

CLOUDERA

DATA WAREHOUSE HANDS-ON

STUDENT GUIDE

[Preface](#)

[Pre-requisites](#)

[Workshop Overview](#)

[LAB 1 - Getting Started](#)

[VERIFY RESOURCES](#)

[DEFINE CDP WORKLOAD PASSWORD](#)

[Step 1 : Login](#)

[Step 2 : Go to Profile](#)

[Step 3 : Set Workload Password](#)

[LAB 2 - RAW LAYER](#)

[DATABASE AND TABLE CREATION](#)

[Step 1 : Go To Hue \[Hive Cluster\]](#)

[Step 2 : CREATE DATABASE](#)

[Step 3 : CREATE EXTERNAL TABLES](#)

[Step 4 : VERIFY IF THE TABLES ARE CREATED](#)

[Step 5 : QUERY CREATED TABLES USING IMPALA VIRTUAL WAREHOUSE](#)

[LAB 3 - DATA LAKEHOUSE](#)

[CURATED LAYER CREATION](#)

[Step 1 : CREATE and LOAD PLANES TABLE](#)

[Step 2 : DESCRIBE CREATED TABLE](#)

[Step 3 : CREATE AND LOAD AIRLINES TABLE](#)

[Step 4 : CHECK ACID CAPABILITIES](#)

[MIGRATE HIVE TO ICEBERG TABLE](#)

[Utilize Table Migration Feature](#)

[Use Create table as Select\(CTAS\)](#)

[CREATE ICEBERG TABLES](#)

[LAB 4 - PERFORMANCE OPTIMIZATION AND TABLE MAINTENANCE](#)

[Iceberg in-place Partition Evolution\[Performance Optimization\]](#)

[Step 1 : Open HUE for the CDW Hive Virtual Warehouse](#)

[Step 2 : Execute Query](#)

[Step 3 : Check for optimizations](#)

[Iceberg Snapshots \[Table Maintenance\]](#)

[Iceberg Time Travel \[Table Maintenance\]](#)

[Iceberg Rollback \[Table Maintenance\] - DO NOT RUN](#)

[Iceberg Rollback \[Table Maintenance\] - DO NOT RUN](#)

[Materialized Views \[Performance Optimization\]](#)

[LAB 5 - SECURITY AND GOVERNANCE](#)

[Step 1 : Add New Policy](#)

[Step 2 : Test New Policy](#)

[LAB 6 - CLOUDERA DATA VISUALIZATION](#)

[Data Modeling](#)

Dashboard Creation

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:

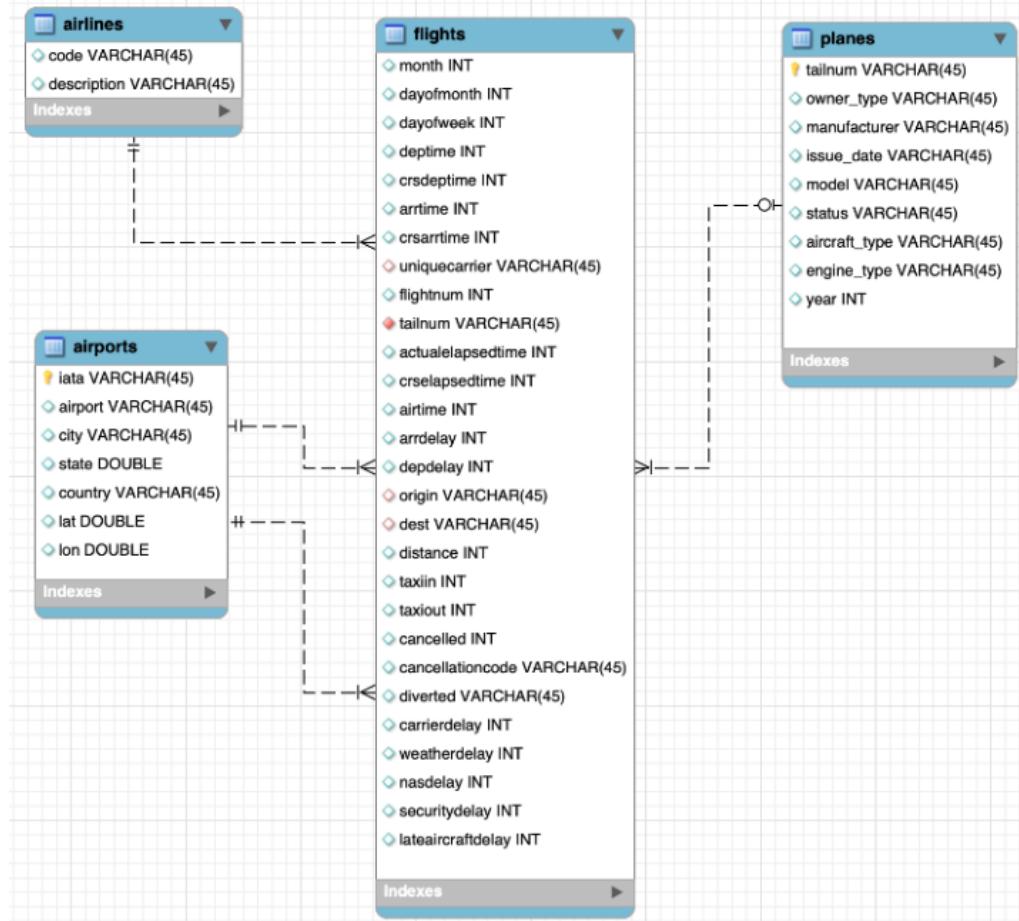
We will dive into this Scenario to show Cloudera Data Warehouse (CDW) is used to enable the Aircraft company to gain competitive advantage - and at the same time it highlights the performance and automation capabilities that help ensure performance is maintained while controlling costs.

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

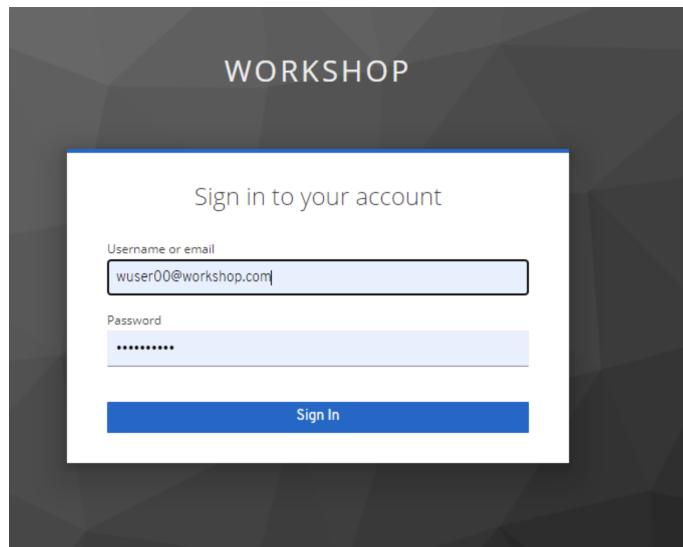
For this workshop you will use the following credentials and resources

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

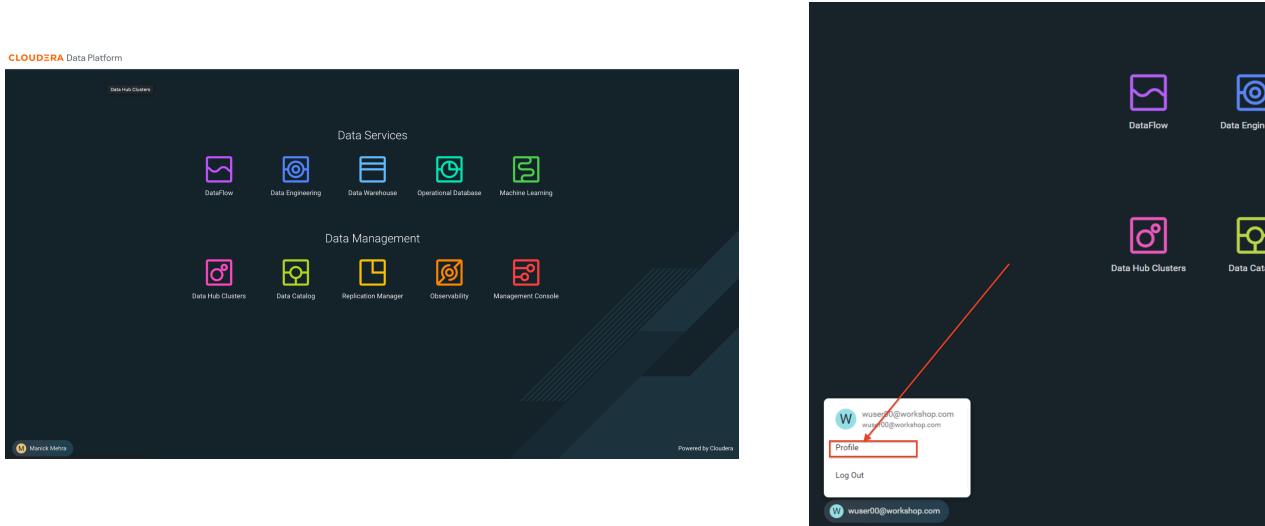
Step 1 : Login

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**

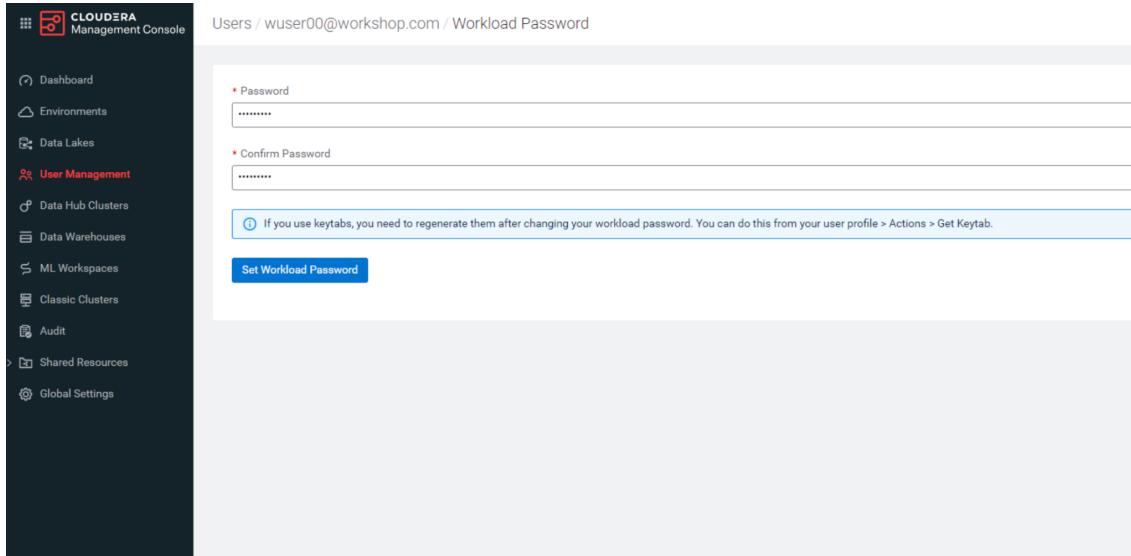


Step 3 : Set Workload Password

Click option Set Workload Password.

The image shows a user profile page for 'wuser00@workshop.com'. The page includes fields for Name, Email, Workload User Name, CRN, Tenant ID, Identity Provider, Last Interactive Login, and Profile Management. Under the 'Workload Password' section, there is a link labeled 'Set Workload Password' with the note '(Workload password is currently set)'. Below this, there are tabs for 'Access Keys', 'Roles', 'Resources', 'Groups', and 'SSH Keys'. The 'Access Keys' tab is selected, showing a message 'No access keys found.' and a 'Generate Access Key' button. A red arrow points from the text 'Click option Set Workload Password.' to the 'Set Workload Password' link.

Enter the Password shared by the instructor and Confirm Password.

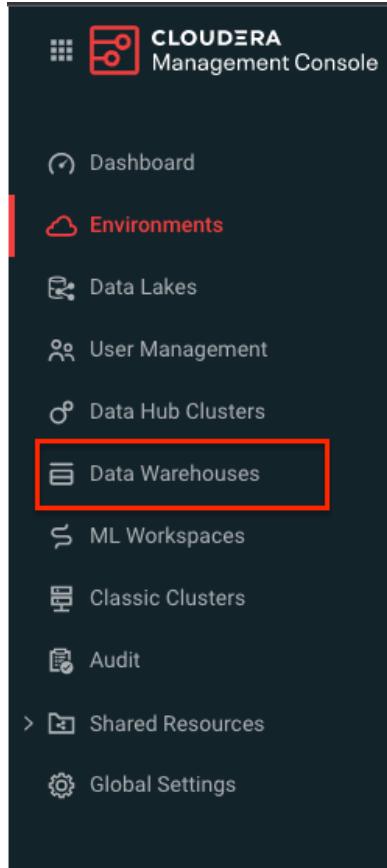


Click the button **Set Workload Password**.

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.

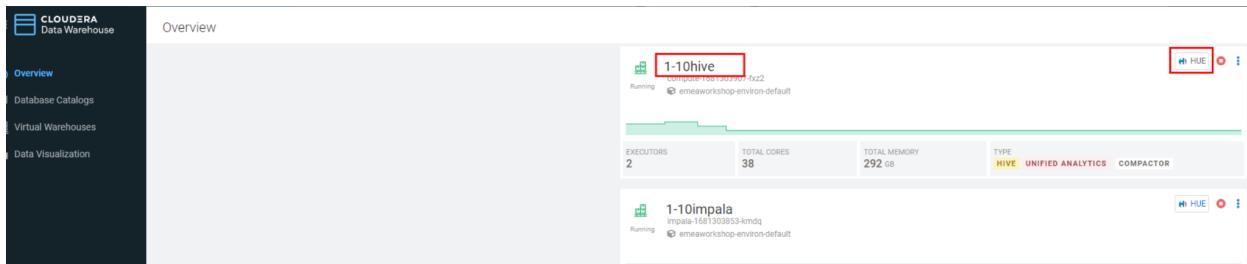
To access Data Warehouse data service click on Data Warehouse on the left.



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;
```

The screenshot shows the Cloudera Hive interface. At the top, there's a header with a 'Hive' icon, 'Add a name...', and 'Add a description...'. The main area contains a code editor with the following SQL statements:

```
1 CREATE DATABASE ${user_id}_airlines_raw;
2 CREATE DATABASE ${user_id}_airlines;
```

Below the editor, there's a dropdown menu with 'user_id' selected, and a suggestion 'apac01' is shown. At the bottom, there are tabs for 'Query History' and 'Saved Queries', with the message 'You don't have any saved queries.'

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}%';
```

The screenshot shows the Cloudera Manager Hive interface. At the top, there is a search bar with the placeholder "Add a name..." and "Add a description...". Below the search bar, a query editor window contains the command: "SHOW DATABASES LIKE '\${user_id}%';". To the right of the query editor, there is a dropdown menu with the value "user_id" selected. Below the query editor, the log output shows the execution of the command:

```
INFO : Completed compilling Command(queryId=hive_20230522064314_72b2e3cb-157f-4656- INFO : Executing command(queryId=hive_20230522064314_72b2e3cb-157f-4656- INFO : Starting task [Stage-0:DDL] in serial mode INFO : Completed executing command(queryId=hive_20230522064314_72b2e3cb- INFO : OK
```

At the bottom of the interface, there are three tabs: "Query History", "Saved Queries", and "Results (2)". The "Results (2)" tab is selected. The results table has a header "database_name" and contains two rows:

database_name
1 apac100_airlines
2 apac100_airlines_raw

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, crsarrrtime int, uniquecarrier
string, flightnum int, tailnum string, actualelapsedtime int, crselapsedtime int, airtime
int, arrdelay int, depdelay int, origin string, dest string, distance int, taxiin int,
taxiout int, cancelled int, cancellationcode string, diverted string, carrierdelay int,
weatherdelay int, nasdelay int, securitydelay int, lateaircraftdelay int, 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");
```

The screenshot shows the Cloudera Manager Hive interface. In the top navigation bar, there is a search bar labeled "Search data and saved documents...". Below the navigation bar, the "Hive" tab is selected. On the left, a sidebar titled "Databases" lists several databases: default, information_schema, salmazon, salmazon_airlines, salmazon_airlines_raw, sys, and wuser0_airlines, wuser0_airlines_raw. The main area displays the following Hive SQL code:

```

1 DROP IF EXISTS $user_id.airlines_raw.flights_csv;
2 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, actualtailnum string);
3 ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
4 STORED AS TEXTFILE LOCATION 's3a://meta-workshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/flights' tblproperties("skip.header.line.count"="1");
5 ;
6 ;
7 drop table if exists $user_id.airlines_raw.planes_csv;
8 CREATE EXTERNAL TABLE $user_id.airlines_raw.panes_csv(tailnum string, owner_type string, manufacturer string, issue_date string, model string, status string, aircraft_type string, engine_type string, year int);
9 ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
10 STORED AS TEXTFILE LOCATION 's3a://meta-workshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/planes' tblproperties("skip.header.line.count"="1");
11 ;
12 drop table if exists $user_id.airlines_raw.airlines_csv;
13 CREATE EXTERNAL TABLE $user_id.airlines_raw.airlines_csv(code string, description string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
14 STORED AS TEXTFILE LOCATION 's3a://meta-workshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/airlines' tblproperties("skip.header.line.count"="1");
15 ;
16 drop table if exists $user_id.airlines_raw.airports_csv;
17 CREATE EXTERNAL TABLE $user_id.airlines_raw.airports_csv(ata string, airport string, city string, state DOUBLE, country string, lat DOUBLE, lon DOUBLE);
18 ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n';
19 STORED AS TEXTFILE LOCATION 's3a://meta-workshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/airports' tblproperties("skip.header.line.count"="1");

```

Below the code, a red box highlights the placeholder "\$user_id" in the query. The output pane shows the execution results:

```

INFO : PURGE: 0 VALIDATION FAILURE: 0 UNKNOWN: 0
INFO : Starting task [Stage-0-DDL] in serial mode
INFO : Completed executing command(queryId:hive_20230403130802_58eb9a45-f693-4fbc-a6ff-85483230b002); Time taken: 0.272 seconds
INFO : OK

```

The "Success" message is displayed.

At the bottom, the "Query History" section shows the following log entries:

- a few seconds ago CREATE EXTERNAL TABLE wuser0_airlines_raw.airports_csv(ata 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://meta-workshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/airports' tblproperties("skip.header.line.count"="1")
- a few seconds ago drop table if exists wuser0_airlines_raw.airports_csv
- a few seconds ago CREATE EXTERNAL TABLE wuser0_airlines_raw.airlines_csv(code string, description string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'; STORED AS TEXTFILE LOCATION 's3a://meta-workshop/my-data/meta-cdw-workshop/airlines-raw/airlines-csv/airlines' tblproperties("skip.header.line.count"="1")
- a few seconds ago drop table if exists wuser0_airlines_raw.airlines_csv

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.

The screenshot shows the Cloudera Manager Hue interface. On the left, the 'Databases' sidebar lists several databases: default, information_schema, salmazan, salmazan_airlines, salmazan_airlines_raw, sys, wuser00_airlines, and wuser00_airlines_raw. The main area is titled 'Hive' and shows a query editor with the following DDL:

