Hands On Lab - Introduction to Apache Iceberg on Cloudera Data Warehouse

This document explores some of the core features of Apache Iceberg, including table creation, data insertion, schema evolution, and time travel.

Enter your assigned username e.g. user001

By entering your assigned username and clicking the **Update Examples** button, the code examples you'll be running below will make it easy to identify which tables are yours. Click the reset button to start again if you make a mistake.

e.g., john.doe | Update Examples | Reset

```
changed_user=USERNAME;
```

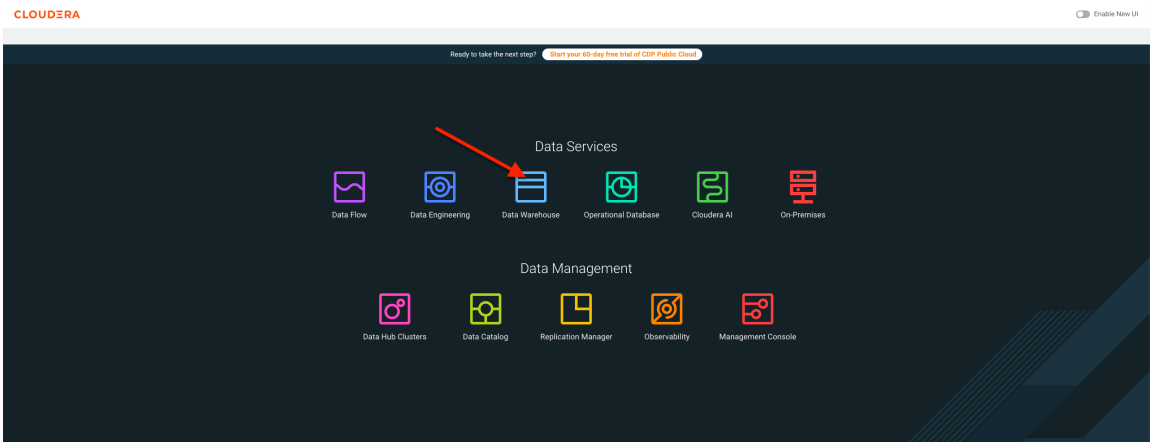
Important

If you are unable to use the web version of this documentation (functionality above), ensure you change "USERNAME" in all the code to your assigned user:
e.g. `DROP TABLE IF EXISTS default.USERNAME_managed_countries;` is changed to `DROP TABLE IF EXISTS default.user001_managed_countries;`

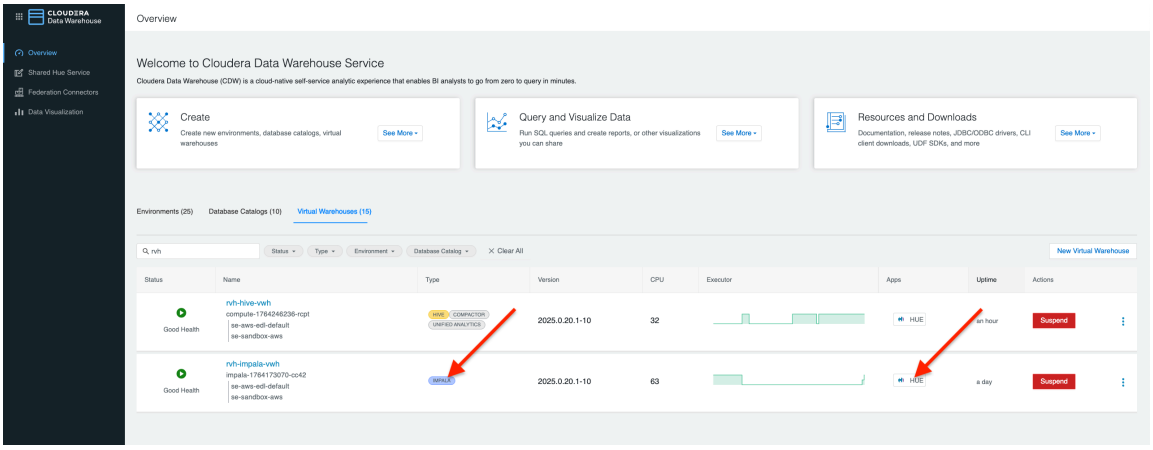
Table of Contents

Open Impala In Cloudera Data Warehouse Service

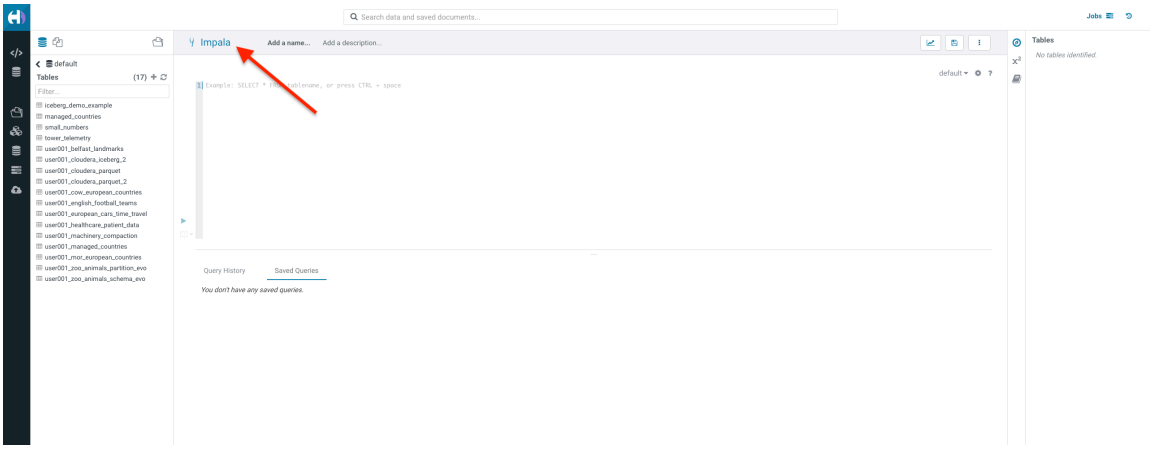
In the Cloudera Control Place select the Data Warehouse tile.



In the Data Warehouse Service locate the **Impala** Virtual Warehouse and click on the Hue button.



This will open the Hue IDE for Impala and you're ready to proceed.



1. Creating Iceberg Tables

What is an Iceberg table?

An **Iceberg Table** is a table where Iceberg manages both the metadata and the data itself. It is a fully integrated table that Iceberg can track and manage. When you drop an Iceberg Table, both the metadata and the data are removed.

Use an Iceberg Table when:

- You need Iceberg to fully handle both the data and metadata.
- You want to manage the entire lifecycle of the table automatically.
- You require atomic operations, such as partition evolution, schema evolution, and time travel.

Key Benefits and Limitations

Benefits:

- Simplified data management.
- Automatic metadata handling.
- Built-in features like time travel and schema evolution.

Limitations:

- Dropping the table automatically deletes all data.

Note

By default, when creating an Iceberg table, it will be a **Copy-on-Write (COW)** table. You can explicitly specify the table type as Copy-on-Write (COW) or Merge-on-Write (MOR) using table properties.

