

Cloudera Runtime 7.3.2

Iceberg support for Atlas

Date published: 2020-07-28

Date modified: 2026-03-31

The Cloudera logo is displayed in a bold, orange, sans-serif font. The word "CLOUDERA" is written in all caps, with the letter 'E' in "CLouDERA" featuring a unique design where the top bar is a horizontal line that extends to the right and then turns down to form the top of the letter.

<https://docs.cloudera.com/>

Legal Notice

© Cloudera Inc. 2026. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Iceberg support for Atlas.....	4
How Atlas works with Iceberg.....	4
Using the Spark shell.....	5
Using the Impala shell.....	20

Iceberg support for Atlas

Atlas integration with Iceberg helps you identify the Iceberg tables to scan data and provide lineage support. Learn how Atlas works with Iceberg and what schema evolution, partition specification, partition evolution are with examples.

How Atlas works with Iceberg

You can use Atlas to find, organize, and manage different aspects of data about your Iceberg tables and how they relate to each other. This enables a range of data stewardship and regulatory compliance use cases.

The Atlas connectors distinguish between Hive and Iceberg tables. The Iceberg table is available in a typedef format which implies that the underlying data can be retrieved by querying the Iceberg table. All attributes of the Hive table are available in the Iceberg table and this equivalence is achieved by creating the Iceberg table as a sub-type of the underlying Hive table. Optionally, the Iceberg table can also be queried by Hive or Impala engine. For more information about Iceberg and related concepts, see [Apache Iceberg features](#).

Both Iceberg and Hive tables have equality in Atlas in terms of data tagging. Data evolution and transformation are features unique to Iceberg tables. Iceberg adds tables to compute engines including Spark, Hive and Impala using a high-performance table format that works just like a SQL table. Also, the lineage support for Iceberg table is available. For example, when a Hive table is converted to Iceberg format, the lineage is displayed for the conversion process in Atlas UI.



Attention: Whenever a classification is applied on Iceberg entities, Tag-based policies are supported for Iceberg entities.

- Migration of Hive tables to Iceberg is achieved with the following:
 - Using in-place migration by running a Hive query with the ALTER TABLE statement and setting the table properties.
 - Executing CTAS command from Hive table to the Iceberg table.
- Schema evolution allows you to easily change a table's current schema to accommodate data that changes over time. Schema evolution enables you to update the schema that is used to write new data while maintaining backward compatibility with the schemas of your old data. Later the data can be read together assuming all of the data has one schema.
 - Iceberg tables supports the following schema evolution changes:
 - Add – add a new column to the table or to a nested struct
 - Drop– remove an existing column from the table or a nested struct
 - Rename– rename an existing column or field in a nested struct
 - Update– widen the type of a column, struct field, map key, map value, or list element
 - Reorder – change the order of columns or fields in a nested struct
 - Partition specification allows you to initiate queries faster by grouping similar rows together when writing.

As an example, queries for log entries from a logs table usually include a time range, like the following query for logs between 10 A.M. and 12 A.M.

```
SELECT level, message FROM logs
WHERE event_time BETWEEN '2018-12-01 10:00:00' AND '2018-12-01 12:00:00'
```

Configuring the logs table to partition by the date of event_time groups log events into files with the same event date. Iceberg keeps track of that date and uses it to skip files for other dates that do not have useful data.

- Partition evolution across Iceberg table partitioning can be updated in an existing table because queries do not reference partition values directly.

When you evolve a partition specification, the old data written with an earlier specification remains unchanged. New data is written using the new specification in a new layout. The metadata for each of the partition versions is stored separately.

Due to this nature of partition evolution, when you start writing queries, you get split planning. This is where each partition layout plans files separately using the filter it derives for that specific partition layout.

Related Information

[Using the Spark shell](#)

[Using the Impala shell](#)

[Apache Iceberg features](#)

Using the Spark shell

Using Spark, you can create an Iceberg table followed by schema evolution, partition specification, and partition evolution.

Before you begin

You must configure the Spark shell as such you have included the valid Spark runtime version.

Run the following command in your Spark shell to create a new Iceberg table

Procedure

1. `spark.sql("CREATE TABLE spark_catalog.default.sample_1 (id bigint COMMENT 'unique id', data string) USING iceberg");`
2. Navigate accordingly in the Atlas UI to view the changes.

