

..

Using Apache Iceberg

Date published: 2023-01-24

Date modified: 2023-01-24

CLOUDERA

Legal Notice

© Cloudera Inc. 2025. 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

Apache Iceberg features.....	5
Alter table feature.....	5
Create table feature.....	6
Create table as select feature.....	8
Create partitioned table as select feature.....	8
Create table ... like feature.....	9
Describe table metadata feature.....	9
Drop table feature.....	10
Expire snapshots feature.....	10
Insert table data feature.....	11
Load data inpath feature.....	12
Load or replace partition data feature.....	13
Merge feature.....	13
Migrate Hive table to Iceberg feature.....	14
Changing the metadata location.....	14
Partition evolution feature.....	15
Partition transform feature.....	15
Query metadata tables feature.....	17
Rollback table feature.....	17
Schema evolution feature.....	18
Schema inference feature.....	19
Time travel feature.....	20
Truncate table feature.....	21
Update and delete data features.....	21
 Best practices for Iceberg in CDP.....	 22
 Feature limitations.....	 22
 Prerequisites.....	 22
 Accessing Iceberg tables.....	 23
Editing a policy to access Iceberg files.....	25
Creating a policy to query an Iceberg table.....	27
 Creating an Iceberg table.....	 29
 Creating an Iceberg partitioned table.....	 29
 Expiring snapshots.....	 30

Inserting data into a table.....	31
Migrating a Hive table to Iceberg.....	31
Selecting an Iceberg table.....	32
Running time travel queries.....	33
Updating an Iceberg partition.....	33
Test driving Iceberg from Impala.....	34
Test driving Iceberg from Hive.....	36
Hive demo data.....	38
Iceberg data types.....	40
Iceberg table properties.....	41

Apache Iceberg features

You can quickly build on your past experience with SQL to analyze Iceberg tables.

Iceberg features include security and governance, and other Cloudera Data Platform benefits, described in [Apache Iceberg in CDP](#).

Impala queries are table-format agnostic. For example, Impala options are supported in queries of Iceberg tables from Impala. You can run nested, correlated, analytic queries on all supported table types.

Most Hive queries are table-format agnostic. This documentation does not attempt to show every possible query supported from Hive and Impala. For example, creating a view is not covered, but you can create a view of any table from Hive or Impala using SQL syntax. The following topics show many examples of how to run queries on Iceberg tables from Hive and Impala.

Alter table feature

In Hive or Impala, you can use ALTER TABLE to set table properties. From Impala, you can use ALTER TABLE to rename a table, to change the table owner, or to change the role of the table owner. From Hive, you can alter the metadata location of the table if the new metadata does not belong to another table; otherwise, an exception occurs.

You can convert an Iceberg v1 table to v2 by setting a table property as follows: 'format-version' = '2'.

Hive or Impala syntax

```
ALTER TABLE table_name SET TBLPROPERTIES table_properties;
```

- table_properties

A list of properties and values using the following syntax:

```
('key' = 'value', 'key' = 'value', ... )
```

Impala syntax

```
ALTER TABLE table_name RENAME TO new_table_name;
```

```
ALTER TABLE table_name SET OWNER USER user_name;
```

```
ALTER TABLE table_name SET OWNER ROLE role_name;
```

Hive example

```
ALTER TABLE test_table SET TBLPROPERTIES('metadata_location'='hdfs://ice_table/metadata/v1.metadata.json');
      ALTER TABLE test_table2 SET TBLPROPERTIES('format-version' = '2');
```

Impala examples

```
ALTER TABLE t1 RENAME TO t2;
```

```
ALTER TABLE ice_table1 set OWNER USER john_doe;
```

```
ALTER TABLE ice_table2 set OWNER ROLE some_role;
```

```
ALTER TABLE ice_8 SET TBLPROPERTIES ('read.split.target-size'='268435456');
```

```
ALTER TABLE ice_table3 SET TBLPROPERTIES('format-version' = '2');
```

Create table feature

You use CREATE TABLE from Impala or CREATE EXTERNAL TABLE from Hive to create an external table in Iceberg. You learn the subtle differences in these features for creating Iceberg tables from Hive and Impala. You also learn about partitioning.

Hive and Impala handle external table creation a little differently, and that extends to creating tables in Iceberg. By default, Iceberg tables you create are v1. To create an Iceberg v2 table from Hive or Impala, you need to set a table property as follows: 'format-version' = '2'.

Iceberg table creation from Hive

From Hive, CREATE EXTERNAL TABLE is recommended to create an Iceberg table in CDP.

When you use the EXTERNAL keyword to create the Iceberg table, by default only the schema is dropped when you drop the table. The actual data is not purged. Conversely, if you do not use EXTERNAL, by default the schema and actual data is purged. You can override the default behavior. For more information, see the Drop table feature.

From Hive, you can create a table that reuses existing metadata by setting the metadata_location table property to the object store path of the metadata. The operation skips generation of new metadata and re-registers the existing metadata. Use the following syntax:

```
CREATE EXTERNAL TABLE ice_fm_hive (i int) STORED BY ICEBERG TBLPROPERTIES ('
metadata_location'='<object store or file system path>')
```

See examples below.

Iceberg table creation from Impala

From Impala, CREATE TABLE is recommended to create an Iceberg table in CDP. Impala creates the Iceberg table metadata in the metastore and also initializes the actual Iceberg table data in the object store.

The difference between Hive and Impala with regard to creating an Iceberg table is related to Impala compatibility with Kudu, HBase, and other tables. For more information, see the Apache documentation, ["Using Impala with Iceberg Tables"](#).

Metadata storage of Iceberg tables

When you create an Iceberg table using CREATE EXTERNAL TABLE in Hive or using CREATE TABLE in Impala, HiveCatalog creates an HMS table and also stores some metadata about the table on your object store, such as S3. Creating an Iceberg table generates a metadata.json file, but not a snapshot. In the metadata.json, the snapshot-id of a new table is -1. Inserting, deleting, or updating table data generates a snapshot. The Iceberg metadata files and data files are stored in the table directory under the warehouse folder. Any optional partition data is converted into Iceberg partitions instead of creating partitions in the Hive Metastore, thereby removing the bottleneck.

To create an Iceberg table from Hive or from Impala, you associate the Iceberg storage handler with the table using one of the following clauses, respectively:

- Hive: STORED BY ICEBERG
- Impala: STORED AS ICEBERG or STORED BY ICEBERG

Supported file formats

You can write Iceberg tables in the following formats:

- From Hive: Parquet (default), Avro, ORC

- From Impala: Parquet

Impala supports writing Iceberg tables in only Parquet format. Impala does not support defining both file format and storage engine. For example, `CREATE TABLE tbl ... STORED AS PARQUET STORED BY ICEBERG` works from Hive, but not from Impala.

You can read Iceberg tables in the following formats:

- From Hive: Parquet, Avro, ORC
- From Impala: Parquet, Avro, ORC



Note: Reading Iceberg tables in Avro format from Impala is available as a technical preview. Cloudera recommends that you use this feature in test and development environments. It is not recommended for production deployments.

Hive syntax

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type, ... )]
  [PARTITIONED BY [SPEC]([col_name][, spec(value)][, spec(value)]...)]
  [STORED AS file_format]
  STORED BY ICEBERG
  [TBLPROPERTIES ('key'='value', 'key'='value', ...)]
```

Impala syntax

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type, ... )]
  [PARTITIONED BY [SPEC]([col_name][, spec(value)][, spec(value)]...)]
  STORED {AS | BY} ICEBERG
  [TBLPROPERTIES (property_name=property_value, ...)]
```

Hive examples

```
CREATE EXTERNAL TABLE ice_1 (i INT, t TIMESTAMP, j BIGINT) STORED BY ICEBERG
;
  CREATE EXTERNAL TABLE ice_2 (i INT, t TIMESTAMP) PARTITIONED BY (j B
IGINT) STORED BY ICEBERG;
  CREATE EXTERNAL TABLE ice_4 (i int) STORED AS ORC STORED BY ICEBERG;
  CREATE EXTERNAL TABLE ice_5 (i int) STORED BY ICEBERG TBLPROPERTIES (
'metadata_location'='hdfs://ice_table/metadata/v1.metadata.json')
  CREATE EXTERNAL TABLE ice_6 (i int) STORED AS ORC STORED BY ICEBERG
TBLPROPERTIES ('format-version' = '2');
```

```
CREATE EXTERNAL TABLE ice_1 (i INT, t TIMESTAMP, j BIGINT) STORED BY ICEBERG
;
  CREATE EXTERNAL TABLE ice_2 (i INT, t TIMESTAMP) PARTITIONED BY (j B
IGINT) STORED BY ICEBERG;
  CREATE EXTERNAL TABLE ice_4 (i int) STORED AS ORC STORED BY ICEBERG;
  CREATE EXTERNAL TABLE ice_5 (i int) STORED BY ICEBERG TBLPROPERTIES (
'metadata_location'='hdfs://ice_table/metadata/v1.metadata.json')
  CREATE EXTERNAL TABLE ice_6 (i int) STORED AS ORC STORED BY ICEBERG
TBLPROPERTIES ('format-version' = '2');
```

Impala examples

```
CREATE TABLE ice_7 (i INT, t TIMESTAMP, j BIGINT) STORED BY ICEBERG; //creat
es only the schema
```

```
CREATE TABLE ice_8 (i INT, t TIMESTAMP) PARTITIONED BY (j BIGINT) STORED BY
ICEBERG; //creates schema and initializes data
CREATE TABLE ice_v2 (i INT, t TIMESTAMP) PARTITIONED BY (j BIGINT) STORED BY
ICEBERG TBLPROPERTIES ('format-version' = '2'); //creates a v2 table
```

Related Information

[Drop table feature](#)

[Partition transform feature](#)

Create table as select feature

You can create an Iceberg table based on an existing Hive or Impala table.

The create table as select (CTAS) query can optionally include a partitioning spec for the table being created.