Table Creation Example

IMPALA

```
-- Drop the table if it exists
DROP TABLE IF EXISTS default.USERNAME_managed_countries;

-- Create the table in Impala
CREATE TABLE default.USERNAME_managed_countries (
    country_code STRING,
    country_name STRING,
    population INT,
    area DOUBLE
) STORED BY ICEBERG;

-- Insert data into the table
INSERT INTO default.USERNAME_managed_countries VALUES
    ('FR', 'France', 67391582, 643801.0),
    ('DE', 'Germany', 83149300, 357022.0),
    ('IT', 'Italy', 60262770, 301340.0);

-- Read data from the table
SELECT * FROM default.USERNAME_managed_countries;

-- Describe the table to show its schema
DESCRIBE FORMATTED default.USERNAME_managed_countries;

-- Show the table's creation script
SHOW CREATE TABLE default.USERNAME_managed_countries;
```

2. Inserts, Updates, and Deletes

In Iceberg, data manipulation (inserts, updates, deletes) is performed using standard SQL commands.

Inserting & Updating Data

Updates modify existing records based on a condition.

Best Practices:

- Ensure the schema is well-defined.
- Perform updates only when necessary to avoid frequent schema changes.
- Monitor table performance as data grows, especially with large updates.

Code Example

IMPALA

```
-- Drop the table if it exists
DROP TABLE IF EXISTS default.USERNAME_english_football_teams;

-- Create the table for football teams in England
CREATE TABLE default.USERNAME_english_football_teams (
    team_id STRING,
    team_name STRING,
    team_city STRING,
    team_stadium STRING
) STORED BY ICEBERG;

-- Inserting data into the table
INSERT INTO default.USERNAME_english_football_teams VALUES
    ('T001', 'Manchester United', 'Manchester', 'Old Trafford'),
    ('T002', 'Liverpool', 'Liverpool', 'Anfield'),
    ('T003', 'Chelsea', 'London', 'Stamford Bridge');

-- Select all data from the table
SELECT * FROM default.USERNAME_english_football_teams;

-- Update Stadium Name
UPDATE default.USERNAME_english_football_teams
SET team_stadium = 'New Stamford Bridge'
WHERE team_id = 'T003';

-- Select the updated data
SELECT * FROM default.USERNAME_english_football_teams;
```

Handling Data Deletions

Iceberg uses a **snapshot mechanism**, so deletions add a new snapshot but do not immediately remove the physical data. This ensures that deleted data can still be recovered.

Considerations:

- Deletions are versioned and can be reverted through time travel.
- You can configure Iceberg to perform data compaction after deletion for performance optimization.

Code Example

IMPALA

```
-- Drop the table if it exists
DROP TABLE IF EXISTS default.USERNAME_english_football_teams;

-- Create the table for football teams in England with ICEBERG storage
CREATE TABLE default.USERNAME_english_football_teams (
    team_id STRING,
    team_name STRING,
    team_city STRING,
    team_stadium STRING
) STORED BY ICEBERG;

-- Inserting data into the table
INSERT INTO default.USERNAME_english_football_teams VALUES
    ('T001', 'Manchester United', 'Manchester', 'Old Trafford'),
    ('T002', 'Liverpool', 'Liverpool', 'Anfield'),
    ('T003', 'Chelsea', 'London', 'Stamford Bridge');

-- Select all data from the table
SELECT * FROM default.USERNAME_english_football_teams;

-- Delete using Team ID
DELETE FROM default.USERNAME_english_football_teams WHERE team_id = 'T003';

-- Select the updated data
SELECT * FROM default.USERNAME_english_football_teams;
```

3. Iceberg Table Types (COW and MOR)

Iceberg tables support different storage strategies to balance performance, storage efficiency, and query speed. This section introduces the two primary approaches.

- **Copy-on-Write (COW)**: Ensures immutability by writing new files on every update, making it ideal for ACID transactions and historical auditing.
- **Merge-on-Read (MOR)**: Optimizes write performance by storing changes as delta files, merging them at query time—useful for real-time ingestion.

Each strategy has trade-offs, making them suitable for different workloads.

Merge-On-Read (MOR)

- Writes are efficient.
- Reads are less efficient due to read amplification, but regularly scheduled compaction can reduce inefficiency.
- A good choice when streaming.
- A good choice when frequently writing or updating, such as running hourly batch jobs.
- A good choice when the percentage of data change is low.

Copy-On-Write (COW)

- Reads are efficient.
- A good choice for bulk updates and deletes, such as running a daily batch job.
- Writes less efficient due to write amplification, but the need for compaction is reduced.
- A good choice when the percentage of data change is high.

Iceberg Copy-on-Write (COW) Table

What is it?

Copy-on-Write (COW) is where instead of modifying data directly, the system creates a complete copy of the data file with the changes applied. This method makes reading data incredibly fast and efficient, as queries can simply access a clean, final version of a file without any extra processing. The downside, however, is that writing data can be slow and expensive. Even a tiny update to a single row forces the entire file to be duplicated and rewritten. This makes frequent, small changes inefficient and can lead to conflicts if multiple writes occur at the same time. While this approach is poorly suited for minor edits, it becomes ideal for large, bulk updates where changing a significant portion of the file is necessary anyway.

How to create an COW Table:

IMPALA

```
DROP TABLE IF EXISTS default.USERNAME_cow_european_countries;

CREATE TABLE default.USERNAME_cow_european_countries (
    country_code STRING,
    country_name STRING,
    population BIGINT,
    area_km2 DOUBLE,
    last_updated TIMESTAMP
) STORED BY ICEBERG
TBLPROPERTIES (
    'write.format.default'='orc',
    'write.delete.mode'='copy-on-write',  -- Enable COW for delete operations
    'write.update.mode'='copy-on-write',  -- Enable COW for update operations
    'write.merge.mode'='copy-on-write'    -- Enable COW for compaction
);

DESCRIBE EXTENDED default.USERNAME_cow_european_countries;
```

Iceberg Merge-on-Read (MOR) Table

What is it?

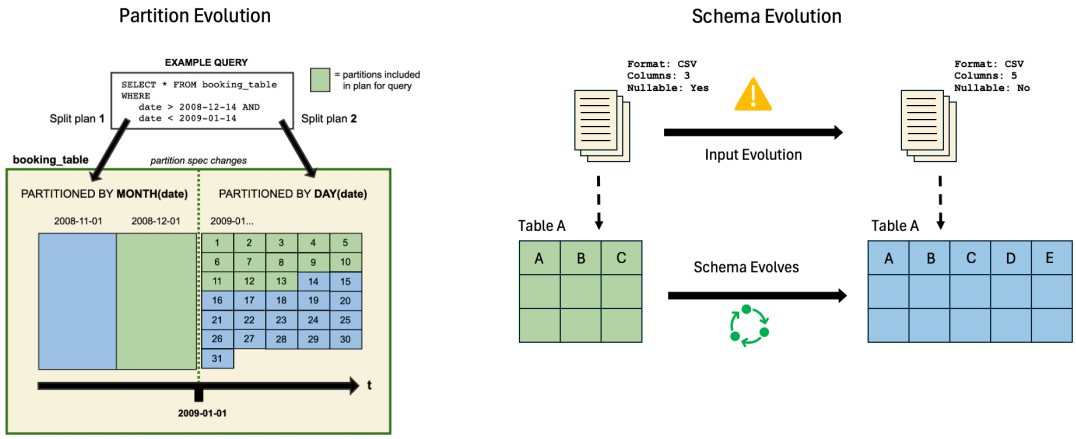
Merge-on-Read (MOR) is where, instead of rewriting large files for every modification, changes are simply recorded in separate, smaller files. This approach makes writing new data, like updates or deletions, significantly faster. The trade-off is that more work is required during a read operation; the system must combine the original data with the separate change files on the fly to present the most current version. In Apache Iceberg, this is handled using delete files. When you update or delete a row, the change is logged in a delete file. During a query, Iceberg uses these delete files to know which rows to ignore from the old data files and which new rows to include. Eventually, compaction merges the original data and all the changes into new, clean files, which speeds up future reads.