```
1 USE ${user_id}_airlines_raw;
2
3 SHOW TABLES;
```

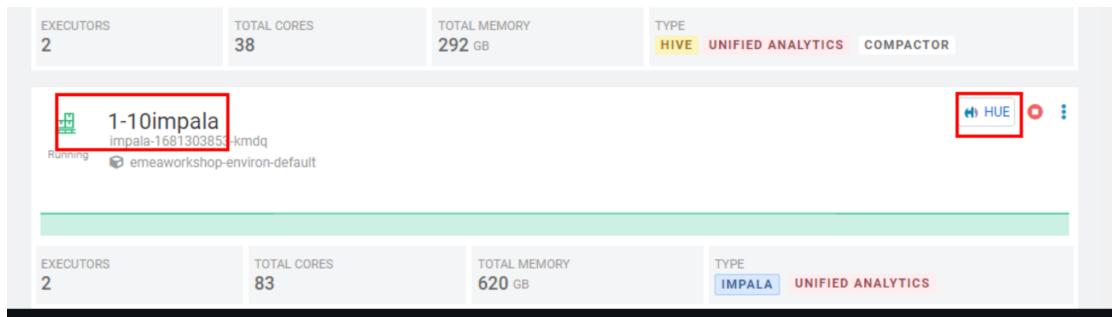
Below the query editor, a results table is displayed with the following data:

tab_name
1 airlines_csv
2 airports_csv
3 flights_csv
4 planes_csv

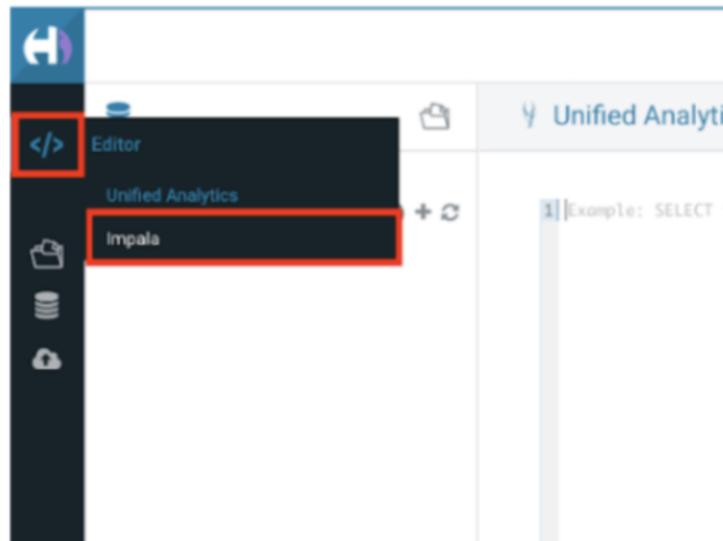
The results table is highlighted with a red box.

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;
```

A screenshot of the Cloudera Hue Editor. The top navigation bar shows the URL 'hue-meta-cdw-impala-workshop-vw.dw-meta-workshop.dp5i-5vk.cloudera.site/hue/editor?editor=428'. The main interface has a sidebar on the left with 'Tables' (1) and 'customer' listed. The central area shows a code editor with the query 'select count(*) from \${user_id}_airlines_raw.flights_csv;'. Below the code editor is a progress bar showing four queries running, each at 100% completion. At the bottom, there are tabs for 'Query History', 'Saved Queries', and 'Results (1)'. The 'Results (1)' tab is active, displaying a single row of results: 'count(*)' with a value of '86289323'. This result is also highlighted with a red box.

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);
```

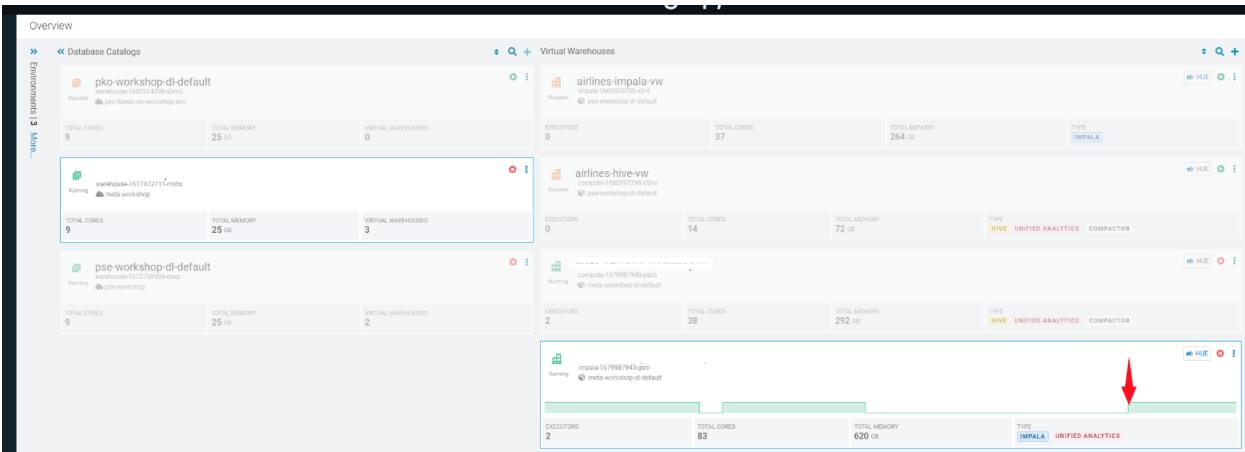
```

SELECT model,
       engine_type
  FROM ${user_id}.airlines_raw.flights_csv
 WHERE planes.csv.tailnum IN
      (SELECT tailnum
        (SELECT tailnum,
                count(),
                avg(dependency) AS avg_delay,
                max(dependency),
                avg(txout),
                avg(cancelled),
                avg(delayed),
                max(weatherdelay),
                avg(asesdelay),
                max(securitydelay),
                avg(securitydelay),
                max(securitydelay),
                avg(lateaircrefutele),
                avg(latetimeschedule),
                avg(latetimeschedule),
                avg(distance)
       FROM ${user_id}.airlines_raw.flights_csv
      WHERE tailnum IN ('N8085',
                         'N8751',
                         'N8752',
                         'N880A')
      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.

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.

The screenshot shows the Cloudera Impala Query Editor interface. At the top, there's a search bar and a toolbar with various icons. Below that, a sidebar lists tables: `airlines.csv`, `airports.csv`, `flights.csv`, and `planes.csv`. The main area contains a query editor window with the following SQL code:

```
1 SELECT model,
2     engine_type
3 FROM t1(user_id)_airlines_raw_planes_csv
4 WHERE tailnum IN (SELECT tailnum
5     FROM t1(user_id)_airlines_raw_planes_csv
6     GROUP BY tailnum)
7     (SELECT tailnum,
8         count(*),
9         avg(deptime) AS avg_deplay,
10        max(deptime),
11        avg(cancelled),
12        max(cancelled),
13        avg(taxiout),
14        max(taxiout),
15        avg(taxiin),
16        max(taxiin),
17        avg(securitydelay),
18        max(securitydelay),
19        avg(lateaircraftdelay),
20        max(lateaircraftdelay),
21        avg(delay),
22        max(delay),
23        avg(distance),
24        max(distance),
25    FROM t1(user_id)_airlines_raw_flights_csv
26    WHERE tailnum IN ('N965US',
27                      'N885AA',
28                      'N852NE',
29                      'N852NE')
30    GROUP BY tailnum AS delays);
31
```

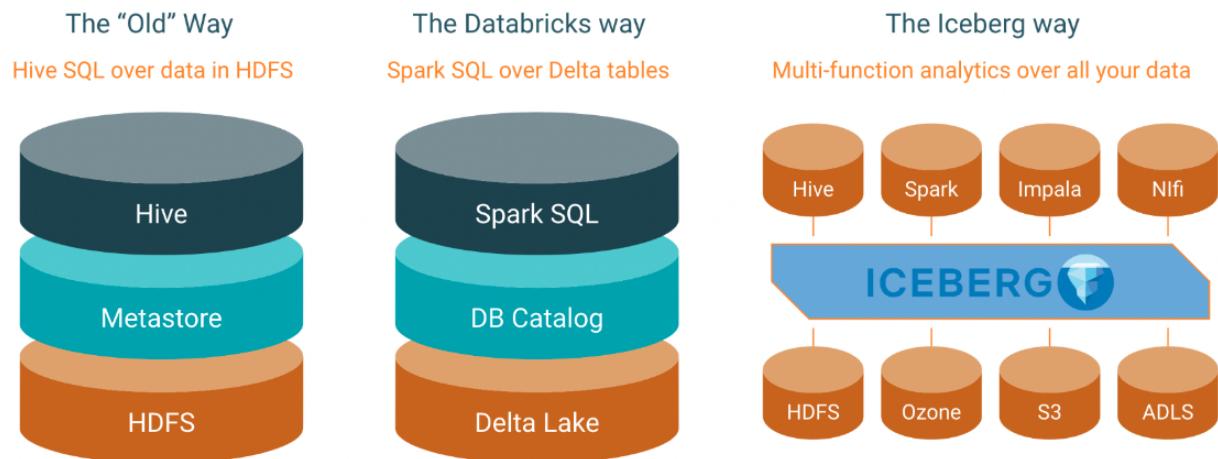
Below the query, there's a dropdown menu set to `3m, 24s wuser00_airlines_raw` with a red box highlighting it. A message box says "Admission result : Queued". The results section shows one row of data:

model	engine_type
A330-223	Turbo-Fan

The URL in the browser address bar is eb40fd4ce343098e17ba7c7100000000.

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.

A screenshot of the Cloudera Manager interface showing a service named "1-10hive". The service status is "Running" and it is associated with a specific compute node and environment. The service details panel shows metrics like EXECUTORS (2), TOTAL CORES (38), and TOTAL MEMORY (292 GB). The "TYPE" section indicates the service is of type HIVE, with options for UNIFIED ANALYTICS and COMPACTOR. A red box highlights the "HUE" label in the top right corner of the service card.

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');

```

The screenshot shows the Cloudera Manager Hive interface. At the top, there are tabs for 'Hive', 'Add a name...', and 'Add a description...'. Below the tabs, there is a code editor containing the DDL for creating an external table. A red box highlights the 'user_id' field in the 'STORED AS PARQUET' section. The output window below the editor shows the command being executed and a success message. The bottom part of the interface includes 'Query History' and 'Saved Queries' tabs, and a log window showing the execution details.

```

1 drop table if exists ${user_id}_airlines.planes;
2
3 CREATE EXTERNAL TABLE ${user_id}_airlines.planes (
4     tailnum STRING, owner_type STRING, manufacturer STRING,
5     issue_date STRING,
6     model STRING, status STRING, aircraft_type STRING, engine_type
7     STRING, year INT
8 )
9 STORED AS PARQUET
10 TBLPROPERTIES ('external.table.purge'='true');

[▶] user_id | wuser00

```

0.54s default ▾

INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId=hive_20230404040720_729ec642-a65b-4c40-ad52-f21784a49a48); Time taken: 0.465 seconds
INFO : OK

Success.

Query History Saved Queries

a few seconds ago CREATE EXTERNAL TABLE wuser00_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;

```

The screenshot shows the Cloudera Manager Hive interface. At the top, there are tabs for 'Hive', 'Add a name...', and 'Add a description...'. Below the tabs, there is a code editor containing the INSERT INTO statement. A red box highlights the 'user_id' field in the table reference. The output window below the editor shows the command being executed and its completion. The bottom part of the interface includes 'Query History' and 'Saved Queries' tabs, and a log window showing the execution details.

```

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

```

[▶] user_id | wuser00

INFO : EXECUTING STATEMENT
INFO : Table wuser00_airlines.planes stats: [numFiles=1, numRows=5029, totalSize=60020, rawDataSize=45261, numFilesErasureCoded=0]
INFO : Table wuser00_airlines.planes stats: [numFiles=1, numRows=5029, totalSize=60020, rawDataSize=45261, numFilesErasureCoded=0]

- 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;
```

The screenshot shows the Cloudera Manager Hive interface. At the top, there is a search bar with 'Hive' and buttons for 'Add a name...' and 'Add a description...'. Below the search bar, a query editor window displays the command: 'SELECT * FROM \${user_id}_airlines.planes LIMIT 100;'. A dropdown menu next to the user_id placeholder shows 'wuser00'. The status bar indicates '2.38s wuser00_airlines'. The log pane shows the execution details:

```
INFO : Completed compiling command(queryId=hive_20230404041340_3e3aa0e9-d926-4547-9d5f-46ea40c45181); Time taken: 1.868 seconds
INFO : Executing command(queryId=hive_20230404041340_3e3aa0e9-d926-4547-9d5f-46ea40c45181): SELECT * FROM wuser00_airlines.planes LIMIT 100
INFO : Completed executing command(queryId=hive_20230404041340_3e3aa0e9-d926-4547-9d5f-46ea40c45181); Time taken: 0.004 seconds
INFO : OK
```

Below the log, there are tabs for 'Query History', 'Saved Queries', and 'Results (100+)'. The 'Results (100+)' tab is selected, showing a table with columns: planes.tailnum, planes.owner_type, planes.manufacturer, planes.issue_date, planes.model, planes.status, planes.aircraft_type, and planes.engine_type. The results list the first 8 rows of the table, all of which have the tail number 'N050AA'.

	planes.tailnum	planes.owner_type	planes.manufacturer	planes.issue_date	planes.model	planes.status	planes.aircraft_type	planes.engine_type
1	N050AA							
2	N051AA							
3	N052AA							
4	N054AA							
5	N055AA							
6	N056AA							
7	N057AA							
8	N058AA							

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

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

user_id wuser00

```
INFO : Executing command(queryId=hive_20230413110148_0bc2fd50-5853-4a77-94ae-401c81d88528): DESCRIBE FORMATTED wuser00_airlines.planes
INFO : Starting task [Stage-0:DML] in serial mode
INFO : Completed executing command(queryId=hive_20230413110148_0bc2fd50-5853-4a77-94ae-401c81d88528); Time taken: 0.025 seconds
INFO : OK
```

col_name	data_type	comment
1 tailnum	string	
2 owner_type	string	
3 manufacturer	string	

16 LastAccessTime:	UNKNOWN	NULL
17 Retention:	0	NULL
18 Location:	s3a:// use/tables/external/hive/wuser00_airlines.db/planes	NULL
19 Table Type:	EXTERNAL_TABLE	NULL
20 Table Parameters:	NULL	NULL
21	COLUMN_STATS_ACCURATE	{"BASIC_STATS":true,"COLUMN_STATS":{}}
22	EXTERNAL	TRUE
23	bucketing_version	2
24	external.table.purge	true
25	numFiles	1
26	numRows	5029
27	rawDataSize	45261
28	totalSize	60020
29	transient_lastDdlTime	1681383501
30	NULL	NULL
31 # Storage Information	NULL	NULL
32 SerDe Library:	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe	NULL
33 InputFormat:	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat	NULL
34 OutputFormat:	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat	NULL

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;
```

The screenshot shows the Cloudera Manager Hive interface. At the top, there are tabs for 'Hive' (selected), 'Add a name...', and 'Add a description...'. Below the tabs, the query text is displayed:

```
1 drop table if exists ${user_id}_airlines.airlines_orc;
2 CREATE TABLE ${user_id}_airlines.airlines_orc
3 STORED AS ORC
4 AS
5   SELECT * FROM ${user_id}_airlines_raw.airlines_csv;
```

The status bar at the bottom right indicates '4.24s wuser00_airlines'. The results pane shows the output of the query execution:

```
INFO : Table wuser00_airlines.airlines_orc stats: [numFiles=2, numRows=1492, totalSize=17492, rawDataSize=0, numFilesErasureCoded=0]
INFO : Completed executing command(queryId=hive_20230404045854_9c62ed96-4e5b-4e68-a523-3f9102f8c187); Time taken: 2.499 seconds
INFO : OK
```

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");
```

The screenshot shows the Cloudera Manager Hive interface. At the top, there are tabs for 'Hive' (selected), 'Add a name...', and 'Add a description...'. Below the tabs, the query text is displayed:

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

The status bar at the bottom right indicates 'Executing command(queryId=hive_20230404051850_d5bbff094-f03c-401f-8ce0-da569c8067c1): SELECT * FROM wuser00_airlines.airlines_orc WHERE code IN ("UA", "XX", "PAW")'. The results pane shows the output of the query execution:

```
INFO : Executing command(queryId=hive_20230404051850_d5bbff094-f03c-401f-8ce0-da569c8067c1): SELECT * FROM wuser00_airlines.airlines_orc WHERE code IN ("UA", "XX", "PAW")
INFO : Completed executing command(queryId=hive_20230404051850_d5bbff094-f03c-401f-8ce0-da569c8067c1); Time taken: 0.000 seconds
INFO : OK
```

The results table has two columns: 'airlines_orc.code' and 'airlines_orc.description'. The single row returned is highlighted with a red border:

airlines_orc.code	airlines_orc.description
1 UA	United Air Lines Inc.

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");
```

The screenshot shows the Apache Hive interface. In the top navigation bar, there are tabs for 'Hive' (selected), 'Add a name...', and 'Add a description...'. On the right side of the header are three small icons: a line graph, a magnifying glass, and a gear. Below the header, the main area contains a code editor with the following SQL query:

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

Below the code editor, there is a dropdown menu showing 'user_id' and 'wuser00'. A tooltip indicates that 'wuser00' is selected. To the right of the dropdown is a timestamp: '2.39s wuser00_airlines'. Further down, the output of the query execution is shown in a log-like format:

```
INFO : Table wuser00_airlines.airlines_orc stats: [numFiles=2, numRows=1492, totalSize=17484, rawDataSize=0, numFilesErasureCoded=0]  
INFO : Completed executing command(queryId=hive_202308052034_c953dff4-01a0-4200-8f14-0caaf96cea2); Time taken: 1.587 seconds  
INFO : OK
```

At the bottom of the interface, there is a message box with a red border containing the text 'Success.'.