Hive examples

```
CREATE EXTERNAL TABLE ctas STORED BY ICEBERG AS SELECT i, t, j FROM ice_1;
```

Impala examples

```
CREATE TABLE ctas STORED BY ICEBERG AS SELECT i, b FROM ice_11;
```

Create partitioned table as select feature

You can create a partitioned Iceberg table by selecting another table. You see an example of how to use `PARTITIONED BY` and `TBLPROPERTIES` to declare the partition spec and table properties for the new table.

You see an example of using a [partition transform](#) with the `PARTITIONED BY SPEC` clause.

The newly created table does not inherit the partition spec and table properties from the source table in `SELECT`. The Iceberg table and the corresponding Hive table is created at the beginning of the query execution. The data is inserted / committed when the query finishes. So for a transient period the table exists but contains no data.

Hive syntax

```
CREATE [EXTERNAL] TABLE prod.db.sample
  USING iceberg
  PARTITIONED BY (part)
  TBLPROPERTIES ('key'='value')
  AS SELECT ...
```

Hive examples

```
CREATE EXTERNAL TABLE ctas STORED BY ICEBERG AS SELECT i, t, j FROM ice_1;

CREATE EXTERNAL TABLE ctas_part PARTITIONED BY(z) STORED BY ICEBERG TBLPROPE
RTIES ('format-version'='2')
AS SELECT x, ts, z FROM t;
CREATE EXTERNAL TABLE ctas_part_spec PARTITIONED BY SPEC (month(d)) STORED
BY ICEBERG TBLPROPERTIES ('format-version'='2')
AS SELECT x, ts, d FROM source_t;
```


Impala examples

```
CREATE TABLE ctas STORED BY ICEBERG AS SELECT i, b FROM ice_11;

CREATE TABLE ctas_part PARTITIONED BY(b) STORED BY ICEBERG AS SELECT i, s, b
FROM ice_11;
CREATE TABLE ctas_part_spec PARTITIONED BY SPEC (month(d)) STORED BY ICEBERG
TBLPROPERTIES ('format-version'='2')
AS SELECT x, ts, d FROM source_t;
```

Create table ... like feature

You learn by example how to create an empty table based on another table.

From Hive or Impala, you can create an Iceberg table schema based on another table. The table contains no data. The table properties of the original table are carried over to the new table definition. The following examples show how to use this feature:

Hive example

```
CREATE EXTERNAL TABLE target LIKE source STORED BY ICEBERG;
```

Impala example

```
CREATE TABLE target LIKE source STORED BY ICEBERG;
```

Describe table metadata feature

You can use certain Hive and Impala show and describe commands to get information about table metadata. You can also query metadata tables.

The following table lists SHOW and DESCRIBE commands supported by Hive and Impala.

Command Syntax	Description	SQL Engine Support
SHOW CREATE TABLE table_name	Reveals the schema that created the table.	Hive and Impala
SHOW FILES IN table_name	Lists the files related to the table.	Impala
SHOW PARTITIONS table_name	Returns the Iceberg partition spec, just the column information, not actual partitions or files.	Impala
DESCRIBE [EXTENDED] table_name	The optional EXTENDED shows all the metadata for the table in Thrift serialized form, which is useful for debugging.	Hive and Impala
DESCRIBE [FORMATTED] table_name	The optional FORMATTED shows the metadata in tabular format.	Hive
DESCRIBE HISTORY table_name [BETWEEN timestamp1 AND timestamp2]	Optionally limits the output history to a period of time.	Impala

Hive example

```
DESCRIBE t;
```

Hive output includes the following information:

col_name	data_type	comment
x	int	
y	int	
	NULL	NULL
# Partition Transform Information	NULL	NULL
# col_name	transform_type	NULL
y	IDENTITY	NULL

The output of DESCRIBE HISTORY includes the following columns about the snapshot. The first three are self-explanatory. The is_current_ancestor column value is TRUE if the snapshot is the ancestor of the table:

- creation_time
- snapshot_id
- parent_id
- is_current_ancestor

Impala examples

```
DESCRIBE HISTORY ice_t FROM '2022-01-04 10:00:00';
DESCRIBE HISTORY ice_t FROM now() - interval 5 days;
DESCRIBE HISTORY ice_t BETWEEN '2022-01-04 10:00:00' AND '2022-01-05 10:00:00';
```

Drop table feature

The syntax you use to create the table determines the default behavior when you drop the Iceberg table from Hive or Impala.

If you use CREATE TABLE, the external.table.purge flag is set to true. When the table is dropped, the contents of the table directory (actual data) are removed. If you use CREATE EXTERNAL TABLE from Hive, the external.table.purge flag is set to false. Dropping a table purges the schema only. The actual data is not removed. You can explicitly set the external.table.purge property to true to drop the data as well as the schema.

To prevent data loss during migration of a table to Iceberg, do not drop or move the table during migration. Exception: If you set the table property 'external.table.purge'='FALSE', no data loss occurs if you drop the table.

Hive or Impala syntax

```
DROP TABLE [IF EXISTS] table_name
```

Hive or Impala example

```
ALTER TABLE t SET TBLPROPERTIES('external.table.purge'='true');
DROP TABLE t;
```

Related Information

[Create table feature](#)

Expire snapshots feature

You can expire snapshots that Iceberg generates when you create or modify a table. During the lifetime of a table the number of snapshots of the table accumulate. You learn how to remove snapshots you no longer need.

You should periodically expire snapshots to delete data files that are no longer needed, and to reduce the size of table metadata. Each write to an Iceberg table from Hive creates a new snapshot, or version, of a table. Snapshots can be used for time-travel queries, or for rollbacks. The table can be rolled back to any valid snapshot. Snapshots accumulate until they are expired by the `expire_snapshots` operation.

You use the following syntax to expire snapshots older than a timestamp or timestamp expression:

Hive or Impala syntax

```
ALTER TABLE ... EXECUTE expire_snapshots(<timestamp expression>)
```

Hive or Impala example

The first example removes snapshots having a timestamp older than August 15, 2022 1:50 pm. The second example removes snapshots from 10 days ago and before.

```
ALTER TABLE ice_11 EXECUTE expire_snapshots('2022-08-15 13:50:00');
ALTER TABLE ice_t EXECUTE expire_snapshots(now() - interval 10 days);
```

Preventing snapshot expiration

You can prevent expiration of recent snapshots by configuring the `history.expire.min-snapshots-to-keep` table property. You can use the alter table feature to set a property.

Table data and orphan maintenance

The contents of the table directory (actual data) might, or might not, be removed when you drop the table. An orphan data file can remain when you drop an Iceberg table, depending on the `external.table.purge` flag table property. An orphaned data file is one that has contents in the table directory, but no snapshot.

Expiring a snapshot does not remove old metadata files by default. You must clean up metadata files using `write.metadata.delete-after-commit.enabled=true` and `write.metadata.previous-versions-max` table properties. For more information, see "Iceberg table properties" below. Setting this property controls automatic metadata file removal after metadata operations, such as expiring snapshots or inserting data.

Related Information

[Iceberg table properties](#)

Insert table data feature

From Hive and Impala, you can insert data into Iceberg tables using the standard `INSERT INTO` a single table. `INSERT` statements work for V1 and V2 tables.

You can replace data in the table with the result of a query. To replace data, Hive and Impala dynamically overwrite partitions that have rows returned by the `SELECT` query. Partitions that do not have rows returned by the `SELECT` query, are not replaced. Using `INSERT OVERWRITE` on tables that use the `BUCKET` partition transform is not recommended. Results are unpredictable because dynamic overwrite behavior would be too random in this case.

From Hive, CDP also supports inserting into multiple tables as a technical preview; however, this operation is not atomic, so data consistency of Iceberg tables is equivalent to that of Hive external tables. Changes within a single table will remain atomic.

Inserting, deleting, or updating table data generates a snapshot. A new snapshot corresponds to a new manifest list. Manifest lists are named `snap-*.avro`.

Iceberg specification defines sort orders. At this point, Hive doesn't support defining sort orders. But if there are sort orders defined by using other engines Hive can utilize them on write operations. For more information about sorting, see [sort orders specification](#).

Hive or Impala syntax

```
INSERT INTO TABLE tablename VALUES values_row [, values_row ...]

INSERT INTO TABLE tablename1 select_statement1 FROM tablename2

INSERT OVERWRITE TABLE tablename1 select_statement1 FROM tablename2
```

Hive or Impala examples

```
CREATE TABLE ice_10 (i INT, s STRING, b BOOLEAN) STORED BY ICEBERG;
INSERT INTO ice_10 VALUES (1, 'asf', true);
CREATE TABLE ice_11 (i INT, s STRING, b BOOLEAN) STORED BY ICEBERG;
INSERT INTO ice_11 VALUES (2, 'apache', false);
INSERT INTO ice_11 SELECT * FROM ice_10;
SELECT * FROM ice_11;
INSERT OVERWRITE ice_11 SELECT * FROM ice_10;
```

Hive example

```
FROM customers
  INSERT INTO target1 SELECT customer_id, first_name;
  INSERT INTO target2 SELECT last_name, customer_id;
```

Load data inpath feature

From Impala, you can load Parquet or ORC data from a file in a directory on your file system or object store into an Iceberg table. You might need to set the mem_limit or pool configuration (max-query-mem-limit, min-query-mem-limit) to accommodate the load.

Impala syntax

```
LOAD DATA INPATH '<path to file>' INTO table t;
```

Impala example

In this example, you create a table using the LIKE clause to point to a table stored as Parquet. This is required for Iceberg to infer the schema. You also load data stored as ORC.

```
CREATE TABLE test_iceberg LIKE my_parquet_table STORED AS ICEBERG;
SET MEM_LIMIT=1MB;
```

```
LOAD DATA INPATH '/tmp/some_db/parquet_files/'
INTO TABLE iceberg_tbl;
```

```
LOAD DATA INPATH '/tmp/some_db/orc_files/'
INTO TABLE iceberg2_tbl;
```

```
CREATE TABLE test_iceberg LIKE my_parquet_table STORED AS ICEBERG;
SET MEM_LIMIT=1MB;
```

```
LOAD DATA INPATH '/tmp/some_db/parquet_files/' INTO TABLE iceberg_tbl;
```

```
LOAD DATA INPATH '/tmp/some_db/orc_files/' INTO TABLE iceberg2_tbl;
```

Load or replace partition data feature

There is no difference in the way you insert data into a partitioned or unpartitioned Iceberg table.

