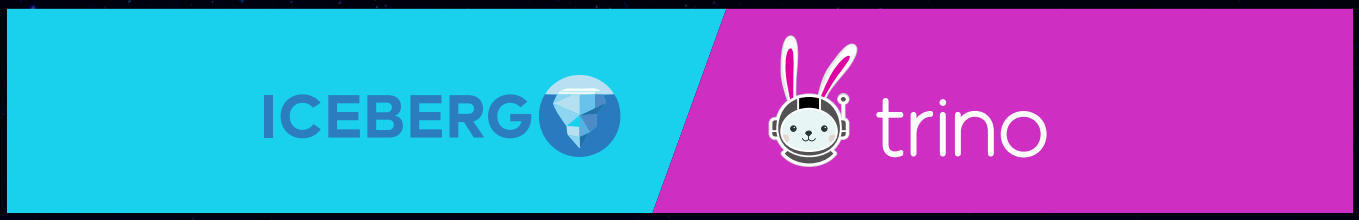
Starburst

E-book

How to optimize your data lake with Iceberg and Trino



Overview

This whitepaper offers an analysis of the key requirements of an effective modern data lake, why the Apache Iceberg table format and Trino processing engine are ideal choices for data-driven organizations, and how to optimize and tune your data lake architecture to deliver the most value to end users and the business as a whole

Contents

What makes a data lake modern?	2
The new center of data gravity	3
Iceberg and the modern data lake	4
How to optimize Iceberg and Trino	6
Starburst Galaxy and Iceberg	17

What makes a data lake modern?

As the data lake devolved into a swamp, the properties of what is now called a modern data lake began to take shape. A modern data lake must be built on commodity storage & compute, along with open file and table formats such as those from Apache Iceberg, Azure Delta Lake, and Apache Hudi. The data in the lake should also be easily accessible to a high-performance, scalable query engine that can interact with all of your organization's data, including volumes residing outside the lake. A modern data lake must ingest, process, catalog, integrate, and store your data at scale, and it must do so cost effectively. Additionally, the lake should:

- **Manage metadata**
- **Strengthen security and governance**
- **Enable analytics and exploration**
- **Maintain data quality**

Finally, a modern data lake should deliver clear benefits to multiple groups within your organization, from analysts and data scientists to line-of-business operators and C-level executives. An effectively architected lake provides:

- **A single point of access for analytics and governance**
- **Advanced warehouse-like capabilities**
- **Vendor-agnostic architecture**
- **Predictable cost savings**
- **Unlimited scale**

Architecting a modern data lake that satisfies the above requirements and delivers these and other benefits to the business is not as straightforward as many vendors would have you believe. The purpose of this white paper is to detail in clear terms why you should optimize your modern data lake with Iceberg and Trino, and how to do so efficiently.

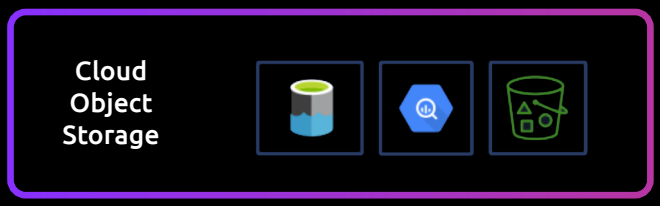
The new center of data gravity

The concept of data gravity arose more than a decade ago to describe how data and the services that needed access to this data tended to agglomerate around core locations with physical infrastructure. Today, however, data lakes are the new center of gravity for modern, data-driven enterprises. The question these organizations face is how to optimize data analytics and drive maximal value out of their modern data lake.

First, an organization needs to make four key decisions:

1. Cloud provider

The largest providers of cloud data lake storage services today are Amazon S3, Microsoft Azure Blob Storage or Azure Data Lake, and Google Cloud Storage. Each offers cost-effective scale, separation of storage and compute, and an evolving suite of additional AI and ML tools.

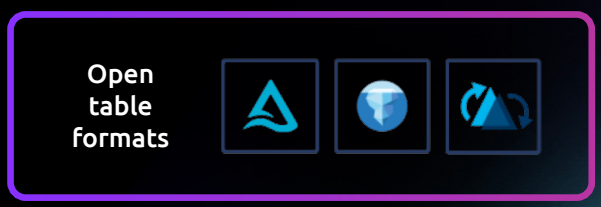


2. Data format

A traditional or cloud data warehouse converts your data into a proprietary format that subjects you to vendor lock-in and limits your flexibility going forward. A modern data lake makes use of open file formats like Parquet, ORC, and Avro that allow you to maintain optionality and ownership.

