Starburst Academy

# Trino and Starburst Training Series: Modern table formats & Apache Iceberg

v1.0.0

## Session 2 of 5: Full Series Information

This workshop is focused on creating & querying data lake tables. These are the goals for this session.

- Move beyond Hive to understand the benefits of modern table formats such as Apache Iceberg.
- Run ACID compliant transactions to modify your data and understand how this creates new table versions.
- Explore time-travel queries and table rollbacks.
- Modify the partitioning strategy without rebuilding your table.

## Table of Contents


**Lab 1: Create Starburst Galaxy account and a data lake catalog (15 mins)** 1
**Lab 2: Create and populate Iceberg tables (15 mins)** 8
**Lab 3: Data modifications and snapshots with Iceberg (15 mins)** 15
**Lab 4: Exercise advanced features of Iceberg (15 mins)** 25

# Lab 1: Create Starburst Galaxy account and a data lake catalog (15 mins)

## Learning objectives

- This lab will walk you through the process of creating a new account within Starburst Galaxy. You will create a domain name and password and set up your account to begin using sample data. A data lake catalog, and associated privileges, will be created. Finally, you will connect this new catalog to an existing cluster.

## Prerequisites

- None.

## Activities

1. Sign up for Starburst Galaxy (one-time event)
2. Configure a data lake catalog
3. Set permissions
4. Add to cluster
5. Start the cluster

## Step 1 - Sign up for Starburst Galaxy (one-time event)

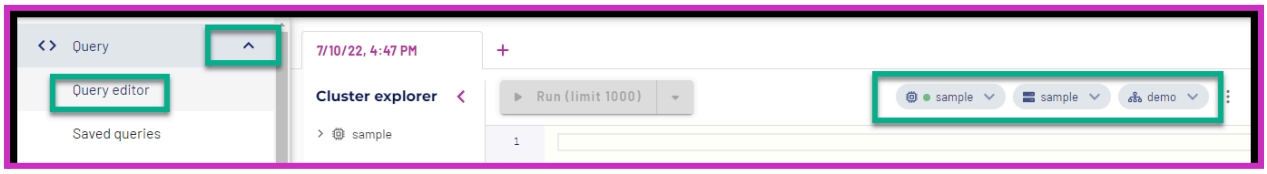
***For each webinar in the 5-part series, you will be using your own Starburst Galaxy environment.***
***If you have already registered for Starburst Galaxy, you may skip this FIRST step.***

To sign up for Starburst Galaxy, follow the instructions on the free registration page at https://www.starburst.io/platform/starburst-galaxy/start/.

**Note:** When prompted, choose to **not** connect your data sources, but choose to use the sample data.

Follow these steps to open the **Query editor**.

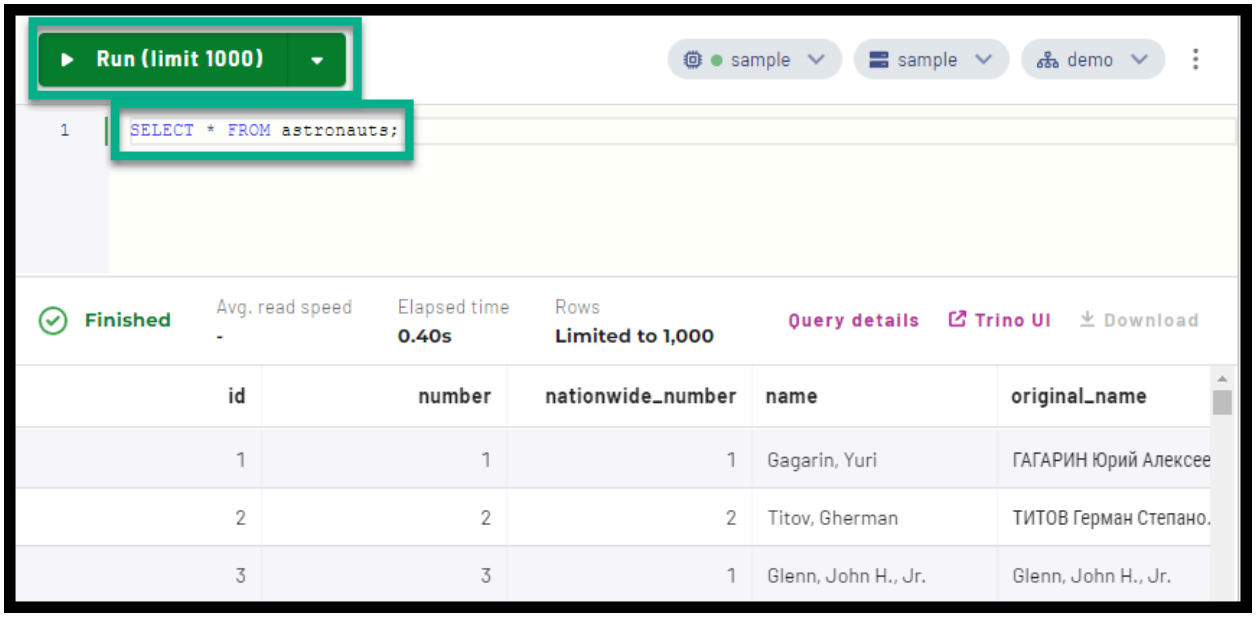
- Expand **Query** from the left menu
- Click **Query editor**
- In the **Select cluster** drop down, select **free-cluster**
- In the **Select catalog** drop down, select **sample**
- In the **Select schema** drop down, select **demo**




© 2022-2024 Starburst Data Inc. All rights reserved. 

Paste the following SQL into the editor and then click **Run (limit 1000)**.

```
SELECT * FROM astronauts;
```

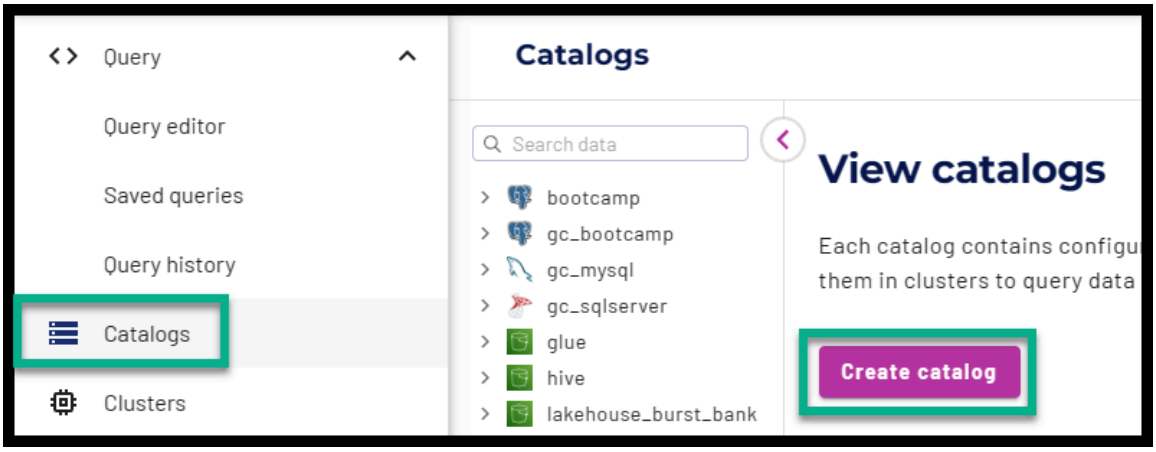



## Step 2 - Configure a data lake catalog

*For each webinar in the 5-part series, a different set of AWS keys will be used for the data lake catalog.* ***Even if you participated in a prior webinar, you must perform this step.***

If you previously created a catalog that starts with `w1-`, you should remove the catalog at some point. **Note**: You will have to remove it from any assigned clusters first.

Click **Catalogs** in the menu on the left and then click the **Create catalog** button.

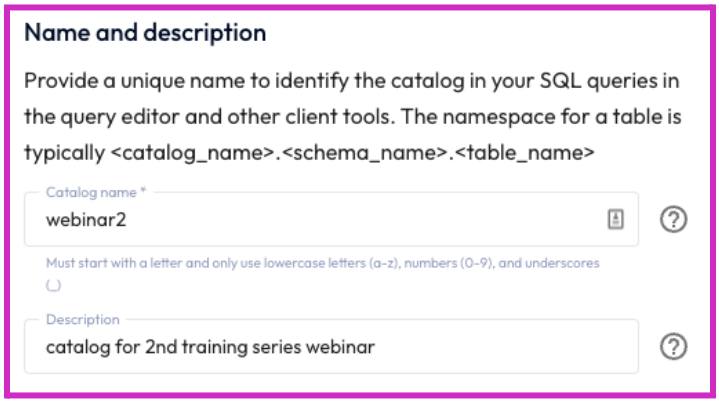



Click the **Amazon S3** tile.