Working with partitions is easy because you write the query in the same way for the following operations:

- Insert into, or replace, an unpartitioned table
- Insert into, or replace, an identity partitioned table
- Insert into, or replace, a transform-partitioned table

Do not use INSERT OVERWRITE on tables that went through partition evolution. Truncate such tables first, and then INSERT the tables.

Impala example

```
CREATE TABLE ice_12 (i int, s string, t timestamp, t2 timestamp) STORED BY ICEBERG;
```

Hive or Impala examples

```
INSERT INTO ice_12 VALUES (42, 'impala', now(), to_date(now()));
INSERT OVERWRITE ice_t VALUES (42, 'impala', now(), to_date(now()));
```

Merge feature

You can perform actions on an Iceberg table based on the results of a join with a v2 Iceberg table.

Hive syntax

```
MERGE INTO <target table> AS T USING <source expression/table> AS S
ON <boolean expression1>
WHEN MATCHED [AND <boolean expression2>] THEN UPDATE SET <set clause list>
WHEN MATCHED [AND <boolean expression3>] THEN DELETE
WHEN NOT MATCHED [AND <boolean expression4>] THEN INSERT VALUES <value list>
```

Hive example

Use the MERGE INTO statement to update an Iceberg table based on a staging table:

```
MERGE INTO customer USING (SELECT * FROM new_customer_stage) sub ON sub.id =
customer.id
  WHEN MATCHED THEN UPDATE SET name = sub.name, state = sub.new_state
  WHEN NOT MATCHED THEN INSERT VALUES (sub.id, sub.name, sub.state);
```

Create an Iceberg table and merge it with a non-Iceberg table.

```
create external table target_ice(a int, b string, c int) partitioned by spec
(bucket(16, a), truncate(3, b)) stored by iceberg stored as orc tblproperties
('format-version'='2');
create table source(a int, b string, c int);
...

merge into target_ice as t using source src ON t.a = src.a
when matched and t.a > 100 THEN DELETE
when matched then update set b = 'Merged', c = t.c + 10
when not matched then insert values (src.a, src.b, src.c);
```

Migrate Hive table to Iceberg feature

CDP supports table migration from Hive tables to Iceberg tables using ALTER TABLE to set the table properties. You set the storage_handler table property to the Iceberg storage handler.

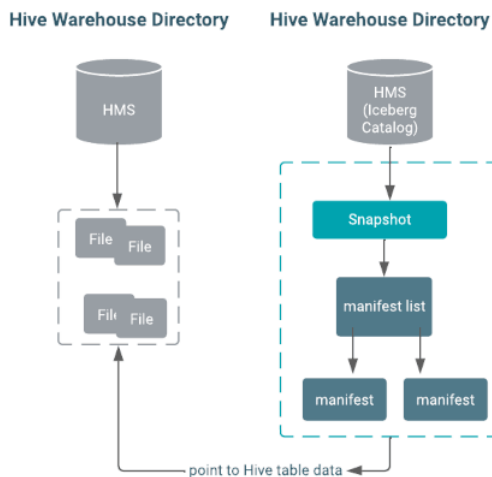
Impala does not support table migration in this release. The topic "[Test driving Iceberg from Impala](#)" shows how to create Iceberg tables from Impala tables.



Note: Do not drop or move the old table during a migration operation. Doing so will delete the data files of the old and new tables. Exception: If you set the table property 'external.table.purge'='FALSE', no data loss occurs when you drop the table.

In-place table migration process

In-place table migration saves time generating Iceberg tables. There is no need to regenerate data files. Only metadata, which points to source data files, is regenerated, as shown in the following diagram:



Hive example

```
ALTER TABLE table_name SET TBLPROPERTIES
('storage_handler'='org.apache.iceberg.mr.hive.HiveIcebergStorageHandler');
```

Changing the metadata location

From Hive, you can change the table metadata location, also known as the snapshot location.

Prerequisites

- The new location must contain exactly the same metadata json file as the old location.
- Before changing the metadata location, you must migrate the table to Iceberg.

After migrating a table to Iceberg, you can change the metadata location using ALTER TABLE as shown below:

Hive example

```
ALTER TABLE t set TBLPROPERTIES ('metadata_location'='<path>/hivemetadate/00
003-a1ada2b8-fc86-4b5b-8c91-400b6b46d0f2.metadata.json');
```

Partition evolution feature

Evolving a partition means changing it without rewriting data files. To evolve an Iceberg partition from Hive or Impala, you learn to use ALTER to change identity partitions. By setting a partition spec for an identity transformation partition, you alter the table.

You use the SET PARTITION SPEC clause in an ALTER statement to change the identity partition for a table. A partition spec change results in a new metadata.json and a commit, but does not create a new snapshot.

Hive or Impala syntax

```
ALTER TABLE table_name SET PARTITION SPEC ([col_name][, spec(value)][, spec(value)]...)]
```

- spec

The specification for a transform listed in the next topic, "Partition transform feature".

Hive or Impala examples

```
ALTER TABLE t
SET PARTITION SPEC ( TRUNCATE(5, level), HOUR(event_time),
BUCKET(15, message), price);
ALTER TABLE ice_p
SET PARTITION SPEC (VOID(i), VOID(d), TRUNCATE(3, s), HOUR(t), i);
```

Related Information

[Partition transform feature](#)

Partition transform feature

From Hive or Impala, you can create a table using identity partitioning in which every value is a single partition, or the partition is calculated from values using transformations. You learn supported transformations and see examples of how to partition a table.

Using CREATE TABLE ... PARTITIONED BY you create identity-partitioned Iceberg tables. Identity-partitioned Iceberg tables are similar to the Hive or Impala partitioned tables, which are stored in the same directory structure as the data files. Iceberg stores the partitioning columns of identity-partitioned Iceberg tables in a different directory structure from the data files if the tables are migrated to Iceberg from Hive external tables. Iceberg handles the tables and files regardless of the location.

Hive and Impala support Iceberg advanced partitioning through the PARTITION BY SPEC clause. Using this clause, you can define the Iceberg partition fields and partition transforms.

The following table lists the available transformations of partitions and corresponding transform spec.

Transformation	Spec	Supported by SQL Engine
Partition by year	years(time_stamp) year(time_stamp)	Hive and Impala
Partition by month	months(time_stamp) month(time_stamp)	Hive and Impala
Partition by a date value stored as int (dateint)	days(time_stamp) date(time_stamp)	Hive
Partition by hours	hours(time_stamp)	Hive
Partition by a dateint in hours	date_hour(time_stamp)	Hive
Partition by hashed value mod N buckets	bucket(N, col)	Hive and Impala
Partition by value truncated to L, which is a number of characters	truncate(L, col)	Hive and Impala

Strings are truncated to length L. Integers and longs are truncated to bins. For example, `truncate(10, i)` yields partitions 0, 10, 20, 30 ...

The idea behind transformation partition by hashed value mod N buckets is the same as [hash bucketing for Hive tables](#). A hashing algorithm calculates the bucketed column value (modulus). For example, for 10 buckets, data is stored in column value % 10, ranging from 0-9 (0 to n-1) buckets.

You use the `PARTITIONED BY SPEC` clause to partition a table by an identity transform.

Hive syntax

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type)[, time_stamp TIMESTAMP] )]
  [PARTITIONED BY SPEC([col_name][, spec(value)][, spec(value)]...)]
  [STORED AS file_format]
  STORED BY ICEBERG
  [TBLPROPERTIES (property_name=property_value, ...)]
```

Where `spec(value)` represents one or more of the following transforms:

- `YEARS(col_name)`
- `MONTHS(col_name)`
- `DAYS(col_name)`
- `BUCKET(bucket_num,col_name)`
- `TRUNCATE(length, col_name)`

Impala syntax

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name
  [(col_name data_type, ... )]
  [PARTITIONED BY SPEC([col_name][, spec(value)][, spec(value)]...)]
  STORED (AS | BY) ICEBERG
  [TBLPROPERTIES (property_name=property_value, ...)]
```

Where `spec(value)` represents one or more of the following transforms:

- `YEARS(col_name)`
- `MONTHS(col_name)`
- `DAYS(col_name)`
- `BUCKET(bucket_num,col_name)`
- `TRUNCATE(length, col_name)`

Hive example

The following example creates a top level partition based on column `i`, a second level partition based on the hour part of the timestamp, and a third level partition based on the first 1000 characters in column `j`.

```
CREATE EXTERNAL TABLE ice_3 (i INT, t TIMESTAMP, j BIGINT) PARTITIONED BY SP
EC (i, HOUR(t), TRUNCATE(1000, j)) STORED BY ICEBERG;
```

Impala examples

```
CREATE TABLE ice_13 (i INT, t TIMESTAMP, j BIGINT) PARTITIONED BY SPEC (i, H
OUR(t), TRUNCATE(1000, j)) STORED BY ICEBERG;
```


The following examples show how to use the PARTITION BY SPEC clause in a CREATE TABLE query from Impala. The same transforms are available in a CREATE EXTERNAL TABLE query from Hive.

```
CREATE TABLE ice_t(id INT, name STRING, dept STRING)
PARTITIONED BY SPEC (bucket(19, id), dept)
STORED BY ICEBERG
TBLPROPERTIES ('format-version'='2');
```

```
CREATE TABLE ice_ctas
PARTITIONED BY SPEC (truncate(1000, id))
STORED BY ICEBERG
TBLPROPERTIES ('format-version'='2')
AS SELECT id, int_col, string_col FROM source_table;
```

Related Information

[Create table feature](#)

[Partition evolution feature](#)

[Creating an Iceberg partitioned table](#)

Query metadata tables feature

From Hive, you can query Iceberg metadata tables as you would query a Hive table. For example, you can use projections, joins, filters, and so on.

