CLOUDERA

DATA WAREHOUSE HANDS-ON

STUDENT GUIDE

CLOUDERA DATA WAREHOUSE WORKSHOP

Preface

Working for an Aircraft Engine company, the company wants to increase competitive advantage in two keyways:

- Engineer better, more fault tolerant aircraft engines.
- Be proactive in predictive maintenance on engines, and faster discovery-to-fix in new engine designs.

This will be a three-phase plan:

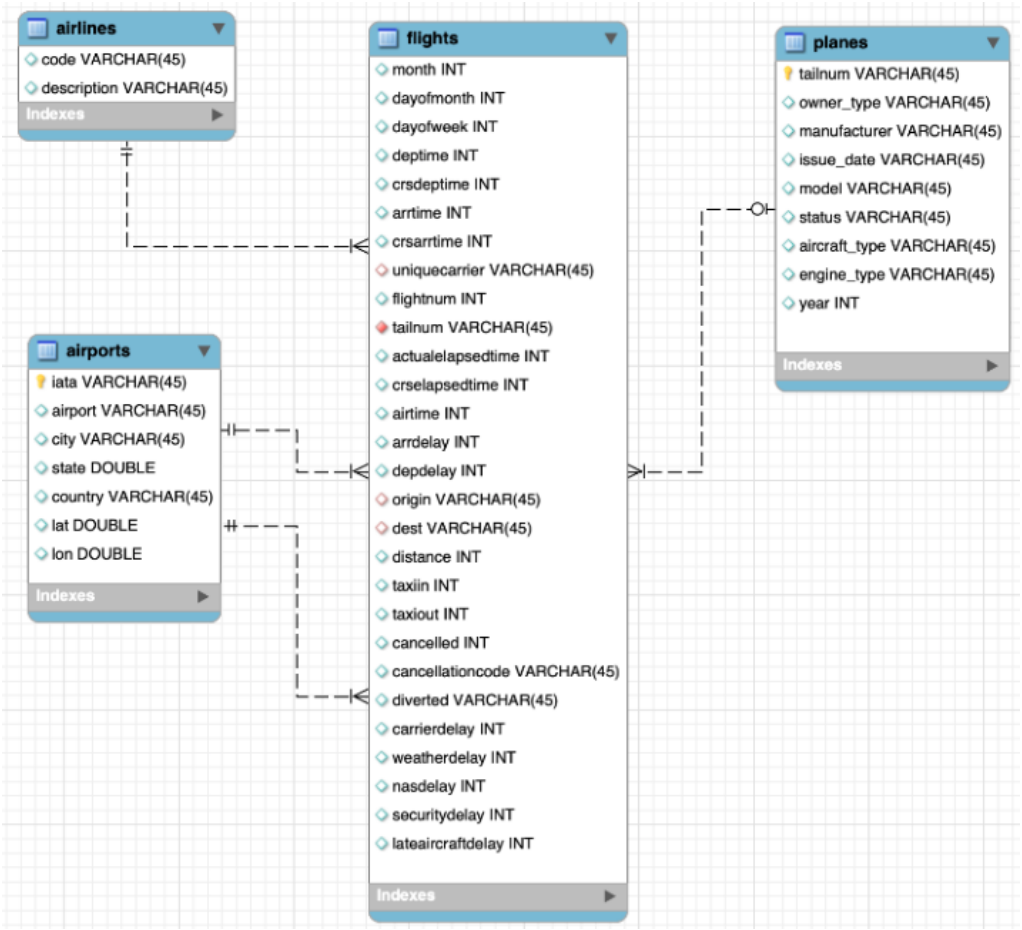
- Phase One: Understand how our current engines contribute to airline flight delays and fix for future engines.
- Phase Two: Implement an ongoing reporting service to support ongoing engineering efforts to continuously improve engines based on delay data.
- Phase Three: Move to real-time analysis to fix things before they break both in engines already sold, and in new engine designs.

To do this, we're going to build a data warehouse & data lakehouse to create reports that engineers can use to improve our engines. The following people will get to work:

We will dive into this Scenario to show Cloudera Data Warehouse (CDW) is used to enable the Aircraft company to gain competitive advantage - and at the same time it highlights the performance and automation capabilities that help ensure performance is maintained while controlling costs.

The Hands on Labs will take you through how to use the Cloudera Data Warehouse service to quickly explore raw data, create curated versions of the data for simple reporting and dashboarding, and then scale up usage of the curated data by exposing it to more users.

ER - Diagram of the data
Fact Table: flights (86M rows)
Dimension Table: airlines (1.5k rows), airports (3.3k rows) and planes (5k rows)



Pre-requisites

- Laptop with a supported OS (Windows 7 not supported) or MacBook.
- A modern browser - Google Chrome (IE, Firefox, Safari not supported).

Workshop Overview

Below are the high-level steps for what we will be doing in the workshop.

- **[Lab 1 & 2]: General introduction to CDW** to get ourselves oriented for the workshop.
    - **As an Admin:** Create and enable the BI analyst team with a Virtual Warehouse.
    - **As a BI Analyst**: Get familiar with CDW on CDP and set up our first VW to start working.
    - **As a BI Analyst:** Wrangle our first set of data - sent to us as a series of .csv files exported from “somewhere else”.
    - **As an Admin:** Monitor the VW and watch as it scales up and down, suspends, etc.
    - **As a BI Analyst:** Start digging into the data - looking for “needle in a haystack” - running a complex query that will find which engines seem to be correlated to airplane delays for any reason.
- **[Lab 3]: Set it up.**
    - **As an Admin:** Create and enable the BI analyst team with a Virtual Warehouse.
    - **As a BI Analyst:** Get familiar with CDW on CDP, and set up our first VW to start working.
    - **As a BI Analyst:** Wrangle our first set of data - sent to us as a series of .csv files exported from “somewhere else”.
    - **As an Admin:** Monitor the VW and watch as it scales up and down, suspends, etc.
    - **As a BI Analyst:** Start digging into the data - looking for “needle in a haystack” - running a complex query that will find which engines seem to be correlated to airplane delays for any reason.
- **[Lab 4]: Making it better.**
    - **As a BI Analyst:** Start curating data and building a data lakehouse to improve quality by tweaking data, performance by optimizing schema structures, and ensure reliability and trustworthiness of the data through snapshots, time travel, and rollback.
    - Create Hive ACID tables and tweak data for consistency (ex: airline name changes - ensure reporting is consistent with the new name to avoid end user confusion, a new airline joins our customer list, make sure they’re tracked for future data collection, etc..).
    - Migrate Tables to Iceberg (We want snapshot and rollback).
    - Create new Iceberg tables (we want partitioning).
- **[Lab 5]: Optimizing for production.**
    - Loading more data - change partitioning to maintain performance (NOTE: Ongoing ELT = CDE?).
    - Bad data is loaded - use time travel to detect, and rollback to resolve.
    - Introduce materialized views to support scaling to 1000’s of simultaneous users.
    - As an admin: Monitor, report, kill queries that run amock, etc.
- **[Lab 6]: Security & Governance.**
    - Check on the lineage to enable governance/audit.

    - Row level security to make sure only relevant party can see data.
- **[Lab 7]: Cloudera Data Visualization**
    - Data Modeling for the lakehouse.
    - Data Visualization for insights.

LAB 1 - Getting Started

VERIFY RESOURCES

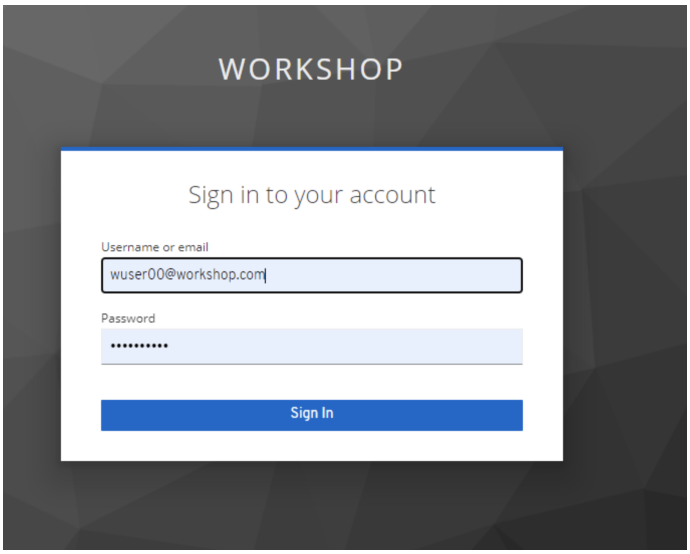
For this workshop you will use the following credentials and resources

Workshop Login Username	<Check with your instructor>
Workshop Login Password	<Check with your instructor>
CDP Workload User (${user_id})	<Check with your instructor>
CDP Workload Password	<Check with your instructor>
Hive Virtual Warehouse Name	<Check with your instructor>
Impala Virtual Warehouse Name	<Check with your instructor>

DEFINE CDP WORKLOAD PASSWORD

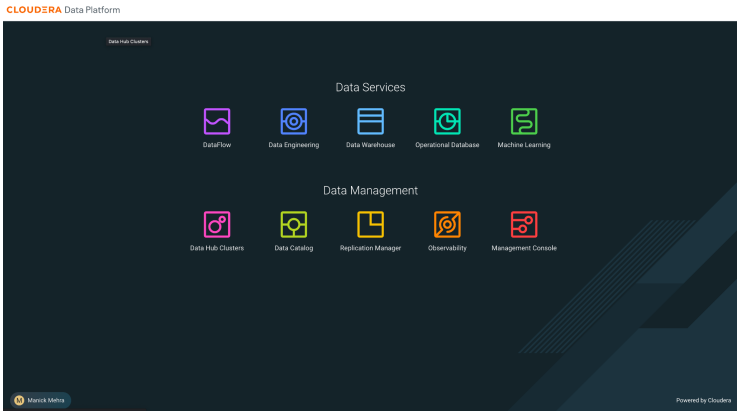
Step 1 : Login

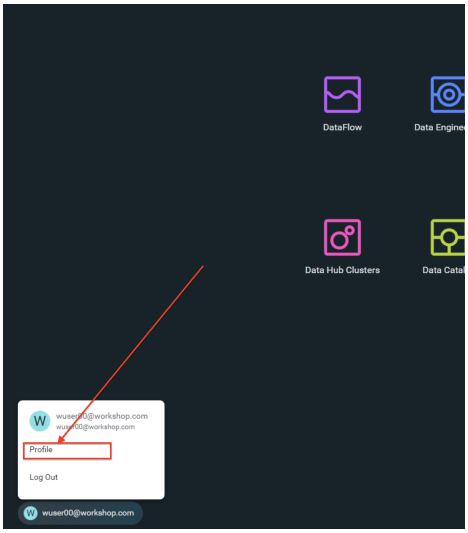
Login to the environment using the URL and credentials provided.. The login page will look like this.



Step 2 : Go to Profile

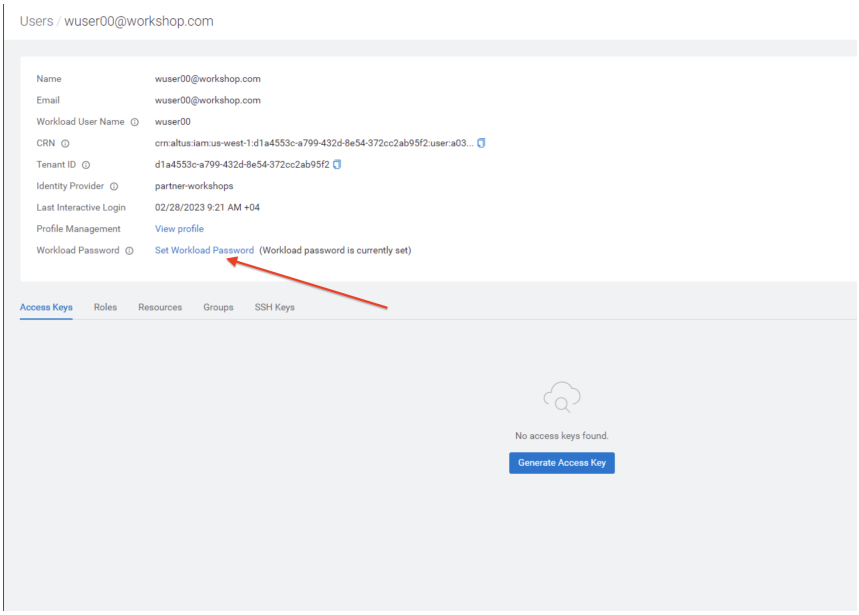
Once you log in successfully you will be able to see the CDP HomePage with all the data services and the management services. Click on the username at the bottom left of the screen and select **PROFILE**



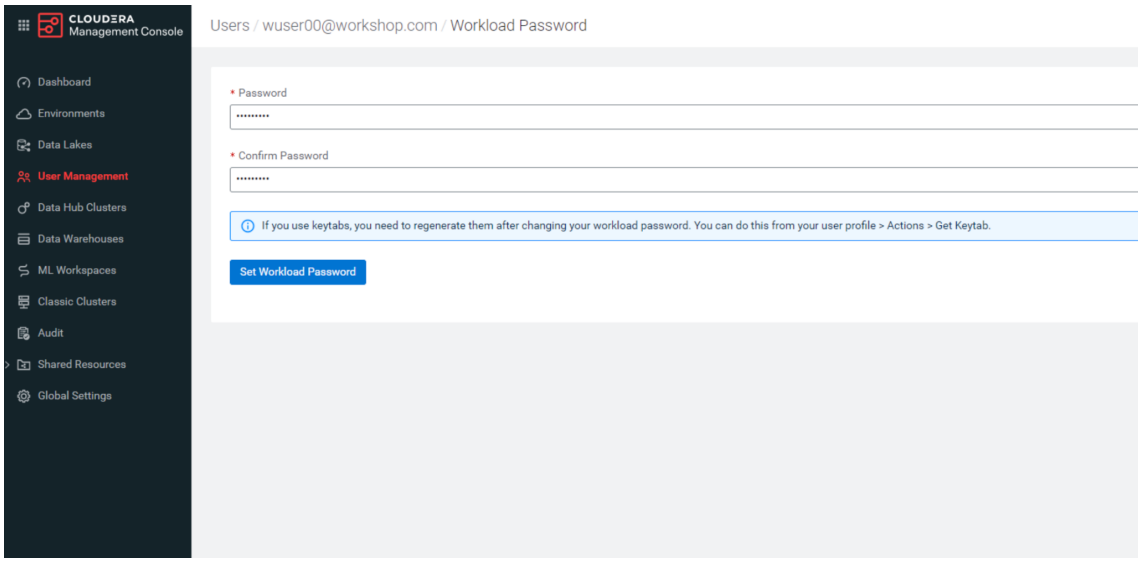


Step 3 : Set Workload Password

Click option Set Workload Password.



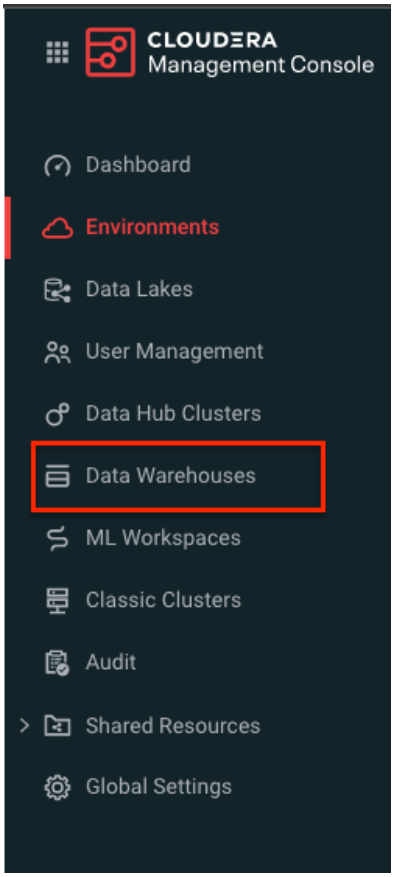
Enter the Password shared by the instructor and Confirm Password.



Click the button **Set Workload Password.**

LAB 2 - RAW LAYER

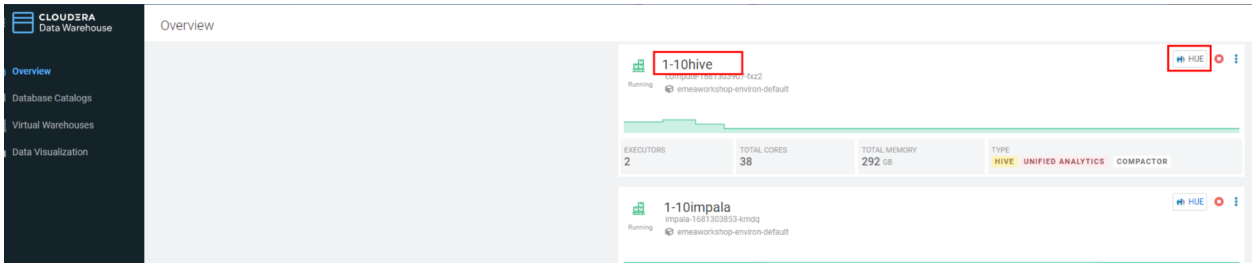
The objective of this step is to create External tables on top of raw CSV files sitting in cloud storage (In this case it has been stored in AWS S3 by the instructor) and then run a few queries to access the data via SQL using HUE.
To access Data Warehouse data service click on Data Warehouse on the left.



DATABASE AND TABLE CREATION

Step 1 : Go To Hue [Hive Cluster]

Hue is associated with each of the virtual clusters that are present under the Database Catalog. In the virtual cluster that has been assigned to you select HUE from the top right corner of the virtual cluster.



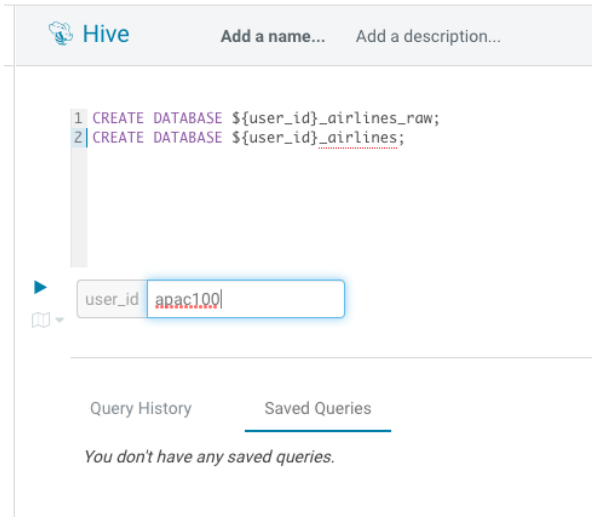
Step 2 : CREATE DATABASE

Create new databases. Enter the following query and then make sure that you enter the user assigned to you as a prefix(replace `${user_id}`) to the database name.

```
Unset
CREATE DATABASE ${user_id}_airlines_raw;
CREATE DATABASE ${user_id}_airlines;
```

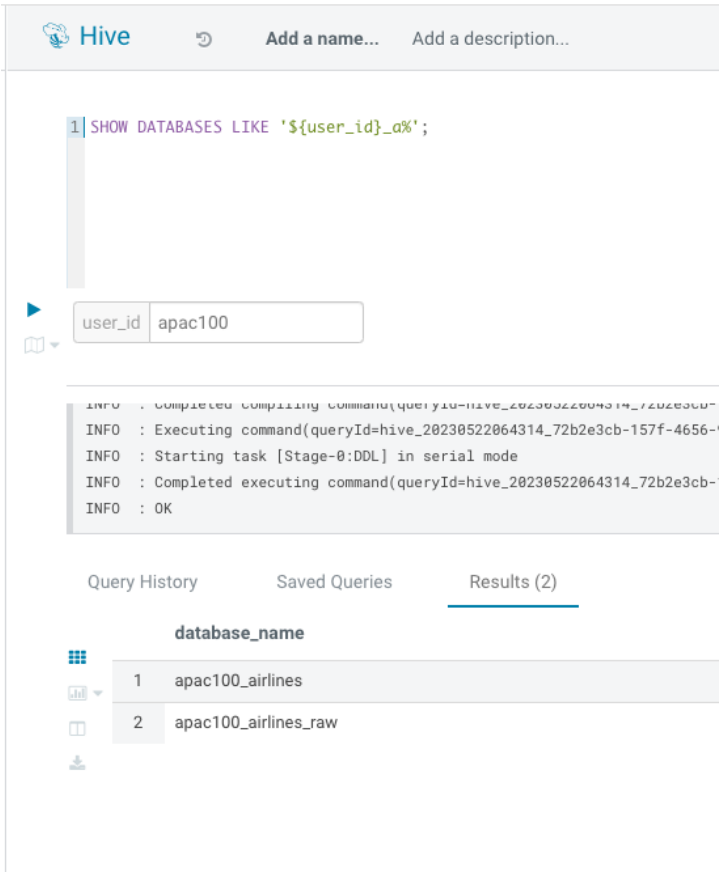
Example Query

```
Unset
CREATE DATABASE apac01_airlines_raw;
CREATE DATABASE apac01_airlines;
```



Verify if the database is created using the following query. Do not forget to replace the ${user-id} with your actual username

```
Unset
SHOW DATABASES LIKE '${user_id}%';
```



Step 3 : CREATE EXTERNAL TABLES

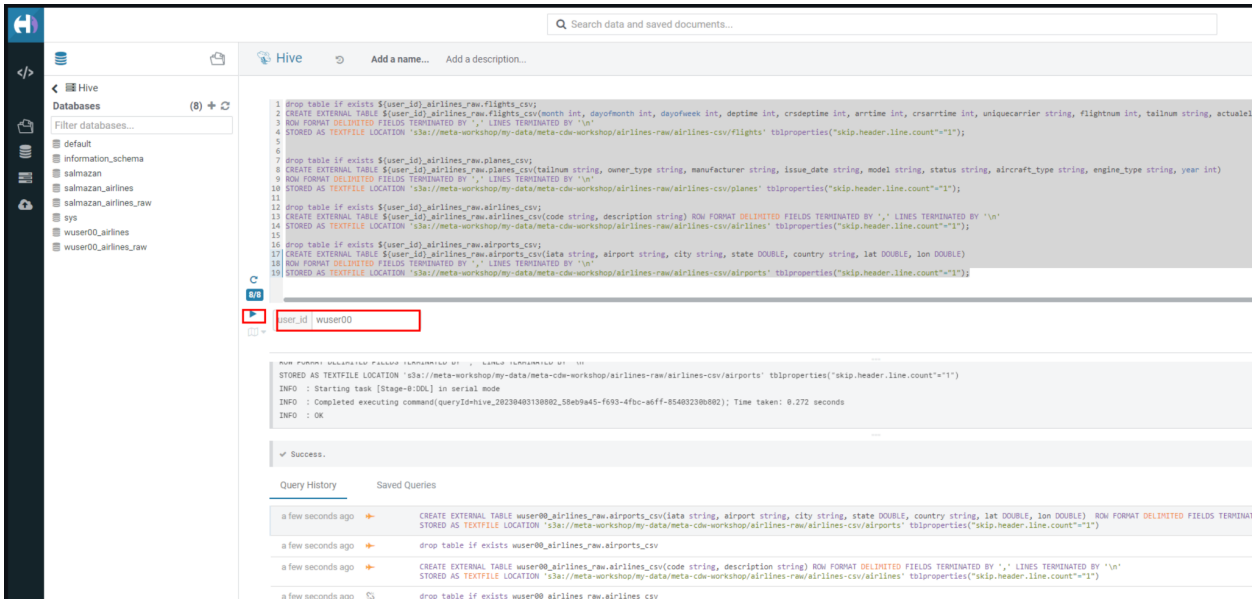
Run the following DDL in the editor.

This will create External Tables on CSV Data Files that have been uploaded previously by your instructor in AWS S3. This provides a fast way to allow SQL layers on top of data in cloud storage.

NOTE : Replace ${user_id} with the user name assigned to you

Unset

```
--FLIGHTS TABLE
drop table if exists ${user_id}_airlines_raw.flights_csv;
CREATE EXTERNAL TABLE ${user_id}_airlines_raw.flights_csv(month int, dayofmonth int,
dayofweek int, deptime int, crsdeptime int, arrtime int, crsarrtime int, uniquecarrier
string, flightnum int, tailnum string, actualelapsedtime int, crselapsedtime int, airtime
int, arrdelay int, depdelay int, origin string, dest string, distance int, taxiin int,
taxiout int, cancelled int, cancellationcode string, diverted string, carrierdelay int,
weatherdelay int, nasdelay int, securitydelay int, lateaircraftdelay int, year int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION
's3a://handsonworkshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/flights'
tblproperties("skip.header.line.count"="1");

--PLANES TABLE
drop table if exists ${user_id}_airlines_raw.planes_csv;
CREATE EXTERNAL TABLE ${user_id}_airlines_raw.planes_csv(tailnum string, owner_type
string, manufacturer string, issue_date string, model string, status string, aircraft_type
string, engine_type string, year int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION
's3a://handsonworkshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/planes'
tblproperties("skip.header.line.count"="1");


--AIRLINES TABLE
drop table if exists ${user_id}_airlines_raw.airlines_csv;
CREATE EXTERNAL TABLE ${user_id}_airlines_raw.airlines_csv(code string, description
string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION
's3a://handsonworkshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/airlines'
tblproperties("skip.header.line.count"="1");

--AIRPORT TABLE
drop table if exists ${user_id}_airlines_raw.airports_csv;
CREATE EXTERNAL TABLE ${user_id}_airlines_raw.airports_csv(iata string, airport string,
city string, state DOUBLE, country string, lat DOUBLE, lon DOUBLE)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS TEXTFILE LOCATION
's3a://handsonworkshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/airports'
tblproperties("skip.header.line.count"="1");
```

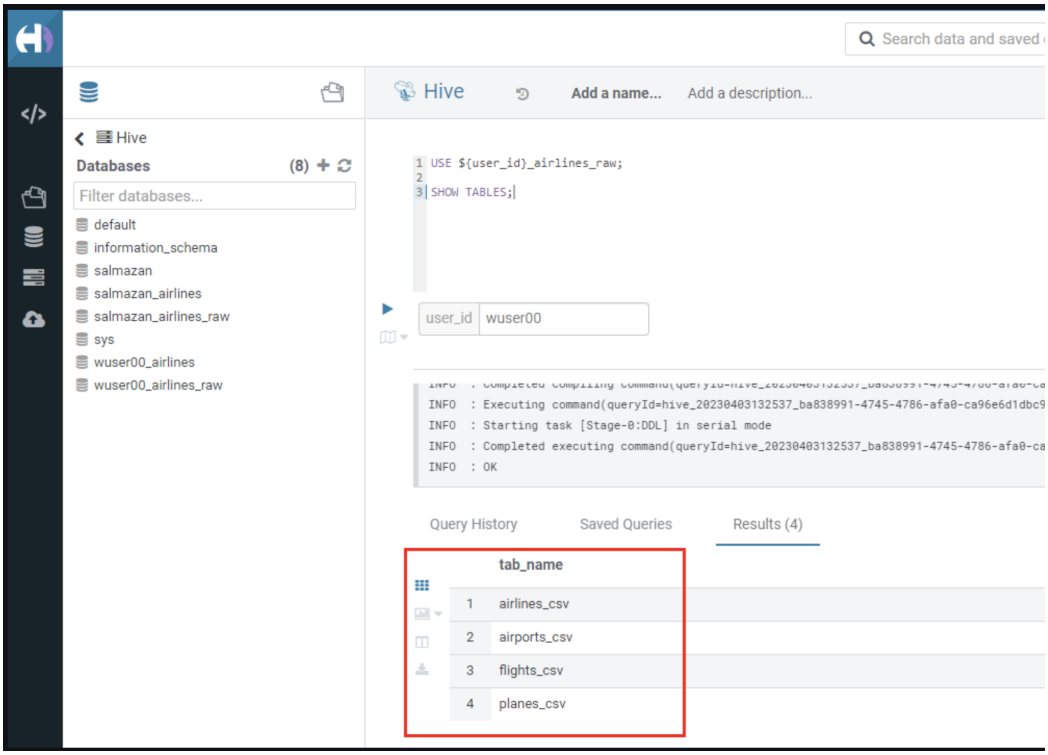


Step 4 : VERIFY IF THE TABLES ARE CREATED

Run the following queries in the editor to verify if the tables are created correctly.

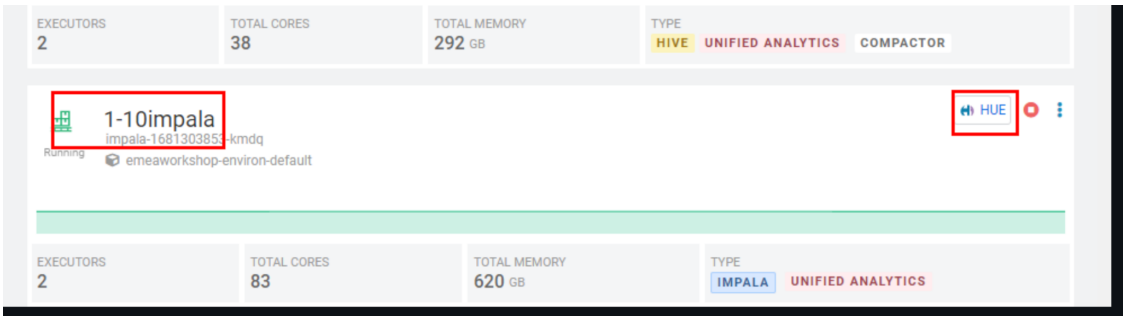
```
Unset
USE ${user_id}_airlines_raw;
SHOW TABLES;
```

Make sure that 4 tables (airlines_csv, airports_csv, flights_csv, planes_csv) are created as shown below.

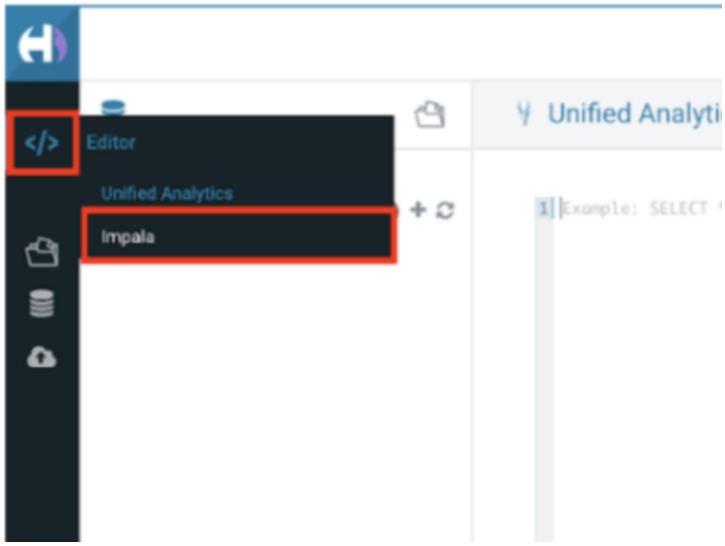


Step 5 : QUERY CREATED TABLES USING IMPALA VIRTUAL WAREHOUSE

Go to the page where now you will access HUE of an Impala virtual warehouse assigned to you. Click on HUE for impala as shown in the screenshot below.

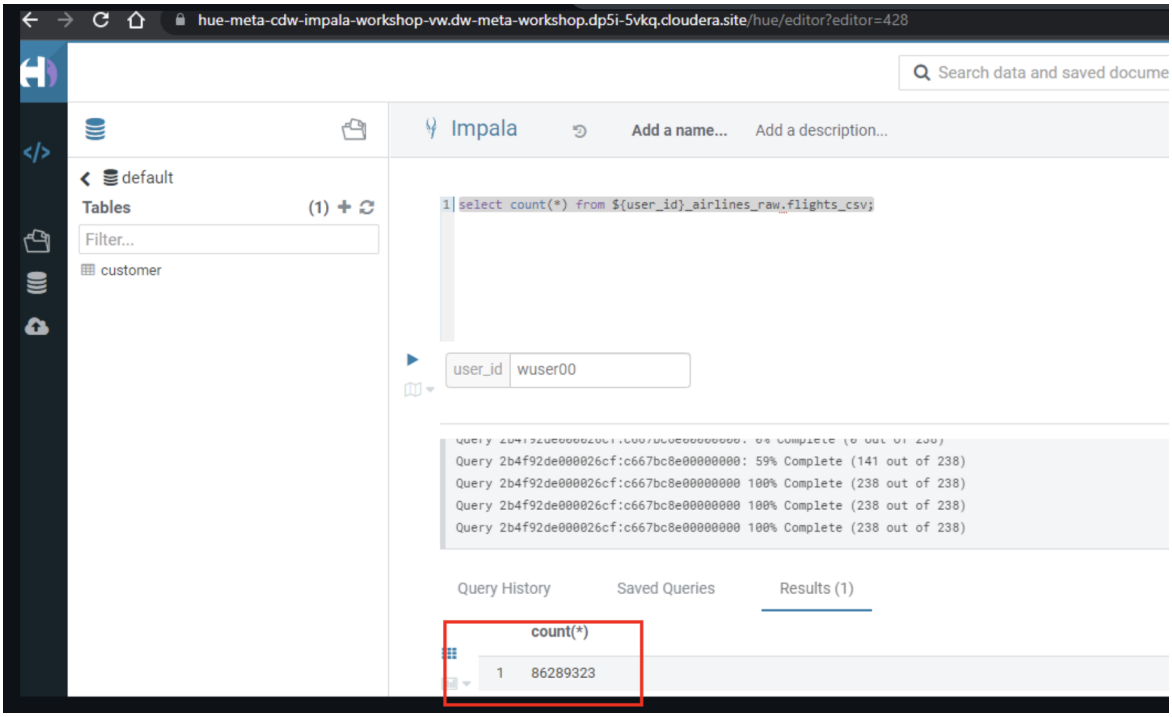


Make sure that you click to get Impala instead of default in the HUE browser as shown below and then click the refresh button.



Now, copy paste the following in the HUE editor and click on Run as shown below.

```
Unset
select count(*) from ${user_id}_airlines_raw.flights_csv;
```

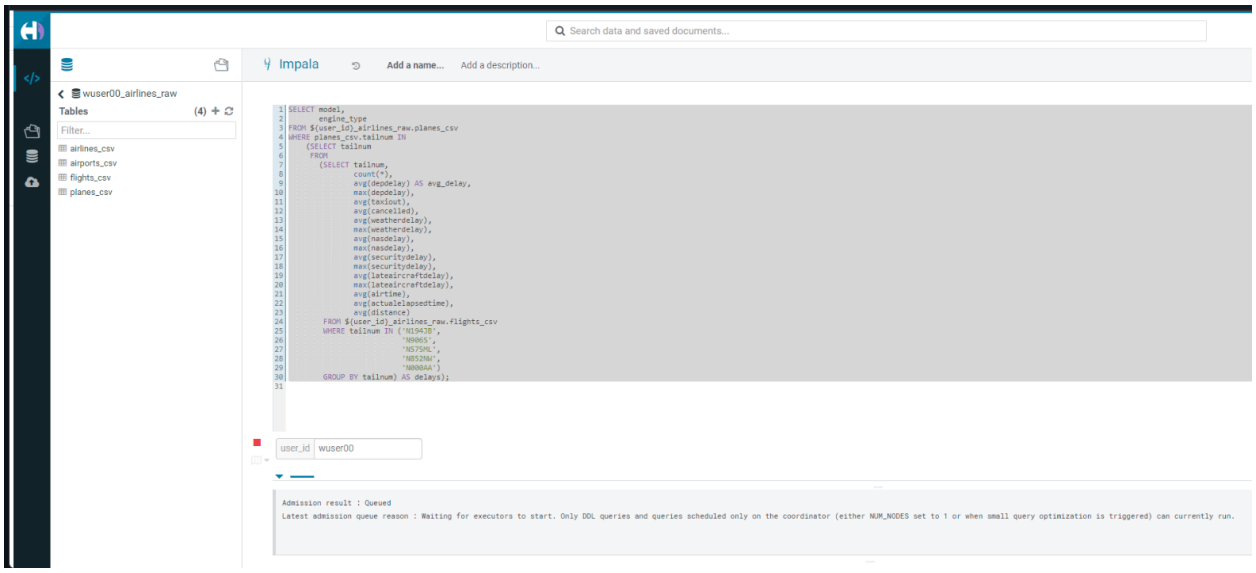


Over the course of this workshop we will execute many queries which are compute heavy and this will make the virtual warehouse that is assigned to you to auto scale up and down based on the workload.
We will now run a compute heavy query and check how that affects the scaling up/down of the virtual warehouse. This autoscaling can be observed from the graph that shows each of the virtual warehouses as shown in the image below. Look specifically for the warehouse you are using.