© 2022-2024 Starburst Data Inc. All rights reserved.

Use the information below to configure your catalog.

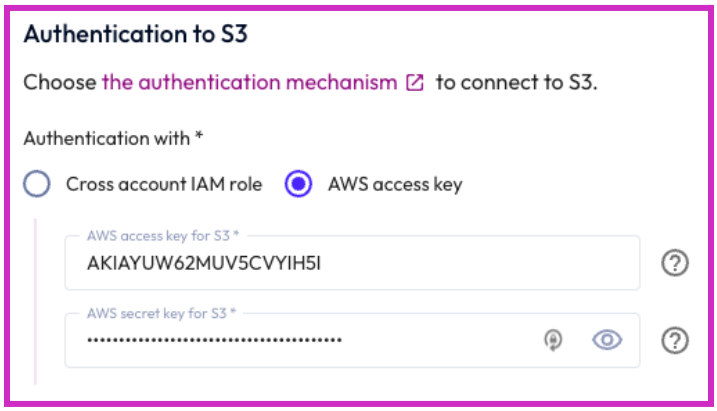
**Catalog name**: `webinar2`
**Description**: `catalog for 2nd training series webinar`




**Authentication with**: select the radio button **AWS access key**

**AWS access key for S3**: `AKIAYUW62MUV5CVYIH5I`
**AWS secret key for S3**: `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`

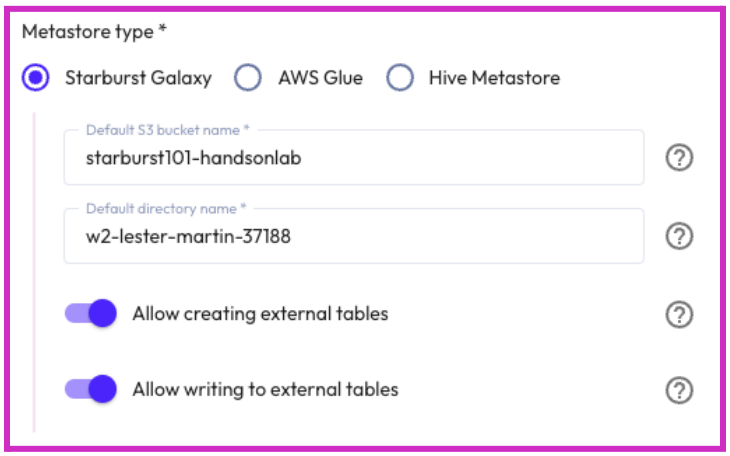
**Note:** These AWS credentials will only be operational through the weekend following the webinar. You will not be able to utilize this catalog beyond that point and should remove it from your Galaxy configuration.




© 2022-2024 Starburst Data Inc. All rights reserved. 

**Metastore type**: select the radio button **Starburst Galaxy**
**Default S3 bucket name:** `starburst101-handsonlab`
**Default directory name:** w2-fname-lname-postalcode (ex: `w2-lester-martin-37188`)

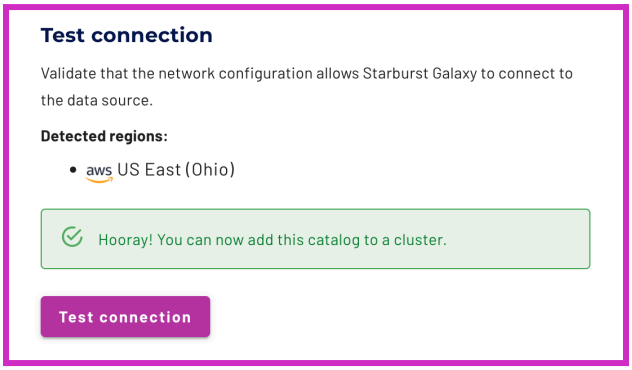
**Allow creating external tables**: enable the slider
**Allow writing to external tables**: enable the slider



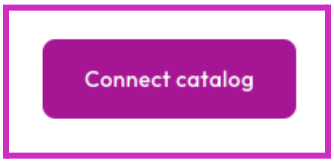

**Default table format:** ensure the radio button is selected to **Iceberg**

© 2022-2024 Starburst Data Inc. All rights reserved. 

Validate the connection by hitting **Test connection.** Your catalog should return the same message indicating that you can now add the catalog. Confirm you see the **Hooray! You can now add this catalog to a cluster** message.

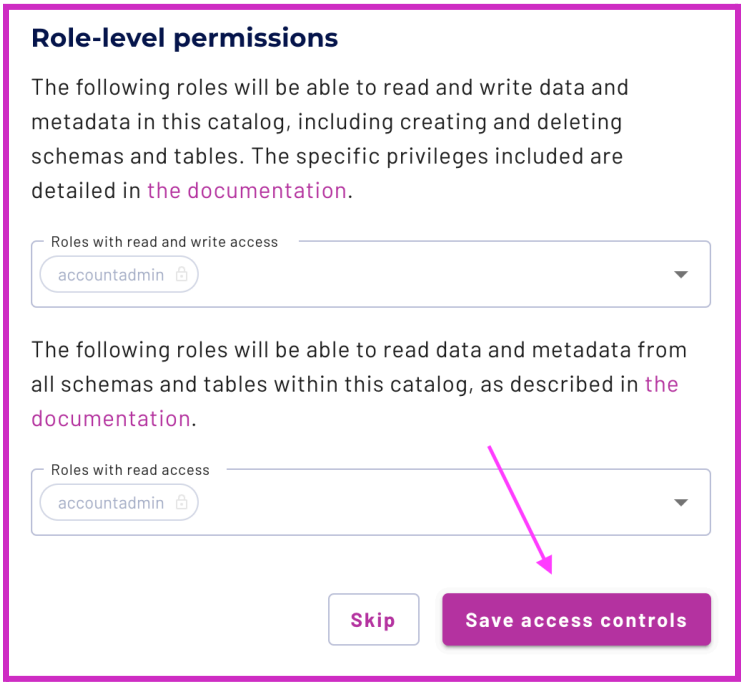



Select **Connect catalog**. This will save the credentials for your Amazon S3 catalog.




## Step 3 - Set permissions

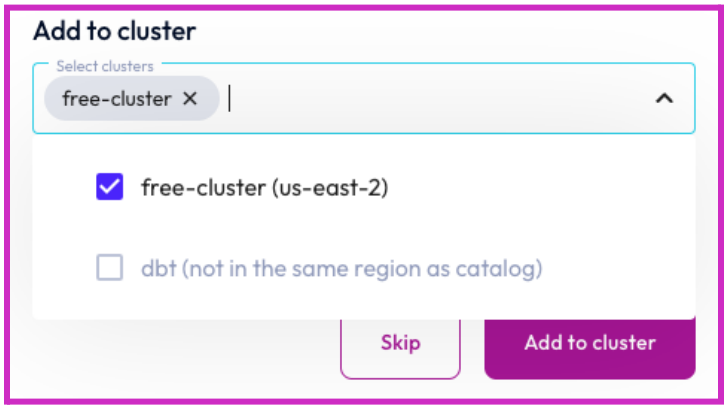
Next, accept the default permissions for your catalog by selecting the button **Save access controls**.



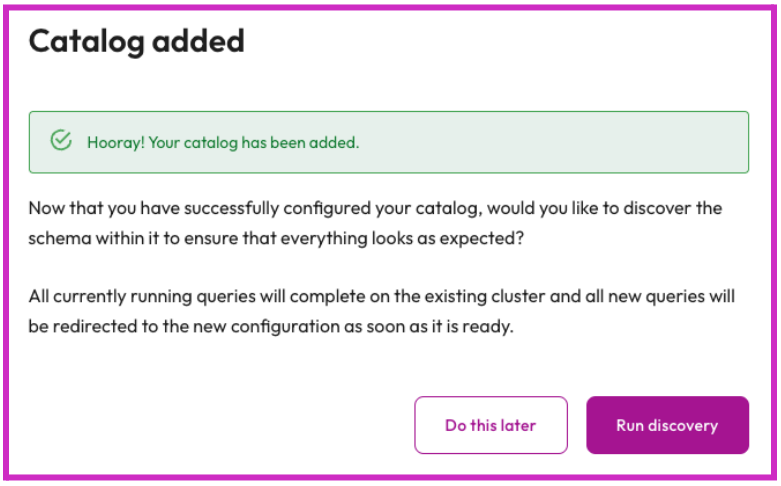

© 2022-2024 Starburst Data Inc. All rights reserved. 

## Step 4 - Add to cluster

Select `free-cluster` in the **Select clusters** pulldown and then click on **Add to cluster**.