3. Table format

Another critical architectural choice is your table format. The leading formats today are Iceberg, Delta Lake, and Hudi – we will detail why we prefer Iceberg below.



4. Analytics engine

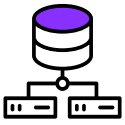
Finally, you will need a high-performance, scalable processing engine optimized for the modern data lake. Since any large organization will still have data residing outside the lake, this processing engine should be able to query the lake and multiple additional data sources simultaneously and in a high-performance way.

Although we are agnostic with regards to both cloud provider and data format, we highly recommend the combination of the Apache Iceberg table format and Trino as the optimal architecture for modern, data-driven organizations.

Iceberg and the modern data lake

Originally created by Netflix, the Iceberg table format provides database-type functionality on top of cloud object stores. With Iceberg, organizations can build a true data lakehouse with an open architecture that avoids vendor and technology lock-in. Iceberg is an open source table format and both open source and proprietary engines have been rushing to support it. This kind of excitement and effort is a clear indicator of a technology's value and outlook. A few of the basic benefits:

Choice: You can pick from any one of a growing number of engines that support Iceberg and select the one that best fits your needs.



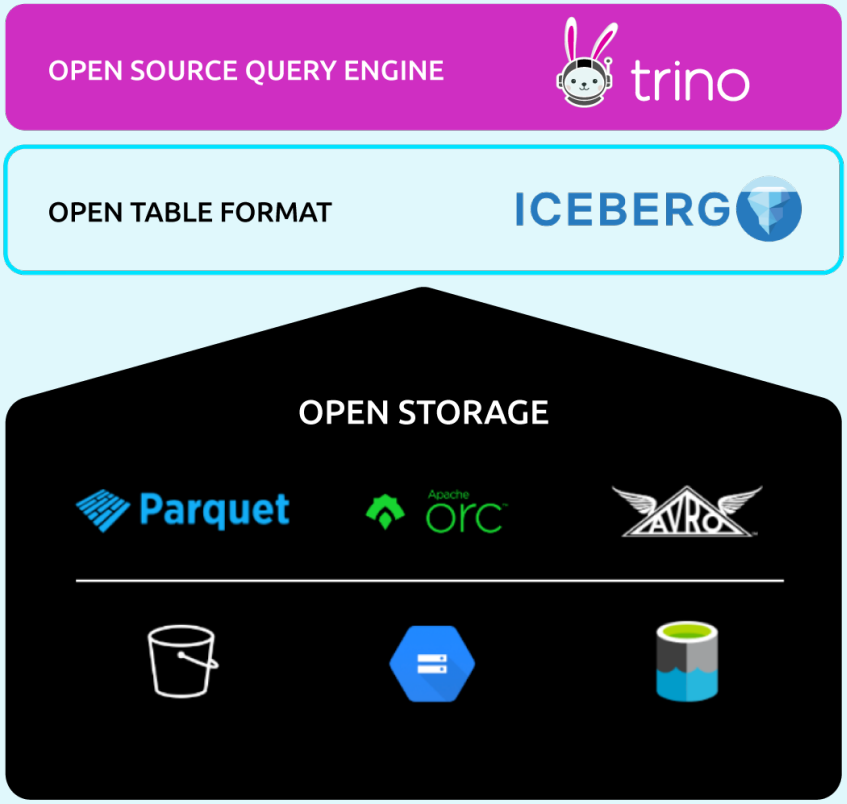
Optionality: The data in Iceberg remains your data in your account, and Iceberg metadata is always available to all engines.

DML: Data can easily be modified to meet compliance requirements such as GDPR and satisfy other regulator use cases.

Database Qualities: Iceberg supports full schema evolution and delivers fast performance and reduced cloud object store retrieval costs; plus, end users query Iceberg tables like they would a database.

Technically, Iceberg is a layer of metadata that sits over your object storage and provides a transaction log for each table. This log keeps track of the current state of the table and maintains an updated snapshot of the files that belong to the table. This reduces the amount of data that needs to be ready during a query, thereby accelerating performance.

The Trino processing engine is an ideal engine for this exciting new table format because it provides full support for Iceberg features and it is very easy to deploy and use. At Starburst, we have made Iceberg the default table format in Starburst Galaxy, our fully managed Trino platform.