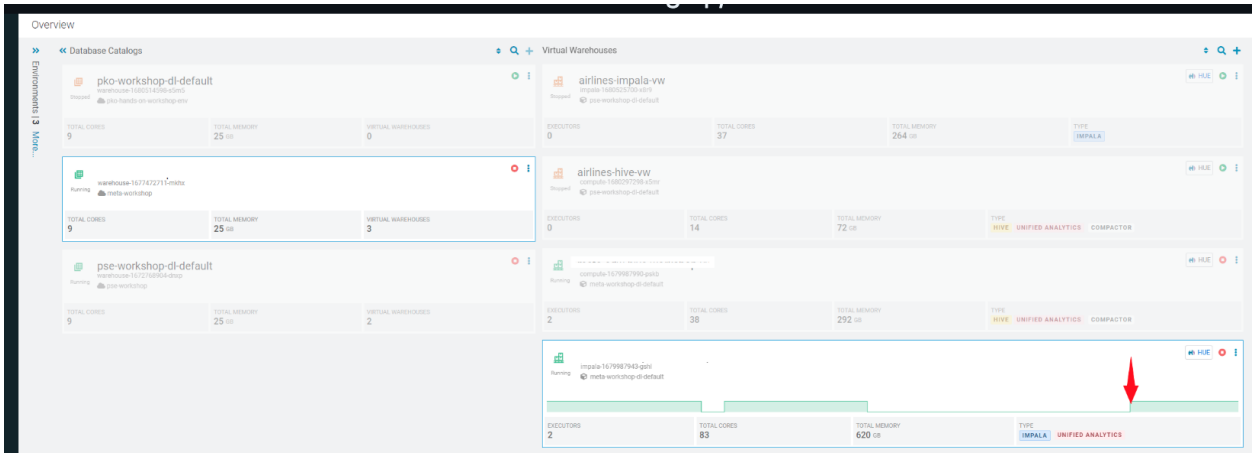
Query:
DO NOT forget to change the ${user_id}

```
Unset
SELECT model,
       engine_type
FROM ${user_id}_airlines_raw.planes_csv
WHERE planes_csv.tailnum IN
    (SELECT tailnum
     FROM
       (SELECT tailnum,
               count(*),
               avg(depdelay) AS avg_delay,
               max(depdelay),
               avg(taxiout),
               avg(cancelled),
               avg(weatherdelay),
               max(weatherdelay),
               avg(nasdelay),
               max(nasdelay),
               avg(securitydelay),
               max(securitydelay),
               avg(lateaircraftdelay),
               max(lateaircraftdelay),
               avg(airtime),
               avg(actualelapsedtime),
               avg(distance)
        FROM ${user_id}_airlines_raw.flights_csv
        WHERE tailnum IN ('N194JB',
                          'N906S',
                          'N575ML',
                          'N852NW',
                          'N000AA')
```

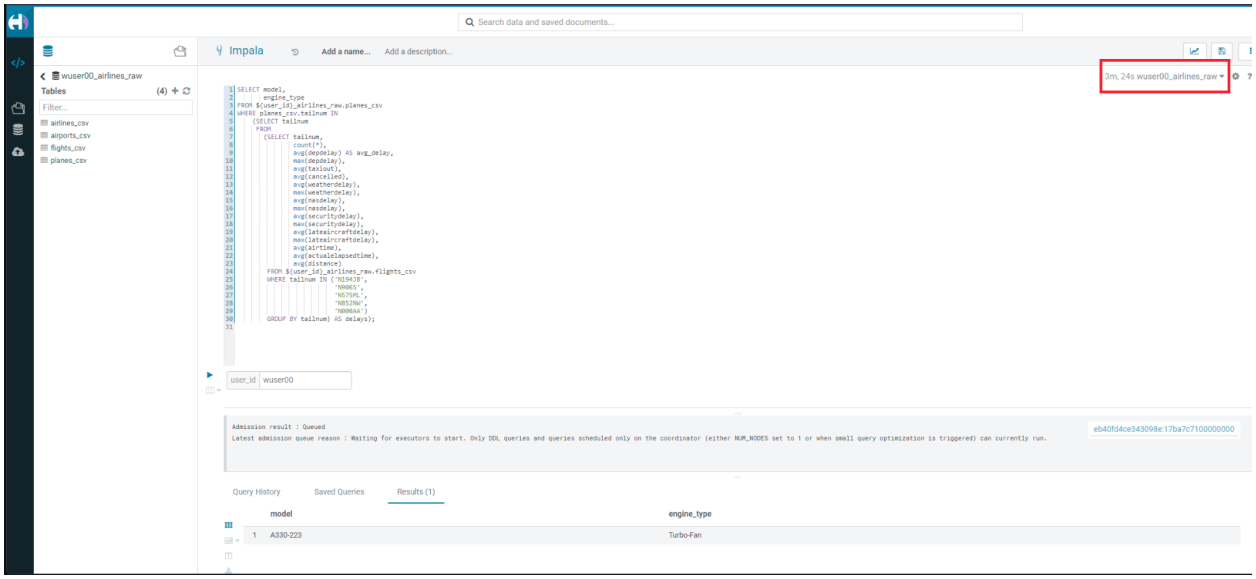
```
        GROUP BY tailnum) AS delays);
```



Post execution, look at the graph on your virtual warehouse.

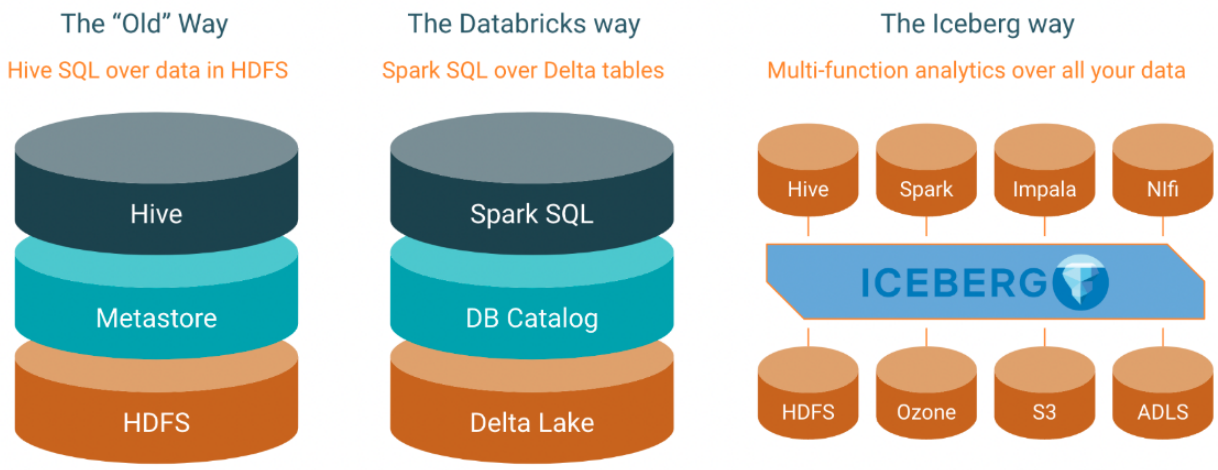


Go back to the editor and observe the amount of time the query has taken to complete.


Search data and saved documents...
Impala
Add a name...
Add a description...
wuser00_airlines_raw
Tables (4)
Filter...
airlines_csv
airports_csv
flights_csv
planes_csv
3m, 24s wuser00_airlines_raw
SELECT model,
engine_type
FROM ${user_id}_airlines_raw.planes_csv
WHERE planes_csv.tailnum IN
(SELECT tailnum
FROM
(SELECT tailnum,
count(*),
avg(depdelay) AS avg_delay,
max(depdelay),
avg(taxiout),
avg(cancelled),
avg(weatherdelay),
max(weatherdelay),
avg(nasdelay),
max(nasdelay),
avg(securitydelay),
max(securitydelay),
avg(lateaircraftdelay),
max(lateaircraftdelay),
avg(airtime),
avg(actualelapsedtime),
avg(distance)
FROM ${user_id}_airlines_raw.flights_csv
WHERE tailnum IN ('N194JB',
'N906CS',
'N575ML',
'N852NW',
'N800AA')
GROUP BY tailnum) AS delays);
user_id
wuser00
Admission result : Queued
Latest admission queue reason : Waiting for executors to start. Only DDL queries and queries scheduled only on the coordinator (either NUM_NODES set to 1 or when small query optimization is triggered) can currently run.
eb40fd4ce343098e:17ba7c7100000000
Query History
Saved Queries
Results (1)
model
engine_type
1
A330-223
Turbo-Fan

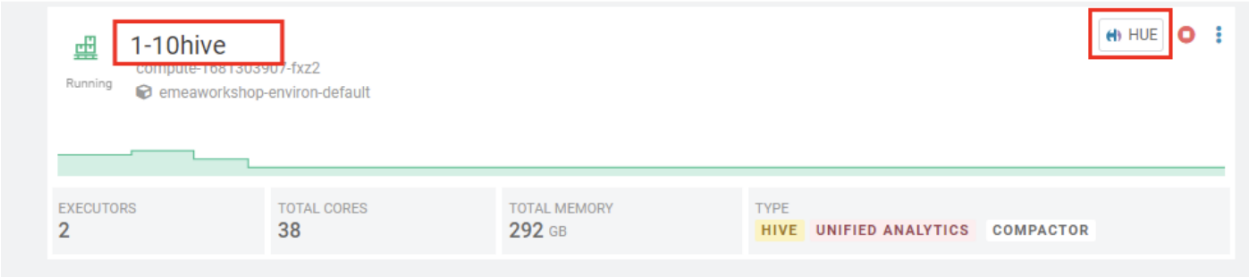
LAB 3 - DATA LAKEHOUSE

In this Lab we will take steps to make use of Hive and Iceberg Table formats to provide us with best of both world scenarios in our Data Lakehouse.



CURATED LAYER CREATION

NOTE : Make sure that you are using the HUE of the HIVE virtual warehouse that has been assigned to you.



Step 1 : CREATE and LOAD PLANES TABLE

- From the data that is stored in the RAW layer(CSV format) we will now create a table using that data
  Create planes table in Hive table format and stored its data in parquet file format.

```
Unset
drop table if exists ${user_id}_airlines.planes;
```

```
CREATE EXTERNAL TABLE ${user_id}_airlines.planes (
  tailnum STRING, owner_type STRING, manufacturer STRING,
issue_date STRING,
  model STRING, status STRING, aircraft_type STRING,  engine_type
STRING, year INT
)
STORED AS PARQUET
TBLPROPERTIES ('external.table.purge'='true');
```

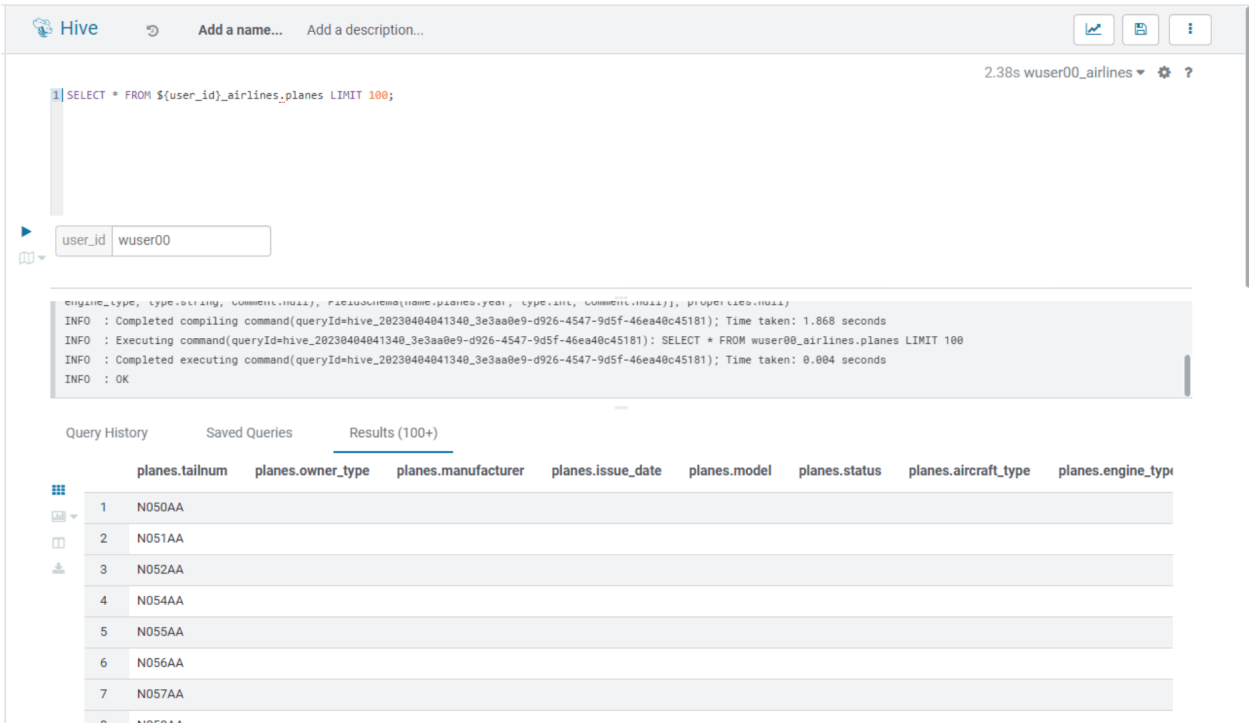
- Load planes table with data from the Raw layer table planes_csv.

Unset

```
INSERT INTO ${user_id}_airlines.planes
  SELECT * FROM ${user_id}_airlines_raw.planes_csv;
```

- Once the data is loaded successfully, run the below query to verify if the table now contains data.

```
Unset
SELECT * FROM ${user_id}_airlines.planes LIMIT 100;
```



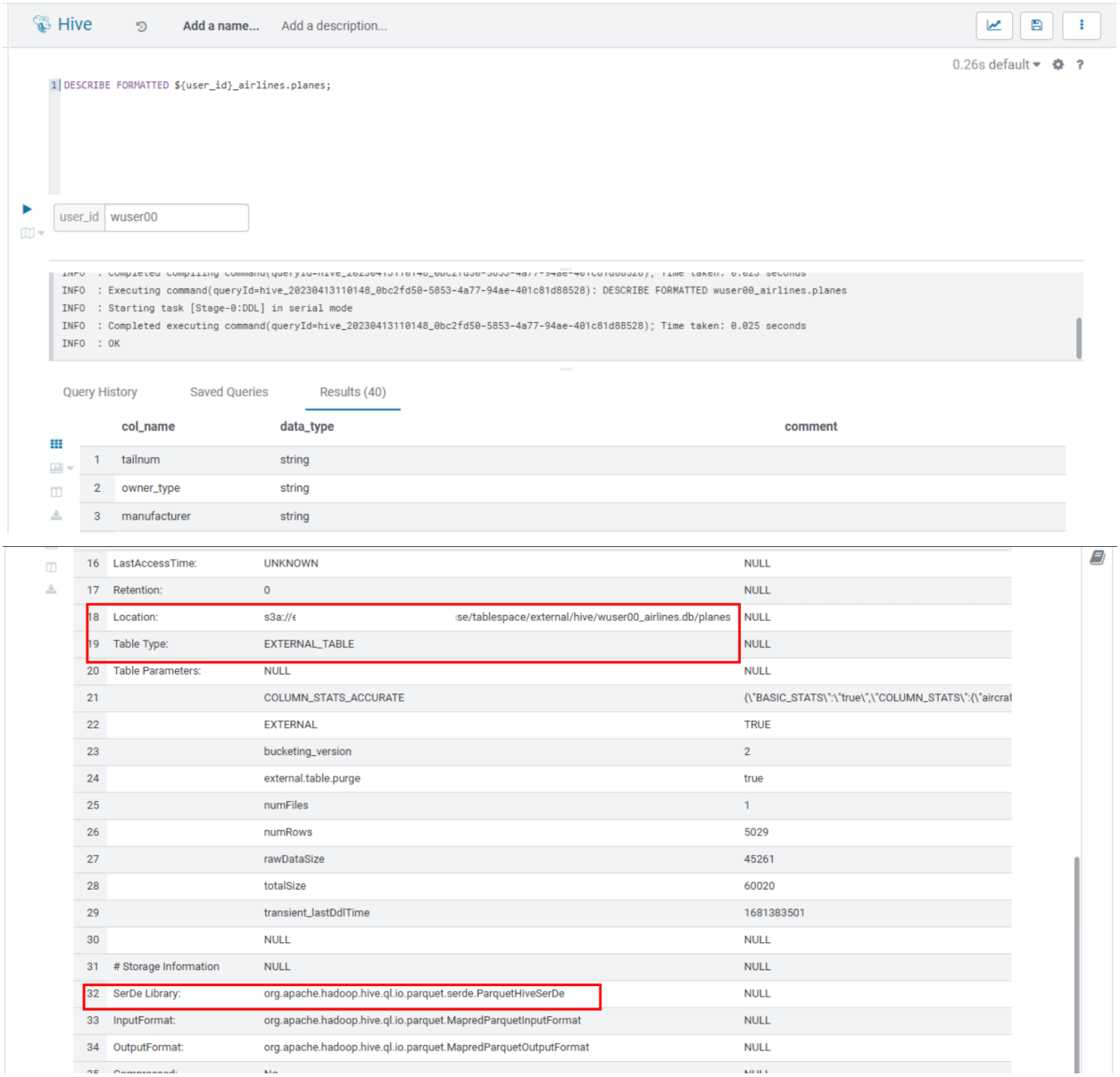
Step 2 : DESCRIBE CREATED TABLE

- Execute the following command.

```
Unset
DESCRIBE FORMATTED ${user_id}_airlines.planes;
```

In the output look for the following.

- **Location:** s3a://handsonworkshop/my-data/warehouse/tablespace/external/hive/wuser00_airlines.db/planes
- **Table Type:** EXTERNAL_TABLE
- **SerDe Library:** org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe



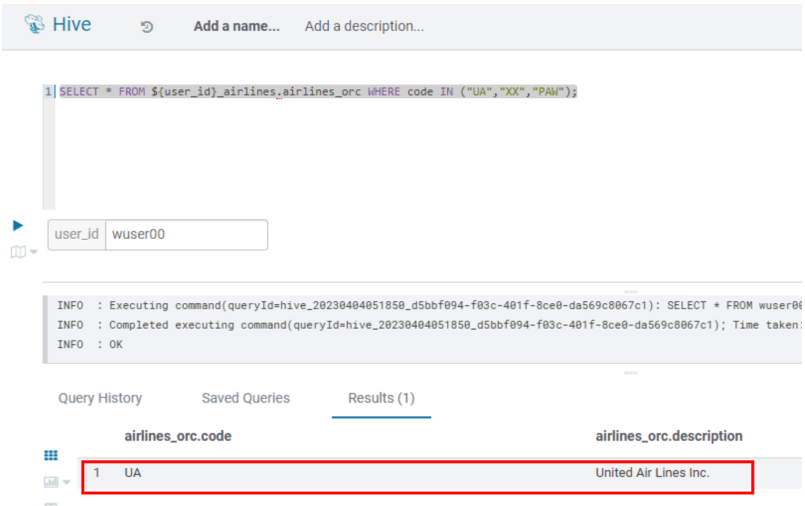
Step 3 : CREATE AND LOAD AIRLINES TABLE

Create airlines table in Hive table format and orc file format. This table should also be fully ACID capable. We will use Create Table As Select (CTAS). Since, airlines table can change we need the ability to Insert/Update/Delete records.
Run the following query to create the table

```
Unset
drop table if exists ${user_id}_airlines.airlines_orc;
CREATE TABLE ${user_id}_airlines.airlines_orc
STORED AS ORC
AS
  SELECT * FROM ${user_id}_airlines_raw.airlines_csv;
```

Run the following query to check data in the airlines_orc table and it should return only 1 row for code 'UA'.

```
Unset
SELECT * FROM ${user_id}_airlines.airlines_orc WHERE code IN
("UA","XX","PAW");
```



Step 4 : CHECK ACID CAPABILITIES

- **Insert New record**
  We shall now add a new record to the airlines_orc table to see some Hive ACID capabilities.

```
Unset
INSERT INTO ${user_id}_airlines.airlines_orc
VALUES("PAW","Paradise Air");
```

- **Update Existing Record**
  Let's update an existing record to change the description of United Airlines to Adrenaline Airlines to see more of the ACID capablities provided by Hive ACID. Run the following SQL.

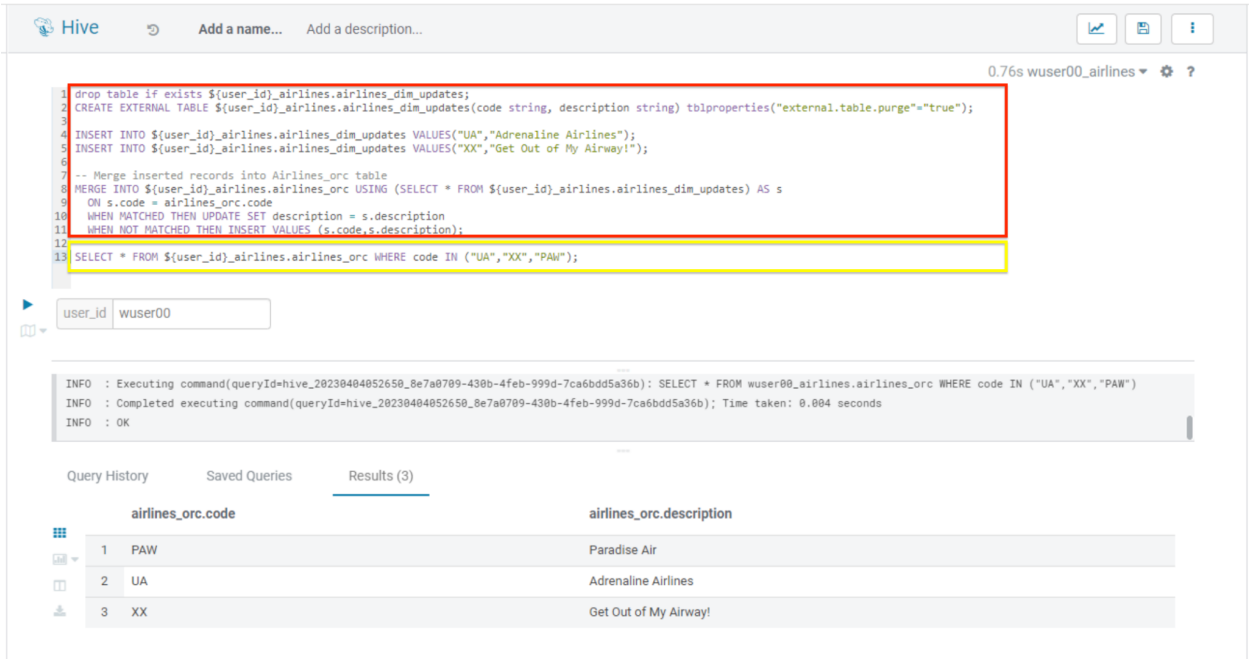
```
Unset
drop table if exists ${user_id}_airlines.airlines_dim_updates;
CREATE EXTERNAL TABLE
${user_id}_airlines.airlines_dim_updates(code string, description
string) tblproperties("external.table.purge"="true");

INSERT INTO ${user_id}_airlines.airlines_dim_updates
VALUES("UA","Adrenaline Airlines");
INSERT INTO ${user_id}_airlines.airlines_dim_updates
VALUES("XX","Get Out of My Airway!");
```

```
-- Merge inserted records into Airlines_orc table
MERGE INTO ${user_id}_airlines.airlines_orc USING (SELECT * FROM
${user_id}_airlines.airlines_dim_updates) AS s
  ON s.code = airlines_orc.code
  WHEN MATCHED THEN UPDATE SET description = s.description
  WHEN NOT MATCHED THEN INSERT VALUES (s.code,s.description);
```

Run the following query to return the following result - codes XX and PAW were inserted rows, and code UA which had its description value changed from United Air Lines Inc. to Adrenaline Airlines.

Unset

```
SELECT * FROM ${user_id}_airlines.airlines_orc WHERE code IN
("UA","XX","PAW");
```



MIGRATE HIVE TO ICEBERG TABLE

If you already have created a Data Warehouse using the Hive Table Format but would like to take advantage of the features offered in the Iceberg Table Format, you have 2 options.

- Utilize the table Migration feature
- Use Create Table as Select (CTAS)

Utilize Table Migration Feature

Run the following SQL and note what happens next.

```
Unset
ALTER TABLE ${user_id}_airlines.planes
SET TBLPROPERTIES
('storage_handler'='org.apache.iceberg.mr.hive.HiveIcebergStorage
Handler');

DESCRIBE FORMATTED ${user_id}_airlines.planes;
```

The following changes occured:

- This migration to Iceberg happened in-place & there was no re-writing of data that occurred as part of this process. It retained the File Format of parquet for the Iceberg table as well. There was a Metadata file that is created, which you can see when you run the DESCRIBE FORMATTED.

- In the output look for the following fields - look for the following (see image with highlighted fields) key values:

Location	Data is stored in cloud storage and in this case AWS S3 in the same location as the Hive Table Format.
Table Type	Indicates that it is an EXTERNAL TABLE.
MIGRATED_TO_ICEBERG	Indicates that the table has migrated to ICEBERG.[TRUE]
table_type	Indicates ICEBERG table format.
metadata_location	Indicates the location of metadata which is path to cloud storage.
storage_handler	org.apache.iceberg.mr.hive.HiveIcebergStorageHandler.
SerDe Library	org.apache.iceberg.mr.hive.HiveIcebergSerDe.

Use Create table as Select(CTAS)

Run the following SQL to create airports table using CTAS. Notice the syntax to create an Iceberg Table within Hive is "***Stored by Iceberg***".

```
Unset
drop table if exists ${user_id}_airlines.airports;
CREATE EXTERNAL TABLE ${user_id}_airlines.airports
STORED BY ICEBERG AS
  SELECT * FROM ${user_id}_airlines_raw.airports_csv;

DESCRIBE FORMATTED ${user_id}_airlines.airports;
```

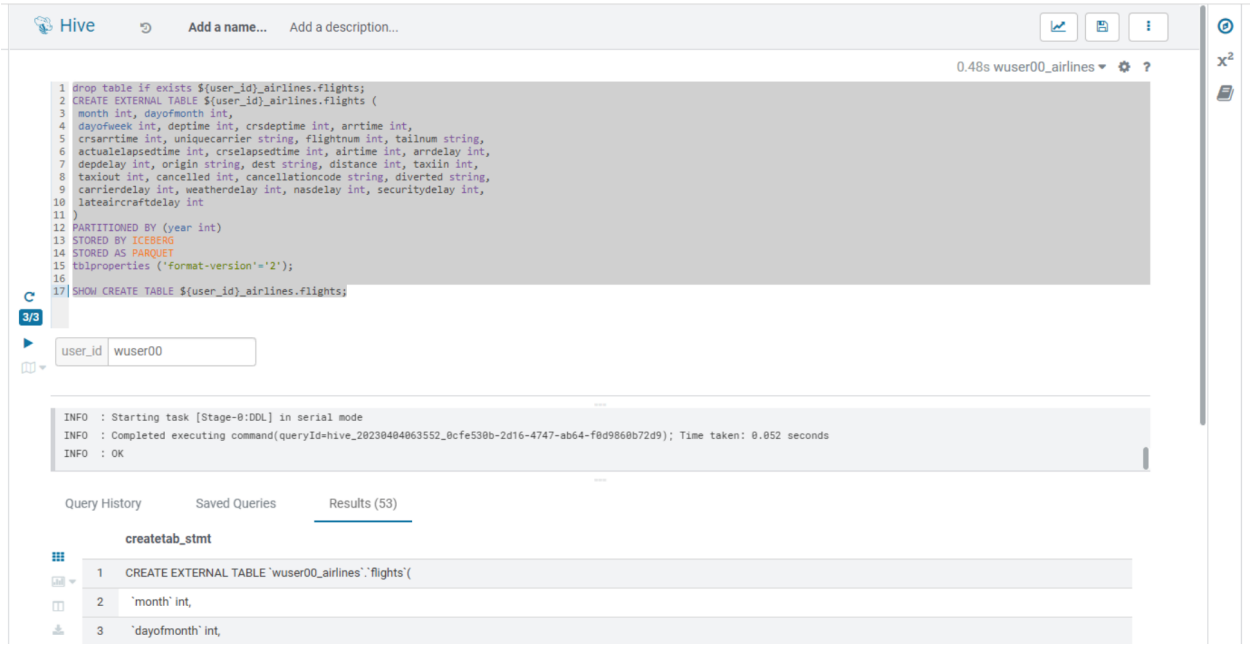
Look for: Table Type, Location (location of where table data is stored), SerDe Library, and in Table Parameters look for properties storage_handler, metadata_location, and table_type.

CREATE ICEBERG TABLES

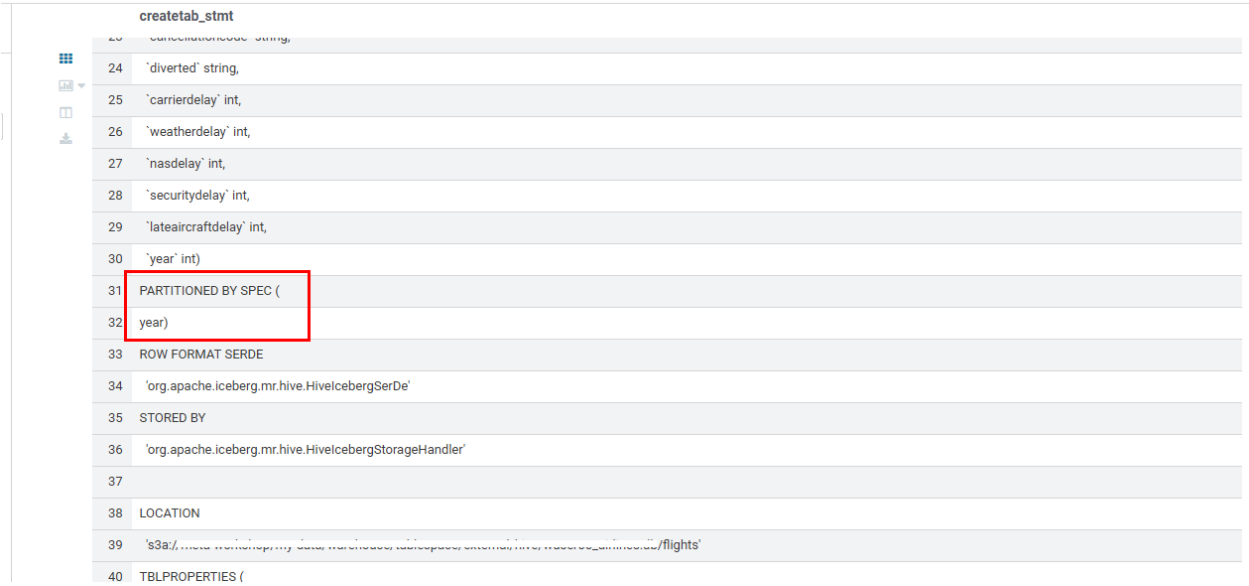
In this step we will create a partitioned table, in Iceberg Table Format, stored in Parquet File Format. Other than that we could specify other file formats that are supported for Iceberg which are: ORC and Avro.

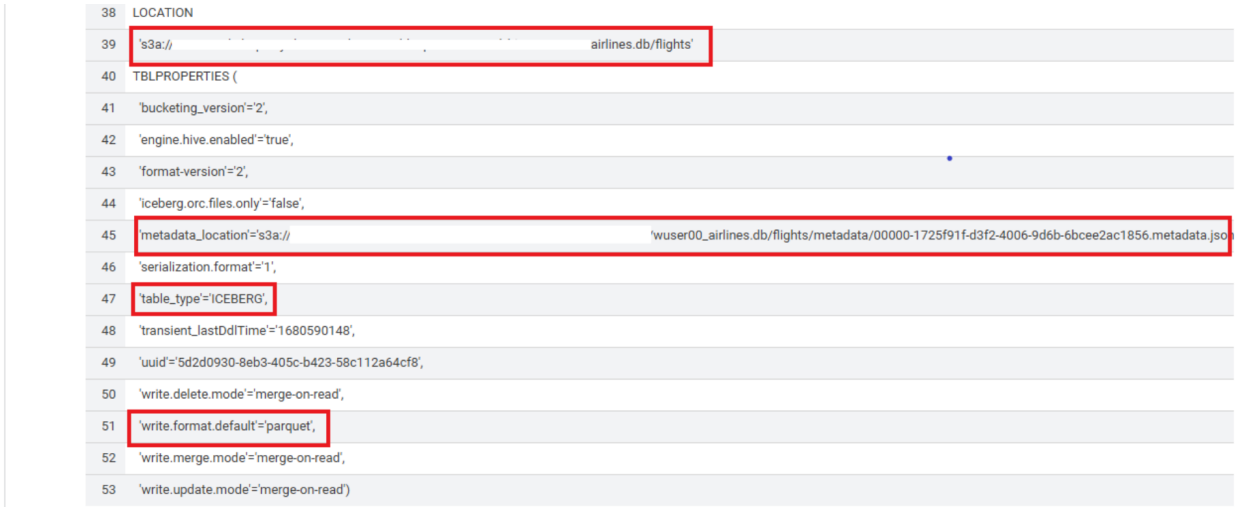
```
Unset
drop table if exists ${user_id}_airlines.flights;
CREATE EXTERNAL TABLE ${user_id}_airlines.flights (
 month int, dayofmonth int,
 dayofweek int, deptime int, crsdeptime int, arrtime int,
 crsarrtime int, uniquecarrier string, flightnum int, tailnum
string,
 actualelapsedtime int, crselapsedtime int, airtime int, arrdelay
int,
 depdelay int, origin string, dest string, distance int, taxiin
int,
 taxiout int, cancelled int, cancellationcode string, diverted
string,
 carrierdelay int, weatherdelay int, nasdelay int, securitydelay
int,
```

```
 lateaircraftdelay int
)
PARTITIONED BY (year int)
STORED BY ICEBERG
STORED AS PARQUET
tblproperties ('format-version'='2');

SHOW CREATE TABLE ${user_id}_airlines.flights;
```



The SHOW CREATE TABLE command is the unformatted version of DESCRIBE FORMATTED command. Pay attention to the PARTITIONED BY SPEC, where we have partitioned the flights table using the **year** column.





We insert data into this table it will write data together within the same partition (ie. all 2006 data is written to the same location, all 2005 data is written to the same location, etc.). This command will take some time to run.

