

7.1.9

Using Apache Iceberg

Date published: 2023-08-31

Date modified: 2024-07-19

CLOUDERA

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

Legal Notice

© Cloudera Inc. 2024. 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.....	7
Create partitioned table as select feature.....	7
Create table ... like feature.....	8
Describe table metadata feature.....	8
Drop table feature.....	8
Expire snapshots feature.....	9
Insert table data feature.....	10
Load data inpath feature.....	10
Load or replace partition data feature.....	11
Flexible partitioning.....	11
Partition evolution feature.....	12
Partition transform feature.....	12
Rollback table feature.....	13
Select Iceberg data feature.....	14
Schema evolution feature.....	14
Schema inference feature.....	15
Time travel feature.....	15
Truncate table feature.....	16
 Best practices for Iceberg in CDP.....	 16
Making row-level changes on V2 tables only.....	17
 Performance tuning.....	 17
Caching manifest files.....	18
Configuring manifest caching in Cloudera Manager.....	18
 Unsupported features and limitations.....	 18
 Accessing Iceberg tables.....	 20
Editing a storage handler policy to access Iceberg files on HDFS or S3.....	20
Creating a SQL policy to query an Iceberg table.....	22
 Accessing Iceberg files in Ozone.....	 24
 Creating an Iceberg table.....	 26
 Creating an Iceberg partitioned table.....	 26

Expiring snapshots.....	27
Inserting data into a table.....	27
Selecting an Iceberg table.....	28
Running time travel queries.....	28
Updating an Iceberg partition.....	29
Test driving Iceberg from Impala.....	29
Iceberg data types.....	31
Iceberg table properties.....	32

Apache Iceberg features

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

This documentation describes how to use Iceberg features in [Open Data Lakehouse](#). From Impala, you can create and query Iceberg tables. Impala queries are table-format agnostic. For example, Impala options are supported in queries of Iceberg tables. You can run nested, correlated, or analytic queries on all supported table types.

This documentation does not attempt to show every possible query supported from Impala. Many examples of how to run queries on Iceberg tables from Impala are covered.

Supported ACID transaction properties

Iceberg supports atomic and isolated database transaction properties. Writers work in isolation, not affecting the live table, and perform a metadata swap only when the write is complete, making the changes in one atomic commit.

Iceberg uses snapshots to guarantee isolated reads and writes. You see a consistent version of table data without locking the table. Readers always see a consistent version of the data without the need to lock the table. Writers work in isolation, not affecting the live table, and perform a metadata swap only when the write is complete, making the changes in one atomic commit.

Iceberg partitioning

The Iceberg partitioning technique has performance advantages over conventional partitioning, such as Apache Hive partitioning. Iceberg hidden partitioning is easier to use. Iceberg supports in-place partition evolution; to change a partition, you do not rewrite the entire table to add a new partition column, and queries do not need to be rewritten for the updated table. Iceberg continuously gathers data statistics, which supports additional optimizations, such as partition pruning.

Iceberg uses multiple layers of metadata files to find and prune data. Hive and Impala keep track of data at the folder level and not at the file level, performing file list operations when working with data in a table. Performance problems occur during the execution of multiple list operations. Iceberg keeps track of a complete list of files within a table using a persistent tree structure. Changes to an Iceberg table use an atomic object/file level commit to update the path to a new snapshot file. The snapshot points to the individual data files through manifest files.

The manifest files track several data files across many partitions. These files store partition information and column metrics for each data file. A manifest list is an additional index for pruning entire manifests. File pruning increases efficiency.

Iceberg relieves Hive metastore (HMS) pressure by storing partition information in metadata files on the file system/object store instead of within the HMS. This architecture supports rapid scaling without performance hits.

Alter table feature

In 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.

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

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', ... )
```

```
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;
```

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');
```

Related Information

[Expire snapshots feature](#)

Create table feature

You use `CREATE TABLE` from Impala to create an external table in Iceberg. You also learn about partitioning.

From Impala, 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 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.

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 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 Impala, you associate the Iceberg storage handler with the table using `STORED AS ICEBERG` or `STORED BY ICEBERG`.

Supported file formats

Impala supports writing Iceberg tables in only Parquet format. Impala does not support defining both file format and storage engine. You can read Iceberg tables in the following formats: 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.