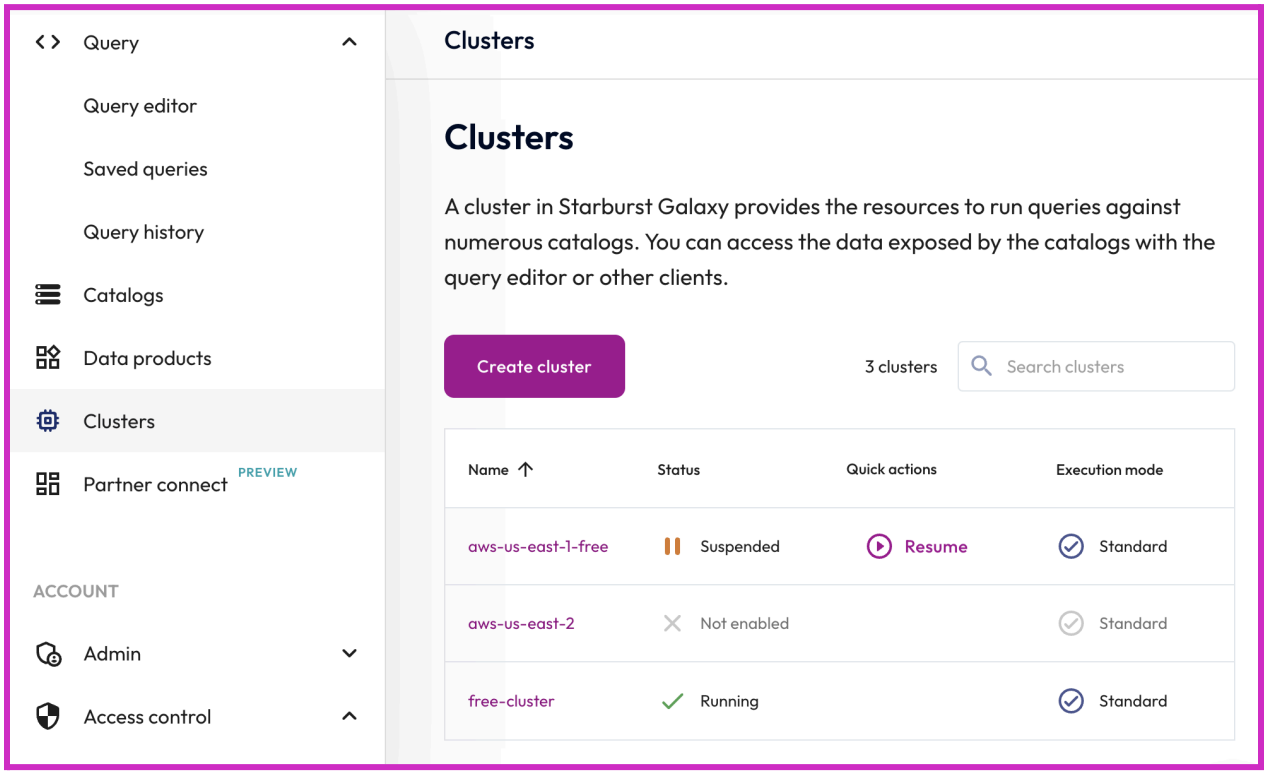

Click **Do this later** in the **Catalog added** pop-up.




© 2022-2024 Starburst Data Inc. All rights reserved. 

## Step 5 - Start the cluster

Click on **Clusters** on the left menu to see a list of the configured clusters in your account. Likely, you will only have one named `free-cluster`. The screenshot below shows multiple clusters.




If the **Status** for `free-cluster` is **Running**, then you are done with this step. If the **Status** is **Suspended**, click on **Resume** under **Quick actions** and wait for it to report as **Running**.

## END OF LAB EXERCISE

© 2022-2024 Starburst Data Inc. All rights reserved.

# Lab 2: Create and populate Iceberg tables (15 mins)

## Learning objectives

- In this lab, you will be introduced to the Iceberg table format and its many advantages. You will learn how to create Iceberg tables, add records, and query values using Iceberg. You will also explore the metadata tables to see how what kind of information is stored to enable versioning with snapshots.

## Prerequisites

- Lab 1 - Create Starburst Galaxy account and a data catalog

## Activities

1. Understand the motivation for modern table formats
2. Construct, populate, and query an Iceberg table
3. Understand where metadata is stored
4. Inspect table details using metadata tables
5. Add additional records to the table
6. Verify additional snapshot was created

## Step 1 - Understand the motivation for modern table formats

A "table format" is an open-source mechanism that manages and tracks all the files and metadata that make up a table. Apache Hive is the first-generation table format. The following table raises the limitations of Hive that drove the creation of modern table formats.

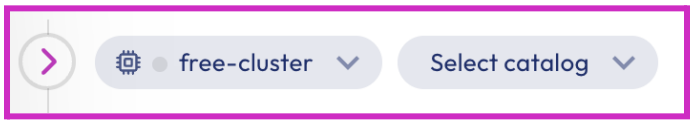
| | |
|---|---|
| **Metadata** | The Hive Metastore (HMS) impacts scalability & performance |
| **Data manipulation** | Inconsistent and limited `INSERT`, `UPDATE`, `DELETE`, and `MERGE` abilities |
| **Structural rigidity** | Difficult to evolve the schema and impossible to change partitioning |
| **Table history** | No inherent table history concepts for rollback or "as of" querying |

Table formats do not replace file formats. Modern table formats such as Apache Iceberg, Delta Lake, and Apache Hudi continue to use existing file formats such as Apache ORC, Apache Parquet, and Apache AVRO.

© 2022-2024 Starburst Data Inc. All rights reserved. 

## Step 2 - Construct, populate, and query an Iceberg table

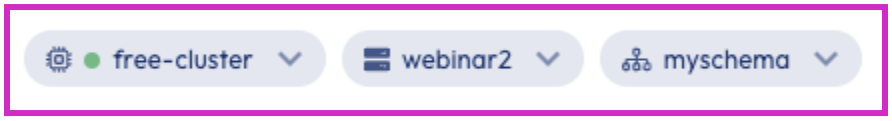
Select **Query** and **Query editor** from the left menu and then make sure `free-cluster` is the selected cluster.




Paste and **Run** the following SQL into a new editor tab.

```
CREATE SCHEMA webinar2.myschema;
USE webinar2.myschema;
```

This created a schema that you can create tables within. It also updates the pulldowns to the right of the cluster name to your newly created schema.




Run the following Data Definition Language (DDL) statement to create a new table named `my_iceberg_tbl` as shown below. Notice the `WITH` clause ensuring that it is created with the Iceberg type.

```
CREATE TABLE my_iceberg_tbl (
   id integer,
   name varchar(55),
   description varchar(255)
) WITH (
   TYPE = 'iceberg', FORMAT = 'parquet'
);
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

Add three new records to the table using the code below.