The following images provide information about Iceberg table creation process.

Navigation: [Back To Results](#) | Search: sample_1 | User: admin

sample_1 (iceberg_table)

Classifications: [+](#)
Terms: [+](#)

Properties | Lineage | Relationships | Classifications | Audits | Tasks

Technical properties

columns (2): id, data

createTime: 03/27/2023 10:38:28 AM (IST)

db: default@primary

[User-defined properties](#) [Add](#)

[Labels](#) [Add](#)

[Business Metadata](#) [Add](#)

Navigation: [Back To Results](#) | Search: sample_1 | User: admin

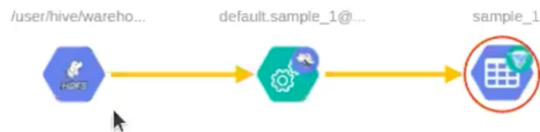
sample_1 (iceberg_table)

Classifications: [+](#)
Terms: [+](#)

Properties | **Lineage** | Relationships | Classifications | Audits | Tasks

○ Current Entity ⌛ In Progress → Lineage → Impact

[Refresh](#) [Screenshot](#) [Settings](#) [Filter](#) [Search](#) [Zoom In](#) [Zoom Out](#) [Reset](#)



Cloudera Atlas interface showing the lineage of an Iceberg table. The search bar contains "sample_1". The "Lineage" tab is selected, showing a flow from a Hive warehouse to a table named "sample_1". A detailed view of the "iceberg_table" is shown on the right.

Classifications: [+](#)

Terms: [+](#)

Properties **Lineage** Relationships Classifications Audits Tasks

○ Current Entity ⌛ In Progress → Lineage → Impact

iceberg_table

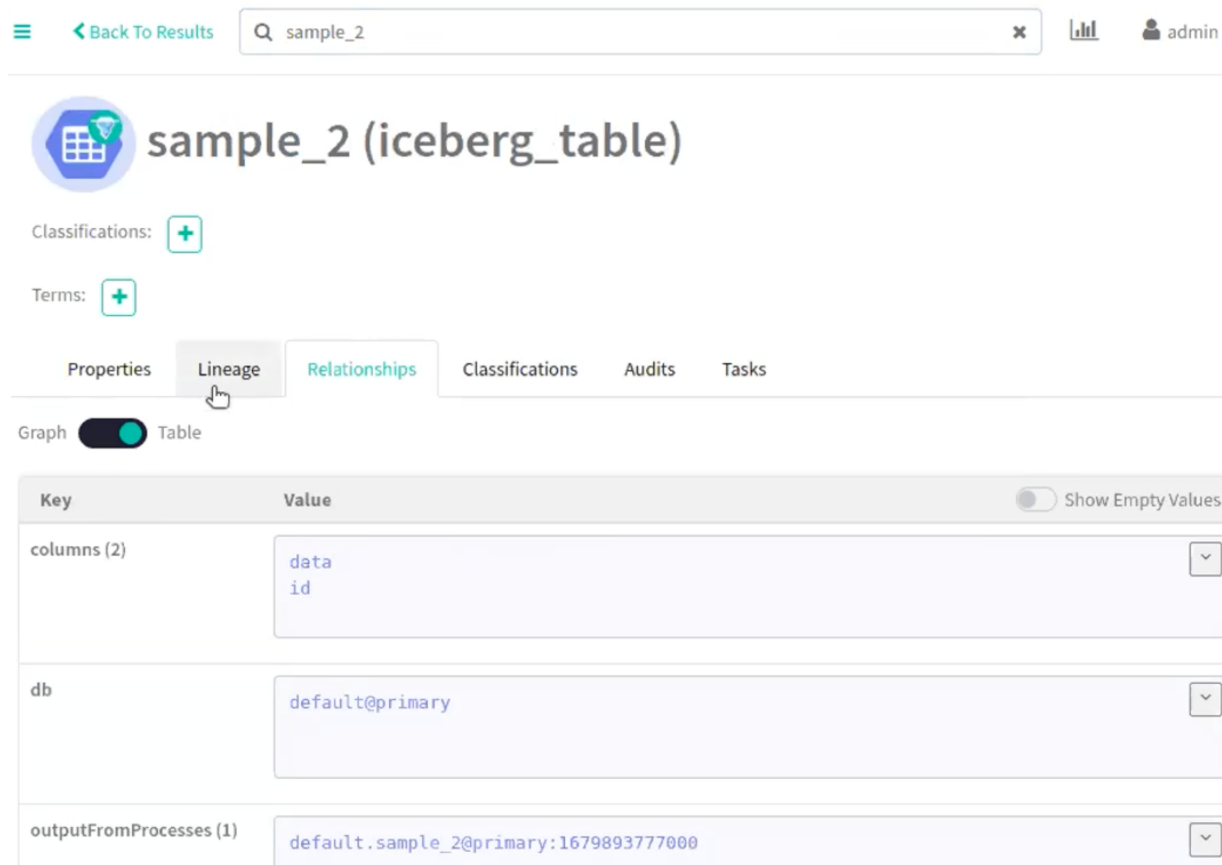
guid	3b7b9d10-d773-4281-a318-7d30c6886c9f
typeName	iceberg_table
name	sample_1
qualifiedName	default.sample_1@primary
owner	spark
createTime	1679893708000
status	ACTIVE
classifications	N/A

