

---

## 2 Hours to Data Innovation with Cloudera Data Warehouse

July 19, 2023



## Cloudera Data Warehouse Workshop

### Self Service Hands On Lab

#### Index:

Business Use Case	3
Introduction to the Lab Data	4
Lab Setup	6
Get Started - Log into CDP	6
Lab 1 - Business Analyst: Explore Completed Dashboard	8
Lab 2 - Business Analyst: Self Service (in CDW)	12
Lab 3 - Administrator: “Productionalize” the Open Data Lakehouse (Optional Bonus Material)	34
Optional Lab 1 - CDW Tour	49
Optional Lab 2 - Self Service File Upload	50
Optional Lab 3 - Performance Optimizations (Impala VW)	53
Optional Lab 4 - Data Visualization	57
Optional Lab 5 - Data Security & Governance	74
Optional Lab 6 - Iceberg Table Rollback	80

## Business Use Case

The following Labs will take you through a Self-Service Analytics example. We will use a fictitious company named, Special Marketing Group (SMG). A little company background on SMG is that they are a marketing data aggregator company that is currently working closely with Duty-Free Shops in many airports throughout the U.S. and 20+ countries. SMG is working with the Duty-Free Shop owners on an initiative to drive additional revenue into the shops. The duty-free shop owners want to partner with specific airlines to offer discounts through the airlines to their passengers who spend more time in an international terminal than warranted for their flight.

SMG believes that it can help drive more revenue by driving more traffic to the duty-free shops. SMG realizes that they have some questions that need to be answered, such as:

- Which airlines have the most passengers who have long layovers built into their tickets?
- Which airlines have the most passengers who have delayed international legs in their itinerary causing an extended layover in their flight itinerary?
- Which airlines have the most passengers who are displaced by a missed international connection, caused by a delay in their previous leg?

By answering these burning business questions SMG will be able to work with duty-free shop owners to develop campaigns that they can run to drive additional revenue. Since, SMG also does quite a bit of business with many of the major airlines and believes that the analytics used to drive more revenue for duty-free shops could also help the airlines improve customer satisfaction during extended wait times, and duty-free shops can increase sales.

Our Labs today will walk through the steps to learn how SMG uses the power of Cloudera's Data Warehouse Data Service to leverage the Self-Service capabilities to build out this new solution. If/when the Business Analyst proves that this new data will work to solve the need, they can turn it over to the admin team to "productionalize" it.

### GET STARTED

- Preparatory work for us to get ourselves oriented in CDW and ready to build out the use case
- Login to CDP
- Learn a little about Cloudera Data Warehouse (CDW) Data Service

### SEE THE END RESULT (WHAT IS CREATED) (LAB 1)

- Explore the created Dashboard after it has been "productionalized"
  - Analyst first validates that the combination of data answers questions & builds initial Dashboard to start gaining insights on the data
  - Administrator picks up and formalizes the data ingestion for the new data, incorporates it into the Data Lakehouse, adds security, & completes the Dashboard

### BUSINESS ANALYST - SELF SERVICE TO VALIDATE NEW DATA ANSWERS QUESTIONS (LAB 2)

- Upload new data - passenger ticket manifest

- See if new data can help to answer questions prior to undertaking a complete project
- See how to upload data
- Answer burning business questions to see if the new passenger data
  - Combine uploaded data with existing data warehouse
  - Start digging into the data - running a few queries to see if the new dataset can help with answering the burning questions
  - Visualize the data - modify existing Dashboard to add new content with the data just added

## ADMINISTRATOR - “PRODUCTIONALIZE” INTO OPEN DATA LAKEHOUSE (LAB 3)

- See “how the sausage was made” - how to take advantage of Apache Iceberg from end to end to deliver a modern Open Data Lakehouse solution
  - Migrate existing tables to Iceberg Table Format
  - Create a new Iceberg table
  - Load data
  - Some Key Features of Iceberg
    - Partition evolution
    - Time Travel
- Monitor the Virtual Warehouses (compute) and watch as it scales up and down, suspends, etc.
- Security & Governance before launching to the masses

## CONTINUE ON POST THIS WORKSHOP

*Work with your Breakout SE to go thru these Labs after we wrap up*

- Cloudera Data Warehouse (CDW) Tour
- “Make it Better” - Performance Improvements
  - Materialized Views - improve performance
  - Monitor, report, kill queries that run amuck, etc...
- Develop Rich Visualizations for Analysts & Engineers to use insights - more on Cloudera Data Viz

---

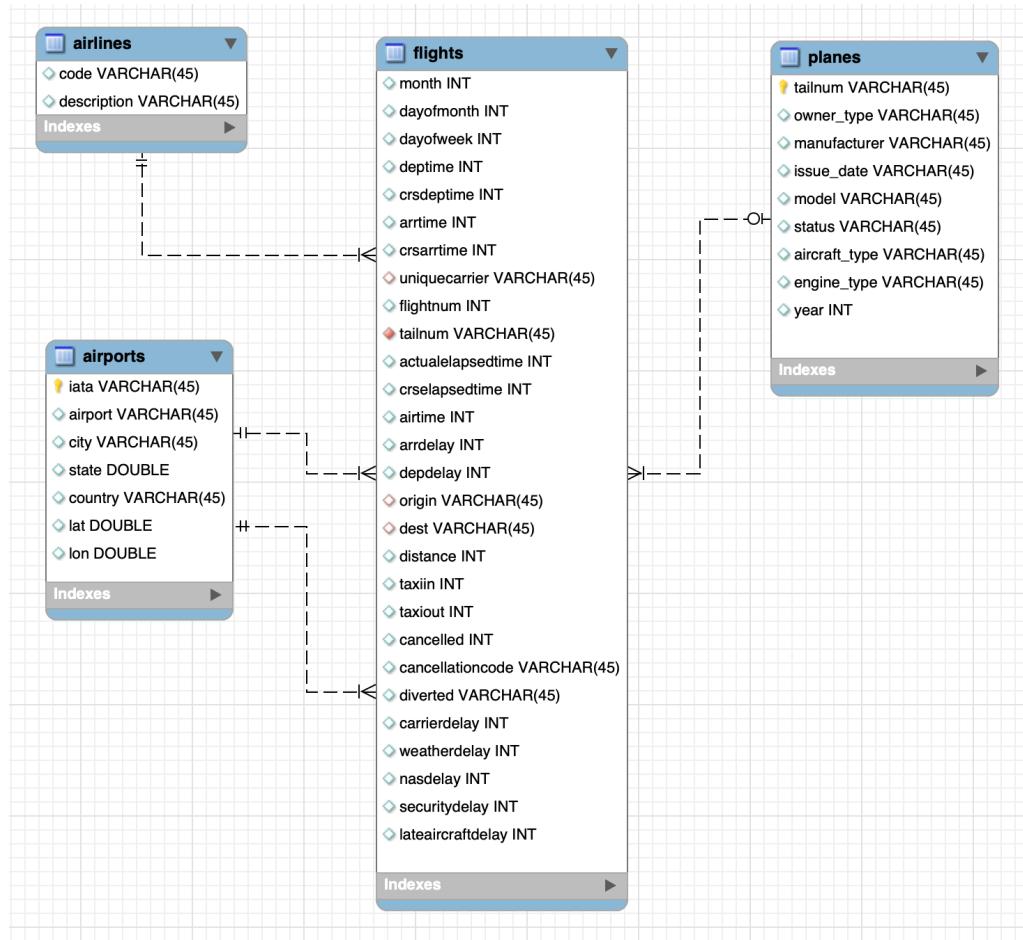
## Introduction to the Lab Data

For the following Workshop Hands On Labs, we will dive into this Scenario to show Cloudera Data Warehouse (CDW) is used to enable SMG to gain a 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 demo Data Lakehouse:

- **Fact table:** flights (86M rows)
- **Dimension tables:** airlines (1.5k rows), airports (3.3k rows) and planes (5k rows)



## Self Service data file - Passenger Ticket manifest:

- This is a CSV file with 1,000 records
- Each record represents a unique Passenger Ticket with 2 Flight Legs
- It contains the following schema

```

ticketnumber BIGINT
leg1flightnum BIGINT
leg1uniquecarrier STRING
leg1origin STRING
leg1dest STRING
leg1month BIGINT
leg1dayofmonth BIGINT
leg1dayofweek BIGINT
leg1deptime BIGINT
leg1arrtime BIGINT
leg2flightnum BIGINT
leg2uniquecarrier STRING
leg2origin STRING
leg2dest STRING

```

leg2month BIGINT  
leg2dayofmonth BIGINT  
leg2dayofweek BIGINT  
leg2deptime BIGINT  
leg2arrrtime BIGINT

## Lab Setup

Workshop Attendees will be provided with a Login. Please enter your details below for reference throughout these lab exercises.

- Lab Group #: \_\_\_\_\_ (this will be your breakout room number)
  - You will use this to replace the “#” within the lab exercises with the number you enter here
- URL: \_\_\_\_\_
  - Use Incognito Browser window
- Login:
  - User: \_\_\_\_\_
  - Password: \_\_\_\_\_

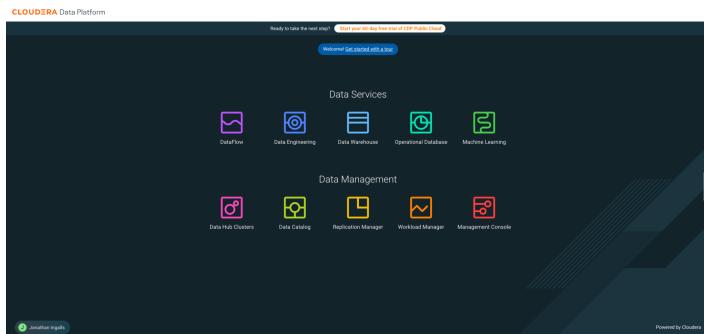
In the labs when you see:

- \${user\_id} or <user\_id> - this will indicate to use your User (Workload User) provided for you to login to CDP
- \${bucket} - this will indicate to use the bucket provided to you

## Get Started - Log into CDP

In this Lab you will login to CDP and complete your user setup.

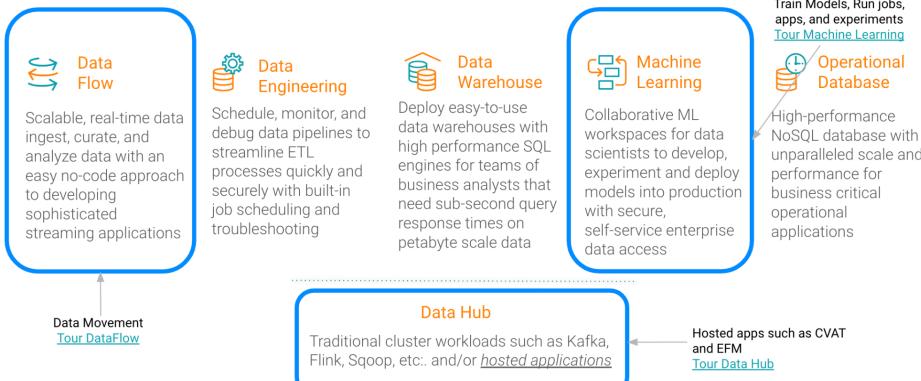
1. Log in to CDP - open a browser and open the URL from the Lab Setup section
  - When prompted use the Login details from the Lab Setup section for the User/PW
2. On the HOME page of CDP you will see all of the Services available to you



- Multi-function analytics - Data Services; today we'll just focus on the Cloudera Data Warehouse Data Service

# CLOUDERA

- There are other Data Services that usually are part of an Analytic use case
  - Data Flow - for data ingestion needs
  - Data Engineering - for ELT/ETL, transformations, data wrangling, etc.
  - Machine Learning - for Data Science teams to collaboratively build and productionalize ML/AI applications and models for the Enterprise
- However, Cloudera provides other services such as Data Catalog, Replication Manager, Workload Manager, and Management Console provide continuity and functionality throughout the platform.



Note: The platform removes the necessity to spend unnecessarily on integration tax, where other solutions combine various pieces together in the ability to provide continuity and functionality throughout an end-to-end use case but this is difficult because it introduces many moving parts.

---

## Lab 1 - Business Analyst: Explore Completed Dashboard

In this Lab you will explore the Dashboard that will be created as the result of the Self-Service Labs you will complete shortly.

### 3. Open Cloudera Data Visualization, which is part of the Cloudera Data Warehouse Data Service

- First let's open Cloudera Data Warehouse (CDW) -
  - Click on the Cloudera Data Warehouse (CDW) tile



- We'll come back to this screen in Lab 2 for more details

- Now let's open Cloudera Data Visualization (CDV) - to explore the finished product
  - Open Cloudera Data Visualization (CDV or Data Viz)
    - On the left navigation panel click "Data Visualization" - this will open a new browser tab
    - On the row with **airlines-dataviz-#**, to the right, click the Data VIZ button

- If you see the "What's New" page, you can read it, or click on the GOT IT button

- Cloudera Data Visualization (CDV) Home page

- 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

- HOME - this is the starting point; it shows some statistics at the top, followed by some quick access details to recent content - Queries, Connections, Datasets, and Dashboards
- SQL - allows you to manually build queries against data to perform quick discovery against the data. Below is an example of a query that was built and Run

The screenshot shows the Cloudera Data Visualization interface with the 'SQL' tab selected. A query is being run against the 'main' database, specifically the 'cereals' table. The results show various cereal types with their nutritional information. The columns include cereal\_name, manufacturer\_code, cold\_or\_hot, calories, protein\_grains, fat\_grains, sodium\_mg, dietary\_fiber\_grains, complex\_carbohydrates\_grains, sugars\_grains, display\_shelf, and potassium\_mg.

cereal_name	manufacturer_code	cold_or_hot	calories	protein_grains	fat_grains	sodium_mg	dietary_fiber_grains	complex_carbohydrates_grains	sugars_grains	display_shelf	potassium_mg
100% Bran	N	C	70	4	1	130	10	5	6	3	230
100% Natural Bran	O	C	120	3	5	15	2	8	8	3	135
All Bran	K	C	70	4	1	260	9	7	5	3	320
All-Bran with Extra Fiber	K	C	50	4	0	140	14	8	0	3	230
Almond Delight	R	C	110	2	2	250	1	14	8	5	-1
Apple_Cinnamon_Oatmeal	G	C	110	2	2	180			10	1	25
Apple_Jacks	K	C	110	2	0	125	1	11	14	2	30
Basil_A	O	C	120	9	2	210	2	18	8	3	100
Bran_Chips	R	C	90	2	1	200	4	19	6	1	125

- VISUALS - an area for viewing/building/modifying visuals, dashboards, and applications

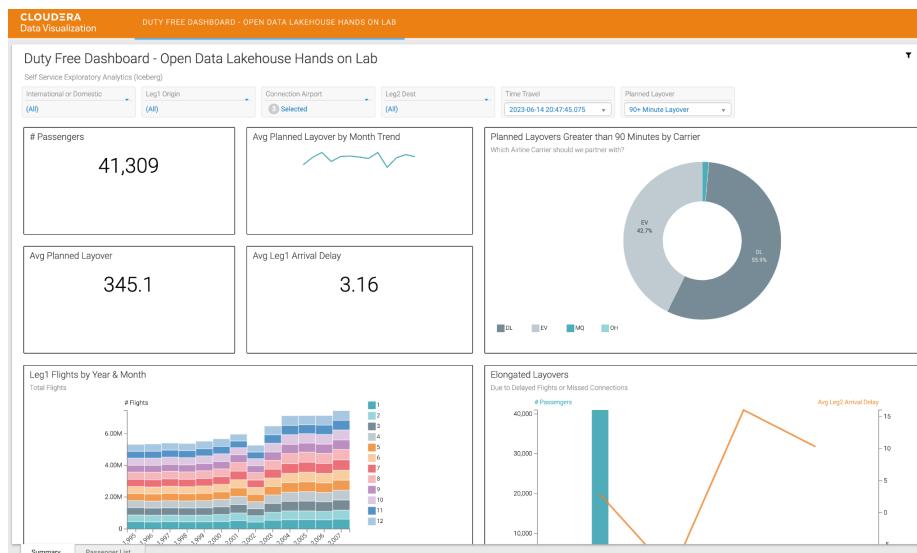
The screenshot shows the Cloudera Data Visualization interface with the 'VISUALS' tab selected. It displays a workspace titled 'All' containing several dashboards. One dashboard is visible, titled 'All', which includes four cards: 'Deficiency Details', 'State of NYC', 'Store Details', and 'Cereal Comparisons'.

- DATA - interface for access to datasets (aka: metadata model) image below, connections, and the Connection Explorer

The screenshot shows the Cloudera Data Visualization interface with the 'DATA' tab selected. It displays a dataset named 'Open Data Lakehouse HOL'. The 'Data Model' section shows a diagram of entities and their relationships. An entity 'unique\_tickets' is connected to entities 'airports', 'airports\_1', 'airlines', 'airlines\_1', 'flights', and 'flights\_1'. Buttons for 'SHOW DATA' and 'Apply Display Format' are visible at the bottom.