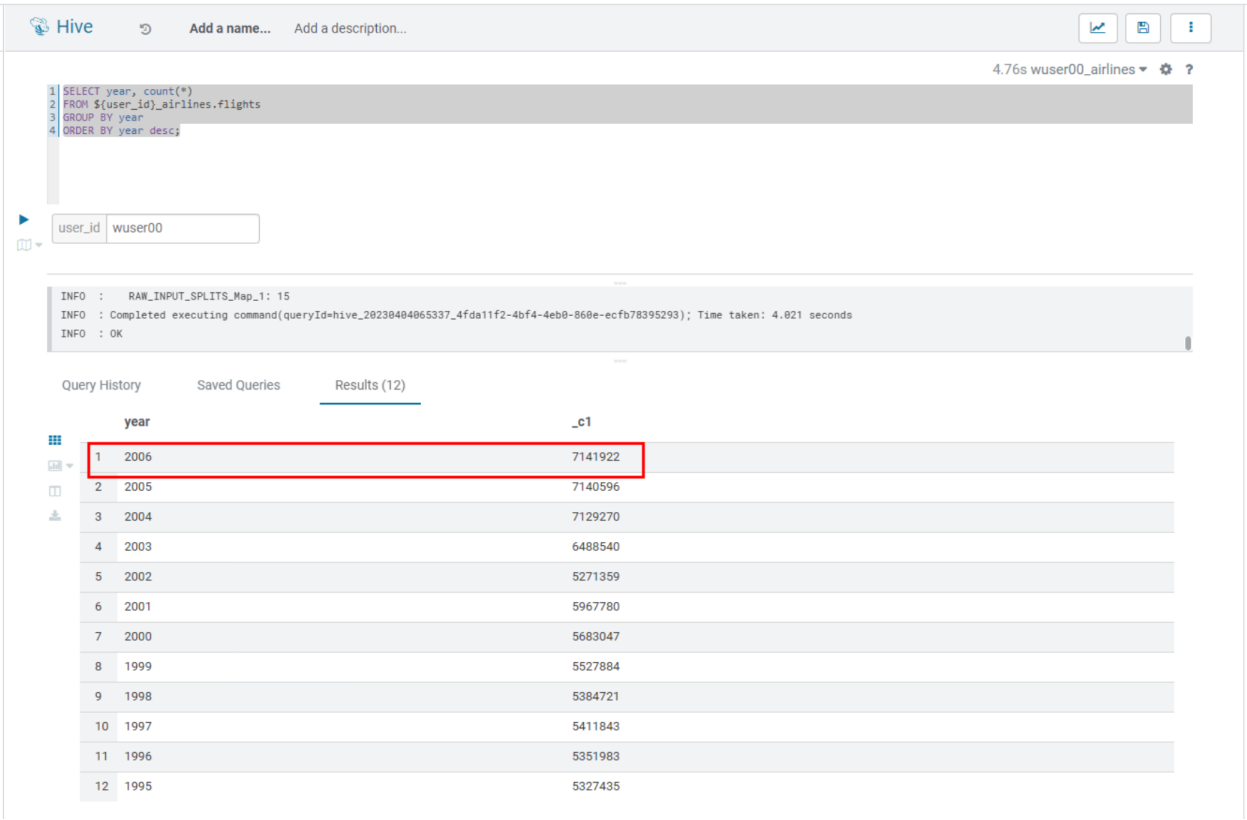
Unset

```
INSERT INTO ${user_id}_airlines.flights
SELECT * FROM ${user_id}_airlines_raw.flights_csv
WHERE year <= 2006;
```

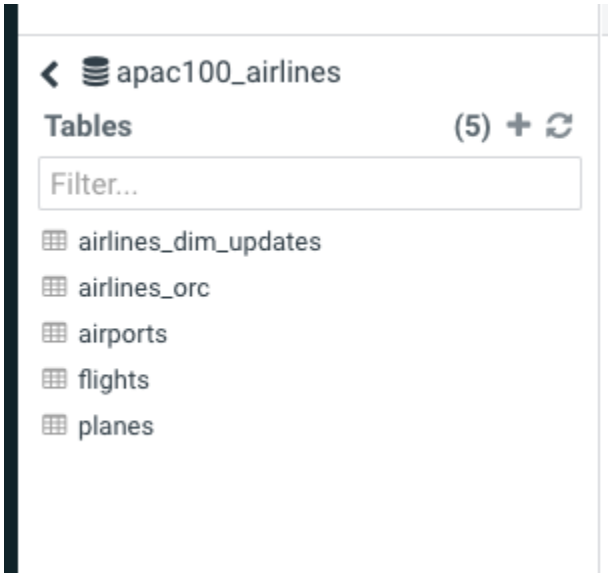
Run the following SQL and notice that each of the years have a range of data within a few million flights (each record in the flights table counts as a flight).

Unset

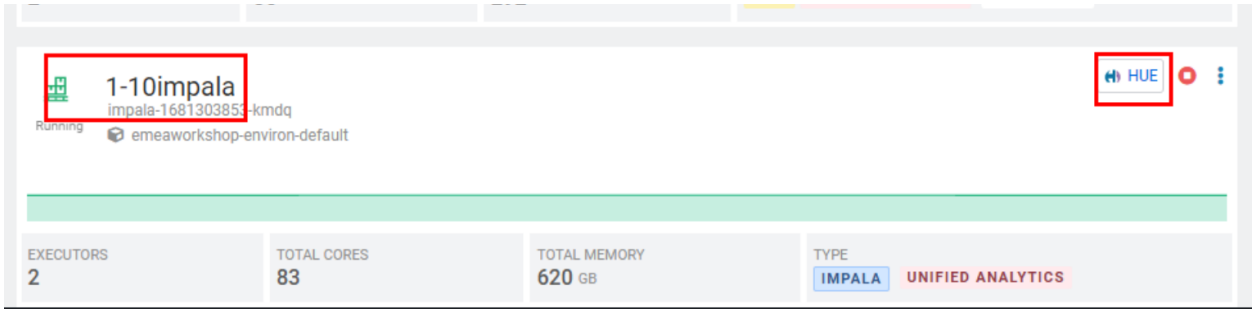
```
SELECT year, count(*)
FROM ${user_id}_airlines.flights
GROUP BY year
ORDER BY year desc;
```



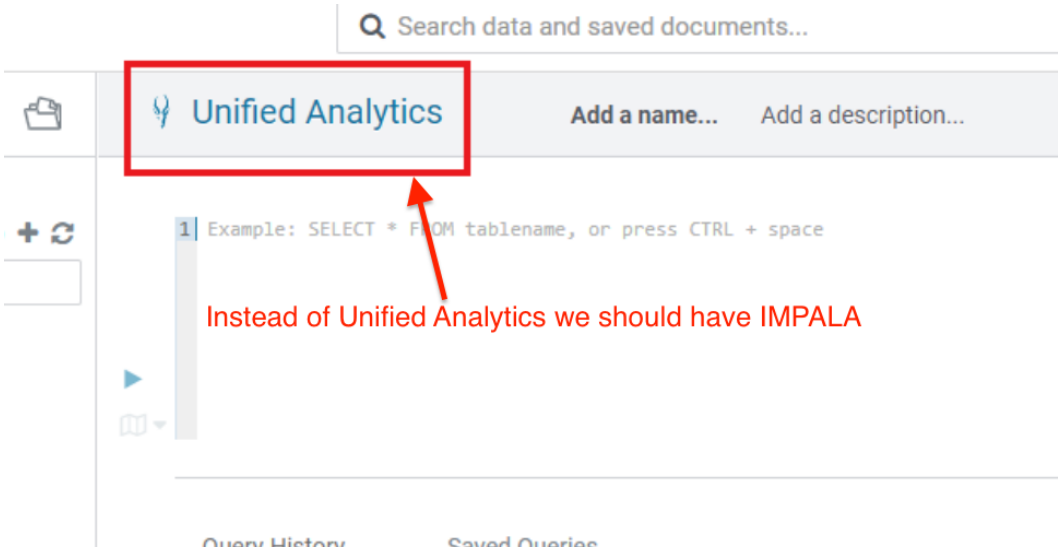
Now, make sure that the following 5 tables are created up until this point as shown in the screenshot below.



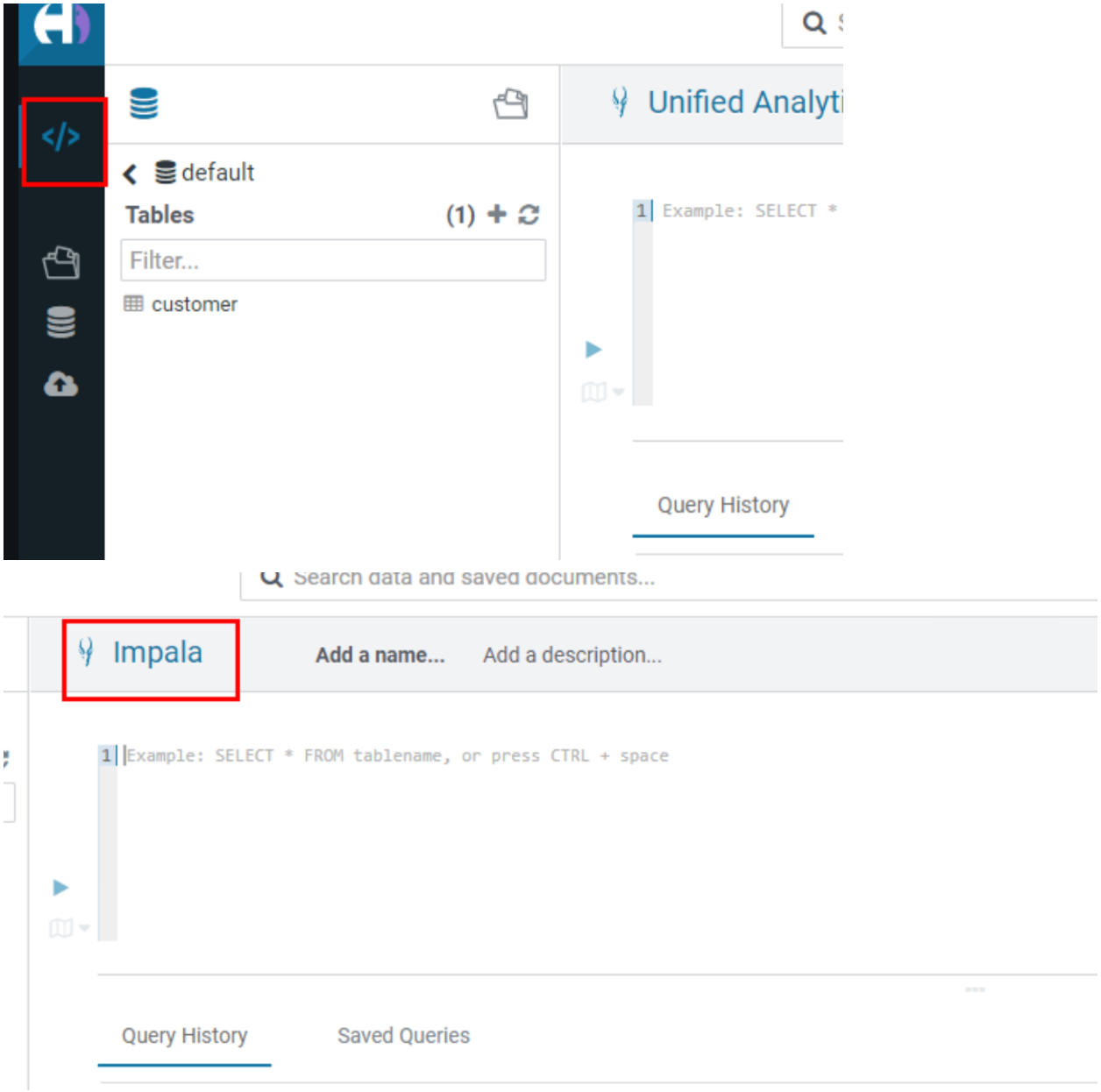
As a final step here let's run the same analytic query we ran against the Raw layer now in our Data Lakehouse DW, to see what happens with performance. From the cloudera console select the IMPALA virtual warehouse assigned to you



Make sure that 'Unified Analytics' is NOT selected.



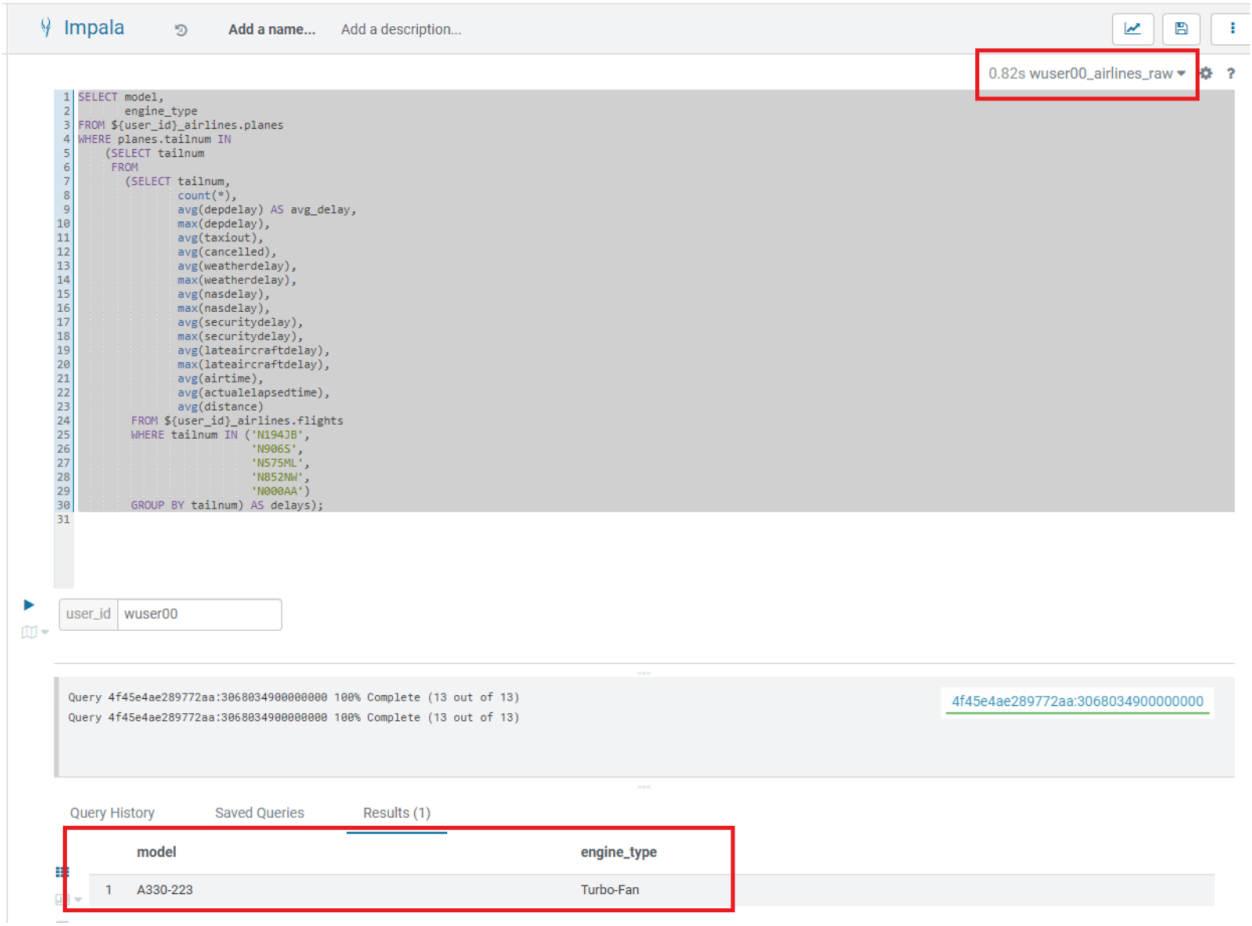
Instead click on the Editor option in the left top corner and select Impala editor.



Now run the following query again.

```
Unset
SELECT model,
       engine_type
FROM ${user_id}_airlines.planes
```

```
WHERE planes.tailnum IN
    (SELECT tailnum
     FROM
       (SELECT tailnum,
               count(*),
               avg(depdelay) AS avg_delay,
               max(depdelay),
               avg(taxiout),
               avg(cancelled),
               avg(weatherdelay),
               max(weatherdelay),
               avg(nasdelay),
               max(nasdelay),
               avg(securitydelay),
               max(securitydelay),
               avg(lateaircraftdelay),
               max(lateaircraftdelay),
               avg(airtime),
               avg(actualelapsedtime),
               avg(distance)
        FROM ${user_id}_airlines.flights
        WHERE tailnum IN ('N194JB',
                          'N906S',
                          'N575ML',
                          'N852NW',
                          'N000AA')
        GROUP BY tailnum) AS delays);
```



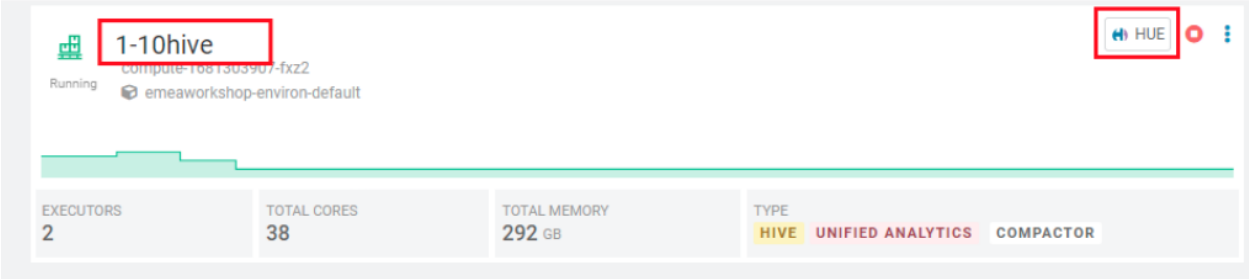
The Data Lakehouse DW query performs significantly better than the same query running against the CSV data.

LAB 4 - PERFORMANCE OPTIMIZATION AND TABLE MAINTENANCE

In this Step we will look at some of the performance optimization and table maintenance tasks that can be performed to ensure the best possible TCO, while ensuring the best performance.

Iceberg in-place Partition Evolution[Performance Optimization]

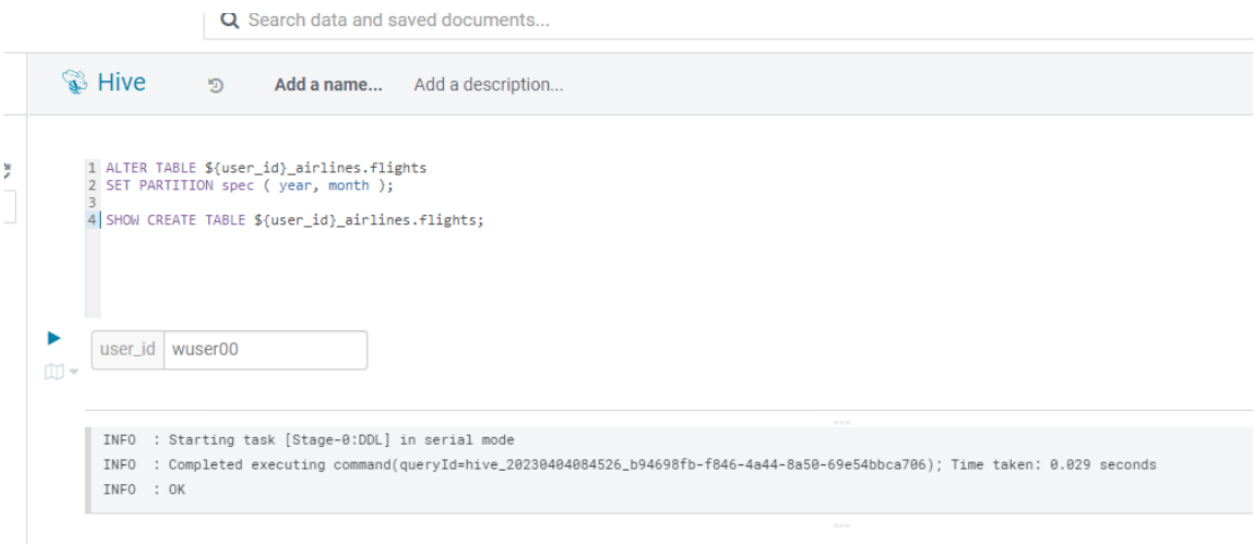
Step 1 : Open HUE for the CDW Hive Virtual Warehouse



Step 2 : Execute Query

One of the key features for Iceberg tables is the ability to evolve the partition that is being used over time.

```
Unset
ALTER TABLE ${user_id}_airlines.flights
SET PARTITION spec ( year, month );

SHOW CREATE TABLE ${user_id}_airlines.flights;
```



Check for the following where now the partition is by year, month.

```
31  PARTITIONED BY SPEC (
32  year,
33  month)
34  ROW FORMAT SERDE
35   'org.apache.iceberg.mr.hive.HiveIcebergSerDe'
36  STORED BY
37   'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler'
38
```

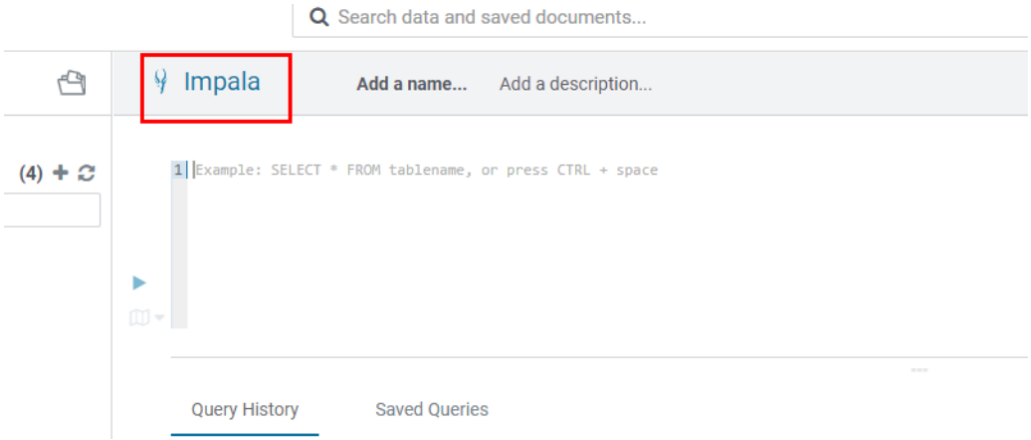
Step 3 : Check for optimizations

- Load new data into the flights table using the NEW partition definition. This query will take a while to run

Unset

```
INSERT INTO ${user_id}_airlines.flights
SELECT * FROM ${user_id}_airlines_raw.flights_csv
WHERE year = 2007;
```

- Go to **IMPALA** virtual warehouse and switch the Editor to use IMPALA instead of UNIFIED ANALYTICS

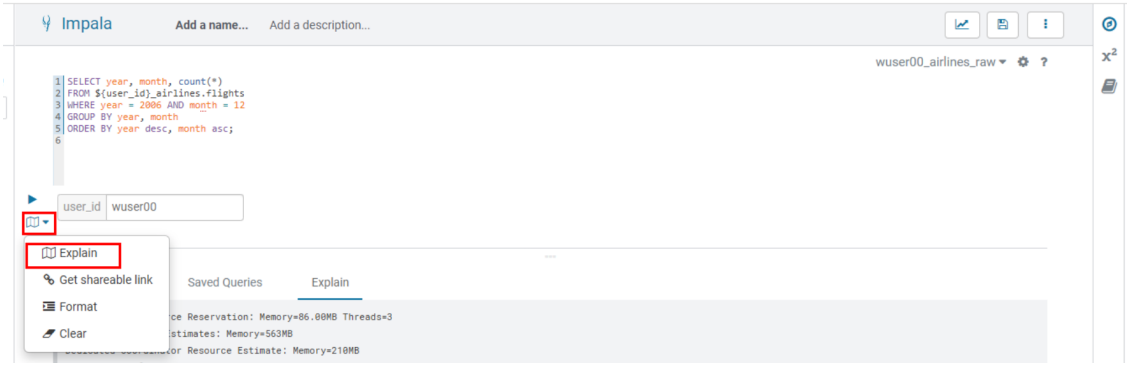


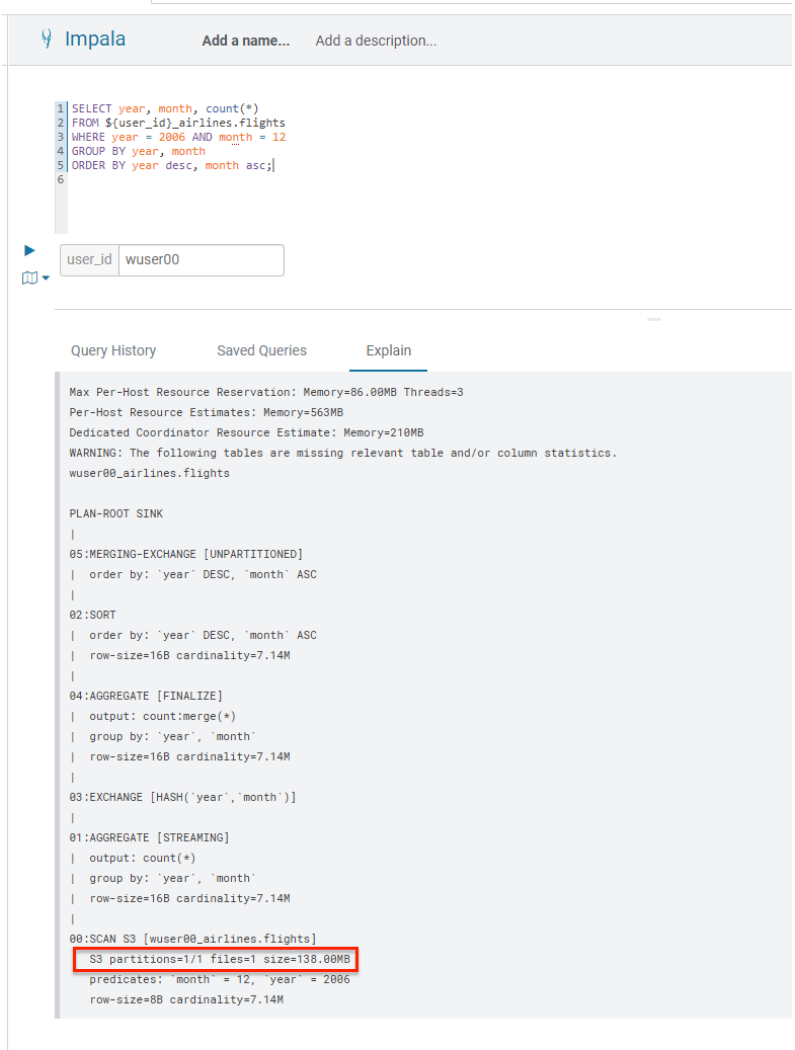
- Copy/paste the following in the HUE Editor, but **DO NOT** execute the query.

Unset

```
SELECT year, month, count(*)
FROM ${user_id}_airlines.flights
WHERE year = 2006 AND month = 12
GROUP BY year, month
ORDER BY year desc, month asc;
```

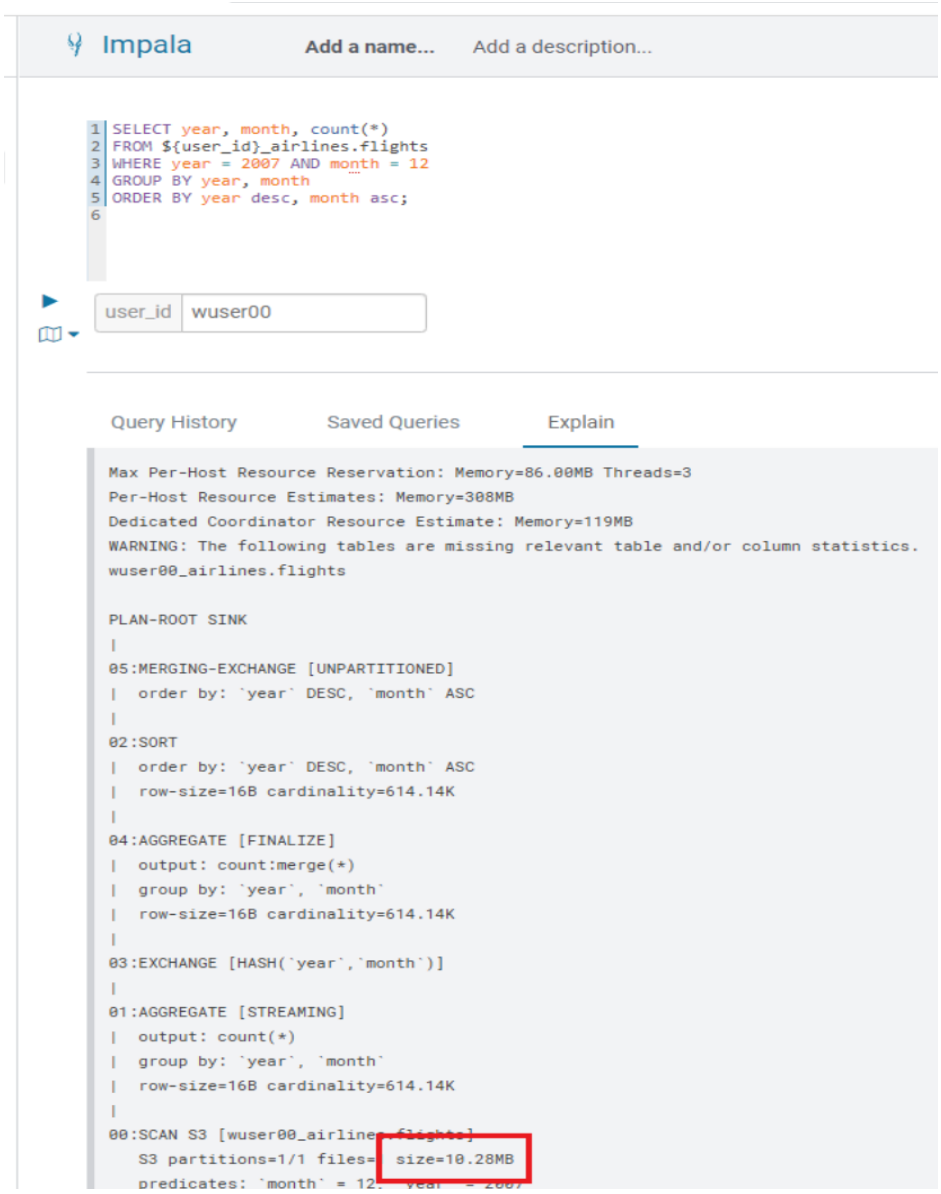
Run Explain Plan against the above analytic queries to see how the new partition helps.





- Copy/paste the following in the HUE Editor, but DO NOT execute the query but check the EXPLAIN PLAN.

```
Unset
SELECT year, month, count(*)
FROM ${user_id}_airlines.flights
WHERE year = 2007 AND month = 12
GROUP BY year, month
ORDER BY year desc, month asc;
```



In the output notice the amount of data that needs to be scanned for this query, about 11 MB, is significantly less than that of the first, 138 MB. This shows an important capability of Iceberg, Partition Pruning. Meaning that much less data is scanned for this query and only the selected month of data needs to be processed.

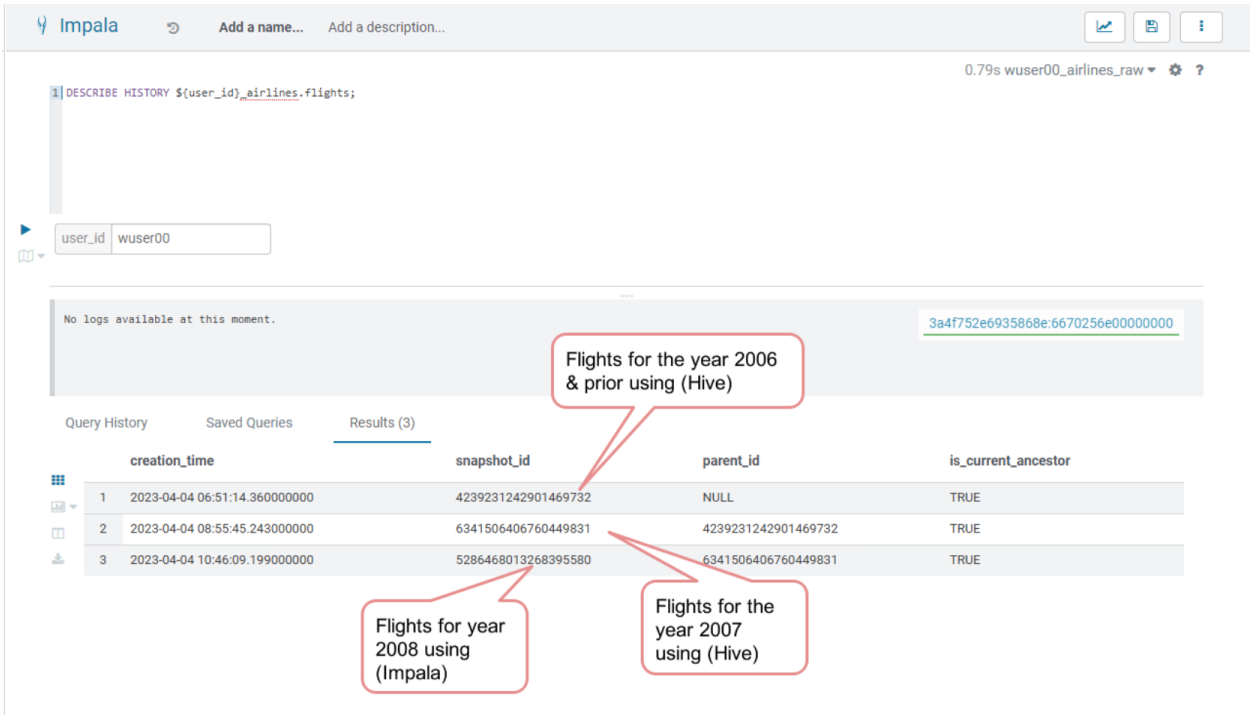
Iceberg Snapshots [Table Maintenance]

In the previous steps we have loaded data into the flights iceberg table. We will insert more data into it. Each time we add (update or delete) data a snapshot is captured. The snapshot is important for eventual consistency & to allow multiple read/writes concurrently (from various engines or same engine).

```
Unset
INSERT INTO ${user_id}_airlines.flights
SELECT * FROM ${user_id}_airlines_raw.flights_csv
WHERE year >= 2008;
```

To see snapshots, execute the following SQL.

```
Unset
DESCRIBE HISTORY ${user_id}_airlines.flights;
```



In the output there should be 3 Snapshots, described below. Note that we have been reading/writing data from/to the Iceberg table from both Hive & Impala. It is an important aspect of Iceberg Tables that they support multi-function analytics - ie. many engines can work with Iceberg tables (Cloudera Data Warehouse [Hive & Impala], Cloudera Data Engineering [Spark], Cloudera Machine Learning [Spark], Cloudera DataFlow [NiFi], and DataHub Clusters).