How to create an MOR Table:

IMPALA

```
DROP TABLE IF EXISTS default.USERNAME_mor_european_countries;

CREATE TABLE default.USERNAME_mor_european_countries (
    country_code STRING,
    country_name STRING,
    population BIGINT,
    area_km2 DOUBLE,
    last_updated TIMESTAMP
) STORED BY ICEBERG
TBLPROPERTIES (
    'format-version'='2',
    'write.format.default'='parquet',
    'write.delete.mode'='merge-on-read',
    'write.update.mode'='merge-on-read',
    'write.merge.mode'='merge-on-read'
);

DESCRIBE EXTENDED default.USERNAME_mor_european_countries;
```

4. Schema and Partition Evolution



Schema Evolution in Iceberg

Schema evolution in Iceberg allows you to modify the structure of your tables over time. This includes adding, renaming, and removing columns while ensuring that historical data remains accessible without requiring a full rewrite of the table.

Schema Evolution Operations include:

- Adding columns: You can add new columns without affecting the existing data or operations.
- Renaming columns: The renaming of columns is supported without requiring data migration.
- Changing column types: You can change the type of a column, as long as it is compatible with the existing data.
- Dropping columns: Columns can be safely dropped if they are no longer needed.

Why is Schema Evolution important?

- Adapting to business needs: As your data requirements evolve, schema changes are often necessary without disrupting production workflows.
- Backwards compatibility: Allows for schema changes that are compatible with existing data, meaning that you can evolve the schema without breaking old queries or affecting historical data.
- Simplifying data management: Allows incremental changes to the schema without needing full table rewrites.

Example Schema Evolution:

IMPALA

```
-- Drop the table if it exists
DROP TABLE IF EXISTS default.USERNAME_zoo_animals_schema_evo;

-- Create the initial Iceberg table
CREATE TABLE default.USERNAME_zoo_animals_schema_evo (
    animal_id STRING,
    animal_name STRING
) STORED BY ICEBERG;

-- Insert sample data
INSERT INTO default.USERNAME_zoo_animals_schema_evo VALUES
('A001', 'Lion'),
('A002', 'Elephant'),
('A003', 'Giraffe');

-- View the Data
SELECT * FROM default.USERNAME_zoo_animals_schema_evo;

-- View the schema
DESCRIBE FORMATTED default.USERNAME_zoo_animals_schema_evo;

-- Add a new column to the table
ALTER TABLE default.USERNAME_zoo_animals_schema_evo ADD COLUMNS (habitat
STRING);

-- Insert new data into the updated schema
INSERT INTO default.USERNAME_zoo_animals_schema_evo VALUES
('A004', 'Zebra', 'Savanna'),
('A005', 'Panda', 'Bamboo Forest');

-- View the Data
SELECT * FROM default.USERNAME_zoo_animals_schema_evo;

-- View the schema
DESCRIBE FORMATTED default.USERNAME_zoo_animals_schema_evo;

-- View the create table
SHOW CREATE TABLE default.USERNAME_zoo_animals_schema_evo;
```

Partition Evolution

artition evolution refers to the ability to modify the partitioning strategy of an Iceberg table after it has been created. This can involve changing the partitioning key (column used for partitioning) or adding new partitioning columns. Unlike traditional partitioning schemes, Iceberg allows for flexible partition evolution without needing to rewrite the entire dataset. Partitioning evolution can help optimize query performance and manage large datasets more efficiently.

Common Partitioning Strategies

- Time-based partitioning: Commonly used for time-series data, partitioning by date or timestamp can help in partition pruning, making queries faster.
- Range or hash partitioning: For datasets with discrete values like animal species, partitioning by a range of values or using hash partitioning can help balance the data across partitions.

You can change the partitioning strategy after the table has been created, even if the data already exists. This allows you to optimize partitioning as your query patterns evolve and take advantage of query optimization with regards to partition pruning as the data evolves and grows within the table.

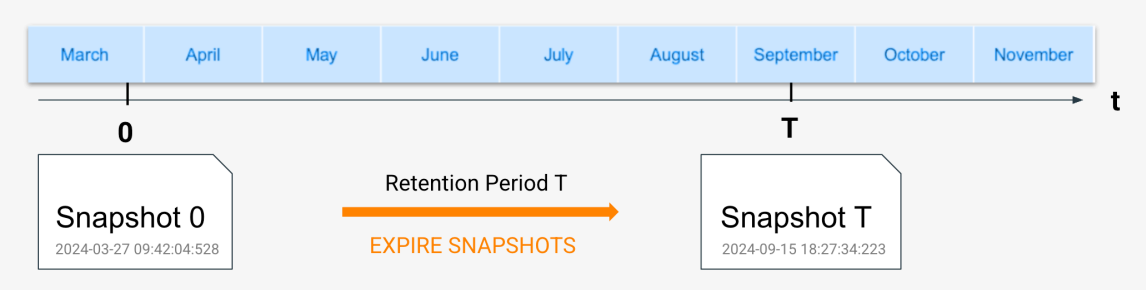
Example Partition Evolution:

IMPALA

```
-- DROP TABLE IF EXISTS
DROP TABLE IF EXISTS default.USERNAME_zoo_animals_partition_evo;

-- CREATE THE INITIAL ICEBERG TABLE PARTITIONED BY 'animal_id'
CREATE TABLE default.USERNAME_zoo_animals_partition_evo (
    species_name STRING,
    habitat STRING
)
PARTITIONED BY (animal_id STRING)
STORED BY ICEBERG;

-- INSERT DATA INTO THE INITIAL PARTITIONING SCHEME
INSERT INTO default.USERNAME_zoo_animals_partition_evo VALUES
    ('A001', 'Lion', 'Savanna'),
    ('A002', 'Tiger', 'Forest');

-- SHOW TABLE DESCRIPTION
DESCRIBE FORMATTED default.USERNAME_zoo_animals_partition_evo;

-- ALTER TABLE TO ADD A NEW PARTITION FIELD 'habitat'
ALTER TABLE default.USERNAME_zoo_animals_partition_evo
SET PARTITION SPEC (animal_id, habitat);

-- INSERT DATA AFTER THE PARTITION SCHEME CHANGE
INSERT INTO default.USERNAME_zoo_animals_partition_evo VALUES
    ('A003', 'Elephant', 'Grassland'),
    ('A004', 'Panda', 'Mountain');

-- SHOW TABLE DESCRIPTION AFTER CHANGE
DESCRIBE FORMATTED default.USERNAME_zoo_animals_partition_evo;

-- QUERY DATA
SELECT * FROM default.USERNAME_zoo_animals_partition_evo;
```

New data inserted after the partitioning change will adhere to the new scheme.