Lineage flow: /user/hive/wareho... → default.sample_1@... → sample_1

Run the following command in your Spark shell to create a Schema Evolution in a new table. For example - sample_2.

3. `spark.sql("CREATE TABLE spark_catalog.default.sample_2 (id bigint COMMENT 'unique id', data string) USING iceberg");`
4. Navigate accordingly in the Atlas UI to view the changes.

The following image provide information about Iceberg schema evolution process.



Cloudera Atlas interface showing the configuration for 'sample_2 (iceberg_table)'. The 'Lineage' tab is selected, and the 'Table' view is active. The configuration table shows the following details:

Key	Value
columns (2)	data id
db	default@primary
outputFromProcesses (1)	default.sample_2@primary:1679893777000


Run the following command in your Spark shell to include a column:

5. `spark.sql("ALTER TABLE spark_catalog.default.sample_2 ADD COLUMN (add_col_1 string)");`
6. Navigate accordingly in the Atlas UI to view the changes.

The following images provide information about Iceberg schema creation process.


sample_2 (iceberg_table)

Classifications: 

Terms: 

- Properties
- Lineage
- Relationships
- Classifications
- Audits
- Tasks

✓ Technical properties

columns (3) 

```
data
add_col_1
id
```

createTime 03/27/2023 10:39:37 AM (IST)

db 

```
default@primary
```

lastAccessTime 01/07/1970 05:29:24 PM (IST)

> User-defined properties 

> Labels 

> Business Metadata 

The screenshot displays the Cloudera Atlas interface for an Iceberg table. At the top, there is a navigation bar with a 'Back To Results' link, a search box for entities, and a user profile for 'admin'. Below this, a table header shows 'Users', 'Timestamp', and 'Actions'. The main content area shows the details for the table 'sample_2', which was updated by 'spark' on 03/27/2023 at 10:39:57 AM (IST). The table is categorized as an 'EXTERNAL_TABLE'.

The details are organized into two main sections: 'Technical properties' and 'Relationship properties'.

Technical properties:

- comment: N/A
- createTime: 03/27/2023 10:39:37 AM (IST)
- lastAccessTime: 01/07/1970 05:29:24 PM (IST)
- name: sample_2
- owner: spark
- parameters: { owner: "spark", }
- qualifiedName: default.sample_2@primary
- retention: 2147483647
- tableType: EXTERNAL_TABLE

Relationship properties:

- columns (3): id, data, add_col_1
- db: default
- partitionKeys: N/A
- sd: default.sample_2@primary_storage

Run the following command in your Spark shell to include the second column:

7. `spark.sql("ALTER TABLE spark_catalog.default.sample_2 ADD COLUMN (add_col_2 string)");`
8. Navigate accordingly in the Atlas UI to view the changes.

The following image provide information about Iceberg schema creation process.

Cloudera Atlas interface showing the details for an Iceberg table named `sample_2`.