- Click on the VISUALS tab at the top of the screen (in black banner)

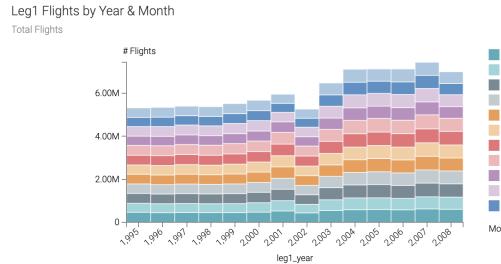
- On the left side of the page make sure you have the All or Public Workspace selected
- Click on the tile “SMG Duty Free Analysis” to open the Application (in the bottom right of the tile you will see a purple icon identifying an Application)
  - Click the drop-down arrow next to the “Planned Layover” prompt and change it from “All Passengers” to “30+ Minutes” - since the initial questions SMG was interested in pursuing were related to passengers with “Long” planned layovers - select “90+ Minutes”.
  - Next, click the drop-down arrow next to “Connection Airport” - Select a few of the Airports that are interested in this new business program
    - Scroll (or search) and click the checkbox next to - ATL, LGA, and ORD
    - Each time you select another query is being executed against the Open Data Lakehouse database tables - shows the performance of Apache Iceberg
  - From these filters (prompt selections) you gain considerable insights - look at the “Planned Layovers Greater than 90 Minutes by Carrier” visual and the answer to the question seems clear as to “Which Carriers should we partner?”.



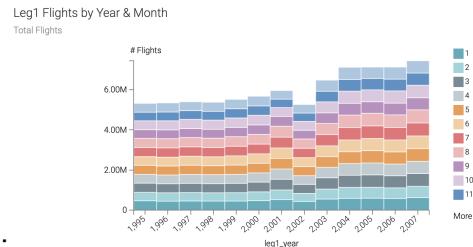
- From this you can see what Airlines (Carriers) that we could partner with due to Long Planned Layovers and see that Delta Airlines (DL) & Atlantic Southeast

(EV) have combined planned layovers for over 95% of Passengers with Planned Layovers of over 90 minutes

- What if I wanted to see the flight details from a previous data load? Click the drop-down arrow below “Time Travel” and select the middle value. You will see the chart titled “Leg1 Flights by Year & Month”.



Before:



After (no 2008):

- You'll see that after you make the selection, you should see the stacked bar for year 2008 is removed from the visual. Meaning this data was not yet loaded at the selected point in time.
- You've just performed your first **Time Travel** without even knowing. Time Travel is a key feature of Apache Iceberg.
- Please go on to the next lab but feel free if you have time to leave this tab open and come back to explore more if you have time after completing the remaining labs.
  - You could do something like the following:
    - At the bottom of the page click on the Passenger List page - now that we've filtered the data to a specific target set of Passengers that can be sent an offer, I'd like to take a look at the list of passengers who would be sent an offer.
    - Filter the list of passengers based on Layover Type - select the drop-down under “Elongated Layover Type” and select “Missed Connection”
    - Click on the checkbox next to “Exclude Selected”. This will exclude those passengers who have missed their connection.
      - For the Duty Free owner, this may not be a good target, so you might want to exclude them from the offer. However, for partnering with the Airline, this might be something we could use to improve customer satisfaction for these 24 passengers and auto rebook these passengers for instance.

## Lab 2 - Business Analyst: Self Service (in CDW)

In this Lab you will take advantage of the Self Service capabilities within CDW to upload a data file and combine it with an existing Data Lakehouse. You will then perform analytics (SQL & Visualizations) on the combined data to ensure that the burning business questions can first be answered before “productionalizing” the new data source.

### 4. Navigate back to CDW Overview

- Click on the browser tab where CDW is open -
- Click on Overview in the left navigation

### 5. The following is a quick explanation of the CDW User Experience so that you have a basic knowledge of the data service. The CDW Data Service allows you to create independent, self-service data warehouses and data marts that autoscale up and down to meet your varying workload demands. It provides isolated compute instances for each data warehouse/mart, has built-in automatic optimization, and ultimately enables you to meet SLAs - at the same time save costs.

- There are 2 areas on this screen - Database Catalogs (DBC), and Virtual Warehouses (VW), for today's lab, we will just concentrate on Virtual Warehouses or VW for short
  - Database Catalogs (DBC) - is a logical collection of table and view metadata, security permissions, and other information. As you create databases, tables, views, etc. in the Data Warehouse, it collects the metadata. When you activate an Environment for CDW,

the CDW service will automatically create a default DBC associated with this Environment.

- DBCs are in the middle column

The screenshot shows a single entry in the Database Catalogs list:

- Name:** wkshp-2hours-...
- Type:** warehouse
- Environment:** warehouse-1686763104-vgvv
- Cloud:** wkshp-2hours-cdw-env
- Status:** Running

Below the list are summary statistics:

TOTAL CORES 9	TOTAL MEMORY 25 GB	VIRTUAL WAREHOUSES 4
------------------	-----------------------	-------------------------

- Virtual Warehouses (VW) - is a set of compute resources running in Kubernetes to execute the queries. This is something that can allow for Self Service capabilities or can be controlled by Administrators. A VW binds compute and storage to execute secured queries that access tables and views of your data via the DBC. A VW can scale automatically to ensure performance even with high concurrency, and can auto-scale down in situations of low demand. Tools that access data via JDBC/ODBC can connect to VWs to run queries

The screenshot shows a single entry in the Virtual Warehouses list:

- Name:** airlines-hive-vw-1
- Type:** compute
- Environment:** compute-1686764902-lt4s
- Cloud:** wkshp-2hours-cdw-dl-default
- Status:** Stopped

Below the list are summary statistics:

EXECUTORS 0	TOTAL CORES 12	TOTAL MEMORY 56 GB	TYPE HIVE UNIFIED ANALYTICS COMPACTOR
----------------	-------------------	-----------------------	--

- See options available for a VW - click on the button on the top right corner of tile **airlines-hive-vw-#**

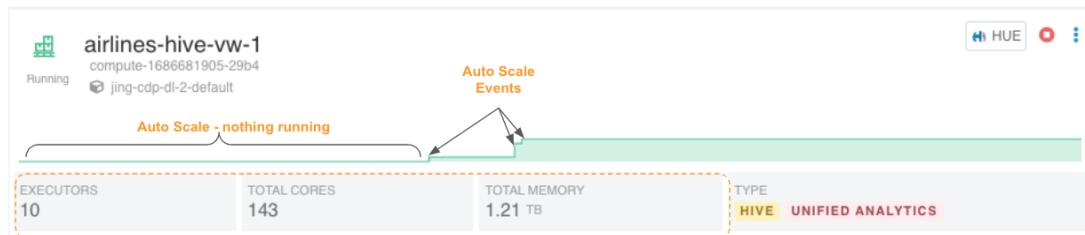
The screenshot shows the same Virtual Warehouses list as before, but with the **airlines-hive-vw-1** tile selected. A context menu is open on the right side of the screen:

- Clone
- Rebuild
- Edit
- Delete
- Upgrade
- Copy JDBC URL
- Set Compactor
- Collect Diagnostic Bundle

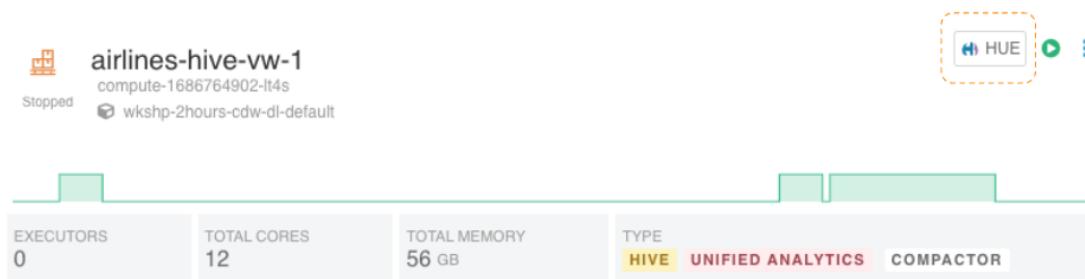
The list of options will vary slightly depending on whether this is a Hive or Impala VW. This is also where you would go to get the JDBC URL and Driver to use to connect your Business Intelligence software to this VW, and when a new version is released you could Upgrade this VW to the latest release without having to upgrade all VWs at the same time.

- **Click on the *airlines-hive-vw-#* tile** - this will highlight the Environment and DBCs that are associated with this VW. In fact when you click on any tile within any column, it will do the same thing by showing you what it is related to, for easy understanding of the interdependence of these items.

- These tiles will display information about current workload on a VW. Below you will see items indicating when a VW is idle and has auto scaled so nothing is running for this VW; and where the VW needed to auto-scale up to handle incoming SQL requests handling concurrency



- Click on the *airlines-hive-vw-#* tile** - In the upper right corner click on the HUE button to enter into the SQL Editor



## 6. Explore the Data Lakehouse

- You will be using HUE for the ***airlines-hive-vw-#* Virtual Warehouse** until you get to Step 13

Search data and saved documents...

Hive Add a name... Add a description...

- Copy & paste the following SQL into the query Editor window

```
-- Run prior to importing Passenger Tickets; to ensure correct access and that everything is good to go
-- Query to find all international flights: flights where destination airport country is not the same as origin airport country
SELECT DISTINCT
  flightnum,
  uniquecarrier,
  origin,
  dest,
  month,
  dayofmonth,
  `dayofweek`
FROM
  airlines.flights f
JOIN airlines.airports oa ON f.origin = oa.iata
```

```

JOIN airlines.airports da ON oa.country <> da.country
WHERE f.dest = da.iata
ORDER BY
    month ASC, dayofmonth ASC
;

```

```

1 -- Run prior to importing Passenger Tickets; to
2 -- Query to find all international flights: fli;
3 SELECT DISTINCT
4   flightnum,
5   uniquecarrier,
6   origin,
7   dest,
8   month,
9   dayofmonth,
10  `dayofweek`
11 FROM
12   airlines.flights f
13 JOIN airlines.airports oa
14   | ON f.origin = oa.iata
15 JOIN airlines.airports da
16   | ON oa.country <> da.country
17 WHERE
18   f.dest = da.iata
19 ORDER BY
20   month ASC,
21   dayofmonth ASC
22 ;
23

```

- Click on the ➤ to the bottom left of the SQL window to run this SQL command to test and ensure you have access to the Data Lakehouse
  - You should see the something similar to the following in the Results tab

	flightnum	uniquecarrier	origin	dest	month	dayofmonth	dayofweek
1	2785	XE	BTR	IAH	1	1	2
2	952	DL	BTR	ATL	1	1	4
3	2174	XE	IAH	BTR	1	1	4
4	3687	MQ	DFW	BTR	1	1	2
5	1115	DL	ATL	BTR	1	1	4
6	2274	XE	IAH	BTR	1	1	4
7	1672	DL	ATL	RTD	1	1	7

- This validates that you have the correct permissions via the SDX security to access the Data Lakehouse

7. Upload Passenger Ticket data file - already completed for you and shown as part of the main presentation
  - If you'd like to give it a try this can be found in the Post Lab section - work with your Breakout Room moderator to set up time.
  - Below are some images that highlight the 3 steps to upload this file: 1) Open the Importer, 2) Pick a file & options for the file upload, 3) Name the table, specify table options, and provide the table metadata (column names, data types, etc.)

1) Sidebar showing tables: airline\_ontime.orc, airlines, airports.

2) Importer dialog for passenger\_tickets\_1k.csv:

FlightNumber	leg1Flightnum	leg1uniquecarrier	legFlight	leg1dest	leg1length	leg1daysinmonth	leg1daysinweek	leg1deptime	leg1arrtime
480108720330	662	NW	FHD	MFM	10	10	2	1407	1407
230115204414	550	NW	LAX	MFM	10	10	2	155	1117
419108179365	262	NW	SAT	MFM	10	10	2	596	731
742088162666	1024	NW	MED	MFM	10	10	2	617	737

3) Importer dialog for airlines.unique\_tickets\_1k:

Name	Annotation	Type	Length	Default
a.leg1uniquecarrier		String	255	230115204414
a.leg1distance		Double	600	550
a.leg1arrtime		String	100	0000000000
a.leg1depart		String	100	0000000000
a.leg1origin		String	100	0000000000
a.leg1dest		String	100	0000000000
a.leg1airlineid		Integer	10	1
a.leg1airlineidname		String	10	1

- Since the data is already uploaded we will use the table **airlines.unique\_tickets\_1k**

- Explore the Data Lakehouse and Passenger Tickets (unique\_tickets) data to see if it can answer the burning questions. The Data Analyst can use a SQL Editor to perform this task by executing queries against this data.
  - Delete the current query from the SQL Window
  - Copy & paste the following SQL into the SQL Editor window

```
-- Run after Uploading the Passenger Tickets data to see if we can answer the "burning
questions" to support Duty Free Stores
-- Number of passengers on the airline that have long, planned layovers for a flight (good
target to send promotion to)
SELECT
    a.leg1uniquecarrier as carrier,
    count(a.leg1uniquecarrier) as passengers
FROM
    airlines.unique_tickets_1k a
where
    a.leg2deptime - a.leg1arrtime > 90
group by
    a.leg1uniquecarrier
;
```

- Execute the Query by clicking on the ➤ button
  - Review the output to see the number of passenger with a long planned layover (> 90 minutes) by Airline Carrier - one of the questions we wanted to answer

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

```

1 -- Run after Uploading the Passenger Tickets data to see if we can an
2 -- Number of passengers on the airline that have long, planned layove
3 SELECT
4   a.leg1uniquecarrier as carrier,
5   count(a.leg1uniquecarrier) as passengers
6 FROM
7   airlines.unique_tickets_1k a
8 where
9   a.leg2deptime - a.leg1arrtime > 90
10 group by
11   a.leg1uniquecarrier
12;
13

```

Below the code editor is a log window showing the execution of the command:

```

INFO : ANALYZE TABLE unique_tickets_1k;
INFO : RAW_INPUT_SPLITS_Map_1: 1
INFO : savedToCache: true
INFO : Completed executing command(queryId=hive_20230615133755_bad51879-dad2
INFO : OK

```

At the bottom, there are tabs for 'Query History', 'Saved Queries', and 'Results (3)'. The 'Results' tab is selected, displaying a table with two columns: 'carrier' and 'passengers'.

carrier	passengers
EV	315
NW	617
MQ	1

- Delete the current query from the SQL Window
- Copy & paste the following SQL into the SQL Editor window

```

-- Number of passengers on airlines that have elongated layovers for a flight caused by
delayed connection (potential customer satisfaction issue)
SELECT
  a.leg1uniquecarrier as carrier,
  count(a.leg1uniquecarrier) as passengers
FROM
  airlines.unique_tickets_1k a
JOIN airlines.flights o
  ON a.leg1flightnum = o.flightnum
    AND a.leg1uniquecarrier = o.uniquecarrier
    AND a.leg1origin = o.origin
    AND a.leg1dest = o.dest
    AND a.leg1month = o.month
    AND a.leg1dayofmonth = o.dayofmonth
    AND a.leg1dayofweek = o.`dayofweek`
JOIN airlines.flights d
  ON a.leg2flightnum = d.flightnum
    AND a.leg2uniquecarrier = d.uniquecarrier
    AND a.leg2origin = d.origin
    AND a.leg2dest = d.dest
    AND a.leg2month = d.month
    AND a.leg2dayofmonth = d.dayofmonth
    AND a.leg2dayofweek = d.`dayofweek`
WHERE o.depdelay > 60
group by
  a.leg1uniquecarrier
;
```

- Execute the Query by clicking on the ▶ button
  - Review the output to see that we can answer another burning question

The screenshot shows the Cloudera Data Visualization (CDV) interface. At the top, there's a navigation bar with 'Hive' selected, and input fields for 'Add a name...' and 'Add a description...'. Below this is a code editor containing a complex Hive query. The query joins multiple tables (airlines.unique\_tickets\_1k and airlines.flights) based on flight numbers and carrier codes, filtering by date ranges and day of week. A WHERE clause is present at the bottom. To the right of the code editor, a log window displays INFO messages related to the query execution, including split information and completion status. Below these windows is a results table titled 'Results (3)' with columns 'carrier' and 'passengers'. The data shows three rows: MQ with 1 passenger, EV with 31 passengers, and NW with 6 passengers.

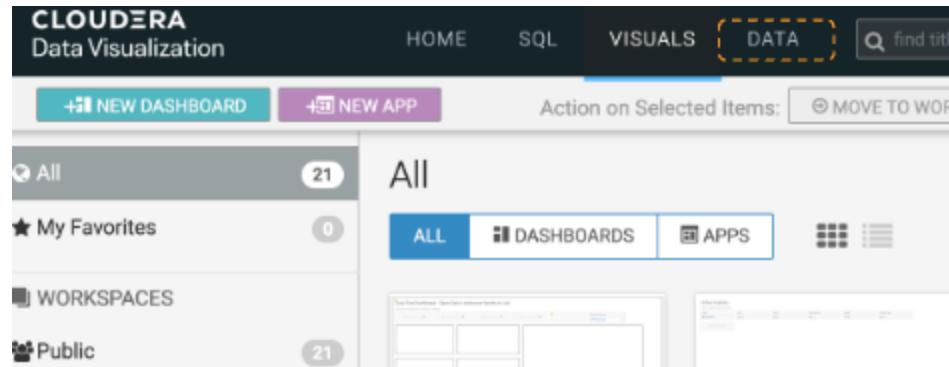
carrier	passengers
MQ	1
EV	31
NW	6

- Note: this query combines both Hive table format (unique\_tickets\_1k) with Iceberg table format (flights). Allows you to migrate to Iceberg over time
- Now that we've validated that we can answer the business questions, it's time to visualize the data to see the insights we can gain from combining this data

## 9. Visualize the Data Lakehouse and Passenger Ticket data to gain insights and communicate what we are trying to accomplish with the Admin team.