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, ...)]
```

Impala examples

```
CREATE TABLE ice_7 (i INT, t TIMESTAMP, j BIGINT) STORED BY ICEBERG; //creates 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
```

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.

Impala example

```
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.

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.

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 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:

Impala example

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

Describe table metadata feature

You can use certain 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 Impala.

Command Syntax	Description	Impala Support
SHOW CREATE TABLE table_name	Reveals the schema that created the table.	yes
SHOW FILES IN table_name	Lists the files related to the table.	yes
SHOW PARTITIONS table_name	Returns the Iceberg partition spec, just the column information, not actual partitions or files.	yes
DESCRIBE [EXTENDED] table_name	The optional EXTENDED shows all the metadata for the table in Thrift serialized form, which is useful for debugging.	yes
DESCRIBE [FORMATTED] table_name	The optional FORMATTED shows the metadata in tabular format.	no
DESCRIBE HISTORY table_name [BETWEEN timestamp1 AND timestamp2]	Optionally limits the output history to a period of time.	yes

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 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.

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.

Impala syntax

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

Impala example

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

Related Information

[Expire snapshots 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 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:

Impala syntax

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

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);
```

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.

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

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". Setting this property controls automatic metadata file removal after metadata operations, such as expiring snapshots or inserting data.

Related Information

[Iceberg table properties](#)

[Alter table feature](#)

[Drop table feature](#)

Insert table data feature

From 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, Impala dynamically overwrites 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.

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. For more information about sorting, see [sort orders specification](#).

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
```

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;
```

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;
```

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;
```

Impala examples

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

Flexible partitioning

Iceberg partition evolution, which is a unique Iceberg feature, and the partition transform feature, greatly simplify partitioning tables and changing partitions.

Partitions based on transforms are stored in the Iceberg metadata layer, not in the directory structure. You can change the partitioning completely, or just refine existing partitioning, and write new data based on the new partition layout--no need to rewrite existing data files. For example, change a partition by month to a partition by day.

Use partition transforms, such as IDENTITY, TRUNCATE, BUCKET, YEAR, MONTH, DAY, HOUR. Iceberg solves scalability problems caused by having too many partitions. Partitioning can also include columns with a large number of distinct values. Partitioning is hidden in the Iceberg metadata layer, eliminating the need to explicitly write partition columns (YEAR, MONTH for example) or to add extra predicates to queries for partition pruning.

```
SELECT * FROM tbl WHERE ts = '2023-04-21 20:56:08'
```

```
AND YEAR = 2023 AND MONTH = 4 AND DAY = 21
```

Year, month, and day can be automatically extracted from '2023-04-21 20:56:08' if the table is partitioned by DAY(ts)

Partition evolution feature

Partition evolution means you can change the partition layout of the table without rewriting existing data files. Old data files can remain partitioned by the old partition layout, while newly added data files are partitioned based on the new layout.

You can use the ALTER TABLE SET PARTITION SPEC statement to change the partition layout of an Iceberg table. A change to the partition spec results in a new metadata.json and a commit, but does not create a new snapshot.

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".

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);
```

Partition transform feature

You can use one or more partition transforms to partition your data. Each transform is applied to a single column. Identity-transform means no transformation; the column values are used for partitioning. The other transforms apply a function to the column values and the data is partitioned by the transformed values.

Using CREATE TABLE ... PARTITIONED BY you create identity-partitioned Iceberg tables. Identity-partitioned Iceberg tables are similar to the Impala partitioned tables and are stored in the same directory structure as the Impala partitioned tables.

Impala supports 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 from Impala
Partition by year	years(time_stamp) year(time_stamp)	yes
Partition by month	months(time_stamp) month(time_stamp)	yes
Partition by a date value stored as int (dateint)	days(time_stamp) date(time_stamp)	no
Partition by hours	hours(time_stamp)	no
Partition by a dateint in hours	date_hour(time_stamp)	no
Partition by hashed value mod N buckets	bucket(N, col)	yes
Partition by value truncated to L, which is a number of characters	truncate(L, col)	yes

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.

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)

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.

```
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;
```

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.