Get the details of the snapshots and store it in a notepad.

	creation_time	snapshot_id	parent_id	is_current_ancestor
1	2023-04-04 06:51:14.360000000	4239231242901469732	NULL	TRUE
2	2023-04-04 08:55:45.243000000	6341506406760449831	4239231242901469732	TRUE
3	2023-04-04 10:46:09.199000000	5286468013268395580	6341506406760449831	TRUE

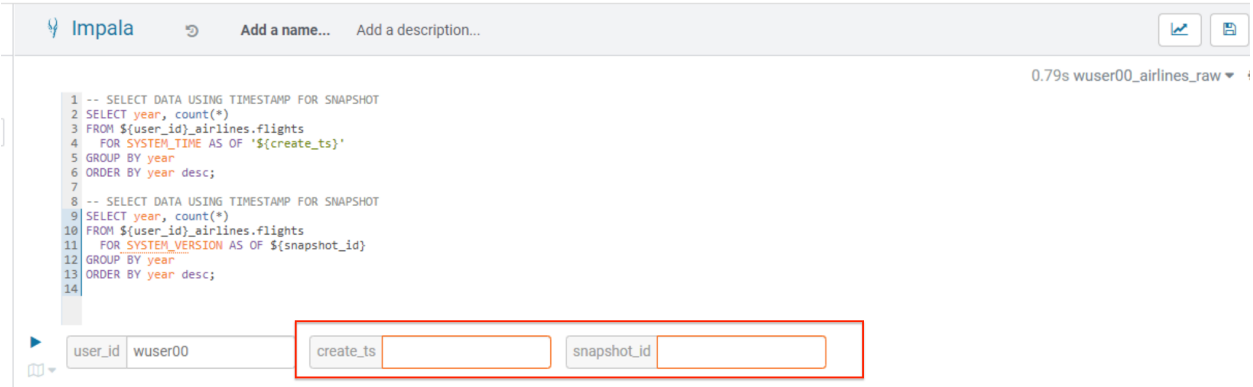
```
creation_time
-------------
2023-04-04 06:51:14.360000000
2023-04-04 08:55:45.243000000
2023-04-04 10:46:09.199000000


snapshot_id
------------
4239231242901469732
6341506406760449831
5286468013268395580
```

Iceberg Time Travel [Table Maintenance]

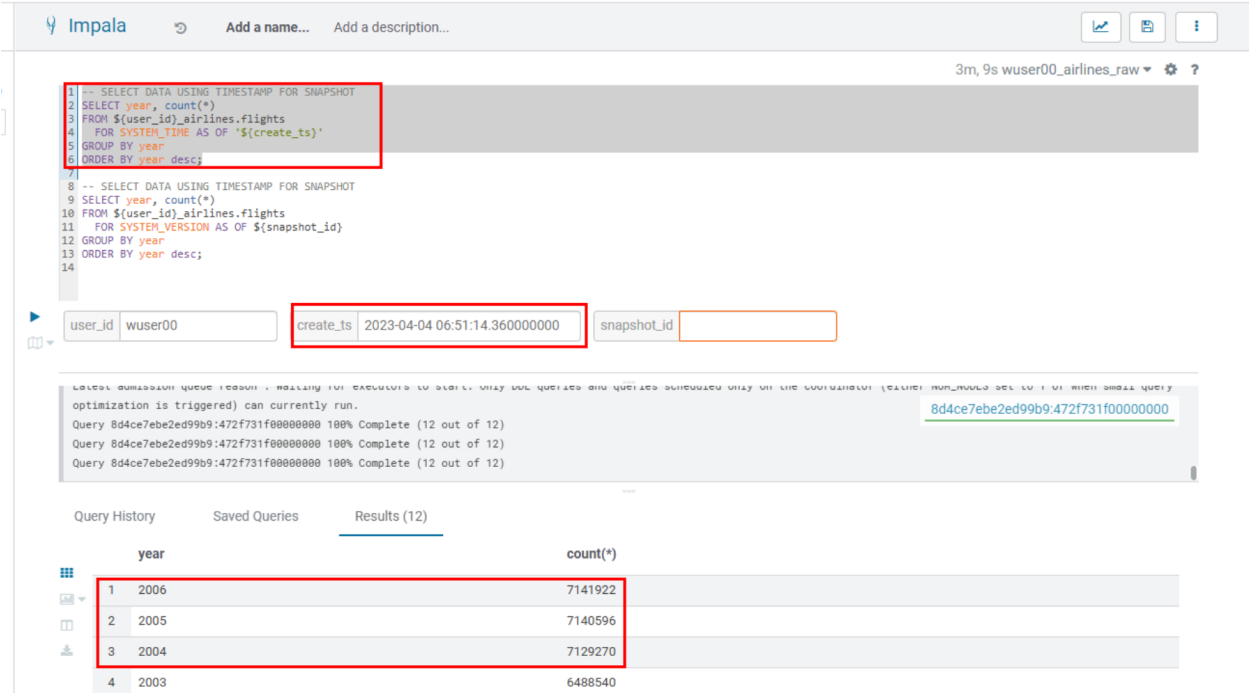
- Copy/paste the following data into the Impala Editor, but DO NOT execute.

```
Unset
-- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
SELECT year, count(*)
FROM ${user_id}_airlines.flights
  FOR SYSTEM_TIME AS OF '${create_ts}'
GROUP BY year
ORDER BY year desc;

-- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
SELECT year, count(*)
FROM ${user_id}_airlines.flights
  FOR SYSTEM_VERSION AS OF ${snapshot_id}
GROUP BY year
ORDER BY year desc;
```

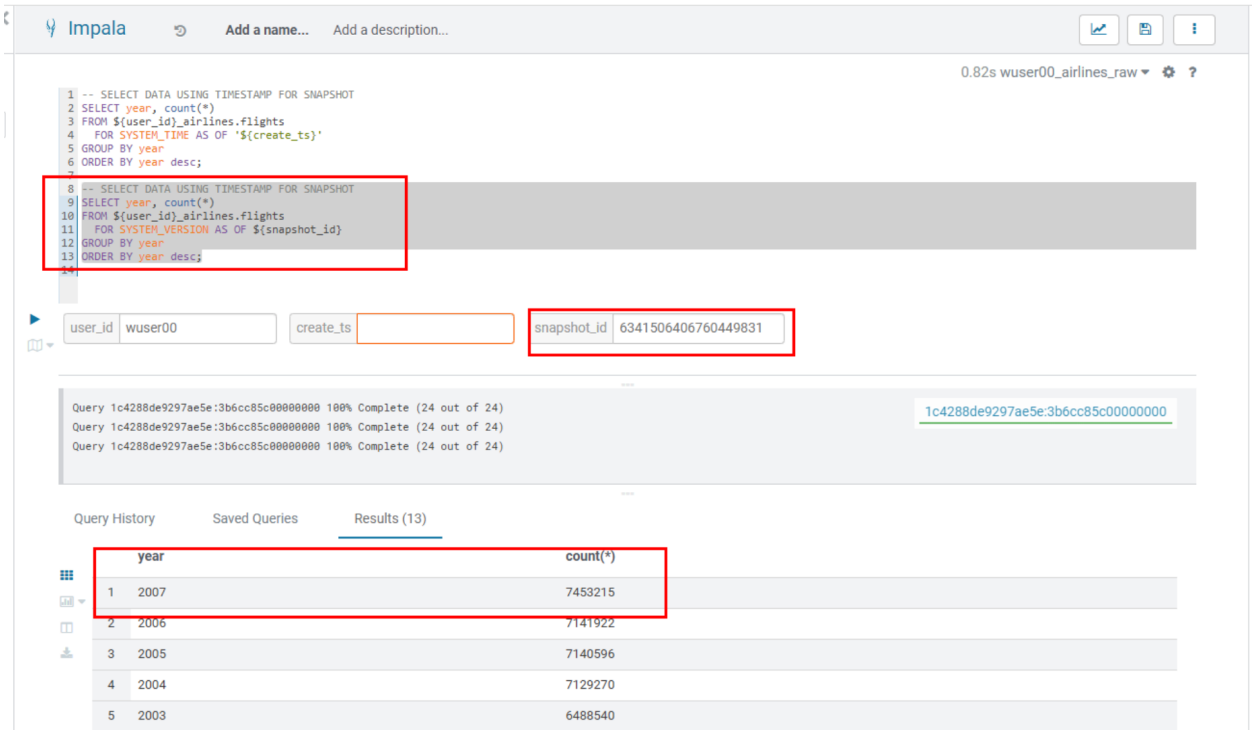
After pasting the query you will see the following two options for **create_ts** and **snapshot_id**



- From the notepad just copy the first value of the timestamp. It could be the date or the timestamp. Paste it in the create_ts box. In our case the value was 2023-04-04 06:51:14.360000000. Then execute the higlighted query only (1st query).



- From the notepad just copy the second value of the snapshot id. In our case the value was 6341506406760449831. Paste it in the snapshot_id box. Then execute the highlighted query only (2nd query).



Iceberg Rollback [Table Maintenance] - DO NOT RUN

Sometimes data can be loaded incorrectly, due to many common issues - missing fields, only part of the data being loaded, bad data, etc.

In situations like this data would need to be removed, corrected, and reloaded. Iceberg can help with the Rollback command to remove the “unwanted” data.

This leverages Snapshot IDs to perform this action by using a simple ALTER TABLE command as follows. We will NOT RUN this command in this lab.

```
Unset
-- ALTER TABLE ${user_id}_airlines.flights EXECUTE
ROLLBACK(${snapshot_id});
```

Iceberg Rollback [Table Maintenance] - DO NOT RUN

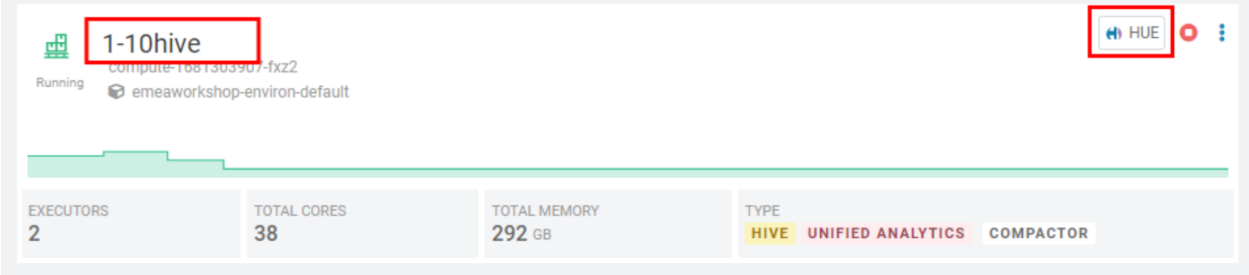
As time passes it might make sense to expire old Snapshots, instead of the Snapshot ID you use the Timestamp to expire old Snapshots. You can do this manually by running a simple ALTER TABLE command as follows. We will NOT RUN this command in this lab.

```
Unset
-- Expire Snapshots up to the specified timestamp
-- BE CAREFUL: Once you run this you will not be able to Time
Travel for any Snapshots that you Expire ALTER TABLE
${user_id}_airlines.flights
-- ALTER TABLE ${user_id}_airlines_maint.flights EXECUTE
expire_snapshots('${create_ts}');
```

Materialized Views [Performance Optimization]

This can be used for both Iceberg tables and Hive tables.

Go to the HUE UI of the HIVE virtual warehouse assigned to you



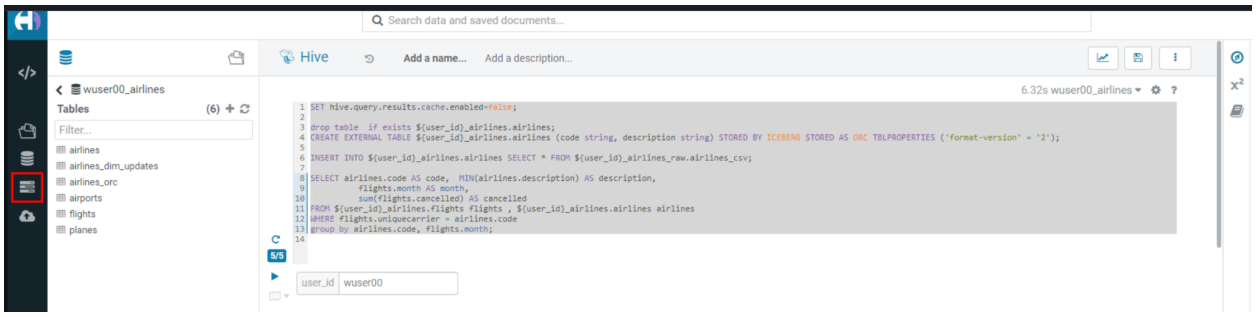
- Copy/paste the following, make sure to highlight the entire block, and execute the following.

```
Unset
SET hive.query.results.cache.enabled=false;

drop table  if exists ${user_id}_airlines.airlines;
CREATE EXTERNAL TABLE ${user_id}_airlines.airlines (code string,
description string) STORED BY ICEBERG STORED AS ORC TBLPROPERTIES
('format-version' = '2');

INSERT INTO ${user_id}_airlines.airlines SELECT * FROM
${user_id}_airlines_raw.airlines_csv;
```

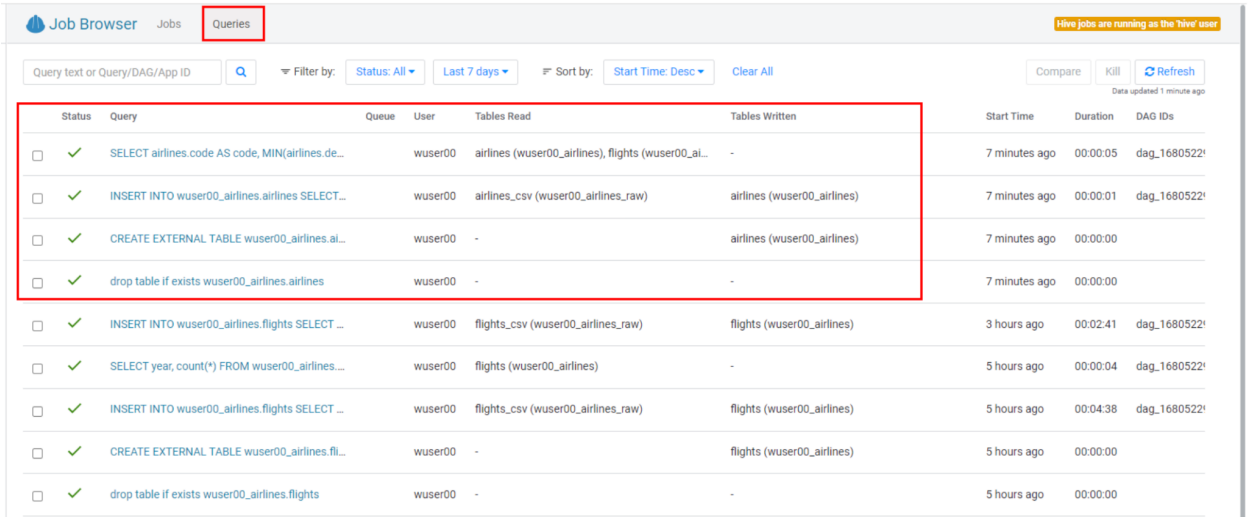
```
SELECT airlines.code AS code,  MIN(airlines.description) AS
description,
          flights.month AS month,
          sum(flights.cancelled) AS cancelled
FROM ${user_id}_airlines.flights flights ,
${user_id}_airlines.airlines airlines
WHERE flights.uniquecarrier = airlines.code
group by airlines.code, flights.month;
```

Note: Hive has built in performance improvements, such as a **Query Cache** that stores results of queries run so that similar queries don't have to retrieve data, they can just get the results from the cache. In this step we are turning that off using the SET statement, this will ensure when we look at the query plan, we will not retrieve the data from the cache.
Note: With this query you are combining an Iceberg Table Format (flight table) with a Hive Table Format (airlines ORC table) in the same query.

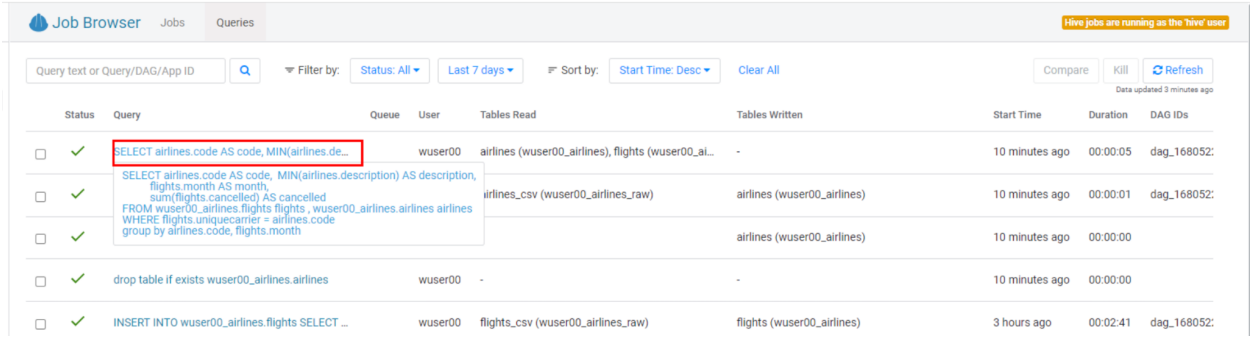
- Let's look at the Query Plan that was used to execute this query. On the left side click on Jobs, as shown in the screenshot below.



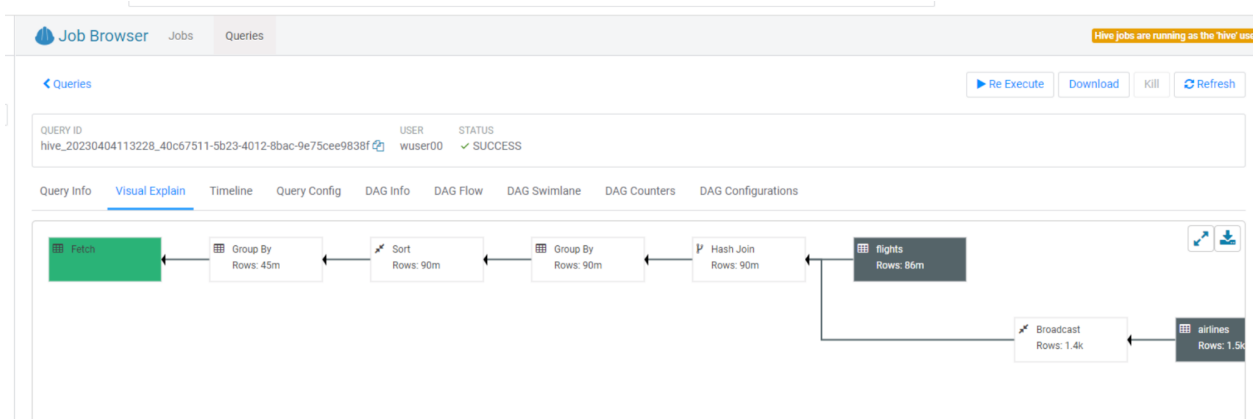
- Then click on Queries. This is where an Admin will go when he wants to investigate the queries. In our case for this lab, we'd like to look at the query we just executed to see how it ran and the steps taken to execute the query. Administrators would also be able to perform other monitoring and maintenance tasks for what is running (or has been run). Monitoring and maintenance tasks could include cancel (kill) queries, see what is running, analyze whether queries that have been executed are optimized, etc.



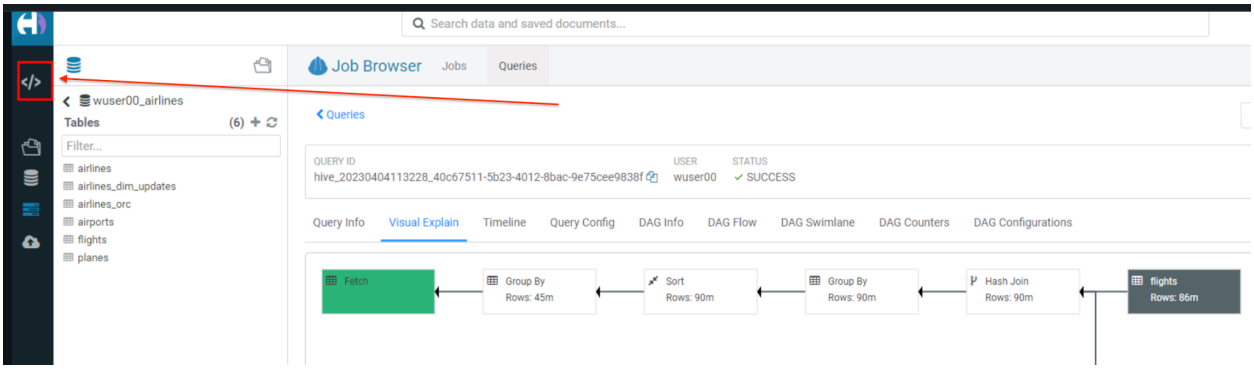
- Click on the first query as shown below. Make sure that this is the latest query. You can look at the `Start Time' field here to know if it's the latest or not.



- This is where you can analyze queries at a deep level. For this lab let's take a look at the Explain details, by clicking on the **Visual Explain** tab. It might take a while to appear, please click on refresh. This plan shows that this query needs to Read flights (86M rows) and airlines (1.5K rows) with hash join, group, and sort. This is a lot of data processing and if we run this query constantly it would be good to reduce the time this query takes to execute.

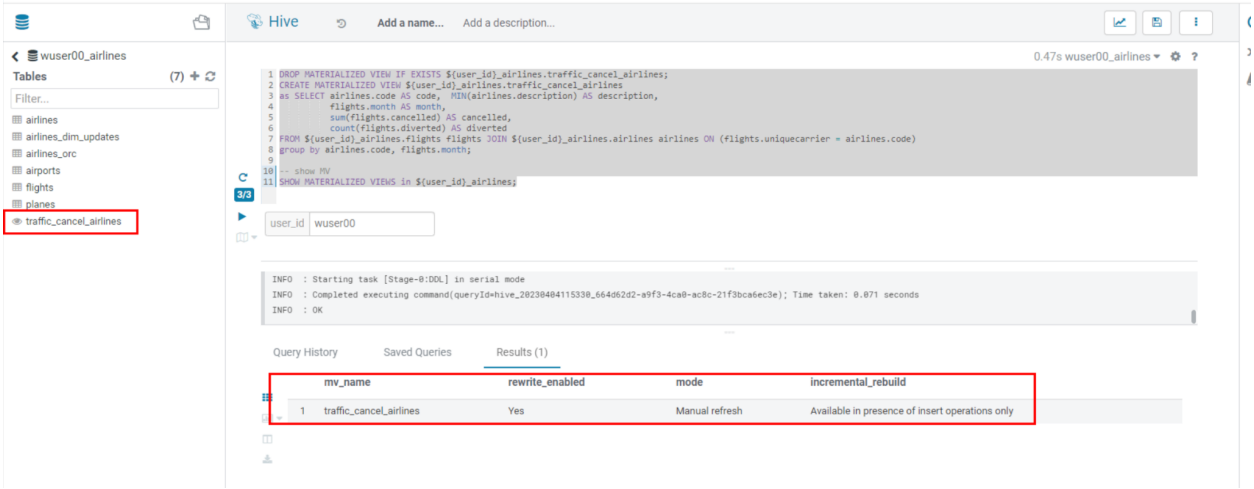


- Click on the Editor option on the left side as shown.



- **Create Materialized View (MV) -** Queries will transparently be rewritten, when possible, to use the MV instead of the base tables. Copy/paste the following, highlight the entire block, and execute.

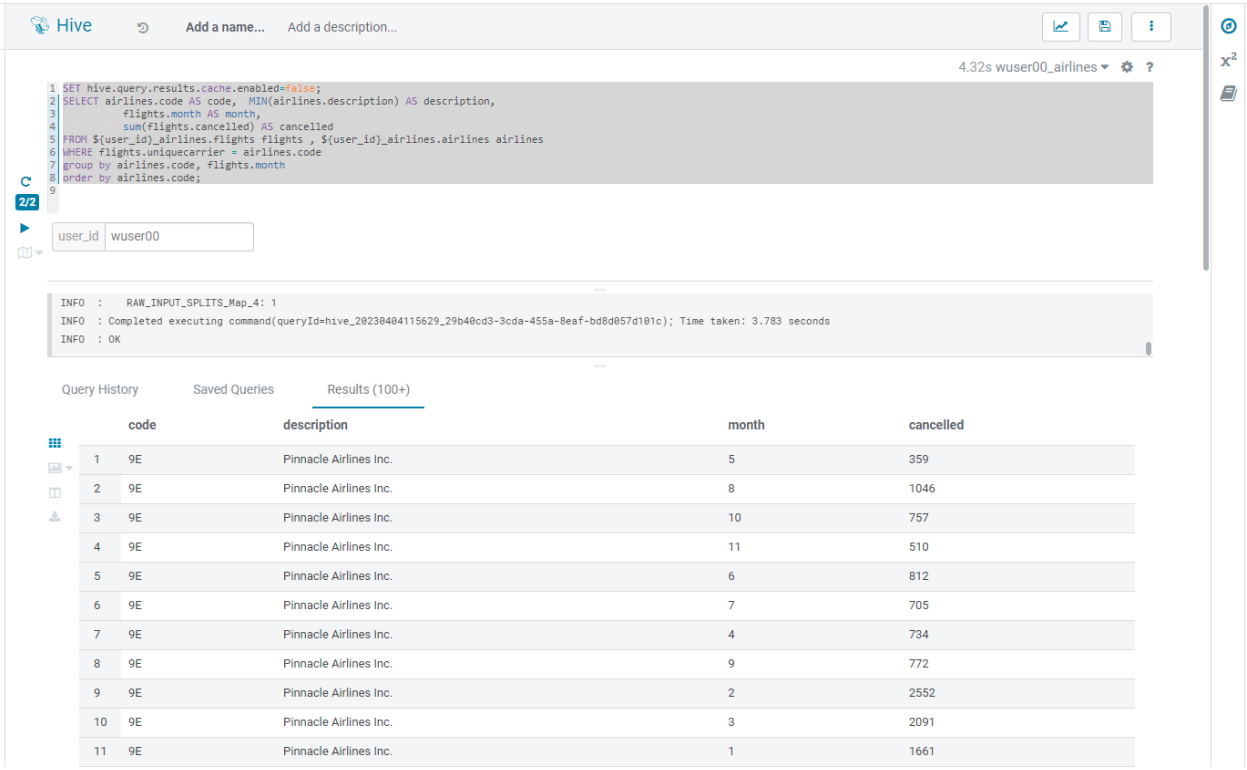
```
Unset
DROP MATERIALIZED VIEW IF EXISTS
${user_id}_airlines.traffic_cancel_airlines;
CREATE MATERIALIZED VIEW ${user_id}_airlines.traffic_cancel_airlines
as SELECT airlines.code AS code,  MIN(airlines.description) AS
description,
          flights.month AS month,
          sum(flights.cancelled) AS cancelled,
          count(flights.diverted) AS diverted
FROM ${user_id}_airlines.flights flights JOIN
${user_id}_airlines.airlines airlines ON (flights.uniquecarrier =
airlines.code)
group by airlines.code, flights.month;

-- show MV
SHOW MATERIALIZED VIEWS in ${user_id}_airlines;
```



- Run Dashboard Query again to see usage of the MV - Copy/paste the following, make sure to highlight the entire block, and execute the following. This time an order by was added to make this query must do more work.

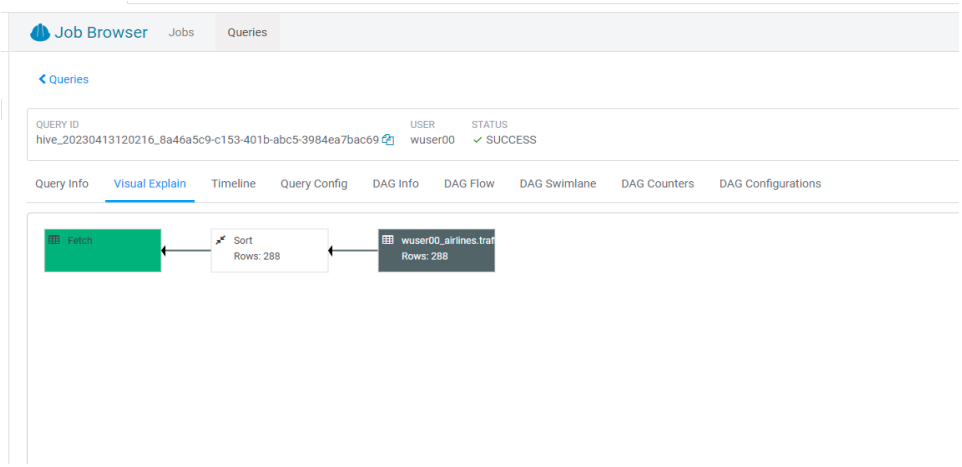
Unset

```
SET hive.query.results.cache.enabled=false;
SELECT airlines.code AS code,  MIN(airlines.description) AS
description,
          flights.month AS month,
          sum(flights.cancelled) AS cancelled
FROM ${user_id}_airlines.flights flights ,
${user_id}_airlines.airlines airlines
WHERE flights.uniquecarrier = airlines.code
group by airlines.code, flights.month
order by airlines.code;
```



	code	description	month	cancelled
1	9E	Pinnacle Airlines Inc.	5	359
2	9E	Pinnacle Airlines Inc.	8	1046
3	9E	Pinnacle Airlines Inc.	10	757
4	9E	Pinnacle Airlines Inc.	11	510
5	9E	Pinnacle Airlines Inc.	6	812
6	9E	Pinnacle Airlines Inc.	7	705
7	9E	Pinnacle Airlines Inc.	4	734
8	9E	Pinnacle Airlines Inc.	9	772
9	9E	Pinnacle Airlines Inc.	2	2552
10	9E	Pinnacle Airlines Inc.	3	2091
11	9E	Pinnacle Airlines Inc.	1	1661

Let's look at the Query Plan that was used to execute this query. On the left menu select Jobs. On the Jobs Browser - select the Queries tab to the right of the Job browser header. Hover over & click on the Query just executed (should be the first row). Click on the **Visual Explain** tab.

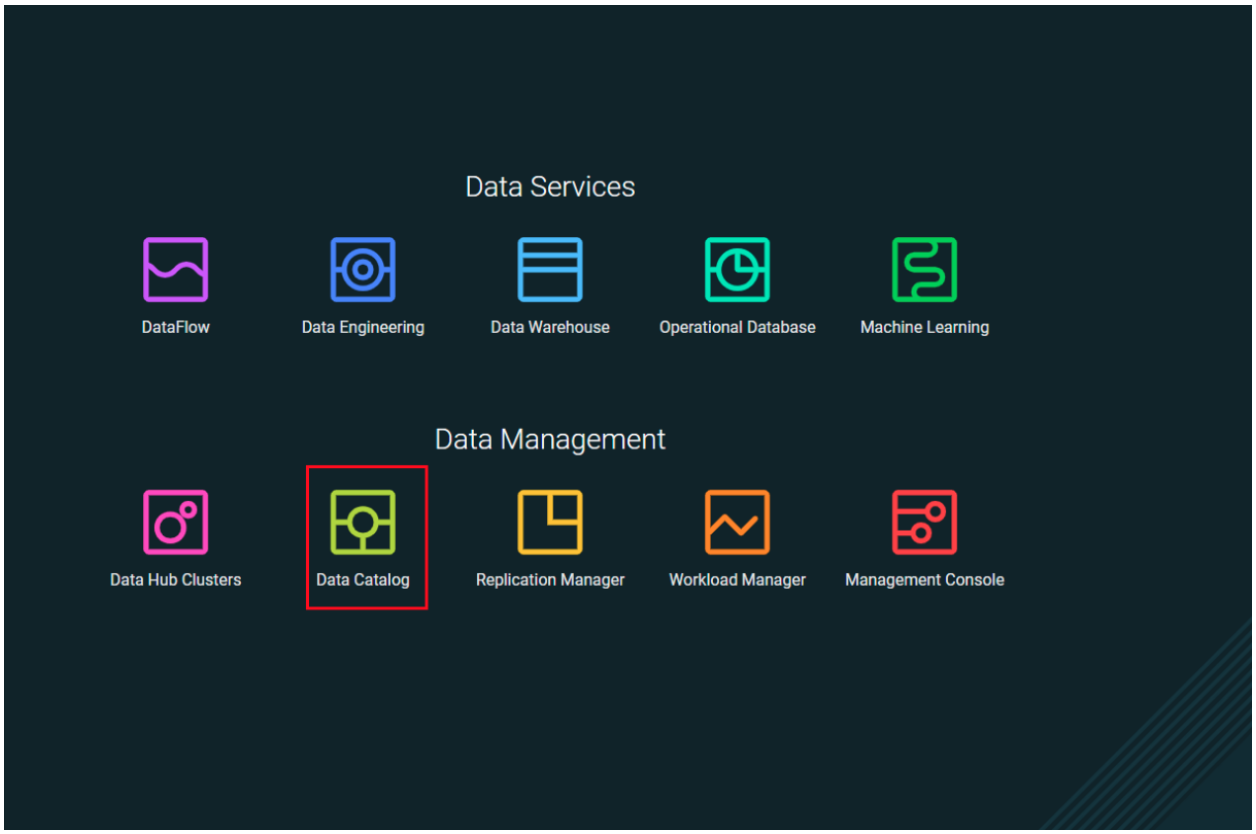


With query rewrite the materialized view is used and the new plan just reads the MV and sorts the data vs. reading flights (86M rows) and airlines (1.5K rows) with hash join, group and sorts. This results in significant reduction in run time for this query.

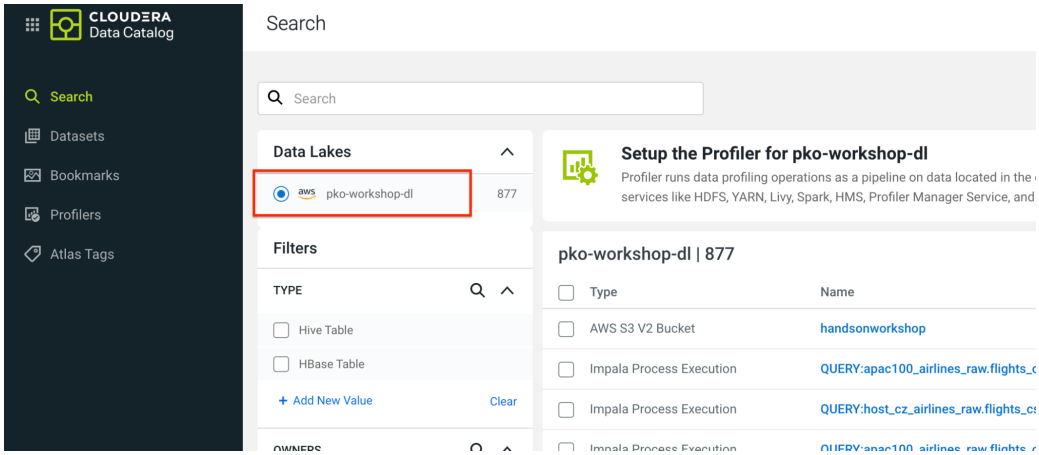
LAB 5 - SECURITY AND GOVERNANCE

In this Lab you will experience the combination of what the Data Warehouse and the Shared Data Experience (SDX) offers. SDX enables you to provide Security and Governance tooling to ensure that you will be able to manage what is in the CDP Platform without having to stitch together multiple tools.