The **Audits** tab is selected, displaying a log entry:

Users	Timestamp	Actions
spark	03/27/2023 10:40:27 AM (IST)	Entity Updated

Below the audit, the table's properties are shown:

- Technical properties:**
 - comment: N/A
 - createTime: 03/27/2023 10:39:37 AM (IST)
 - lastAccessTime: 01/07/1970 05:29:24 PM (IST)
 - name: sample_2
- Relationship properties:**
 - columns (4): id, data, add_col_1, add_col_2
 - db: default

Run the following command in your Spark shell to create a Partition Specification in a new table (`sample_3`):

9. `spark.sql("CREATE TABLE spark_catalog.default.sample_3 (id bigint,data string,category string,ts timestamp) USING iceberg PARTITIONED BY (bucket(16, id), days(ts), category)");`
10. Navigate accordingly in the Atlas UI to view the changes.

The following images provide information about Iceberg partition specification process.

sample_3 (iceberg_table)

Classifications: [+](#)

Terms: [+](#)

Properties

Lineage

Relationships

Classifications

Audits

Tasks

Technical properties

columns (4)

ts
id
category

createTime 03/27/2023 10:41:24 AM (IST)

db default@primary

lastAccessTime 01/07/1970 05:31:11 PM (IST)

[User-defined properties](#) [Add](#)

[Labels](#) [Add](#)

[Business Metadata](#) [Add](#)

createTime	03/27/2023 10:41:24 AM (IST)
db	default@primary
lastAccessTime	01/07/1970 05:31:11 PM (IST)
name	sample_3
owner	spark
parameters	{ owner: "spark", "current-schema": "
partitionSpec (3)	category, id_bucket, ts_day
qualifiedName	default.sample_3@primary
retention	2147483647
sd	default.sample_3@primary_stora

- Run the following command in your Spark shell to create a Partition Evolution in a new table (sample_3):
11. `spark.sql("ALTER TABLE spark_catalog.default.sample_3 ADD PARTITION FIELD years(ts)");`
 12. Navigate accordingly in the Atlas UI to view the changes.

The following images provide information about Iceberg partition evolution process.

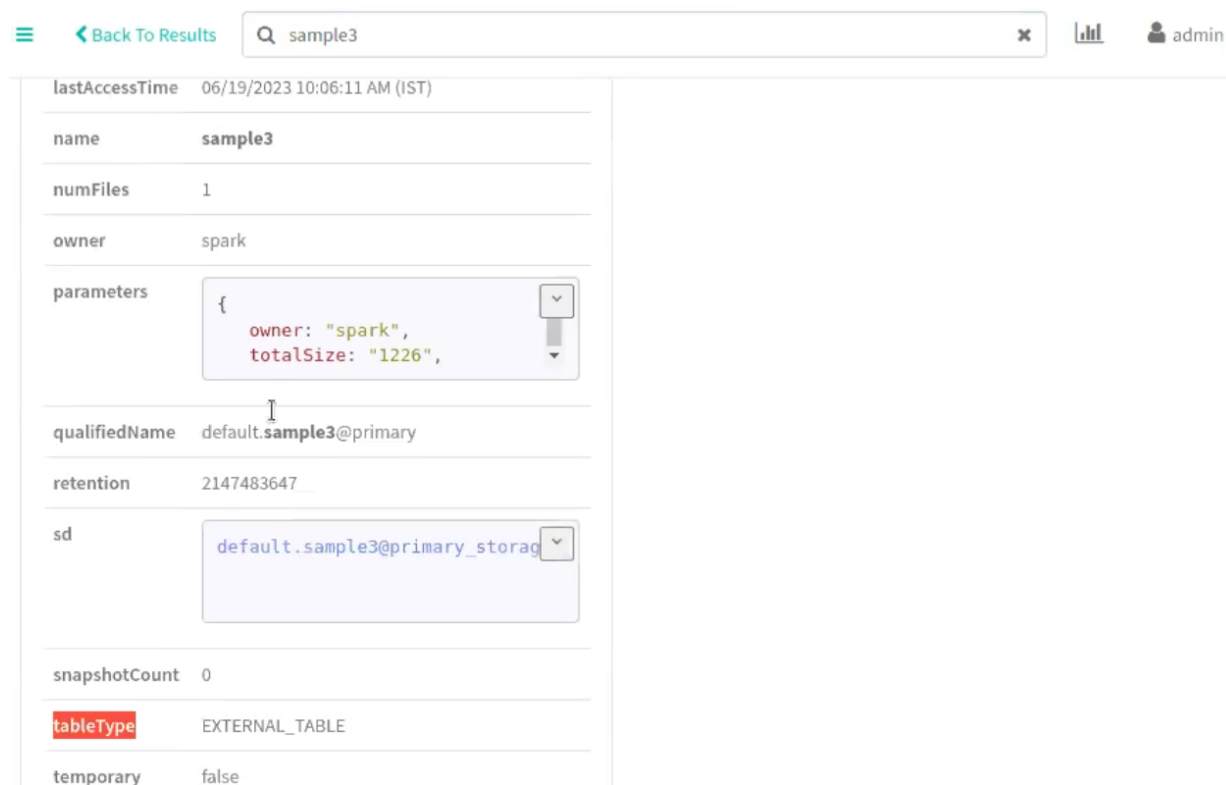
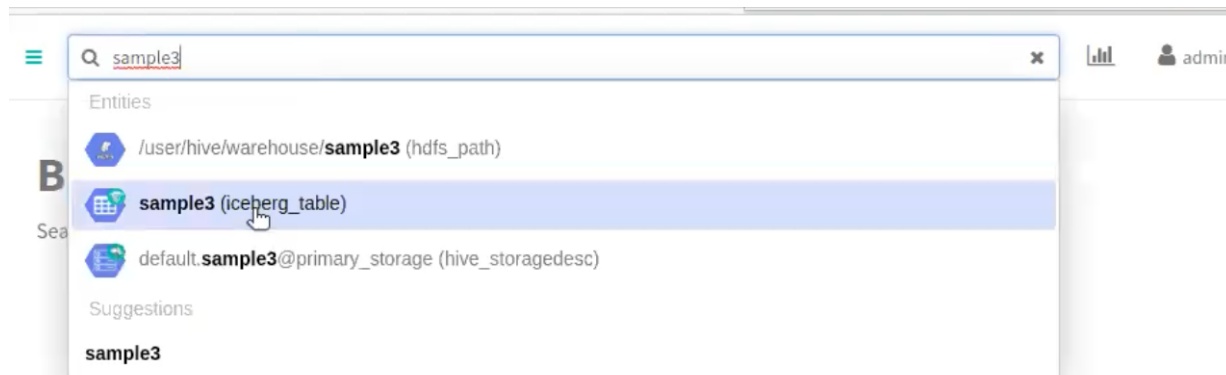
name	sample_3
owner	spark
parameters	<pre>{ owner: "spark", previous_metadata_location:</pre>
partitionSpec (4)	<pre>category, ts_year, id_bucket, ts_day</pre>
qualifiedName	default.sample_3@primary
retention	2147483647
sd	default.sample_3@primary_stora
tableType	EXTERNAL_TABLE
temporary	false
typeName	iceberg_table