Impala syntax

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

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')
```

Select Iceberg data feature

You can read Iceberg tables from Impala as you would any table. Joins, aggregations, and analytical queries, for example, are supported.

Impala supports reading V2 tables with [position deletes](#).

Impala example

```
SELECT * FROM ice_t;

SELECT count(*) FROM ice_t i LEFT OUTER JOIN other_t b
ON (i.id = other_t.fid)
WHERE i.col = 42;
```

Schema evolution feature

You learn that the 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 Impala table schema. A change to the schema of the Iceberg table by an outside entity, such as Spark, changes the corresponding Impala table. You can change the Iceberg table using ALTER TABLE to make the following changes:

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.

Impala syntax

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

ALTER TABLE table_name CHANGE COLUMN col_name1 col_name2 type

ALTER TABLE table_name DROP COLUMN col_name
```

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 Impala, you can base a new Iceberg table on a schema in a Parquet file.

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. 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.

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, ...)]
```

Impala example

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

Time travel feature

From 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.

Snapshot storage is incremental and dependent on the frequency and scale of updates. By default, Impala uses 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.

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.

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;
```

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.

Impala syntax

```
TRUNCATE [TABLE] table_name
```

Impala example

```
TRUNCATE t;
```

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).

Making row-level changes on V2 tables only

Learn the types of workloads best suited for Iceberg and when using V2 tables might improve query response.

Iceberg atomic DELETE and UPDATE operations resemble traditional RDBMS systems, but are not suitable for OLTP workloads. Iceberg is not designed to handle high frequency transactions. To handle very large datasets and frequent updates, use Apache Kudu.

Use Iceberg for managing large, infrequently changing datasets. You can update and delete Iceberg V2 tables at the [row-level](#) and not incur the overhead of rewriting the data files of V1 tables. Iceberg stores information about the deleted records in [position delete files](#). These files store the file paths and positions of the deleted records, eliminating the need to rewrite the files. Iceberg performs a DELETE plus an INSERT operation in a single transaction. This technique speeds up queries. Query engines scan the data files and delete files associated with a snapshot and merge them, removing the deleted rows. For example, to remove all data belonging to a single customer:

```
DELETE FROM ice_tbl WHERE user_id = 1234;
```

To update a column value in a specific record:

```
UPDATE ice_tbl SET col_v = col_v + 1 WHERE id = 4321;
```

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

Performance tuning

Impala uses its own C++ implementation to deal with Iceberg tables. This implementation provides significant performance advantages over other engines.

To tune performance, try the following actions:

- 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)
- Read Iceberg V2 tables from Hive using vectorization when heavy table scanning occurs as in `SELECT COUNT(*) FROM TBL_ICEBERG_PART`.
 - `set hive.llap.io.memory.mode=cache;`
 - `set hive.llap.io.enabled=true;`
 - `set hive.vectorized.execution.enabled=true;`
- Use Iceberg from Impala for querying Iceberg tables when latency is a concern.

The massively parallel SQL query engine, backend executors written in C++, and frontend (analyzer, planner) written in Java delivers query results fast.

- Cache manifest files as described in the next topic.

Caching manifest files

Apache Iceberg provides a mechanism to cache the contents of Iceberg manifest files in memory. The manifest caching feature helps to reduce repeated reads of small Iceberg manifest files from remote storage by Impala Coordinators and Catalogd.

Impala caches table metadata in CatalogD and the local Coordinator's catalog, making table metadata analysis fast if the targeted table metadata and files were previously accessed. Impala might analyze the same table multiple times across concurrent query planning and also within single query planning, so caching is very important.

Having a frontend written in Java, Impala can directly analyze many aspects of the Iceberg table metadata through the Java Library provided by Apache Iceberg. Metadata analysis such as listing the table data file, selecting the table snapshot, partition filtering, and predicate filtering is delegated through Iceberg Java API.

To use the Iceberg Java API while still maintaining fast query planning, Iceberg implements caching strategies in the Iceberg Java Library similar to those used by Apache Impala. The Iceberg manifest caching feature constitutes these caching strategies. For more information about manifest caching, see the [Iceberg Manifest Caching Blog](#).

Configuring manifest caching in Cloudera Manager

You can enable or disable manifest caching for Impala Coordinators and Catalogd by setting properties in Cloudera Manager.