You can also use the describe table metadata feature to get information about metadata. The following Iceberg metadata tables are available from Hive:

- files
- entries
- snapshots
- manifests
- partitions

Hive Syntax

To reference a metadata table, use the full name of the table as shown in the following syntax:

```
<DATABASE_NAME>.<TABLE_NAME>.<METADATA_TABLE_NAME>
```

Hive Example

```
SELECT * FROM default.table_a.files;
```

Rollback table feature

In the event of a problem with your table, you can reset a table to a good state as long as the snapshot of the good table is available. You can roll back the table data based on a snapshot id or a timestamp.

When you modify an Iceberg table, a new snapshot of the earlier version of the table is created. When you roll back a table to a snapshot, a new snapshot is created. The creation date of the new snapshot is based on the Timezone of your session. The snapshot id does not change.

Hive and Impala syntax

```
ALTER TABLE test_table EXECUTE rollback(snapshotID);
```

```
ALTER TABLE test_table EXECUTE rollback('timestamp');
```

Hive and Impala examples

The following example rolls back to an earlier table, creating a new snapshot having a new creation date timestamp, but keeping the same snapshot id 3088747670581784990.

```
ALTER TABLE ice_t EXECUTE ROLLBACK(3088747670581784990);
```

The following example rolls the table back to the latest snapshot having a creation timestamp earlier than '2022-08-08 00:00:00'.

```
ALTER TABLE ice_7 EXECUTE ROLLBACK('2022-08-08 00:00:00')
```

Schema evolution feature

You learn that the Hive or Impala schema changes when the associated Iceberg table changes. You see examples of changing the schema.

Although you can change the schema of your table over time, you can still read old data files because Iceberg uniquely identifies schema elements. A schema change results in a new metadata.json and a commit, but does not create a new snapshot.

The Iceberg table schema is synchronized with the Hive/Impala table schema. A change to the schema of the Iceberg table by an outside entity, such as Spark, changes the corresponding Hive/Impala table. You can change the Iceberg table using ALTER TABLE to make the following changes:

From Hive:

- Add a column
- Replace a column
- Change a column type or its position in the table

From Impala:

- Add a column
- Rename a column
- Drop a column
- Change a column type

An unsafe change to a column type, which would require updating each row of the table for example, is not allowed. The following type changes are safe:

- int to long
- float to double
- decimal(P, S) to decimal(P', S) if precision is increased

You can drop a column by changing the old column to the new column.

Hive syntax

```
ALTER TABLE table_name ADD COLUMNS (col_name type[, ...])
ALTER TABLE table_name CHANGE COLUMN col_old_name col_new_name type

ALTER TABLE table_name CHANGE COLUMN col_old_name col_new_name type [FIRST|
AFTER col_name] [existing_col_name
]
ALTER TABLE table_name REPLACE COLUMNS (col_name type)
```

Impala syntax

```
ALTER TABLE table_name ADD COLUMNS(col_name type[, ...])

ALTER TABLE table_name CHANGE COLUMN col_old_name col_new_name type
ALTER TABLE table_name DROP COLUMN col_name
```

Hive examples

```
ALTER TABLE t ADD COLUMNS(message STRING, price DECIMAL(8,1));

ALTER TABLE t REPLACE COLUMNS (i int comment '...', a string, ...);
ALTER TABLE t CHANGE COLUMN col_x col_x DECIMAL (22, 3) AFTER col_y;
```

Impala examples

```
ALTER TABLE ice_12 ADD COLUMNS(message STRING, price DECIMAL(8,1));
ALTER TABLE ice_12 DROP COLUMN i;

ALTER TABLE ice_12 CHANGE COLUMN s str STRING;
```

Schema inference feature

From Hive or Impala, you can base a new Iceberg table on a schema in a Parquet file. You see a difference in the Hive and Impala syntax and examples.

From Hive, you must use FILE in the CREATE TABLE LIKE ... statement. From Impala, you must omit FILE in the CREATE TABLE LIKE ... statement. The column definitions in the Iceberg table are inferred from the Parquet data file when you create a table like Parquet from Hive or Impala. Set the following table property for creating the table:

```
hive.parquet.infer.binary.as = <value>
```

Where <value> is binary (the default) or string.

This property determines the interpretation of the unannotated Parquet binary type. Some systems expect binary to be interpreted as string.

Hive syntax

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name LIKE FILE PARQUET 'object_storage_path_of_parquet_file'
[PARTITIONED BY [SPEC]([col_name][, spec(value)][, spec(value)]...)]
[STORED AS file_format]
STORED BY ICEBERG
[TBLPROPERTIES (property_name=property_value, ...)]
```

Impala syntax

```
CREATE TABLE [IF NOT EXISTS] [db_name.]table_name LIKE PARQUET 'object_storage_path_of_parquet_file'
[PARTITIONED BY [SPEC]([col_name][, spec(value)][, spec(value)]...)]
STORED (AS | BY) ICEBERG
[TBLPROPERTIES (property_name=property_value, ...)]
```

Hive example

```
CREATE TABLE ctlf_table LIKE FILE PARQUET 'hdfs://files/schema.parq'
      STORED BY ICEBERG;
```

```
CREATE TABLE ctlf_table LIKE FILE PARQUET 'hdfs://files/schema.parq'
      STORED BY ICEBERG;
```

Impala example

```
CREATE TABLE ctlf_table LIKE PARQUET 'hdfs://files/schema.parq'
      STORED BY ICEBERG;
```

```
CREATE TABLE ctlf_table LIKE PARQUET 'hdfs://files/schema.parq'
      STORED BY ICEBERG;
```

Time travel feature

From Hive or Impala, you can run point in time queries for auditing and regulatory workflows on Iceberg tables. Time travel queries can be time-based or based on a snapshot ID.

Iceberg generates a snapshot when you create, or modify, a table. A snapshot stores the state of a table. You can specify which snapshot you want to read, and then view the data at that timestamp. In Hive, you can use projections, joins, and filters in time travel queries. You can add expressions to the timestamps, as shown in the examples. You can expire snapshots.

Snapshot storage is incremental and dependent on the frequency and scale of updates. By default, Hive and Impala use the latest snapshot. You can query an earlier snapshot of Iceberg tables to get historical information. Hive and Impala use the latest schema to query an earlier table snapshot even if it has a different schema.

Hive or Impala syntax

```
SELECT * FROM table_name FOR SYSTEM_TIME AS OF 'time_stamp' [expression]
SELECT * FROM table_name FOR SYSTEM_VERSION AS OF snapshot_id [expression]
```

- time_stamp

The state of the Iceberg table at the time specified by the UTC timestamp.

- snapshot_id

The ID of the Iceberg table snapshot from the history output.

Hive or Impala examples

```
SELECT * FROM t FOR SYSTEM_TIME AS OF '2021-08-09 10:35:57' LIMIT 100;
SELECT * FROM t FOR SYSTEM_VERSION AS OF 3088747670581784990 limit 100;
SELECT * from ice_11 FOR SYSTEM_TIME AS OF now() - interval 30 minutes;
```

Related Information

[Expiring snapshots](#)

Truncate table feature

Truncating an Iceberg table removes all rows from the table. A new snapshot is created. Truncation works for partitioned and unpartitioned tables.

Although the table data and the table and column stats are cleared, the old snapshots and their data files continue to exist to support time travel in the future.

Hive syntax

```
TRUNCATE table_name
```

Impala syntax

```
TRUNCATE [TABLE] table_name
```

Hive or Impala example

```
TRUNCATE t;
```

Update and delete data features

From Hive, you can update and delete data in a V2 Iceberg table.

Hive updates and deletes Iceberg tables using position delete files, one type of encoding defined by the [Iceberg Spec](#). Impala reads, but does not write, position updates and deletes. Hive and Impala do not support equality deletes, the other type of encoding. As a typical user, you are oblivious to these encodings. If you have a problem with updates or deletes in the following situations, an equality delete file in the table is the likely cause:

- In Change Data Capture (CDC) applications
- In upserts from Apache Flink
- From a third-party engine

Inserting, deleting, or updating table data generates a snapshot. A new snapshot corresponds to a new manifest list. Manifest lists are named snap-*.avro.

Hive syntax

```
UPDATE tablename SET column = value [, column = value ...] [WHERE expres  
sion]
```

```
DELETE FROM tablename [WHERE expression]
```

Hive example

```
create external table tbl_ice(a int, b string, c int) stored by iceberg stor  
ed as orc tblproperties ('format-version'='2');  
insert into tbl_ice values (1, 'one', 50), (2, 'two', 51), (3, 'three', 52),  
  (4, 'four', 53), (5, 'five', 54), (111, 'one', 55), (333, 'two', 56);  
update tbl_ice set b='Changed' where b in (select b from tbl_ice where a <  
4);  
delete from tbl_ice where a <= 2,1;
```

Best practices for Iceberg in CDP

Based on large scale TPC-DS benchmark testing, performance testing and real-world experiences, Cloudera recommends several best practices when using Iceberg.

Follow these key best practices listed below when using Iceberg:

- Use Iceberg as intended for analytics.

The table format is designed to manage a large, slow-changing collection of files. For more information, see the [Iceberg spec](#).

- Reduce read amplification

Monitor the growth of positional delta files, and perform timely compactions.

- Speed up drop table performance, preventing deletion of data files by using the following table properties:

```
Set external.table.purge=false and gc.enabled=false
```

- Tune the following table properties to improve concurrency on writes and reduce commit failures: `commit.retry.num-retries` (default is 4), `commit.retry.min-wait-ms` (default is 100)
- Maintain a relatively small number of data files under the iceberg table/partition directory for efficient reads. To alleviate poor performance caused by too many small files, run the following queries:

```
TRUNCATE TABLE target;
INSERT OVERWRITE TABLE target select * from target FOR SYSTEM_VERSION AS
OF <preTruncateSnapshotId>;
```

- To minimize the number of delete files and file handles and improve performance, ensure that the Spark write.distribution.mode table property value is “hash” (the default setting for Spark Iceberg 1.2.0 onwards).

The following topics list additional best practices.

Feature limitations

Apache Iceberg in CDP has some limitations you need to understand.

The following features have limitations or are not supported in this release:

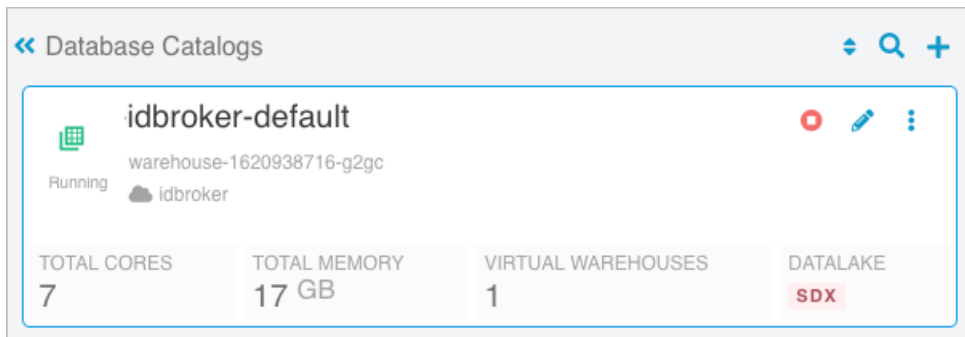
- When the underlying table is changed, you need to rebuild the materialized view manually, or use the Hive query scheduling to rebuild the materialized view.
- From Impala, you can read, but not write, [position updates and deletes](#).
- Hive and Impala do not support equality updates and deletes.
- An equality delete file in the table is the likely cause of a problem with updates or deletes in the following situations:
 - In Change Data Capture (CDC) applications
 - In upserts from Apache Flink
 - From a third-party engine

Prerequisites

You need to set up your environment and meet Data Lake prerequisites for querying Iceberg tables in CDP. You learn which query editors are supported and which roles are required.

The following list covers prerequisites for using Iceberg.

- You are using the Cloudera Data Warehouse (CDW).
- You must have access to a CDP environment that was activated from your environment UI. Your DATALAKE type must be SDX as shown below:



- Data Lake version 7.2.12.1 or higher is required.
- One of the following query editors are required to query Iceberg tables:
 - Hue (recommended)
 - A JDBC client
 - The Impala shell for remote users
- You must have the required role: DWUser.
- You must obtain permission to run SQL queries from the Env Admin, who must add you to the Hadoop SQL Storage Handler and Hadoop SQL policies.
- You must use the HadoopFileIO. S3FileIO is not supported. URLs in metadata starting with s3://... cause query failure.

Related Information

[Grant permission to run SQL queries](#)

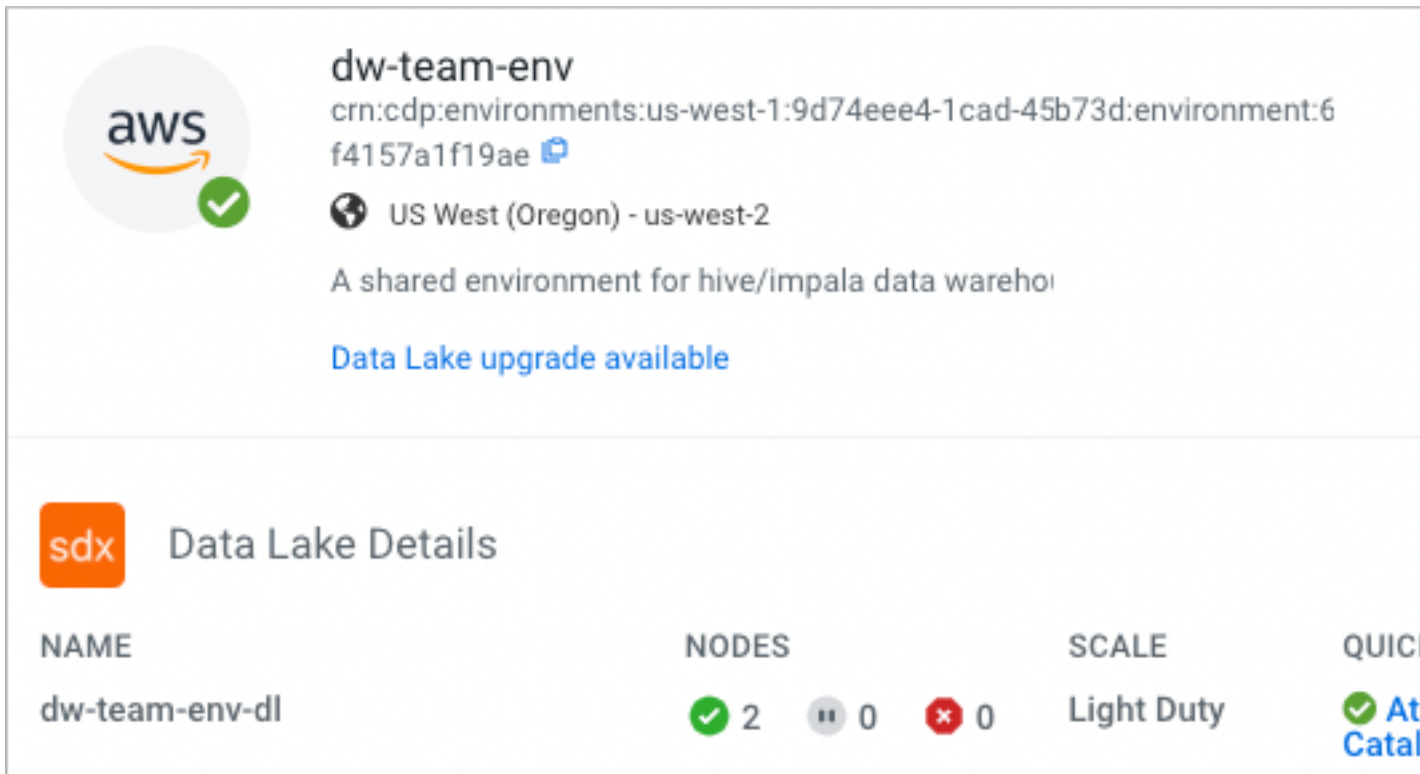
[Test driving Iceberg from Hive](#)

Accessing Iceberg tables

CDP uses Apache Ranger to provide centralized security administration and management. The Ranger Admin UI is the central interface for security administration. You can use Ranger to create two policies that allow users to query Iceberg tables.

Using Ranger with Iceberg is a technical preview and not recommended for General Data Protection Regulation (GDPR) applications.

How you open the Ranger Admin UI differs from one CDP service to another. In Management Console, you can select your environment, and then click **Environment Details Quick Links Ranger**.



aws

dw-team-env
 crn:cdp:environments:us-west-1:9d74eee4-1cad-45b73d:environment:6f4157a1f19ae

US West (Oregon) - us-west-2

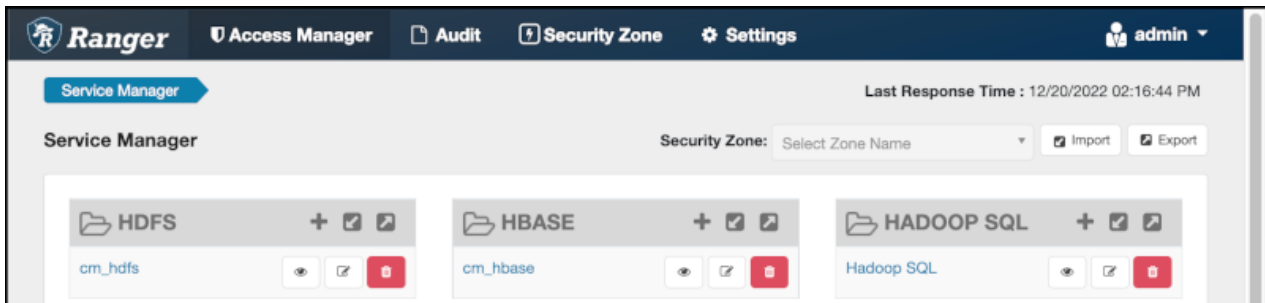
A shared environment for hive/impala data warehouse

[Data Lake upgrade available](#)

sdx **Data Lake Details**

NAME	NODES	SCALE	QUICK
dw-team-env-dl	2 0 0	Light Duty	At Catal

You log into the Ranger Admin UI, and the Ranger Service Manager appears.



Ranger Access Manager Audit Security Zone Settings admin

Service Manager Last Response Time : 12/20/2022 02:16:44 PM

Service Manager Security Zone: Select Zone Name Import Export

HDFS +
 cm_hdfs

HBASE +
 cm_hbase

HADOOP SQL +
 Hadoop SQL

The default policies that appear differ from service to service. You need to set up two Hadoop SQL policies to query Iceberg tables, and depending on your use case, you might want to include a third policy: an IAM policy to secure access to your data storage:

- One to authorize users to access the Iceberg files
Follow steps in "Edit a policy to access Iceberg files" below.
- One to authorize users to query Iceberg tables
Follow steps in "Creating a policy to query an Iceberg table" below.
- An optional IAM policy for data storage
Until Ranger with Iceberg policies are GA, if you need tight security, use ["S3 buckets, IAM roles, and policies for logs, backups, and data storage"](#) in addition to the Hadoop SQL Storage Handler policy to secure Iceberg files on your object store.

You can find complete Ranger documentation for Public Cloud in ["Security"](#).

Prerequisites

- Obtain the RangerAdmin role.

- Get the user name and password your Administrator set up for logging into the Ranger Admin.

The default credentials for logging into the Ranger Admin Web UI are admin/admin123.

Editing a policy to access Iceberg files

You learn how to edit the existing default Hadoop SQL Storage Handler policy to access files. This policy is one of the two Ranger policies required to use Iceberg.

About this task

The Hadoop SQL Storage Handler policy allows references to Iceberg table storage location, which is required for creating or altering a table. You use a storage handler when you create a file stored as Iceberg on the file system or object store.

In this task, you specify Iceberg as the storage-type and allow the broadest access by setting the URL to *.

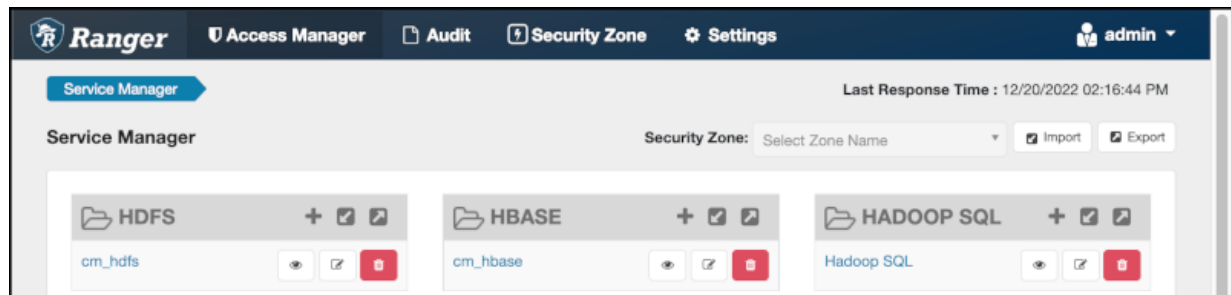
The Hadoop SQL Storage Handler policy supports only the RW Storage permission. A user having the required RW Storage permission on a resource, such as Iceberg, that you specify in the storage-type properties, is allowed only to reference the table location (for create/alter operations) in Iceberg. The RW Storage permission does not provide access to any table data. You need to create the Hadoop SQL policy described in the next topic in addition to this Hadoop SQL Storage Handler policy to access data in tables.

For more information about these policy settings, see [Ranger Storage Handler documentation](#).

Procedure

1. Log into Ranger Admin Web UI.

The Ranger Service Manager appears:



2. In Policy Name, enable the all - storage-type, storage-url policy.

List of Policies : Hadoop SQL

Q Search for your policy...

Policy ID	Policy Name	Policy Labels	Status
8	all - global	--	Enabled
9	all - database, table, column	--	Enabled
10	all - database, table	--	Enabled
11	all - storage-type, storage-url	--	Enabled

3. In Service Manager, in Hadoop SQL, select Edit  and edit the all storage-type, storage-url policy.
4. Below Policy Label, select storage-type, and enter iceberg..
5. In Storage URL, enter the value *, enable Include.

Policy Type **Access**

Policy ID **11**

Policy Name * all - storage-type, storage-url **Enabled**

Policy Label Policy Label

storage-type * iceberg

Storage URL * * **Include**

For more information about these policy settings, see [Ranger storage handler documentation](#).

6. In Allow Conditions, specify roles, users, or groups to whom you want to grant RW storage permissions.

You can specify PUBLIC to grant access to Iceberg tables permissions to all users. Alternatively, you can grant access to one user. For example, add the systest user to the list of users who can access Iceberg:

Allow Conditions:

Select Role	Select Group	Select User
Select Roles	Select Groups	× hive × beacon × dpprofiler × hue × admin × impala × systest

For more information about granting permissions, see [Configure a resource-based policy: Hadoop-SQL](#).

7. Add the RW Storage permission to the policy.
8. Save your changes.

Creating a policy to query an Iceberg table

You learn how to set up the second required policy for using Iceberg. This policy manages SQL query access to Iceberg tables.

About this task

You create a Hadoop SQL policy to allow roles, groups, or users to query an Iceberg table in a database. In this task, you see an example of just one of many ways to configure the policy conditions. You grant (allow) the selected roles, groups, or users the following add or edit permissions on the table: Select, Update, Create, Drop, Alter, and All. You can also deny permissions.

For more information about creating this policy, see [Ranger documentation](#).

Procedure

1. Log into Ranger Admin Web UI.
The Ranger Service Manager appears.
2. Click Add New Policy.

3. Fill in required fields.

For example, enter the following required settings:

- In Policy Name, enter the name of the policy, for example IcebergPolicy1.
- In database, enter the name of the database controlled by this policy, for example icedb.
- In table, enter the name of the table controlled by this policy, for example icetable.
- In columns, enter the name of the column controlled by this policy, for example enter the wildcard asterisk (*) to allow access to all columns of icetable.
- Accept defaults for other settings.

4. Scroll down to Allow Conditions, and select the roles, groups, or users you want to access the table.

You can use Deny All Other Accesses to deny access to all other roles, groups, or users other than those specified in the allow conditions for the policy.

5. Select permissions to grant.

For example, select Create, Select, and Alter. Alternatively, to provide the broadest permissions, select All.

Ignore RW Storage and other permissions not named after SQL queries. These are for future implementations.

6. Click Add.

Creating an Iceberg table

A step-by-step procedure describes how to create an Apache Iceberg table from a Hive or Impala Virtual Warehouse. You see how to access and use the recommended query editor Hue to create an Iceberg table.

About this task

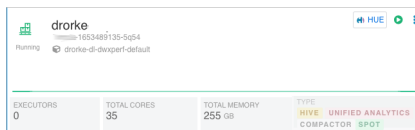
In this task, you create an Iceberg table in Cloudera Data Warehouse. In a Virtual Warehouse, you open Hue, and use Hive or Impala to create a table.

Before you begin

- You must meet the prerequisites to query Iceberg tables from a Virtual Warehouse mentioned earlier, including obtaining Ranger access permissions.

Procedure

- Create a new Virtual Warehouse, or select an existing one.
- In Cloudera Data Warehouse Overview, select a Virtual Warehouse, and click Hue.



- Select a database.
- Enter a query to create a simple Iceberg table in the default Parquet format.
Hive example:

```
CREATE EXTERNAL TABLE ice_t1 (i int, s string, ts timestamp, d date)
STORED BY ICEBERG;
```

Impala example:

```
CREATE TABLE ice_t2 (i int, s string, ts timestamp, d date)
STORED BY ICEBERG;
```

In CDP, CREATE EXTERNAL TABLE, and just CREATE TABLE, are valid from Hive. You use the EXTERNAL keyword from Hive to create the Iceberg table to purge the data when you drop the table. In CDP, from Impala, you must use CREATE TABLE to initialize the Iceberg table.

- Click  to run the query.

Related Information

[Adding a new Virtual Warehouse](#)

[Submitting queries with Hue](#)

Creating an Iceberg partitioned table

The ease of use of the Iceberg partitioning is clear from an example of how to partition a table using the backward compatible, identity-partition syntax. Alternatively, you can partition an Iceberg table by column values from Hive or Impala.

About this task

You can specify partitioning that is backward compatible with Iceberg V1 using the `PARTITION BY` clause. This type of table is called an identity-partitioned table. For more information about partitioning, see the [Apache Iceberg documentation](#).

Before you begin

Procedure

1. In Hue, select a database.
2. Create an identity-partitioned table.
Hive:

```
CREATE EXTERNAL TABLE ice_ext1 (i int, s string, ts timestamp, d date) P
ARTITIONED BY (state string)
STORED BY ICEBERG
STORED AS ORC;
```

Impala:

```
CREATE TABLE ice_ext2 (i int, s string, ts timestamp, d date) PARTITIONED
BY (state string)
STORED BY ICEBERG;
```

3. Click  to run the query.
4. Create a table and specify an identity transform, such as bucket, truncate, or date, using the Iceberg V2 `PARTITION BY SPEC` clause.
Hive:

```
CREATE TABLE ice_t_transforms_1 (i int, s string, ts timestamp, d date)
PARTITIONED BY SPEC (TRUNCATE(10, i), BUCKET(11, s), YEAR(ts))
STORED BY ICEBERG;
```

Impala:

```
CREATE TABLE ice_t_transforms (i int, s string, ts timestamp, d date)PAR
TITIONED BY SPEC (TRUNCATE(10, i), BUCKET(11, s), YEAR(ts))STORED AS ICE
BERG;
```

5. Click  to run the query.

Related Information

[Adding a new Virtual Warehouse](#)

[Submitting queries with Hue](#)

[Partition transform feature](#)

Expiring snapshots

You can expire snapshots of an Iceberg table using an `ALTER TABLE` query from Hive or Impala. You should periodically expire snapshots to delete data files that are no longer needed, and reduce the size of table metadata.

About this task

Each write to an Iceberg table creates a new snapshot, or version, of a table. Snapshots can be used for time-travel queries, or the table can be rolled back to any valid snapshot. Snapshots accumulate until they are expired by the `expire_snapshots` operation.

Procedure

Enter a query to expire snapshots older than the following timestamp: '2021-12-09 05:39:18.689000000'

```
ALTER TABLE test_table EXECUTE expire_snapshots('2021-12-09 05:39:18.689000000 00');
```

Related Information

[Time travel feature](#)

Inserting data into a table

You can append data to an Iceberg table by inserting values or by selecting the data from another table. You can update data, replacing the old data.

You use the `INSERT` command in one of the following ways to populate an Iceberg table from Hive:

- `INSERT INTO t VALUES (1, 'asf', true);`
- `INSERT INTO t SELECT * FROM s;`
- `INSERT OVERWRITE t SELECT * FROM s;`

Examples

```
INSERT INTO t VALUES (1, 'asf', true);
INSERT INTO t SELECT * FROM s;
INSERT OVERWRITE t SELECT * FROM s;
```

Migrating a Hive table to Iceberg

You see how to use a simple `ALTER TABLE` statement to migrate an external Hive table to an Iceberg table. You see how to configure table input and output by setting table properties.

About this task

When you migrate an external Hive table to Iceberg, Hive makes the following changes:

- Converts the `storage_handler`, `serde`, `inputformat` and `outputformat` properties of the table in HMS to use the Iceberg specific classes.
- Reads the footers of the existing data files and generates the necessary Iceberg metadata files based on the footers.
- Commits all the data files to the Iceberg table in a single commit.



Note: To prevent loss of new and old table data during migration of a table to Iceberg, do not drop or move the old table during migration. Exception: If you set the table property `'external.table.purge'=FALSE`, no data loss occurs when you drop the table.

Before you begin

- You must meet the prerequisites for using Iceberg mentioned earlier.
- .

Restriction: Migrating an Impala table to Iceberg is not supported in this release.

Procedure

1. Log in to the CDP web interface and navigate to the Data Warehouse service.
2. In the Data Warehouse service, in the Overview page, locate your Hive Virtual Warehouse, and click Hue.
Instead of using Hue, you can connect over JDBC to the Hive Virtual Warehouse, and run the query.
3. Enter a query to use a database.
For example:

```
USE mydb;
```

4. Enter a Hive query to migrate an existing external Hive table to an Iceberg v2 table.
For example:

```
ALTER TABLE tbl
SET TBLPROPERTIES ('storage_handler'='org.apache.iceberg.mr.hive.HiveIcebergStorageHandler',
'format-version' = '2');
```

Do not drop the table as explained above unless you set the 'external.table.purge' table property to false.

5. Click  to run the queries.
An Iceberg V2 table is created. The Hive table remains intact.

Selecting an Iceberg table

You see an example of how to read an Apache Iceberg table, and understand the advantages of Iceberg.

About this task

Working with timestamps in Iceberg, you do not need to know whether the table is actually partitioned by month, day or hour, based on the timestamp value. You can simply supply a predicate for the timestamp value and Iceberg converts the timestamp to month/day/hour transparently. Hive/Impala must maintain actual partition values in a separate column (for example, ts_month or ts_day). Forgetting to reference the derived partition column in your query can lead to inadvertent full table scans.

By default iceberg.table_identifier is not set in CDP, so you can use the familiar <db_name.<table_name> in queries.

Before you begin

- You must meet the prerequisites to query Iceberg tables mentioned earlier.

Procedure

1. Use a database.
For example:
2. Query an Iceberg table partitioned by city.
For example:

```
SELECT * FROM ice_t2 WHERE city="Bangalore";
```


Running time travel queries

You query historical snapshots of data using the `FOR SYSTEM_TIME AS OF '<timestamp>' FOR SYSTEM_VERSION AS OF <snapshot_id>` clauses in a select statement. You see how to use AS OF to specify a snapshot of your Iceberg data at a certain time.

About this task

You can inspect the history of an Iceberg table to see the snapshots. You can query the metadata of the Iceberg table using a `SELECT ... AS OF` statement to run time travel queries. You use history information from a query of the database to identify and validate snapshots, and then query a specific snapshot AS OF a certain Timestamp value.

Before you begin

- You must be aware of the table history.
However, this can include commits that have been rolled back.
- You must have access to valid snapshots.
- You must meet the prerequisites to query Iceberg tables mentioned earlier.

Procedure

1. View the table history.

```
SELECT * FROM db.table.history;
```

2. Check the valid snapshots of the table.

```
SELECT * FROM db.table.snapshots;
```

3. Query a specific snapshot by providing the timestamp and snapshot_id.

```
SELECT * FROM T  
FOR SYSTEM_TIME AS OF <TIMESTAMP>;  
SELECT * FROM t  
FOR SYSTEM_VERSION AS OF <SNAPSHOT_ID>;
```

Updating an Iceberg partition

You see how to update Iceberg table partitioning in an existing table and then how to change the partitioning to be more granular.

About this task

Partition information is stored logically, and only in table metadata. When you update a partition spec, the old data written with an earlier spec remains unchanged. New data is written using the new spec in a new layout. Metadata for each of the partition versions is separate.

Before you begin

- You must meet the prerequisites to query Iceberg tables mentioned earlier.

Procedure

1. Create a table partitioned by year.
Hive

```
CREATE EXTERNAL TABLE ice_t (i int, j int, ts timestamp)
PARTITIONED BY SPEC (truncate(5, j), year(ts))
STORED BY ICEBERG;
```

Impala:

```
CREATE TABLE ice_t (i int, j int, ts timestmap)
PARTITIONED BY SPEC (truncate(5, j), year(ts))
STORED BY ICEBERG;
```

2. Split the data into manageable files using buckets.

```
ALTER TABLE ice_t SET PARTITION SPEC (bucket(13, i));
```

3. Partition the table by month.

```
ALTER TABLE ice_t SET PARTITION SPEC (truncate(5, j), month(ts));
```

Test driving Iceberg from Impala

You complete a task that creates Iceberg tables from Impala with mock data that you can test drive using your own queries. You learn how to work with partitioned tables.

Before you begin

- You must meet the prerequisites to query Iceberg tables mentioned earlier, including obtaining Ranger access permissions.

Procedure

1. In Impala, use a database.
2. Create an Impala table to hold mock data for this task.

```
create external table mock_rows stored as parquet as
select x from (
with v as (values (1 as x), (1), (1), (1), (1))
select v.x from v, v v2, v v3, v v4, v v5, v v6
) a;
```

3. Create another Impala table based on mock_rows.

```
create external table customer_demo stored as parquet as
select
FROM_TIMESTAMP(DAYS_SUB(now(), cast ( TRUNC(RAND(7)*365*1) as bigint)), '
yyyy-MM') as year_month,
DAYS_SUB(now(), cast ( TRUNC(RAND(7)*365*1) as bigint)) as ts,
CONCAT(
  cast ( TRUNC(RAND(1) * 250 + 2) as string), '.',
  cast ( TRUNC(RAND(2) * 250 + 2) as string), '.',
  cast ( TRUNC(RAND(3) * 250 + 2) as string), '.',
  cast ( TRUNC(RAND(4) * 250 + 2) as string)
) as ip,
```

```

CONCAT("USER_", cast ( TRUNC(RAND(4) * 1000) as string), '@somedomain.com')
as email,
CONCAT("USER_", cast ( TRUNC(RAND(5) * 1000) as string)) as username,
CONCAT("USER_", cast ( TRUNC(RAND(6) * 100) as string)) as country,
cast( RAND(8)*10000 as double) as metric_1,
cast( RAND(9)*10000 as double) as metric_2,
cast( RAND(10)*10000 as double) as metric_3,
cast( RAND(11)*10000 as double) as metric_4,
cast( RAND(12)*10000 as double) as metric_5
from mock_rows
;

```

4. Create another Impala table based on mock_rows.

```

create external table customer_demo2 stored as parquet as
select
FROM_TIMESTAMP(DAYS_SUB(now() , cast ( TRUNC(RAND(7)*365*1) as bigint)),
'yyyy-MM') as year_month,
DAYS_SUB(now() , cast ( TRUNC(RAND(7)*365*1) as bigint)) as ts,
CONCAT(
  cast ( TRUNC(RAND(1) * 250 + 2) as string), '.',
  cast ( TRUNC(RAND(2) * 250 + 2) as string), '.',
  cast ( TRUNC(RAND(3) * 250 + 2) as string), '.',
  cast ( TRUNC(RAND(4) * 250 + 2) as string)
) as ip,
CONCAT("USER_", cast ( TRUNC(RAND(4) * 1000) as string), '@somedomain.com')
as email,
CONCAT("USER_", cast ( TRUNC(RAND(5) * 1000) as string)) as username,
CONCAT("USER_", cast ( TRUNC(RAND(6) * 100) as string)) as country,
cast( RAND(8)*10000 as double) as metric_1,
cast( RAND(9)*10000 as double) as metric_2,
cast( RAND(10)*10000 as double) as metric_3,
cast( RAND(11)*10000 as double) as metric_4,
cast( RAND(12)*10000 as double) as metric_5
from mock_rows
;

```

5. Create an Iceberg table from the customer_demo table.

```

CREATE TABLE customer_demo_iceberg STORED BY ICEBERG AS SELECT * FROM cu
stomer_demo;

```

6. Insert into the customer_demo_iceberg table the results of selecting all data from the customer_demo2 table.

```

INSERT INTO customer_demo_iceberg select * from customer_demo2;
INSERT INTO customer_demo_iceberg select * from customer_demo2;
INSERT INTO customer_demo_iceberg select * from customer_demo2;

```

7. Create an Iceberg table partitioned by the year_month column and based on the customer_demo_iceberg table.

```

CREATE TABLE customer_demo_iceberg_part PARTITIONED BY(year_month) STORED
BY ICEBERG
AS SELECT ts, ip , email, username , country, metric_1 , metric_2 , metric
_3 , metric_4 , metric_5, year_month
FROM customer_demo_iceberg;

```

8. Split the partitioned data into manageable files.

```

ALTER TABLE customer_demo_iceberg_part SET PARTITION SPEC (year_month,BU
CKET(15, country));

```

9. Insert the results of reading the customer_demo_iceberg table into the partitioned table.

```
INSERT INTO customer_demo_iceberg_part (year_month, ts, ip, email, username, country, metric_1, metric_2, metric_3, metric_4, metric_5)
SELECT year_month, ts, ip, email, username, country, metric_1, metric_2, metric_3, metric_4, metric_5
FROM customer_demo_iceberg;
```

10. Run time travel queries on the Iceberg tables, using the history output to get the snapshot id, and substitute the id in the second SELECT query.

```
SELECT * FROM customer_demo_iceberg FOR SYSTEM_TIME AS OF '2021-12-09 05:39:18.689000000' LIMIT 100;
DESCRIBE HISTORY customer_demo_iceberg;
SELECT * FROM customer_demo_iceberg FOR SYSTEM_VERSION AS OF <snapshot id> LIMIT 100;
```

Test driving Iceberg from Hive

You learn how to access the Hive demo data, which you can use to get hands-on experience running Iceberg queries.

About this task

A DWAdmin can optionally load demo data in Hue when you create a new Database Catalog.

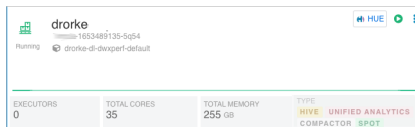
The Admin enables Load Demo Data when creating the Database Catalog. Users can then query sample airline demo data in Hue.

Before you begin

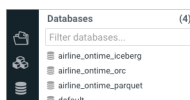
- You must meet the prerequisites to query Iceberg tables from a Virtual Warehouse mentioned earlier.
- You have access to a Hive Virtual Warehouse, having a Database Catalog in which demo data has been loaded.
- You obtained the required role for querying the Virtual Warehouse: DWUser

Procedure

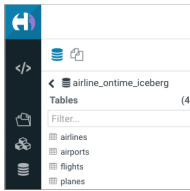
1. In Cloudera Data Warehouse Overview, select a Hive Virtual Warehouse, and click Hue.



2. In Hue, expand the default database and verify that the airline demo data is available in your Virtual Warehouse. You see the following list of demo databases:



3. Select `airline_ontime_iceberg` to use the `airline_ontime_iceberg` database.
4. Take a look at the tables in the `airline_ontime_iceberg` database.



Flights is the fact table. It has 100M rows and three dimensions, airline, airports, and planes. This records flights for more than 10 years in the US, and includes the following details:

- origin
- destination
- delay
- air time

5. Become familiar with the Iceberg airline queries to set up this database. See the next topic.

6. Query the demo data from Hive.

For example, find the flights that departed each year, by IATA code, airport, city, state, and country. Find the average departure delay.