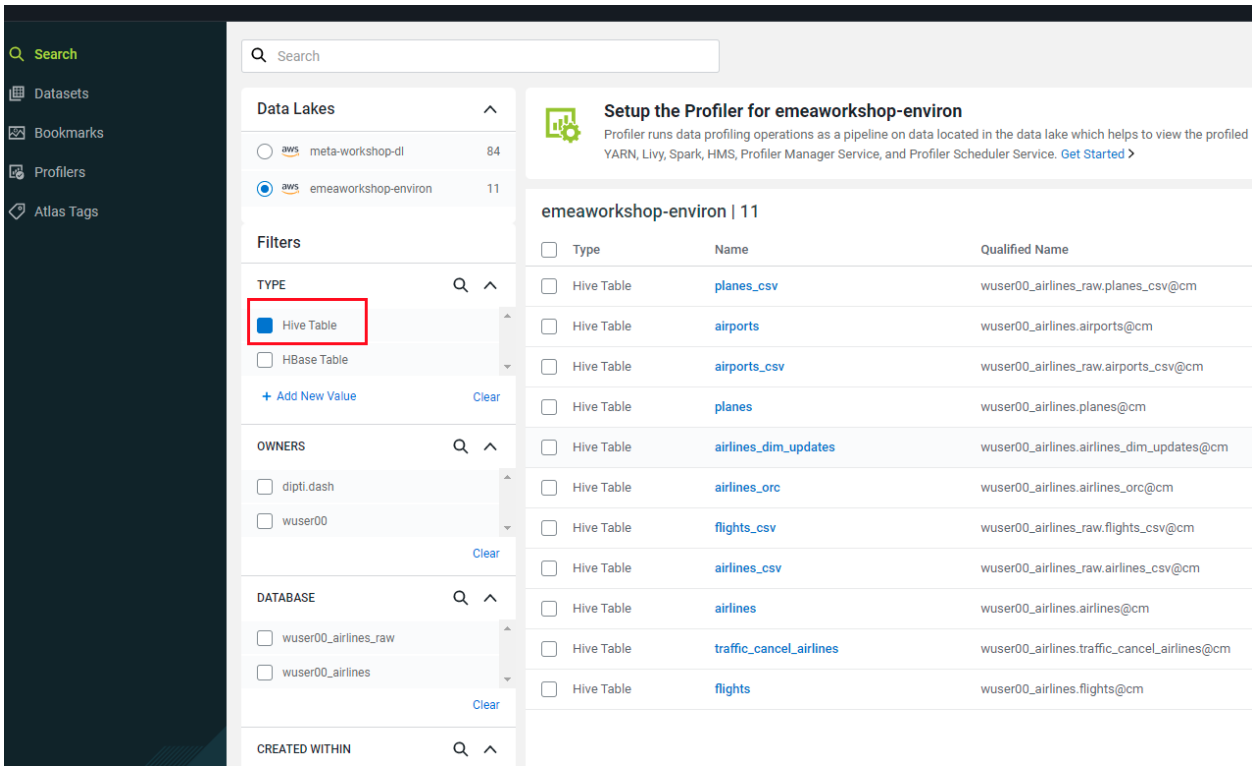
- Go to the Cloudera Data Platform Console and click on Data Catalog



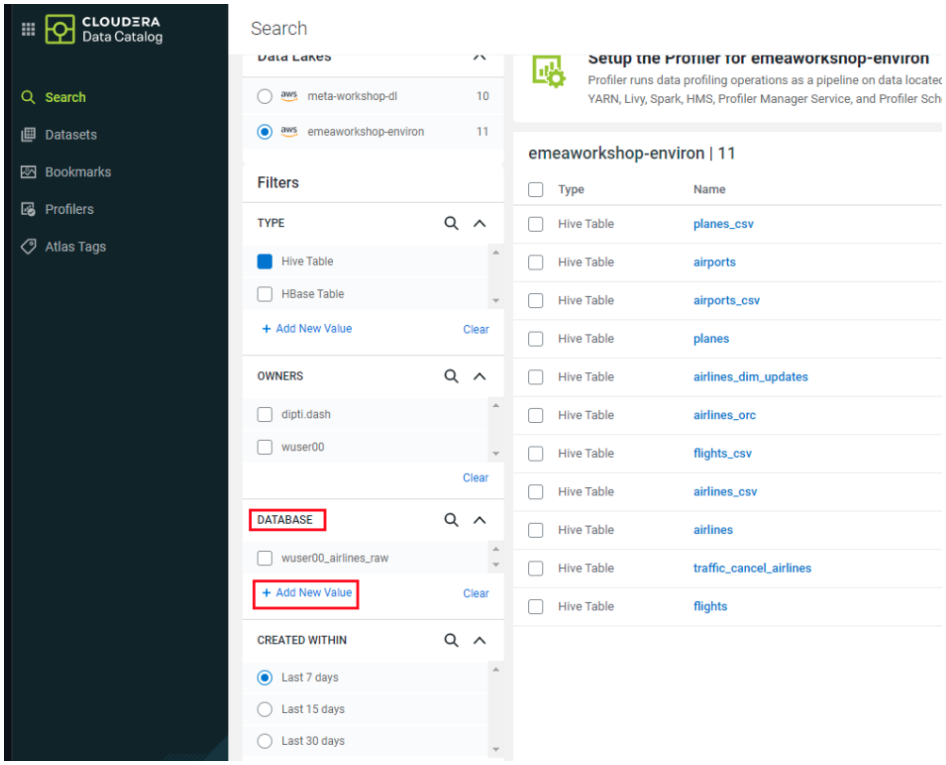
- Change the radio button to select the appropriate data lake. In this case it is <environment-name-shared-by-instructor>

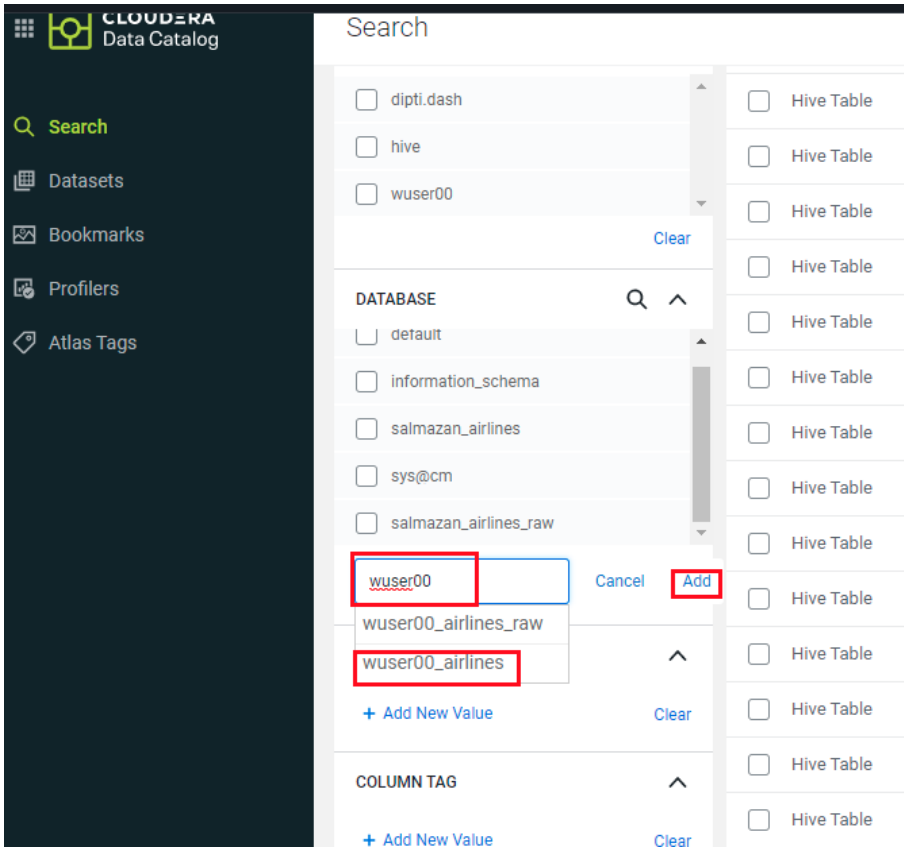


- Filter for Assets we created - below the Data Lakes on the left of the screen under Filters, select TYPE to be Hive Table. The right side of the screen will update to reflect this selection.

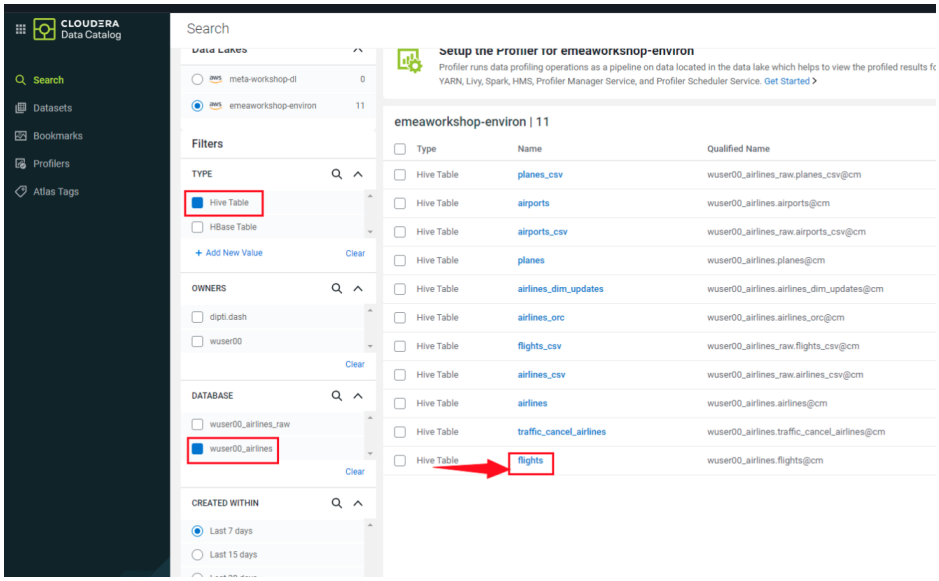


- Under DATABASE, click Add new Value. In the box that appears start typing your <user_id> when you see the <user_id>_airlines database pop up select it.

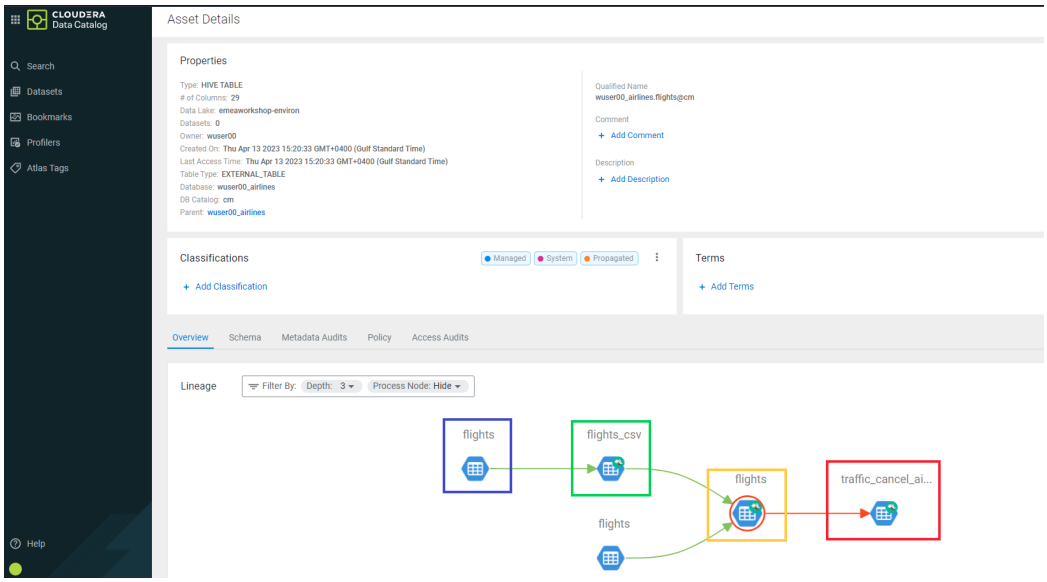

CLOUDERA Data Catalog
Search
Datasets
Bookmarks
Profilers
Atlas Tags
Search
meta-workshop-dl 10
emeaworkshop-environ 11
Filters
TYPE
Hive Table
HBase Table
+ Add New Value
Clear
OWNERS
dipti.dash
wuser00
Clear
DATABASE
wuser00_airlines_raw
+ Add New Value
Clear
CREATED WITHIN
Last 7 days
Last 15 days
Last 30 days
Setup the Profiler for emeaworkshop-environ
Profiler runs data profiling operations as a pipeline on data locate
YARN, Livy, Spark, HMS, Profiler Manager Service, and Profiler Sch
emeaworkshop-environ | 11
Type
Name
Hive Table planes_csv
Hive Table airports
Hive Table airports_csv
Hive Table planes
Hive Table airlines_dim_updates
Hive Table airlines_orc
Hive Table flights_csv
Hive Table airlines_csv
Hive Table airlines
Hive Table traffic_cancel_airlines
Hive Table flights


CLOUDERA Data Catalog
Search
Datasets
Bookmarks
Profilers
Atlas Tags
Search
dipti.dash
hive
wuser00
Clear
DATABASE
default
information_schema
salmazan_airlines
sys@cm
salmazan_airlines_raw
wuser00
Cancel
Add
wuser00_airlines_raw
wuser00_airlines
+ Add New Value
Clear
COLUMN TAG
+ Add New Value
Clear
Hive Table

- You should now see the tables and materialized views that have been created in the <user_id>_airlines database. Click on flights in the Name column to view more details on the table.

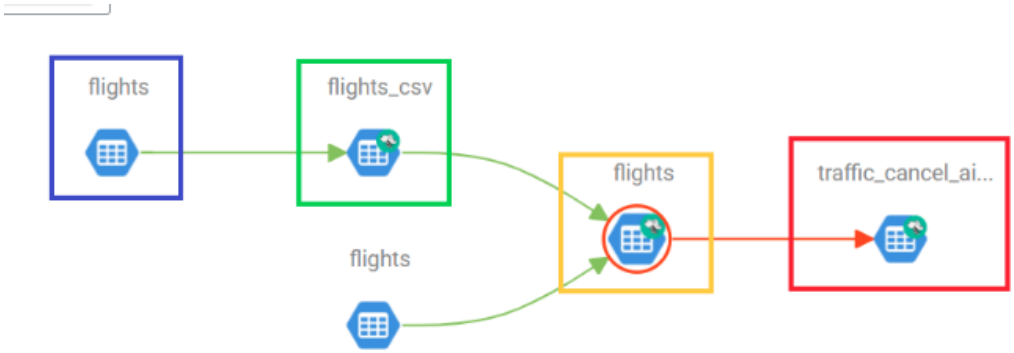


- This page shows information about the flights table such as the table owner, when the table was created, when it was last accessed, and other properties. Below the summary details is the Overview tab which shows the lineage - hover over the flights click on the "i" icon that appears to see more detail on this table.

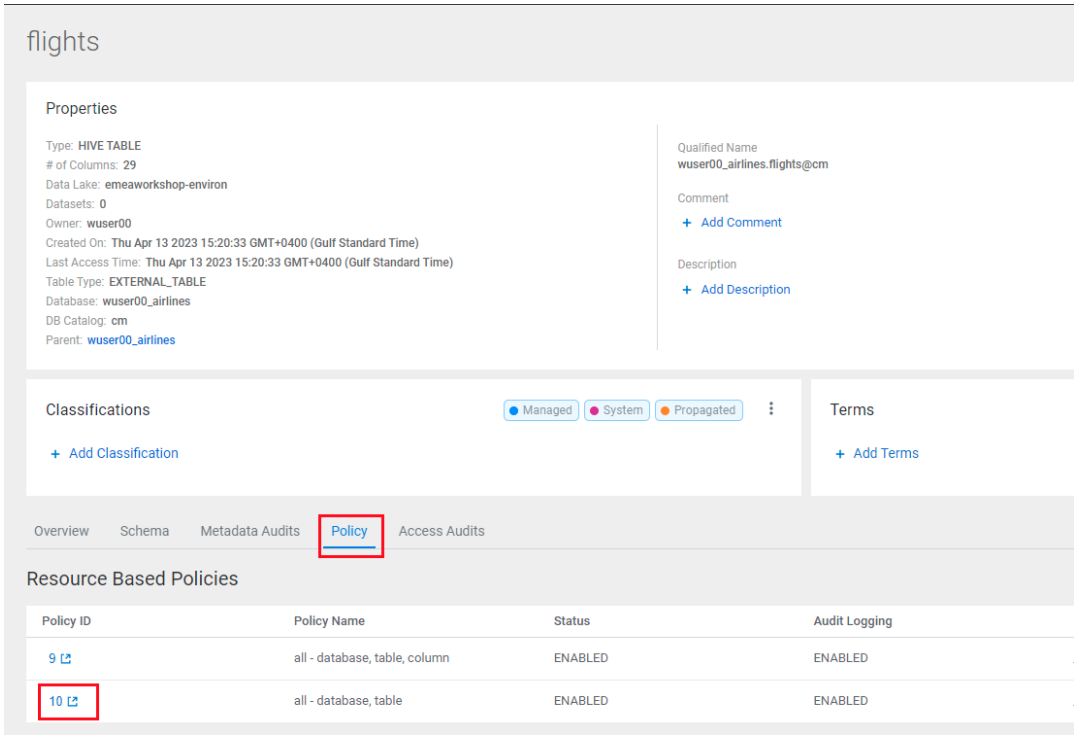


- The lineage shows:
  - [blue box] - flights data file residing in an s3 folder.
  - [green box] - is showing how the flights_csv Hive table is created, this table was created and points to the data location of flights (blue box) s3 folder.
  - [orange box]- is showing the flights Iceberg table and how it is created, it uses data from flights_csv Hive table (CTAS).

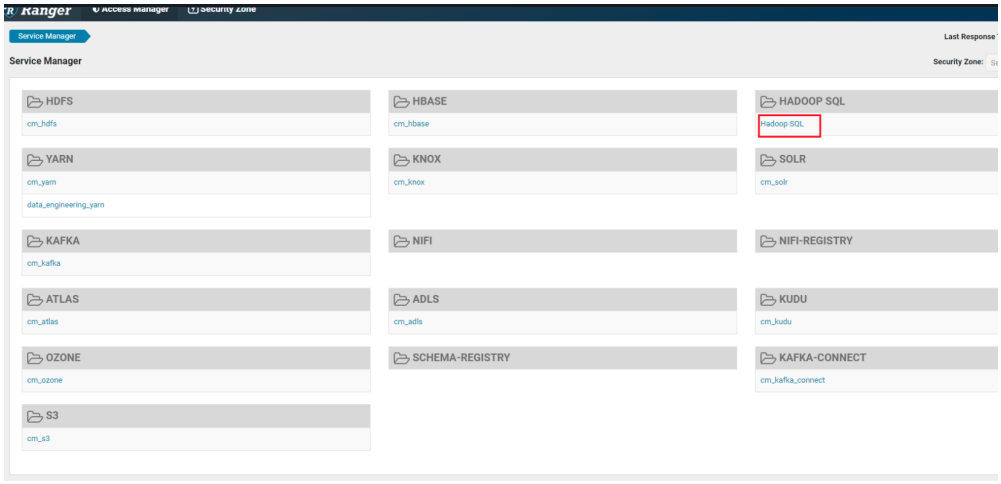
- [red box] - traffic_cancel_airlines is a Materialized View that uses data from the flights Iceberg table.



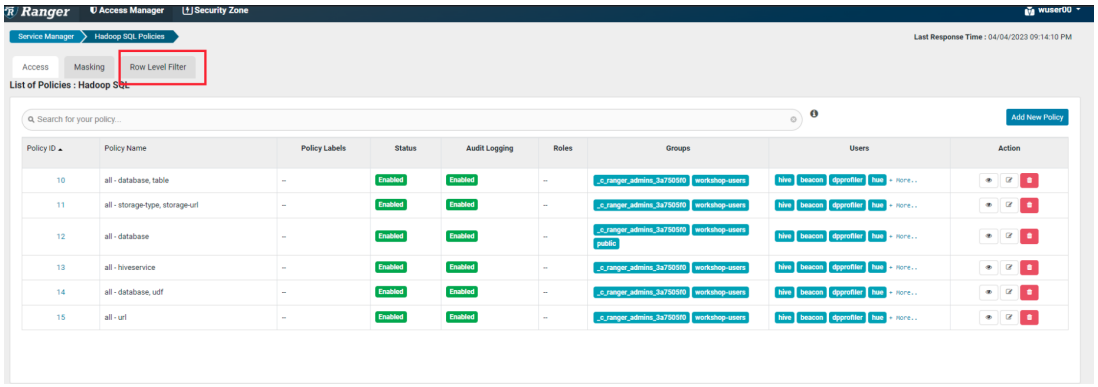
- Click on the Policy tab to see what security policies have been applied on this table. Click on the arrow next all - database, table Policy Name to the number as shown in the screenshot



- It will open Ranger which is for access management. Using Security (Ranger) - we can modify and create security policies for the various CDP Data Services. Click on Hadoop SQL link in the upper right corner - to view the security policies in place for CDW. Here, I will stick to the CDW related security features.

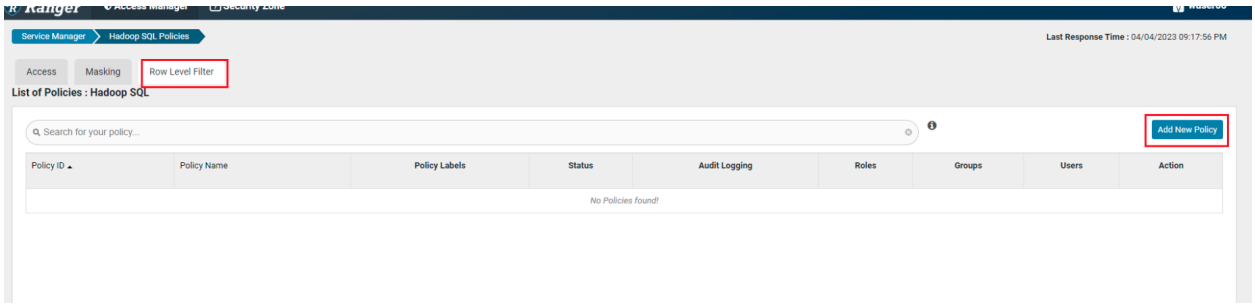


- This screen shows the general Access related security policies - who has access to which Data Lakehouse databases, tables, views, etc. Click on the Row Level Filter tab to see the policies to restrict access to portions of data.



Step 1 : Add New Policy

- There are currently no policies defined. Click on the Add New Policy button on the top right corner.

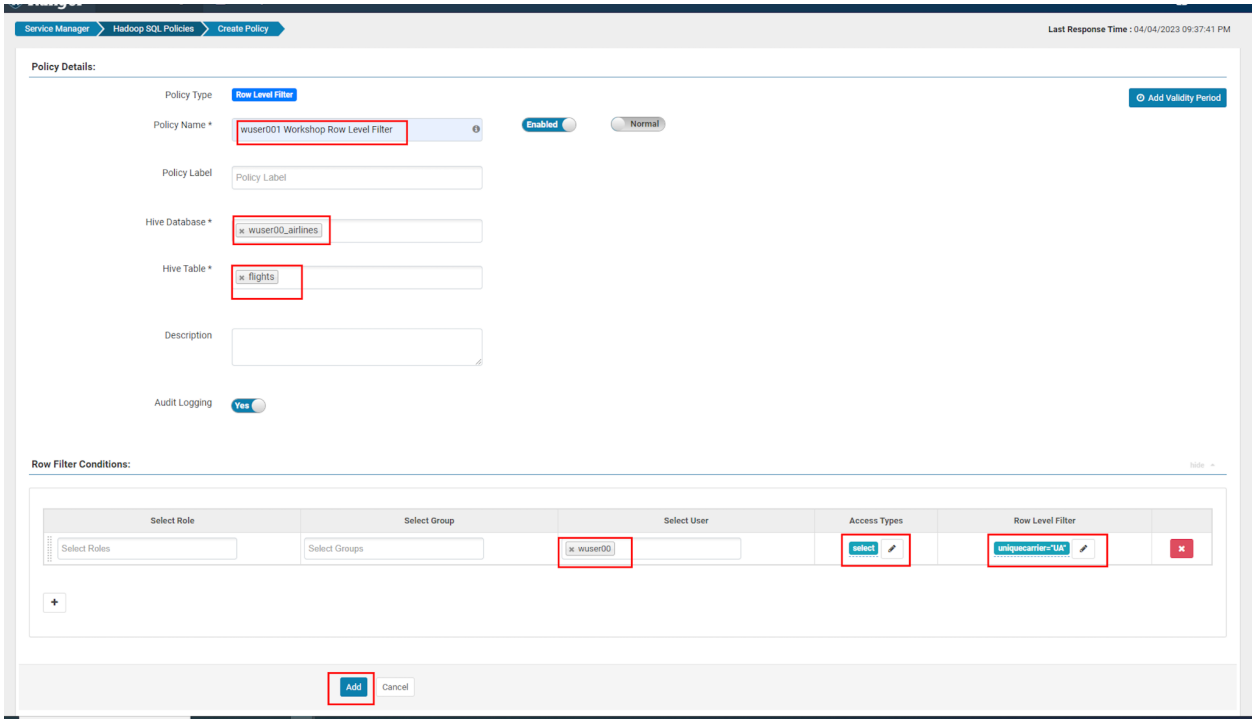


- Fill out the form as follows.
  Policy Name: <user_id>_RowLevelFilter (Ex: wuser00_RowLevelFilter)
  Hive Database: <user_id>_airlines (Ex: wuser00_airlines)
  Hive Table: flights (start typing, once you see this table in the list, select it)

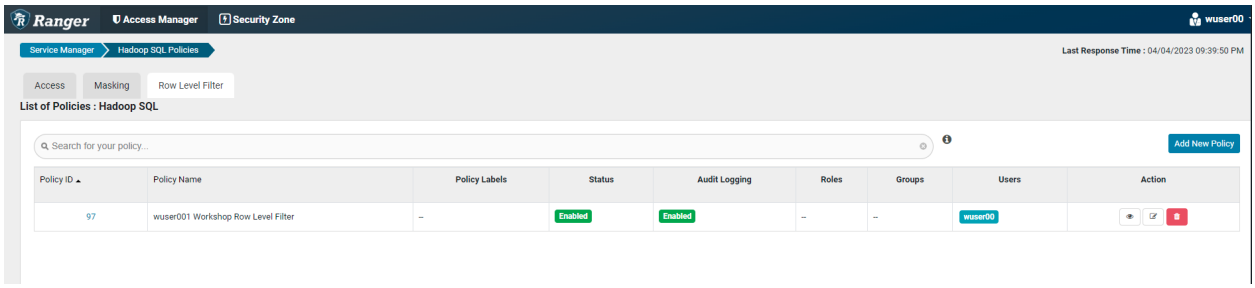
Row Filtering Conditions:

- Select User: <user_id>
- Access Types: select
- Row Level Filter: uniquecarrier="UA"

Click the Add button to accept this Policy.



The new policy is added to the Row Level Filter policies (as below).



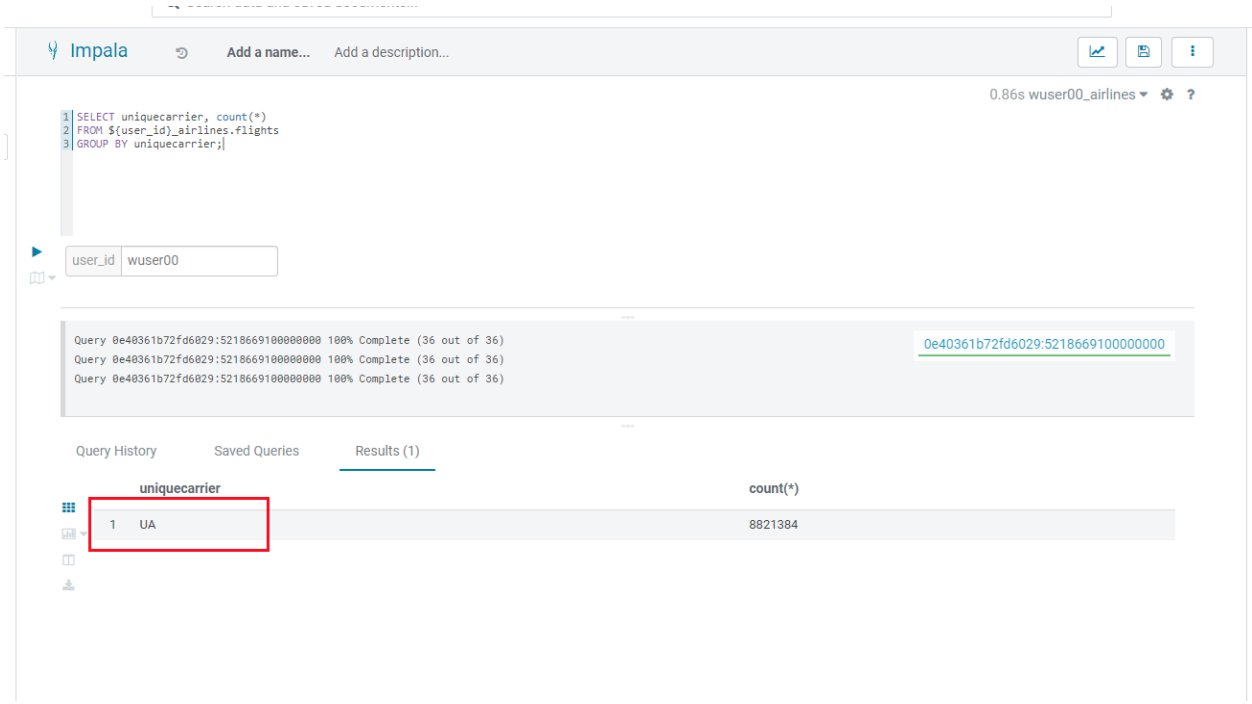
Step 2 : Test New Policy

- Test the policy is working - Open HUE for the CDW Impala Virtual Warehouse assigned to you and execute the following query.

```
Unset
SELECT uniquecarrier, count(*)
```

```
FROM ${user_id}_airlines.flights
GROUP BY uniquecarrier;
```

You should now only see 1 row returned for this query - after the policy was applied you will only be able to access uniquecarrier = UA and no other carriers.

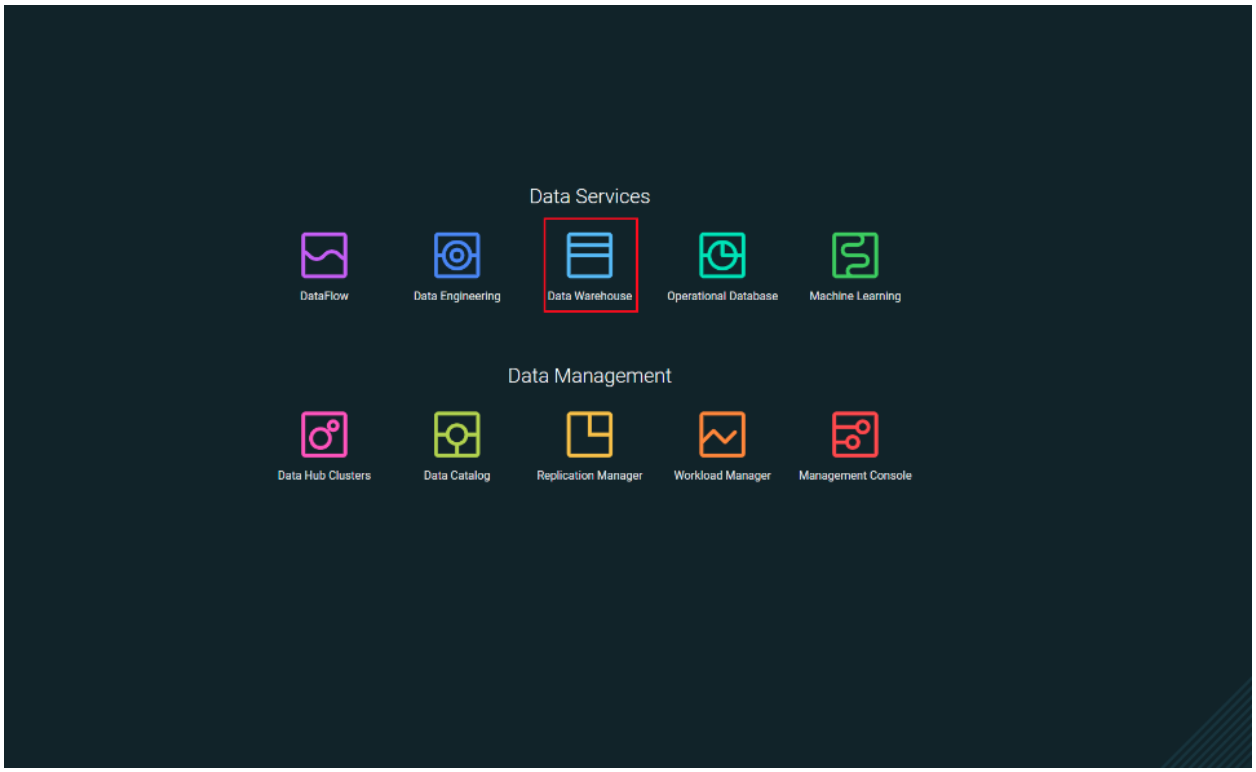


LAB 6 - CLOUDERA DATA VISUALIZATION

In this step you will build a Logistics Dashboard using Cloudera Data Visualization. The Dashboard will include details about flight delays and cancellations. But first we will start with Data Modeling.

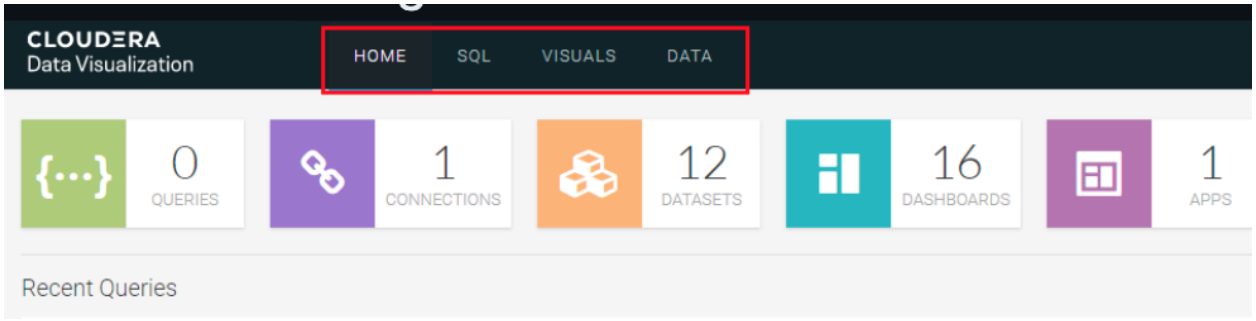
Data Modeling

- If you are not on the CDP home page, then go there and click on the following CDW icon to go into Cloudera Data Warehouse.