In the next section, we will review how to optimize an Iceberg / Trino deployment to drive the most value across your organization, using Starburst Galaxy.

How to optimize Iceberg and Trino

Efficient partitioning

The difference between a query that takes minutes or one that requires hours may depend on partitioning, or narrowing down the scope of the data that needs to be read for a query. The specifics of how data is partitioned is critical. This was one of the drawbacks of Hive-based tables.

For example, most tables you'd want to partition have a column with a timestamp (day, month, year) indicating when each row of data was created. If you wanted to partition by day on a Hive table, you'd have to break out the associated column into three separate columns – one each for day, month, and year. Then you'd have to teach your users to include these columns in their queries.

```
CREATE TABLE
  orders (
    event_id INTEGER,
    created_ts TIMESTAMP,
    metric INTEGER,
    year VARCHAR,
    month VARCHAR,
    day VARCHAR
  );
```

Iceberg allows you to partition on that original column, which means end users can query the table just like they would in a database. Plus, Trino is smart enough to only look at files that meet the partition requirement of the query, and it takes into account time zones – another data type Hive didn't support.

```
-- create iceberg table partitioned by day on the created_ts column
CREATE TABLE
  orders_iceberg (
    event_id INTEGER,
    created_ts TIMESTAMP(6),
    metric INTEGER
  )
WITH
  (
    TYPE = 'iceberg',
    partitioning = ARRAY['day(created_ts)']
  );
-- insert rows
INSERT INTO orders_iceberg VALUES (1,timestamp '2022-09-10 10:45:38.527000',5.5);
INSERT INTO orders_iceberg VALUES (1,timestamp '2022-09-11 03:12:23.522000',5.5);
INSERT INTO orders_iceberg VALUES (1,timestamp '2022-09-12 10:46:13.516000',5.5);
INSERT INTO orders_iceberg VALUES (1,timestamp '2022-09-13 04:34:05.577000',5.5);
INSERT INTO orders_iceberg VALUES (1,timestamp '2022-09-14 09:10:23.517000',5.5);
-- query the table only looking for certain days
SELECT * FROM orders_iceberg
WHERE created_ts BETWEEN date '2022-09-10' AND date  '2022-09-12';
```

Performance optimizations

Since Iceberg stores metadata on all the files that belong to a table for a given snapshot, along with associated statistics, this table format can be highly performant.

1. Partitioning reduces the number of files that need to be read and a feature known as "file skipping" further narrows that count. Making use of these features in conjunction with Trino accelerates queries simply because less data needs to be read to generate results.
2. Constant ingestion of new data can slow performance by forcing a system to read too many small files, but Iceberg includes a feature that allows you to scan for and combine any files under 100MB into larger ones to maximize query performance. This is highly recommended, as you want to ingest and make this data available for queries as soon as possible, without reducing overall performance.
3. Two different cleanup operations – one removes snapshots older than 7 days, the other older files – further optimize performance. It's a good idea to utilize these tools to keep performance up to par.

To scan the table for small files and make them larger, you simply issue the following command:

```
ALTER TABLE <table> EXECUTE optimize;
```

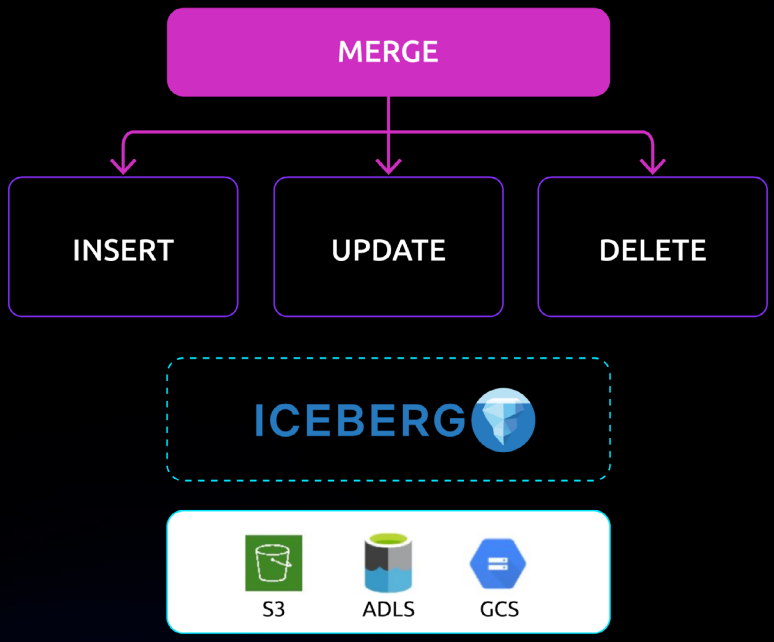
This will look for any files under 100MB and combine them into larger ones. You can also choose the file size if 100MB:

```
ALTER TABLE <table> EXECUTE optimize(file_size_threshold => '10MB');
```