- **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 capabilities 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");
```

The screenshot shows the Cloudera Manager Hive interface. At the top, there are tabs for 'Hive', 'Add a name...', 'Add a description...', and some icons. Below the tabs, a query editor window displays the following code:

```
1 drop table if exists ${user_id}_airlines.airlines_dim_updates;
2 CREATE EXTERNAL TABLE ${user_id}_airlines.airlines_dim_updates(code string, description string) tblproperties("external.table.purge"="true");
3
4 INSERT INTO ${user_id}_airlines.airlines_dim_updates VALUES("UA","Adrenaline Airlines");
5 INSERT INTO ${user_id}_airlines.airlines_dim_updates VALUES("XX","Get Out of My Airway!");
6
7 -- Merge inserted records into Airlines_orc table
8 MERGE INTO ${user_id}_airlines.airlines_orc USING (SELECT * FROM ${user_id}_airlines.airlines_dim_updates) AS s
9   ON s.code = airlines_orc.code
10  WHEN MATCHED THEN UPDATE SET description = s.description
11  WHEN NOT MATCHED THEN INSERT VALUES (s.code,s.description);
12
13 SELECT * FROM ${user_id}_airlines.airlines_orc WHERE code IN ("UA","XX","PAW");
```

The code between lines 7 and 13 is highlighted with a red box. Below the code, there is a dropdown menu set to 'user_id' and a text input field containing 'wuser00'. The output window shows the command being executed and completed successfully:

```
INFO : Executing command(queryId=hive_20230404052650_8e7a0709-430b-4feb-999d-7ca6bdd5a36b): SELECT * FROM wuser00_airlines.airlines_orc WHERE code IN ("UA","XX","PAW")
INFO : Completed executing command(queryId=hive_20230404052650_8e7a0709-430b-4feb-999d-7ca6bdd5a36b): Time taken: 0.004 seconds
INFO : OK
```

At the bottom, a results table is displayed with two columns: 'airlines_orc.code' and 'airlines_orc.description'. The data is as follows:

airlines_orc.code	airlines_orc.description
1 PAW	Paradise Air
2 UA	Adrenaline Airlines
3 XX	Get Out of My Airway!

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 occurred:

- 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_IC EBERG	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 screenshot shows the Cloudera Manager Hive interface. In the top navigation bar, 'Hive' is selected. Below the navigation bar, there are fields for 'Add a name...' and 'Add a description...'. The main area contains the DDL code for creating the 'flights' table. The code includes dropping the table if it exists, creating an external table with specific columns and types, partitioning by year, and storing the data in ICEBERG format as PARQUET. The final line of the code is 'SHOW CREATE TABLE \${user_id}_airlines.flights;'. Below the code editor, there is a status message indicating the task has started and completed successfully. At the bottom, there are tabs for 'Query History', 'Saved Queries', and 'Results (53)', with 'createtab_stmt' selected.

```

1 drop table if exists ${user_id}_airlines.flights;
2 CREATE EXTERNAL TABLE ${user_id}_airlines.flights (
3   month int,
4   dayofmonth int,
5   deptime int,
6   crsdeptime int,
7   arrtime int,
8   dayofweek int,
9   uniquecarrier string,
10  flightnum int,
11  tailnum string,
12  actualElapsedTime int,
13  crsElapsedTime int,
14  airtime int,
15  arrDelay int,
16  depDelay int,
17  origin string,
18  dest string,
19  distance int,
20  taxiIn int,
21  taxiOut int,
22  cancelled int,
23  cancellationCode string,
24  diverted string,
25  carrierDelay int,
26  weatherDelay int,
27  nasDelay int,
28  securityDelay int,
29  lateAircraftDelay int
30 )
31 PARTITIONED BY (year int)
32 STORED BY ICEBERG
33 STORED AS PARQUET
34 tblproperties ('format-version'='2');
35
36 SHOW CREATE TABLE ${user_id}_airlines.flights;

```

INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId=hive_20230404063552_0cfe530b-2d16-4747-ab64-f0d9860b72d9); Time taken: 0.052 seconds
INFO : OK

Query History Saved Queries Results (53)

createtab_stmt

- 1 CREATE EXTERNAL TABLE `wuser00_airlines`.`flights`(`
- 2 `month` int,
- 3 `dayofmonth` int,

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.

```

createtab_stmt
24   `diverted` string,
25   `carrierdelay` int,
26   `weatherdelay` int,
27   `nasdelay` int,
28   `securitydelay` int,
29   `lateaircraftdelay` int,
30   `year` int)
31 PARTITIONED BY SPEC (
32   year)
33 ROW FORMAT SERDE
34   'org.apache.iceberg_mr.hive.HiveIcebergSerDe'
35 STORED BY
36   'org.apache.iceberg_mr.hive.HiveIcebergStorageHandler'
37
38 LOCATION
39   's3a://.../airlines/airlines_raw/flights'
40 TBLPROPERTIES (
38 LOCATION
39   's3a://...'                                     airlines.db/flights'
40 TBLPROPERTIES (
41   'bucketing_version'=2,
42   'engine.hive.enabled'=true,
43   'format-version'=2,
44   'iceberg.orc.files.only'=false,
45   'metadata_location'='s3a://...'                  'wuser00_airlines.db/flights/metadata/00000-1725f91f-d3f2-4006-9d6b-6bcee2ac1856.metadata.json'
46   'serialization.format'=1,
47   'table_type'=ICEBERG,
48   'transient_lastDdlTime'=1680590148,
49   'uuid'=5d2d0930-8eb3-405c-b423-58c112a64cf8,
50   'write.delete.mode'=merge-on-read',
51   'write.format.default'=parquet,
52   'write.merge.mode'=merge-on-read',
53   'write.update.mode'=merge-on-read')

```

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;

```

The screenshot shows the Cloudera Manager Hive interface. At the top, there are tabs for 'Hive' and 'Add a name...', and buttons for 'Add a description...' and 'Run'. On the right, it shows '4m, 42s wuser00_airlines' with a dropdown arrow, a gear icon, and a question mark icon. Below the tabs, there's a code editor with the following SQL:

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

Below the code editor, there's a dropdown menu showing 'user_id' and 'wuser00'. The main pane displays the execution log:

```
INFO : Table wuser00_airlines.flights stats: [numFiles=12, numRows=71826380, totalSize=1149101435, rawDataSize=0, numFilesErasureCoded=0]
INFO : Completed executing command(queryId=hive_20230404064635_5174a97d-8eb3-4246-9db9-55dacfdc1fa4); Time taken: 281.849 seconds
INFO : OK
```

At the bottom, a message says 'Success.'

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;
```

The screenshot shows the Cloudera Hive interface. At the top, there is a search bar with 'Hive' selected, and fields to 'Add a name...' and 'Add a description...'. Below the search bar, a query is being run:

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

The user ID is set to 'wuser00'. The status bar indicates '4.76s wuser00_airlines'.

Below the query editor, the log output shows:

```
INFO : RAW_INPUT_SPLITS_Map_1: 15  
INFO : Completed executing command(queryId=hive_20230404065337_4fd11f2-4bf4-4eb0-860e-ecfb78395293); Time taken: 4.021 seconds  
INFO : OK
```

At the bottom, there are tabs for 'Query History', 'Saved Queries', and 'Results (12)'. The 'Results (12)' tab is selected, displaying a table with two columns: 'year' and '_c1'. The first row, '1 2006', is highlighted with a red box. The data is as follows:

year	_c1
1 2006	7141922
2 2005	7140596
3 2004	7129270
4 2003	6488540
5 2002	5271359
6 2001	5967780
7 2000	5683047
8 1999	5527884
9 1998	5384721
10 1997	5411843
11 1996	5351983
12 1995	5327435

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

The screenshot shows the Cloudera Metastore UI. On the left, there is a sidebar with a back arrow and the database name 'apac100_airlines'. In the center, there is a 'Tables' section with a '(5)' indicator and a '+' button. Below it is a 'Filter...' input field. A list of five tables is displayed:

- airlines_dim_updates
- airlines_orc
- airports
- flights
- planes

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

The screenshot shows the Cloudera Manager interface. At the top, there is a service card for '1-10impala'. The card includes the service name, status ('Running'), and environment ('emeaworkshop-environ-default'). In the top right corner of the card, there is a red box highlighting the 'HUE' icon. Below the card, there is a summary bar with metrics: EXECUTORS (2), TOTAL CORES (83), TOTAL MEMORY (620 GB), and TYPE (IMPALA, UNIFIED ANALYTICS). The 'UNIFIED ANALYTICS' tab is selected, indicated by a blue background.

Make sure that 'Unified Analytics' is NOT selected.

The screenshot shows the Hue interface. At the top, there is a search bar labeled 'Search data and saved documents...'. Below it, there is a navigation bar with icons for folder, plus, and refresh. The 'Unified Analytics' tab is highlighted with a red box and has a red arrow pointing to it from the text below. To the right of the tab, there are buttons for 'Add a name...' and 'Add a description...'. Below the tabs, there is a text input field containing the placeholder 'Example: SELECT * FROM tablename, or press CTRL + space'. A red annotation text 'Instead of Unified Analytics we should have IMPALA' is overlaid on the interface.

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

Unified Analyti

< default

Tables (1) +

Filter...

customer

Example: SELECT *

Query History

Search data and saved documents...

Impala Add a name... Add a description...

Example: SELECT * FROM tablename, or press CTRL + space

Query History Saved Queries

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 screenshot shows the Cloudera Impala Query Editor interface. At the top, there's a toolbar with icons for file operations and a search bar labeled "Add a name... Add a description...". Below the toolbar, a status bar displays "0.82s wuser00_airlines_raw". The main area contains a SQL query:

```
1 SELECT model,
2        engine_type
3   FROM ${user_id}_airlines.planes
4  WHERE planes.tailnum IN
5      (SELECT tailnum
6         FROM
7            (SELECT tailnum,
8                   count(*),
9                   avg(depdelay) AS avg_delay,
10                  max(depdelay),
11                  avg(taxiout),
12                  avg(cancelled),
13                  avg(weatherdelay),
14                  max(weatherdelay),
15                  avg(nasdelay),
16                  max(nasdelay),
17                  avg(securitydelay),
18                  max(securitydelay),
19                  avg(lateaircraftdelay),
20                  max(lateaircraftdelay),
21                  avg(airtime),
22                  avg(actualelapsedtime),
23                  avg(distance)
24             FROM ${user_id}_airlines.flights
25            WHERE tailnum IN ('N194JB',
26                               'N906S',
27                               'N575ML',
28                               'N852NW',
29                               'N000AA')
30          GROUP BY tailnum) AS delays);
```

Below the query, there's a parameter input field with "user_id" set to "wuser00". The status bar also shows two completed queries: "Query 4f45e4ae289772aa:3068034900000000 100% Complete (13 out of 13)" and "Query 4f45e4ae289772aa:3068034900000000 100% Complete (13 out of 13)".

At the bottom, there's a "Results (1)" tab selected, showing a single row of data:

model	engine_type
A330-223	Turbo-Fan

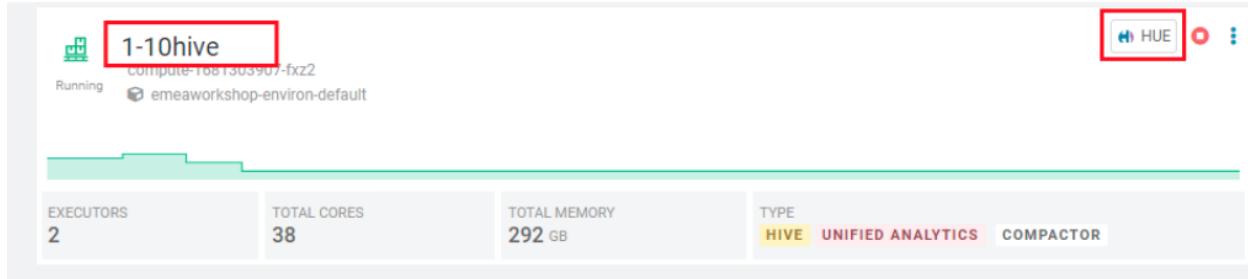
The Data Lakehouse DW query performs significantly better than the same query running against the CSV data.

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;
```

The screenshot shows a Hive interface with the following DDL statements:

```
1 ALTER TABLE ${user_id}_airlines.flights
2 SET PARTITION spec ( year, month );
3
4 SHOW CREATE TABLE ${user_id}_airlines.flights;
```

Below the code, there is a dropdown menu with the value "user_id" selected. The output log shows:

```
INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId=hive_20230404084526_b94698fb-f846-4a44-8a50-69e54bbca706); Time taken: 0.029 seconds
INFO : OK
```

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

The screenshot shows a table definition with the following rows:

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;
```

The screenshot shows the Cloudera Manager Hive Editor interface. At the top, there are tabs for 'Hive' (selected), 'Add a name...', and 'Add a description...'. On the right side, there are icons for saving, running, and more. Below the tabs, a code editor window displays the following SQL query:

```

1| INSERT INTO ${user_id}_airlines.flights
2| SELECT * FROM ${user_id}_airlines_raw.flights_csv
3| WHERE year = 2007;
4|

```

Below the code editor, a results panel shows the output of the query:

```

INFO : HS2 Host: [hiveserver2-0], Query ID: [hive_20230404085305_f33f8481-16f2-4dbe-be2e-c2f10b8eedd5], Dag ID: [dag_1680522906415_0001_18], DAG Session ID: [application_1680522906415_0001]
INFO : Status: Running (Executing on YARN cluster with App id application_1680522906415_0001)

```

At the bottom of the results panel, a message indicates success:

✓ Success.

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

The screenshot shows the Cloudera Manager Impala Editor interface. At the top, there is a search bar labeled 'Search data and saved documents...' and a navigation bar with tabs for 'Add a name...' and 'Add a description...'. Below the navigation bar, there is a button labeled 'Impala' which is highlighted with a red box. To the left of the main area, there is a sidebar with a folder icon and a '(4)' indicating the number of saved queries. The main area contains a code editor with the following example query:

```

1| Example: SELECT * FROM tablename, or press CTRL + space

```

At the bottom of the interface, there are tabs for 'Query History' (selected) and 'Saved Queries'.

- 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.

The screenshot shows the HUE Impala Editor interface with two queries and their corresponding EXPLAIN plans.

Query 1 (Top):

```

1 SELECT year, month, count(*)
2 FROM ${user_id}.airlines.flights
3 WHERE year = 2006 AND month = 12
4 GROUP BY year, month
5 ORDER BY year desc, month asc;
6
  
```

Query 2 (Bottom):

```

1 SELECT year, month, count(*)
2 FROM ${user_id}_airlines.flights
3 WHERE year = 2006 AND month = 12
4 GROUP BY year, month
5 ORDER BY year desc, month asc;
6
  
```

EXPLAIN Plan (Bottom Query):

```

Max Per-Host Resource Reservation: Memory=86.00MB Threads=3
Per-Host Resource Estimates: Memory=563MB
Dedicated Coordinator Resource Estimate: Memory=210MB
WARNING: The following tables are missing relevant table and/or column statistics.
wuser00_airlines.flights

PLAN-ROOT SINK
|
05:MERGING-EXCHANGE [UNPARTITIONED]
|   order by: `year` DESC, `month` ASC
|
02:SORT
|   order by: `year` DESC, `month` ASC
|   row-size=16B cardinality=7.14M
|
04:AGGREGATE [FINALIZE]
|   output: count:merge(*)
|   group by: `year`, `month`
|   row-size=16B cardinality=7.14M
|
03:EXCHANGE [HASH(`year`, `month`)]
|
01:AGGREGATE [STREAMING]
|   output: count(*)
|   group by: `year`, `month`
|   row-size=16B cardinality=7.14M
|
00:SCAN S3 [wuser00_airlines.flights]
|   S3 partitions=1/1 files=1 size=138.00MB
|   predicates: `month` = 12, `year` = 2006
|   row-size=8B cardinality=7.14M
  
```

The EXPLAIN plan highlights the S3 scan operation, showing it reads from a single partition with 1 file of size 138.00MB, using predicates to filter for month 12 and year 2006, and a row size of 8B with a cardinality of 7.14M.

- 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;
```

The screenshot shows the Impala UI interface. At the top, it says "Impala" and has fields for "Add a name..." and "Add a description...". Below the editor area, there are tabs for "Query History", "Saved Queries", and "Explain". The "Explain" tab is selected, displaying the execution plan. The plan starts with a "SCAN S3" node, which is highlighted with a red box. This node has attributes: "S3 partitions=1/1 files= 1 size=10.28MB predicates: 'month' = 12, 'year' = 2007". Following this are several stages: "EXCHANGE [HASH('year', 'month')]", "AGGREGATE [STREAMING]", "SORT", "MERGING-EXCHANGE [UNPARTITIONED]", and finally "PLAN-ROOT SINK". The plan also includes resource information: "Max Per-Host Resource Reservation: Memory=86.00MB Threads=3", "Per-Host Resource Estimates: Memory=308MB", "Dedicated Coordinator Resource Estimate: Memory=119MB", and a warning about missing statistics for the "wuser00_airlines.flights" table.

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;
```

The screenshot shows the Impala UI interface. At the top, there are tabs for 'Impala' (selected), 'Add a name...', and 'Add a description...'. Below the tabs, a code editor displays the following SQL query:

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

Below the code editor, there is a dropdown menu with the value 'user_id' set to 'wuser00'. The main pane shows the execution status of the query:

```
Query d841b67438da05e5:bd899ebc00000000. 0% Complete (0 out of 238)
Query d841b67438da05e5:bd899ebc00000000: 81% Complete (194 out of 238)
Query d841b67438da05e5:bd899ebc00000000 100% Complete (238 out of 238)
Query d841b67438da05e5:bd899ebc00000000 100% Complete (238 out of 238)
Query d841b67438da05e5:bd899ebc00000000 100% Complete (238 out of 238)
```

At the bottom of the main pane, a message indicates success:

✓ Success.

To see snapshots, execute the following SQL.

Unset

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

Impala Add a name... Add a description... 0.79s wuser00_airlines_raw ?

1 DESCRIBE HISTORY \${user_id}_airlines.flights;

No logs available at this moment.

3a4f752e6935868e:6670256e00000000

	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

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**

```

1 -- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
2 SELECT year, count(*)
3 FROM ${user_id}_airlines.flights
4 FOR SYSTEM_TIME AS OF '${create_ts}'
5 GROUP BY year
6 ORDER BY year desc;
7
8 -- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
9 SELECT year, count(*)
10 FROM ${user_id}_airlines.flights
11 FOR SYSTEM_VERSION AS OF ${snapshot_id}
12 GROUP BY year
13 ORDER BY year desc;
14

```

- 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 highlighted query only (1st query).

```

1 -- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
2 SELECT year, count(*)
3 FROM ${user_id}_airlines.flights
4 FOR SYSTEM_TIME AS OF '${create_ts}'
5 GROUP BY year
6 ORDER BY year desc;
7
8 -- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
9 SELECT year, count(*)
10 FROM ${user_id}_airlines.flights
11 FOR SYSTEM_VERSION AS OF ${snapshot_id}
12 GROUP BY year
13 ORDER BY year desc;
14

```

year	count(*)
1 2006	7141922
2 2005	7140596
3 2004	7129270
4 2003	6488540

- 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).

```

1 -- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
2 SELECT year, count(*)
3 FROM ${user_id}_airlines.flights
4 FOR SYSTEM_TIME AS OF '${create_ts}'
5 GROUP BY year
6 ORDER BY year desc;
7
8 -- SELECT DATA USING TIMESTAMP FOR SNAPSHOT
9 SELECT year, count(*)
10 FROM ${user_id}_airlines.flights
11 FOR SYSTEM_VERSION AS OF ${snapshot_id}
12 GROUP BY year
13 ORDER BY year desc;
14

```

user_id wuser00 create_ts snapshot_id 6341506406760449831

Query 1c4288de9297ae5e:3b6cc85c00000000 100% Complete (24 out of 24)
 Query 1c4288de9297ae5e:3b6cc85c00000000 100% Complete (24 out of 24)
 Query 1c4288de9297ae5e:3b6cc85c00000000 100% Complete (24 out of 24)

year	count(*)
1 2007	7453215
2 2006	7141922
3 2005	7140596
4 2004	7129270
5 2003	6488540

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

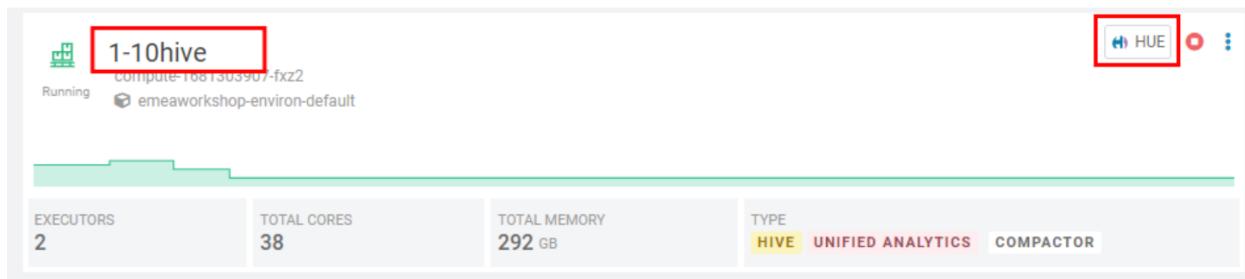
As time passes it might make sense to expire old Snapshots, instead of the Snapshot ID you use the Timestamp to expire old Snapshots. You can do this manually by running a simple ALTER TABLE command as follows. We will NOT RUN this command in this lab.