- Click on the browser tab where CDV is open -
- To create visualization in Cloudera Data Visualization (CDV), the Business Analyst would have to do the following:
  - Create a Dataset (aka: Metadata Layer or Data Model) that will join the uploaded Passenger Tickets (unique\_tickets\_1k) table with the existing Data Lakehouse tables
  - Create Visualizations create visuals that present information so that users can gain insights to make better decisions. A single Dashboard can have any number of visuals on it.
  - Make the Visualization available to the appropriate stakeholders - by default Dashboards will first be created and edited in the user's "Private" Workspace. Once the Dashboard is ready, it is move to a Workspace that the appropriate users can view and interact with the Dashboard(s)
- Create a Dataset (Metadata Layer)
  - The first step is to create a Metadata layer that joins the Data Lakehouse tables and Passenger Tickets table (unique\_tickets\_1k). To do this we'll create what we call a Dataset.
  - On the top banner click on the DATA tab

# CLOUDERA



- Make sure the “Airlines Open Data Lakehouse” connection on the left navigation menu is selected

The screenshot shows the datasets list for the "Airlines Impala 1" connection. The left sidebar shows connections: All Connections, Airlines Hive 1, Airlines Impala 1 (selected), and samples. The main area displays a table of datasets:

Title/Table	ID	Created	Last Updated	Modified By	# Dashboards
Flights Hands on Lab Created from SQL	15	Jun 14, 2023	37 minutes ago	jingalls	2
Open Data Lakehouse HOL airlines.unique_tickets, airlines.airports, airlines.airport...	16	Jun 14, 2023	37 minutes ago	jingalls	2
Lab 2 Self Service Open Data Lakehouse airlines.unique_tickets, airlines.airports, airlines.airport...	18	Jun 16, 2023	6 hours ago	jingalls	0
Duty Free Iceberg physical jing_airlines.unique_tickets, jing_airlines.flights_route....	13	Jun 14, 2023	a day ago	jingalls	0
Flights airlines.flights, airlines.planes, airlines.airlines, ...	17	Jun 14, 2023	a day ago	jingalls	2

- For today’s labs we’ll fast forward to a Dataset that has already been created to save time.
  - **If you’d like to go through this process, please work with your Breakout Room Moderator to schedule time to walk through the Lab in the Optional Lab section**
- Click on “Lab 2 Self Service Open Data Lakehouse” to open the Dataset that was already created

The screenshot shows the dataset detail page for "Lab 2 Self Service Open Data Lakehouse". The title is "Flights Hands on Lab" and it is noted as "Created from SQL". Below the title, there is a preview of the dataset structure: "Open Data Lakehouse HOL" followed by the schema "airlines.unique\_tickets, airlines.airports, airlines.airport...". The row for "Lab 2 Self Service Open Data Lakehouse" is highlighted with a yellow dashed box.

- Dataset Detail page - provides information about this Metadata Model, such as the connection, tables referenced, option settings, and date information (created/updated). Where you see the pencil icon, indicates that you can make changes.

**Dataset Detail**

Dataset: Lab 2 Self Service Open Data Lakehouse

**Detail**

Dataset: [Lab 2 Self Service Open Data Lakehouse](#)

Table: [airlines.unique\\_tickets](#), [airlines.airports](#), [airlines.airports](#), ...

Connection Type: [Impala](#)

Data Connection: [Airlines Impala 1](#)

Description:

Join Elimination: [Enabled](#)

Result Cache: [From Connection](#)

Incremental Results: [Disabled](#)

ID: 18

Created on: Jun 16, 2023 09:46 AM

Created by: jingalls

Last updated: Jun 16, 2023 09:47 AM

Last updated by: jingalls

- Clone Dataset - click the CLONE DATASET button at the top of the page

Dataset: Clone of Lab 2 Self Service Open Data Lakehouse

**Detail**

- Click the to the right of Dataset to Rename this Dataset

**Detail**

Dataset: [Clone of Lab 2 Self Service Open Data Lakehouse](#)

Table: [airlines.unique\\_tickets](#), [airlines.airports](#), [airlines.airports](#), ...

- Replace the “Clone of” with “<user-id>“ (use your user id in place of <user-id>)

Dataset: [user999 Lab 2 Self Service Open Data Lakeh](#)

- Click the SAVE button
- You now have created your own copy of the Dataset that can now be modified for today’s lab

- Data Model - click on Data Model in the left navigation menu to what tables are in this Dataset

**Dataset Detail**

Related Dashboards

**Fields**

**Data Model**

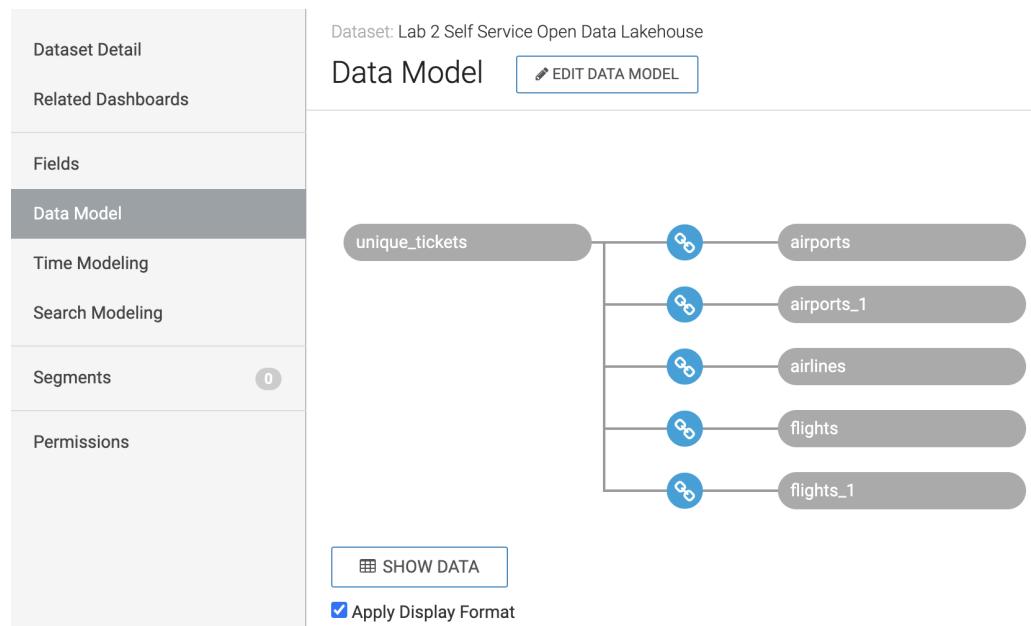
Time Modeling

Search Modeling

Dataset: L

**Detail**

Shows the tables, and relationships between tables in this Dataset



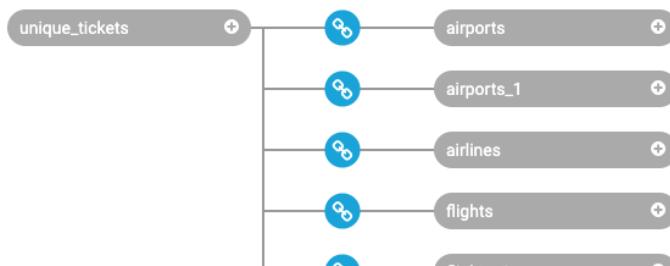
Tables are dark gray and relationships (joins) are identified by the blue

- In this Data Model you can see that the unique\_tickets table has already been joined to several other tables that are part of the Data Lakehouse (airports, airlines, flights, etc), but there is one more relationship that needs to be defined for this to be complete - we need to get the Leg2 Airline Carrier details

- Click the EDIT DATA MODEL button at the top of this page

Dataset: user999 Lab 2 Self Service Open Data Lakehouse

Data Model [UNDO](#) [SAVE](#)



[SHOW DATA](#)

Apply Display Format

- Click the “+” in the gray oval to the right of unique\_tickets - to add a table join
- Click the drop down below Database Name and select “airlines”

- Click the drop down below Table Name and select the “airlines” table

Table Browser x

Choose the table you want to join. You will be able to select the columns that are joined in the next step.

Database Name ▼  
airlines

Table Name ▼  
airlines

CANCEL SELECT

- Click on the SELECT button to add this table
- You will be presented with the Edit Join definition window (if not click the  button between unique\_tickets & airlines\_1)

Edit Join x

CLEAR FIELDS

airlines.unique\_tickets      airlines\_1  
 select column... = select column... ✖  
▶ sample data ▶ sample data

Join Expressions  
 If you enter multiple expressions they will automatically have an "AND" logic between them  
 Click to update in SQL expression editor

+ ADD JOIN PAIR + ADD JOIN EXPRESSION

CANCEL APPLY

- Click the drop down below airlines.unique\_tickets and select leg2uniquecarrier
- Click the drop down below airlines\_1 and select code

Edit Join x

CLEAR FIELDS

airlines.unique\_tickets      airlines\_1  
 leg2uniquecarrier = code ✖  
▶ sample data ▶ sample data

Join Expressions  
 If you enter multiple expressions they will automatically have an "AND" logic between them  
 Click to update in SQL expression editor

+ ADD JOIN PAIR + ADD JOIN EXPRESSION

CANCEL APPLY

- Click the APPLY button

- This will create a join between the unique\_tickets table and the airlines table in your user database.



- To view or edit the join that was created click on  between unique\_tickets and airlines (at the bottom) Source/Target column



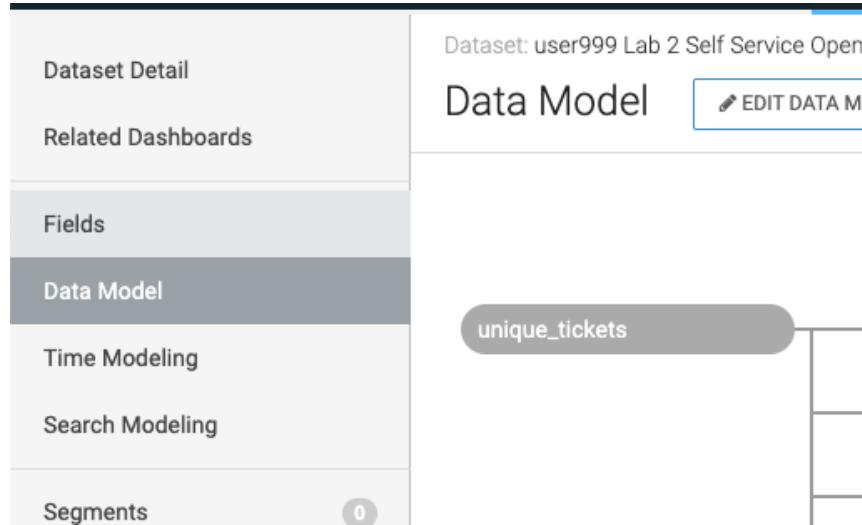
This will show the join type and allow you to make changes. You can see by default it created a Left outer join which you can change by selecting any of the other types such as Inner, Outer, or Right. This Join is exactly what we need, so no need to make any changes

A screenshot of a "Join Details" dialog box. At the top, there are two buttons: "SHOW DATA" (which is highlighted with a blue border) and "Apply Display Format" (with a checked checkbox). Below these are four radio button options for join types: "Inner" (grey), "Left" (blue, selected and highlighted with a blue border), "Right" (grey), and "Outer" (grey). Further down, under "Source Column" and "Target Column", there is a dropdown menu showing "leg2uniquecarrier" and "=" followed by "code". At the bottom of the dialog are two buttons: "DELETE JOIN" and "EDIT JOIN".

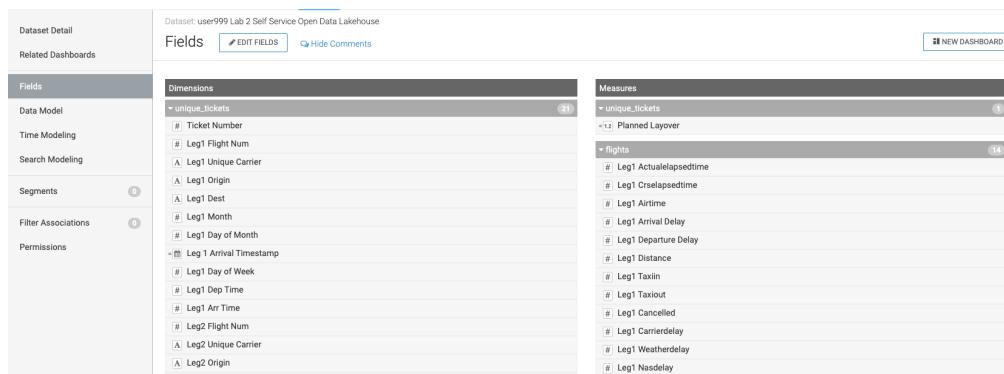
- Click on the SHOW DATA button to see a preview of the data - this will run a query to combine the data from our Data Lakehouse with the data

we uploaded to make sure that the data looks good. Scroll to the right to look at all of the data from each table, and scroll down to see more data

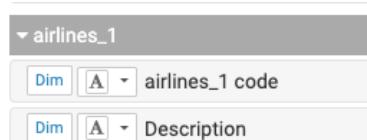
- At the top of the page click the SAVE button
  - Fields - on the left navigation menu select Fields



- Click the EDIT FIELDS button at the top of the page



- Under Dimensions - scroll down to the airlines 1 header



This is what was added from adding the `airlines.airlines` table to this Dataset. From here we can apply business rules and context to this Dataset.

- Rename a Field - Description to Leg2 Airline Carrier

- Click on the button to the right of Description

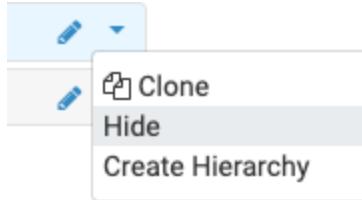
Edit Field Parameters x

Basic Settings	Display Format	Color
Base Column: description		
Display Name		
<input type="text" value="Description"/>		
Field Comment		
<input type="text" value="Enter field comment"/>		
Default Aggregation		
<input type="button" value="Maximum"/>		
Geo Type		
<input type="button" value="None"/>		
<input checked="" type="checkbox"/> Show field in data detail screen <input checked="" type="checkbox"/> Show field in Visual Designer <input type="checkbox"/> Use as a partition column for Analytical Views		
Category		
<input checked="" type="radio"/> Dimension <input type="radio"/> Measure		
<input type="button" value="CANCEL"/> <input style="background-color: #0072BD; color: white; border: 1px solid #0072BD;" type="button" value="APPLY"/>		

- In the Display Name text box change this to Leg2 Airline Carrier

Basic Settings	Display Format	Color
Base Column: description		
Display Name		
<input type="text" value="Leg2 Airline Carrier"/>		

- Click the APPLY button
- Hide a Field
  - The airlines\_1 code is the same values as the leg2uniquecarrier field, so no need to have this twice
  - Click the and select Hide



- Select Hide
- The airlines\_1 section should look like this

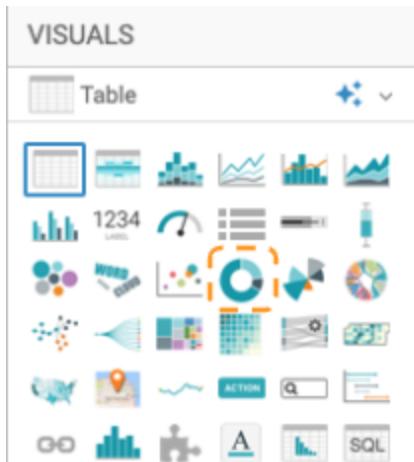
- Click the SAVE button at the top of the page
- There is so much more you can do with a Dataset, including adding calculated fields, assigning default aggregations, apply formatting, and strong data types to help with building visualizations quickly.