About this task

By default, only 8 catalogs can have their manifest cache active in memory.

To connect to more than 8 HDFS clusters, in Cloudera Manager, configure `iceberg.io.manifest.cache.fileio-max` in `catalogd_java_opts` and the coordinator `impalad_embedded_java_opts`.

In the following task, you enable Iceberg manifest caching for the Impala Coordinator and Catalog Server.

Procedure

1. Navigate to Cloudera Manager. .
2. Search for Iceberg.

Enable Iceberg manifest caching

☒ IMPALA-1 (Service-Wide)

iceberg_manifest_cache_enabled

 iceberg_manifest_cache_enabled

By default, manifest caching is enabled, but if you have turned it off, check Impala-1 (Service-Wide) to re-enable.

Unsupported features and limitations

Cloudera does not support all features in Apache Iceberg. The list of unsupported features for Cloudera Data Platform (CDP) differs from release to release. Also, Apache Iceberg in CDP has some limitations you need to understand.

Unsupported features

The following features are not supported in this release of CDP:

- Tagging and branching

A technical preview is supported from Hive (not Impala or Spark) in Cloudera Data Warehouse Public Cloud.

- Equality deletes
- Reading files outside the table directory

An unauthorized party who knows the underlying schema and file location outside the table location can rewrite the manifest files within one table location to point to the data files in another table location to read your data.

- Buckets defined from Hive do not create like buckets in Iceberg.

For more information, see "Bucketing workaround" below.

- Using Iceberg tables as Spark Structured Streaming sources or sinks
- PyIceberg
- Migration of Delta Lake tables to Iceberg

Limitations

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](#).
- Equality updates and deletes are not supported as previously mentioned.
- 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
- An Iceberg table that points to another Iceberg table in the HiveCatalog is not supported.

For example:

```
CREATE EXTERNAL TABLE ice_t
STORED BY ICEBERG
TBLPROPERTIES ('iceberg.table_identifier'='db.tb');
```

- See also Iceberg data types [limitations](#) and unsupported data types.

Bucketing workaround

A query from Hive to define buckets/folders in Iceberg do not create the same number of buckets/folders as the same query creates in Hive. In Hive bucketing by multiple columns using the following clause creates 64 buckets maximum inside each partition.

```
| CLUSTERED BY (
|   id,
|   partition_id)
| INTO 64 BUCKETS
```

Defining bucketing from Hive on multiple columns of an Iceberg table using this query creates 64*64 buckets/folders; consequently, bucketing by group does not occur as expected. The operation will create many small files at scale, a drag on performance.

Add multiple bucket transforms (partitions) to more than one column in the current version of Iceberg as follows:

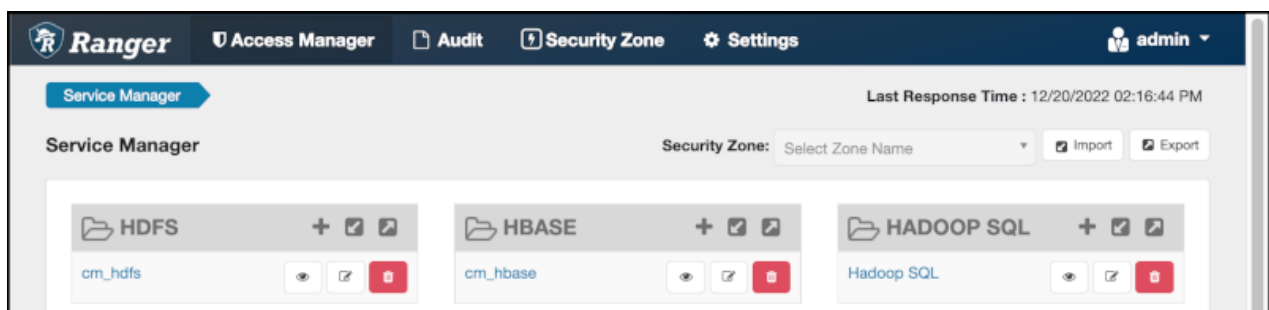
```
bucket(p, col1, col2) =[ bucket(m, col1) , bucket(n, col2) ] where p = m * n
```

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.