5. Time Travel and Rollbacks



Time Travel in Iceberg

Time travel in Iceberg allows you to query a table as it existed at a specific point in time in the past. This feature leverages Iceberg's snapshot-based architecture to track all changes made to the data over time. When you perform time travel, Iceberg will provide data based on the state of the table at a specified snapshot or timestamp. Time travel is supported by specifying a timestamp or snapshotid when querying the table, which enables access to historical data without having to maintain separate copies of the data.

Time Travel Benefits:

- Enables **historical queries** for auditing and investigating historical trends.
- Allows **data recovery** from accidental corruption.
- Simplifies rollbacks by querying an earlier snapshot.

Example Time Travel:

IMPALA

```
-- DROP TABLE IF EXISTS
DROP TABLE IF EXISTS default.USERNAME_european_cars_time_travel;

-- CREATE ICEBERG TABLE FOR EUROPEAN CARS
CREATE TABLE default.USERNAME_european_cars_time_travel (
    car_id STRING,
    car_make STRING,
    car_model STRING,
    car_country STRING
)
STORED BY ICEBERG;

-- INSERT INITIAL DATA
INSERT INTO default.USERNAME_european_cars_time_travel VALUES
    ('C001', 'Volkswagen', 'Golf', 'Germany'),
    ('C002', 'BMW', 'X5', 'Germany');

-- LIST THE AVAILABLE SNAPSHOTS
SELECT * FROM default.USERNAME_european_cars_time_travel.snapshots;

-- Perform an update operation to modify the data
UPDATE default.USERNAME_european_cars_time_travel
SET car_model = 'Polo'
WHERE car_id = 'C001';

-- LIST THE AVAILABLE SNAPSHOTS AFTER UPDATE
SELECT * FROM default.USERNAME_european_cars_time_travel.snapshots;

-- INSERT NEW CAR DATA (NOT A EUROPEAN CAR)
INSERT INTO default.USERNAME_european_cars_time_travel
VALUES ('C003', 'FORD', 'F150', 'USA');

-- LIST THE AVAILABLE SNAPSHOTS AFTER INSERT
SELECT * FROM default.USERNAME_european_cars_time_travel.snapshots;

-- CHECK CURRENT TABLE DATA
SELECT * FROM default.USERNAME_european_cars_time_travel;

-- FETCH THE SNAPSHOT ID BEFORE MOST RECENT APPEND (i.e. The operation should =
overwrite)
SELECT * FROM default.USERNAME_european_cars_time_travel.snapshots;

-- TIME TRAVEL TO BEFORE THE INSERT OF THE USA CAR
SELECT *
FROM default.USERNAME_european_cars_time_travel
FOR SYSTEM_VERSION AS OF <USE_SNAPSHOT_ID_FROM_PREVIOUS_QUERY>;
```

Rollback Using Snapshots

Rollback in Iceberg allows you to revert the table's state to a specific snapshot, undoing any subsequent changes. This is useful in scenarios where data corruption, accidental deletion, or

unwanted changes occur. By rolling back to a previous snapshot, you can restore the table to its desired state.

Key Points on Rollback:

- Rollback to a Snapshot: You can roll back the table by specifying a snapshotid that corresponds to the point in time you wish to revert to.
- How it works: The rollback operation rewrites the table to the state of the specified snapshot, effectively "reverting" any changes made after that snapshot.
- Usage: Rollback can be useful in production environments where you need to ensure data integrity and recover from accidental modifications.

Example Rollback:

IMPALA

```
-- DROP TABLE IF EXISTS
DROP TABLE IF EXISTS default.USERNAME_european_cars_time_travel;

-- CREATE ICEBERG TABLE FOR EUROPEAN CARS
CREATE TABLE default.USERNAME_european_cars_time_travel (
    car_id STRING,
    car_make STRING,
    car_model STRING,
    car_country STRING
)
STORED BY ICEBERG;

-- INSERT INITIAL DATA
INSERT INTO default.USERNAME_european_cars_time_travel VALUES
    ('C001', 'Volkswagen', 'Golf', 'Germany'),
    ('C002', 'BMW', 'X5', 'Germany');

-- LIST THE AVAILABLE SNAPSHOTS
SELECT * FROM default.USERNAME_european_cars_time_travel.snapshots;

-- Perform an update operation to modify the data
UPDATE default.USERNAME_european_cars_time_travel
SET car_model = 'Polo'
WHERE car_id = 'C001';

-- LIST THE AVAILABLE SNAPSHOTS AFTER UPDATE
SELECT * FROM default.USERNAME_european_cars_time_travel.snapshots;

-- INSERT NEW CAR DATA (NOT A EUROPEAN CAR)
INSERT INTO default.USERNAME_european_cars_time_travel
VALUES ('C003', 'FORD', 'F150', 'USA');

-- FETCH THE SNAPSHOT ID BEFORE MOST RECENT APPEND (i.e. The Operation should =
overwrite)
SELECT * FROM default.USERNAME_european_cars_time_travel.snapshots;

-----------------------------------------------------------------------------
-
-- ROLLBACK TO AN EARLIER SNAPSHOT (Manual Process in Impala)
-----------------------------------------------------------------------------
-

ALTER TABLE default.USERNAME_european_cars_time_travel
EXECUTE ROLLBACK(<USE_SNAPSHOT_ID_FROM_PREVIOUS_QUERY>);

-- CHECK DATA AFTER ROLLBACK
SELECT * FROM default.USERNAME_european_cars_time_travel;
```

6. Table Migration

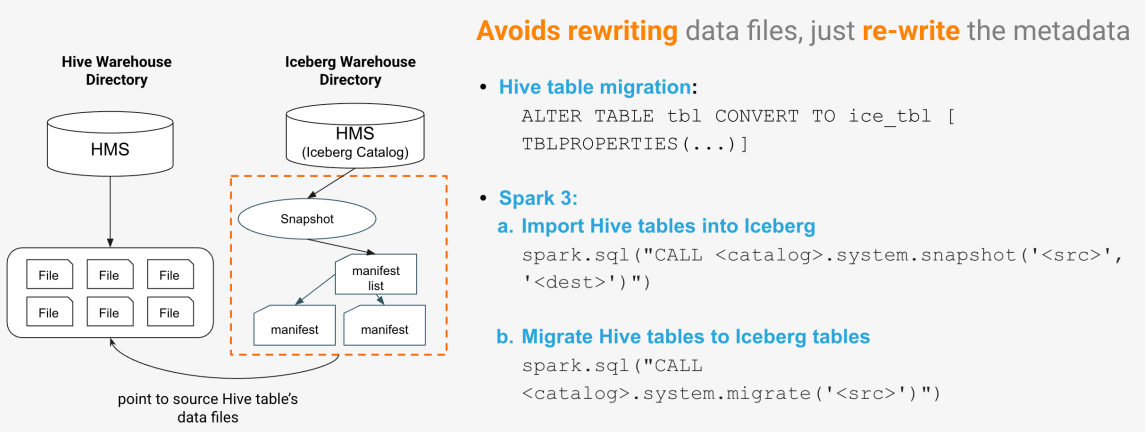
Iceberg Table Migration

Two primary methods exist for migrating existing tables (e.g. Parquet) to Iceberg:

> **Info**
>
> Using an in-place table migration is the fastest, most efficient way to convert tables to Iceberg tables as just the metadata is rewritten, not the data files. CTAS offers greater flexibility for creating a new table and controlling the migration process.