- Create a Dashboard - click the NEW DASHBOARD button in the top right corner

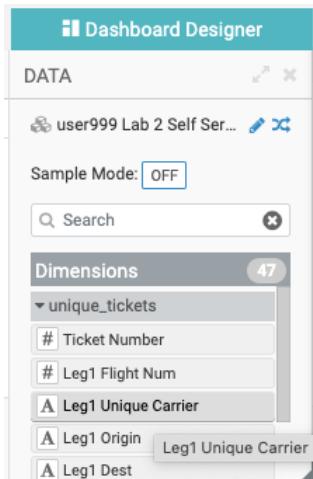
- This will automatically build a default visual for you like this

- Let's change this to answer one of the burning questions - which airline carrier should we partner with

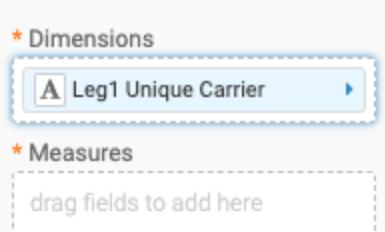
- Toward the right of the screen under VISUALS and select the Pie Chart visual (3rd row in the middle)



- Under VISUALS click on the box under Dimensions - this is called a Shelf. To add items to a Shelf you can drag and drop or in this case select it from the far right of the screen from the Dataset we just created
  - On the far right of the screen under DATA click on Leg1 Unique Carrier



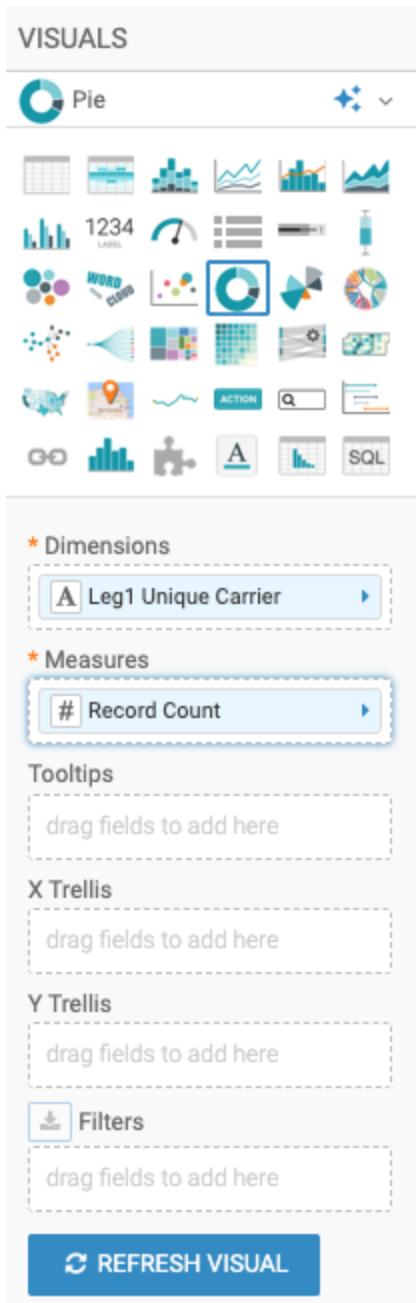
- This will add this to the Dimension Shelf



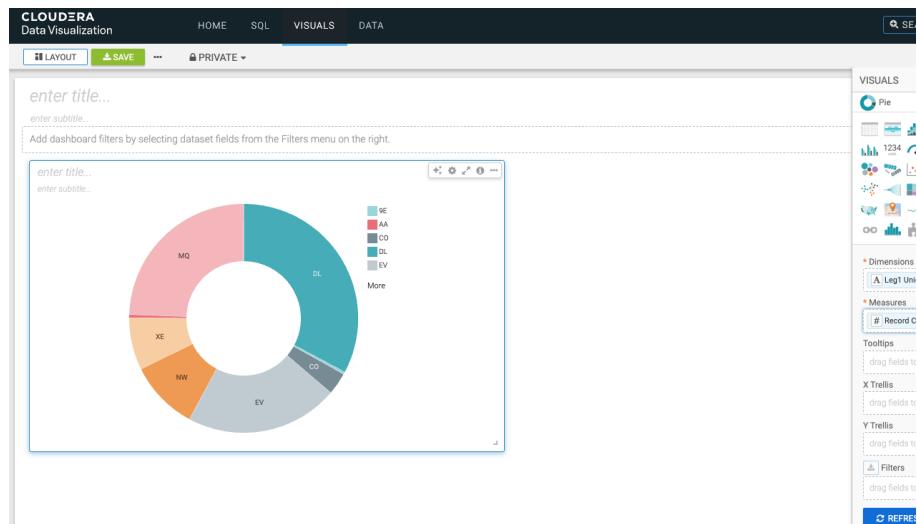
- On the far right of the screen, drag & drop Record Count from under DATA > Measures > unique\_tickets to the box below Measures

The screenshot shows the Cloudera Dashboard Designer interface. At the top, there's a header bar with the title "Dashboard Designer". Below it is a "DATA" tab. Under the DATA tab, there's a section titled "Dimensions" with a count of 47 items. The dimensions listed under "unique\_tickets" include: Ticket Number, Leg1 Flight Num, Leg1 Unique Carrier, Leg1 Origin, Leg1 Dest, Leg1 Month, Leg1 Day of Month, Leg 1 Arrival Timestamp, Leg1 Day of Week, Leg1 Dep Time, Leg1 Arr Time, Leg2 Flight Num, Leg2 Unique Carrier, and Leg2 Origin. Below the Dimensions section is a "Measures" section with a count of 30 items. The measures listed under "unique\_tickets" include: Record Count, sum(1) over flights, and Record Count. The "sum(1) over flights" measure is highlighted with a black box and a callout arrow pointing to it.

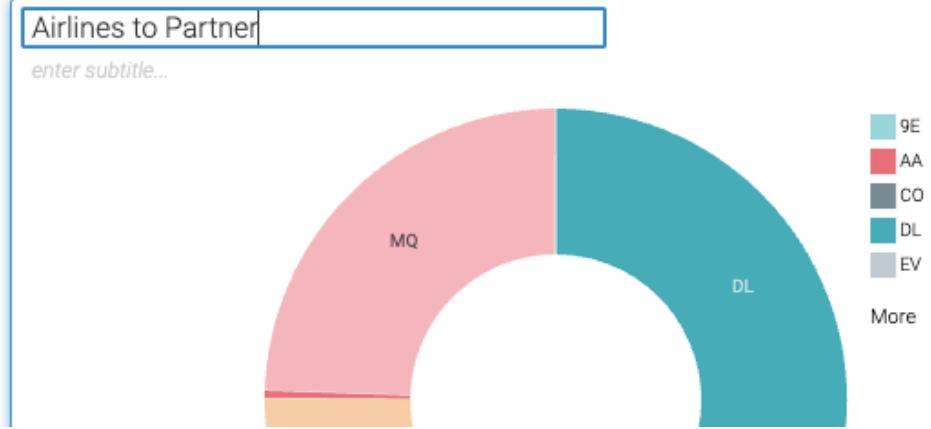
- It should look like the following screen. This will combine data from the unique\_tickets table (file upload) and the existing Data Lakehouse to display the number of passengers by each Airline



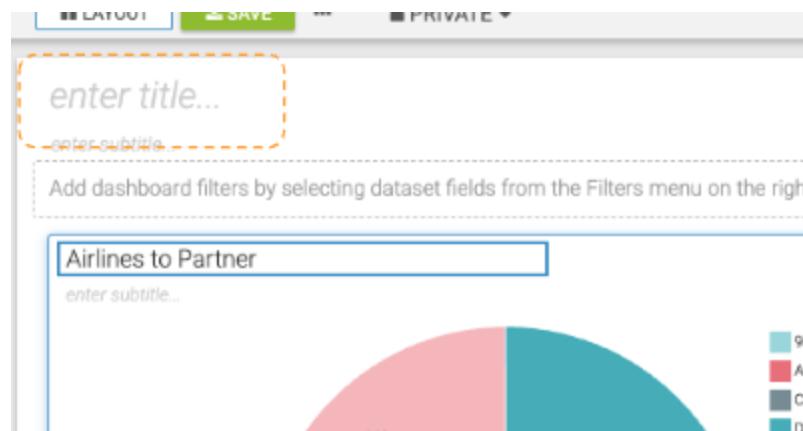
- Click the REFRESH VISUAL button
- On the left of the screen the visual is updated to this



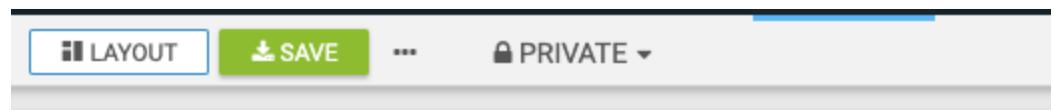
- Add a title to the chart by clicking on “enter title...” above the chart. Change it to “Airlines to Partner” and press enter



- Add a title (name) for the Dashboard - at the top of the visualization click on “enter title...”



- Type in “<user-id> Self Service Dashboard” (replace <user-id> with your user id) and press enter



## user999 Self Service Dashboard

*enter subtitle...*

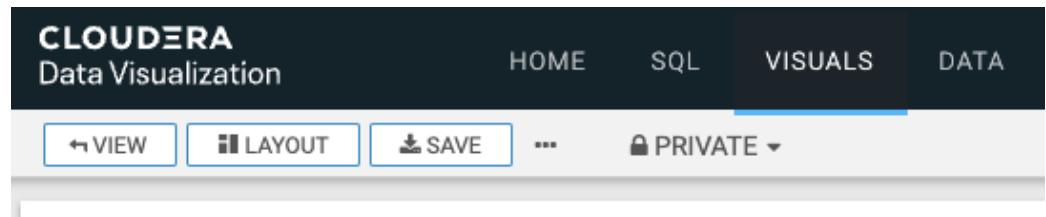
Add dashboard filters by selecting dataset fields from the Filters menu on the right

### Airlines to Partner

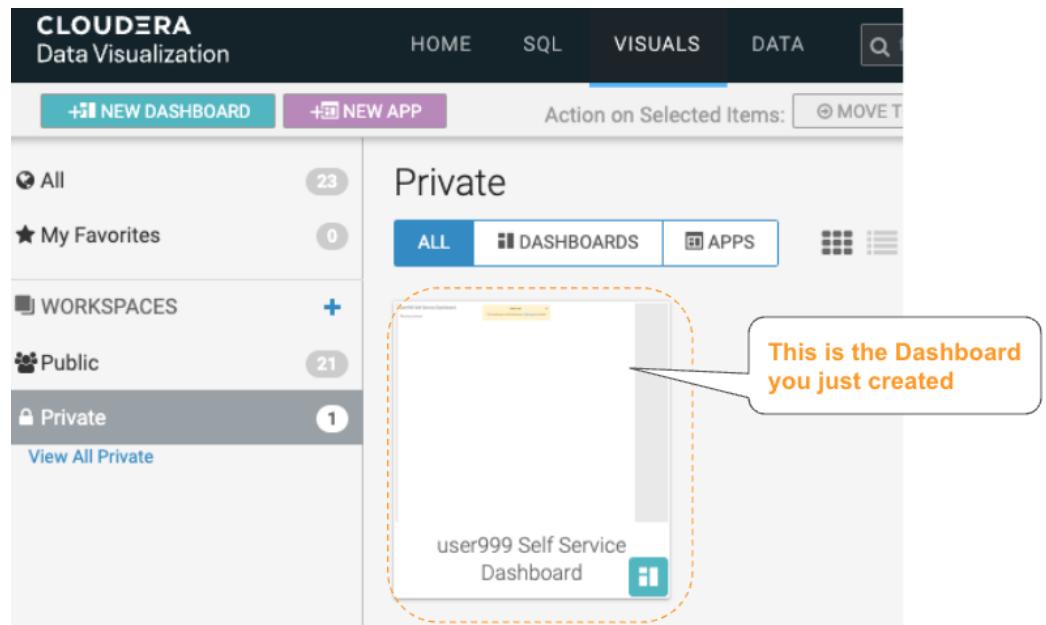
*enter subtitle...*



- Press the SAVE button at the top of the Dashboard - you have now created a Dashboard combining the uploaded data and the existing Data Lakehouse
- Click on the VISUALS tab at the top of the page

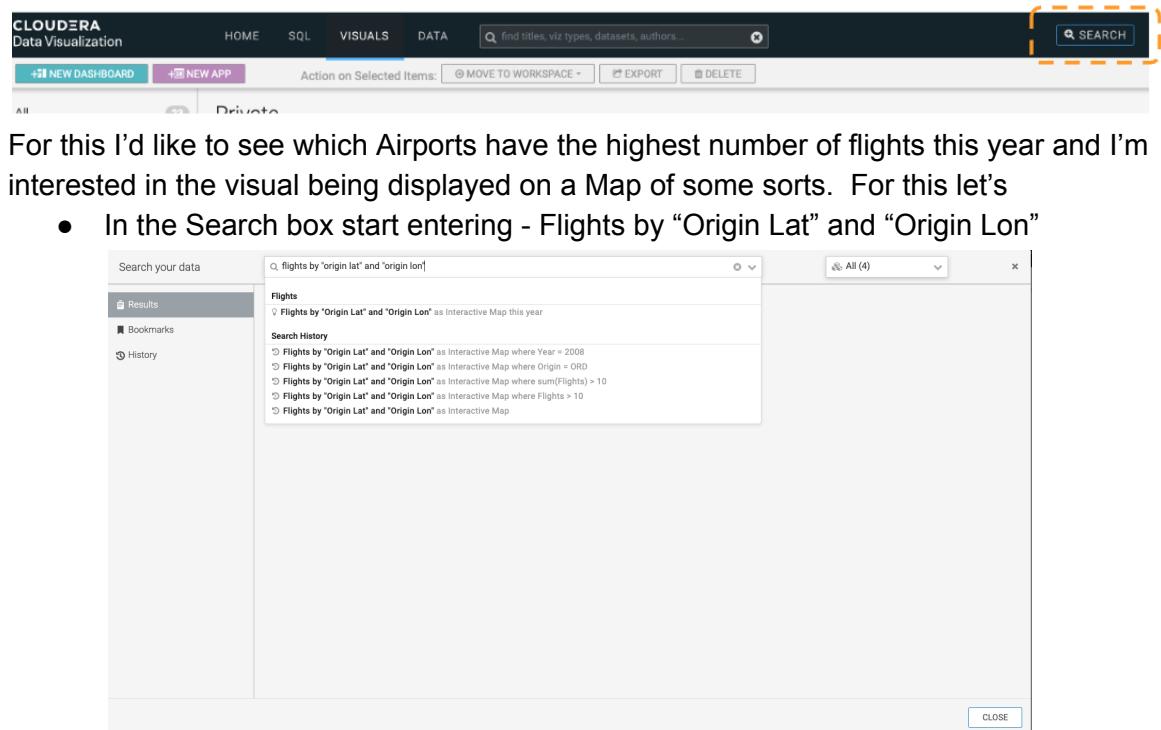


- On the right of the screen make sure you have Private selected to see the Dashboard you just created

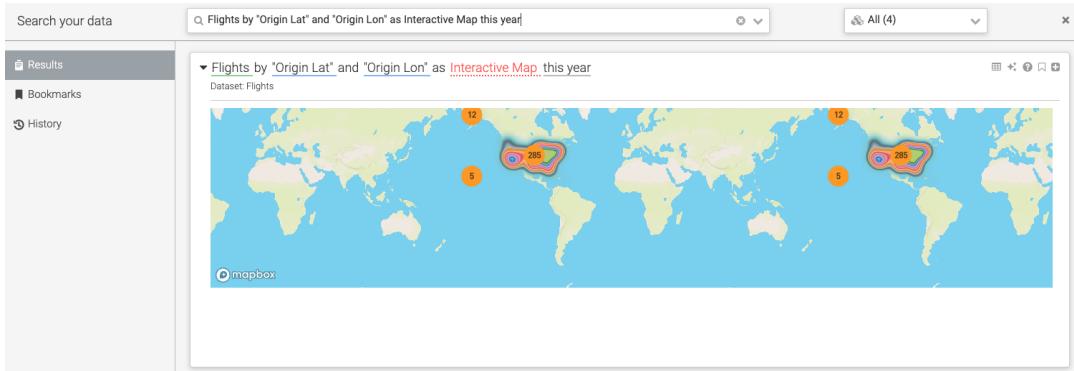


[OPTIONAL]

- Add another Visual to the Dashboard you just created using Natural Language Search
  - Instead of having to create a visual manually let's take a look at another way to add visuals. Using Natural Language Search to create visuals and then add them to Dashboards eliminates the step to manually configure a visual and allows users to do this in a much more simplified way
  - Click on the SEARCH button in the top right corner of the banner

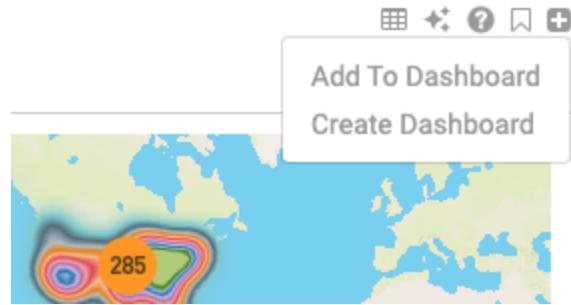


- You'll see below will show some helpful details to help with finishing your search. Under Flights select the entry - Flights by "Origin Lat" and "Origin Lon" as Interactive Map this year



This visualization shows what was asked for. From here you can continue to change what is displayed, Bookmark it to share or return to, or just continue on.