If your Iceberg table becomes very large and the optimize command above is taking too long to run, you can just optimize the files that have arrived recently:

```
ALTER TABLE <table> EXECUTE optimize WHERE "$file_modified_time" > current_date - interval '1' day;
```

This will look for files that have arrived since yesterday and optimize them. On a very active table where lots of changes are taking place, this will greatly reduce the amount of time the optimize command takes. By optimizing these and other features, you will see a measurable impact on performance.



Data is stored in AWS S3 and other clouds as immutable objects that cannot be modified. Using the Iceberg connector, Trino allows full DML, or data manipulation language. This includes:

Insert: Data is constantly added to the lake, and Trino's Iceberg connector supports a standard insert statement. To eliminate conflicts, update, deletes, and merges should be run in serial and/or batch against a single table.

```
INSERT INTO
  customer_iceberg
VALUES
  (
    90000,
    'Testing',
    '33 Main',
    3,
    '303-867-5309',
    323,
    'MACHINERY',
    'Testing Iceberg'
  );
```

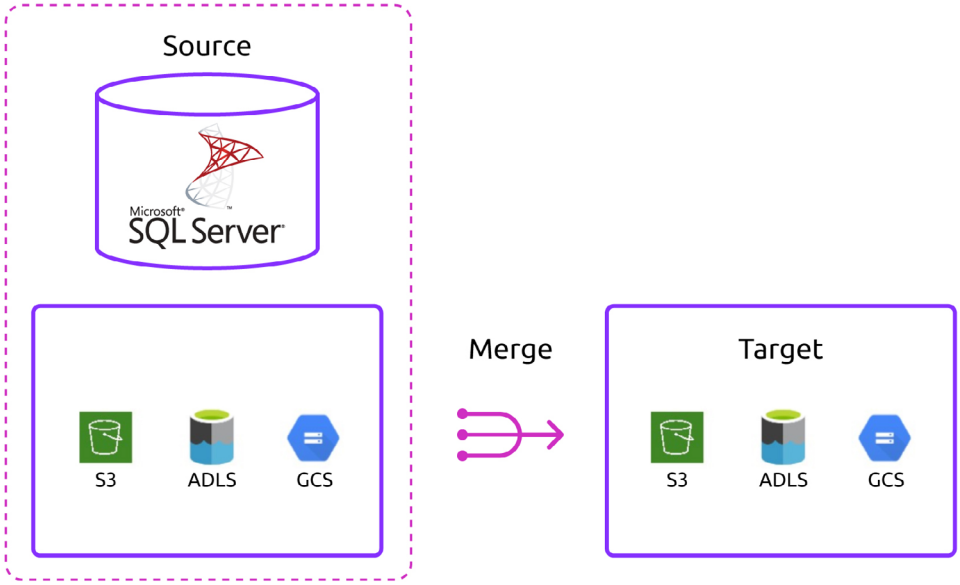
Update: With Trino and the Iceberg connector, updates act just like ordinary updates, and preserve read integrity of current select statements executing against a table.

```
UPDATE customer_iceberg SET name = 'Tim Rogers' WHERE custkey = 2732;
```

Delete: Typically rows are updated to be flagged as deleted through a status column or something similar, but a row or set of rows can be deleted through a simple statement. Yet this doesn't physically delete the data off storage; a separate procedure expire_snapshots can be executed to ensure that.

```
DELETE FROM customer_iceberg WHERE custkey = 2732;
```

Merge: These statements are used to add logic-based operations to SQL statements and are often used when you have new or modified data staged in a table first. Trino makes it easy to insert rows, update existing ones, and generally allows you to leverage data warehousing techniques that have not been available with data lakes. By providing these data warehousing features, merge is a true game-changer for the data lake.



To use merge, you can either stage data that needs to be inserted or updated into your target table or you can use data directly from the source table(s).