1. **In-Place Migration:**

In-place migration from Parquet to Iceberg allows seamless conversion without moving data or creating a new table and is almost instantaneous as only the metadata is rewritten. Data files are not affected.



Code Example:

IMPALA

```
-- STEP 1: CREATE A REGULAR PARQUET TABLE WITH SAMPLE DATA
DROP TABLE IF EXISTS default.USERNAME_cloudera_parquet;

CREATE EXTERNAL TABLE default.USERNAME_cloudera_parquet (
    cloudera_employee STRING,
    cloudera_role STRING
)
STORED AS PARQUET;

-- STEP 2: INSERT RECORDS INTO THE PARQUET TABLE
INSERT INTO default.USERNAME_cloudera_parquet VALUES
    ('Joe Cur', 'SE'),
    ('Jane Pas', 'PS');

-- STEP 3: DISPLAY THE CONTENTS OF THE PARQUET TABLE
SELECT * FROM default.USERNAME_cloudera_parquet;

-- STEP 4: DESCRIBE THE PARQUET TABLE BEFORE MIGRATION
DESCRIBE FORMATTED default.USERNAME_cloudera_parquet;

-- STEP 5: CONVERT TO AN ICEBERG TABLE (Metadata Rewrite)
ALTER TABLE default.USERNAME_cloudera_parquet CONVERT TO ICEBERG;

-- STEP 6 - VALIDATE CONVERSION WAS SUCCESSFUL
DESCRIBE FORMATTED default.USERNAME_cloudera_parquet;
```

2. **CTAS Migration (CREATE TABLE AS SELECT):**

Creates a **new** Iceberg table and inserts data from an existing Parquet/Hive table. This is ideal when a fresh Iceberg table is needed or more control over the migration process is desired.

Code Example:

IMPALA

```
-- STEP 1: CREATE A REGULAR PARQUET TABLE WITH SAMPLE DATA
DROP TABLE IF EXISTS default.USERNAME_cloudera_parquet_2;

CREATE EXTERNAL TABLE default.USERNAME_cloudera_parquet_2 (
    cloudera_employee STRING,
    cloudera_role STRING
)
STORED AS PARQUET;

-- STEP 2: INSERT RECORDS INTO THE PARQUET TABLE
INSERT INTO default.USERNAME_cloudera_parquet_2 VALUES
    ('Joe Cur', 'SE'),
    ('Jane Pas', 'PS');

-- STEP 3: DISPLAY THE CONTENTS OF THE PARQUET TABLE
SELECT * FROM default.USERNAME_cloudera_parquet_2;

-- STEP 4: DESCRIBE THE PARQUET TABLE BEFORE MIGRATION
DESCRIBE FORMATTED default.USERNAME_cloudera_parquet_2;

-- STEP 5 - CREATE TABLE AS X
CREATE TABLE default.USERNAME_cloudera_iceberg_2 STORED BY ICEBERG AS
SELECT * FROM default.USERNAME_cloudera_parquet_2;
```

7. Table Maintenance

Iceberg Compaction

Iceberg compaction is the process of merging small data files within an Iceberg table into larger files to improve query performance and reduce metadata overhead. Iceberg writes immutable files, and over time, frequent inserts, updates, and deletes can lead to many small files that impact efficiency.

Why Compaction is Important:

- Optimizes Read Performance by reducing the number of files scanned.
- Reduces Metadata Overhead.
- Enhances Storage Efficiency.

What's the Impact?

- Reduced Query Latency: Faster scans with fewer files.
- Lower Metadata Management Overhead: Smaller metadata files and fewer manifest entries.
- Potential Higher Write Costs: Compacting too frequently can increase write costs if not managed properly.

Code Example:

IMPALA

```
-- DROP THE TABLE IF IT EXISTS
DROP TABLE IF EXISTS default.USERNAME_machinery_compaction;

-- CREATE A NEW ICEBERG TABLE FOR CONSTRUCTION MACHINERY
CREATE TABLE default.USERNAME_machinery_compaction (
    machine_id STRING,
    model STRING,
    manufacturer STRING,
    weight DOUBLE,
    status STRING
)
STORED BY ICEBERG;

-- INSERT MULTIPLE SMALL FILES BY WRITING DATA IN MULTIPLE TRANSACTIONS
INSERT INTO default.USERNAME_machinery_compaction VALUES
    ('M001', 'Excavator X1', 'Caterpillar', 12500.5, 'Active'),
    ('M002', 'Bulldozer B2', 'Komatsu', 14500.0, 'Inactive');

INSERT INTO default.USERNAME_machinery_compaction VALUES
    ('M003', 'Crane C3', 'Liebherr', 17500.2, 'Active'),
    ('M004', 'Dump Truck D4', 'Volvo', 22000.8, 'Active');

INSERT INTO default.USERNAME_machinery_compaction VALUES
    ('M005', 'Concrete Mixer CM5', 'Schwing Stetter', 9500.6, 'Inactive'),
    ('M006', 'Loader L6', 'John Deere', 12800.4, 'Active');

-- REWRITE DATA FILES TO OPTIMIZE FILE SIZES (SIZE 100MB)
OPTIMIZE TABLE default.USERNAME_machinery_compaction
(FILE_SIZE_THRESHOLD_MB=100);
```

Iceberg Expiring Snapshots

Iceberg maintains a history of table snapshots, allowing for time travel and rollback. Expiring snapshots is the process of removing older snapshots that are no longer needed to free up storage and improve metadata performance.

Why is it important?

- Manages Storage Growth: Prevents unnecessary accumulation of outdated snapshots.
- Improves Query Planning: Reduces metadata size, making query planning more efficient.
- Controls Data Retention: Helps enforce compliance policies by retaining only necessary snapshots.

What's the Impact?

- Frees Up Storage Space: Reduces disk usage by removing old metadata and data files.

- Improves Query Performance: Smaller metadata means faster query planning.
- Irreversible Data Loss: Once expired, snapshots cannot be restored, so retention policies must be carefully set.
- Cleans Up Old Manifest Files: Expiring snapshots removes outdated manifest files, keeping metadata management efficient.