- This is displaying what I was interested in and this would be a good addition to the Dashboard.
- In the top right corner of the visual - click on the button and select Add to Dashboard



- Click on “<user-id> Lab 2 Self Service Open Data Lakehouse” to expand this section (for <user-id> look for your user id)

Pick a dashboard x

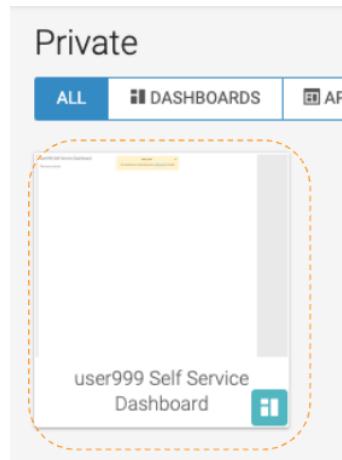
search dashboard ids, titles, datasets...	
	Flights ②
	Cereals ①
	Earthquake Data January 2019 ①
	Food Stores Inspection in NYC ③
	Global Information Security Threats ②
	Iris ①
	NYC Taxicab Rides ①
	Open Data Lakehouse HOL ②
	Restaurant Inspection SF ①
	Restaurant Inspection SF ①
	US State Populations Over Time ②
	World Life Expectancy ③
	user999 Lab 2 Self Service Open Data Lakehouse ①

- Click on the Dashboard you previously created

	World Life Expectancy ③
	user999 Lab 2 Self Service Open Data Lakehouse ①
<div style="border: 2px dashed orange; padding: 5px;"> <span style="border: 1px solid #ccc; padding: 2px;">user999 Self Service Dashboard</span> <span style="margin-left: 20px;">Sheets:</span> <span style="border: 1px solid #ccc; padding: 2px;">Sheet 1</span> <span style="margin-left: 10px;">▼</span> </div>	

- You should get a message indicating that this visualization was added to your dashboard

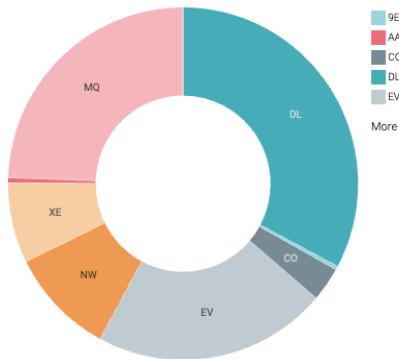
- Click on the CLOSE button in the bottom right corner of the Search window
- Now let's view the updated Dashboard
  - Click on the “<user-id> Self Service Dashboard” tile



- You should now see the following - we just now asked a question and created a Dashboard from it

user999 Self Service Dashboard

Airlines to Partner



Flights by "Origin Lat" and "Origin Lon" as Interactive Map this year



## Lab 3 - Administrator: “Productionalize” the Open Data Lakehouse

In this Lab you will be an Administrator and will use CDW to create and build an Open Data Lakehouse and take advantage of some of the key features with this approach.

Execute the following SQL/DDL/DML statements.

### 10. Stay in HUE for the CDW **Hive** Virtual Warehouse - **airlines-hive-vw-#**

- There should be an open CDW Browser tab, open the  browser tab
- Create a user Database to store all of the tables you will create in the following lab steps
  - In the SQL Editor window create a database for the remaining lab exercises, execute the following.
    - Delete the current query from the SQL Window
    - Copy & paste the SQL below

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

- The “\${...}” notation is a hint to HUE to create a parameter. As you copy/paste in this you will see that HUE determined that there is a parameter that needs to be entered.
- In the “user\_id” parameter box, enter your user id (see Lab Setup section)

- Execute the Query by clicking on the  button
- Check to see if the Database was created
  - Delete the current query from the SQL Window
  - Copy & paste the SQL below

```
SHOW DATABASES;
```

- Execute the Query by clicking on the  button to see that the database was created

### Results

```
DATABASE_NAME
...
user001_airlines
...
<user_id>_airlines
...
user100_airlines
...
```

There may be many databases, scroll & look for the one that starts with your <user\_id>

- Let's simulate existing tables in our Data Lakehouse
  - Create planes table in *Hive Table Format*, stored in Parquet file format
    - Delete the current query from the SQL Window
    - Copy & paste the SQL below

```
-- CREATE HIVE TABLE FORMAT STORED AS PARQUET
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');

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

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the ➤ button

The screenshot shows the Cloudera SQL Editor interface. At the top, there is a toolbar with icons for Hive, Add a name..., and Add a description... . Below the toolbar is the SQL query editor area. The query is:

```

1 -- CREATE HIVE TABLE FORMAT STORED AS PARQUET
2 drop table if exists ${user_id}_airlines.planes;
3
4 CREATE EXTERNAL TABLE ${user_id}_airlines.planes (
5     tailnum STRING, owner_type STRING, manufacturer STRING, issue_date STRING,
6     model STRING, status STRING, aircraft_type STRING, engine_type STRING, year INT
7 )
8 STORED AS PARQUET
9 TBLPROPERTIES ('external.table.purge'='true');
10
11 INSERT INTO ${user_id}_airlines.planes
12     SELECT * FROM airlines_csv.planes_csv;
  
```

The line number 11 is highlighted with a blue background. To the left of the editor, there is a status bar showing "3/3". Below the editor is a log window displaying the execution logs:

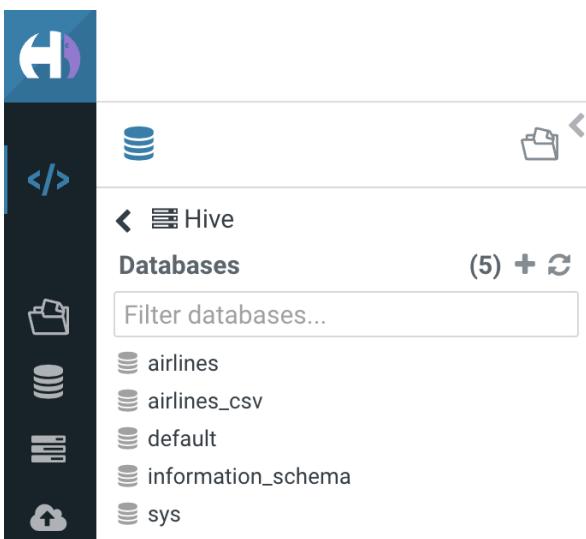
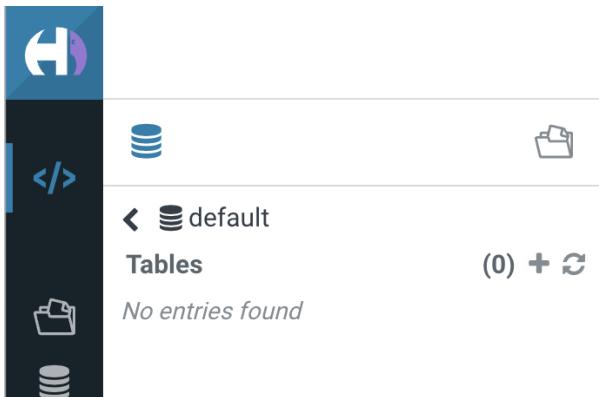
```

INFO : Total jobs = 1
INFO : Launching Job 1 out of 1
INFO : Starting task [Stage-1:MAPRED] in serial mode
INFO : Subscribed to counters: [] for queryId: hive_20230615142823_3087fc9-bf98-4b18-94fd-09e5874a535f
INFO : Tez session hasn't been created yet. Opening session
  
```

At the bottom of the screen, there are tabs for "Query History" and "Saved Queries", and a status bar indicating "a few seconds ago".

# CLOUDERA

- Switch Database to the user Database
  - On the left side of the SQL Editor, click on the < arrow next to default



- To the right of Databases click on the button to refresh the list of Databases

## Databases

Filter databases...

- airlines
- airlines\_csv
- default
- information\_schema
- sys
- user999\_airlines

- Click on your <user-id>\_airlines Database - this will allow you to track what has been created in your database through the next steps

user999\_airlines

Tables

(1) +

Filter...

planes

- Delete the current query from the SQL Window
- Copy & paste the SQL below

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

- Execute the Query by clicking on the button
  - In the output look for the following fields - scroll down to look for the following properties: Location, Table Type, and SerDe Library (this value should reflect the ParquetHiveSerDe, indicating this is a Hive Table Format)

18	Location	<a href="#">s3a://path-2-cloud-object-storage</a>	NULL
19	Table Type	EXTERNAL_TABLE	NULL
20	Table Parameters:	NULL	NULL
21	COLUMN_STATS_ACCURATE	0(BASIC_STATS)	
22	EXTERNAL	TRUE	
23	bucketing_version	2	
24	external_table.purge	true	
25	numFiles	1	
26	numRows	5029	
27	rowFormatSize	45261	
28	totalSize	111491	
29	transient_lastDdlTime	1668538014	
30	NULL	NULL	
31	# Storage Information	NULL	NULL
32	Serde Library:	<a href="#">org.apache.hadoop.hive.io.parquet.serde.ParquetHiveSerDe</a>	NULL
33	InputFormat:	<a href="#">org.apache.hadoop.hive.io.parquet.MappedParquetInputFormat</a>	NULL
34	OutputFormat:	<a href="#">org.apache.hadoop.hive.io.parquet.MappedParquetOutputFormat</a>	NULL

## 11. Build the Data Lakehouse

- Stay in HUE for the CDW Hive Virtual Warehouse - [airlines-hive-vw](#).
- Migrating Hive Table Format to Iceberg Table format - 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 two (2) options: 1) Utilize the in-place table Migration feature; or 2) Use Create Table as Select (CTAS)

- Option 1: Migrate the planes table in our Data Lakehouse from Hive Table Format to Iceberg Table Format using the Migration Utility.
  - Delete the current query from the SQL Window
  - Copy & paste the SQL below

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

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

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the button

- This migration to Iceberg happened in-place, there was no rewriting 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 - scroll down to look for the following properties: look for the following (see image with highlighted fields) key values: Table Type, Location (location of where table data is stored), SerDe Library, and in Table Parameters look for properties MIGRATE\_TO\_ICEBERG, storage\_handler, metadata\_location, and table\_type
  - Location - Data is stored in cloud storage in this case S3 in the same location as the Hive Table Format
  - Metadata\_location - Since there is no need to regenerate data files with in-place table migration, you save time generating Iceberg tables. Only metadata is regenerated, which points to source data files. Removes Hive Metastore as bottleneck.
  - Table\_type - indicates "ICEBERG" table format
  - Storage\_handler & SerDe Library - indicate what Serializer/Deserializer to use when reading/writing data in this case the "HiveIcebergSerDe"

18	Location:	s3a://path-2-cloud-object-storage	hive/user001_airlines.db/planes	NULL
19	Table Type:	EXTERNAL_TABLE		NULL
20	Table Parameters:	NULL		NULL
21		EXTERNAL		TRUE
22	MIGRATE_TO_ICEBERG	true		
23	bucketing_version	2		
24	engine.hive.enabled	true		
25	external.table.purge	true		
26	last_modified_by	jingails		
27	last_modified_time	1674156492		
28	metadata_location	s3a://path-2-cloud-object-storage		
29	numFiles	1		
30	numRows	5029		
31	previous_metadata_location	s3a://goes-se-sandbox01/warehouse/tablespace/extern		
32	rawDataSize	45261		
33	schema.name-mapping.default	[{\n  \\"field-id\\": 1,\n  \\"names\\": [\\\"latinum\\\"]}], {\n  \\"f\\		
34	storage_handler	org.apache.iceberg.mr.hive.HiveIcebergStorageHandler		
35	table_type	ICEBERG		
36	totalSize	111491		
37	transient_lastDdlTime	1674156492		
38	uuid	2280a4cc-9950-4d80-8615-2999eed8d30f		
39	write.format.default	parquet		
40	NULL	NULL		
41	# Storage Information	NULL		
42	SerDe Library:	org.apache.iceberg.mr.hive.HiveIcebergSerDe		NULL
43	InputFormat:	org.apache.iceberg.mr.hive.HiveIcebergInputFormat		NULL
44	OutputFormat:	org.apache.iceberg.mr.hive.HiveIcebergOutputFormat		NULL
45	Compressed:	No		NULL
46	Sort Columns:	[]		NULL

- Option 2: Create airports table in *Iceberg Table Format*, using Create Table As Select (CTAS). Notice the syntax to create an Iceberg Table within Hive is "**Stored by Iceberg**"
  - Delete the current query from the SQL Window

- Copy & paste the SQL below

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

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

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the ▶ button
  - In the output - look for the following key values (just like previously): Table Type, Location (location of where table data is stored), SerDe Library, and in Table Parameters look for properties MIGRATE\_TO\_ICEBERG, storage\_handler, metadata\_location, and table\_type
  - On the left you should see the airports table added to the list of Tables for your user Database
- Slowly changing dimension table airlines
  - Iceberg is fully ACID compliant and can execute Insert, Update, Delete, and Merge Into statements
  - Delete the current query from the SQL Window
  - Copy & paste the SQL below

```
-- SLOWLY CHANGING DIMENSION TABLE USE ACID CAPABILITIES OF ICEBERG
drop table if exists ${user_id}_airlines.airlines;

CREATE EXTERNAL TABLE ${user_id}_airlines.airlines (
  code string,
  description string
)
STORED BY ICEBERG
STORED AS PARQUET
tblproperties('format-version'='2');

-- LOAD DATA
INSERT INTO ${user_id}_airlines.airlines
  SELECT * FROM airlines_csv.airlines_csv;

-- Check data to see a few records
SELECT *
FROM ${user_id}_airlines.airlines
WHERE code IN ("04Q", "05Q");
```

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the ▶ button
  - This will create the Iceberg Table, load initial data, and display a few rows that will be modified in the next step
- Delete the current query from the SQL Window

- Copy & paste the SQL below

```
-- SLOWLY CHANGING DIMENSION TABLE USE ACID CAPABILITIES OF ICEBERG
MERGE INTO ${user_id}_airlines.airlines AS t
  USING airlines_csv.airlines_csv s ON t.code = s.code
WHEN MATCHED AND t.code = "04Q" THEN DELETE
WHEN MATCHED AND t.code = "05Q" THEN UPDATE SET description = "Comlux Aviation"
WHEN NOT MATCHED THEN INSERT VALUES (s.code, s.description);

-- Check data to see records were deleted or updated
SELECT *
FROM ${user_id}_airlines.airlines
WHERE code IN ("04Q", "05Q");
```

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the ▶ button
  - The Merge Into statement will check to see if a record needs to be Deleted (when the record key matches and the code = 04Q), when a new record needs to be Inserted (key doesn't match), and when an Update is needed (when the key matches and the code = 05Q)
  - Compare the output to see that only one record is returned with the description set to "Comlux Aviation"
- Creating an Iceberg Table - for this step create a partitioned table, in Iceberg Table Format, stored in Parquet File Format. Optionally, you could specify other File Formats, the supported formats for Iceberg are: Parquet, ORC, and Avro.
  - Delete the current query from the SQL Window
  - Copy & paste the SQL below

```
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,
  crsarrrtime int, uniquecarrier string, flightnum int, tailnum string,
  actualelapsedtime int, crselapsedtime int, airtime int, arrdelay int,
  depdelay int, origin string, dest string, distance int, taxiin int,
  taxiout int, cancelled int, cancellationcode string, diverted string,
  carrierdelay int, weatherdelay int, nasdelay int, securitydelay int,
  lateaircraftdelay int
)
PARTITIONED BY (year int)
STORED BY ICEBERG
STORED AS PARQUET
tblproperties ('format-version'='2');

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

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the ▶ button

- Looking at the SHOW CREATE TABLE output, scroll down and notice the output is the unformatted version of the Describe Formatted as we would expect. The main item to pay attention to here is the PARTITIONED BY SPEC, currently we've partitioned by just the "year" column

```

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://path-2-cloud-object-storage`/user001_airlines.db/flights'
40 TBLPROPERTIES (
41   `bucketing_version`='2',
42   `engine.hive.enabled`='true',
43   `metastore_location`='s3a://path-2-cloud-object-storage'`/user001_airlines.db/flights/metadata/00000-2ba52a08-8af7-4eeb-87f7-eb4ad4133be5.metadata.json',
44   `serialization.format`='1',
45   `table_type`='ICEBERG',
46   `transient_lastDdlTime`='1674157757',
47   `uid`='837acSee-79e6-48d2-bf43-a82c04b1d2ba',
48   `write.format.default`='parquet')

```

- When 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.)
  - Delete the current query from the SQL Window
  - Copy & paste the SQL below

```

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

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

```

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the ▶ button
  - The Insert will insert data from years 1995 to 2006
  - 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)

The screenshot shows the Cloudera SQL Editor interface. At the top, there are tabs for 'Query History', 'Saved Queries', and 'Results (12)'. Below these, there's a table with the header 'year'. The results show the following data:

year	
1	2006
2	2005
3	2004
4	2003
5	2002
6	2001
7	2000
8	1999
9	1998
10	1997
11	1996
12	1995

At the bottom of the editor, there's a command history window showing the executed SQL statements and their log output.

```

1 INSERT INTO ${user_id}_airlines.flights
2 SELECT * FROM airlines_csv.flights_csv
3 WHERE year <= 2006;
4
5 SELECT year, count(*)
6 FROM ${user_id}_airlines.flights
7 GROUP BY year
8 ORDER BY year desc;
9
10 user_id user001
11
12 INFO : ANONYMOUS:runSqlQuery_1:
INFO : INPUT_FILES_Map_1: 1
INFO : RAM_INPUT_SPLITS_Map_1: 15
INFO : Completed executing command(queryId=hive_20230119204024_5863964c-9b1f-4098-a0e9-58585d4a257); Time taken: 2.54 seconds
INFO : OK

```

## 12. Performance improvements

- Check that you created the tables for the Data Lakehouse - on the list of tables to the left of the Editor window you should see the following tables

Tables	(4) +
Filter...	
airlines	
airports	
flights	
planes	

- Iceberg in-place Partition Evolution [Performance Optimization]

- One of the key features for Iceberg tables is the ability to evolve the partition that is being used over time
    - Delete the current query from the SQL Window
    - Copy & paste the SQL below

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

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

- Select all of the content in the SQL Editor
- Execute the Query by clicking on the button
  - This was an in-place Partition Evolution, meaning that the existing data is not rewritten as part of the ALTER TABLE execution. What will happen is the next data that is loaded will use the new Partition definition.
  - In the output scroll to see the PARTITIONED BY SPEC to see that it is now both year and month

```
30   `year` int)
31   PARTITIONED BY SPEC (
32     year,
33     month)
```

## 13. Performance Improvements - Open HUE for the CDW Impala Virtual Warehouse - [airlines-impala-vw](#).

- There should be an open CDW Browser tab, open the browser tab
- Click on the [airlines-impala-vw#](#) tile
  - In the upper right corner click on the HUE button to enter into the SQL Editor



- Load new data into the flights table using the NEW partition definition - this will load new data to take advantage of the new partition specification. This also shows how Iceberg also supports

multiple engines by allowing both Hive & Impala to create, load, query, and or modify Iceberg tables.