```
INSERT INTO my_iceberg_tbl
 (id, name, description)
VALUES
 (101, 'Leto', 'Ruler of House Atreides'),
 (102, 'Jessica', 'Concubine of the Duke'),
 (103, 'Paul', 'Son of Leto (aka Dale Cooper)');
```

Verify the 3 rows are present

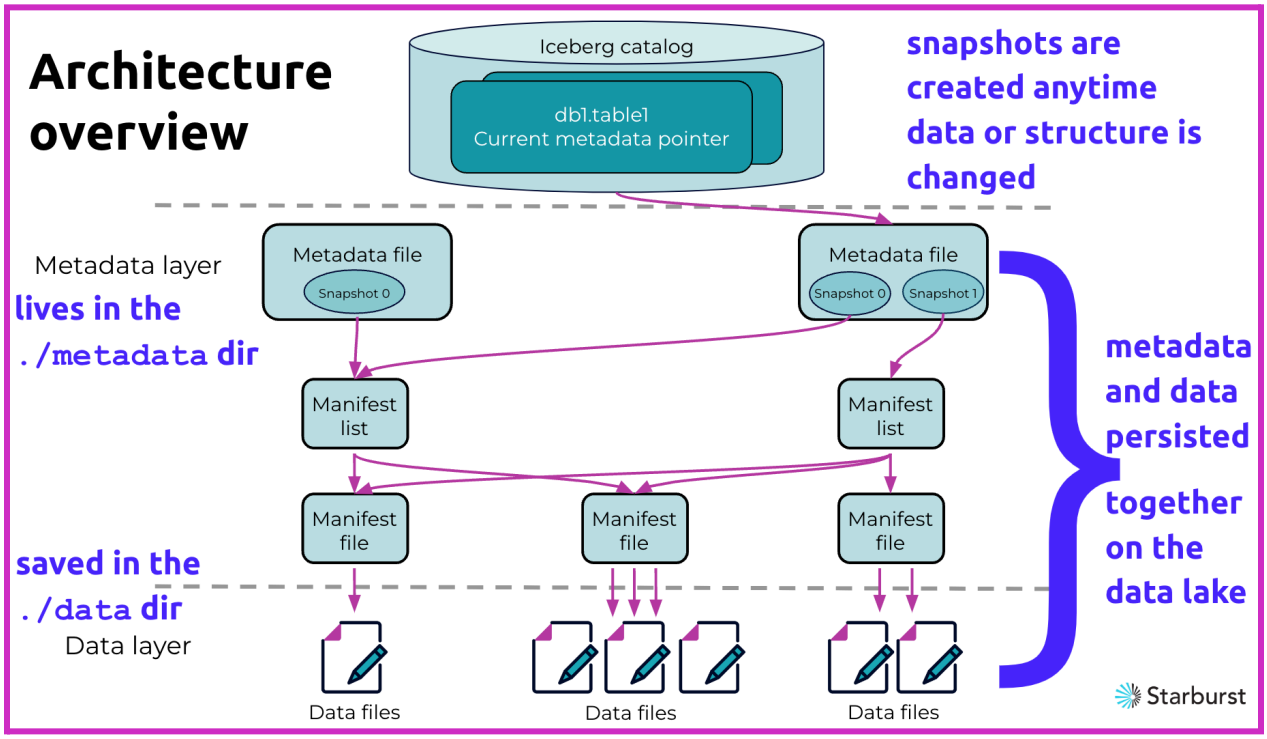
```
SELECT * FROM my_iceberg_tbl;
```

| id | name | description |
|---|---|---|
| 101 | Leto | Ruler of House Atreides |
| 102 | Jessica | Concubine of the Duke |
| 103 | Paul | Son of Leto (aka Dale Cooper) |

© 2022-2024 Starburst Data Inc. All rights reserved.

## Step 3 - Understand where metadata is stored

Iceberg stores table metadata (schema, partitioning details, statistics, versioning information, etc) on the data lake alongside the actual data files. The following diagram presents the richness of this architectural approach.




The previous diagram will not be fully described in this exercise, but the following excerpt was taken from the Apache Iceberg Specification and may aid your understanding.

> *This table format tracks individual data files in a table instead of directories. This allows writers to create data files in-place and only adds files to the table in an explicit commit.*
>
> *Table state is maintained in metadata files. All changes to table state create a new metadata file and replace the old metadata with an atomic swap. The table metadata file tracks the table schema, partitioning config, custom properties, and snapshots of the table contents. A snapshot represents the state of a table at some time and is used to access the complete set of data files in the table.*
>
> *Data files in snapshots are tracked by one or more manifest files that contain a row for each data file in the table, the file's partition data, and its metrics. The data in a snapshot is the union of all files in its manifests. Manifest files are reused across snapshots to avoid rewriting metadata that is slow-changing. Manifests can track data files with any subset of a table and are not associated with partitions.*

© 2022-2024 Starburst Data Inc. All rights reserved. 

*The manifests that make up a snapshot are stored in a manifest list file. Each manifest list stores metadata about manifests, including partition stats and data file counts. These stats are used to avoid reading manifests that are not required for an operation.*

The key points to understand are:

1. Data and metadata live together inside the table's data lake table location
    a. The metadata subdirectory contains JSON and AVRO files to represent multiple versions of a data
    b. The data subdirectory contains the actual data files (supports ORC, Parquet, and AVRO) that span all the various versions identified in the metadata
2. Snapshots (i.e. versions) are created anytime the structure or the data changes

Based on these points, the my_iceberg_tbl now has 2 snapshots:

1. The first was created when the table was created
2. The second was created when 3 rows were added to it

## Step 4 - Inspect table details using metadata tables

The [Iceberg connector — Starburst Enterprise]() metadata table documentation identifies ways to obtain the various metadata that is being stored for Iceberg tables without the need to inspect these files directly from the data lake. You will use several of these tables in this lab guide, and you can also explore others on your own that may not be utilized in these instructions.

Run the following query against the `$history` metadata table to see the snapshot identifiers for the first 2 versions.

```
SELECT made_current_at,
       snapshot_id, parent_id
  FROM "my_iceberg_tbl$history"
 ORDER BY made_current_at;
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

The first `snapshot_id` in the list refers to the initial version of the table which was created from the table being constructed. The next one is from the `INSERT` statement execution. Notice that its `parent_id` value (`1795314903365825559` in this example) is the same `snapshot_id` from the row before it.

The `$files` metadata table provides a detailed overview of the data files in the current snapshot. Run the following query which shows only one file currently present.

```
SELECT
 substring(file_path, position('/data/' IN file_path) + 6)
   AS file_path,
 record_count,
 value_counts,
 null_value_counts,
 lower_bounds,
 upper_bounds
FROM
 "my_iceberg_tbl$files";
```

Here is an explanation of some of the information returned.

| Column | Value | Notes |
|---|---|---|
| `file_path` | `20240207...`*`shortened...`*<br>`a694.parquet` | A specific file name. The remainder of the columns relate to this particular file |
| `record_count` | `3` | There are 3 records in the file |
| `value_counts` | `{ 1 = 3, 2 = 3, 3 = 3 }` | Each of the 3 columns has 3 values |
| `null_value_counts` | `{ 1 = 0, 2 = 0, 3 = 0 }` | None of the columns have any null values present |
| `lower_bounds` | `{ 1 = 101, 2 = Jessica, 3 = Concubine of the }` | > The `id` field ranges from `101` to `103`<br>> The `name` field ranges from `Jessica` to `Paul`<br>> The `description` field ranges from `Concubine…` to `Son…` |
| `upper_bounds` | `{ 1 = 103, 2 = Paul, 3 = Son of Leto }` | |

© 2022-2024 Starburst Data Inc. All rights reserved.

## Step 5 - Add additional records to the table

Add a few more rows.

```
INSERT INTO my_iceberg_tbl
 (id, name, description)
VALUES
 (104, 'Thufir', 'Mentat'),
 (201, 'Vladimir', 'Ruler of House Harkonnen'),
 (202, 'Rabban', 'Ruthless nephew of Vladimir'),
 (203, 'Feyd-Rautha', 'Savvy nephew of Vladimir (played by Sting)'),
 (301, 'Reverend Mother Gaius Helen Mohiam', null);
```

Run the `$files` query again to see there are two files now.

| _col0 | record_count |
|---|---|
| 20240207_230652_01046_jpk7r-8a16... | 3 |
| 20240208_234503_02733_msd8w-fd... | 5 |

Here is an explanation from the new row with a `record_count` equal to `5`.

| Column | Value | Notes |
|---|---|---|
| `file_path` | `20240208...`*`shortened...`*`fa09.parquet` | A specific file name. The remainder of the columns relate to this particular file |
| `record_count` | `5` | There are 5 records in the file |
| `value_counts` | `{ 1 = 5, 2 = 5, 3 = 5 }` | Each of the 3 columns has 5 values |
| `null_value_counts` | `{ 1 = 0, 2 = 0, 3 = 1 }` | One of the rows has a null for the `description` field |
| `lower_bounds` | `{ 1 = 104, 2 = Feyd-Rautha, 3 = Mentat }` | > The `id` field ranges from `104` to `301`<br>> The `name` field ranges from `Feyd-Rautha` to `Vladimir`<br>> The `description` field ranges from `Mentat` to `Savvy` |
| `upper_bounds` | `{ 1 = 301, 2 = Vladimir, 3 = Savvy nephew of! }` | |

© 2022-2024 Starburst Data Inc. All rights reserved.

| | | |
|---|---|---|
| | | `nephew...` |

## Step 6 - Verify additional snapshot was created

Execute the `$history` query from **Step 4** to verify there are 3 snapshots present now.

```
28  SELECT made_current_at,
29          snapshot_id, parent_id
30   FROM "my_iceberg_tbl$history"
31  ORDER BY made_current_at;
```

Finished | Avg. read speed 3.5 rows/s | Elapsed time 0.86s | Rows 3

| made_current_at | snapshot_id | parent_id |
|---|---|---|
| 2024-02-07 18:05:54.750 America/... | 1795314903365825559 | NULL |
| 2024-02-07 18:06:56.197 America/N... | 4743902169453086837 | 1795314903365825559 |
| 2024-02-08 18:45:06.308 America/... | 3403429904591088466 | 4743902169453086837 |

## END OF LAB EXERCISE

© 2022-2024 Starburst Data Inc. All rights reserved.

# Lab 3: Data modifications and snapshots with Iceberg (15 mins)

## Learning objectives

- In this lesson, you will explore how Iceberg uses snapshot files to create a comprehensive picture of changes made to tables. To do this, you will set up a table, make changes, then view the impact that this has on snapshot files. You will then query the snapshot files using the Iceberg time travel feature, which allows you to view results from the database at various moments in time or even roll the table back to a previous state.