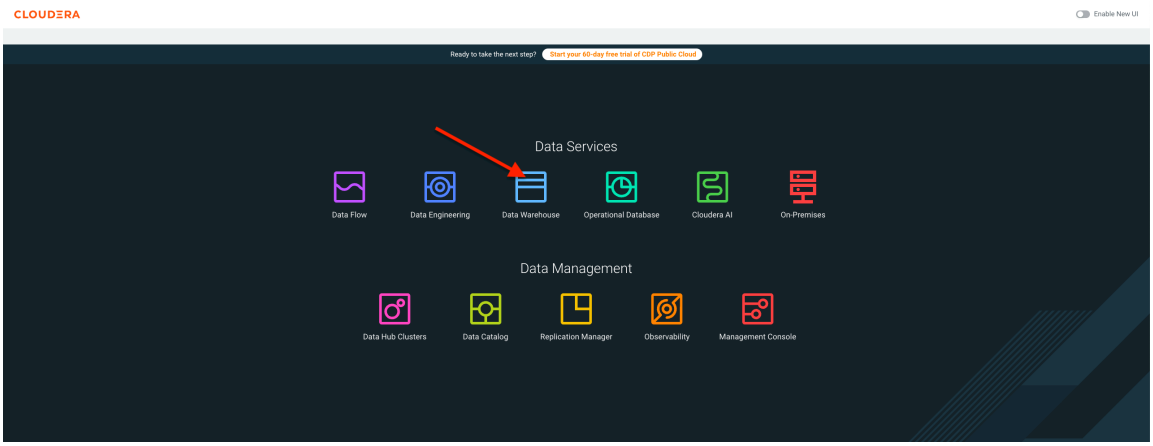
Code Example:

IMPALA

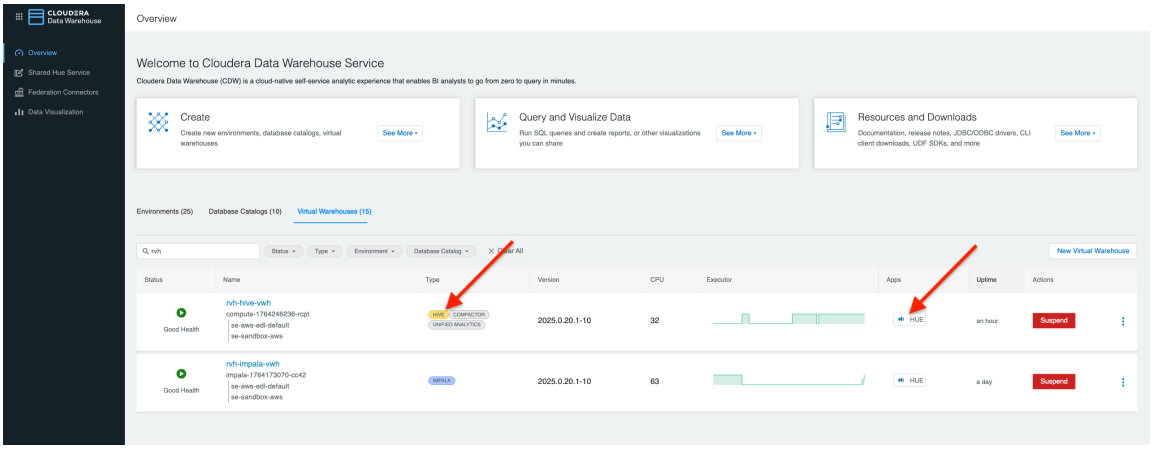
```
-- VIEW ALL SNAPSHOTS FOR THE TABLE
SELECT * FROM default.USERNAME_machinery_compaction.snapshots;

-- GET THE LATEST(NEWEST) SNAPSHOT TIMESTAMP FOR THE TABLE
SELECT MAX(committed_at) FROM default.USERNAME_machinery_compaction.snapshots;

-- EXPIRE ALL SNAPSHOTS ON THE TABLE APART FROM THE LATEST. COPY AND PASTE THE
TIMESTAMP RETURNED FROM THE QUERY ABOVE TO THE STATMENT BELOW.
-- SNAPSHOTS CAN BE EXPIRED BY SNAPSHOT ID OR IN THIS CASE ANYTHING OLDER THAN
THE TIMESTAMP PROVIDED
ALTER TABLE  default.USERNAME_machinery_compaction EXECUTE
EXPIRE_SNAPSHOTS('<timestamp_from_query_above>')

-- VIEW ALL SNAPSHOTS FOR THE TABLE AFTER EXPIRING
SELECT * FROM default.USERNAME_machinery_compaction.snapshots;

-- VERIFY THE CURRENT STATE OF THE TABLE
SELECT * FROM default.USERNAME_machinery_compaction;
```

Open Hive In Cloudera Data Warehouse Service

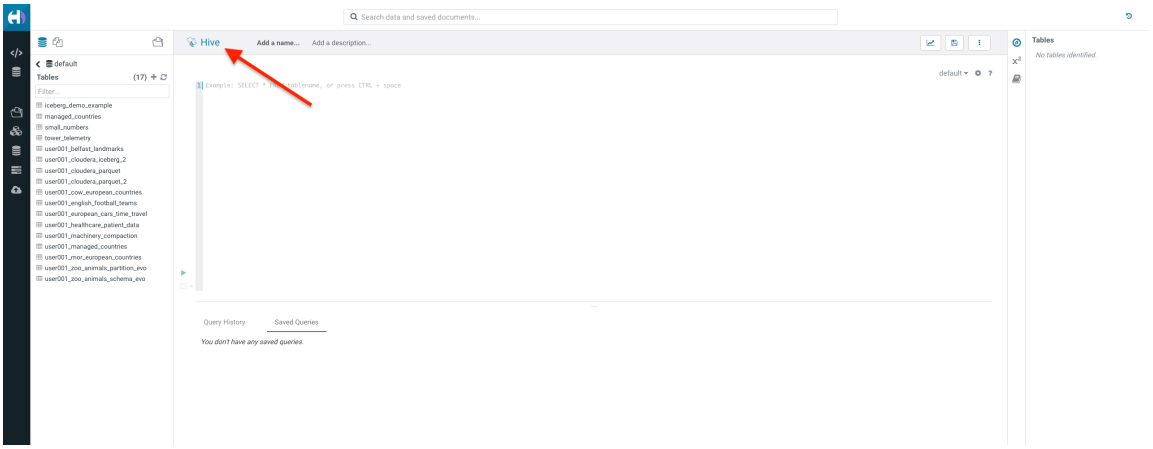
In the Cloudera Control Place select the Data Warehouse tile.



In the Data Warehouse Service locate the **Hive** Virtual Warehouse and click on the Hue button.

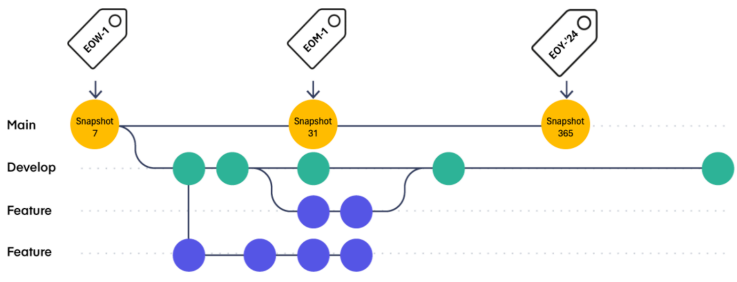


This will open the Hue IDE for Hive and you're ready to proceed.