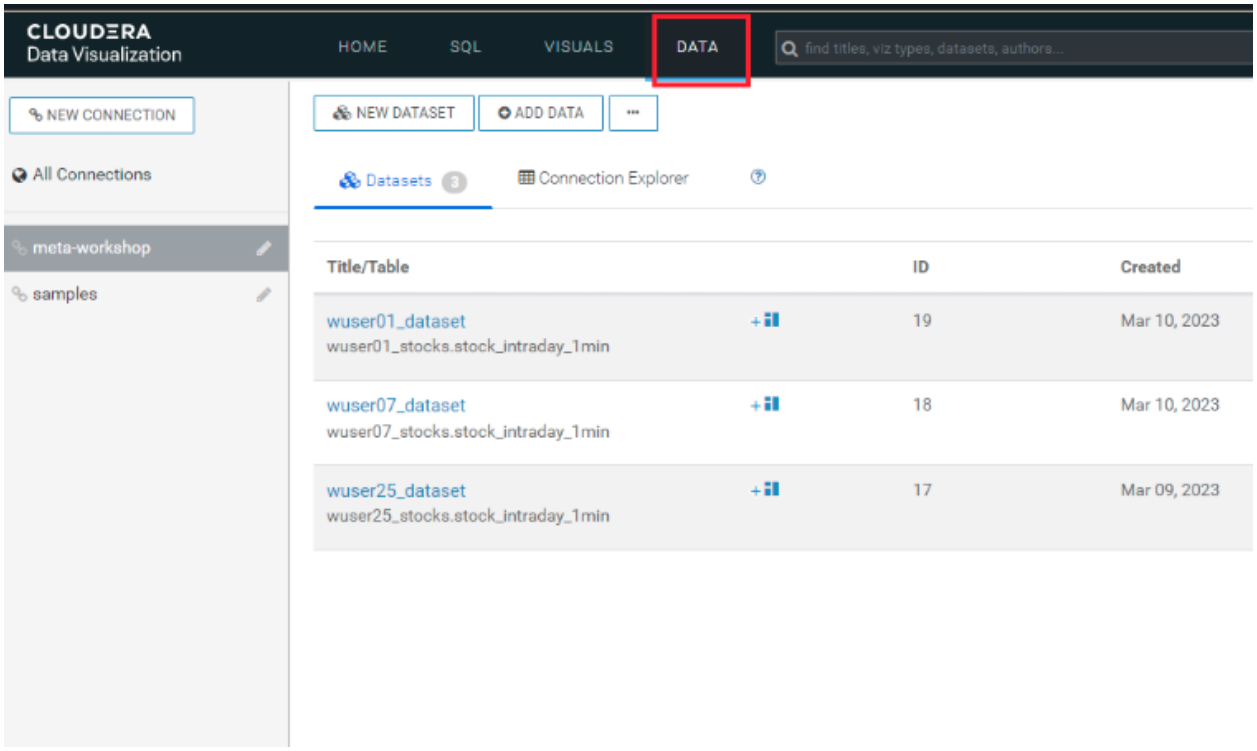


- Click on the Data Visualization option in the left windowpane. You'll see an option Data VIZ next to the data-viz application with the name **hol-data-viz**. It should open a new window.

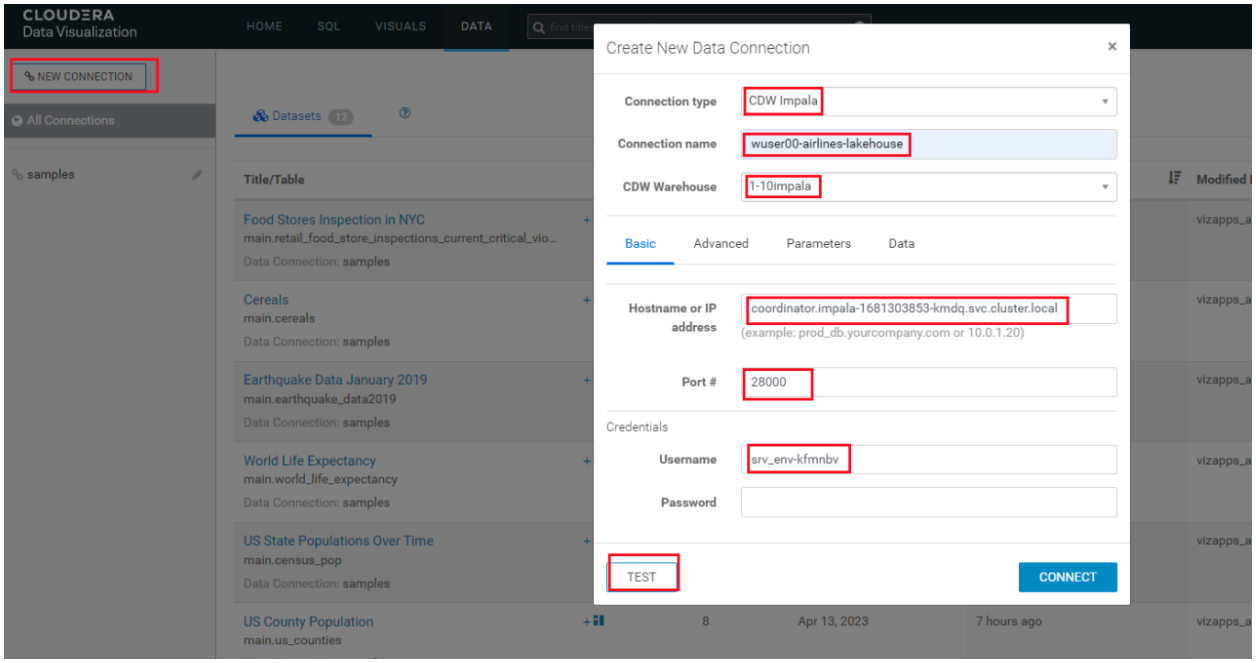
- There are 4 areas of CDV - HOME, SQL, VISUALS, DATA - these are the tabs at the top of the screen in the black bar to the right of the Cloudera Data Visualization banner.

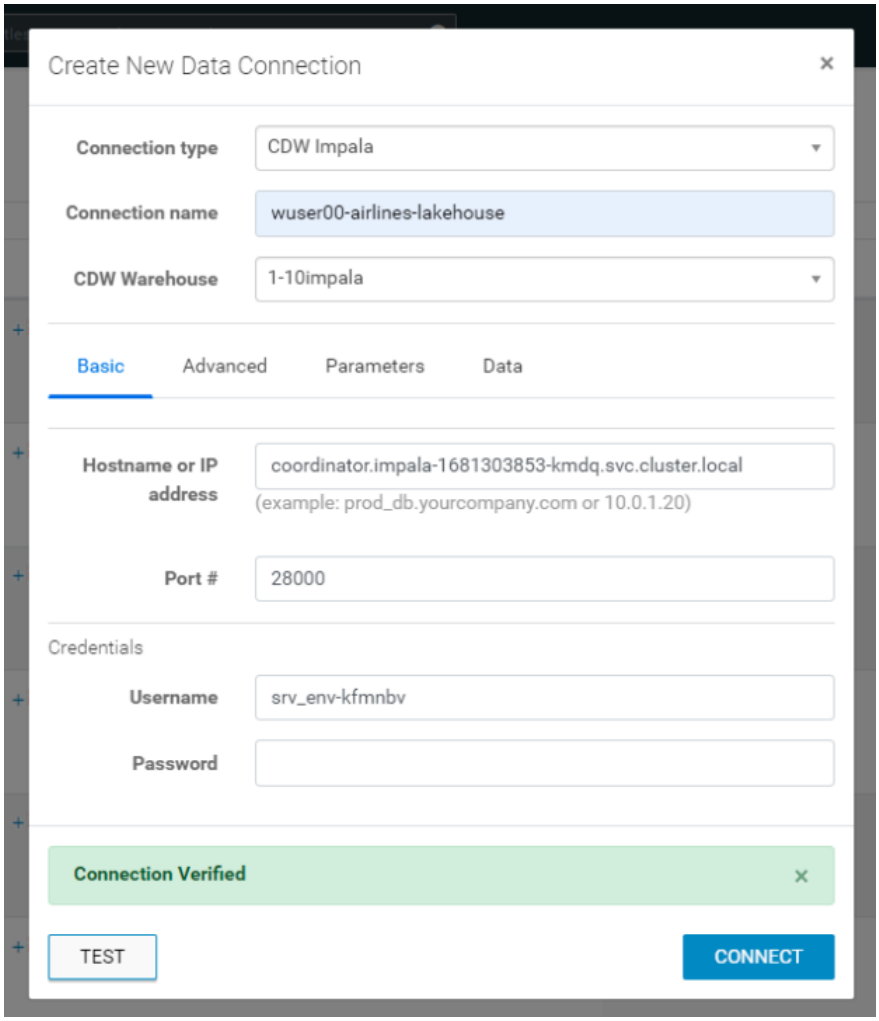


- Click on DATA in the top banner. A Dataset is a Semantic Layer where you can create a business view on top of the data - data is not copied; this is just a logical layer.

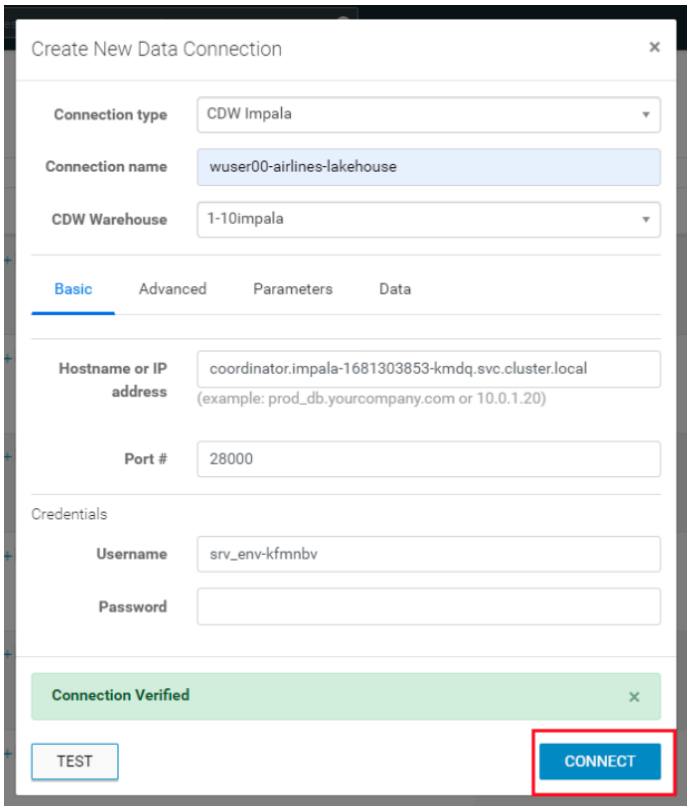


- Create a connection - click on the NEW CONNECTION button on the left menu. Enter the details as shown in the screenshot and click on TEST.
  Connection type: Select CDW Impala.
  Connection name: <user_id>-airlines-lakehouse
  (Ex-wuser00-airlines-lakehouse).
  CDW Warehouse: Make Sure you select the warehouse that is associated with your <user_id>.
  Hostname or IP address: Gets automatically selected.
  Port: Gets automatically filled up.
  Username: Gets automatically filled up.
  Password: Blank

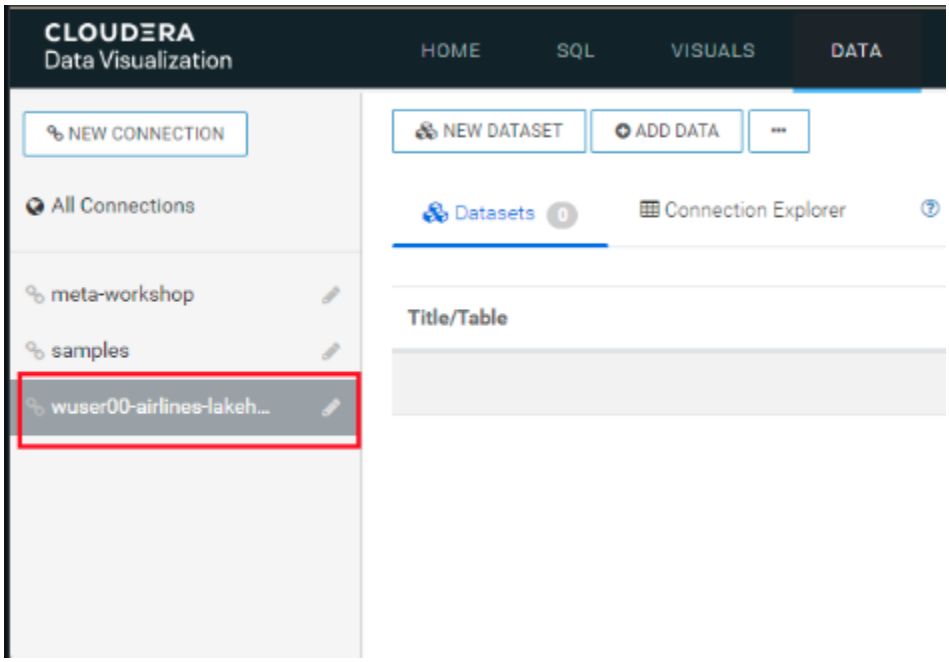

CLOUDERA
Data Visualization
HOME
SQL
VISUALS
DATA
NEW CONNECTION
All Connections
samples
Datasets 12
Title/Table
Food Stores Inspection in NYC
main.retail_food_store_inspections_current_critical_vio...
Data Connection: samples
Cereals
main.cereals
Data Connection: samples
Earthquake Data January 2019
main.earthquake_data2019
Data Connection: samples
World Life Expectancy
main.world_life_expectancy
Data Connection: samples
US State Populations Over Time
main.census_pop
Data Connection: samples
US County Population
main.us_counties
8
Apr 13, 2023
7 hours ago
Create New Data Connection
Connection type
CDW Impala
Connection name
wuser00-airlines-lakehouse
CDW Warehouse
1-10impala
Basic
Advanced
Parameters
Data
Hostname or IP address
coordinator.impala-1681303853-kmdq.svc.cluster.local
(example: prod_db.yourcompany.com or 10.0.1.20)
Port #
28000
Credentials
Username
srv_env-kfmnbv
Password
TEST
CONNECT

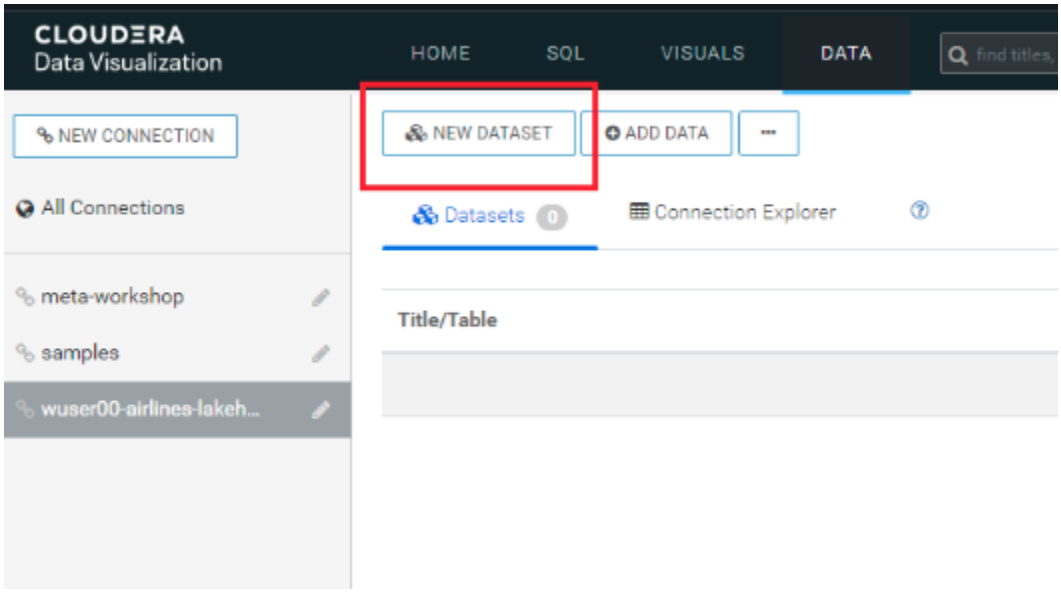

Create New Data Connection
Connection type
CDW Impala
Connection name
wuser00-airlines-lakehouse
CDW Warehouse
1-10impala
Basic
Advanced
Parameters
Data
Hostname or IP address
coordinator.impala-1681303853-kmdq.svc.cluster.local
(example: prod_db.yourcompany.com or 10.0.1.20)
Port #
28000
Credentials
Username
srv_env-kfmnbv
Password
Connection Verified
TEST
CONNECT

- Click on CONNECT.

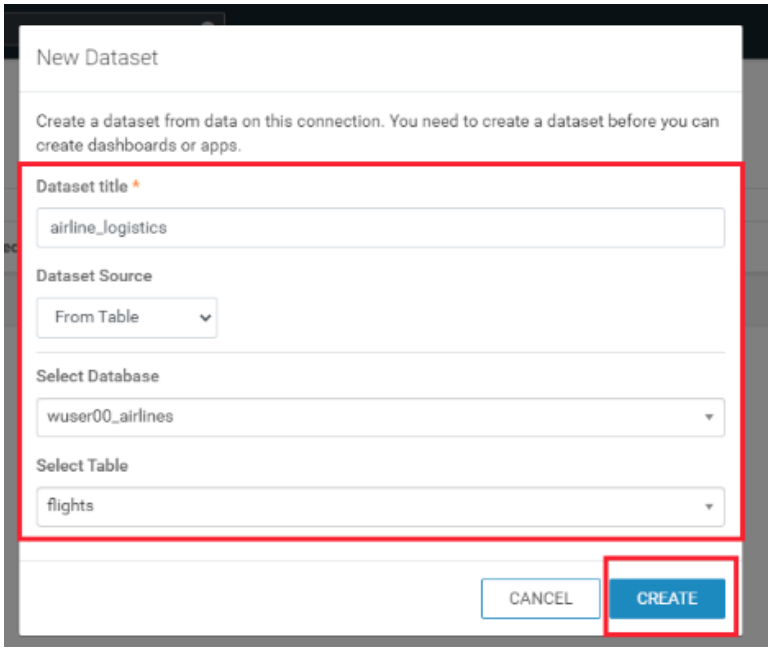


- You will see your connection in the list of connections on the left menu. On the right side of the screen you will see Datasets and the Connection Explorer. Click on NEW DATASET.

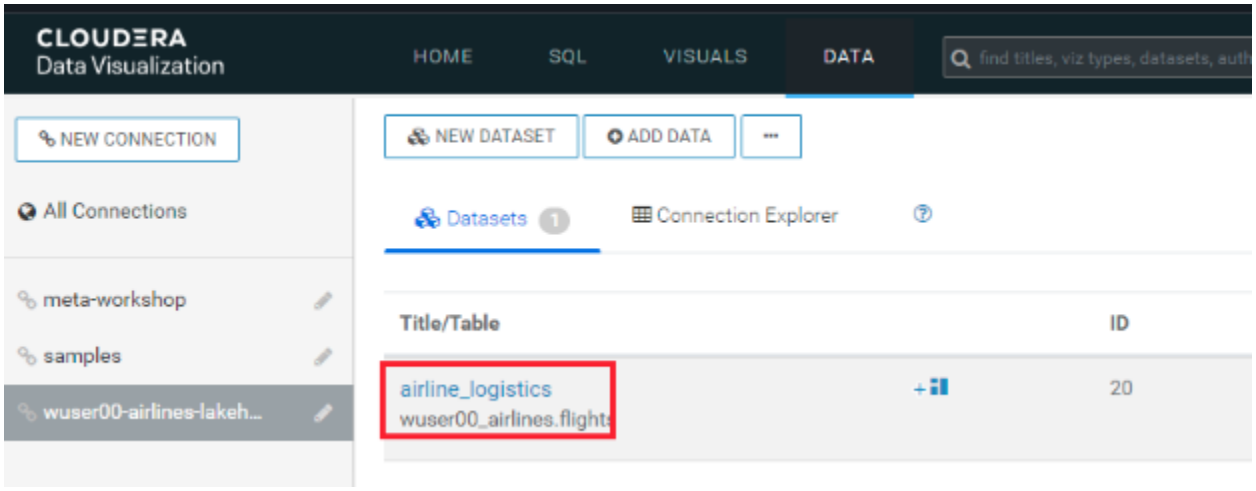




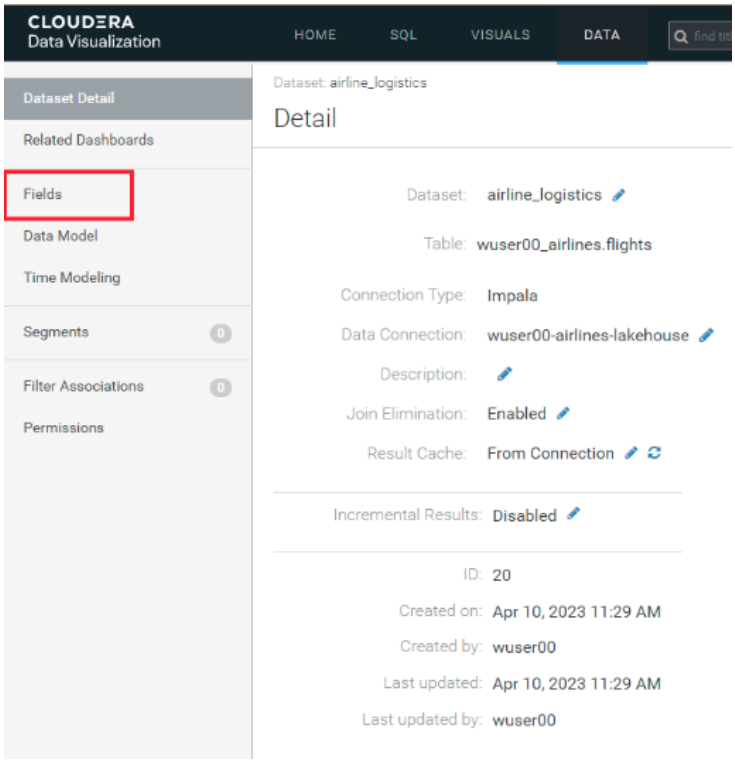
- Fill the details as following and click CREATE. airline_logistics gets created
  Dataset title - airline_logistics.
  Dataset Source - Select From Table (however, you could choose to directly enter a SQL statement instead).
  Select Database - <user_id>_airlines (Make Sure you select the database that is associated with your <user_id>).
  Select Table - flights.

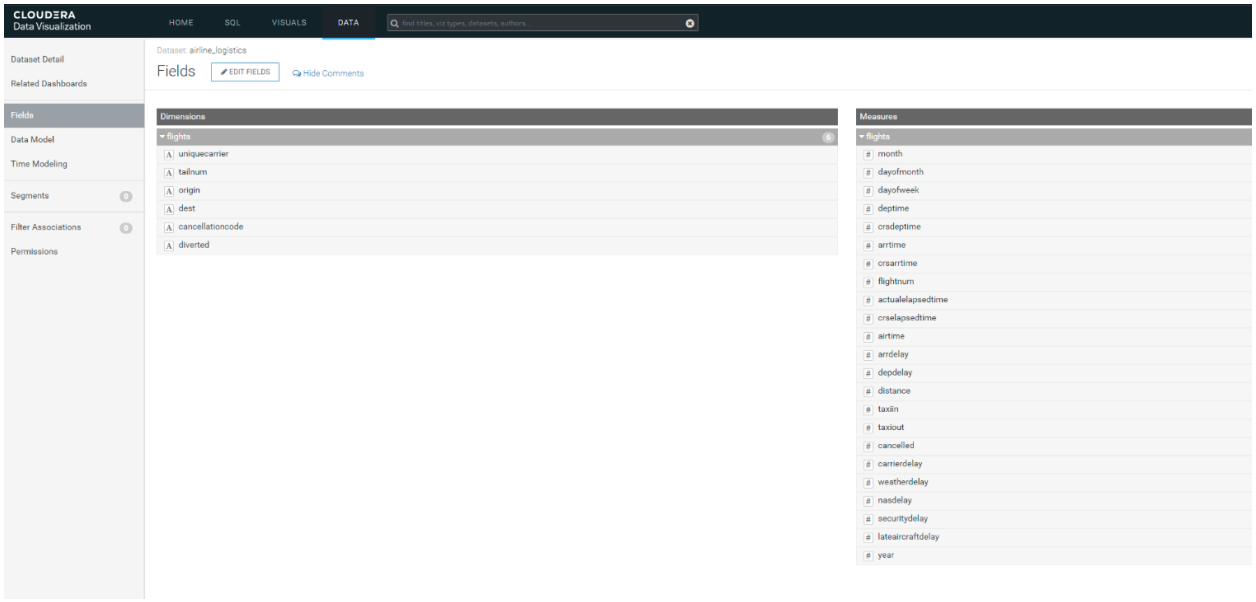


Edit the Dataset - click on airline_logistics on the right of the screen. This will open the details page, showing you information about the Dataset, such as connection details, and options that are set.

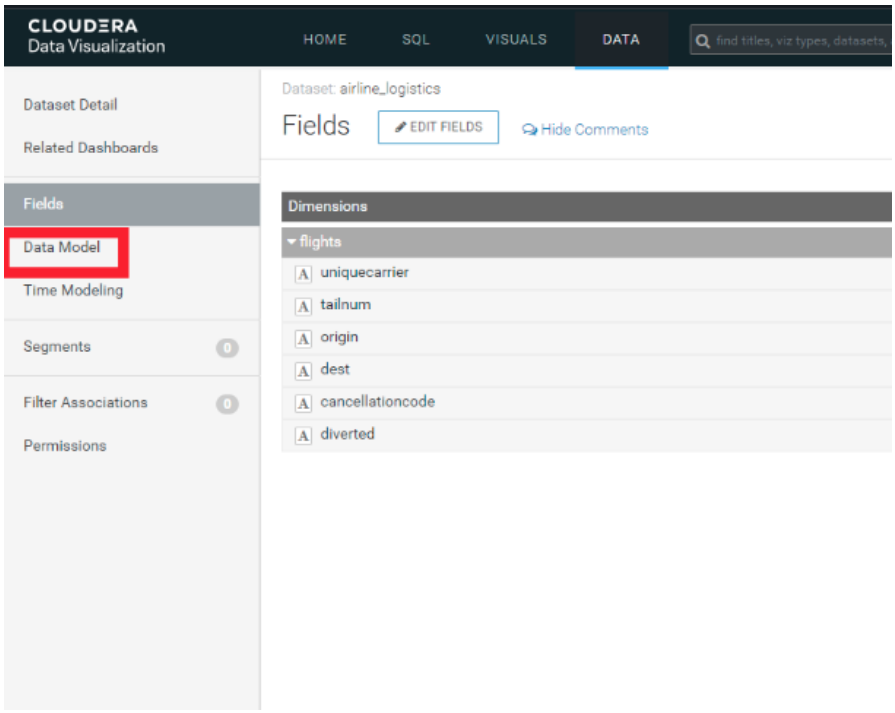


Click on Fields option in the left window pane.

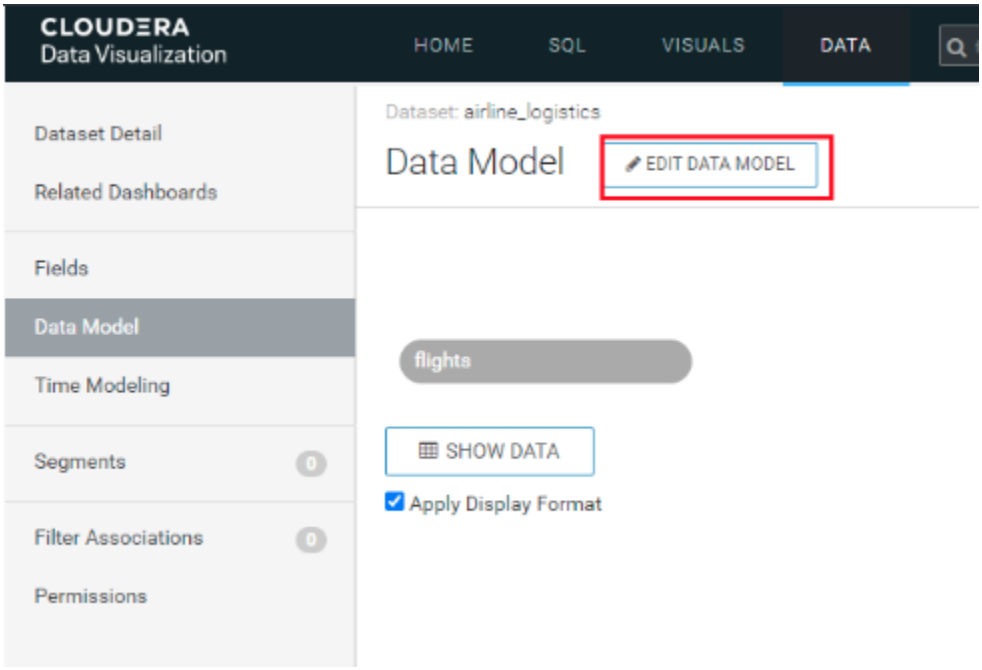




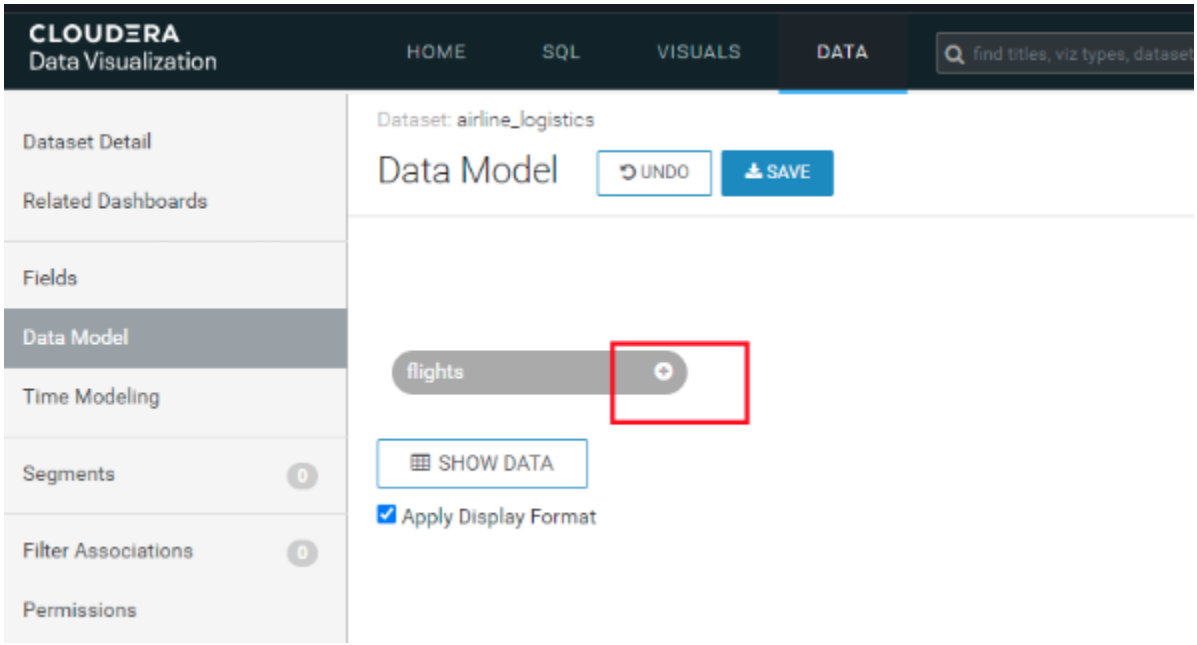
Click on Data Model - for our Dataset we need to join additional data to the flights table including the planes, airlines, and airports tables.



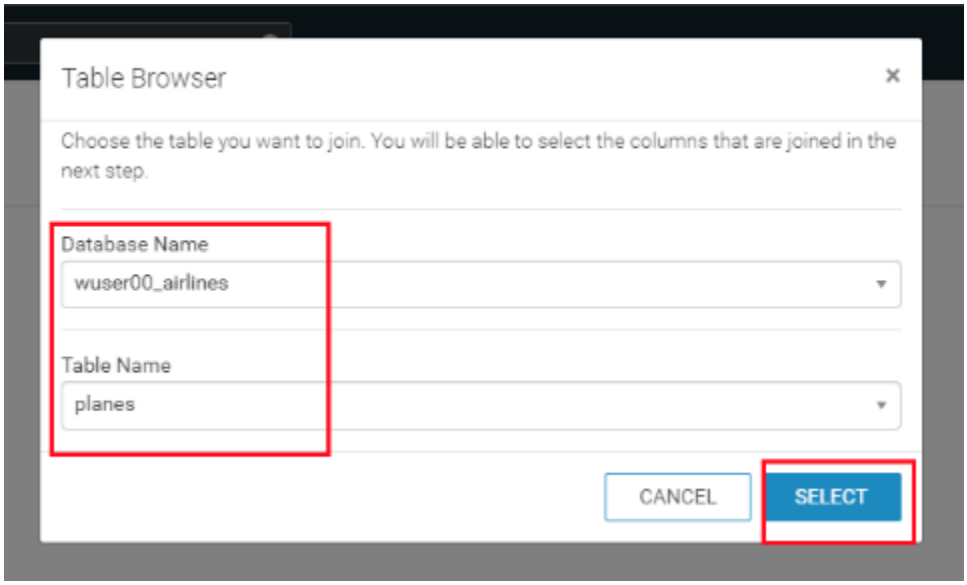
Click on EDIT DATA MODEL.



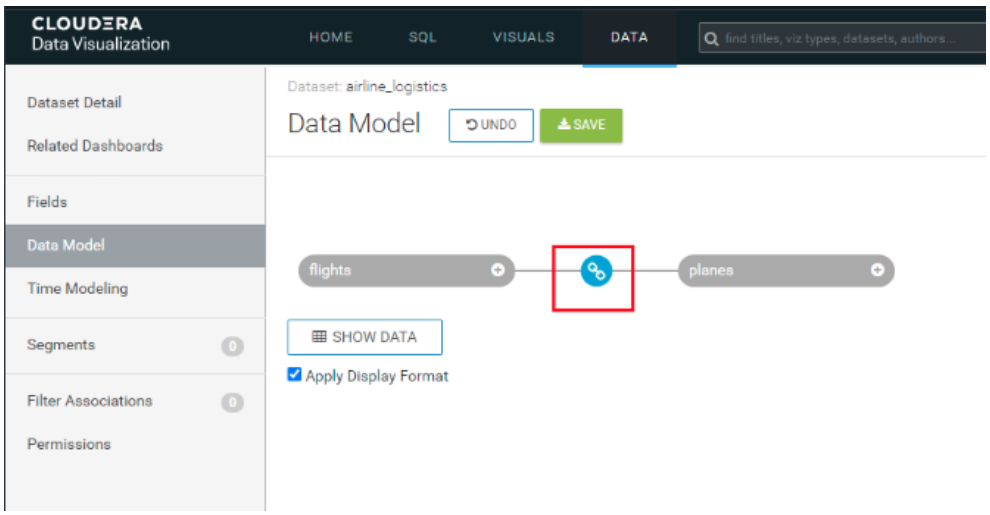
Click on the + icon next to the flights table option.