```
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;
```

```

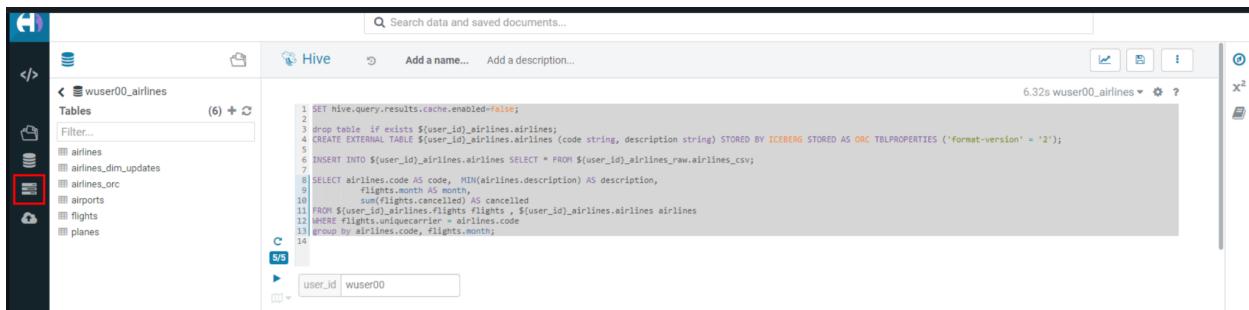
SELECT airlines.code AS code, MIN(airlines.description) AS
description,
flights.month AS month,
sum(flights.cancelled) AS cancelled
FROM ${user_id}_airlines.flights flights ,
${user_id}_airlines.airlines airlines
WHERE flights.uniquecarrier = airlines.code
group by airlines.code, flights.month;

```

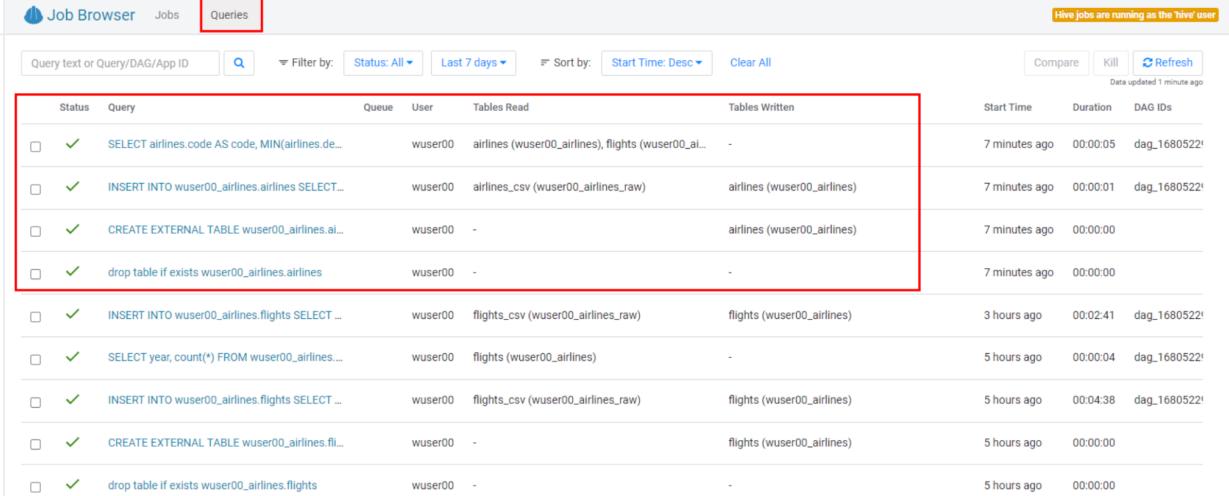
Note: Hive has built in performance improvements, such as a **Query Cache** that stores results of queries run so that similar queries don't have to retrieve data, they can just get the results from the cache. In this step we are turning that off using the SET statement, this will ensure when we look 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.

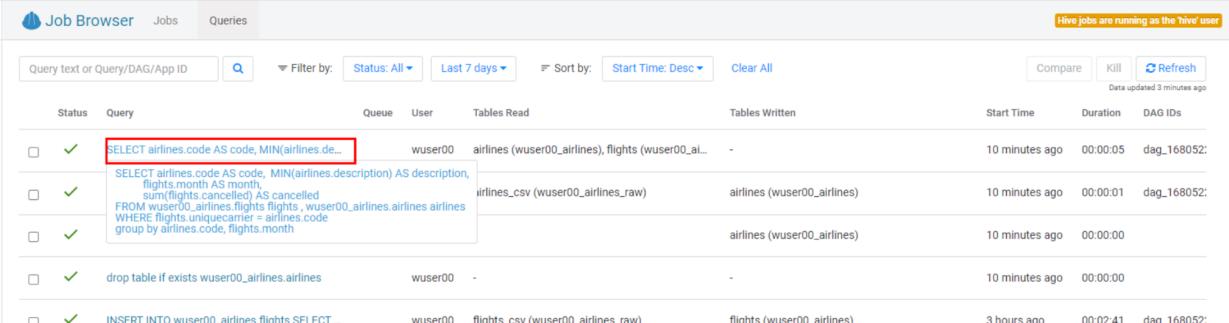


- Then click on Queries. This is where an Admin will go when he wants to investigate the queries. In our case for this lab, we'd like to look at the query we just executed to see how it ran and the steps taken to execute the query. Administrators would also be able to perform other monitoring and maintenance 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.



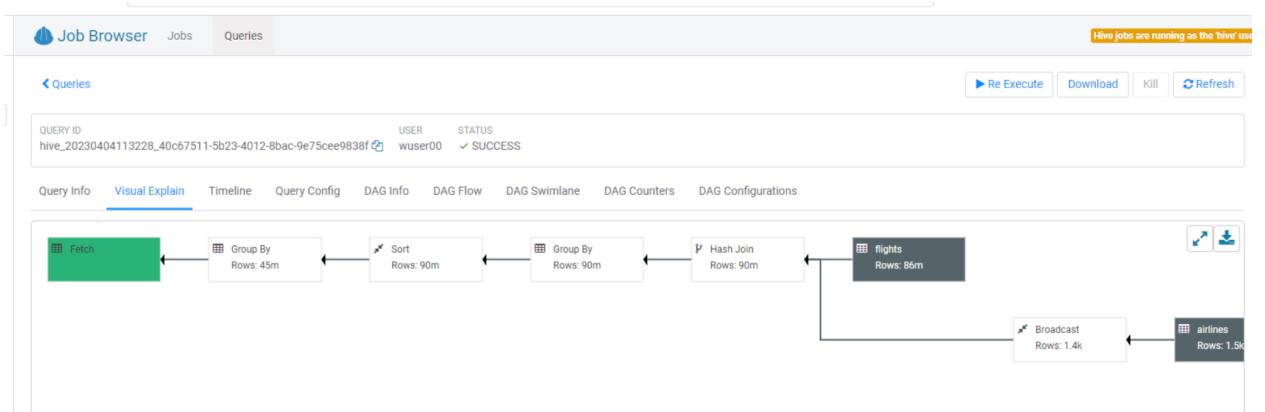
Status	Query	Queue	User	Tables Read	Tables Written	Start Time	Duration	DAG IDs
<input type="checkbox"/>	✓ SELECT airlines.code AS code, MIN(airlines.de...	wuser00	airlines (wuser00_airlines), flights (wuser00_ai...	-	-	7 minutes ago	00:00:05	dag_1680522
<input type="checkbox"/>	✓ INSERT INTO wuser00_airlines.airlines SELECT...	wuser00	airlines_csv (wuser00_airlines_raw)	airlines (wuser00_airlines)	-	7 minutes ago	00:00:01	dag_1680522
<input type="checkbox"/>	✓ CREATE EXTERNAL TABLE wuser00_airlines.ai...	wuser00	-	airlines (wuser00_airlines)	-	7 minutes ago	00:00:00	
<input type="checkbox"/>	✓ drop table if exists wuser00_airlines.airlines	wuser00	-	-	-	7 minutes ago	00:00:00	
<input type="checkbox"/>	✓ INSERT INTO wuser00_airlines.flights SELECT ...	wuser00	flights_csv (wuser00_airlines_raw)	flights (wuser00_airlines)	-	3 hours ago	00:02:41	dag_1680522
<input type="checkbox"/>	✓ SELECT year, count(*) FROM wuser00_airlines...	wuser00	flights (wuser00_airlines)	-	-	5 hours ago	00:00:04	dag_1680522
<input type="checkbox"/>	✓ INSERT INTO wuser00_airlines.flights SELECT ...	wuser00	flights_csv (wuser00_airlines_raw)	flights (wuser00_airlines)	-	5 hours ago	00:04:38	dag_1680522
<input type="checkbox"/>	✓ CREATE EXTERNAL TABLE wuser00_airlines.fli...	wuser00	-	flights (wuser00_airlines)	-	5 hours ago	00:00:00	
<input type="checkbox"/>	✓ drop table if exists wuser00_airlines.flights	wuser00	-	-	-	5 hours ago	00:00:00	

- 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.

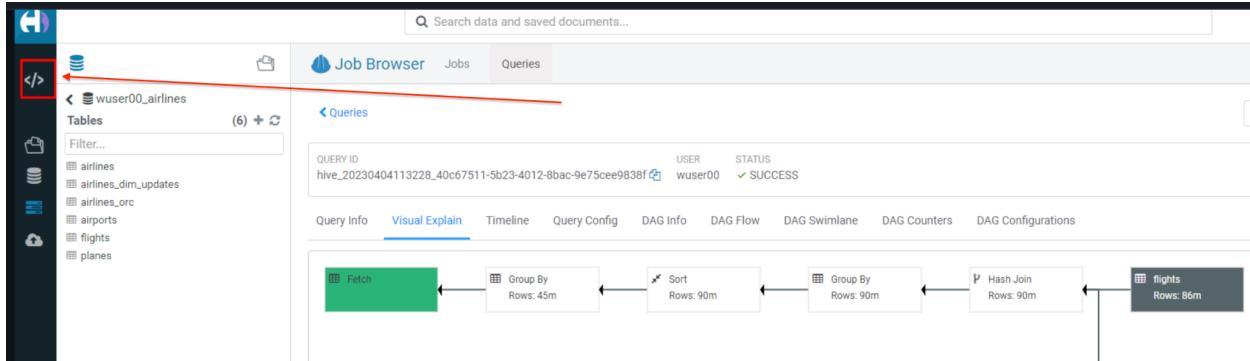


Status	Query	Queue	User	Tables Read	Tables Written	Start Time	Duration	DAG IDs
<input type="checkbox"/>	✓ SELECT airlines.code AS code, MIN(airlines.de...	wuser00	airlines (wuser00_airlines), flights (wuser00_ai...	-	-	10 minutes ago	00:00:05	dag_168052:
<input type="checkbox"/>	✓ SELECT airlines.code AS code, MIN(airlines.description) AS description, flights month AS month, sum(flights.cancelled) AS cancelled FROM wuser00_airlines.flights flights , wuser00_airlines.airlines airlines WHERE flights.uniquecarrier = airlines.code group by airlines.code, flights.month	wuser00	airlines_csv (wuser00_airlines_raw)	airlines (wuser00_airlines)	-	10 minutes ago	00:00:01	dag_168052:
<input type="checkbox"/>	✓ drop table if exists wuser00_airlines.airlines	wuser00	-	-	-	10 minutes ago	00:00:00	
<input type="checkbox"/>	✓ INSERT INTO wuser00_airlines.flights SELECT ...	wuser00	flights_csv (wuser00_airlines_raw)	flights (wuser00_airlines)	-	3 hours ago	00:02:41	dag_168052:

- 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;

```

The screenshot shows the Cloudera Manager Hive interface. In the left sidebar, under 'Tables', there is a table named 'wuser00_airlines'. A red box highlights the table name. The main pane displays the following Hive query:

```

1 DROP MATERIALIZED VIEW IF EXISTS ${user_id}_airlines.traffic_cancel_airlines;
2 CREATE MATERIALIZED VIEW ${user_id}_airlines.traffic_cancel_airlines
3 AS SELECT airlines.code AS code, MIN(airlines.description) AS description,
4   flights.month AS month,
5   sum(flights.cancelled) AS cancelled,
6   count(flights.diverted) AS diverted
7 FROM ${user_id}_airlines.flights flights JOIN ${user_id}_airlines.airlines airlines ON (flights.uniquecarrier = airlines.code)
8 GROUP BY airlines.code, flights.month;
9
10 SHOW MV;
11 SHOW MATERIALIZED VIEWS IN ${user_id}_airlines;
  
```

The status bar at the top right indicates '0.47s wuser00_airlines'. Below the query, a message box shows the execution log:

```

INFO : Starting task [Stage-0:DDL] in serial mode
INFO : Completed executing command(queryId=hive_20230404115330_664d62d2-e9f3-4ca9-ac8c-21f3bca6ec3e); Time taken: 0.871 seconds
INFO : OK
  
```

At the bottom, the 'Results (1)' tab is selected, showing a table with one row:

mv_name	rewrite_enabled	mode	incremental_rebuild
1 traffic_cancel_airlines	Yes	Manual refresh	Available in presence of insert operations only

- 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;
  
```

The screenshot shows the Cloudera Manager Hive interface. At the top, there's a header with 'Hive' and input fields for 'Add a name...' and 'Add a description...'. Below the header is a code editor containing a Hive query:

```

1 SET hive.query.results.cache.enabled=false;
2 SELECT airlines.code AS code, MIN(airlines.description) AS description,
3   flights.month AS month,
4   sum(flights.cancelled) AS cancelled
5 FROM ${user_id}.airlines.flights flights , ${user_id}_airlines.airlines airlines
6 WHERE flights.uniquecarrier = airlines.code
7 group by airlines.code, flights.month
8 order by airlines.code;
9
22
▶ user_id wuser00

```

The status bar at the top right indicates '4.32s wuser00_airlines'.

The middle section displays log messages:

```

INFO : RAW_INPUT_SPLITS_Map_4: 1
INFO : Completed executing command(queryId=hive_2023040115629_29b40cd3-3cda-455a-8eaf-bd8d057d101c); Time taken: 3.783 seconds
INFO : OK