- Copy & Paste the following in the SQL Editor window

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

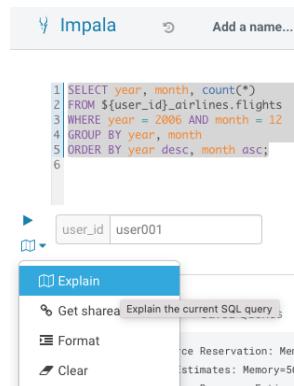
- Execute the Query by clicking on the ➤ button
- Run Explain Plans against some typical analytic queries we might run to see what happens with this new Partition definition.
  - In Impala we are in another engine which is another key feature of Iceberg - multi-function (or multiple engine) analytics. No need to copy the data or do more work to allow access to the same data.
  - Copy/paste the following in the Editor, but do not execute the query

```
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;
```

- In the “user\_id” parameter box, enter your user id (see Lab Setup section)



- Instead select (highlight) the statement click the 📄 button, which is right below the Execute ( ➤ ) button and select Explain



- In the output notice the amount of data that needs to be scanned for this query (it would represent the volume of 1 years worth of data - about 139MB). Up to this point the data was loaded using the Partition of just a year.

Query History    Saved Queries    Explain

```

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.
user999_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 [user999_airlines.flights]
|   S3 partitions=1/1 files=1 size=139.31MB
|   predicates: 'month' = 12, 'year' = 2006
|   row-size=8B cardinality=7.14M

```

- Copy/paste the following in the Editor, but do not execute the query

```

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;

```

- Instead select (highlight) the statement click the  button, which is right below the Execute (▶) button and select Explain

- In the output notice the amount of data that needs to be scanned for this query, about 10MB, is significantly less than that of the first, 138MB. This shows an important capability, Partition Pruning. Meaning that much less data is being scanned for this query and only the selected month of data is being scanned vs scanning the entire year of data. This should result in much faster query execution times.

Query History    Saved Queries    Explain

```

Max Per-Host Resource Reservation: Memory=86.00MB Threads=3
Per-Host Resource Estimates: Memory=308MB
Dedicated Coordinator Resource Estimate: Memory=119MB
WARNING: The following tables are missing relevant table and/or column statistics.
user999_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=614.14K
|
04:AGGREGATE [FINALIZE]
|   output: count:merge(*)
|   group by: `year`, `month`
|   row-size=16B cardinality=614.14K
|
03:EXCHANGE [HASH(`year`, `month`)]
|
01:AGGREGATE [STREAMING]
|   output: count(*)
|   group by: `year`, `month`
|   row-size=16B cardinality=614.14K
|
00:SCAN S3 [user999_airlines.flights]
|   S3 partitions=1/1 files=1 size=10.27MB
|   predicates: `month` = 12, `year` = 2007
|   row-size=8B cardinality=614.14K

```

- Iceberg Snapshots Time Travel - in the previous steps we have been loading data into the flights Iceberg table.
  - Each time data was changed in our Iceberg tables, a Snapshot is automatically captured. This is important for many reasons but the main point of the Snapshot is to ensure eventual consistency and allow for multiple reads/writes concurrently (from various engines or the same engine).
  - Show snapshots
    - Delete the current query from the SQL Window
    - Copy & paste the SQL below

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

- Execute the Query by clicking on the  button

In the output there should be 2 Snapshots, in the example below there are 3 snapshots as this example shows that data was also loaded via Impala and not just Hive. Also, keep in mind we have been reading/writing data from/to the Iceberg table from both Hive & Impala which is indicated by the () in the callouts below. This is an important aspect of Iceberg Tables is 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)

The screenshot shows the Cloudera Impala Editor interface. At the top, there's a header with tabs for 'Impala', 'Add a name...', and 'Add a description...'. Below the header, a query window contains the command: 'DESCRIBE HISTORY \${user\_id}\_airlines.flights;'. A dropdown menu is open next to the user\_id placeholder, showing 'user\_id' and 'user001'. The status bar at the bottom right says '0.756 ut'. Below the query window, a message says 'No logs available at this moment.' To the right of the results table, three callout boxes point to specific rows:

- 'Initial inserts for flights on or before year 2006 (Hive)' points to the first row (creation\_time: 2023-01-19 20:40:16.453000000, snapshot\_id: 1517273939679890388, parent\_id: NULL, is\_current\_ancestor: TRUE).
- 'Flights for year 2007 (Hive)' points to the second row (creation\_time: 2023-01-20 02:45:32.160000000, snapshot\_id: 2995408773555868055, parent\_id: 1517273939679890388, is\_current\_ancestor: TRUE).
- 'Flights for year 2008 (Impala)' points to the third row (creation\_time: 2023-01-20 02:55:36.669000000, snapshot\_id: 1112270214077101459, parent\_id: 2995408773555868055, is\_current\_ancestor: TRUE).

The results table has columns: creation\_time, snapshot\_id, parent\_id, and is\_current\_ancestor. It contains three rows of data.

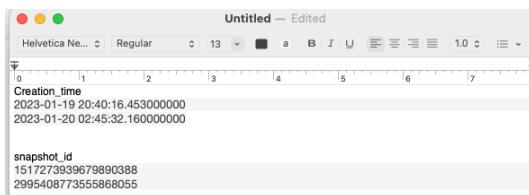
	creation_time	snapshot_id	parent_id	is_current_ancestor
1	2023-01-19 20:40:16.453000000	1517273939679890388	NULL	TRUE
2	2023-01-20 02:45:32.160000000	2995408773555868055	1517273939679890388	TRUE
3	2023-01-20 02:55:36.669000000	1112270214077101459	2995408773555868055	TRUE

## ■ Get Details for Snapshots - open a text editor/notepad

The screenshot shows the Cloudera Impala Editor interface with a results table. The table has columns: creation\_time, snapshot\_id, parent\_id, and is\_current\_ancestor. It contains three rows of data.

	creation_time	snapshot_id	parent_id	is_current_ancestor
1	2023-01-19 20:40:16.453000000	1517273939679890388	NULL	TRUE
2	2023-01-20 02:45:32.160000000	2995408773555868055	1517273939679890388	TRUE
3	2023-01-20 02:55:36.669000000	1112270214077101459	2995408773555868055	TRUE

Text Editor - copy in a couple creation\_time's and snapshot\_id's



- ## ■ Iceberg Time Travel [Table Maintenance] - copy/paste the following data into the Impala Editor, but do not execute.
- Delete the current query from the SQL Window
  - Copy & paste the SQL below

```
-- 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;
```

- ## ■ Once you copy this SQL into the Editor you will see 2 new parameters - create\_ts and snapshot\_id

The screenshot shows the Cloudera Impala Editor interface. A dropdown menu is open next to the user\_id placeholder, showing 'user\_id' and 'user001'. To the right of the user\_id input field are two empty input fields with orange borders, labeled 'create\_ts' and 'snapshot\_id' respectively.

- The first SELECT statement will use the `create_ts`
  - In the `create_ts` parameter box enter a date/time (this can be relative or specific timestamp). From your Text Editor copy the second entry under `creation_time`, paste it into the `create_ts` parameter. For this Time Travel you can use relative time periods and don't have to enter the specific timestamp for the Snapshot.
  - Highlight the first SELECT statement, and execute it

The screenshot shows the Impala Query Editor interface. The query editor window contains the following code:

```

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
  
```

Below the code, there are three input fields: `user_id` (set to `user001`), `create_ts` (set to `2023-01-20`), and `snapshot_id` (empty). The results tab shows a table with the following data:

year	count(*)
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

At the bottom of the results table, there is a note: "Latest admission queue reason : Waiting for executors to start. Only DDL queries and queries scheduled only on red" and two completed queries: "Query 2a4804e20ababc64:4ec4b42b00000000 100% Complete (12 out of 12)" and "Query 2a4804e20ababc64:4ec4b42b00000000 100% Complete (12 out of 12)".

- This returned the the data as it was in our initial load
- The second SELECT statement will use the `snapshot_id`
  - From your Text Editor copy the second entry under `snapshot_id`, paste it into the `snapshot_id` parameter box
  - Highlight the second SELECT statement, and execute it

The screenshot shows the Impala Query Editor interface. The query editor window contains the same code as the previous screenshot, but the `snapshot_id` field now contains the value `2995408773555868055`. The results tab shows a table with the following data:

year	count(*)
1 2007	7453215
2 2006	7141922
3 2005	7140596
4 2004	7129270
5 2003	6488540
6 2002	5271359
7 2001	5967780
8 2000	5683047
9 1999	5527884
10 1998	5384721
11 1997	5411843
12 1996	5351983
13 1995	5327435

At the bottom of the results table, there are three completed queries: "Query 884a3b141e3a84ac:5674683c00000000 100% Complete (19 out of 19)", "Query 884a3b141e3a84ac:5674683c00000000 100% Complete (19 out of 19)", and "Query 884a3b141e3a84ac:5674683c00000000 100% Complete (19 out of 19)".

- This returns the data for the specific Snapshot ID that was specified in the query which was the data for the initial insert for data up to year 2006 and the insert for data for year 2007

- Security & Governance - Now that performance has been optimized, we are almost ready to release this to the users. Before we do that we need to apply security - Security is always on and can be defined with fine grained access control. **To learn more about Security & Governance in CDP - work with your Breakout Room Moderator to schedule time to go thru the Optional Lab at the end of this document**

- A security policy has already been created. This is how it is defined

Policy Details:

Policy Type	Masking
Policy Name *	jingalls-iceberg-fgac
Policy Label	Policy Label
Hive Database *	jingalls_airlines
Hive Table *	planes
Hive Column *	tailnum
Description	
Audit Logging	Yes

Mask Conditions:

Select Role	Select Group	Select User	Access Types	Select Masking Option
Select Roles	Select Groups		select	Hash

- Delete the current query from the SQL Window
- Copy & paste the SQL below      Select all of the content in the SQL Editor
- Execute the Query by clicking on the ▶ button

```
1 SELECT * FROM airlines.planes;
2
3
```

▶

Latest submitted query (version 1, waiting for execution to start. Only one queries per queries scheduled only on the coordinator (version 1).  
UM\_NODES set to 1 or when small query optimization is triggered) can currently run.

Query 17416573231856ea:cd8c269600000000 100% Complete (1 out of 1)  
Query 17416573231856ea:cd8c269600000000 100% Complete (1 out of 1)  
Query 17416573231856ea:cd8c269600000000 100% Complete (1 out of 1)

17416573231856ea:cd8c269600000000

Query History      Saved Queries      Results (5,029)

tailnum	owner_type	manufacturer	issue_date	moc
1 82d6b5373e2aeebd8e9a437f5a7680a21b11f1753721e9cd0968cfa8e8d4dd31				
2 9c02c0930c86b5c841989aa3ddafe9e5a84c9e7780d461e4259dd53fffc7ab				
3 b16ae68d29c050a77412a89d54ae322ab7181c9dbc0e7f63ee983c6d927c8773				
4 a8c94ab3f0aea271b3993f8623ce46c3a5217dd5f9e7532cf05edf39fb2ebf7d				
5 4cd32f01791cb3bb791650daddbcc701b269bea9cfe4edca9e070efa53af5e7				
6 f07afae0cc8e13e9c2602d82e0e691e79cb5eebfbee500020b23c950f02f95de				
7 a2cc953614cd5ee3fd5bedd65beb9897e8234a17c05ae26372ce83b522b1efaf1				
8 0bd27cb31e035e19c7f04b82a53c6d835bd4c6f23569b3065104dbe4e2fc81b9				
9 74876a7424a5e2f67ebec433fd3ebe058cfb2d9b75742aed61eb00d98047130				
10 b1231b80352bcbeae01263b29582437e5a551cd8bbd845c9e0240b75fd4f58f0				
11 c5a63f3d360fa51a6692879211e00h2d4hahf77rfd477q92R0ia4dr-hn022r6227fa				

- You will see that the “tailnum” column is masked so you are unable to see the actual plane’s tail number, instead you see the Hashed value of the tailnum

## (Optional Bonus Material)

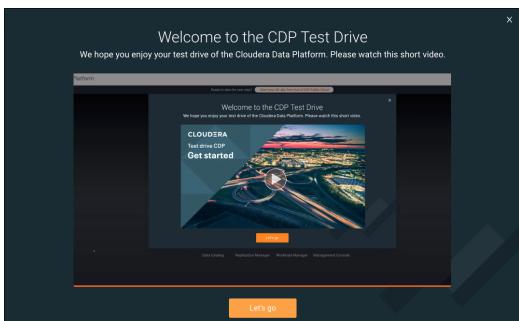
### Optional Lab 1 - CDW Tour

In this Lab you will explore how to take advantage of CDW.

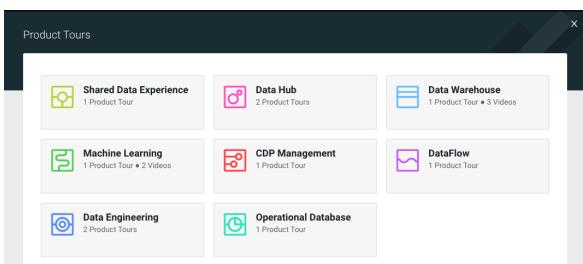
14. Click on the “Get Started with a Tour” at the top of the CDP Home screen



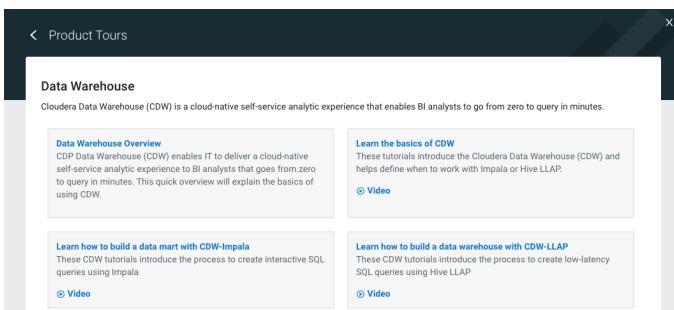
15. [Optional] Watch the “Test Drive” Video on the CDP Home screen (video is ~3 minutes)



16. Click on the “Let’s go” button at the bottom of the video, you can take tours of any of the Data Services
17. Click on the Data Warehouse tour tile



- Here you see the content you can take advantage of - there are 3 tiles where you can watch videos on an area of interest, you will see a “> Video” link on the tile if this is a video, and there is also a tour.



- Click on the “Data Warehouse Overview” tile, this is a click through tour of CDW

- Perform all of the actions for this click through tour (~5 minutes)
  - Once completed click on the “X” in the top right corner to close the window
18. On the Product Tours screen, explore the various topics to learn more about Cloudera Data Warehouse
- 

## Optional Lab 2 - Self Service File Upload

In this lab you will learn how to upload a small file to run SQL queries against where you could join to existing data in your Data Lakehouse.

19. Upload Passenger Ticket data file

- Below highlights the 3 steps to upload this file: 1) Open the Importer, 2) Pick a file & options for the file upload, 3) Name the table, specify table options, and table metadata (column names, data types, etc.)

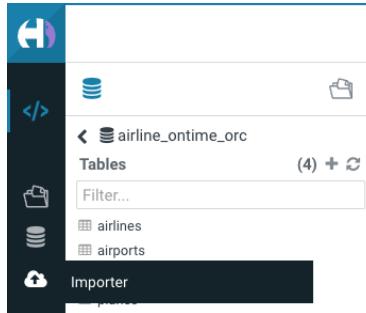
1) Left navigation menu showing 'Importer' selected.

2) 'Importer' page showing 'Pick data from localfile' and a CSV file chosen.