Examples

#1 – If there are rows that don't exist in the target table, insert them. This is a very basic merge statement. The customer_land table below could be a staged table in object storage like S3 or it could be from a source system such as MySQL or SQL Server:

```
MERGE INTO s3lakehouse.blog.customer_base AS b
USING (SELECT * FROM s3lakehouse.blog.customer_land) AS l
ON (b.custkey = l.custkey)
WHEN NOT MATCHED
      THEN INSERT (custkey, name, state, zip, cust_since,last_update_dt)
            VALUES(l.custkey, l.name, l.state, l.zip, l.cust_since,l.last_update_dt);
```

#2 – With merge, we can issue a single statement to insert new rows and update existing ones:

```
MERGE INTO s3lakehouse.blog.customer_base AS b
USING s3lakehouse.blog.customer_land AS l
ON (b.custkey = l.custkey)
WHEN MATCHED and b.name != l.name
THEN UPDATE
SET name = l.name ,
          state = l.state,
          zip = l.zip,
          cust_since = l.cust_since
WHEN NOT MATCHED
      THEN INSERT (custkey, name, state, zip, cust_since,last_update_dt)
            VALUES(l.custkey, l.name, l.state, l.zip, l.cust_since,l.last_update_dt);
```

This statement inserts new rows where the custkey doesn't exist in the target table. It will update rows in the target table if the custkey matches and the name has changed. In real-world situations, there will be numerous columns that are checked to see if they have changed to issue an update. For this simple example, you can see the power of merge and why it's a game changer for a data lake.

#3 – Slowly Changing Dimension (SCD Type 2)

```
MERGE INTO s3lakehouse.blog.customer_base AS b
USING
( SELECT null AS custkey_match, custkey, name, state, zip, cust_since, last_update_dt,'Y' AS active_ind,current_timestamp AS end_dt
FROM s3lakehouse.blog.customer_land
UNION ALL
SELECT
custkey AS custkey_match,custkey, name, state, zip, cust_since, last_update_dt,active_ind,end_dt
FROM s3lakehouse.blog.customer_base
WHERE custkey IN
(SELECT custkey FROM s3lakehouse.blog.customer_land WHERE active_ind = 'Y')
) AS scdChangeRows
ON (b.custkey = scdChangeRows.custkey and b.custkey = scdChangeRows.custkey_match)
WHEN MATCHED and b.active_ind = 'Y' THEN
UPDATE SET end_dt = current_timestamp,active_ind = 'N'
WHEN NOT MATCHED THEN
        INSERT (custkey, name, state, zip, cust_since,last_update_dt,active_ind,end_dt)
            VALUES(scdChangeRows.custkey, scdChangeRows.name, scdChangeRows.state, scdChangeRows.zip,
                    scdChangeRows.cust_since,scdChangeRows.last_update_dt,'Y',null);
```

Optimize: As your Iceberg tables grow, the optimize command allows you to make small files larger to improve performance and clean up metadata, both of which improve queries. See page 6 for details on Iceberg optimization.

This support for database-type updates/deletes/merges and warehouse-like features is one of the most powerful advantages of using Iceberg with Trino as its query engine.

Schema evolution

As business rules and source systems change over time, tables need to be modified accordingly. Trino's Iceberg connector supports schema evolution by allowing for various modifications, including:

Renaming a table

```
ALTER TABLE customer_iceberg RENAME TO customer_iceberg_new;
```

Renaming a column

```
ALTER TABLE customer_iceberg RENAME COLUMN address TO fulladdress;
```

Adding a column

```
ALTER TABLE customer_iceberg ADD COLUMN tier VARCHAR(1);
```

Modifying partition columns

Oftentimes a table is initially partitioned by a column or set of columns only later it's discovered this may not be optimal. With Iceberg, you can modify the partition columns at any time.

For example, initially this table is partitioned by month:

```
CREATE TABLE orders_iceberg
WITH (partitioning = ARRAY['month(orderdate)']) AS
SELECT * FROM tpch.sf1.orders;
```

After reviewing query patterns, it's determined that partitioning by day would perform better as a majority of queries are filtered by certain days. A simple alter table statement as seen below will modify the partitioning on this table from month to day:

```
ALTER TABLE orders_iceberg SET PROPERTIES partitioning = ARRAY['day(orderdate)'];
```

After new data is inserted into the table, you will see a change in the data directory where the table data is stored:

Notice the orderdate_month is now orderdate_day. Note that queries that filter by day will partition prune at the partition day level, but the existing monthly partitions will still need to be searched. If you would like to have the entire table partitioned by day then you could recreate the table using a CTAS (create table as) from the existing table.