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**.

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



Policies for accessing tables on HDFS

The default policies that appear differ from service to service. You need to set up two Hadoop SQL policies to query Iceberg tables:

- One to authorize users to access the Iceberg files
Follow steps in "Editing 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 on HDFS or S3" below.

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 storage handler policy to access Iceberg files on HDFS or S3

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 *.

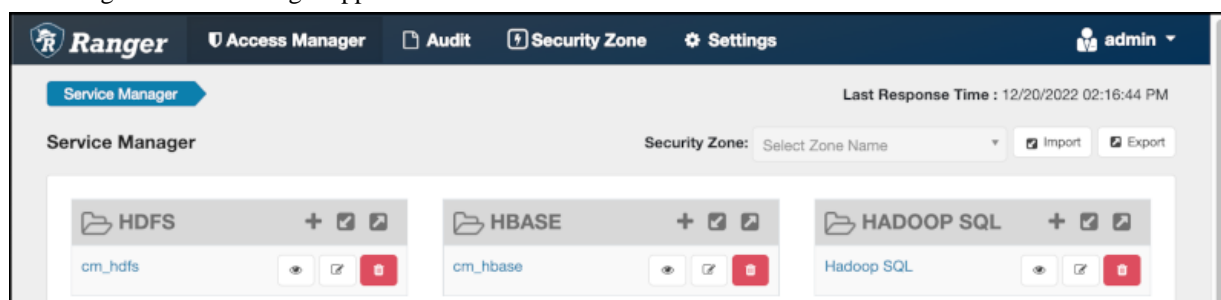
The Hadoop SQL Storage Handler policy supports only the RW Storage permission. A user having the required RW Storage permission on a resource, such as Iceberg, that you specify in the storage-type properties, is allowed only to reference the table location (for create/alter operations) in Iceberg. The 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

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.

The screenshot shows the 'Policy Type' set to 'Access' and 'Policy ID' set to '11'. The 'Policy Name' is 'all - storage-type, storage-uri' with an 'Enabled' toggle. The 'Policy Label' is 'Policy Label'. The 'storage-type' dropdown is set to 'iceberg'. The 'Storage URL' is '*' with an 'Include' toggle.

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:

The 'Allow Conditions' section shows three columns: 'Select Role', 'Select Group', and 'Select User'. The 'Select User' column contains a list of users: hive, beacon, dpprofiler, hue, admin, impala, and systest. The 'Select Role' and 'Select Group' columns are empty.

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 SQL 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.

The screenshot shows the 'Create Policy' form in the Ranger Admin Web UI. The breadcrumb trail at the top indicates the path: Service Manager > Hadoop SQL Policies > Create Policy. The form is titled 'Create Policy'. Under the 'Policy Details' section, the 'Policy Type' is set to 'Access'. The 'Policy Name' is 'IcebergPolicy1' and has an 'Enabled' toggle. The 'Policy Label' is 'Policy Label'. There are three rows for permissions: 'database' (dropdown) with 'icedb' and an 'Include' toggle; 'table' (dropdown) with 'icetable' and an 'Include' toggle; and 'column' (dropdown) with '*' and an 'Include' toggle.

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.

Accessing Iceberg files in Ozone

Learn how to set up policies to give users access to Iceberg files in Ozone. For example, if you query Iceberg tables from Impala, you must set up a Hadoop SQL access policy and Ozone file system access policy.