8. Branching and Merging

Sophisticated snapshot lifecycle management



- Named references to snapshots with their own independent lifecycle.
- The life cycle is controlled by branch and tag level retention policies.
- GDPR, retaining important historical snapshots for auditing, testing / validating

Branching in Iceberg

Branching lets you create isolated environments to work with data (inserting, updating, deleting) without affecting the main production dataset. This is useful for testing new features, running experiments, or isolating changes before they are stable.

Why and when to use branches?

- Testing new features (e.g., testing new health metrics without affecting the existing production data).
- Running experiments or simulations.
- Isolating changes until they are confirmed to be stable and ready to be merged.
- Collaborating with different teams on separate data versions before consolidating the changes.

Example Branching:

HIVE

```
-- DROP THE TABLE IF IT EXISTS, SO WE CAN RERUN THE CODE
DROP TABLE IF EXISTS default.USERNAME_healthcare_patient_data;

-- CREATE AN ICEBERG TABLE FOR HEALTHCARE PATIENT DATA
CREATE TABLE default.USERNAME_healthcare_patient_data (
    patient_id STRING,
    patient_name STRING,
    age INT,
    diagnosis STRING,
    treatment STRING,
    doctor STRING
)
STORED BY ICEBERG;

-- INSERT INITIAL PATIENT DATA INTO THE BASE TABLE (CREATES A SNAPSHOT)
INSERT INTO default.USERNAME_healthcare_patient_data VALUES
    ('P001', 'John Doe', 45, 'Hypertension', 'Beta-blockers', 'Dr. Smith'),
    ('P002', 'Jane Roe', 50, 'Diabetes', 'Insulin', 'Dr. Johnson');

-- VERIFY THE EXISTING DATA
SELECT * FROM default.USERNAME_healthcare_patient_data;

-- CREATE A BRANCH
ALTER TABLE default.USERNAME_healthcare_patient_data CREATE BRANCH
testing_branch;

-- INSERT DATA INTO THE NEW BRANCH
INSERT INTO default.USERNAME_healthcare_patient_data.branch_testing_branch
VALUES
    ('P999', 'Richard V', 99, 'Headache', 'Time', 'Dr. Jeff');

-- VERIFY THE BRANCH DATA
SELECT * FROM default.USERNAME_healthcare_patient_data.branch_testing_branch;

-- MAIN TABLE REMAINS UNAFFECTED
SELECT * FROM default.USERNAME_healthcare_patient_data;
```

Merging Branches in Iceberg

Merging consolidates the changes made in a branch back into the main dataset. Only new or modified records are merged. This allows updates, experiments, or testing to be brought into the mainline data once they are confirmed to be stable or necessary.

How merging works in Iceberg

- You can merge changes from a branch back into the main dataset.
- The merge operation only applies changes from the branch and doesn't affect the main branch unless there are new or modified records that need to be merged.
- Iceberg provides methods to merge only certain changes from a branch or completely merge all data between branches.

Example Merging:

HIVE

```
-- VIEW THE MAIN TABLE DATA
SELECT * FROM default.USERNAME_healthcare_patient_data;

-- VIEW THE BRANCH TABLE DATA
SELECT * FROM default.USERNAME_healthcare_patient_data.branch_testing_branch;

-- MERGE THE BRANCH BACK INTO THE BASE TABLE
MERGE INTO default.USERNAME_healthcare_patient_data AS base
USING default.USERNAME_healthcare_patient_data.branch_testing_branch AS branch
ON base.patient_id = branch.patient_id
WHEN MATCHED THEN UPDATE SET patient_name = branch.patient_name,
                             age = branch.age,
                             diagnosis = branch.diagnosis,
                             treatment = branch.treatment,
                             doctor = branch.doctor
WHEN NOT MATCHED THEN INSERT (patient_id, patient_name, age, diagnosis,
treatment, doctor)
VALUES (branch.patient_id, branch.patient_name, branch.age, branch.diagnosis,
branch.treatment, branch.doctor);

-- VIEW THE MERGED TABLE DATA
SELECT * FROM default.USERNAME_healthcare_patient_data;

-- DROP THE BRANCH AFTER THE MERGE IF NO LONGER NEEDED
ALTER TABLE default.USERNAME_healthcare_patient_data DROP BRANCH
testing_branch;
```

9. Tagging (Versioning)

Tags label specific table versions (snapshots), making it easier to reference or roll back to that particular point in time. Tags simplify accessing a specific version of data, replacing the need to know the snapshot ID.

How to Use Tags:

- Versioning: Tags allow you to mark versions of data with meaningful names like v1.0, snapshot_2025_01_01, or test_run.
- Metadata Management: You can attach custom metadata to snapshots, such as the name of the person who performed a change or the reason for a particular change.
- Tags can be helpful when needing to easily access a specific version or snapshot of data without remembering the snapshot ID.

Example Tagging and Querying:

HIVE

```
-- DROP THE LANDMARKS TABLE IF IT EXISTS
DROP TABLE IF EXISTS default.USERNAME_belfast_landmarks;

-- CREATE ICEBERG TABLE TO STORE LANDMARK DATA
CREATE TABLE IF NOT EXISTS default.USERNAME_belfast_landmarks (
    landmark_id STRING,
    landmark_name STRING,
    location STRING,
    description STRING
)
STORED BY ICEBERG;

-- INSERT SAMPLE DATA INTO THE LANDMARKS TABLE
INSERT INTO default.USERNAME_belfast_landmarks VALUES
    ('L001', 'Titanic Belfast', 'Belfast', 'Interactive museum about the RMS
Titanic'),
    ('L002', 'Belfast Castle', 'Belfast', '19th-century castle offering views
of the city and hills');

-- CREATE TAG FOR THE LANDMARKS TABLE WITH RETENTION PERIOD
ALTER TABLE default.USERNAME_belfast_landmarks CREATE TAG MY_TAG RETAIN 5 DAYS;

-- INSERT ADDITIONAL SAMPLE DATA INTO THE LANDMARKS TABLE
INSERT INTO default.USERNAME_belfast_landmarks VALUES
    ('L003', 'Queen's University Belfast', 'Belfast', 'A prestigious university
known for its research and history'),
    ('L004', 'Stormont', 'Belfast', 'Northern Ireland's parliament buildings
and grounds'),
    ('L005', 'Crumlin Road Gaol', 'Belfast', 'Historic prison turned into a
museum'),
    ('L006', 'Botanic Gardens', 'Belfast', 'Public gardens featuring the Palm
House and Tropical Ravine'),
    ('L007', 'Ulster Museum', 'Belfast', 'A museum housing art, history, and
natural sciences exhibits'),
    ('L008', 'Titanic Dry Dock', 'Belfast', 'The historic dock where the RMS
Titanic was fitted out');


-- QUERY AND DISPLAY THE DATA INSERTED INTO THE LANDMARKS TABLE
SELECT * FROM default.USERNAME_belfast_landmarks;

-- QUERY AND DISPLAY REFERENCE DATA (TAGS) FOR THE LANDMARKS TABLE
SELECT * FROM default.USERNAME_belfast_landmarks.refs;

-- QUERY THE TABLE USING THE CREATED TAG TO SEE THE SNAPSHOT AT THAT SPECIFIC
POINT IN TIME
SELECT * FROM default.USERNAME_belfast_landmarks.tag_MY_TAG;
```

10. Understanding Iceberg Storage

Iceberg Table Definition and Metadata

The `SHOW CREATE TABLE` command confirms the Iceberg table's definition. Checking the HDFS location reveals that Iceberg manages both data (`data/`) and metadata (`metadata/`) directories within the table's storage path.