## Prerequisites

- Lab 2 - Create and populate Iceberg tables

## Activities

1. Create testing table
2. Trace snapshots back to specific queries
3. Add historical order records from a week ago
4. Add activation records from six days ago
5. Add an error record from five days ago
6. Modify some previous records
7. Understanding snapshot files
8. Time travel queries with snapshot id
9. Time travel queries against a point in time
10. Rolling back to a previous state using snapshot files

## Step 1 - Create testing table

Create a new table in your schema with the following Data Definition Language (DDL).

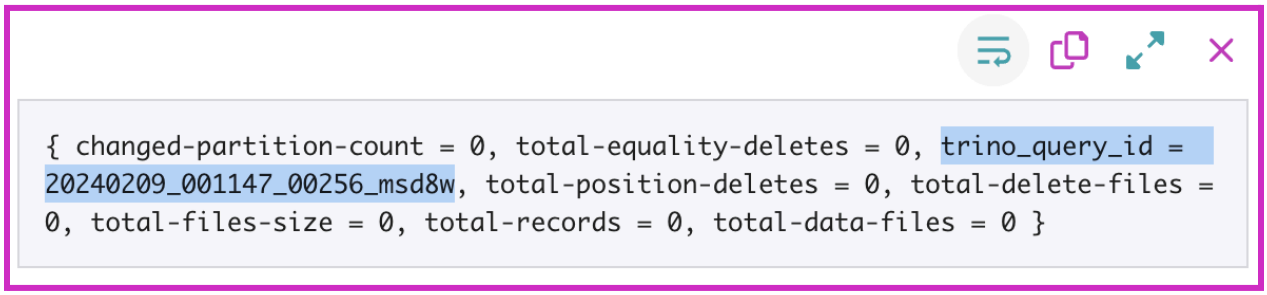
```
USE webinar2.myschema;

CREATE TABLE phone_provisioning (
 phone_nbr bigint,
 event_time timestamp(6),
 action varchar(15),
 notes varchar(150)
)
WITH (
 type='iceberg',
 partitioning=ARRAY['day(event_time)']
);
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

Verify that a snapshot was created for the table creation. You previously used the `$history` metadata table to see the list of snapshots. The `$snapshots` table provides a more detailed view.

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

The output above shows a single row representing the first snapshot which captures the table creation itself. Click on the value of the `summary` column to see the details in a pop-up.

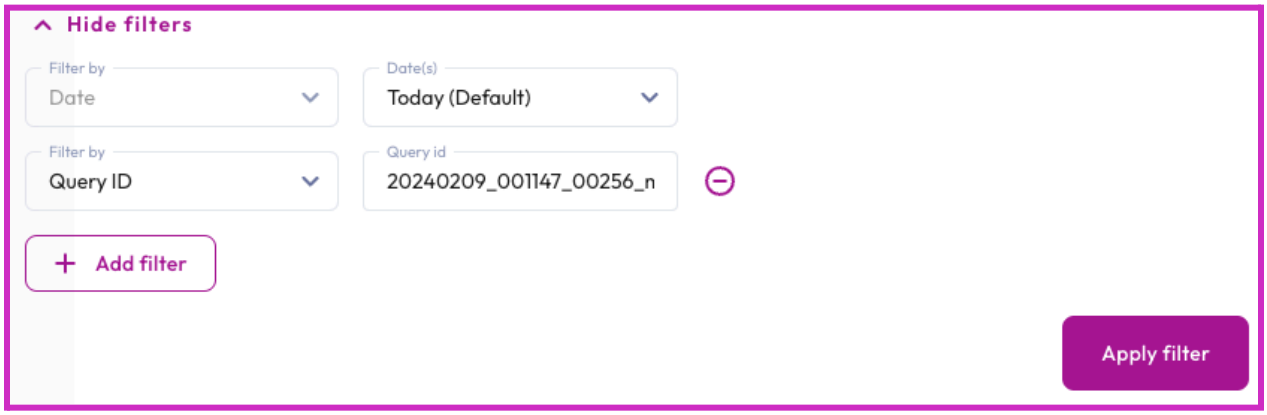



This indicates that no data changes happened as part of this snapshot. In fact, it also shows there are zero `total-records` as you would expect having only run the DDL.

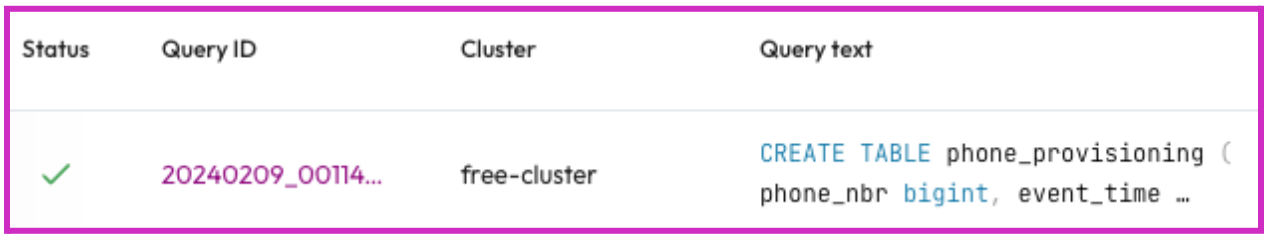
© 2022-2024 Starburst Data Inc. All rights reserved.

## Step 2 - Trace snapshots back to specific queries

Before you close the pop-up window, copy the value for `trino_query_id` onto your clipboard (`20240209_001147_00256_msd8w` in the example above). Click **Query** > **Query history** from the left menu. Press **Add filter** and choose **Query ID** from the **Filter by** pulldown. Paste your clipboard's contents into the **Query id** textbox.




After you click on the **Apply filter** button, you will see that the `summary` column of `$snapshots` allows for traceability back to the query that produces a specific version.




## Step 3 - Add historical order records from a week ago

Add historical records from a week ago to capture the initial orders for two new phone numbers.

```
INSERT INTO
 phone_provisioning (phone_nbr, event_time, action, notes)
VALUES
 (
   1111111, current_timestamp(6) - interval '7' day, 'ordered', null
 ),
 (
   2222222, current_timestamp(6) - interval '7' day, 'ordered', null
 );
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

Verify the two records are present as expected.

```
SELECT * FROM phone_provisioning ORDER BY event_time DESC;
```

Verify a new snapshot was created.

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

There is now a second row with a new `snapshot_id`. More analysis of this metadata will be discussed in **Step 7**.

| committed_at | snapshot_id | parent_id | operation | manifest_list | summary |
|---|---|---|---|---|---|
| 2023-04-21 09:53:3... | 39760182185247010... | NULL | append | s3://edu-train-galax... | { changed-parti |
| 2023-04-21 09:57:4... | 23934521210609399... | 39760182185247010... | append | s3://edu-train-galax... | { changed-parti |

## Step 4 - Add activation records from six days ago

Add historical records from six days ago to capture the activation activity for the same two phone numbers.

```
INSERT INTO
 phone_provisioning (phone_nbr, event_time, action, notes)
VALUES
 (
   1111111, current_timestamp(6) - interval '6' day, 'activated',
null
 ),
 (
   2222222, current_timestamp(6) - interval '6' day, 'activated',
null
 );
```

Verify the records are present as expected, and a new snapshot was created, by running the last two `SELECT` statements in **Step 3**. There will be two more records added to the `phone_provisioning` table and the `$snapshot` table will now have a third record.

© 2022-2024 Starburst Data Inc. All rights reserved. 

Reviewing the `summary` column for the new record in `$snapshot` details that 2 records were added as part of this snapshot.

```
{ changed-partition-count = 1, added-data-files = 1, total-equality-deletes =
0, added-records = 2, trino_query_id = 20240209_011536_01198_msd8w, total-
position-deletes = 0, added-files-size = 773, total-delete-files = 0, total-
files-size = 1536, total-records = 4, total-data-files = 2 }
```

More analysis of this metadata will be presented in **Step 7**.

## Step 5 - Add an error record from five days ago

Add a historical activation record from five days ago to capture an error that was reported on phone number `2222222`.

```
INSERT INTO
  phone_provisioning (phone_nbr, event_time, action, notes)
VALUES
  (2222222, current_timestamp(6) - interval '5' day, 'errorReported',
  'customer reports unable to initiate call');
```

Once again, verify the records are present as expected and a new snapshot was created.

## Step 6 - Modify some previous records

Four days ago, a system error prevented the `notes` column from being populated correctly before it was fixed. Upon review of the problem, it was determined that the following two `UPDATE` commands are needed to modify the affected records.

```
UPDATE phone_provisioning
  SET notes = 'customer requested new number'
WHERE action = 'ordered'
  AND notes is null;

UPDATE phone_provisioning
  SET notes = 'number successfully activated'
WHERE action = 'activated'
  AND notes is null;
```

Review all rows again.

```
SELECT * FROM phone_provisioning ORDER BY event_time DESC;
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

Your `event_time` values will be different, but the rows shown below should be similar to your results.