Displaying Snapshot attributes

Run the following command to display relevant snapshot table parameters as attributes in Atlas. For example: Existing snapshot ID, current snapshot timestamp, snapshot count, number of files and related attributes.

```
spark.sql("CREATE TABLE spark_catalog.default.sample3 ( id int, data string) USING iceberg");
```

```
spark.sql("INSERT INTO default.sample3 VALUES (1, 'TEST')");
```



Run the following command to scale up the snapshot count value.

```
spark.sql("INSERT INTO default.sample3 VALUES (1, 'TEST')");
```

The latest Snapshot ID that Iceberg points to along with the Snapshot timestamp and Snapshot count are updated respectively.

← Back To Results Search entities admin

columns (2) id data

createTime 06/19/2023 10:06:11 AM (IST)

currentSnapshotId 7368381093036867296

currentSnapshotTsMs 1687149412435

db default

lastAccessTime 06/19/2023 10:06:11 AM (IST)

name sample3

numFiles 1

owner spark

Labels Add

Business Metadata Add

← Back To Results Search entities admin

name sample3

numFiles 1

owner spark

parameters {
owner: "spark",
previous_metadata_location: "..."

qualifiedName default.sample3@primary

retention 2147483647

sd default.sample3@primary_s
range

snapshotCount 1

tableType EXTERNAL_TABLE

temporary false

Click on the parameters field to display the details pertaining to the snapshot attributes.

Support for data compaction

Data Compaction is the process of taking several small files and rewriting them into fewer larger files to speed up queries. When performing compaction on an Iceberg table, execute the `rewriteDataFiles` procedure, optionally specifying a filter of which files to rewrite and the desired size of the resulting files.


As an example, in an Atlas instance consider that the number of files and snapshot count are 9.