About this task

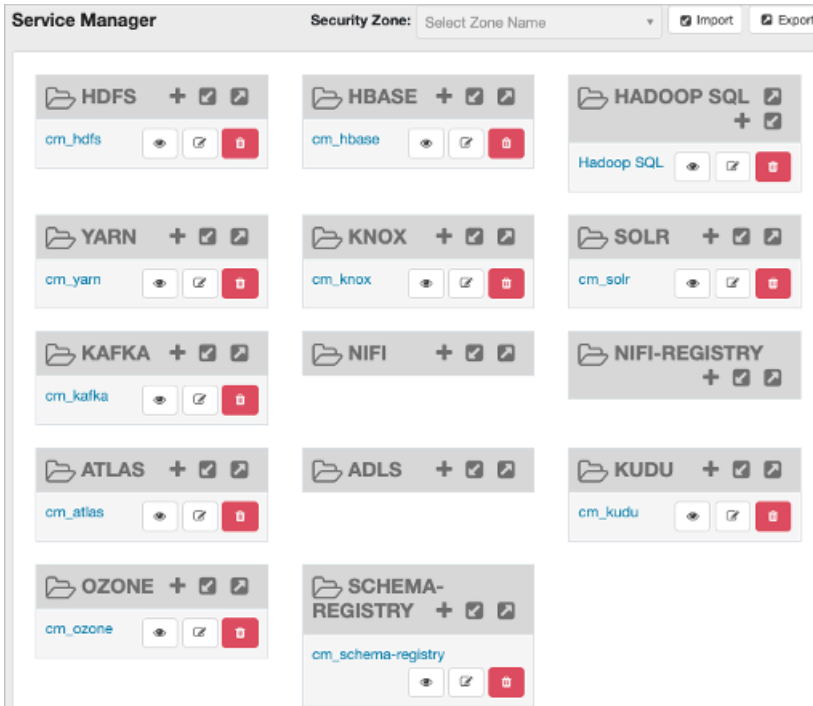
When Ranger is enabled in the cluster, any user other than the default admin user, "om" requires the necessary Ranger permissions and policy updates to access the Ozone filesystem. To create an Iceberg table on the Ozone file system, you need Ranger permissions.


In this task, you first enable Ozone in the Ranger service, and then set up the required policies.

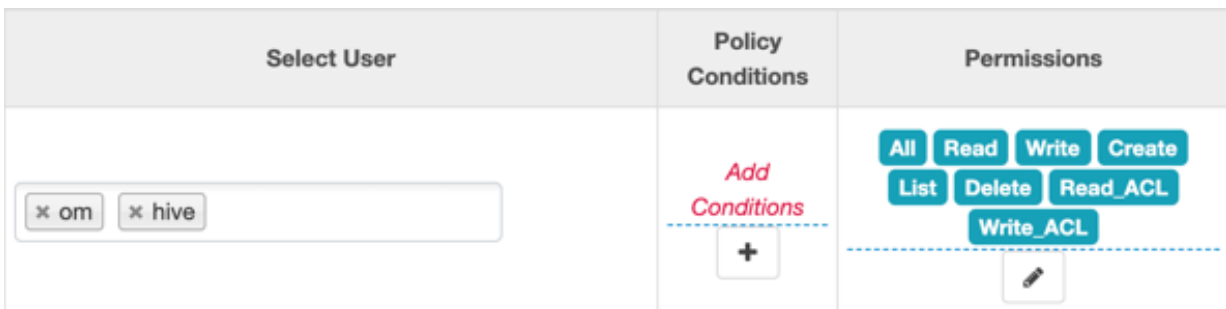
Procedure

1. In Cloudera Manager, click **Clusters Ozone Configuration** to navigate to the configuration page for Ozone.
2. Search for `ranger_service`, and enable the property.

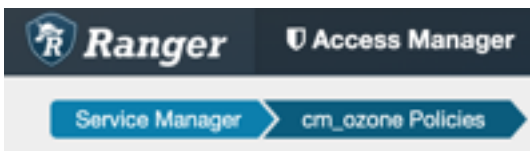
- Click Clusters Ranger Ranger Admin Web UI , enter your user name and password, then click Sign In. The **Service Manager for Resource Based Policies** page is displayed in the Ranger console.




- Click the cm_ozone preloaded resource-based service to modify an Ozone policy.
- In the cm_ozone policies page, click the Policy ID or click  Edit against the "all - volume, bucket, key" policy to modify the policy details.
- In the Allow Conditions pane, add roles, groups, or users, choose the necessary permissions, and then click Save.



- Click the Service Manager link in the breadcrumb trail and then click the Hadoop SQL preloaded resource-based service to update the Hadoop SQL URL policy.



- In the Hadoop SQL policies page, click the Policy ID or click  Edit against the "all - url" policy to modify the policy details.

9. Select roles, users, or groups in addition to the default.

By default, "hive", "hue", "impala", "admin" and a few other users are provided access to all the Ozone URLs. To grant everyone access, add the "public" group to the group list. Every user is then subject to your allow conditions.