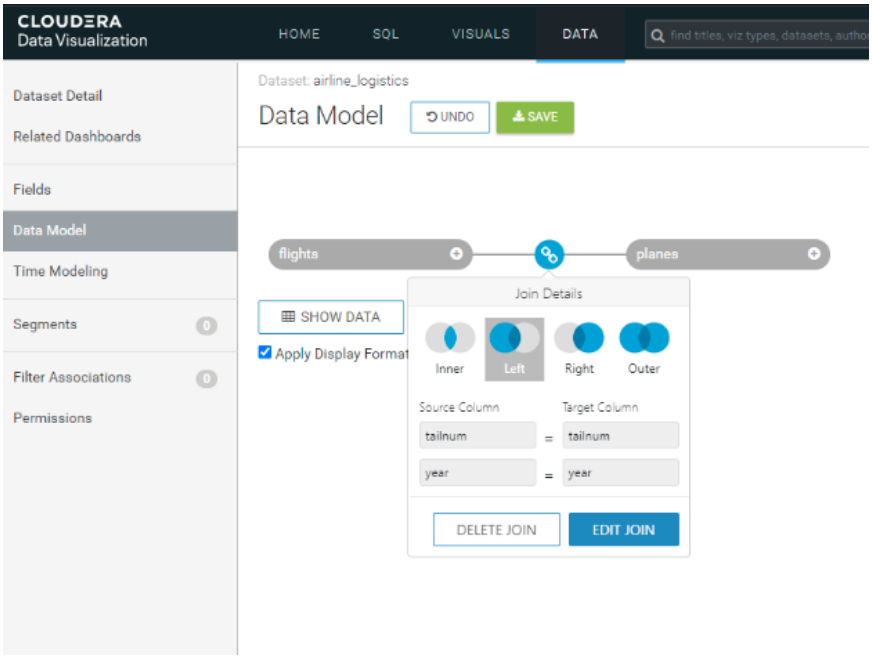


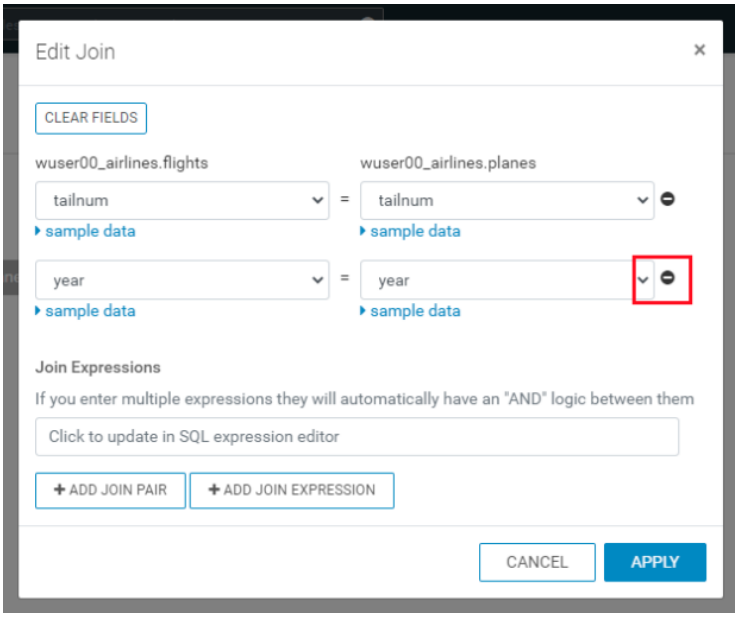
Select the appropriate Database Name base on your user id (Ex: wuser00_airlines) and table name planes.

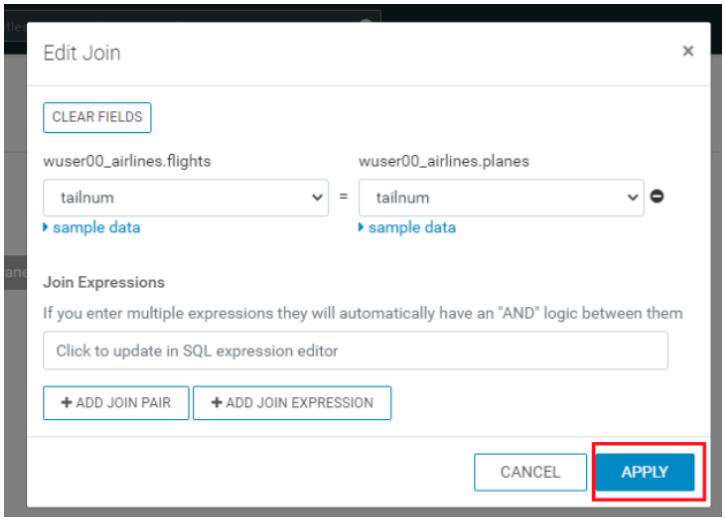


Then click on the joinicon join icon and see that there are 2 join options tailnum & year. Click on EDIT JOIN and then remove the year join by clicking the little - (minus) icon to the right next to the year column. Click on APPLY.

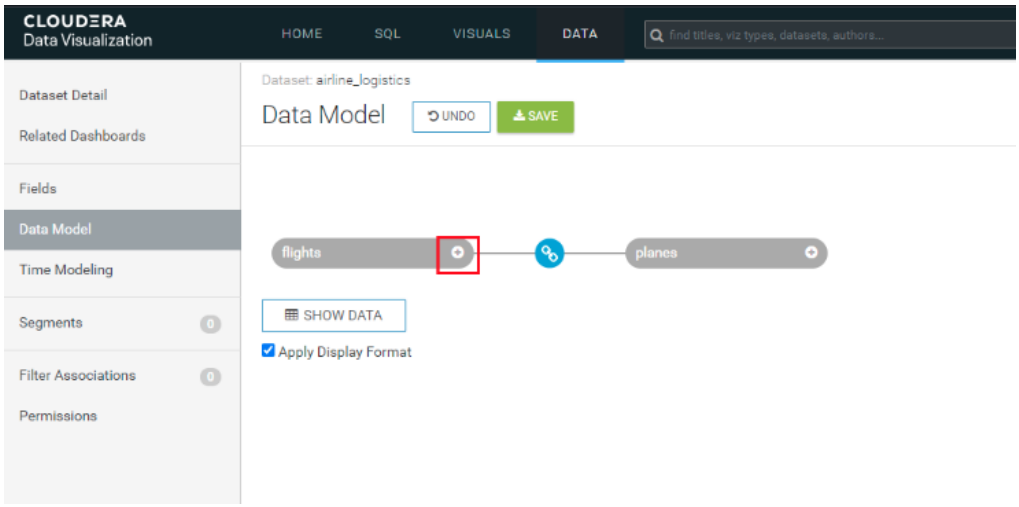


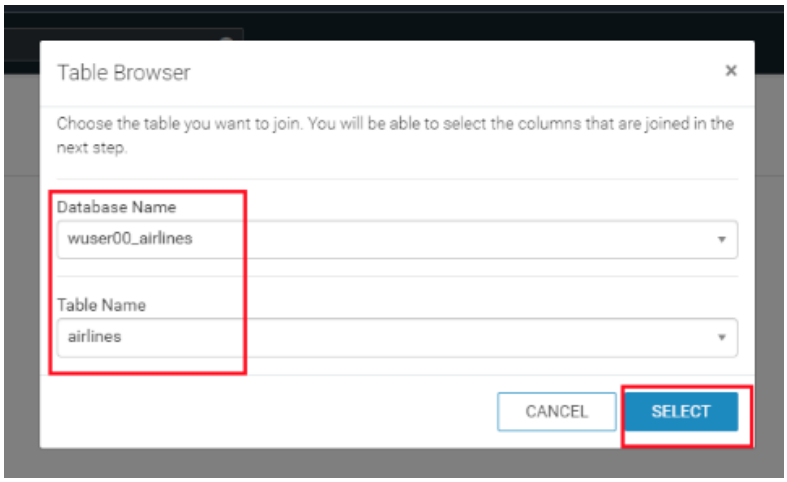

CLOUDERA
Data Visualization
HOME
SQL
VISUALS
DATA
Dataset Detail
Related Dashboards
Fields
Data Model
Time Modeling
Segments
Filter Associations
Permissions
Dataset: airline_logistics
Data Model
UNDO
SAVE
flights
planes
SHOW DATA
Apply Display Format
Join Details
Inner
Left
Right
Outer
Source Column
Target Column
tailnum
tailnum
year
year
DELETE JOIN
EDIT JOIN


Edit Join
CLEAR FIELDS
wuser00_airlines.flights
wuser00_airlines.planes
tailnum
tailnum
sample data
sample data
year
year
sample data
sample data
Join Expressions
If you enter multiple expressions they will automatically have an "AND" logic between them
Click to update in SQL expression editor
+ ADD JOIN PAIR
+ ADD JOIN EXPRESSION
CANCEL
APPLY

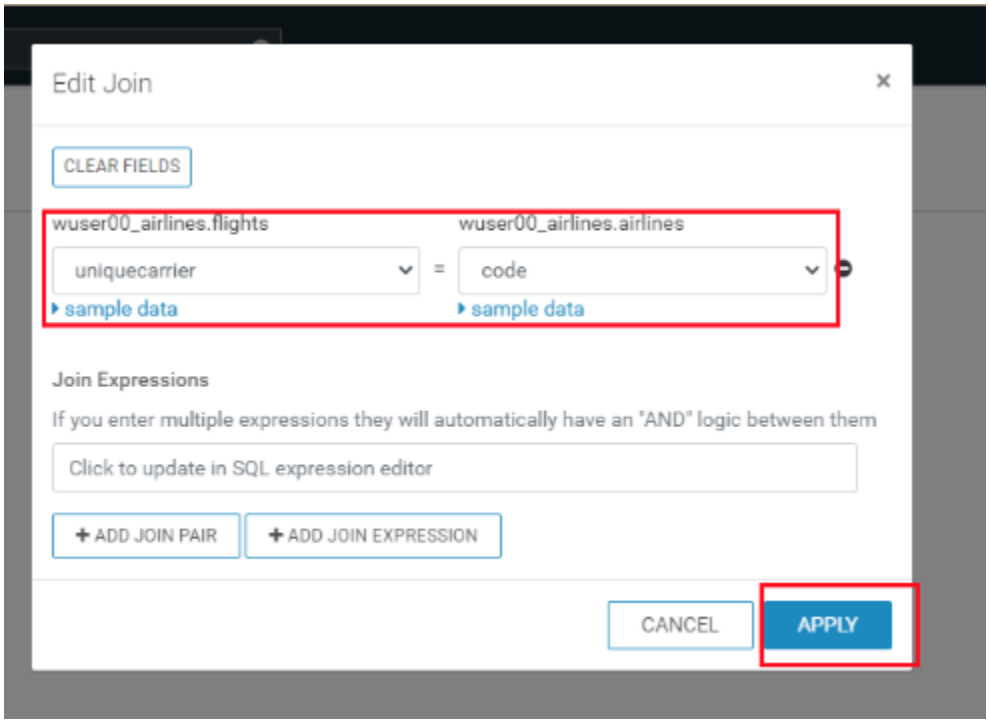


- Now we will create a join between another table. Click on + icon next to flights as shown below. Select the appropriate Database Name based on your <user_id> (Ex: wuser00_airlines) and table name airlines.

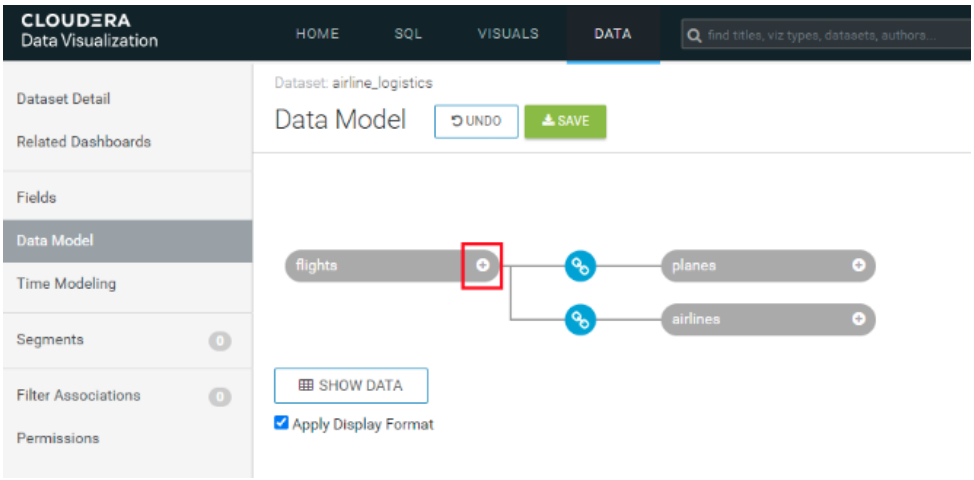


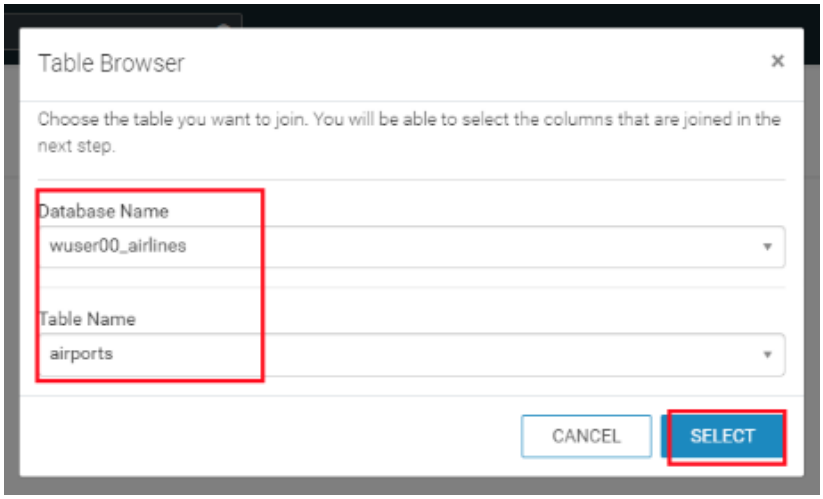


- Make sure you select the column uniquecarrier from flights and column code from airlines table. Click APPLY.

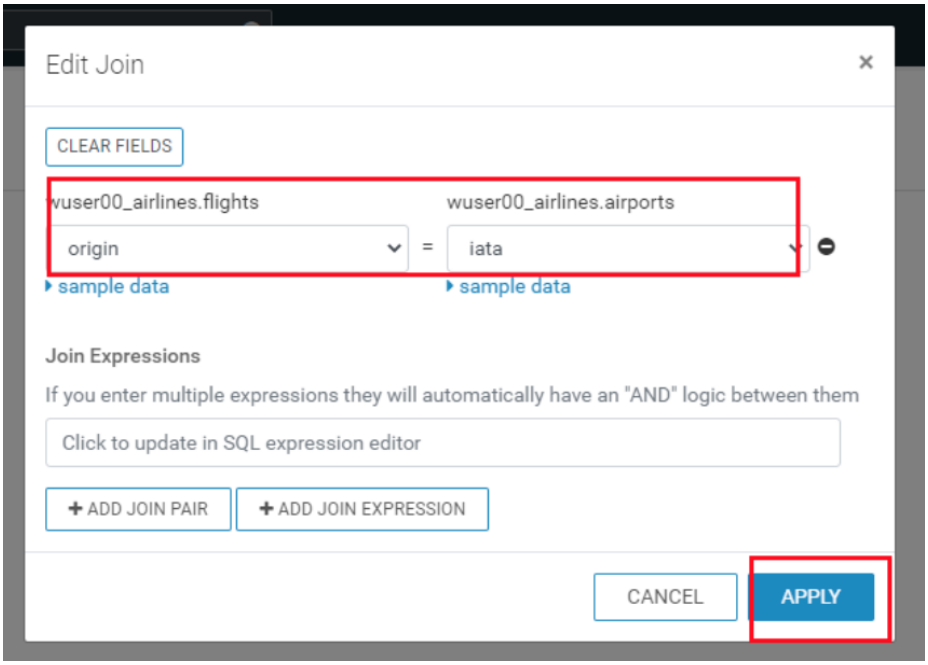


- Click on + icon next to flights as shown below. Select the appropriate Database Name based on your <user_id> (Ex: wuser00_airlines) and table name airports.

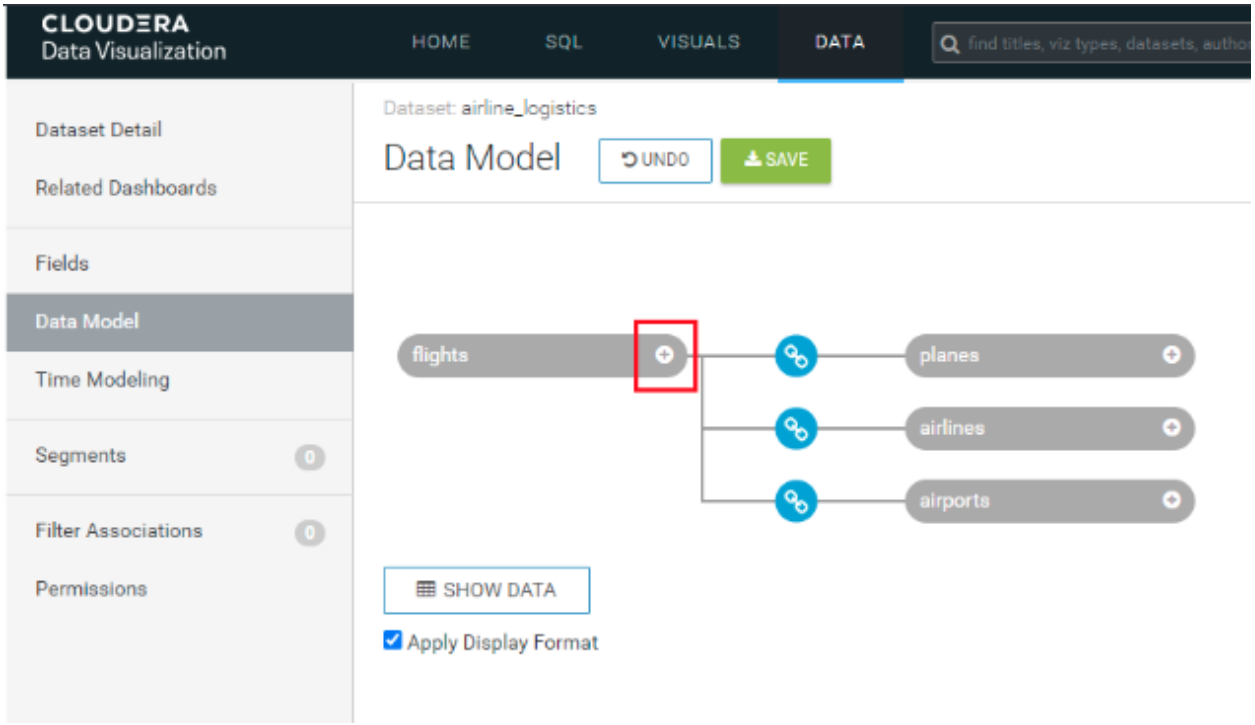




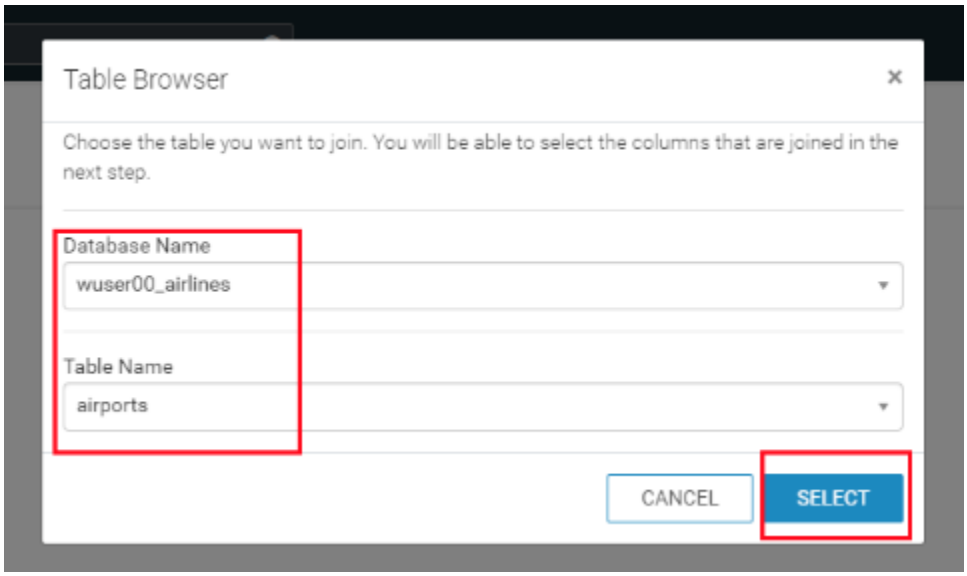
- Make sure you select the column origin from flights and column iata from airports table. Click APPLY.



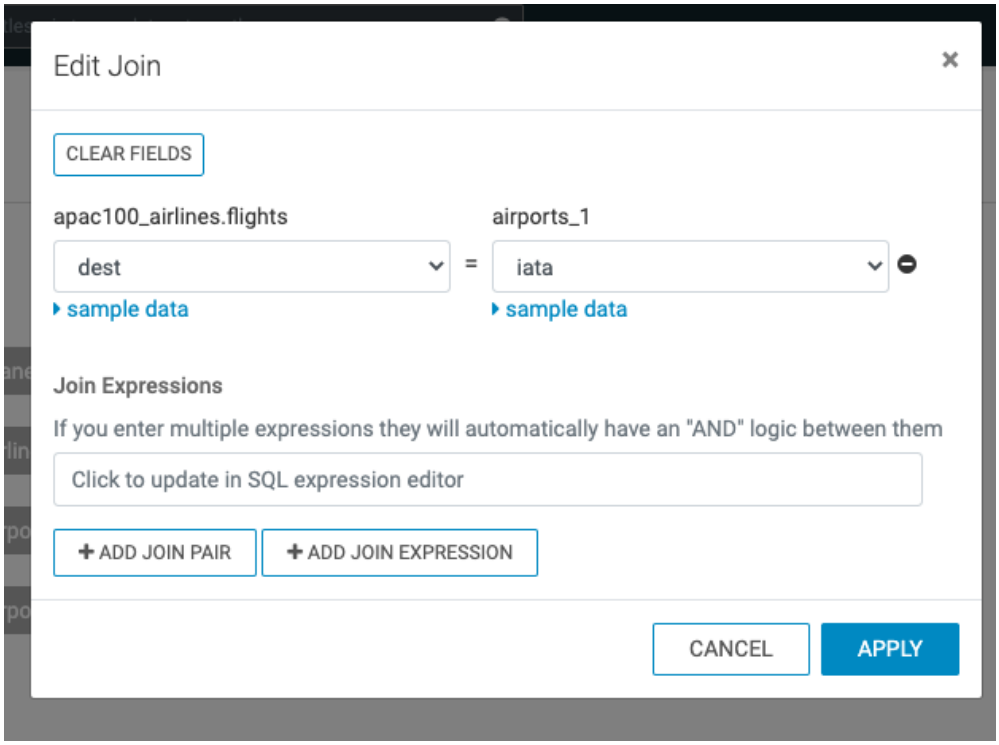
- Click on + icon next to flights as shown below. Select the appropriate Database Name based on your <user_id> (Ex: wuser00_airlines) and table name airports.



- Make sure you select the column dest from flights and column iata from airports table. Click APPLY. Then click on SAVE.

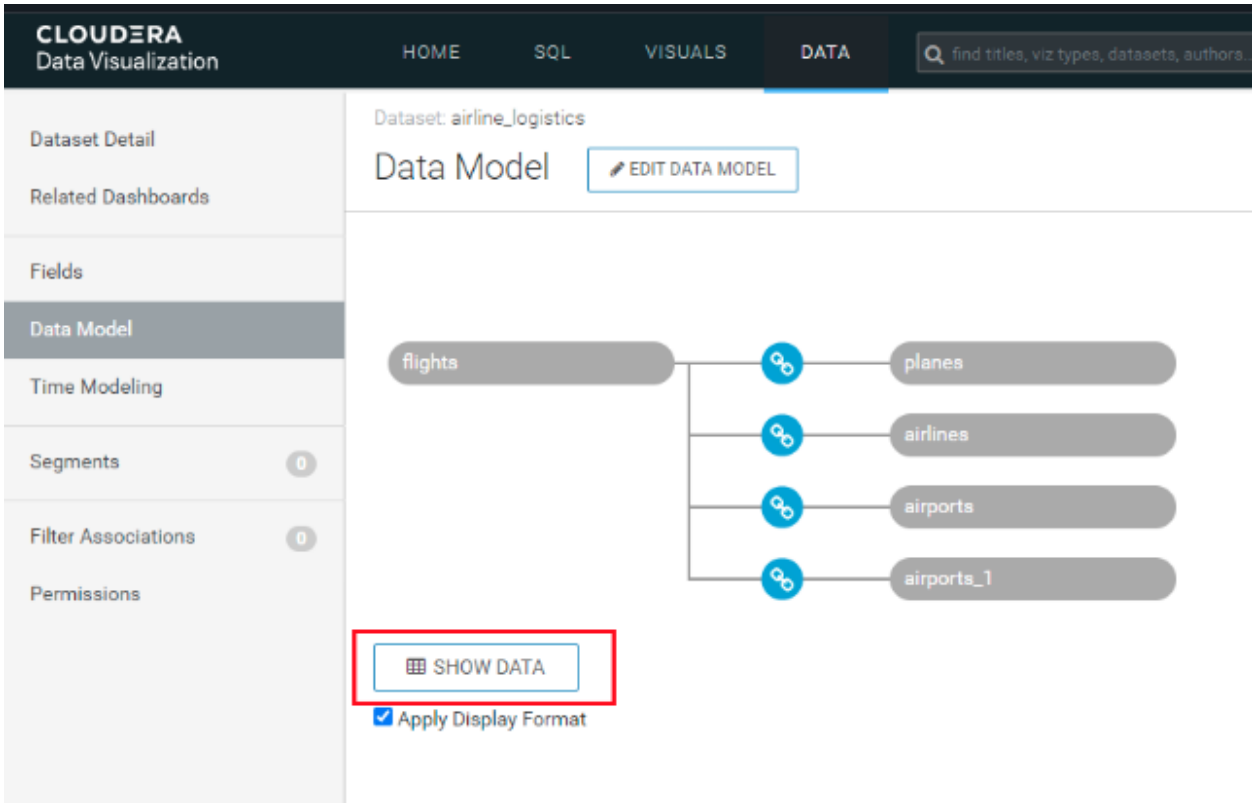


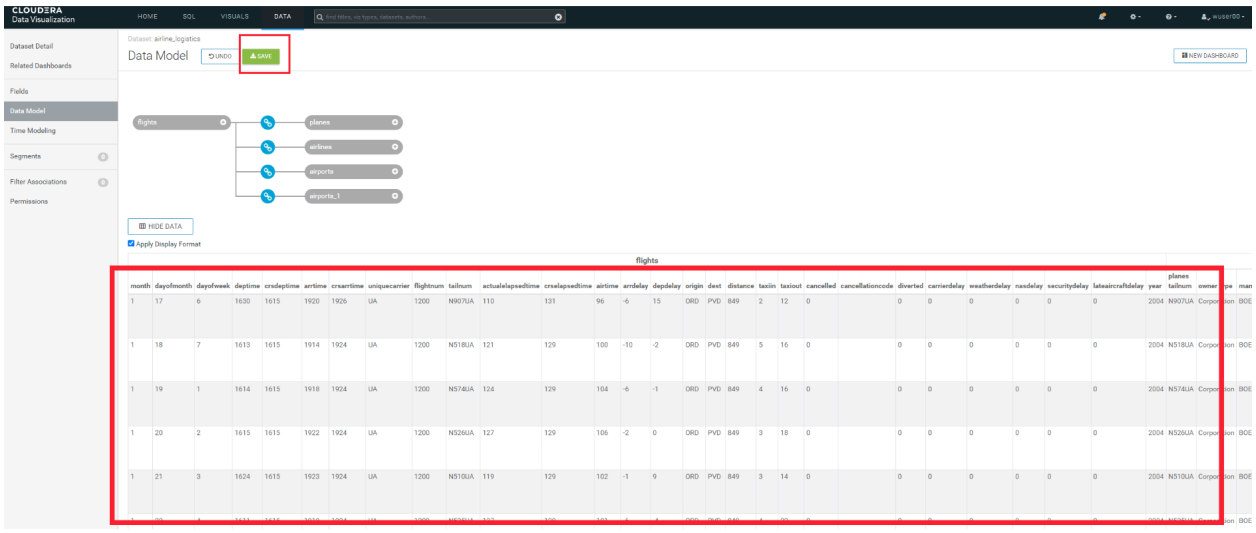
- Select “dest” from the flights table and “iata” from the airports_a table. Click APPLY



  - Verify that you have the joins which are as follows. You can do so by clicking the join icon.
    - flights.tailnum — planes.tailnum
    - flights.uniquecarrier — airlines.code
    - flights.origin — airports.iata
    - flights.dest — airports_1.iata

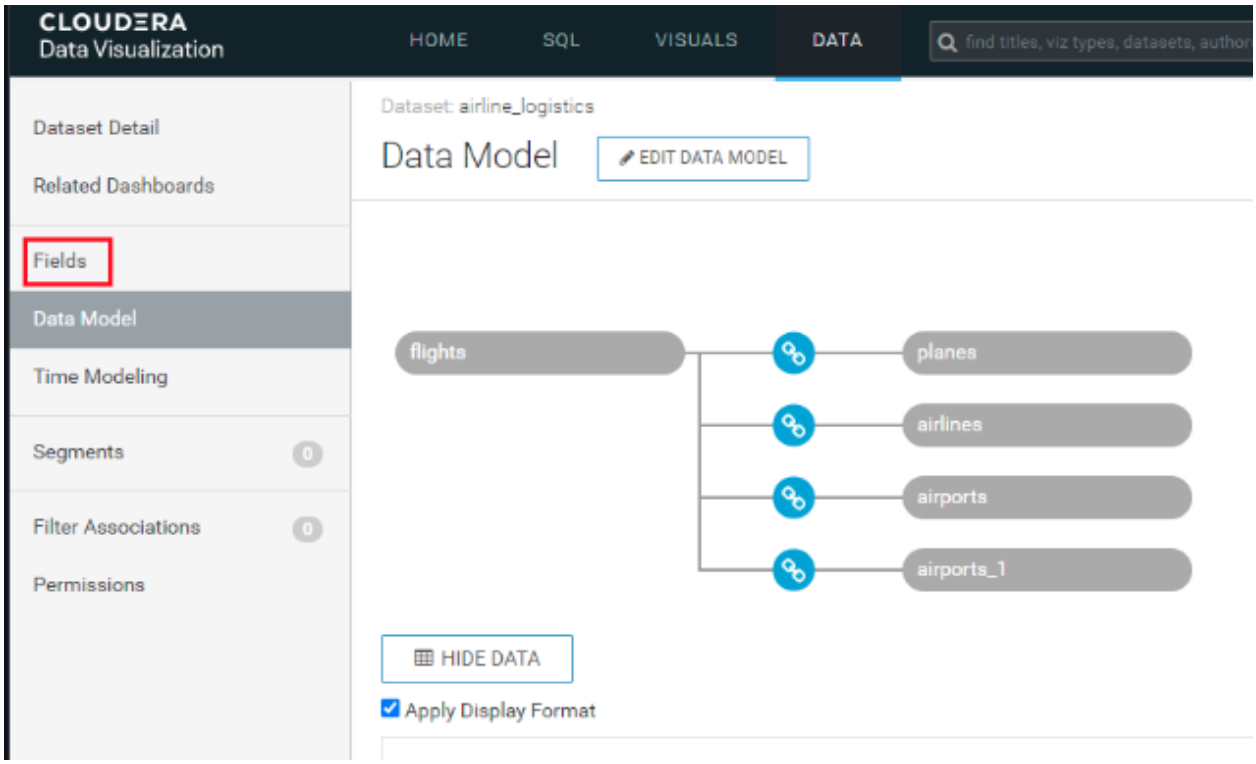
- Click on SHOW DATA. And then click on SAVE.

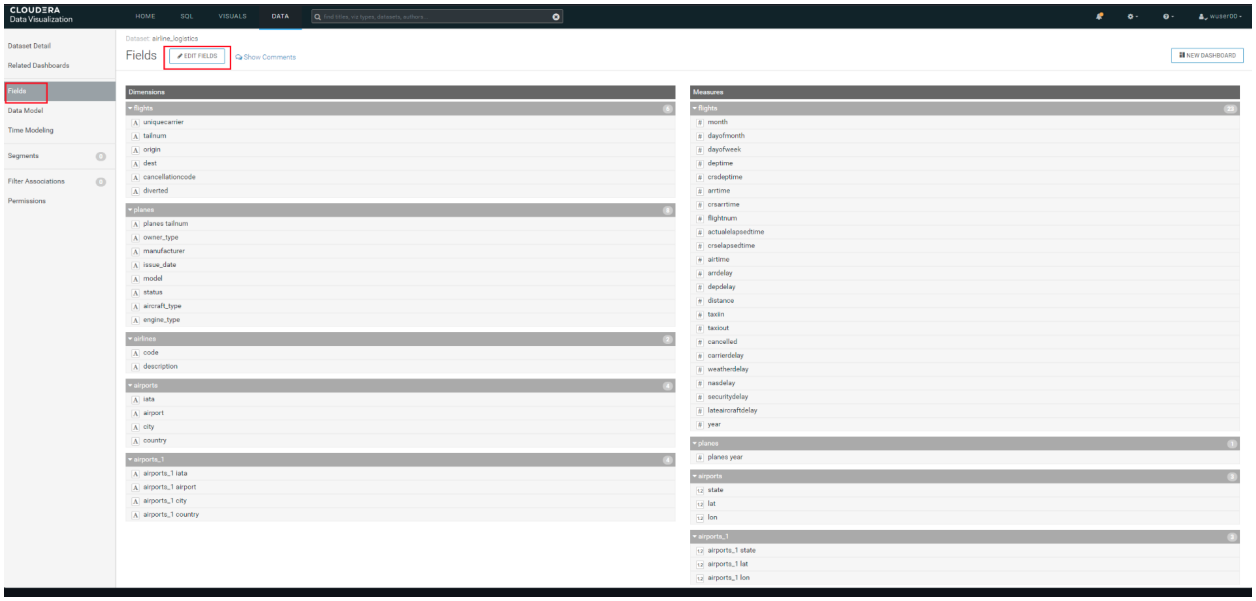

CLOUDERA
Data Visualization
HOME
SQL
VISUALS
DATA
find titles, viz types, datasets, authors...
Dataset Detail
Related Dashboards
Fields
Data Model
Time Modeling
Segments
0
Filter Associations
0
Permissions
Dataset: airline_logistics
Data Model
EDIT DATA MODEL
flights
planes
airlines
airports
airports_1
SHOW DATA
Apply Display Format

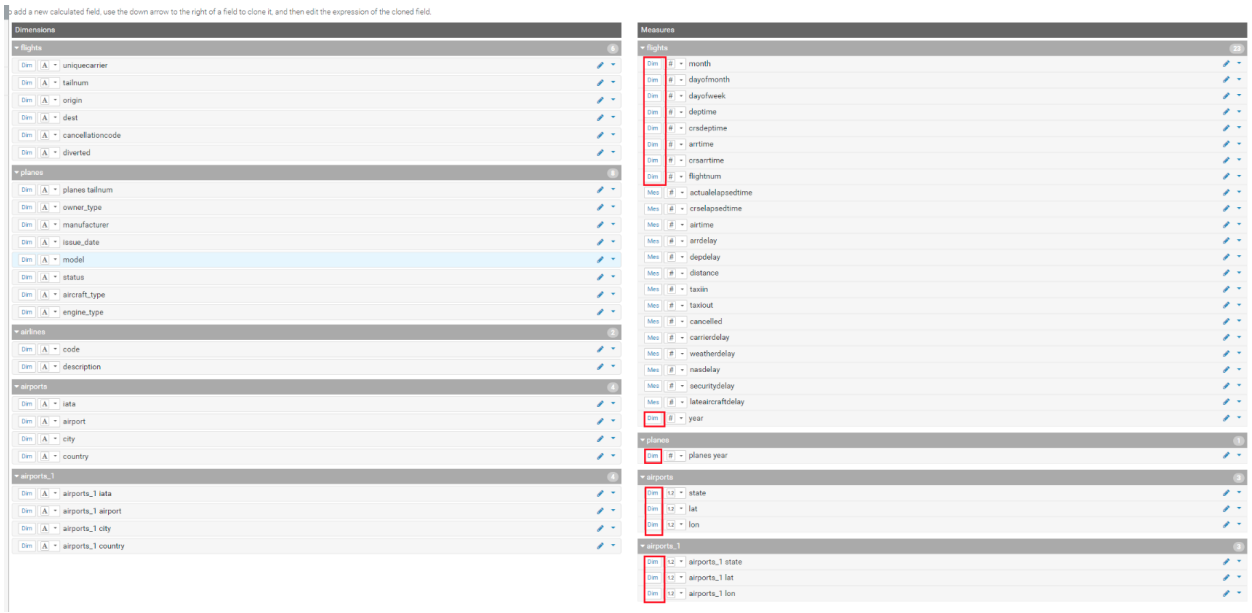

Data Model
UNDO
SAVE
NEW DASHBOARD
HIDE DATA
Apply Display Format
flights

- Click on the Fields column on the left window pane. Then click on EDIT FIELDS. Make sure that you click on the highlighted area to change # (measures icon) next to each column to Dim (dimension icon). The columns are as follows.