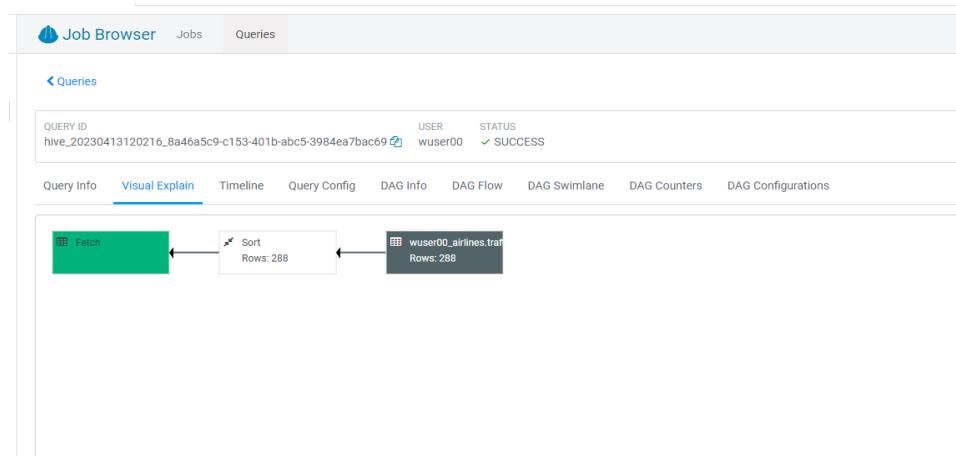
```

The bottom section shows the query results in a table:

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

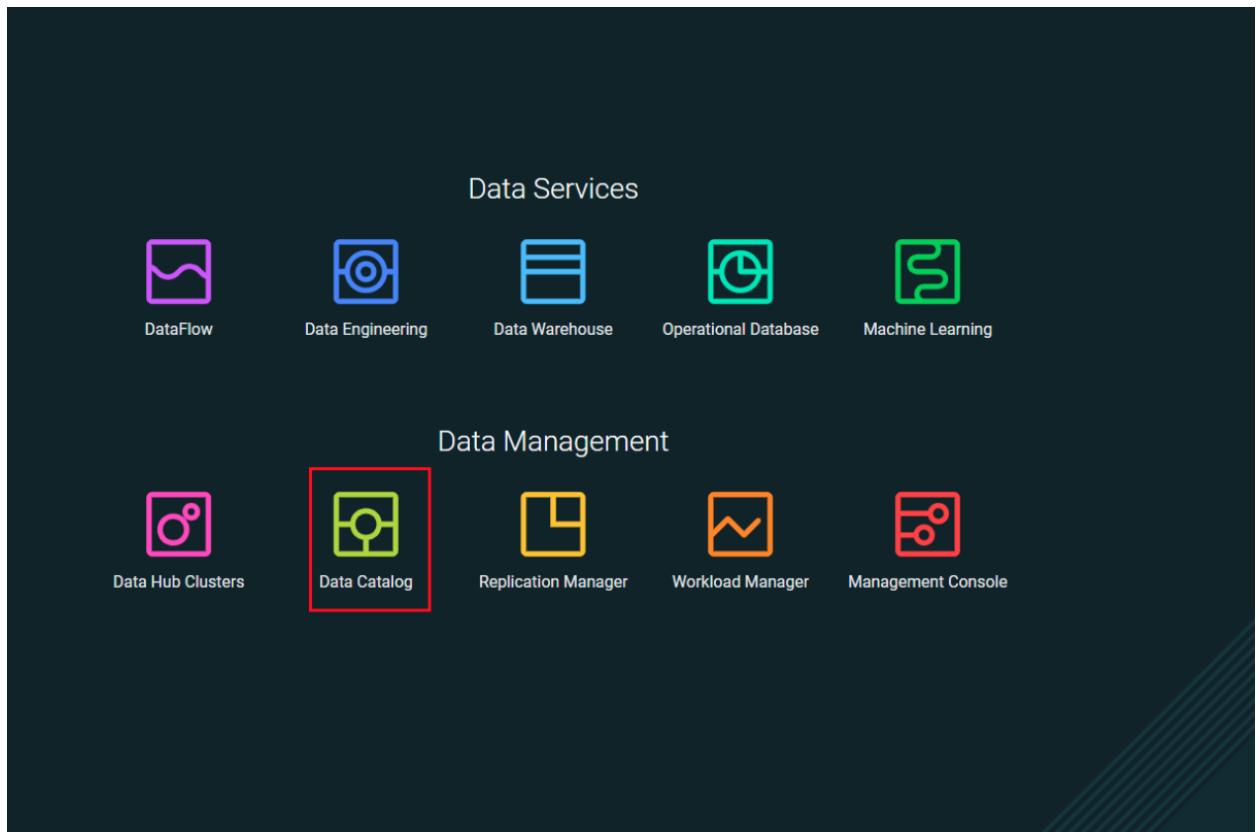
With query rewrite the materialized view is used and the new plan just reads the MV and sorts the data vs. reading flights (86M rows) and airlines (1.5K rows) with hash join, group and sorts. This results in significant reduction in run time for this query.



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>

The screenshot shows the Cloudera Data Catalog interface. On the left is a dark sidebar with navigation links: Search, Datasets, Bookmarks, Profilers, and Atlas Tags. The main area has a search bar at the top. Below it, there's a section titled "Data Lakes" with a dropdown menu showing "aws pko-workshop-dl". A red box highlights this dropdown. Underneath, there's a "Filters" section with a "TYPE" dropdown set to "Hive Table". To the right, there's a summary for "pko-workshop-dl | 877" and a table listing various assets.

Type	Name
AWS S3 V2 Bucket	handsonworkshop
Impala Process Execution	QUERY:apac100_airlines_raw.flights_c
Impala Process Execution	QUERY:host_cz_airlines_raw.flights_ci
Impala Process Execution	QUERY:apac100_airlines_raw.flights_r

- 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.

This screenshot shows the same interface as the previous one, but with a different filter applied. The "TYPE" dropdown in the "Filters" section is now set to "Hive Table", indicated by a red box. The right side of the screen shows a list of "emeaworkshop-environ | 11" assets, all of which are now listed as Hive Tables.

Type	Name	Qualified Name
Hive Table	planes_csv	wuser00_airlines_raw.planes_csv@cm
Hive Table	airports	wuser00_airlines.airports@cm
Hive Table	airports_csv	wuser00_airlines_raw.airports_csv@cm
Hive Table	planes	wuser00_airlines.planes@cm
Hive Table	airlines_dim_updates	wuser00_airlines.airlines_dim_updates@cm
Hive Table	airlines_orc	wuser00_airlines.airlines_orc@cm
Hive Table	flights_csv	wuser00_airlines_raw.flights_csv@cm
Hive Table	airlines_csv	wuser00_airlines_raw.airlines_csv@cm
Hive Table	airlines	wuser00_airlines.airlines@cm
Hive Table	traffic_cancel_airlines	wuser00_airlines.traffic_cancel_airlines@cm
Hive Table	flights	wuser00_airlines.flights@cm

- 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.

The screenshot shows the Cloudera Data Catalog interface. On the left is a sidebar with navigation links: Search, Datasets, Bookmarks, Profilers, and Atlas Tags. The main area is titled "Search" and displays a list of datasets under "Old Lakes". Two datasets are listed: "aws meta-workshop-dl" (10 entries) and "aws emeaworkshop-environ" (11 entries). A green gear icon indicates a "Setup the Profiler for emeaworkshop-environ" task. Below the list are several filter sections: "TYPE" (Hive Table selected), "OWNERS" (dipti.dash, wuser00), "DATABASE" (highlighted with a red box), "CREATED WITHIN" (Last 7 days selected). The results for "emeaworkshop-environ" are shown in a table:

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

This screenshot shows the same Cloudera Data Catalog interface, but the focus is on the "DATABASE" filter section. The sidebar and top navigation are identical. The "DATABASE" section is expanded, showing a list of databases: default, information_schema, salmazan_airlines, sys@cm, and salmazan_airlines_raw. A search bar at the top of the dropdown shows "wuser00". Two database names are highlighted with red boxes: "wuser00" and "wuser00_airlines". To the right of the dropdown, there are "Cancel" and "Add" buttons. The rest of the interface is visible on the right side.

- 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.

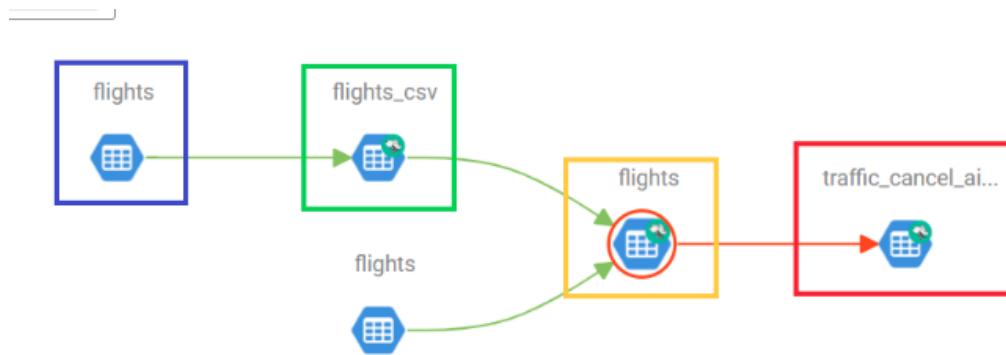
The screenshot shows the Cloudera Data Catalog search interface. The left sidebar has a 'Search' button and links for 'Datasets', 'Bookmarks', 'Profilers', and 'Atlas Tags'. The main area has a 'Setup the Profiler for emeaworkshop-environ' message. The search bar is empty. Below it, the 'emeaworkshop-environ | 11' section shows a table of datasets. The columns are 'Type', 'Name', and 'Qualified Name'. The results include various Hive Tables like 'planes_csv', 'airports', 'airports_csv', etc., and one external table 'flights'. A red box highlights the 'flights' entry, and a red arrow points to the 'flights' link in the 'Name' column.

- 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 screenshot shows the Cloudera Data Catalog Asset Details page for the 'flights' table. The left sidebar has a 'Search' button and links for 'Datasets', 'Bookmarks', 'Profilers', and 'Atlas Tags'. The main area has tabs for 'Asset Details', 'Properties', 'Classifications', and 'Terms'. The 'Properties' tab shows details like Type: HIVE TABLE, # of Columns: 29, Data Lake: emeaworkshop-environ, and Owner: wuser00. The 'Overview' tab is selected, showing lineage information. A lineage graph shows the 'flights' table (blue box) pointing to a 'flights_csv' Hive table (green box), which in turn points to another 'flights' table (orange box). A red box highlights the first 'flights' node in the graph, and a red arrow points to the 'i' icon next to it.

- 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.

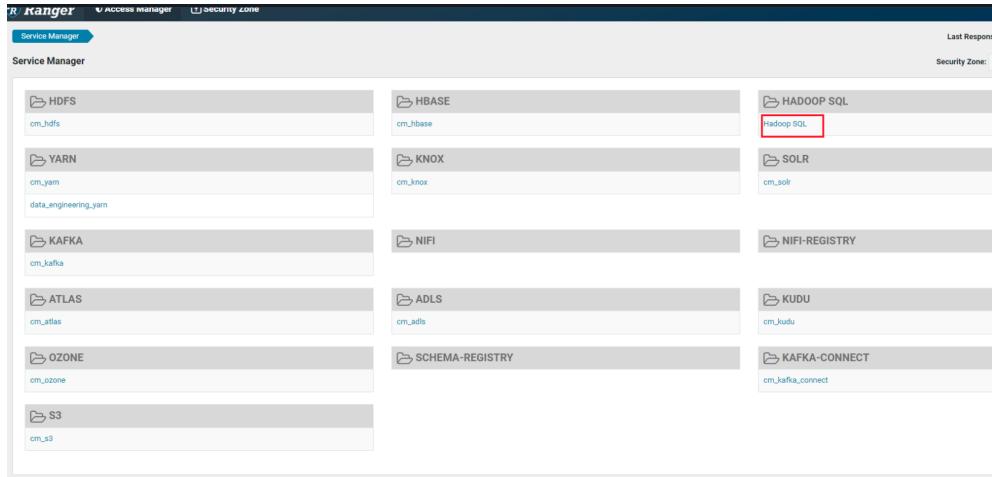


- Click on the Policy tab to see what security policies have been applied on this table. Click on the arrow next all - database, table Policy Name to the number as shown in the screenshot

The screenshot shows the CDP interface for the 'flights' table. At the top, there's a 'Properties' section with details like Type: HIVE TABLE, # of Columns: 29, and Owner: wuser00. Below this is a 'Classifications' section with a '+ Add Classification' button. The main navigation bar at the bottom includes 'Overview', 'Schema', 'Metadata Audits', 'Policy' (which is highlighted with a red box), and 'Access Audits'. Under 'Resource Based Policies', there are two entries:

Policy ID	Policy Name	Status	Audit Logging
9 ↗	all - database, table, column	ENABLED	ENABLED
10 ↗	all - database, table	ENABLED	ENABLED

- 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.

The screenshot shows the Cloudera Ranger Hadoop SQL Policies page. It lists six existing policies:

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
10	all - database, table	-	Enabled	Enabled	--	<code>cl_ranger_admins_3a705090</code> workshop-users	Hive beacon dpprofiler hdp + More...	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
11	all - storage-type, storage-url	-	Enabled	Enabled	--	<code>cl_ranger_admins_3a705090</code> workshop-users	Hive beacon dpprofiler hdp + More...	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
12	all - database	-	Enabled	Enabled	--	<code>cl_ranger_admins_3a705090</code> workshop-users public	Hive beacon dpprofiler hdp + More...	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
13	all - hiveservice	-	Enabled	Enabled	--	<code>cl_ranger_admins_3a705090</code> workshop-users	Hive beacon dpprofiler hdp + More...	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
14	all - database, udf	-	Enabled	Enabled	--	<code>cl_ranger_admins_3a705090</code> workshop-users	Hive beacon dpprofiler hdp + More...	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
15	all - url	-	Enabled	Enabled	--	<code>cl_ranger_admins_3a705090</code> workshop-users	Hive beacon dpprofiler hdp + More...	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Step 1 : Add New Policy

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

The screenshot shows the Cloudera Ranger Hadoop SQL Policies page. The table is empty, indicating no policies have been added. The 'Add New Policy' button is highlighted with a red box.

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action
No Policies found!								

- 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 screenshot shows the 'Create Policy' page for Hadoop SQL Policies. In the 'Policy Details' section, the 'Policy Name' is 'wuser001 Workshop Row Level Filter', 'Policy Type' is 'Row Level Filter', and 'Audit Logging' is set to 'Yes'. In the 'Row Filter Conditions' section, there is one condition: 'Select User' is set to 'wuser001' and 'Access Types' is 'select'. The 'Row Level Filter' field contains 'uniquecarrier="UA"'. At the bottom of the 'Row Filter Conditions' section, there is an 'Add' button, which is highlighted with a red box.

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

The screenshot shows the 'List of Policies' page for Hadoop SQL. The table has columns: Policy ID, Policy Name, Policy Labels, Status, Audit Logging, Roles, Groups, Users, and Action. There is one row with Policy ID 97, Policy Name 'wuser001 Workshop Row Level Filter', Status 'Enabled', Audit Logging 'Enabled', and User 'wuser001'. The 'Action' column for this row contains icons for edit, delete, and preview.

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.

The screenshot shows the Impala Query Editor interface. The query entered is:

```
1 SELECT uniquecarrier, count(*)  
2 FROM ${user_id}_airlines.flights  
3 GROUP BY uniquecarrier;
```

The results pane shows the output of the query:

uniquecarrier	count(*)
1 UA	8821384

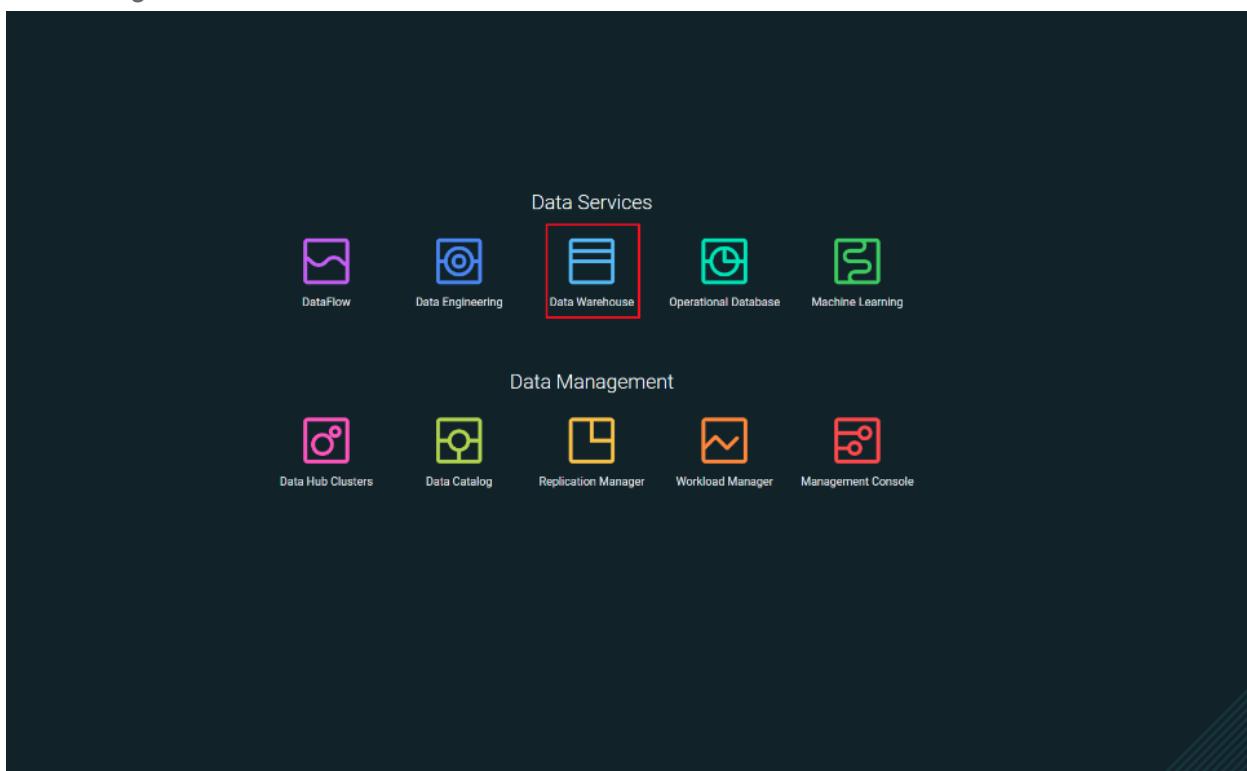
The row for 'UA' is highlighted with a red box. The status bar at the bottom right indicates "0.86s wuser00_airlines".

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

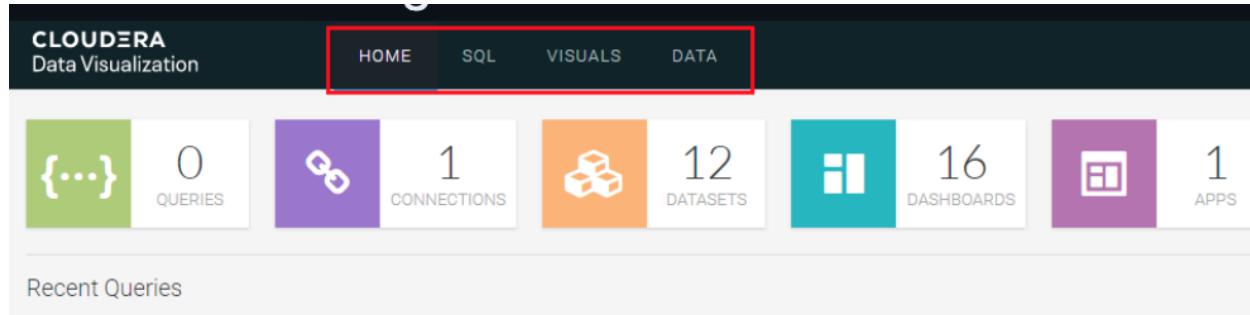
- If you are not on the CDP home page, then go there and click on the following CDW icon to go into Cloudera Data Warehouse.



- 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.

The screenshot shows the Cloudera Data Visualization (CDV) interface. On the left, there's a sidebar with options for Overview, Database Catalogs, Virtual Warehouses, and Data Visualization. The main area is titled 'Data Visualization' with a search bar. A table lists two applications: 'emsworkshop-dataviz' (with ID 'viz-1681366501-58g8') and 'dss-dataviz' (with ID 'viz-1681224602-qw9r'). The 'Data Visualization' column for each row contains a link labeled 'Data VIZ'. The top right of the table has a 'Search' button and an 'ADD NEW' button.

- 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.

This screenshot shows the 'DATA' tab selected in the top banner. On the left, there's a sidebar with 'NEW CONNECTION' and 'All Connections' sections. The main area displays a table of datasets:

Title/Table	ID	Created
wuser01_dataset wuser01_stocks.stock_intraday_1min	19	Mar 10, 2023
wuser07_dataset wuser07_stocks.stock_intraday_1min	18	Mar 10, 2023
wuser25_dataset wuser25_stocks.stock_intraday_1min	17	Mar 09, 2023

- 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

The screenshot shows the Cloudera Data Visualization interface. On the left, there's a sidebar with 'samples' and a list of datasets. In the center, a modal window titled 'Create New Data Connection' is open. The 'Basic' tab is selected. The connection type is set to 'CDW Impala'. The connection name is 'wuser00-airlines-lakehouse' and the CDW Warehouse is '1-10Impala'. Under the 'Basic' tab, the 'Hostname or IP address' is 'coordinator.impala-1681303853-kmdq.svc.cluster.local', the 'Port #' is '28000', and the 'Username' is 'srv_env-kfmnbv'. A red box highlights the 'TEST' button. The 'CONNECT' button is at the bottom right. Below the modal, a green banner says 'Connection Verified'.

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: [redacted]

TEST CONNECT

Connection Verified

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

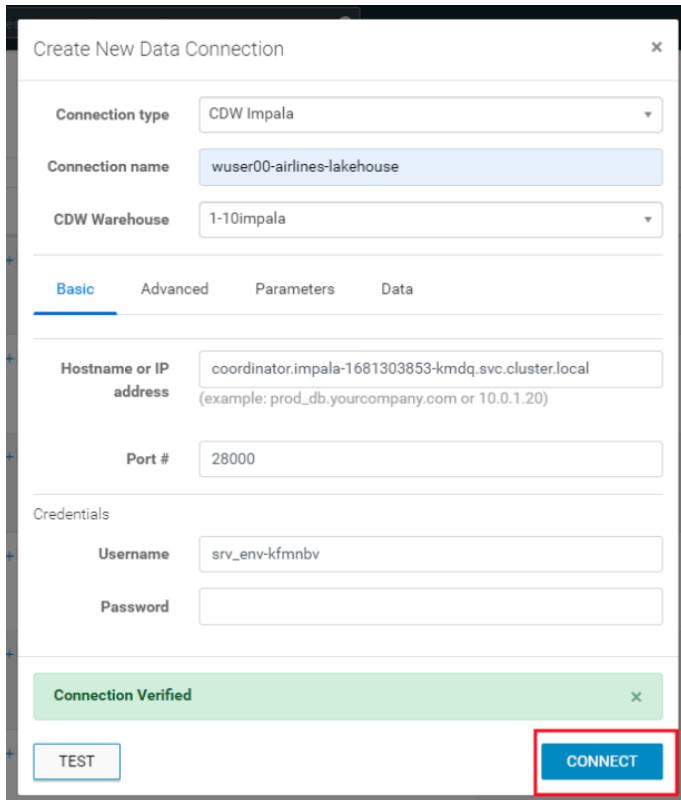
Credentials

Username: srv_env-kfmnbv

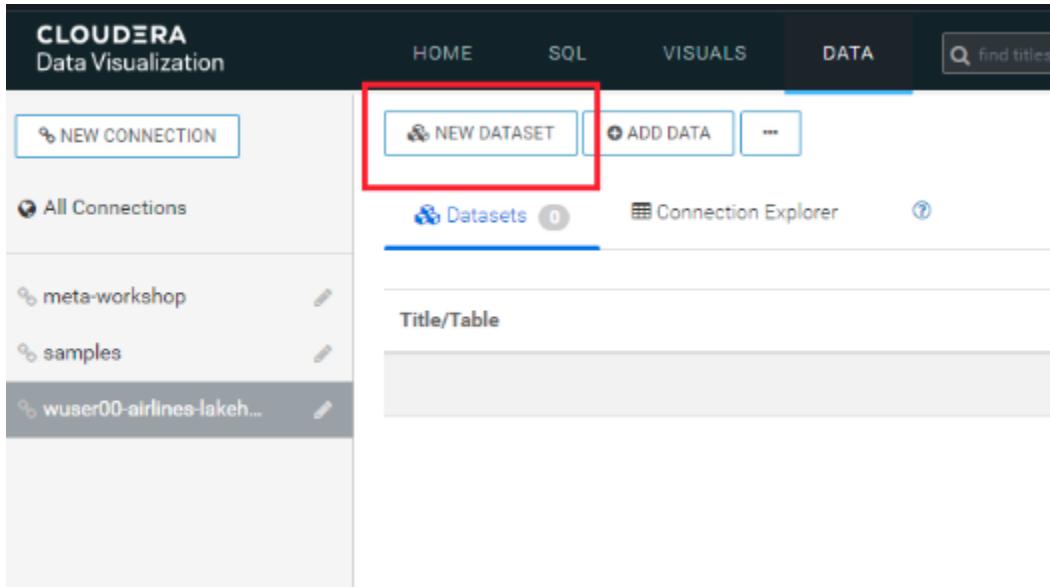
Password: [redacted]

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.

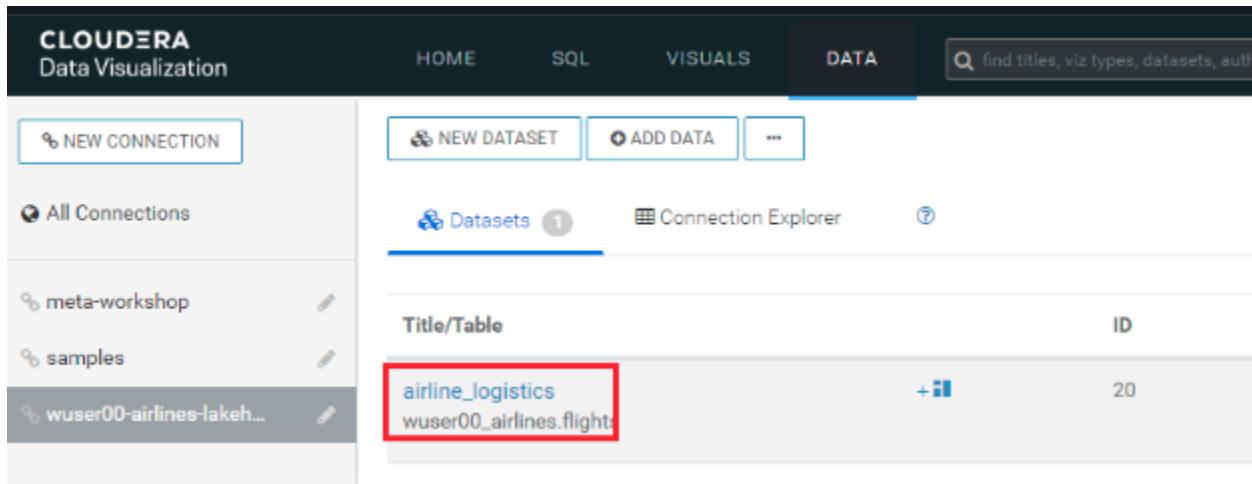
Dataset Source - Select From Table (however, you could choose to directly enter a SQL statement instead).

Select Database - <user_id>_airlines (Make Sure you select the database that is associated with your <user_id>).

Select Table - flights.

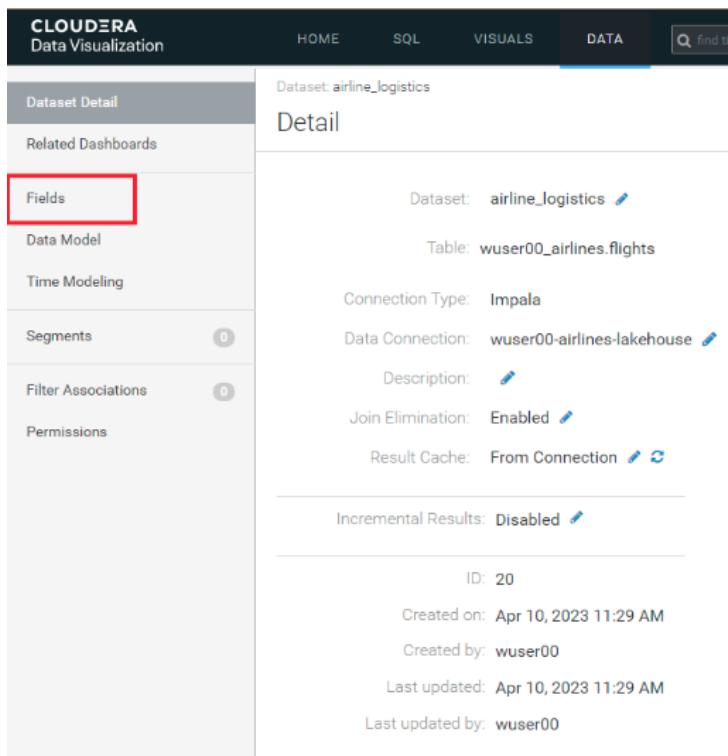
A screenshot of a 'New Dataset' dialog box. The title bar says 'New Dataset'. The main area has a message: 'Create a dataset from data on this connection. You need to create a dataset before you can create dashboards or apps.' Below this, there are four sections: 'Dataset title' (containing 'airline_logistics'), 'Dataset Source' (set to 'From Table'), 'Select Database' (set to 'wuser00_airlines'), and 'Select Table' (set to 'flights'). At the bottom right are 'CANCEL' and 'CREATE' buttons, with 'CREATE' being highlighted by a red box.

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.



The screenshot shows the Cloudera Data Visualization web application. At the top, there's a navigation bar with tabs: HOME, SQL, VISUALS, DATA (which is currently active), and a search bar. Below the navigation bar, there are buttons for NEW CONNECTION, NEW DATASET, ADD DATA, and more. The main area is titled 'Datasets' and shows a table of datasets. One dataset, 'airline_logistics', is highlighted with a red box. The table has columns for 'Title/Table' and 'ID'. The dataset 'airline_logistics' has an ID of 20.

Click on Fields option in the left window pane.



The screenshot shows the 'Dataset Detail' page for the dataset 'airline_logistics'. On the left, there's a sidebar with various options: Dataset Detail (selected), Related Dashboards, Fields (highlighted with a red box), Data Model, Time Modeling, Segments (with 0 items), Filter Associations (with 0 items), and Permissions. The main panel displays dataset details: Dataset: airline_logistics, Table: wuser00_airlines.flights, Connection Type: Impala, Data Connection: wuser00-airlines-lakehouse, Description: (empty), Join Elimination: Enabled, Result Cache: From Connection, Incremental Results: Disabled. It also shows metadata: ID: 20, Created on: Apr 10, 2023 11:29 AM, Created by: wuser00, Last updated: Apr 10, 2023 11:29 AM, Last updated by: wuser00.

The screenshot shows the Cloudera Data Visualization interface. The top navigation bar includes links for HOME, SQL, VISUALS, and DATA, with a search bar. The left sidebar has sections for Dataset Detail, Related Dashboards, Fields (selected), Data Model (highlighted with a red box), Time Modeling, Segments, Filter Associations, and Permissions. The main content area is titled 'Dataset: airline_logistics' and shows 'Fields' with 'EDIT FIELDS' and 'Hide Comments' buttons. It lists 'Dimensions' (flights: uniquecarrier, tailnum, origin, dest, cancellationcode, diverted) and 'Measures' (month, dayofmonth, dayofweek, deptime, crsdeptime, arrtime, crsarrtime, flightnum, actuelapsedtime, crselapsedtime, airtime, ardelay, depdelay, distance, taxin, taxout, cancelled, carrierdelay, weatherdelay, nasdelay, securitydelay, lateaircraftdelay, year).