```
SELECT f.month, a.iata, a.airport, a.city, a.state, a.country
FROM flights f,
airports a
WHERE f.origin = a.iata
GROUP BY
f.month,
a.iata,
a.airport,
a.city,
a.state,
a.country
HAVING COUNT(*) > 10000
ORDER BY AVG(f.DepDelay) DESC
LIMIT 10;
```

Output appears as follows:

f.month	a.iata	a.airport	a.city
a.state	a.country		
12	ORD	Chicago O'Hare International	Chicago
NULL	USA		
6	EWK	Newark Intl	Newark
NULL	USA		
7	JFK	John F Kennedy Intl	New York
NULL	USA		
6	IAD	Washington Dulles International	Chantilly
NULL	USA		
7	EWK	Newark Intl	Newark
NULL	USA		
6	PHL	Philadelphia Intl	Philadelphia
NULL	USA		
1	ORD	Chicago O'Hare International	Chicago
NULL	USA		
6	ORD	Chicago O'Hare International	Chicago
NULL	USA		

```

| 7 | ATL | William B Hartsfield-Atlanta Intl | Atlanta
| NULL | USA | |
| 12 | MDW | Chicago Midway | Chicago
| NULL | USA | |
+-----+-----+-----+-----+
+-----+-----+
10 rows selected (103.812 seconds)

```

7. Split the partitioned data into manageable files.

```
ALTER TABLE airports SET PARTITION SPEC (iata, BUCKET(15, country));
```

Related Information

[Prerequisites](#)

Hive demo data

To test drive Iceberg from Hive, you use demo data in the `airline_online_iceberg` database.

Iceberg Database creation and setup

The Airlines demo data for Iceberg is stored in the `airline_online_iceberg` database. The following queries created and set up this database.

```

create database if not exists airline_ontime_iceberg;
use airline_ontime_iceberg;
set hive.vectorized.execution.enabled=false;
set hive.stats.column.autogather=false;

```

Hive external table creation

The following Hive external tables were created in the `airline_online_iceberg` database:

- airports
- airlines
- planes
- flights

```

create external table if not exists airports (
    iata string,
    airport string,
    city string,
    state double,
    country string,
    lat double,
    lon double
)
stored as orc;

create external table if not exists airlines (
    code string,
    description string
)
stored as orc;
create external table if not exists planes (
    tailnum string,
    owner_type string,
    manufacturer string,

```

```

        issue_date string,
        model string,
        status string,
        aircraft_type string,
        engine_type string,
        year int
    )
    stored as orc;

create external table if not exists flights (
    month int,
    dayofmonth int,
    dayofweek int,
    deptime int,
    crsdeptime int,
    arrtime int,
    crsarrrtime int,
    uniquecarrier string,
    flightnum int,
    tailnum string,
    actualelapsedtime int,
    crselapsedtime int,
    airtime int,
    arrdelay int,
    depdelay int,
    origin string,
    dest string,
    distance int,
    taxiin int,
    taxiout int,
    cancelled int,
    cancellationcode string,
    diverted string,
    carrierdelay int,
    weatherdelay int,
    nasdelay int,
    securitydelay int,
    lateaircraftdelay int
)
partitioned by (year int)
stored as orc;

```

Load data into the newly created tables

```

load data inpath '${datapath}/airline_ontime_iceberg.db/airports' into table
airports;

load data inpath '${datapath}/airline_ontime_iceberg.db/airlines' into table
airlines;

load data inpath '${datapath}/airline_ontime_iceberg.db/planes' into table p
lanes;

load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=1995' i
nto table flights partition (year=1995);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=1996'
into table flights partition (year=1996);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=1997'
into table flights partition (year=1997);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=1998'
into table flights partition (year=1998);

```

```
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=1999' i
nto table flights partition (year=1999);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2000'
into table flights partition (year=2000);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2001'
into table flights partition (year=2001);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2002'
into table flights partition (year=2002);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2003' i
nto table flights partition (year=2003);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2004'
into table flights partition (year=2004);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2005'
into table flights partition (year=2005);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2006'
into table flights partition (year=2006);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2007' i
nto table flights partition (year=2007);
load data inpath '${datapath}/airline_ontime_iceberg.db/flights/year=2008'
into table flights partition (year=2008);
```

Convert these existing Hive external tables to Iceberg tables

```
ALTER TABLE planes ADD CONSTRAINT planes_pk PRIMARY KEY (tailnum) DISABLE NO
VALIDATE;
ALTER TABLE flights ADD CONSTRAINT planes_fk FOREIGN KEY (tailnum) REFEREN
CES planes(tailnum) DISABLE NOVALIDATE RELY;

ALTER TABLE airlines ADD CONSTRAINT airlines_pk PRIMARY KEY (code) DISABLE
NOVALIDATE;
ALTER TABLE flights ADD CONSTRAINT airlines_fk FOREIGN KEY (uniquecarrier)
REFERENCES airlines(code) DISABLE NOVALIDATE RELY;

ALTER TABLE airports ADD CONSTRAINT airports_pk PRIMARY KEY (iata) DISABLE N
OVALIDATE;
ALTER TABLE flights ADD CONSTRAINT airports_orig_fk FOREIGN KEY (origin)
REFERENCES airports(iata) DISABLE NOVALIDATE RELY;
ALTER TABLE flights ADD CONSTRAINT airports_dest_fk FOREIGN KEY (dest) RE
FERENCES airports(iata) DISABLE NOVALIDATE RELY;

ALTER TABLE airports SET TBLPROPERTIES ('storage_handler'='org.apache.iceb
erg.mr.hive.HiveIcebergStorageHandler');
ALTER TABLE airlines SET TBLPROPERTIES ('storage_handler'='org.apache.icebe
rg.mr.hive.HiveIcebergStorageHandler');
ALTER TABLE planes SET TBLPROPERTIES ('storage_handler'='org.apache.iceberg.
mr.hive.HiveIcebergStorageHandler');
ALTER TABLE flights SET TBLPROPERTIES ('storage_handler'='org.apache.iceber
g.mr.hive.HiveIcebergStorageHandler');
```

Iceberg data types

References include Iceberg data types and a table of equivalent SQL data types by Hive/Impala SQL engine types.

Iceberg supported data types

Table 1:

Iceberg data type	SQL data type	Hive	Impala
binary		BINARY	BINARY
boolean	BOOLEAN	BOOLEAN	BOOLEAN
date	DATE	DATE	DATE
decimal(P, S)	DECIMAL(P, S)	DECIMAL (P, S)	DECIMAL (P, S)
double	DOUBLE	DOUBLE	DOUBLE
fixed(L)		BINARY	Not supported
float	FLOAT	FLOAT	FLOAT
int	TINYINT, SMALLINT, INT	INTEGER	INTEGER
list	ARRAY	ARRAY	Read only
long	BIGINT	BIGINT	BIGINT
map	MAP	MAP	Read only
string	VARCHAR, CHAR	STRING	STRING
struct	STRUCT	STRUCT	Read only
time		STRING	Not supported
timestamp	TIMESTAMP	TIMESTAMP	TIMESTAMP
timestampz	TIMESTAMP WITH LOCAL TIME ZONE	Use TIMESTAMP WITH LOCAL TIMEZONE for handling these in queries	Read timestampz into TIMESTAMP values Writing not supported
uuid	none	STRING Writing to Parquet is not supported	Not supported

Data type limitations

An implicit conversion to an Iceberg type occurs only if there is an exact match; otherwise, a cast is needed. For example, to insert a VARCHAR(N) column into an Iceberg table you need a cast to the VARCHAR type as Iceberg does not support the VARCHAR(N) type. To insert a SMALLINT or TINYINT into an Iceberg table, you need a cast to the INT type as Iceberg does not support these types.

Unsupported data types

Impala does not support the following Iceberg data types:

- TIMESTAMPTZ (only read support)
- FIXED
- UUID

Iceberg table properties

The CDP environment for querying tables from Hive overrides some Iceberg table properties. You learn which table properties are supported for querying tables from Impala.

[Iceberg documentation](#) describes all the properties for configuring tables. This documentation focuses on key properties for working with Iceberg tables in CDP.

Iceberg supports concurrent writes by default. You can tune Iceberg v2 table properties for concurrent writes. You set the following properties if you plan to have concurrent writers on Iceberg v2 tables:

- `commit.retry.min-wait-ms`
- `commit.retry.num-retries`

CDP supports adding the Parquet compression type using table properties. For more information, see Iceberg documentation about [Compression Types](#).

You can use the Alter Table feature to set a property. From Hive, the following Iceberg table property overrides are in effect:

- `iceberg.mr.split.size` overrides `read.split.target-size`.
- `read.split.open-file-cost` is overridden.

You can tune Iceberg v2 table properties for concurrent writes. From Impala, the following subset of Iceberg table properties are supported:

- `history.expire.min-snapshots-to-keep`
Valid values: integers. Default = 1
- `write.format.default`
Valid value: Parquet
- `write.metadata.delete-after-commit.enabled`
Valid values: true or false. Default = TBD.
- `write.metadata.previous-versions-max`
Valid values: integers. Default = 100.
- `write.parquet.compression-codec`
Valid values: GZIP, LZ4, NONE, SNAPPY (default value), ZSTD
- `write.parquet.compression-level`
Valid values: 1 - 22. Default = 3
- `write.parquet.row-group-size-bytes`
Valid values: 8388608 (or 8 MB) - 2146435072 (or 2047MB). Overriden by `PARQUET_FILE_SIZE`.
- `write.parquet.page-size-bytes`
Valid values: 65536 (or 64KB) - 1073741824 (or 1GB).
- `write.parquet.dict-size-bytes`
Valid values: 65536 (or 64KB) - 1073741824 (or 1GB)