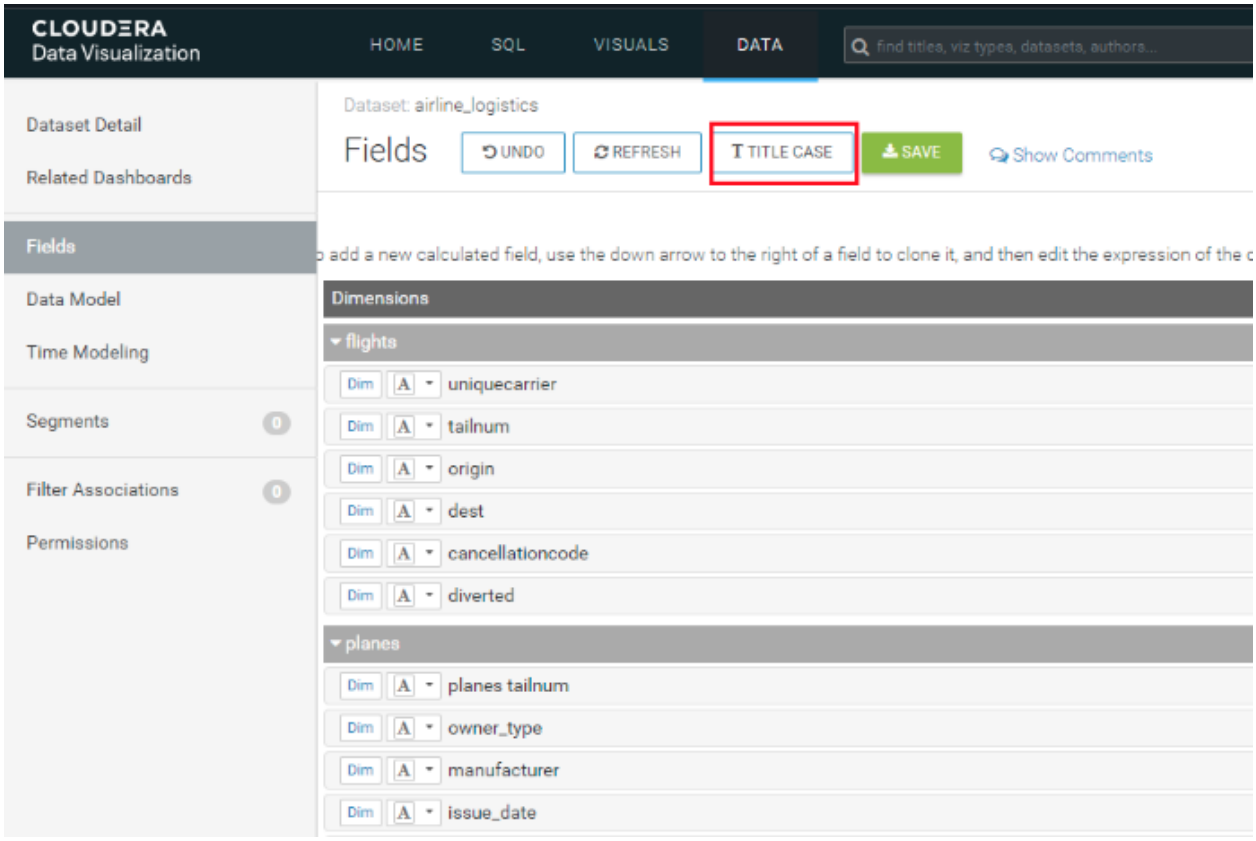
    - **flights table:** Columns (month, dayofmonth, dayofweek, deptime, crsdeptime, arrtime, crsarrtime, flightnum & year)
    - **planes table:** All columns
    - **airports table:** All columns
    - **airports_1 table:** All columns



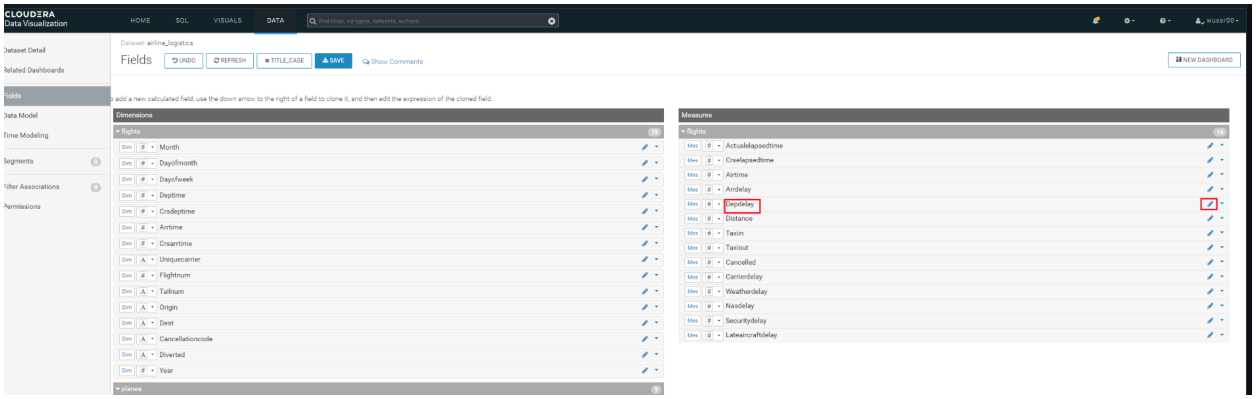




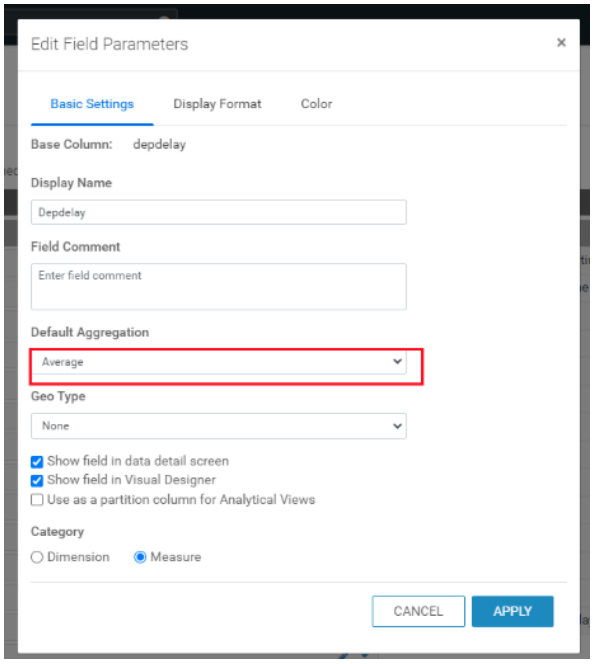
- Click on TITLE CASE. And notice that the column names changes to be Camel case.



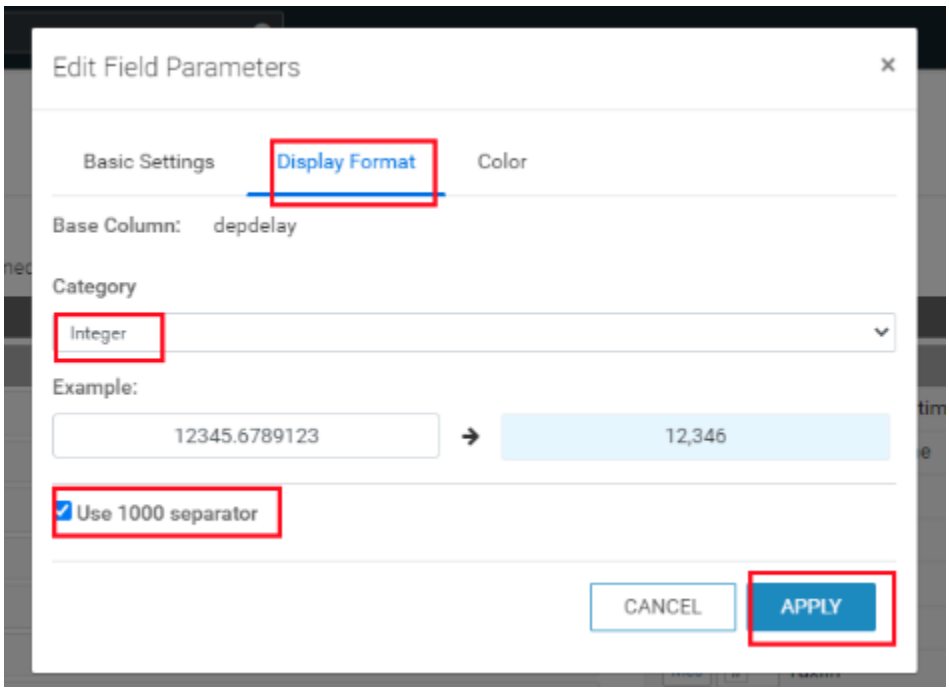
- Click on the pencil icon next to Depdelay icon.



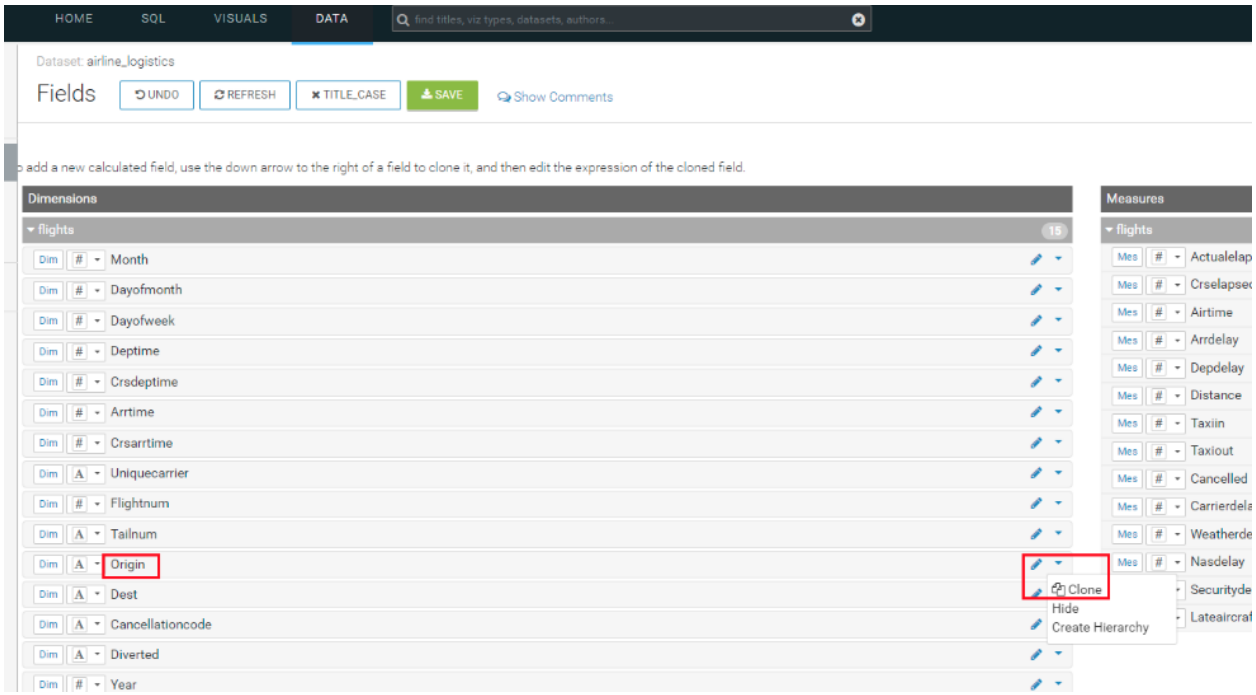
- Change the Default Aggregation to Average.



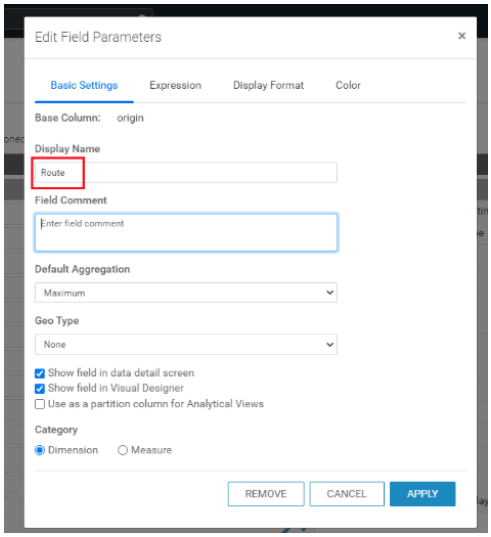
- Click on the Display Format and then change Category to be Integer. Check mark the box next to the Use 1000 separator. Click on APPLY.



Click on the down arrow shown against the Origin column and the click on Clone. A column Copy of Origin is created. Click on the 'pencil' icon next to it.

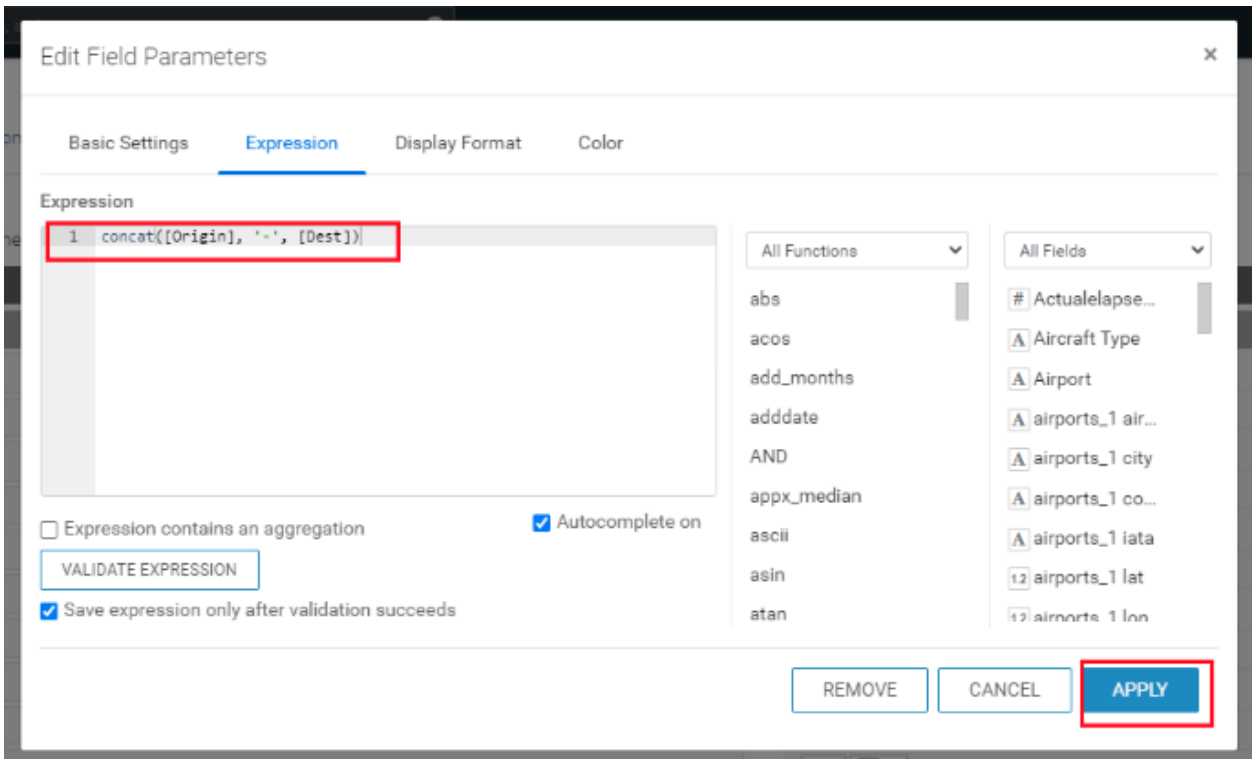


Change the Display Name to Route.

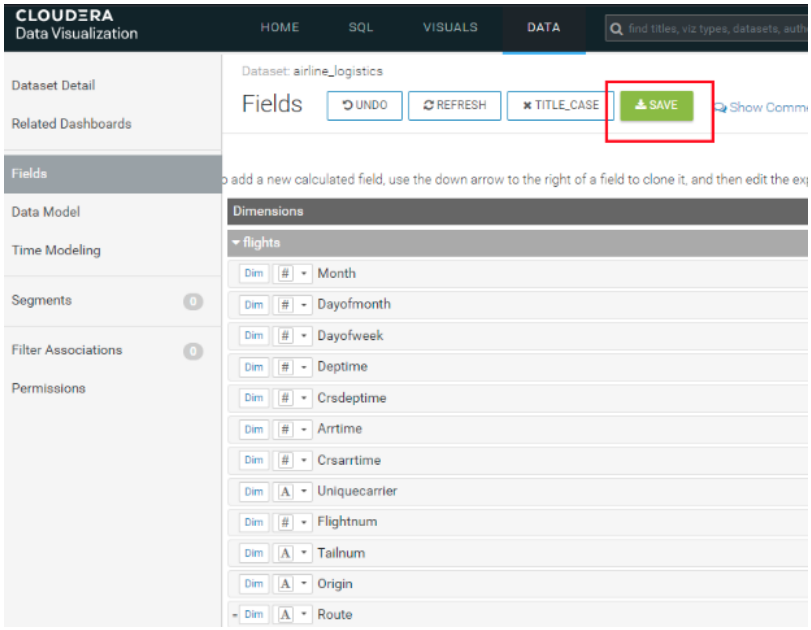


Then click on Expression and enter the following in the Expression editor. Click on APPLY.

```
concat([Origin], '-', [Dest])
```

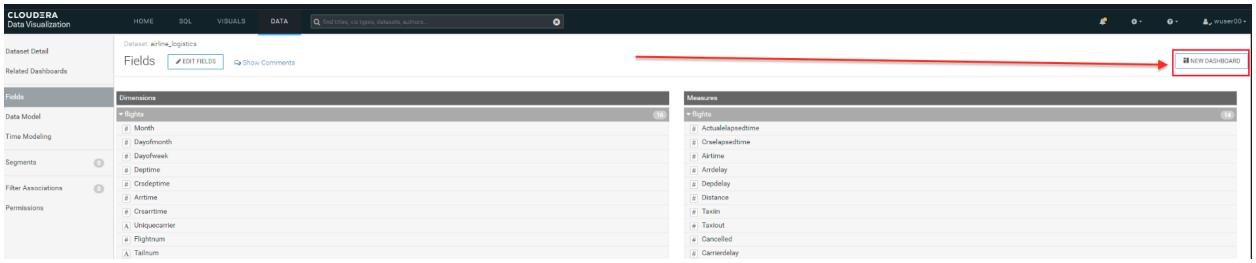


Click on SAVE. We have completed the step of data modeling and now we will create data visualization.

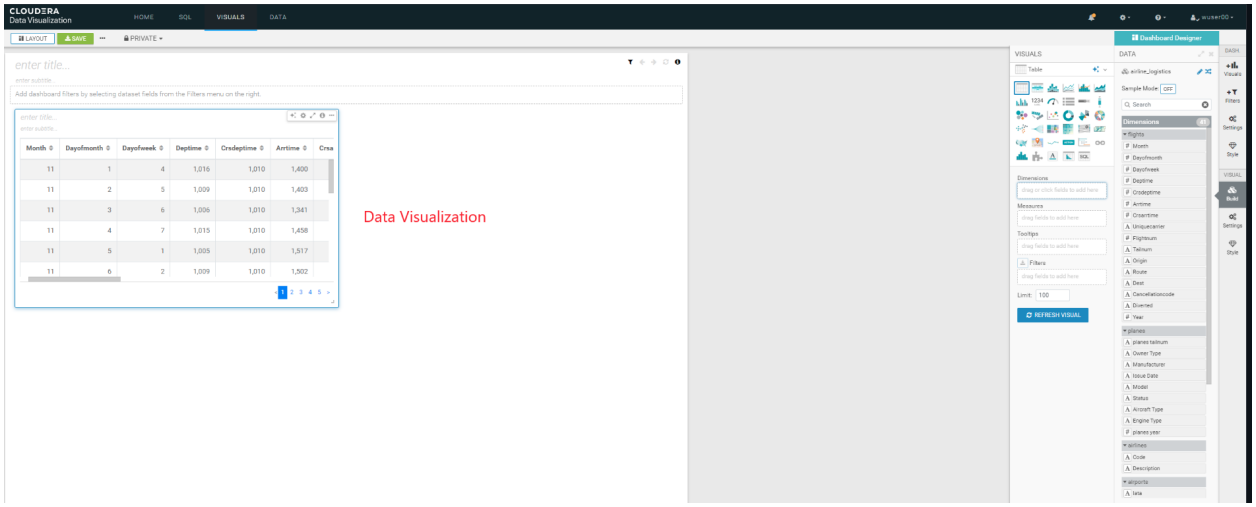


Dashboard Creation

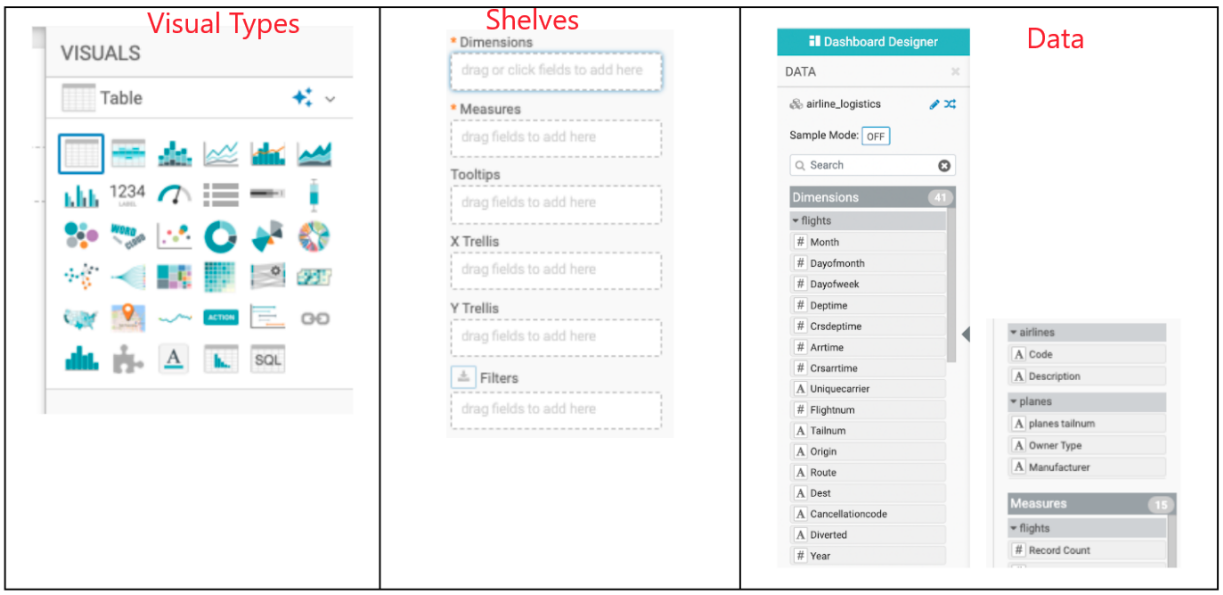
- Now we will create a dashboard page based on the data model that we just created. Click on NEW DASHBOARD.



- You will see the following.



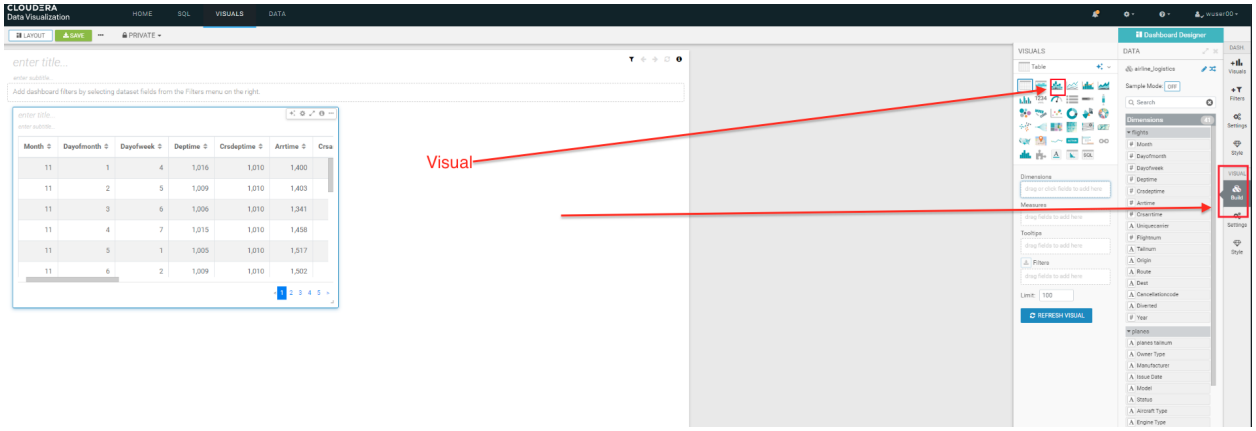
- A quick overview of the screen that you are seeing is as follows. On the right side of the screen there will be a **VISUALS** menu. At the top of this menu, there is a series of Visual Types to choose from. There will be 30+ various visuals to choose from. Below the Visual Types you will see what are called Shelves. The Shelves that are present depend on the Visual Type that is selected. **Shelves** with a * are required, all other Shelves are optional. On the far right of the page there is a **DATA** menu, which identifies the Connection & Dataset used for this visual. Underneath that is the Fields from the Dataset broken down by Dimensions and Measures. With each of these Categories you can see that it is subdivided by each Table in the Dataset.



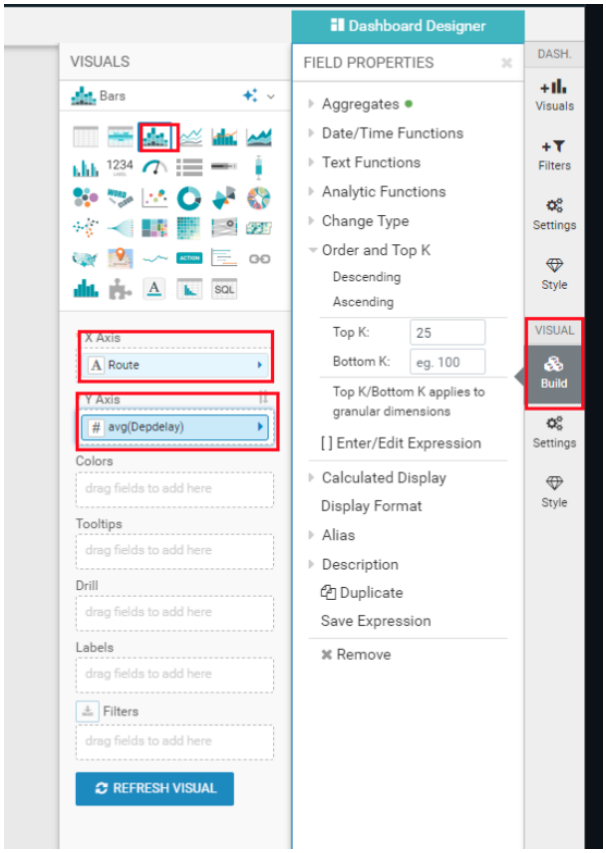
- Let's build 1st visual - Top 25 Routes by Avg Departure Delay. CDV will add a Table visual displaying a sample of the data from the Dataset as the default visualization when you create a new Dashboard or new Visuals on the Dashboard (see New Dashboard

screen above). The next step is to modify (Edit) the default visualization to suit your needs.

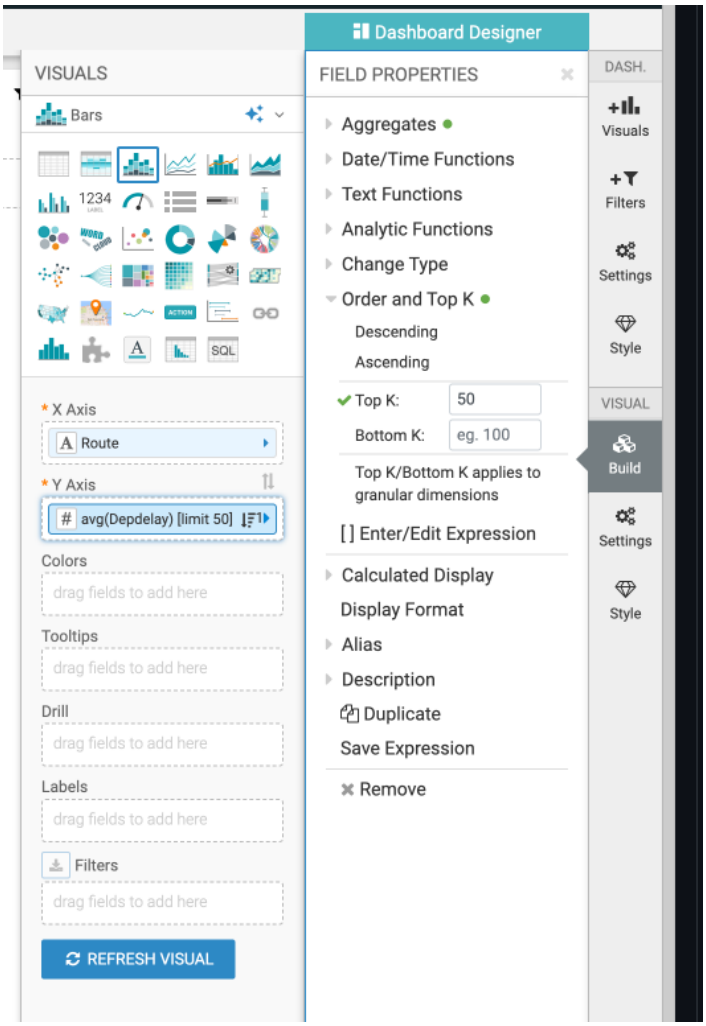
- Pick the Visual Type - Select the Stacked Bar chart visual on the top right as shown below. Make sure Build is selected for it to appear on the right side.

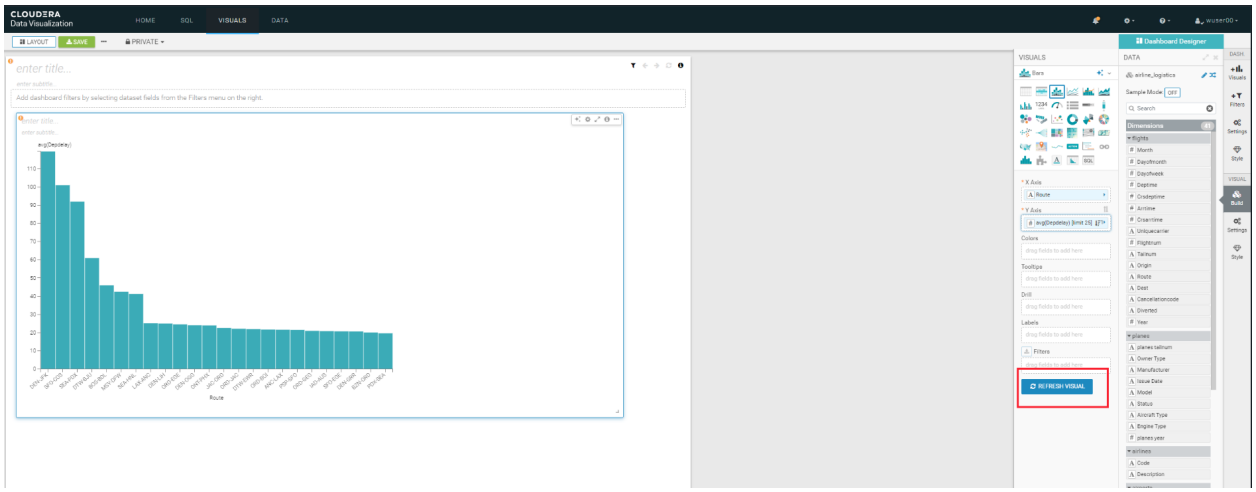


Find Route under Dimensions → flights. Drag to X-Axis. Similarly, find DepDelay under Measures → flights. Drag to Y-Axis. By default the aggergation selected is average and hence you would see avg(Depdelay).

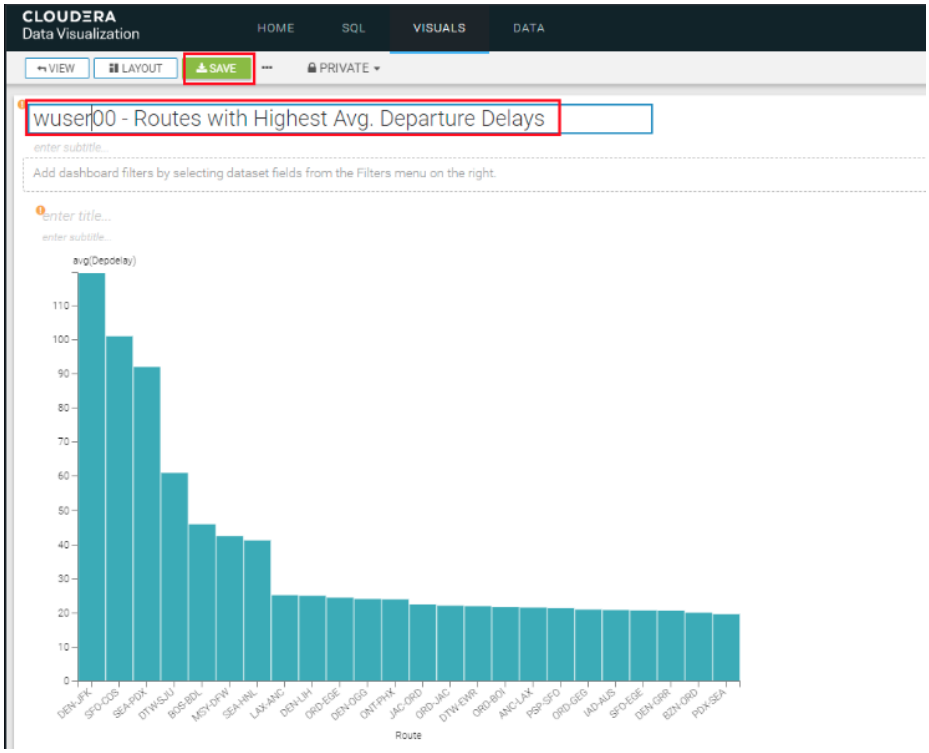


- Click on the arrow next to avg(Depdelay). Enter 50 against the text box labeled Top K. Click on REFRESH VISUAL.





- Click enter title and enter the title based on your user id as - <user_id>- Routes with Highest Avg. Departure Delays. Then click on SAVE.



This concludes the workshop. Hope you had a great time learning something new and useful.