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

This screenshot shows the same interface as the previous one, but with a red box highlighting the 'Data Model' button in the left sidebar under the 'Fields' section. The rest of the interface remains identical, showing the dataset details and field dimensions/measures.

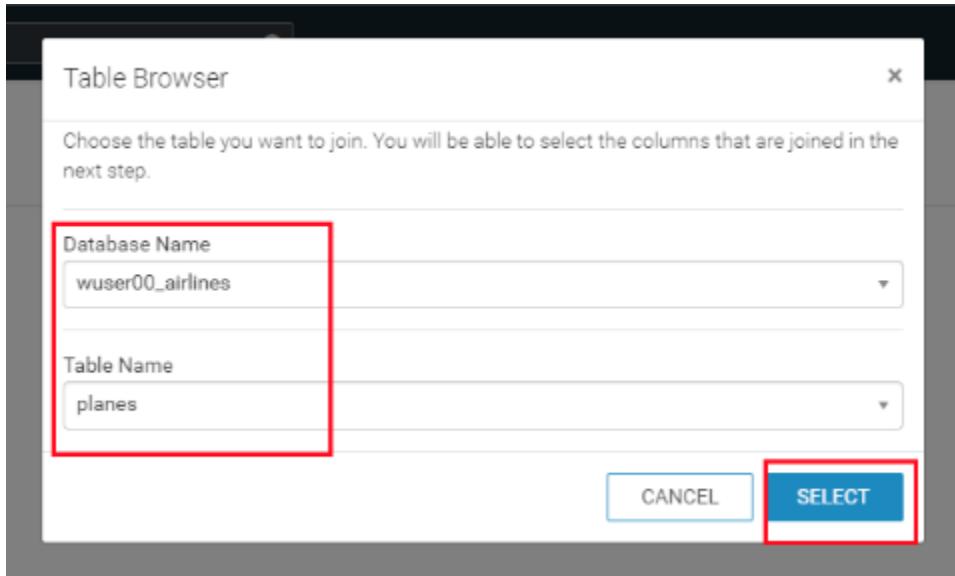
Click on EDIT DATA MODEL.

The screenshot shows the Cloudera Data Visualization interface. The top navigation bar includes links for HOME, SQL, VISUALS, DATA, and a search icon. The left sidebar has sections for Dataset Detail, Related Dashboards, Fields, Data Model (which is selected and highlighted in grey), Time Modeling, Segments (with a '0' badge), Filter Associations (with a '0' badge), and Permissions. The main content area displays the dataset 'airline_logistics'. Under the 'Data Model' section, there is a list item 'flights' with a 'SHOW DATA' button and a checked 'Apply Display Format' checkbox. A red box highlights the 'EDIT DATA MODEL' button at the top of the main content area.

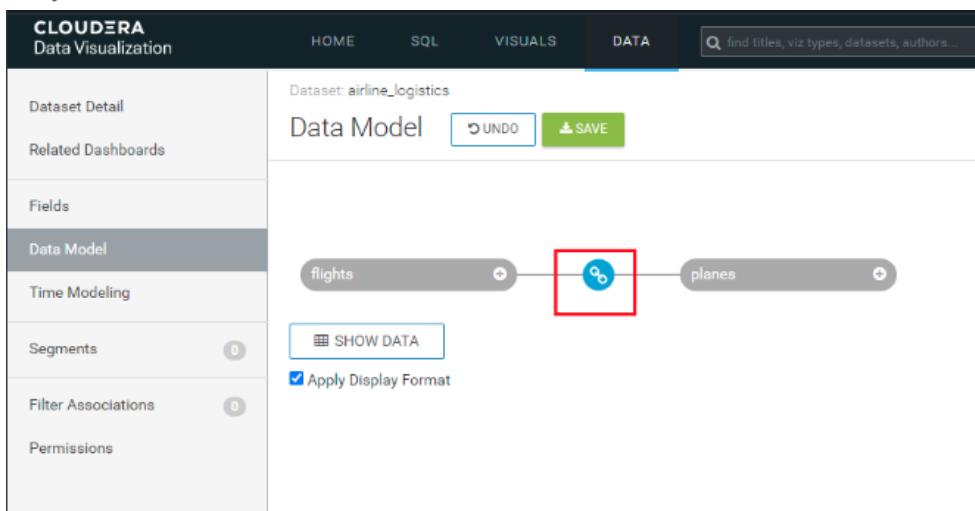
Click on the + icon next to the flights table option.

This screenshot shows the same Cloudera Data Visualization interface as the previous one, but with a different focus. The '+ icon' next to the 'flights' table option in the list is highlighted with a red box. The rest of the interface, including the navigation bar, sidebar, and other content areas, appears identical to the first screenshot.

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



Then click on the join icon 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.



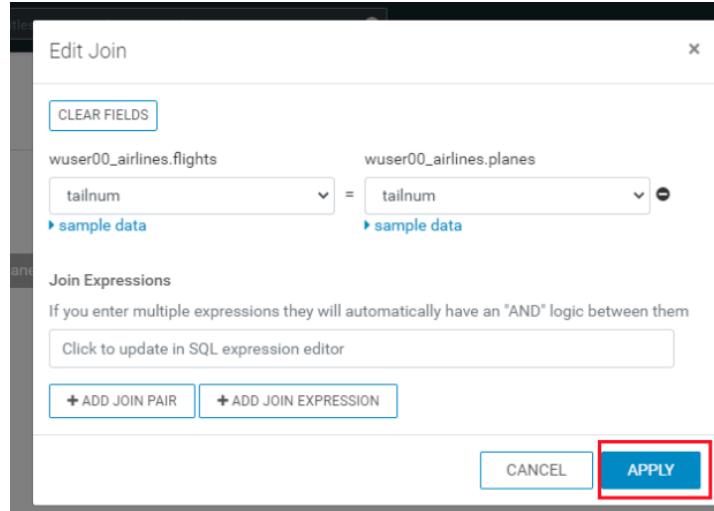
The screenshot shows the Cloudera Data Visualization interface. The top navigation bar includes HOME, SQL, VISUALS, and DATA. A search bar at the top right is set to "find titles, viz types, datasets, authors". The left sidebar has sections for Dataset Detail, Related Dashboards, Fields, Data Model (selected), Time Modeling, Segments (0), Filter Associations (0), and Permissions.

The main area shows a "Dataset: airline_logistics" with a "Data Model" section. It displays a diagram with two nodes: "flights" and "planes" connected by a join line. Below the diagram is a "Join Details" panel with the following settings:

- Join Type:** Left (selected)
- Source Column:** tailnum
- Target Column:** tailnum
- Condition:** year = year

Buttons in the panel include "SHOW DATA", "DELETE JOIN", and "EDIT JOIN".

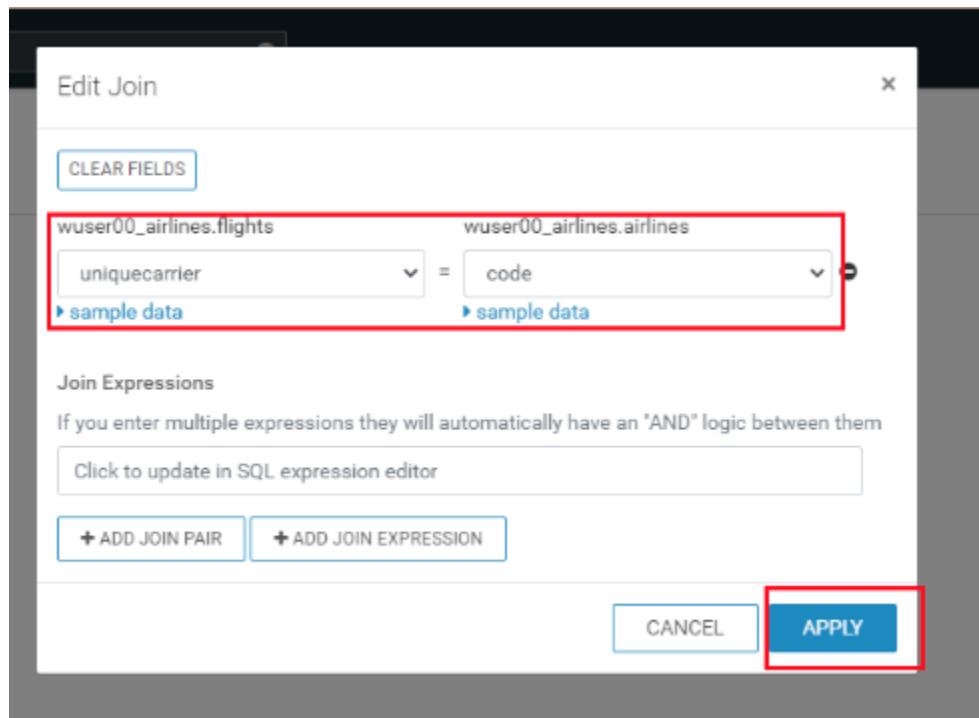
A modal window titled "Edit Join" is open, showing the same join configuration. It includes fields for "wuser00_airlines.flights" and "wuser00_airlines.planes" with "tailnum" and "year" selected. The "year" field in the target column is highlighted with a red box. The modal also contains sections for "Join Expressions" and "Join Editor". Buttons at the bottom of the modal are "CANCEL" and "APPLY".



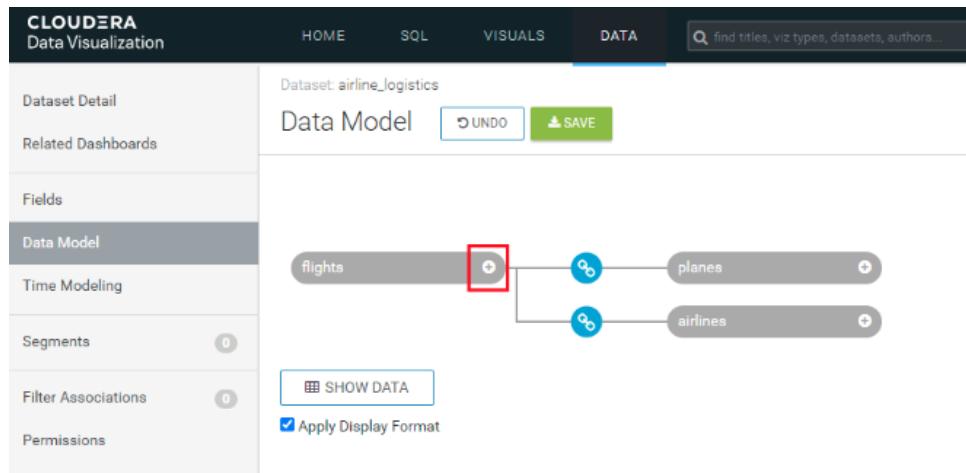
- Now we will create a join between another table. Click on + icon next to flights as shown below. Select the appropriate Database Name based on your <user_id> (Ex: wuser00_airlines) and table name airlines.

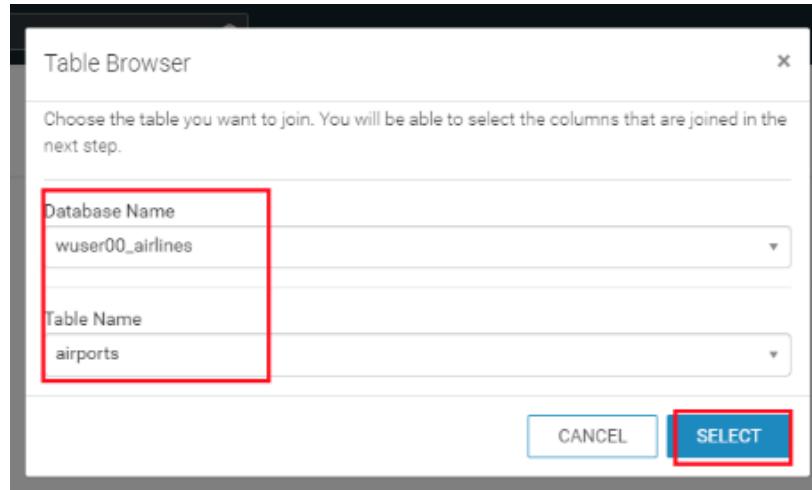
The top screenshot shows the 'Data Model' section of the Cloudera Data Visualization interface. On the left, a sidebar lists 'Dataset Detail', 'Related Dashboards', 'Fields', 'Data Model' (which is selected and highlighted in grey), 'Time Modeling', 'Segments', 'Filter Associations', and 'Permissions'. In the main area, a 'Dataset: airline_logistics' is selected. Below it, the 'Data Model' section shows a diagram with a 'flights' node connected to a 'planes' node via a join line. The 'flights' node has a '+' icon highlighted with a red box. There are 'UNDO' and 'SAVE' buttons above the diagram. Below the diagram are 'SHOW DATA' and 'Apply Display Format' checkboxes. The bottom screenshot shows the 'Table Browser' dialog box. It asks to choose a table to join. It has 'Database Name' set to 'wuser00_airlines' and 'Table Name' set to 'airlines'. At the bottom are 'CANCEL' and 'SELECT' buttons, with 'SELECT' being highlighted with a red box.

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

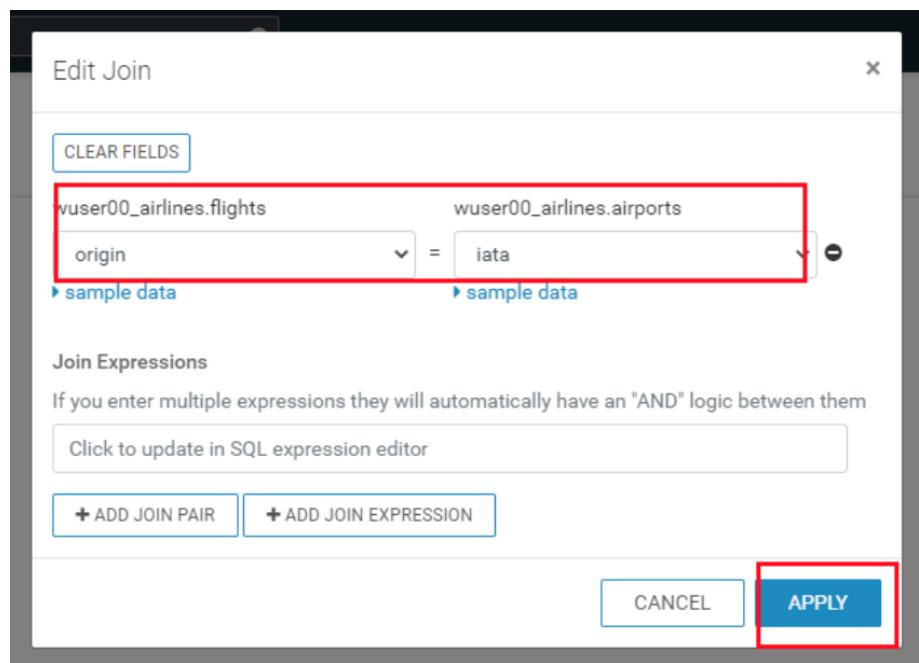


- Click on + icon next to flights as shown below. Select the appropriate Database Name based on your <user_id> (Ex: wuser00_airlines) and table name airports.





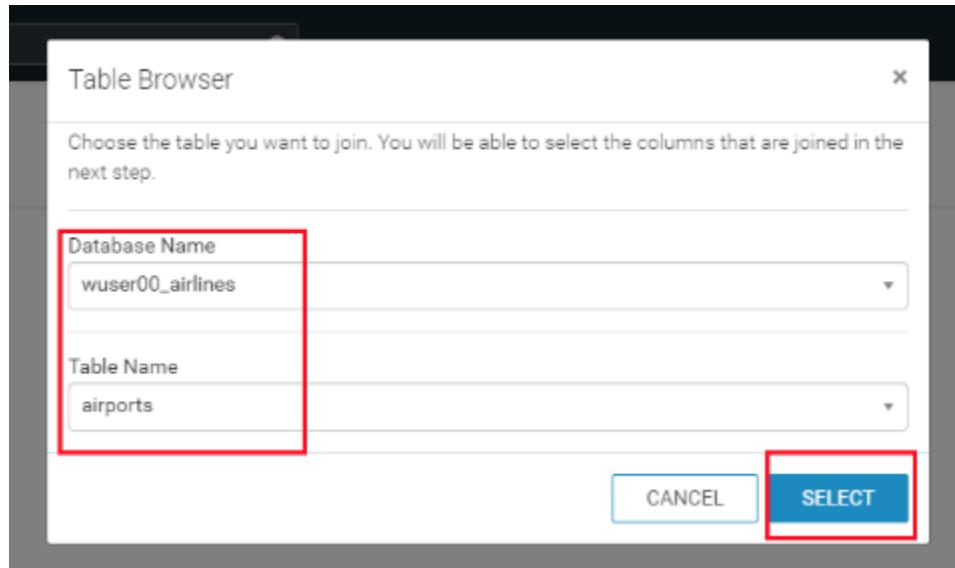
- Make sure you select the column origin from flights and column iata from airports table. Click APPLY.



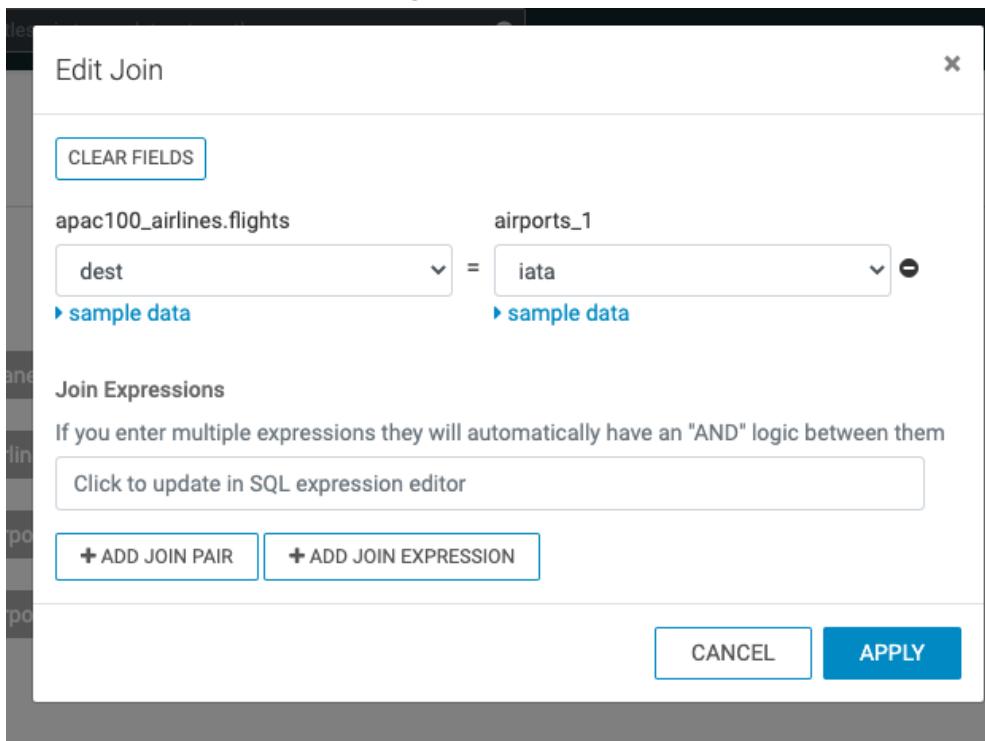
- Click on + icon next to flights as shown below. Select the appropriate Database Name based on your <user_id> (Ex: wuser00_airlines) and table name airports.

The screenshot shows the Cloudera Data Visualization interface. The left sidebar has a 'Data Model' section selected, indicated by a dark grey background. The main area displays a 'Dataset Detail' for 'airline_logistics'. A network diagram shows nodes: 'flights' (highlighted with a red box around its '+'), 'planes', 'airlines', and 'airports'. Below the diagram are buttons for 'SHOW DATA' and 'Apply Display Format'.

- 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

Dataset Detail HOME SQL VISUALS DATA find titles, viz types, datasets, authors

Dataset Detail Related Dashboards Fields Data Model EDIT DATA MODEL

Dataset: airline_logistics

Data Model