Run the following command to perform data compaction

```
spark.sql("CALL spark_catalog.system.rewrite_data_files('default.sample3'),show();
```

```
+-----+-----+
|rewritten_data_files_count|added_data_files_count|
+-----+-----+
|                9|                1|
+-----+-----+
```

The count for the number of rewritten data files is compacted from a total count of 9 to 1.

 [← Back To Results](#)

currentSnapshotTsMs	1687149580886
db	<input type="text" value="default"/>
lastAccessTime	06/19/2023 10:06:11 AM (IST)
name	sample3
numFiles	1
owner	spark
parameters	<pre>{ owner: "spark", previous_metadata_loca</pre>
qualifiedName	default.sample3@primary
retention	2147483647
sd	<input type="text" value="default.sample3@primary_s
rage"/>

snapshotCount 10

Support for metadata rewrite attributes

Iceberg uses metadata in its manifest list and manifest files speed up query planning and to prune unnecessary data files. You can rewrite manifests for a table to optimize the plan to scan the data.

Run the following command to rewrite manifests on a sample table 3.

```
spark.sql("CALL spark_catalog.system.rewrite_manifests('default.sample3').show();
```

```
+-----+-----+
|rewritten_manifests_count|added_manifests_count|
+-----+-----+
|                10   |                1   |
+-----+-----+
```

The count for table manifested data is rewritten from 10 to 1.

← Back To Results <input type="text" value="Search entities"/>	
lastAccessTime	06/19/2023 10:06:11 AM (IST)
manifestsCreated	1
manifestsKept	0
manifestsReplaced	10
name	sample3
numFiles	1
owner	spark

The process is completed.

Related Information

[Using the Impala shell](#)

Using the Impala shell

Using Impala, you can create an Iceberg table followed by Schema evolution, partition specification, partition evolution and CTAS operation.

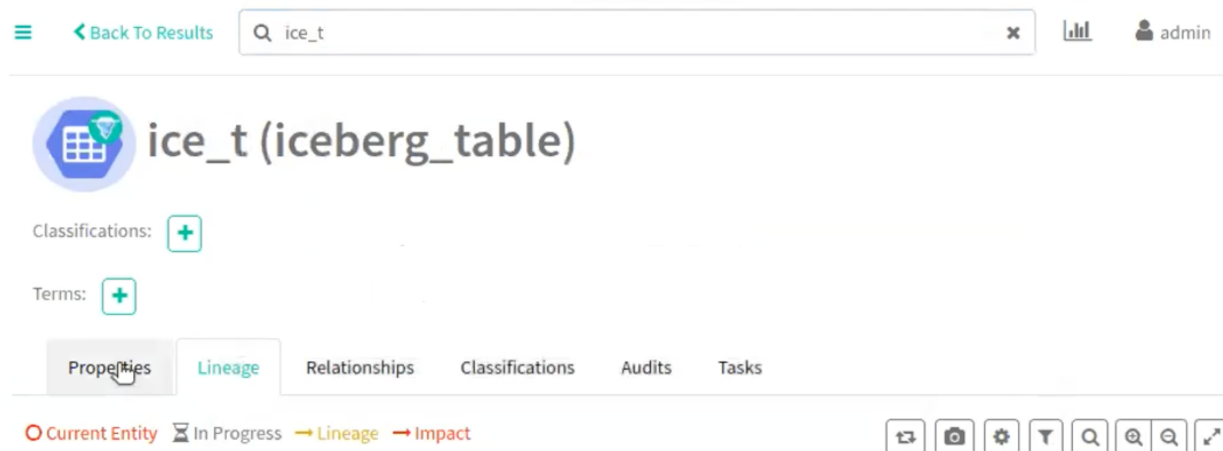
Before you begin

Run the following command in your Impala shell to create a new Iceberg table

Procedure

1. CREATE TABLE ice_t (i INT) STORED AS ICEBERG;
2. Navigate accordingly in the Atlas UI to view the changes.

The following images provide information about Iceberg table creation process.