```
42  SELECT * FROM phone_provisioning ORDER BY event_time DESC;
43
```

Finished | Avg. read speed: 8 rows/s | Elapsed time: 0.63s | Rows: 5 | Query details

| phone_nbr | event_time | action | notes |
|---|---|---|---|
| 2222222 | 2022-12-22 12:43:55.286210 | errorReported | customer reports unable to initiate call |
| 1111111 | 2022-12-21 12:30:31.800953 | activated | number successfully activated |
| 2222222 | 2022-12-21 12:30:31.800953 | activated | number successfully activated |
| 2222222 | 2022-12-20 12:20:50.457304 | ordered | customer requested new number |
| 1111111 | 2022-12-20 12:20:50.457304 | ordered | customer requested new number |

Query the `$snapshots` table again.

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

Two new snapshots were created from the two `UPDATE` statements, as shown at the bottom of the list below.

Finished | Avg. read speed: 8.6 rows/s | Elapsed time: 0.70s | Rows: 6

| committed_at | snapshot_id | parent_id | operation |
|---|---|---|---|
| 2024-02-08 19:11:48.... | 8034741285576235... | NULL | append |
| 2024-02-08 19:41:27.... | 8356658077675739... | 8034741285576235... | append |
| 2024-02-08 20:17:0... | 2845847475539750... | 8356658077675739... | append |
| 2024-02-08 20:23:5... | 4779205906692078... | 2845847475539750... | append |
| 2024-02-08 20:26:0... | 1381994836274965... | 4779205906692078... | overwrite |
| 2024-02-08 20:26:0... | 1159370510651195... | 1381994836274965... | overwrite |

© 2022-2024 Starburst Data Inc. All rights reserved.

## Step 7 - Understanding snapshot files

Iceberg records changes to a table as snapshots. For example, each of the 2 `UPDATE` statements executed created their own snapshot with an `operation` value of `overwrite`. Iceberg cannot perform an in-place update to the underlying immutable data files.

The phrase "overwrite", instead of "update", is more appropriate as Iceberg has to create a file referencing the location in the existing file(s) that contain the record(s) to be updated. This is called a "delete file".

Then, as part of an atomic operation, Iceberg creates a new data file that has the full record being "updated". It essentially is an "add" of the record with all updated columns as well as the existing values for columns not updated.

These new delete files and data files that are created will be read after the preceding data files are. This allows Iceberg to modify the data prior to returning it by applying the delete files (i.e. delete the records) and then including the new data files (which will look like net-new records).

This is somewhat analogous to RDBMS "transaction logs" that store a running history of modifications into. The difference is that the classical databases are creating their transaction logs for recovery & replication purposes. Iceberg creates a series of deltas that will be used in a "merge on read" strategy when the table is queried.

To understand this better, click on the `summary` value for the last snapshot in the list. The **Output** identified below is a subset of all the properties present. They are the ones most important to this discussion.

### Output

```
{
 total-position-deletes = 2,
 total-delete-files = 1,

 added-records = 2,
 added-data-files = 1,
}
```

These totals are for the `UPDATE` statement that had `action = 'activated' AND notes is null` within it. It logically changed two records.

With Iceberg's inability to perform in-place updates on the underlying files, `total-position-deletes = 2` indicates that 2 records were marked for deletion. Fortunately, both of these were placed in a single delete file.

© 2022-2024 Starburst Data Inc. All rights reserved. 

Closely following those deletes, `added-records = 2` indicates the records deleted are being re-added as essentially new inserts. As before, these were assembled into a single new data file.

More information on delete files and the overall strategy around handling the `UPDATE` statement can be found in the [Iceberg Specification](#).

## Step 8 - Time travel queries with snapshot id

So, what can you do with snapshot files? One useful feature is called "time travel". This allows you to query prior versions of the table via the `snapshot_id`. The second row in the results below relates to the second snapshot. The first snapshot was at the empty table creation. The second was the initial `INSERT` statement.

```
45  SELECT * FROM "phone_provisioning$snapshots";
46
```

Finished | Avg. read speed 8.3 rows/s | Elapsed time 0.97s | Rows 8 | Query details

| committed_at | snapshot_id | parent_id | operation |
|---|---|---|---|
| 2022-12-27 12:01:31.5... | 5641615054948073642 | NULL | append |
| 2022-12-27 12:20:52.... | 2701885389952950484 | 5641615054948073642 | append |
| 2022-12-27 12:30:32... | 4503567719363821411 | 2701885389952950484 | append |

Replace `12345678890` in the following query with ***your*** second `snapshot_id` value to see what the table looked like back at that point.

```
SELECT * FROM phone_provisioning
FOR VERSION AS OF 1234567890
ORDER BY event_time DESC;
```

You should see the first two records initially added, similar to the image pictured below.

| phone_nbr | event_time | action | notes |
|---|---|---|---|
| 1111111 | 2022-12-20 12:20:50.457304 | ordered | NULL |
| 2222222 | 2022-12-20 12:20:50.457304 | ordered | NULL |

© 2022-2024 Starburst Data Inc. All rights reserved.

## Step 9 - Time travel queries against a point in time

Another alternative mechanism to leverage time travel is to run a query based on a past point in time. You can exercise this by swapping `VERSION` above with `TIMESTAMP` and replacing the `snapshot_id` with a timestamp.

Familiarize yourself with the documentation on time travel. Many unique use-cases make this feature invaluable.

Previously, we added a record with a timestamp that was 5 days in the past. Run a query to see what that table's contents were 5.5 days ago.

```
SELECT * FROM phone_provisioning
FOR TIMESTAMP AS OF current_timestamp(6) - interval '132' hour
ORDER BY event_time DESC;
```

You should have received an error like the following.

```
No version history table myschema."phone_provisioning" at or before 2024-02-03T14:45:16.295Z
```

Do you understand what happened here? The query above assumed that the `event_time` column's timestamp was being used, but it is only a timestamp-based column and is not directly related to the versioning information. The rows from the `$snapshots` metadata table have a `committed_at` timestamp which is leveraged when `FOR TIMESTAMP AS OF` is utilized.

Run the last query again after changing the `interval` type from `hour` to `minute` and plug in a single-digit number instead of `132`. Increment and/or decrement the number until you get the results you are looking for.

**Hint**: use an appropriate number of minutes that would make the timestamp slightly before the `committed_at` column value from `$snapshots` for the snapshot you are trying to read the data from.

© 2022-2024 Starburst Data Inc. All rights reserved. 

## Step 10 - Rolling back to a previous state using snapshot files

Time travel also provides the ability to roll a table back to a previous snapshot. Use the following steps to determine the current `snapshot_id` and ***copy it into your query editor*** for later use.

To begin, query `$snapshots` again and get the `snapshot_id` value from the row with the most recent `committed_at` timestamp to be the snapshot to rollback to. You could also look for the most recent `snapshot_id` in the `$history` metadata table.

```
-- save the snapshot_id value in the editor
SELECT snapshot_id FROM "phone_provisioning$history"
ORDER BY made_current_at DESC LIMIT 1;
```

Run the following query to remove all the records for one of the phone numbers.

```
DELETE FROM phone_provisioning
 WHERE phone_nbr = 2222222;
```

Verify that 3 of the 5 rows were deleted.

```
SELECT * FROM phone_provisioning ORDER BY event_time DESC;
```

It was determined that the `DELETE` was a mistake and needs to be recovered from. Use the `snapshot_id` you saved earlier in the following command that rolls the table back to before the deletions.

```
CALL webinar2.system.rollback_to_snapshot(
   'myschema', 'phone_provisioning',
    1234567890);
```

Finally, verify the rows have been recovered.

```
SELECT * FROM phone_provisioning ORDER BY event_time DESC;
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

The result should look similar to the image below.

```
68  select * from phone_provisioning order by event_time desc;
```

Finished | Avg. read speed **8.2 rows/s** | Elapsed time **0.61s** | Rows **5** | Query

| phone_nbr | event_time | action | notes |
|---|---|---|---|
| 2222222 | 2022-12-27 18:57:12.697568 | errorReported | customer reports unable to initiate call |
| 1111111 | 2022-12-26 18:56:15.196527 | activated | number successfully activated |
| 2222222 | 2022-12-26 18:56:15.196527 | activated | number successfully activated |
| 2222222 | 2022-12-25 18:55:41.053462 | ordered | customer requested new number |
| 1111111 | 2022-12-25 18:55:41.053462 | ordered | customer requested new number |

## END OF LAB EXERCISE

© 2022-2024 Starburst Data Inc. All rights reserved.