```

graph LR
    flights --> planes
    flights --> airlines
    flights --> airports
    flights --> airports_1
  
```

SHOW DATA Apply Display Format

Dataset Detail HOME SQL VISUALS DATA find titles, viz types, datasets, authors

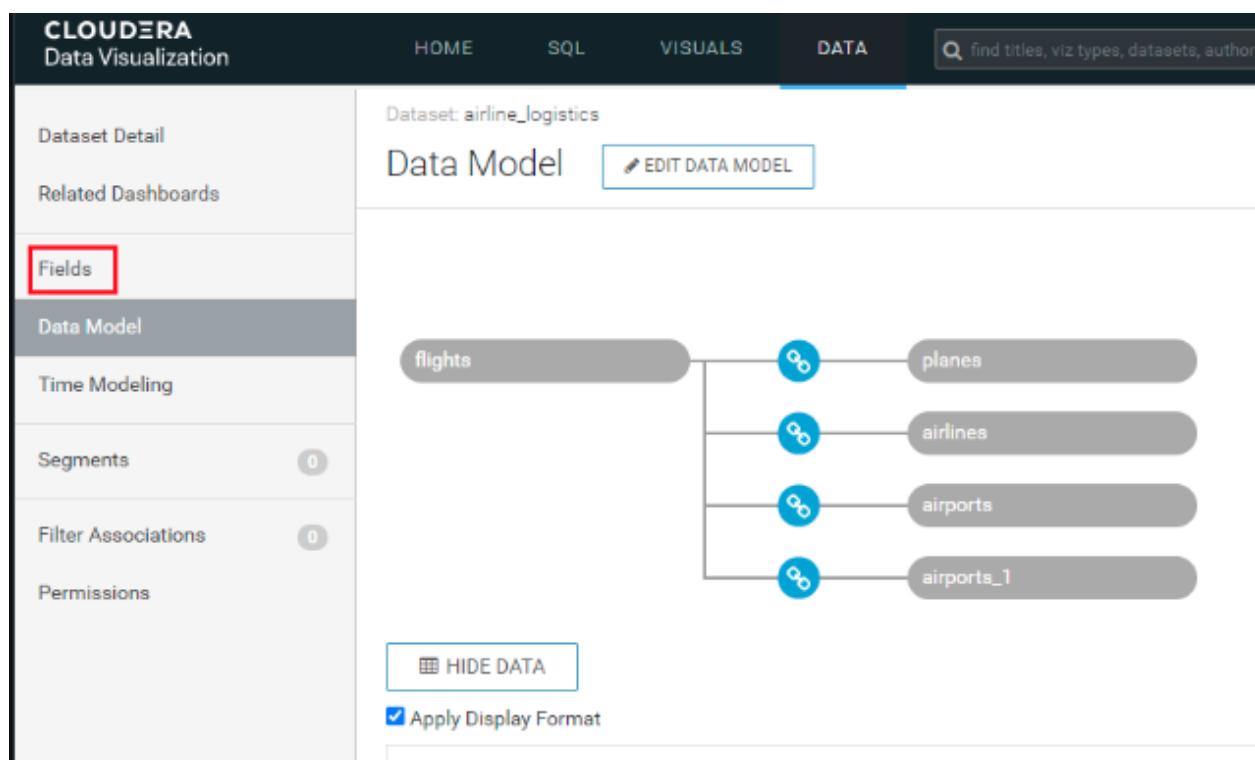
Dataset: airline_logistics UNDO SAVE NEW DASHBOARD

Data Model

HIDE DATA Apply Display Format

month	dayofmonth	dayofweek	deptime	crdelaytime	airline	crsuntime	unispecifier	rflghtnum	tslnum	actualtslgsptime	crslapsedtime	airtime	srtdelay	depdelay	origin	dest	distance	taxin	taxout	cancelled	cancellationcode	diverted	carrierdelay	weatherdelay	nasdelay	securitydelay	lateaircraftdelay	year	tailnum	carrier	man
1	17	6	1630	1613	1920	1926	UA	1200	N907UA	110	131	96	-6	-13	ORD	PVD	849	2	12	0	0	0	0	0	0	0	2004	N907UA	Corporation	BOE	
1	18	7	1613	1615	1914	1924	UA	1200	N518UA	121	129	100	-10	-2	ORD	PVD	849	5	16	0	0	0	0	0	0	0	2004	N518UA	Corporation	BOE	
1	19	1	1614	1615	1918	1924	UA	1200	N574UA	124	129	104	-6	-1	ORD	PVD	849	4	16	0	0	0	0	0	0	0	2004	N574UA	Corporation	BOE	
1	20	2	1615	1615	1922	1924	UA	1200	N526UA	127	129	106	-2	0	ORD	PVD	849	3	18	0	0	0	0	0	0	0	2004	N526UA	Corporation	BOE	
1	21	3	1624	1615	1923	1924	UA	1200	N510UA	119	129	102	-1	9	ORD	PVD	849	3	14	0	0	0	0	0	0	0	2004	N510UA	Corporation	BOE	

- 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



The screenshot shows the Cloudera Data Visualization interface. On the left, there's a sidebar with options like HOME, SQL, VISUALS, and DATA. Under DATA, it says "Dataset Detail" and "Dataset: airline_logistics". The main area is divided into two sections: "Fields" and "Measures".

Fields Section:

- Dimensions:**
 - flights: A. uniquecarrier, A. tailnum, A. origin, A. dest, A. cancellationcode, A. diverted
 - planes: A. planes tailnum, A. owner_type, A. manufacturer, A. issue_date, A. model, A. status, A. aircraft_type, A. engine_type
 - airlines: A. code, A. description
 - airports: A. iata, A. airport, A. city, A. country
 - airports_1: A. airports_1 iata, A. airports_1 airport, A. airports_1 city, A. airports_1 country

Measures Section:

- flights:**
 - # month
 - # daymonth
 - # dayofweek
 - # deptime
 - # crsdeptime
 - # arntime
 - # crsarrtime
 - # flightnum
 - # actualElapsedTime
 - # crsElapsedTime
 - # arrTime
 - # arrDelay
 - # depDelay
 - # distance
 - # taxiin
 - # taxiout
 - # cancelled
 - # carrierDelay
 - # weatherDelay
 - # nasDelay
 - # securityDelay
 - # lateArrivalDelay
 - # year
- planes:**
 - # planes year
- airports:**
 - # state
 - # lat
 - # lon
- airports_1:**
 - # airports_1 state
 - # airports_1 lat
 - # airports_1 lon

In both sections, several fields are highlighted with red boxes. In the Measures section, the first six items under flights (month, daymonth, dayofweek, deptime, crsdeptime, arntime) are also highlighted with red boxes.

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

Dataset Detail
Related Dashboards

Fields

Dataset: airline_logistics

Fields UNDO REFRESH TITLE CASE SAVE Show Comments

Dimensions

- flights
 - Dim [A] uniquecarrier
 - Dim [A] tailnum
 - Dim [A] origin
 - Dim [A] dest
 - Dim [A] cancellationcode
 - Dim [A] diverted
- planes
 - Dim [A] planes tailnum
 - Dim [A] owner_type
 - Dim [A] manufacturer
 - Dim [A] issue_date

- Click on the pencil icon next to Depdelay icon.

Dataset Detail
Related Dashboards

Fields

Dataset: airline_logistics

Fields UNDO REFRESH X TITLE.CASE SAVE Show Comments

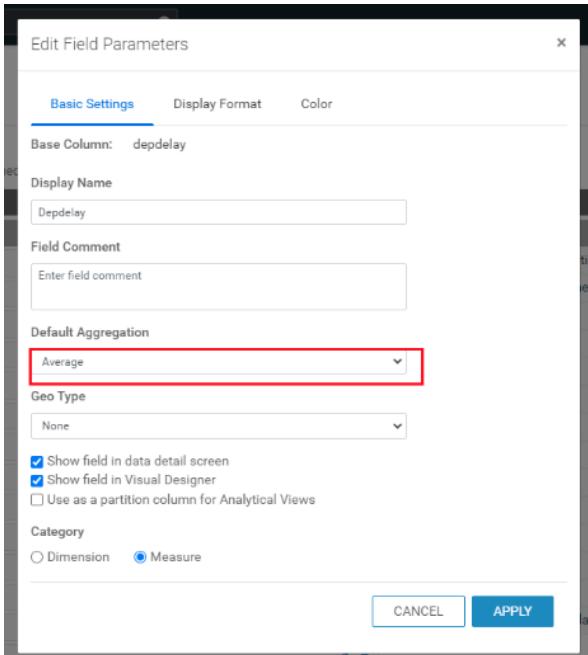
Dimensions

- flights
 - Dim [A] - Month
 - Dim [A] - Daymonth
 - Dim [A] - Dayweek
 - Dim [A] - Depdelay
 - Dim [A] - Arntime
 - Dim [A] - Crsarntime
 - Dim [A] - Uniquecarrier
 - Dim [A] - Fltnum
 - Dim [A] - Tailnum
 - Dim [A] - Origin
 - Dim [A] - Dest
 - Dim [A] - Cancellationcode
 - Dim [A] - Diverted
 - Dim [A] - Year
- planes

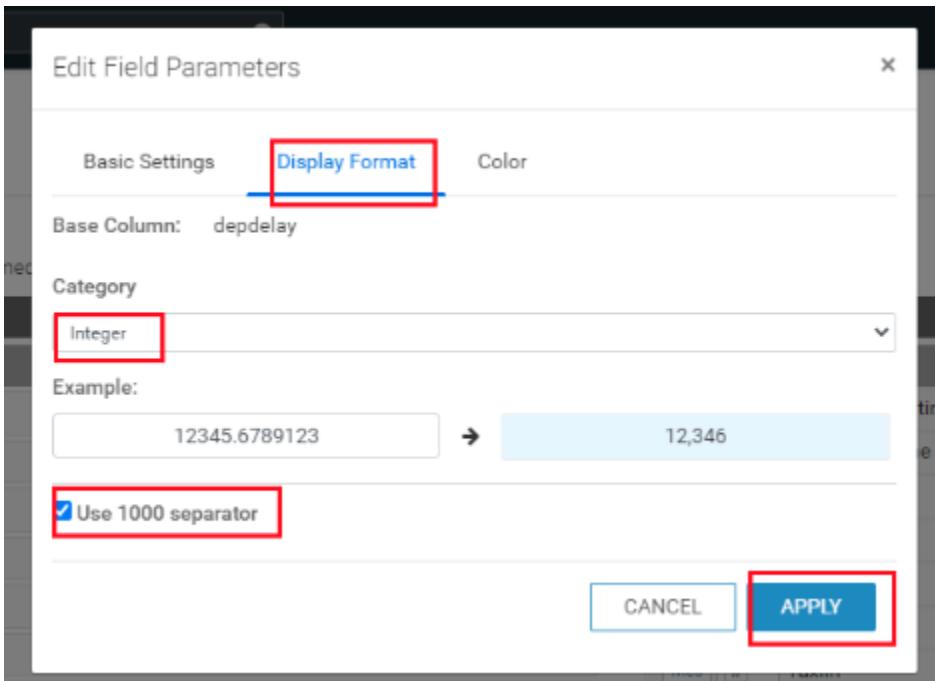
Measures

- Rights
 - Min [D] - Actualelapsedtime
 - Min [D] - Crsdepdelaytime
 - Min [D] - Airline
 - Min [D] - Ambelby
 - Min [D] - Depdelay
 - Min [D] - Distance
 - Min [D] - Taxin
 - Min [D] - Taxout
 - Min [D] - Cancelled
 - Min [D] - Carrndelay
 - Min [D] - Weatherdelay
 - Min [D] - Nasdelay
 - Min [D] - Securitydelay
 - Min [D] - Lateaircraftdelay

- 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.

To add a new calculated field, use the down arrow to the right of a field to clone it, and then edit the expression of the cloned field.

Dimensions

- flights
 - Dim # Month
 - Dim # Dayofmonth
 - Dim # Dayofweek
 - Dim # Deptime
 - Dim # Crsdeptime
 - Dim # Arftime
 - Dim # Crsarrrtime
 - Dim A Uniquecarrier
 - Dim # Flightnum
 - Dim A Tailnum
 - Dim A **Origin**
 - Dim A Dest
 - Dim A Cancellationcode
 - Dim A Diverted
 - Dim # Year

Measures

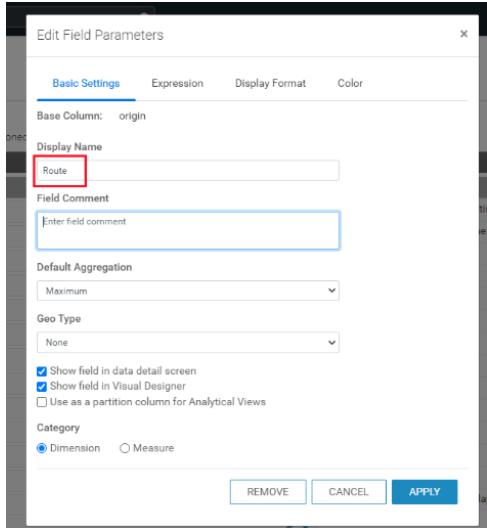
- flights
 - Mes # Actualelap
 - Mes # Crselapsed
 - Mes # Airtime
 - Mes # Arrendelay
 - Mes # Depdelay
 - Mes # Distance
 - Mes # Taxin
 - Mes # Taxout
 - Mes # Cancelled
 - Mes # Carrierdelay
 - Mes # Weatherdelay
 - Mes # Nasdelay
 - Securitydelay
 - Lateaircraft

To add a new calculated field, use the down arrow to the right of a field to clone it, and then edit the expression of the cloned field.

Dimensions

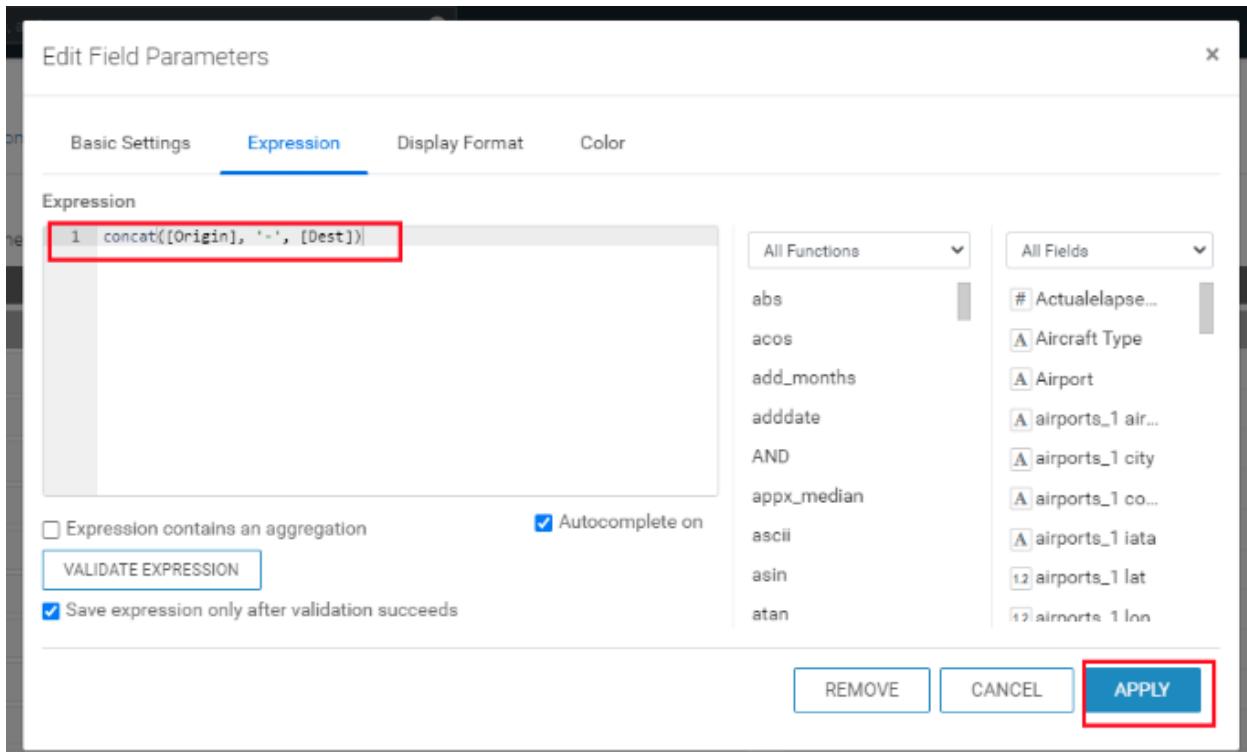
- flights
 - Dim # Month
 - Dim # Dayofmonth
 - Dim # Dayofweek
 - Dim # Deptime
 - Dim # Crsdeptime
 - Dim # Arftime
 - Dim # Crsarrrtime
 - Dim A Uniquecarrier
 - Dim # Flightnum
 - Dim A Tailnum
 - Dim A Origin
 - = Dim A **Copy of Origin**
 - Dim A Dest

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.

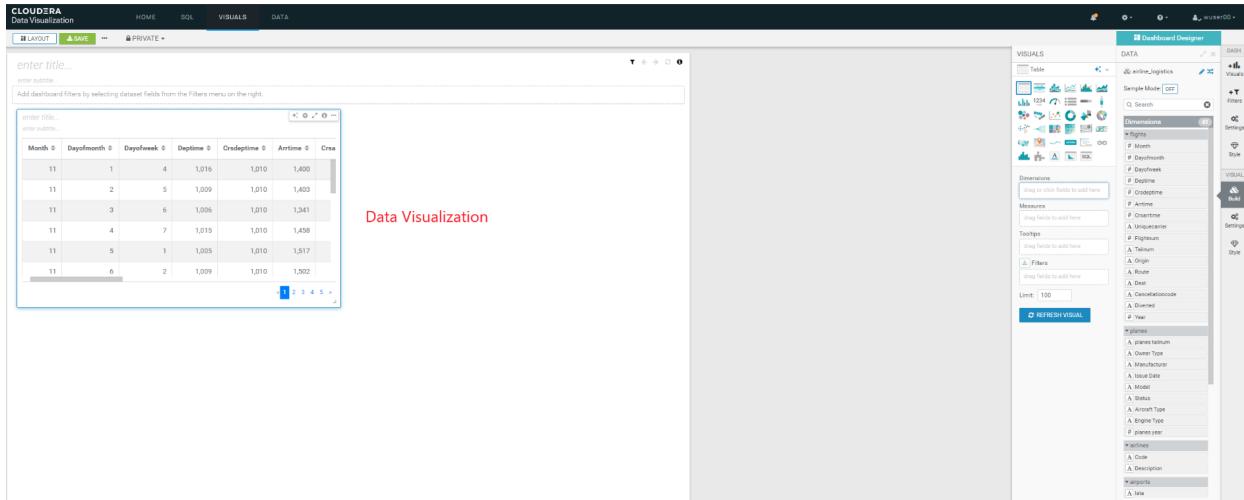
The screenshot shows the Cloudera Data Visualization interface. The top navigation bar includes links for HOME, SQL, VISUALS, DATA, and a search bar. The main area is titled 'Dataset: airline_logistics'. On the left, there's a sidebar with sections for Dataset Detail, Related Dashboards, Fields (which is currently selected), Data Model, Time Modeling, Segments, Filter Associations, and Permissions. The 'Fields' panel displays a list of dimensions under the 'flights' category, including Month, Dayofmonth, Dayofweek, DepTime, CrsDepTime, ArrTime, CrsArrTime, UniqueCarrier, FlightNum, TailNum, Origin, and Route. At the top of the Fields panel, there are buttons for UNDO, REFRESH, TITLE_CASE, and a large green 'SAVE' button, which is highlighted with a red box.

Dashboard Creation

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

This screenshot shows the same interface as the previous one, but with a red arrow originating from the 'NEW DASHBOARD' button located in the top right corner of the 'Fields' panel and pointing towards the bottom right corner of the screen. The 'Fields' panel on the left lists dimensions like Month, Dayofmonth, Dayofweek, DepTime, CrsDepTime, ArrTime, CrsArrTime, UniqueCarrier, FlightNum, TailNum, Origin, and Route. The 'Measures' panel on the right lists measures such as ActualElapsedTime, OneWayDistance, Altitude, ArrDelay, DepDelay, Distance, Taxin, Taxout, and Cancelled.

- 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.

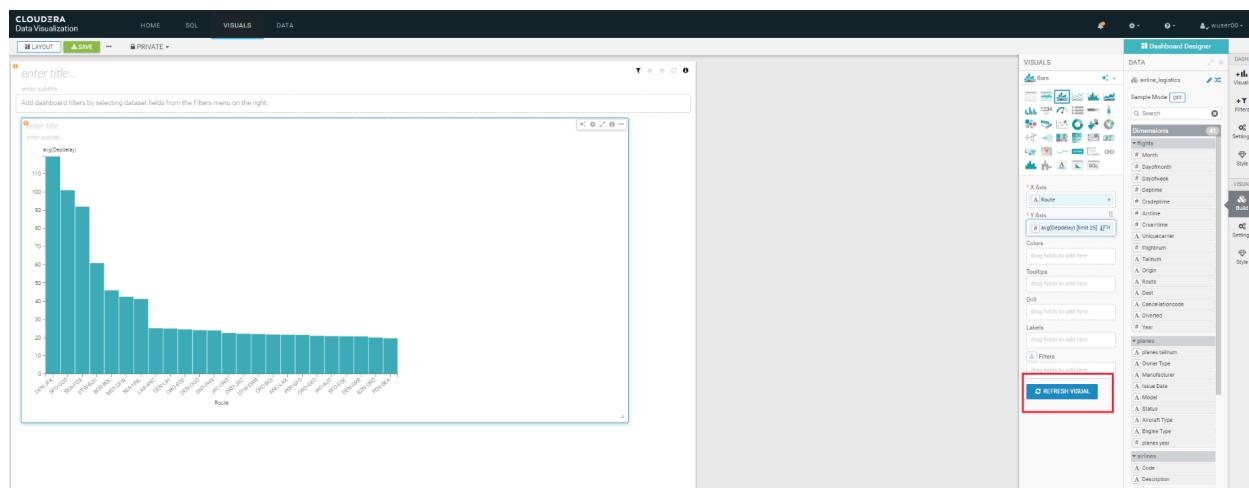
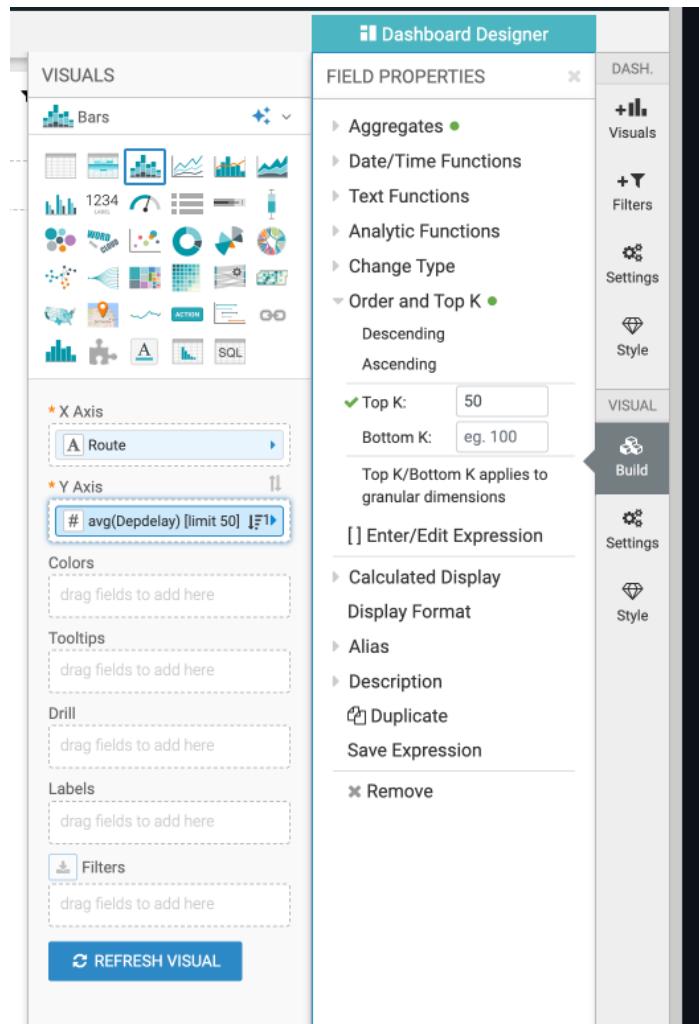
- Pick the Visual Type - Select the Stacked Bar chart visual on the top right as shown below. Make sure Build is selected for it to appear on the right side.

The screenshot shows the Cloudera Data Visualization interface. On the left, there's a table with columns: Month, Dayofmonth, Dayofweek, DepTime, CrtDepTime, Artime, and CrsA. The table has 6 rows of data. In the center, there's a large empty area labeled 'enter title...' and 'enter subtitle...'. On the right, the 'Dashboard Designer' sidebar is open, showing various sections like 'VISUALS', 'DATA', 'DASH.', 'FILTERS', 'SETTINGS', and 'STYLE'. A red arrow points from the 'Visual' section in the left sidebar to the Stacked Bar chart icon in the 'VISUALS' library. Another red arrow points from the 'Build' button in the right sidebar to the 'Build' section in the 'FIELD PROPERTIES' panel.

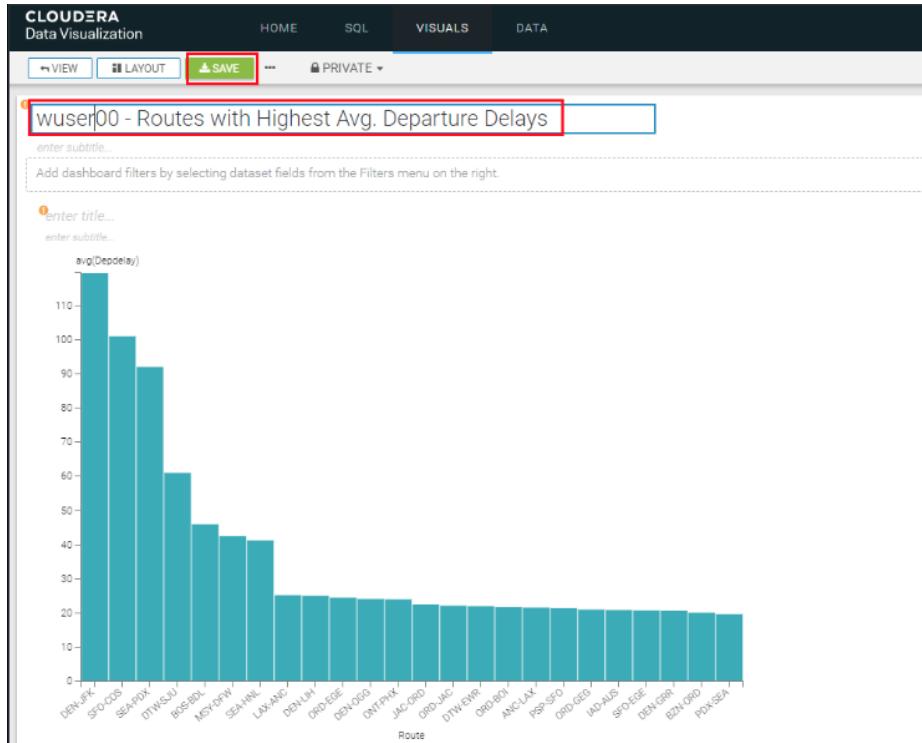
Find Route under Dimensions → flights. Drag to X-Axis. Similarly, find DepDelay under Measures → flights. Drag to Y-Axis. By default the aggregation selected is average and hence you would see avg(Depdelay).

The screenshot shows the 'Dashboard Designer' interface with the 'Build' section of the 'FIELD PROPERTIES' panel highlighted. The 'Build' button is highlighted with a red box. The 'X Axis' field contains 'Route' and the 'Y Axis' field contains '# avg(Depdelay)'. Other fields like 'Top K', 'Bottom K', 'Calculated Display', 'Display Format', 'Alias', 'Description', 'Duplicate', 'Save Expression', and 'Remove' are also visible.

- 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.