Run the following command in your Impala shell to create a schema evolution:

3. CREATE TABLE ice_t_2 (i INT) STORED AS ICEBERG;

ice_t_2 (iceberg_table)

Classifications: +

Terms: +

Properties Lineage Relationships Classifications Audits Tasks

Technical properties

columns (1) i

createTime 03/27/2023 10:48:21 AM (IST)

db default@primary

User-defined properties Add

Labels Add

Business Metadata Add

Run the following command in your Impala shell to add a column to the existing table (ice_t_2):

- alter table ice_t_2 ADD COLUMNS (add_col_1 string);



[← Back To Results](#)



ice_t_2 (iceberg_tab

Classifications:



Terms:



Properties

Lineage

Relationships

Classification

Technical properties

columns (2)

i
add_col_1

createTime

03/27/2023 10:48:21 AM (IST)

db

default@primary

Run the following command in your Impala shell to create a partition specification.

5. CREATE TABLE ice_part_spec (s string , b string) PARTITIONED BY SPEC (truncate(3, s)) STORED AS ICEBERG ;

The screenshot shows the Cloudera Atlas interface for the table 'ice_part_spec (iceberg_table)'. The 'Audits' tab is selected, displaying a table with the following data:

Users	Timestamp	Actions
impala	03/27/2023 10:49:28 AM (IST)	Entity Created

Showing 1 records From 1 - 25. Page Limit: 25

The screenshot shows the Cloudera Atlas interface for the table 'ice_part_spec (iceberg_table)'. The 'Properties' tab is selected, displaying the following properties:

lastAccessTime	01/07/1970 05:39:15 PM (IST)
name	ice_part_spec
owner	impala
parameters	{ "external.table.purge": "TRUE", }
partitionSpec (1)	s.trunc
qualifiedName	default.ice_part_spec@primary
retention	2147483647
sd	default.ice_part_spec@primary_orage
tableType	EXTERNAL_TABLE
temporary	false
typeName	iceberg_table

Run the following command in your Impala shell to create a partition evolution.

6. CREATE TABLE ice_p (i INT, d DATE, s STRING, t TIMESTAMP) PARTITIONED BY SPEC (BUCKET(5, i), MONTH(d), TRUNCATE(3, s), HOUR(t)) STORED AS ICEBERG;

The screenshot shows the Cloudera Atlas interface for the table 'ice_p'. The search bar at the top contains 'ice_p'. The table details are as follows:

- lastAccessTime:** 01/07/1970 05:40:00 PM (IST)
- name:** ice_p
- owner:** impala
- parameters:** { "external.table.purge": "TRUE", }
- partitionSpec (4):** s_trunc, t_hour, i_bucket, d_month
- qualifiedName:** default.ice_p@primary
- retention:** 2147483647
- sd:** default.ice_p@primary_storage
- tableType:** EXTERNAL_TABLE

Run the following command in your Impala shell to modify the partition specification

7. ALTER TABLE ice_p SET PARTITION SPEC (VOID(i), VOID(d), TRUNCATE(3, s), HOUR(t), i);

[← Back To Results](#)

Search entities

lastAccessTime 01/07/1970 05:40:00 PM (IST)

name ice_p

owner impala

parameters

```
{  
  "previous_metadata_location":  
  "hdfs://atlas-
```

partitionSpec
(5)

```
d_null, s_trunc, t_hour, i_nul  
i
```

qualifiedName default.ice_p@primary

retention 2147483647

sd

```
default.ice_p@primary_storage
```

Run the following commands in your Impala shell to create the contents of one table (ice_t_3) to another table (ice_t_4).

8. `CREATE TABLE ice_t_3 (i INT) STORED AS ICEBERG;`
9. `CREATE TABLE ice_t_4 STORED AS ICEBERG as select * from ice_t_3;`

☰ < Back To Results






partitionSpec (5)

✓ impala

Name: ice_p

Q ice_t_4

Entities

-  /user/hive/warehouse/ice_t_4
-  default.ice_t_4@primary:-100
-  **ice_t_4 (iceberg_table)**
-  default.ice_t_4@primary_stor
-  default.ice_t_4@primary:1679

Suggestions

ice_t_4

✓ Technical properties

comment	N/
createTime	03/27/2023 10:50:13 AM (IS
lastAccessTime	01/07/1970 05:40:00 PM (IS
name	ice_

← Back To Results admin

ice_t_4 (iceberg_table)

Classifications: +
Terms: +

Properties Lineage Relationships Classifications Audits Tasks

○ Current Entity ⌛ In Progress → Lineage → Impact



The process is completed.

Related Information

[Using the Spark shell](#)