Example to create a new table partitioned by day from the existing table:

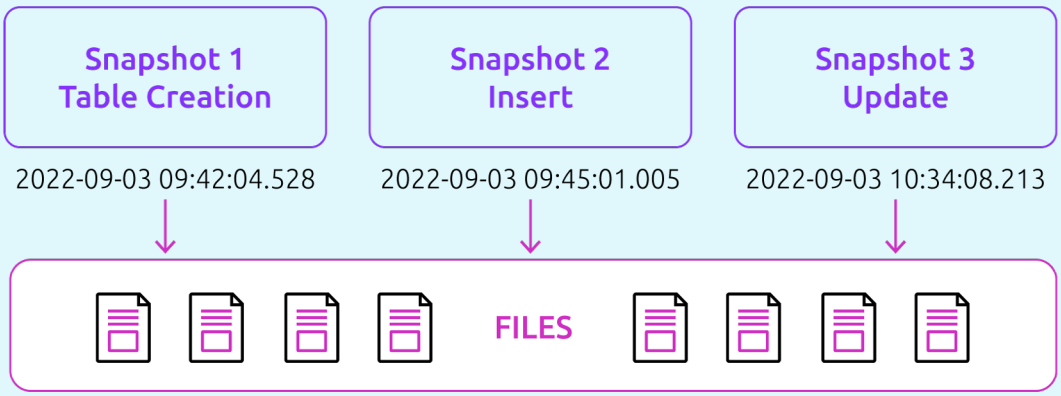
```
CREATE TABLE orders_iceberg_new
WITH (partitioning = ARRAY['day(orderdate)']) AS
SELECT * FROM orders_iceberg;
```

These types of capabilities are not available in Hive. The difference with Iceberg is that it's a table format with associated metadata, so modifying the table is relatively trivial. Schema evolution in Trino's Iceberg connector is both powerful and easy to use, and database veterans will be pleased to see these functions added to the modern data lake.

Time travel & rollbacks

Trino adds a feature that allows you to look back in time at a table's history through snapshots. Each table has a handy metadata table, and you can use this to see the snapshots on a table, view what type of operation was performed and when it was executed, and specify a timeframe to retrieve an older snapshot. This also allows you to roll back a table to a previous snapshot if a row was accidentally deleted or updated, assuming the snapshot hasn't been cleaned up and deleted for performance reasons.

In the diagram below, a new snapshot is created for the table creation, insert and update.



To see the snapshots on a table, you can use the metadata table that exists for each table:

```
SELECT * FROM "customer_iceberg$snapshots";
```

committed_at	snapshot_id	parent_id	operation
2022-09-18 07:18:09.002...	5043425904354141100	NULL	append
2022-09-18 07:18:16.394 ...	7162059766425589400	5043425904354141100	append
2022-09-18 07:18:25.475 ...	3117754680069542695	7162059766425589400	overwrite

The above snapshot table shows the create, insert and update operations on the customer_iceberg table. You can see what type of operation was performed and when it was executed.

To select a certain snapshot, you use the "for version as of" syntax. In the following two examples, we show the customer name before and after an update:

```
SELECT custkey,name
FROM customer_iceberg FOR VERSION
AS OF 5043425904354141100 WHERE custkey = 2732;
```

custkey	name
2732	Customer#000002732

```
SELECT custkey,name
FROM customer_iceberg FOR VERSION
AS OF 5043425904354141100 WHERE custkey = 2732;
```

custkey	name
2732	Testing2

You can also specify a timeframe to retrieve an older snapshot of a table. For example, the following query brings back the data for the first snapshot on or before a given timestamp:

```
SELECT custkey,name
FROM customer_iceberg FOR TIMESTAMP
AS OF TIMESTAMP '2022-09-18 07:18:09.002 America/New_York' where custkey = 2732;
```

custkey	name
2732	Customer#000002732

Additionally, with Iceberg you gain the ability to rollback a table to a previous snapshot.

Sometimes this is used when a row was accidentally deleted or updated. As long as the snapshot exists, (it hasn't been cleaned up yet) then you can roll back to any existing snapshot.

For example, in the scenario above, if I wanted to roll back to the state of the table before the update on the customer, then I would issue the following command:

```
CALL iceberg.system.rollback_to_snapshot('demo_tpch', 'customer_iceberg', 5043425904354141100);
```

Then we can query the table again to see the customer's name was "rolled back" to the previous version before the update:

```
SELECT custkey,name
FROM customer_iceberg
WHERE custkey = 2732;
```

custkey	name
2732	Customer#000002732

Once again, time travel and rollbacks are database-type functions that were not available in data lake deployments until now.