# Lab 4: Exercise advanced features of Iceberg (15 mins)

## Learning objectives

- In this lab, you will explore Iceberg's advanced features using an airplane dataset. Specifically, you will learn how tables can evolve in specific ways, including renaming columns, adding additional columns, renaming partitions, and eliminating partitions. At each step, you will see what impact these changes have on the records in the table and investigate how these changes are tracked by Iceberg using snapshot files. Finally, you will learn how to compact small files into fewer/larger ones.

## Prerequisites

- Lab 3 - Data modifications and snapshots with Iceberg

## Activities

1. Create table
2. Insert records
3. Change column name using `ALTER TABLE`
4. Add new records
5. Add additional columns using `ALTER TABLE`
6. Modify existing records using new column layout
7. Rename and partition table
8. Add two new helicopters into modified table
9. Check partition metadata
10. Change the partition strategy (again)
11. Compaction
12. Thoughts on repartitioning previous data…

## Step 1 - Create table

In this lab, you will build and populate a table with airplane data.

```
USE webinar2.myschema;

CREATE TABLE planes (
   tail_number varchar(15),
   name varchar(150),
   color varchar(15)
) WITH ( type = 'iceberg');
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

## Step 2 - Insert records

Insert two new records into the table using the code below.

```
INSERT INTO planes (tail_number, name)
VALUES
 ('N707JT', 'John Travolta''s Boeing 707'),
 ('N1KE', 'Nike corp jet');
```

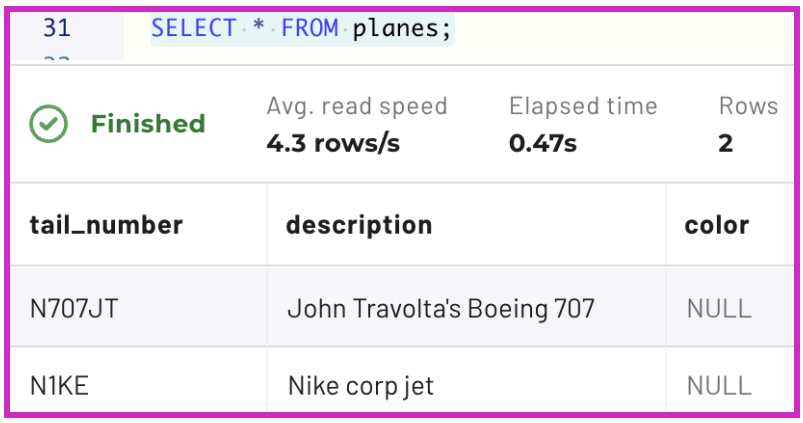
## Step 3 - Change column name using `ALTER TABLE`

At this point, you realize the `name` is really more of a `description`. Change the column name using the code below. Note the use of the `ALTER TABLE` syntax.

```
ALTER TABLE planes RENAME COLUMN name TO description;
```

Verify all is reporting correctly.

```
SELECT * FROM planes;
```



```
31  SELECT * FROM planes;
```

Finished | Avg. read speed 4.3 rows/s | Elapsed time 0.47s | Rows 2

| tail_number | description | color |
|---|---|---|
| N707JT | John Travolta's Boeing 707 | NULL |
| N1KE | Nike corp jet | NULL |

## Step 4 - Add new records

Add another plane.

```
INSERT INTO planes (tail_number, color, description)
VALUES
 ('N89TC', 'white',
  '1975 Bombardier Learjet 35 w/Light Jet classification');
```

© 2022-2024 Starburst Data Inc. All rights reserved.

## Step 5 - Add additional columns using `ALTER TABLE`

At this point, you realize some additional columns are appropriate. Add them now using the code below.

```
ALTER TABLE planes ADD COLUMN class varchar(50);
ALTER TABLE planes ADD COLUMN year integer;
ALTER TABLE planes ADD COLUMN make varchar(100);
ALTER TABLE planes ADD COLUMN model varchar(100);
```

Additionally, the `color` column is not relevant enough to be maintained.

```
ALTER TABLE planes DROP COLUMN color;
```

## Step 6 - Modify existing records using new column layout

Modify the existing three rows to leverage the new column layout.

```
UPDATE planes
  SET class = 'Jet Airliner',
      year = 1964,
      make = 'Boeing',
      model = '707-138B'
WHERE tail_number = 'N707JT';

UPDATE planes
  SET class = 'Heavy Jet',
      year = 2021,
      make = 'Gulfstream',
      model = 'G650'
WHERE tail_number = 'N1KE';

UPDATE planes
  SET class = 'Light Jet',
      year = 1975,
      make = 'Bombardier',
      model = 'Learjet 35',
      description = null
WHERE tail_number = 'N89TC';
```

When complete, verify that the reporting is correct. The output should resemble the image below.

```
SELECT * FROM planes;
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

```
79  SELECT * FROM planes;
80
```

Finished | Avg. read speed **4 rows/s** | Elapsed time **0.74s** | Rows **3** | Query details | Trino

| tail_number | description | class | year | make | model |
|---|---|---|---|---|---|
| N707JT | John Travolta's Boeing... | Jet Airliner | 1964 | Boeing | 707-138B |
| N89TC | NULL | Light Jet | 1975 | Bombardier | Learjet 35 |
| N1KE | Nike corp jet | Heavy Jet | 2021 | Gulfstream | G650 |

## Step 7 - Rename and partition table

Imagine that your company has now decided to carry other types of aircraft, not just planes.

You can use `ALTER TABLE` to rename the `planes` table to allow for additional types, such as helicopters, to be added. The new name for the table will be `aircrafts`.

```
ALTER TABLE planes RENAME TO aircrafts;
```

Also, add partitioning on the `classification` column.

```
ALTER TABLE aircrafts
SET PROPERTIES partitioning = ARRAY['class'];
```

**Note**: Changing the partitioning strategy does NOT repartition previously persisted data.

## Step 8 - Add two new helicopters into modified table

Add two helicopters to the table as new records using the code below.

```
INSERT INTO aircrafts
 (tail_number, class, year, make, model, description)
VALUES
 ('N535NA', 'Helicopter', 1969, 'Sikorsky', 'UH-19D', 'NASA'),
 ('N611TV', 'Helicopter', 2022, 'Robinson', 'R66', null);
```

When complete, verify that these records have been added using the code below.

```
SELECT tail_number, class, year, make, model, description
  FROM aircrafts ORDER BY tail_number;
```

© 2022-2024 Starburst Data Inc. All rights reserved.

```
SELECT tail_number, class, year, make, model, description
  FROM aircrafts ORDER BY tail_number;
```

Finished | Avg. read speed 7.1 rows/s | Elapsed time 0.70s | Rows 5 | Query details

| tail_number | class | year | make | model | description |
|---|---|---|---|---|---|
| N1KE | Heavy Jet | 2021 | Gulfstream | G650 | Nike corp jet |
| N535NA | Helicopter | 1969 | Sikorsky | UH-19D | NASA |
| N611TV | Helicopter | 2022 | Robinson | R66 | NULL |
| N707JT | Jet Airliner | 1964 | Boeing | 707-138B | John Travolta's Boeing 707 |
| N89TC | Light Jet | 1975 | Bombardier | Learjet 35 | NULL |

## Step 9 - Check partition metadata

Check the `$partitions` metadata table to ensure that the metadata for the new records is being collected properly. There should be two records in the new partition. Use the code below.