Select Group	Select User	Permissions
<input type="text" value="x public"/>	<div> <input type="checkbox"/> hive <input type="checkbox"/> beacon <input type="checkbox"/> dpprofiler </div> <div> <input type="checkbox"/> hue <input type="checkbox"/> admin <input type="checkbox"/> impala </div>	<div> <input type="checkbox"/> select <input type="checkbox"/> update <input type="checkbox"/> Create <input type="checkbox"/> Drop <input type="checkbox"/> Alter <input type="checkbox"/> Index </div> <div> <input type="checkbox"/> Lock <input type="checkbox"/> All <input type="checkbox"/> Read <input type="checkbox"/> Write <input type="checkbox"/> ReplAdmin </div> <div> <input type="checkbox"/> Service Admin <input type="checkbox"/> Temporary UDF Admin <input type="checkbox"/> Refresh </div> <div> <input type="checkbox"/> RW Storage </div>

Creating an Iceberg table

You create Iceberg tables from Impala from the command line or from Hue. See how to use the Impala shell to query Iceberg files from the Impala command line.

About this task

When you connect to an impalad running on the same machine, the prompt reflects the current hostname.

```
$ impala-shell
```

When you connect to an impalad running on a remote machine, and impalad listens on a non-default port over the HTTP HiveServer2 protocol.

```
$ impala-shell -i some.other.hostname:port_number --protocol='hs2-http'
```

Before you begin

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

Procedure

1. Note the name of the host running the impalad daemon in your CDP cluster.
If impalad uses a non-default port (something other than port 21050) for impala-shell connections, also note the port number.
2. Use the -i option to connect to impalad.

```
impala-shell -i <hostname:port number> --protocol='hs2-http'
```

3. Enter a query to create a simple Iceberg table in the default Parquet format.

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

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 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](#).

Procedure

1. Select, or use, a database.
2. Create an identity-partitioned table and run the query.

Impala:

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

3. Create a table and specify an identity transform, such as bucket, truncate, or date, using the Iceberg V2 `PARTITION BY SPEC` clause.

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;
```

Expiring snapshots

You can expire snapshots of an Iceberg table using an `ALTER TABLE` query. 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 05:39:18.6890000
00');
```

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;
```

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.

Procedure

1. Use a database.

For example:

```
USE mydatabase;
```

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.

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.

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 obtain 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 customer_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, usern
ame, 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;
```

Iceberg data types

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

Iceberg supported data types

Table 1:

Iceberg data type	SQL data type	Impala
binary		BINARY
boolean	BOOLEAN	BOOLEAN
date	DATE	DATE
decimal(P, S)	DECIMAL(P, S)	DECIMAL (P, S)

Iceberg data type	SQL data type	Impala
double	DOUBLE	DOUBLE
fixed(L)		Not supported
float	FLOAT	FLOAT
int	TINYINT, SMALLINT, INT	INTEGER
list	ARRAY	Read only
long	BIGINT	BIGINT
map	MAP	Read only
string	VARCHAR, CHAR	STRING
struct	STRUCT	Read only
time		Not supported
timestamp	TIMESTAMP	TIMESTAMP
timestampz	TIMESTAMP WITH LOCAL TIME ZONE	Read timestampz into TIMESTAMP values Writing not supported
uuid	none	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.

Iceberg supports two timestamp types:

- timestamp (without timezone)
- timestampz (with timezone)

In Spark 3.3 and earlier, Spark SQL supports a single TIMESTAMP type, which maps to the Iceberg timestampz type. However, Impala is unable to write to Iceberg tables with timestampz columns. To create Iceberg tables from Spark with timestamp rather than timestampz columns, set the following configurations to true:

- spark.sql.iceberg.handle-timestamp-without-timezone
- spark.sql.iceberg.use-timestamp-without-timezone-in-new-tables

Configure these properties only on Spark 3.3 and earlier.

Spark still handles the timestamp column as a timestamp with local timezone. Inconsistent results occur unless Spark is running in UTC.

Unsupported data types

Impala does not support the following Iceberg data types:

- TIMESTAMPTZ (only read support)
- FIXED
- UUID

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.

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.
- `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)