Sorted tables

The ability to sort Iceberg tables provides huge processing performance increases and cost savings. Sorting tables reduces the number of files that need to be read. At a scale of 1000s or even millions of files, this leads to massive reductions in query times. And since you reduce the amount of data that needs to be read, you also reduce the associated object store costs. This applies to both simple and complex queries, and leads to some of the biggest performance gains that have turned up in the data space in recent years.

Below are two tables that show a very simple example of how sorted tables work.

Not sorted:

File Name	Column Name	Min	Max
1	custkey	54	432029
2	custkey	2329	145292
3	custkey	154212	410918

sorted_by=ARRAY['custkey']:

File Name	Column Name	Min	Max
1	custkey	5	12321
2	custkey	23229	112019
3	custkey	113219	420001

Now, the following query was issued:

```
SELECT * FROM customer_iceberg WHERE custkey = 111029;
```

First the "not sorted" table, the min and max are checked in the Iceberg manifest files and since they are not sorted and the custkey we're looking for is 111029, all 3 files need to be read as this value could be found in these files.

For the sorted table, only file 2 would need to be read.

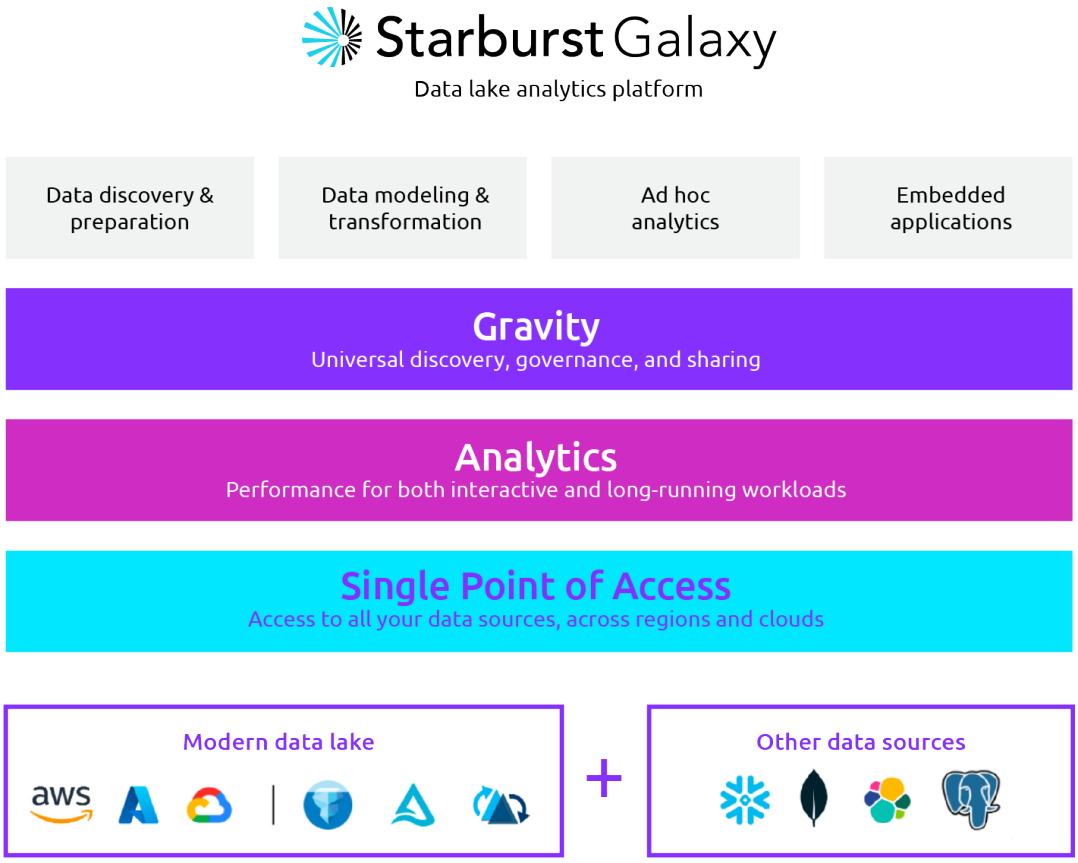
Starburst Galaxy & Iceberg

Architecting a modern data lake around Iceberg tables and Trino extends many of the classic benefits of the data warehouse to the modern data lake while also leveraging this new medium for additional gains. Designing, implementing, and maintaining this architecture can be time consuming if you rely on an open source processing engine like Trino. The alternative is to deploy Starburst Galaxy, the fully managed data lake analytics platform and enterprise-grade version of Trino. Designed by the original team behind Trino, Starburst Galaxy is the ideal analytics engine for the modern data lake.

