different file types
  - E.g. avro from streaming writes, compacted into larger parquet files for analytics
- Scheduling/triggering and the actual compaction work is done by external tools
  - Scheduling/triggering: scheduler, workflow tool, etc.
  - Compaction work execution: processing engine (e.g. Spark, Dremio)



Design Benefits

- **Efficiently make smaller updates**
  - Make changes at the file level
- **Snapshot isolation for transactions**
  - Reads and writes don't interfere with each other and all writes are atomic
  - Concurrent writes
- **Faster planning and execution**
  - List of files defined on the write-side
  - Column stats in manifest files used to eliminate files
- **Reliable metrics for CBOs (vs hive..)**
  - Done on write instead of "infrequent" expensive read job
- **Abstract the physical, expose a logical view**
  - Hidden partitioning
  - Compaction
  - Tables can change over time
  - DE can transparently experiment with table layout
- **Rich schema evolution support**
- **All engines see changes immediately**

Additional Resources

- iceberg.apache.org
- iceberg.apache.org/blogs/
- dremio.com/subsurface/apache-iceberg/
- Get hands on - iceberg.apache.org/getting-started/

Thank You

Presenter: Dipankar

@dipankartnt