- `metadata/` contains snapshots, schema history, and manifest files.
- `data/` contains the actual table data files.

Iceberg uses the `metadata/` directory to manage partitioning and versioning, without relying on Hive Metastore. The ** `data/` ** directory contains the actual table data files.

Code Example

IMPALA

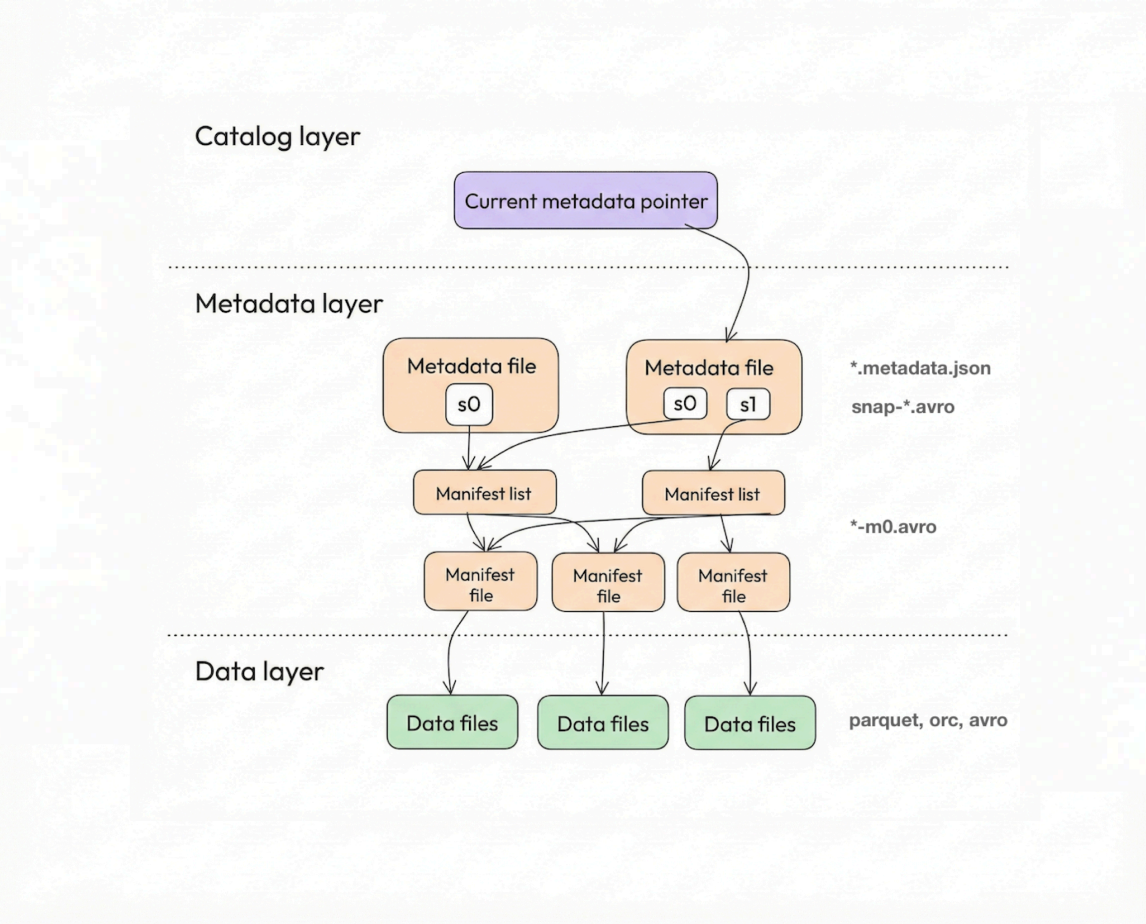
```
-- Get the Iceberg table definition
SHOW CREATE TABLE default.USERNAME_managed_countries;
```

BASH

```
# Check the Object Store directory for Iceberg metadata and data
[jturkington@XYZ-es01 ~]$ hdfs dfs -ls
s3a://.../external/hive/default.db/managed_countries/metadata
-rw-r--r--   3 jturkington hive       1710 2025-02-05 12:55
hdfs://.../external/hive/default.db/managed_countries/metadata/00000-bc161db1-
05f2-4d64-baab-69ca2070db33.metadata.json
-rw-r--r--   3 jturkington hive       6072 2025-02-05 04:05
hdfs://.../external/hive/default.db/managed_countries/metadata/3ecfea4f-9e06-
45a9-bd7c-430fe4758283-m0.avro
-rw-r--r--   3 jturkington hive       3800 2025-02-05 12:55
hdfs://.../external/hive/default.db/managed_countries/metadata/snap-
1185275548636187694-1-f7f549e1-bd07-44da-b170-8973c2e6e3d6.avro
```

Understanding the Metadata Files

Iceberg uses several types of metadata files to track table state and manage its partitions. Below are the types of metadata files found in the `metadata/` directory.



Metadata JSON Files (*.metadata.json)

Example Files: 00000-bc161db1-05f2-4d64-baab-69ca2070db33.metadata.json

Purpose: Stores table-level metadata such as schema, partitioning, snapshots, and file references. Each time the table structure changes (e.g., schema evolution, snapshot creation), a new metadata JSON file is generated. Older metadata files are retained to support time travel and rollback.
Data Type: JSON format (human-readable, structured key-value pairs).
Why? JSON allows Iceberg to store metadata in a flexible, easily accessible format. New versions can be created without modifying existing files, enabling schema evolution.

Snapshot Files (snap-*.avro)

Example Files: snap-1185275548636187694-1-f7f549e1-bd07-44da-b170-8973c2e6e3d6.avro

Purpose: Tracks table state at a specific point in time (snapshot ID, timestamp, manifest list, etc.). Allows for time travel and rollbacks to previous versions of the table.
Data Type: Apache Avro format (binary, optimized for structured data storage).
Why? Storing snapshots in Avro provides efficient serialization while keeping metadata compact and performant. Enables fast lookup of previous states for Iceberg's time travel feature.

Manifest List Files (*-m0.avro)

Example Files: 3ecfea4f-9e06-45a9-bd7c-430fe4758283-m0.avro

Purpose: Stores a list of manifest files associated with a snapshot. Helps Iceberg quickly determine which data files belong to which snapshot without scanning the entire table.
Data Type: Apache Avro format (binary, optimized for fast read/write).
Why? Avro is compact and supports schema evolution, making it ideal for metadata storage. Using Avro instead of JSON for large metadata speeds up querying and file tracking.

How These Files Work Together in Iceberg

Metadata JSON file (*.metadata.json) defines the table schema and references snapshots.

Snapshot file (snap-*.avro) records changes and links to manifest lists.

Manifest list file (*-m0.avro) references manifest files that contain details of individual data files.

These components work together to support partitioning, versioning, and time travel, allowing Iceberg to provide robust table management with features like schema evolution and data consistency.