Fully managed platform

Galaxy is designed to be the easiest and fastest way for organizations to start running queries at interactive speeds using familiar business intelligence and analytics tools. The modern data lake might be the new center of gravity, but large organizations still maintain critical data in at least one other system, and Galaxy enables high-performance queries of distributed datasets, ensuring that you can run analytics on all your data - inside and outside the lake.

The platform takes minutes to set up, and does all the work around designing, provisioning, maintaining, and securing your Trino infrastructure. A long list of proprietary features, including fully managed connectors to various data sources, is included.



Migrating Hive to Iceberg

Apache Hive was a popular choice for storing and processing large amounts of data in Hadoop environments, and it can be used to query any type of cloud object storage, but query latencies can be high. Iceberg is a superior table format for modern data lake architectures, as it allows for faster queries, more efficient data processing, and other added features, relative to Hive. (Unlike Iceberg, there's no support for time travel, schema evolution, or ACID transactions in Hive, for example.)

Recognizing the superiority of Iceberg, Galaxy makes it easy to migrate a Hive table to an Iceberg table. You can run a simple four-step process in Galaxy and quickly begin leveraging the benefits.

1. **Create a S3 Catalog in Starburst Galaxy**
2. **Connect your S3 catalog to a cluster**
3. **Run schema discovery**
4. **Choose your migration method**

Upgrading via the shadow migration process:

This approach creates a second Iceberg table off of the original Hive table. By leveraging a "shadow" process, we are afforded the following benefits:

- The schema and partition modifications are communicated in advance, enabling better management of data files.
- By conducting regular audits, validation, and counts of the data during the migration process, the likelihood of copying over corrupt data decreases. This is because you can clean any imperfect data present in the old table and ensure it does not corrupt the new table.

Upgrading via the in-place method:

The in-place data migration avoids rewriting the data. Instead, you write new Apache Iceberg tables that comprise the already existing files in your S3 instance. By leveraging an "in-place" process, we are afforded the following benefits:

- Data lineage is preserved, as the metadata is preserved.
- This process can be less time-consuming as all data does not need to be restated, in addition to avoiding data-duplication issues.
- If any type of error arises during the metadata writing process, you only need to re-write the metadata, not the underlying data.

	Hive Tables	Iceberg Tables
Open Source	Yes	Yes
Read object storage using SQL	Yes	Yes
File format	Parquet, Orc, Avro	Parquet, Orc, Avro
Performant at scale	Yes	Yes
ACID transactions	No	Yes
Table versioning	No	Yes
Time travel	No	Yes
Schema evolution	No	Yes
Partition evolution	No	Yes
Partition pruning	Yes	Yes

Optimizing data lakes is essential for data-driven organizations seeking to unlock the full potential of their valuable data assets. By leveraging the power of Apache Iceberg table format and Trino, businesses can build modern data lake architectures that deliver superior performance, cost-effectiveness, and scalability. Efficient partitioning, performance optimizations, DML support, schema evolution, and other advanced features provided by Iceberg and Trino contribute to seamless data management and analytics. Embracing these technologies can lead to enhanced decision-making, streamlined workflows, and improved outcomes across multiple business units.

If you are interested in learning more about how to modernize your data lake, or optimize your existing data lake around Iceberg and Trino, contact us for more information.

About Starburst

For data-drive companies, Starburst offersa full-featured analytics platform built on open-source Trino.

Our platform includes the capabilities needed to discover, organize, and consume data without the need for time-consuming and costly migrations. We believe the lake should be the center of gravity and be the starting point for querying disparate data.

With Starburst, teams can access more complete data, lower the cost of infrastructure, use the tools best suited to their specific needs, and avoid vendor lock-in.

Trusted by companies like Novant Health, Assurance, Optum, Pfizer, Sophia Genetics, EMIS Health, Gilead and Genius, Starburst helps companies make better decisions faster on all their data.

To learn more, visit **www.starburst.io** and follow Starburst on Twitter and Linkedin.

A single point of access to all your data

Learn more at www.starburst.io