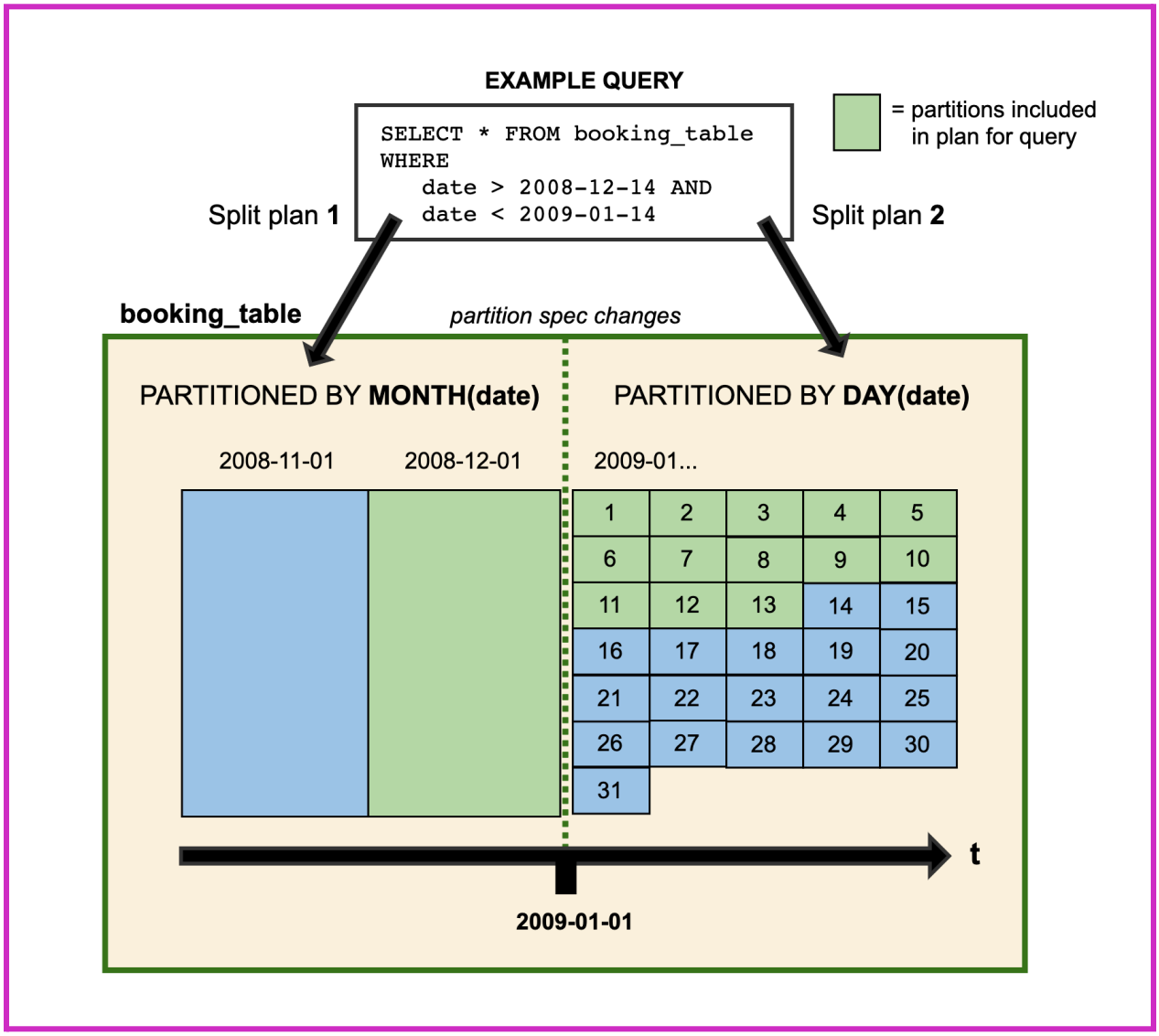
```
SELECT partition, record_count, file_count
  FROM "aircrafts$partitions";
```

| partition | record_count | file_count |
|---|---|---|
| { class = Helicopter } | 2 | 1 |
| { class = NULL } | 6 | 5 |

The second row above indicates that 6 rows were added when there was no partitioning defined. As stated previously, changes to the partitioning strategy do not automatically repartition the data previously persisted.

© 2022-2024 Starburst Data Inc. All rights reserved.

The following visualization and the specification's partition evolution explanation can help in understanding this better.




## Step 10 - Change the partition strategy (again)

Imagine that the needs of the business change. It is now determined that analysts are querying across specific years of manufacturing more than the class of the aircraft. Change the partition strategy to account for this.

```
ALTER TABLE aircrafts
SET PROPERTIES partitioning = ARRAY['year'];
```

Verify that this did not repartition any existing data by running the query from the last step again.

© 2022-2024 Starburst Data Inc. All rights reserved. 

Run this next query to find the distinct `year` values and see that each has only 1 aircraft.

```
SELECT year, count() nbr_for_year
  FROM aircrafts
 GROUP BY year ORDER BY year;
```

| year | nbr_for_year |
|---|---|
| 1964 | 1 |
| 1969 | 1 |
| 1975 | 1 |
| 2021 | 1 |
| 2022 | 1 |

Execute the following statement to add one more aircraft for each year of manufacture.

```
INSERT INTO aircrafts
 (tail_number, class, year, make, model, description)
VALUES
 ('dummy', 'unknown', 1964, 'acme', 'cool', null),
 ('dummy', 'unknown', 1969, 'acme', 'cool', null),
 ('dummy', 'unknown', 1975, 'acme', 'cool', null),
 ('dummy', 'unknown', 2021, 'acme', 'cool', null),
 ('dummy', 'unknown', 2022, 'acme', 'cool', null);
```

Run the `GROUP BY` query again to see there are now 2 aircraft for each year.

© 2022-2024 Starburst Data Inc. All rights reserved.

Now, run the following to show that the 5 new records were all added into separate partitions based on their `year` values.

```
SELECT partition, record_count, file_count, data
  FROM "aircrafts$partitions"
 ORDER BY record_count;
```

Finished | Avg. read speed **8.5 rows/s** | Elapsed time **0.82s** | Rows **7**

| partition | record_count | file_count | data |
|---|---|---|---|
| { class = NULL, year = 1969 } | 1 | 1 | { tail_num |
| { class = NULL, year = 2022 } | 1 | 1 | { tail_num |
| { class = NULL, year = 1964 } | 1 | 1 | { tail_num |
| { class = NULL, year = 2021 } | 1 | 1 | { tail_num |
| { class = NULL, year = 1975 } | 1 | 1 | { tail_num |
| { class = Helicopter, year = NULL } | 2 | 1 | { tail_num |
| { class = NULL, year = NULL } | 6 | 5 | NULL |

The new records were persisted with the `year` partition strategy, the two helicopter records that were persisted when the partition strategy was using `class` are still as they were – just as the 6 original adds are when there was no partitioning value set.

Remember, changing the partitioning definition only impacts new records. It does not automatically alter old records created when the prior partition strategy was in effect.

## Step 11 - Compaction

Take a look at the `$files` metatable which references all files used by the current snapshot.

```
SELECT file_path, record_count, file_size_in_bytes
 FROM "aircrafts$files";
```

© 2022-2024 Starburst Data Inc. All rights reserved. 

Finished | Avg. read speed 17.1 rows/s | Elapsed time 0.64s | Rows 11

| file_path | record_count | file_size_in_bytes |
|---|---|---|
| s3://starburst101-handsonl... | 1 | 975 |
| s3://starburst101-handsonl... | 1 | 1017 |
| s3://starburst101-handsonl... | 1 | 1108 |
| s3://starburst101-handsonl... | 1 | 793 |
| s3://starburst101-handsonl... | 2 | 550 |
| s3://starburst101-handsonl... | 2 | 1004 |
| s3://starburst101-handsonl... | 1 | 905 |
| s3://starburst101-handsonl... | 1 | 905 |
| s3://starburst101-handsonl... | 1 | 905 |
| s3://starburst101-handsonl... | 1 | 905 |
| s3://starburst101-handsonl... | 1 | 905 |

The “merge on read” strategy leaves a sprawl of files as changes continue to be made. If the number of rows being updated or deleted is small for each operation, the size of the delete file(s) and data file(s) can be quite small as we see in the results above.

Even without running modification operations, the frequency and size of periodic, or streaming, ingestion can also create small files. This “small files problem” will ultimately affect performance and scalability. The solution to this is compaction which essentially is reading multiple smaller files and rebuilding & replacing them with fewer larger ones.

Fortunately, we have a simple operation to trigger this maintenance that will eventually be needed. This operation uses a default value (100MB) to decide which files are considered small. Any file whose size is below that threshold is included in the compaction operation.

Additionally, this compaction process is aware of the delete files and accounts for these by excluding the original row when rebuilding a file, eliminates the delete file, and rolls the net-new records that were created into larger files.

Run the following to perform compaction.

```
ALTER TABLE aircrafts EXECUTE optimize;
```

Check the `$files` metatable again.

© 2022-2024 Starburst Data Inc. All rights reserved. 

```
SELECT file_path, record_count, file_size_in_bytes
  FROM "aircrafts$files";
```

| file_path | record_count | file_size_in_bytes |
|---|---|---|
| s3://starburst101-handso... | 2 | 952 |
| s3://starburst101-handso... | 2 | 976 |
| s3://starburst101-handso... | 2 | 1091 |
| s3://starburst101-handso... | 2 | 1015 |
| s3://starburst101-handso... | 2 | 938 |

There are fewer files present now using the query below. These files are still very small in size due to the small number of records inserted into them. Nonetheless, the problem has been greatly reduced.

## Step 12 - Thoughts on repartitioning previous data…

It might seem like fewer/larger files should have been written. Compaction created 5 files because the table's current partition strategy was utilized.

Run this query again to verify this happened.

```
SELECT partition, record_count, file_count, data
  FROM "aircrafts$partitions"
 ORDER BY record_count;
```

Did we just discover a relatively easy way to repartition pre-existing data after changing the partitioning strategy?

Hint: Yes we did! Ask on https://www.starburst.io/community/forum/ if this isn't clear.

## END OF LAB EXERCISE

© 2022-2024 Starburst Data Inc. All rights reserved. 