3) 'Importer' page showing table metadata defined:

Name	Column	Type	Format
id	id	integer	10
legFlightnum	legFlightnum	string	10
legFromcarrier	legFromcarrier	string	10
legLength	legLength	double	10
legIsdirect	legIsdirect	boolean	10
legJourneyTime	legJourneyTime	double	10
legJourneyWeek	legJourneyWeek	double	10
legLastUpdate	legLastUpdate	date	10
legTerminal	legTerminal	string	10

- On the left navigation menu click the button, and select Importer



- On the “Pick Data from Local File” page
  - In the Select Type, choose “Small Local File”, it should be the default value

1) Pick data from localfile

SOURCE

Type: Small Local File

Choose File No file chosen

- Click the **Choose File** button, and browse to the “passengertickets\_1k.csv” file

**SOURCE**

Type: Small Local File  
Choose File: passengertickets\_1k.csv

**FORMAT**

File Type: CSV File  
Field Separator: Comma (,) Record Separator: Newline Quote Character: Double Quote  
 Has Header

**PREVIEW**

ticketnumber	leg1flightnum	leg1uniquecarrier	leg1origin	leg1dest	leg1month	leg1dayofmonth	leg1dayofweek	leg1deptime	leg1arritime
6887528732333	662	NW	PHX	MEM	10	10	2	1407	1907
2331153269614	550	NW	LAX	MEM	10	10	2	555	1117
4191763179365	282	NW	SAT	MEM	10	10	2	556	731
7435945889948	1036	NW	MCI	MEM	10	10	2	617	727

Ensure that the PREVIEW looks good. You could make changes to the CSV options or import other file types like JSON using this Importer

- Click the [Next](#) button
- On the “Move it to table <table-name>” page
  - Under DESTINATION, change Name to **<user-id>\_airlines.unique\_tickets** (replace <user-id> with your user id you logged in with)

**DESTINATION**

Dialect: Hive  
Name: airlines.unique\_tickets\_1k

**PROPERTIES**

Format: Text  
Extras: [Edit](#)  
Partitions: [Add partition](#)

**FIELDS**

Name	Type	Value	Name	Type	Value
ticketnumber	bigint	6887528732333	2331153269614		
leg1flightnum	bigint	662	550		
leg1uniquecarrier	string	NW	NW		
leg1origin	tinyint	PHX	LAX		
leg1dest	smallint	MEM	MEM		
leg1month	int	10	10		
leg1dayofmonth	bigint	10	10		
leg1dayofweek	boolean				
leg1deptime	float				
leg1arritime	string				

- Under FIELDS, change data types with “bigint” to “int” for all the fields except the “ticketnumber” field, it can remain a bigint data type

Name	Type	Value	Name	Type	Value
ticketnumber	bigint	6887528732333	2331153269614		
leg1flightnum	bigint	662	550		
leg1uniquecarrier	string	NW	NW		
leg1origin	int	PHX	LAX		
leg1dest	smallint	MEM	MEM		
leg1month	int	10	10		
leg1dayofmonth	bigint	10	10		
leg1dayofweek	boolean				
leg1deptime	float				
leg1arritime	string				

- The final fields section should look like the following:

FIELDS				
Name	ticketnumber	Type	bigint	6887528732333
Name	leg1flightnum	Type	int	662
Name	leg1uniquecarrier	Type	string	NW
Name	leg1origin	Type	string	PHX
Name	leg1dest	Type	string	MEM
Name	leg1month	Type	int	10
Name	leg1dayofmonth	Type	int	10
Name	leg1dayofweek	Type	int	2
Name	leg1deptime	Type	int	1407
Name	leg1arrtime	Type	int	1907
Name	leg2flightnum	Type	int	175
Name	leg2uniquecarrier	Type	string	NW
Name	leg2origin	Type	string	MEM
Name	leg2dest	Type	string	BTR
Name	leg2month	Type	int	10
Name	leg2dayofmonth	Type	int	10
Name	leg2dayofweek	Type	int	2
Name	leg2deptime	Type	int	2115
Name	leg2arrtime	Type	int	2221

Back Submit

- Click the Submit button - this will create a new table in your database named `unique_tickets` in a Hive table format. Later in the labs you will learn how to migrate this to an Iceberg table. You will be taken to the Table Browser and see a status like the following screen (top right):

The screenshot shows the Cloudera Table Browser interface. On the left, under 'Databases > default > passengertickets\_1k\_csv', there are tabs for 'Overview', 'Sample (100)', and 'Details'. The 'Details' tab is active, showing the following information:

- PROPERTIES:** Table, Managed and stored in location, Created by jingalls on 06/12/2023 3:50 PM-05:00.
- STATS:** Rows 1000, Total size 13.98 KB, Data last updated on 06/12/2023 3:50 PM-05:00.
- SCHEMA:** Column (0) Type Description Sample

On the right side of the browser, there is a separate window titled 'Creating table default.passengertickets\_1k\_csv' showing the progress of the task:

- Output:** Progress: 0.72s, 0.16s, 0.14s, 2.01s.
- Task History:** A few seconds ago, Creating table default.passengertickets\_1k\_csv.

- Once the table is created click on the Details tab of the Table Browser to see the table type and other information on this table. The following highlighted pieces of information show that this table is a Managed Table that is in using a SerDe that shows this is an ORC File Format & in Hive Table Format

**Table Browser**

Databases > default > passengertickets\_1k\_csv

Overview Sample (100) Details

**DETAILED TABLE INFORMATION**

Database:	default
OwnerType:	USER
Owner:	jingalls
CreateTime:	Mon Jun 12 20:50:23 UTC 2023
LastAccessTime:	UNKNOWN
Retention:	0
Location:	s3a://jing-cdp-bucket-2/data/warehouse/tablespace/managed/hive/passengertickets_1k_csv
Table Type:	MANAGED_TABLE
Table	
Parameters:	
COLUMN_STATS_ACCURATE	{"BA":
bucketing_version	{"leg':
numFiles	2
numRows	1
rawDataSize	1000
totalSize	0
transactional	14314
transactional_properties	true
transient_lastDdlTime	defau
	16866

**STORAGE INFORMATION**

SerDe Library:	org.apache.hadoop.hive.ql.io.orc.OrcSerde
InputFormat:	org.apache.hadoop.hive.ql.io.orc.OrcInputFormat
OutputFormat:	org.apache.hadoop.hive.ql.io.orc.OrcOutputFormat
Compressed:	No
Num Buckets:	-1

- You can also view a sample of the data in this table by clicking on the [Sample \(100\)](#) tab

## Optional Lab 3 - Performance Optimizations (Impala VW)

In this Lab we will take a 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.

- Materialized Views [Performance Optimization] - this can be used for both Iceberg tables and Hive Tables to improve performance
  - Open HUE for the CDW **Hive** Virtual Warehouse - [airlines-hive-vw](#). Copy/paste the following, make sure to highlight the entire block, and execute the following

```
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;

```

- 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.

code	description	month	cancelled
AQ	Alaska Airlines Inc.	2	192
AQ	Alaska Airlines Inc.	3	110
TZ	ATA Airlines d/b/a ATA	3	248
9E	Pinnacle Airlines Inc	4	734
EV	Atlantic Southeast Airlines	4	2926
HA	Hawaiian Airlines Inc.	4	58
US	US Airways Inc. (Merged with America West 9/05. Reporting for both starting 10/07.)	4	9831
B6	JetBlue Airways	5	105
EV	Atlantic Southeast Airlines	5	2239
MQ	American Eagle Airline Inc.	5	10532
US	US Airways Inc. (Merged with America West 9/05. Reporting for both starting 10/07.)	7	17853
HP	America West Airlines Inc. (Merged with US Airways 9/05. Stopped reporting 10/07.)	8	3873
YV	Mesa Airlines Inc.	8	2109
FL	AirTran Airways Corporation	9	1564
CO	Continental Air Lines Inc.	10	3693

- Let's take a look at the Query Plan that was used to execute this query
  - On the left menu select Jobs

- This will take you to the Jobs Browser, select the Queries tab to the right of the Job Browser header

- This is where an Admin will go when - a query has run amok and then figures out what to do next. 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 ID	App
✓	SELECT airlines.code AS code, MIN(airlines.de...	jingalls	fights (user001_airlines), airlines_orc (user0...	-	-	9 minutes ago	00:00:12	dag.1674222453092_0001_6#	app
✓	INSERT INTO user001_airlines flights SELECT ...	jingalls	fights.csv (user001_airlines_new)	fights (user001_airlines)	-	12 hours ago	00:06:21	dag.1674182535130_0000_1	app
✓	SELECT year(count(*)) FROM user001_airlines...	jingalls	fights (user001_airlines)	-	-	13 hours ago	00:03:11	-	-

- Hoover over and click on the Query we just executed (it should be the first row)

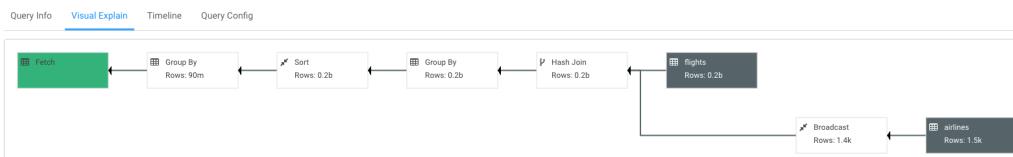
Status	Query	Queue	User	Tables Read
✓	SELECT airlines.code AS code, MIN(airlines.de...	jingalls	airlines (user001_ai...	
✓	SELECT airlines.code AS code, MIN(airlines.description) AS description, flights.month AS month, sum(flights.cancelled) AS cancelled FROM user001_airlines.flights flights, user001_airlines.airlines airlines WHERE flights.uniquecarrier = airlines.code group by airlines.code, flights.month	jingalls	airlines (user001_air...	
✓	SELECT airlines.code AS code, MIN(airlines.de...	jingalls	airlines (user001_a...	

- This is where you can analyze queries at a deep level. For this lab let's take a look at the Explain details, click on Visual Explain tab

Query Info: SELECT airlines.code AS code, MIN(airlines.description) AS description, flights.month AS month, sum(flights.cancelled) AS cancelled FROM user001\_airlines.flights flights, user001\_airlines.airlines airlines WHERE flights.uniquecarrier = airlines.code group by airlines.code, flights.month

Visual Explain: This section shows the execution plan for the query. It includes details like Start Time (2 minutes ago), End Time, Duration, Table Read (airlines (user001\_airlines)), Table Written, Application ID, DAG ID, Session ID, CLAP API ID, Thread, and Queue (None).

- 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 </> button on the left menu, and click Editor, to return the Editor Window

Editor: \${user\_id}\_airlines.traffic\_cancel\_airlines

Tables: (5) + Filter... airlines\_dim\_updates

- 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

```

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 Materialized View traffic\_cancel\_airlines will be displayed in Results and in the left table/view list with a  (View) icon to the left of the MV

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

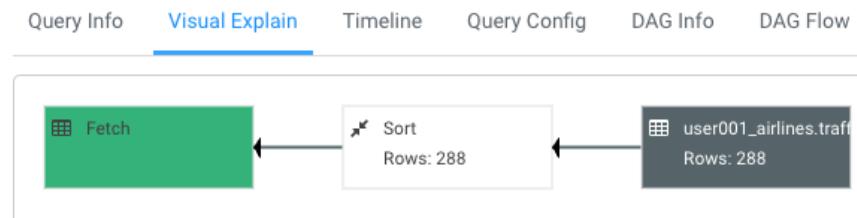
- Run Dashboard Query again to see usage of the MV - Copy/paste the following, make sure to highlight the entire block, and execute the following. This time an Order By was added to make this query have to do more work.

```

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 , ${user_id}_airlines.airlines airlines
 WHERE flights.uniquecarrier = airlines.code
 group by airlines.code, flights.month
 order by airlines.code;

```

- Let's take a 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
  - Hoover over & click on the Query just executed (should be the first row)
  - Click on 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.



- Materialized views also support incremental refreshes when the data from the tables used for the MVs is updated. Please visit the CDW documentation for more information.

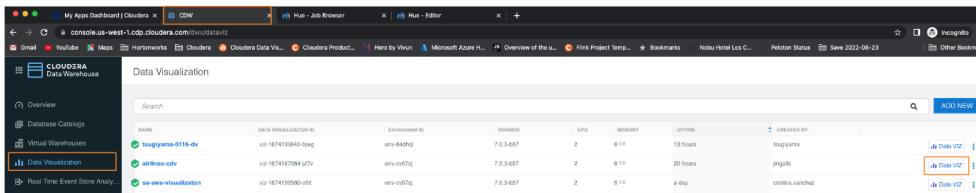
- Return to Query Editor - click on the </> button on the left menu, and click Editor, to return the Editor Window
- 

## Optional Lab 4 - Data Visualization

In this lab you will build a Logistics Dashboard using Cloudera Data Visualization. The Dashboard will include details about flight delays and cancellations. This will be a completely new use case independent of the SMG Use Case, the analysis could be a catalyst to lead to a Data Science application to predict the probability of a flight being canceled or not.

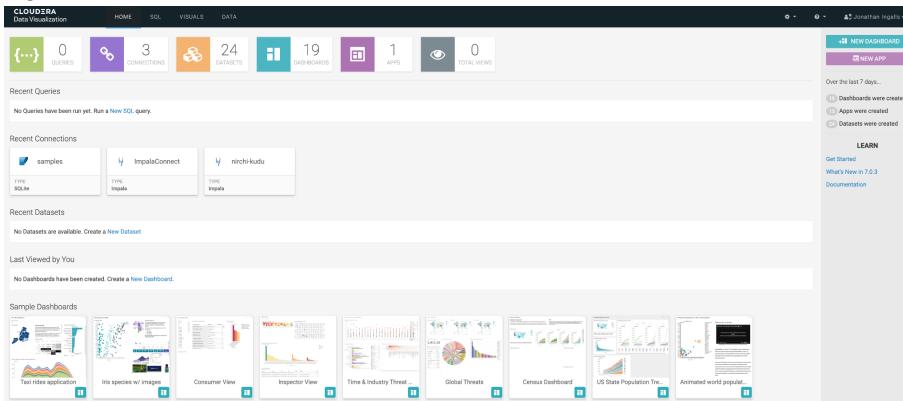
### 21. Open Cloudera Data Visualization (CDV or Data Viz)

- Go to browser tab with CDW open
- On the left navigation panel click “Data Visualization”
- On the row with airlines-cdv, click the Data VIZ button



- If you see the “What’s New” page, you can read it, or click on the GOT IT button

### 22. Home page



- 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
  - Home - this is the starting point; it shows some statistics at the top, followed by some quick access details to recent content - Queries, Connections, Datasets, and Dashboards
  - SQL - allows you to manually build queries against data to perform quick discovery against the data. Below is an example of a query that was built and Run

The screenshot shows the Cloudera Data Visualization interface. At the top, there's a navigation bar with links for HOME, SQL, VISUALS, and DATA. A search bar is also present. Below the navigation, a "Data Connection" dropdown is set to "samples". On the left, a "Connection Explorer" sidebar lists various databases and tables, such as "main", "census\_pop", "cereals", "chicago\_govt\_pay", "earthquake\_dta2019", "infineq\_1559", "irs", "restaurant\_scores\_lives\_ata...", "retail\_food\_store\_inspectio...", "superstore\_sales", "trips", "trips\_detail", "us\_counties", and "world\_life\_expectancy". The main area is titled "Enter SQL below" and contains a code editor with the following SQL query:

```
1 select * from main.cereals;
```

Below the code editor, there are buttons for "RUN", "SAVE QUERY", "SAVE AS DATASET", and "NEW DASHBOARD". A checkbox labeled "Add in a 'LIMIT 100' clause to any SQL select query that does not have a limit clause" is checked. There are also tabs for "Query History", "Saved Queries", and "Results". The "Results" tab is selected, showing the output of the query as a table:

cereal_name	manufacturer_code	cold_or_hot	calories	protein_grams	fat_grams	sodium_mg	dietary_fiber_grams	complex_carbohydrates_grams	sugars_grams	display_shelf	potassium_mg	vita
100% Bran	N	C	70	4	1	130	10	5	6	3	280	25
100% Natural Bran	O	C	120	3	5	15	2	8	8	3	135	0
All Bran	K	C	70	4	1	260	9	7	5	3	320	25
All Bran, with_Extra_Fiber	K	C	50	4	0	140	14	8	0	3	330	25
Almond Delight	B	C	110	2	2	200	1	14	8	3	-1	25
Apple_Cinnamon_Cheerios	G	C	110	2	2	180			10	1	70	25
Apple_Jacks	K	C	110	2	0	125	1	11	14	2	30	25
Basic_A	G	C	130	3	2	210	2	18	8	3	100	25
Bran_Chez	R	C	90	2	1	200	4	15	6	1	125	25

- Visuals - an area for viewing/building/modifying visuals, dashboards, and applications

The screenshot shows the Cloudera Data Visualization interface with the "VISUALS" tab selected in the top banner. The left sidebar shows "My Favorites" and "WORKSPACES". The main area displays several visualizations, including "Deficiency Details: <>county:Queens>>", "State of NYC", "Store Details<->owner.name", and "Cereal Comparisons".

- Data - interface for access to datasets, connections, and the Connection Explorer

23. Build a Dataset (aka. Metadata Layer or Data Model) - 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

The screenshot shows the Cloudera Data Visualization interface with the "DATA" tab selected in the top banner. The left sidebar shows "All Connections" and "samples". The main area lists datasets under "Title/Table":

- Food Stores Inspection in NYC (main.retail\_food\_store\_inspections\_current\_critical\_vio...)
- Cereals (main.cereals)
- Earthquake Data January 2019 (main.earthquake\_dta2019)

Each dataset entry includes fields for ID, Created, Last Updated, Modified By, and # Dashboards.

- Create a connection - click on the NEW CONNECTION button on the left menu

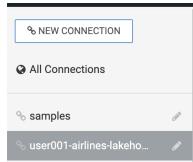
The screenshot shows the "Create New Data Connection" dialog. It has tabs for "Basic", "Advanced", "Parameters", and "Data". The "Basic" tab is selected. The form fields include:

- Connection type: CDW Impala
- Connection name: user001-airlines-lakehouse
- CDW Warehouse: airlines-impala-rw
- Hostname or IP address: coordinator.impala-1674425764-975.svc.cluster.local
- Port #: 28000
- Credentials:
  - Username: svr\_enr-zhxxww
  - Password: (empty field)

At the bottom, there are "TEST" and "CONNECT" buttons.

- Connection type - select CDW Impala
- Name - <user\_id>-airlines-lakehouse

- CDW Warehouse - select drop down and pick the **airlines-impala-vw**
- For this lab each participant will create their own connection in the same Data Viz instance, this would not be normal, you would only have to create a single connection to the same Virtual Warehouse
- Click on the Advanced tab in the middle of the screen, you can see that the details are already populated, there is nothing more to do. If certain settings were required you could change them here.
- Click CONNECT button, to create the Connection
- 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

- Create a new Dataset (aka. Metadata Layer or Data Model)

- Click on NEW DATASET button

New Dataset

Create a dataset from data on this connection. You need to create a dataset before you can create dashboards or apps.

Dataset title \*

Dataset Source

From Table

Select Database

user001\_airlines

Select Table

flights

**CANCEL** **CREATE**

- 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
- Select Table - flights
- Click CREATE

- 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

Dataset Detail

Dataset: airline\_logistics

Detail

Dataset: airline\_logistics [Edit](#)

Table: user001\_airlines.flights

Connection Type: Impala

Data Connection: user001-airlines-lakehouse [Edit](#)

Description: [Edit](#)

Join Elimination: Enabled [Edit](#)

Result Cache: From Connection [Edit](#)

Incremental Results: Disabled [Edit](#)

ID: 13

Created on: Jan 23, 2023 06:19 PM

Created by: user001

Last updated: Jan 23, 2023 06:19 PM

Last updated by: user001

- On the left menu click on Fields - let's quickly see what was created by adding the flights table to the Dataset. When this table was added it took the table's metadata and add the columns as Fields. (we'll come back to this later)

Dataset Detail

Related Dashboards

Fields

Dataset: airline\_logistics

Fields [Edit Fields](#) [Hide Comments](#)

Dimensions

- A: origin
- A: month
- A: dayofmonth
- A: dayofweek
- A: depature
- A: carrier
- A: carriername
- A: arrtime
- A: crsarrtime
- A: flights
- A: crsflights
- A: carriercode
- A: carriername
- A: arrdelay
- A: depdelay
- A: distance
- A: time
- A: tailnum
- A: canceled
- A: convdelay
- A: windspeed
- A: nonstop
- A: accdelayday
- A: lastARRIER

Measures

- x month
- x dayofmonth
- x dayofweek
- x depature
- x carrier
- x crsarrtime
- x flights
- x crsflights
- x carriercode
- x carriername
- x arrdelay
- x depdelay
- x distance
- x time
- x tailnum
- x canceled
- x convdelay
- x windspeed
- x nonstop
- x accdelayday
- x lastARRIER

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

Dataset Detail

Related Dashboards

Fields

Data Model

Dataset: airline\_logistics

Data Model [Edit Data Model](#)

flights

[SHOW DATA](#)

Apply Display Format

- Join planes - click the "+" button next to flights

Dataset: airline\_logistics

Data Model

flights

SHOW DATA

Apply Display Format

Table Browser

Choose the table you want to join. You will be able to select the columns that are joined in the next step.

Database Name: user001\_airlines

Table Name: planes

CANCEL SELECT

Database Name - <user\_id>\_airlines  
 Table Name - planes  
 Click SELECT button

flights

planes

Join Details

Inner Left Right Outer

Source Column Target Column

tailnum = tailnum

year = year

DELETE JOIN EDIT JOIN

Click the (JOIN) between flights and planes to see the join that was created  
 Click EDIT JOIN to modify the join as there is an extra join of year=year  
 Click on the - to the right of the year=year join and click the APPLY button

Edit Join

CLEAR FIELDS

user001\_airlines.flights user001\_airlines.planes

tailnum = tailnum

year = year

Join Expressions

if you enter multiple expressions they will automatically have an "AND" logic between them

Click to update in SQL expression editor

+ ADD JOIN PAIR + ADD JOIN EXPRESSION

CANCEL APPLY

Edit Join

CLEAR FIELDS

airline\_ontime\_iceberg.flights airline\_ontime\_iceberg.planes

tailnum = tailnum

Join Expressions

if you enter multiple expressions they will automatically have an "AND" logic between them

Click to update in SQL expression editor

+ ADD JOIN PAIR + ADD JOIN EXPRESSION

CANCEL APPLY

should be >>

- Join airlines - click the "+" button next to flights

Table Browser

Choose the table you want to join. You will be able to select the columns that are joined in the next step.

Database Name: user001\_airlines

Table Name: airlines

CANCEL SELECT

Database Name - <user\_id>\_airlines  
 Table Name - airlines  
 Click SELECT button

Edit Join

CLEAR FIELDS

user001\_airlines.flights user001\_airlines.airlines

select column... = select column...

Join Expressions

if you enter multiple expressions they will automatically have an "AND" logic between them

Click to update in SQL expression editor

+ ADD JOIN PAIR + ADD JOIN EXPRESSION

CANCEL APPLY

Edit Join

CLEAR FIELDS

user001\_airlines.flights user001\_airlines.airlines

uniquecarrier = code

Join Expressions

if you enter multiple expressions they will automatically have an "AND" logic between them

Click to update in SQL expression editor

+ ADD JOIN PAIR + ADD JOIN EXPRESSION

CANCEL APPLY

>>>

Click the  (JOIN) between flights and airlines to see the join that was created  
 Click EDIT JOIN to modify the join  
 Click on uniquecarrier under flights & code under airlines and click the APPLY button

- Join airports (origin airport) - click the “+” button next to flights

Table Browser x

Choose the table you want to join. You will be able to select the columns that are joined in the next step.

Database Name

Table Name

CANCEL SELECT

Database Name - <**user\_id**>\_airlines

Table Name - airports

Click SELECT button

Edit Join x

CLEAR FIELDS

<input type="text" value="user001_airlines.flights"/> <input type="text" value="origin"/> <a href="#">▶ sample data</a>	<input type="text" value="user001_airlines.airports"/> <input type="text" value="iata"/> <a href="#">▶ sample data</a>
---	--

= 

Join Expressions

If you enter multiple expressions they will automatically have an "AND" logic between them

+ ADD JOIN PAIR + ADD JOIN EXPRESSION

CANCEL APPLY

Click the  (JOIN) between flights and airports to see the join that was created  
 Click EDIT JOIN to modify the join  
 Click on origin under flights & iata under airports and click the APPLY button

- Join airports (destination airport) - click the “+” button next to flights

Table Browser x

Choose the table you want to join. You will be able to select the columns that are joined in the next step.

Database Name

Table Name

CANCEL SELECT

Database Name - <**user\_id**>\_airlines

Table Name - airports

Click SELECT button

Edit Join

[CLEAR FIELDS](#)

user001_airlines.flights	=	airports_1
dest	=	iata
<a href="#">sample data</a>		<a href="#">sample data</a>

Join Expressions  
If you enter multiple expressions they will automatically have an "AND" logic between them

[Click to update in SQL expression editor](#)

[+ ADD JOIN PAIR](#) [+ ADD JOIN EXPRESSION](#)

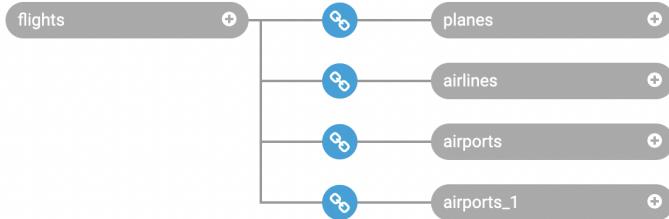
[CANCEL](#) [APPLY](#)

Click the (JOIN) between flights and airports\_1 to see the join that was created  
 Click EDIT JOIN to modify the join  
 Click on the dest under flights & iata under airports and click the APPLY button

- The final Data Model will look like the following...

Dataset: airline\_logistics

Data Model [UNDO](#) [SAVE](#)



[SHOW DATA](#)

Apply Display Format

- Click on the SHOW DATA button to preview the data that will be returned by the model created so far

Dataset: airline\_logistics

Data Model [UNDO](#) [SAVE](#) [NEW DASHBOARD](#)

[HIDE DATA](#)  Apply Display Format

month	dayofmonth	dayofweek	depTime	crsDepTime	arrTime	crsArrTime	uniqueCarrier	flightNum	tailNum	actualElapsedTime	crsElapsedTime	arrTime	arrDelay	depDelay	origin	dest	distance	taxisIn	taxisOut	cancelled	carArrTime
2	1	4	1037	1010	1556	1530	UA	1	N2110UA	559	560	530	26	27	ORD	HNL	4243	13	16	0	
2	2	5	1020	1010	1532	1530	UA	1	N2110UA	552	560	525	2	10	ORD	HNL	4243	12	15	0	
2	3	6	1021	1010	1513	1530	UA	1	N2110UA	532	560	515	-17	11	ORD	HNL	4243	2	15	0	
2	4	7	1016	1010	1512	1530	UA	1	N2110UA	536	560	521	-18	6	ORD	HNL	4243	3	12	0	

- Click on the SAVE button

## ■ Modify Fields - click on Fields in the left menu

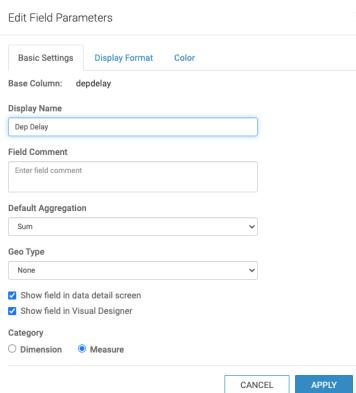
- Click on the EDIT FIELDS button; you will see this new tool bar



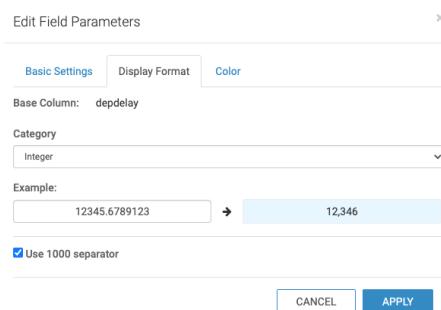
- Click on the TITLE CASE button to format the display names of the fields
- Under the Measures section, click on the Mes button next to month. This changes this to a Dimension.

- Each Field is assigned a Category (Measure or Dimension) - this is important because the type is used when creating visuals. CDV will use the data type to determine this Category. Make sure that Fields fall into the correct Category. This is important because this is used when creating visuals. The way to look at choosing the Category is to use this simple method - if you need to Aggregate (sum, average, etc.) then the Field should be a Measure. The quickest way to change the Category is to use the toggle
- Under the Measures section, click on the Mes button next to the following Fields:
  - Under Measures > flights
    - Dayofmonth
    - Dayofweek
    - Deptme
    - Crsdeptime
    - Arrtime
    - Crsarrrtime
    - Flightnum
    - Year

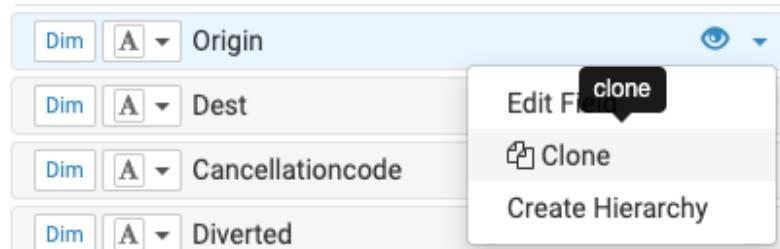
- All Fields under Measures > planes
- All Fields under Measures > airports
- All Fields under Measures > airports\_1
- **Note:** Normally, we would make sure that all Fields fall into the correct Category, instead let's move ahead
- Edit a Field
  - Click the pencil next to Depdelay under the Measures section to edit the field
  - Change the Display Name to Dep Delay & change the Default Aggregation to Average



- Click on the Display Format tab
  - Select Category of Integer and click on Use 1000 separator

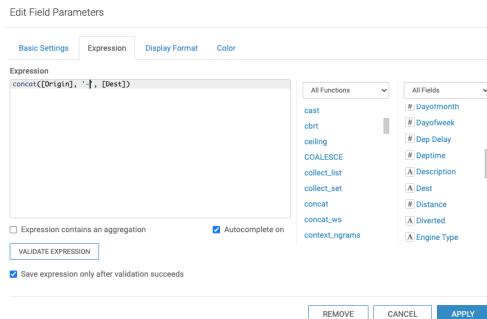


- Click APPLY button to save the changes
- Add a new Field
  - Clone the Origin Field - click the down arrow next to Origin, select Clone



- Click the pencil next to Clone of Origin to edit the field
- Change Display Name to Route
- Click on the Expression tab - enter the following into the Expression window

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



- Click APPLY - since the “Save expression only after validation succeeds” is checked the Field will first be validated to ensure there are no errors then if it is valid will be saved

- Click the SAVE button

24. Create Dashboard - in the top right corner click on the NEW DASHBOARD button

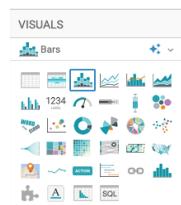
New Dashboard

- Quick Overview the the interface
  - 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.

<u>Visual Types</u>	<u>Shelves</u>	<u>DATA</u>
---------------------	----------------	-------------

The screenshot shows the Cloudera Data Visualization (CDV) interface. On the left, the 'VISUALS' shelf lists various visualization types: Table, Bar, Line, Map, etc. In the center, there are four shelves: 'Dimensions' (with fields for Month, Dayofmonth, Dayofweek, DepTime, CrsDepTime, ArrTime, CrsArrTime), 'Measures' (with avg(Dep Delay)), 'Toolips' (empty), and 'Filters' (empty). On the right, the 'Dashboard Designer' pane shows the 'DATA' section for the 'airline\_logistics' dataset, listing 41 dimensions and 15 measures. Below it, the 'airlines' and 'planes' datasets are also listed.

- 1st Visual - Top 25 Routes by Avg Departure Delay; are there certain Routes with excessive Delays
  - 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

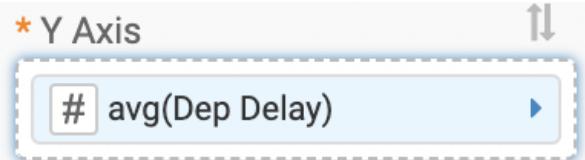


- Edit the Shelves (two (2) ways to add items to a Shelf, you will use both here; for the remainder of this lab you can pick & choose which you use)

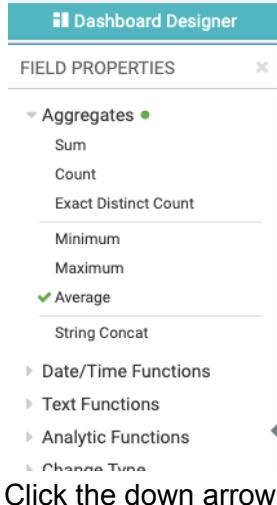
The screenshot shows the 'Dashboard Designer' interface with a 'Table' visual added to the workspace. The table displays data for routes, including Month, Dayofmonth, Dayofweek, DepTime, CrsDepTime, ArrTime, and CrsArrTime. The data shows various route numbers and their corresponding departure and arrival times. To the right, the 'DATA' pane for the 'airline\_logistics' dataset is visible, showing the same dimensions and measures as the previous screenshot.

The screenshot shows the Cloudera Data Visualization interface. The left panel, titled "VISUALS", contains a toolbar with various chart icons (e.g., Bars, Line, Heatmap) and a shelf area for X Axis, Y Axis, Colors, Tooltips, Drill, Labels, and Filters. A blue button at the bottom left says "REFRESH VISUAL". The right panel, titled "DATA", shows a connection named "airline\_logistics" with "Sample Mode: OFF". It includes a search bar and two main sections: "Dimensions" (41 items) and "Measures" (15 items). The "Dimensions" section lists fields like Month, Dayofmonth, Dayofweek, DepTime, CrsDepTime, ArrTime, CrsArrTime, UniqueCarrier, FlightNum, TailNum, Origin, Route, Dest, CancellationCode, Diverted, and Year. The "Measures" section lists fields like Airtime, ArrDelay, DepDelay, Distance, TaxiIn, and TaxiOut.

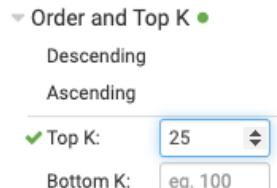
- Option 1: Add Route to the X Axis
  - Click in the X Axis Shelf to select it
  - In the DATA menu find Route under Dimensions > flights
- Option 2: Add Dep Delay to the Y Axis
  - In the DATA menu find Dep Delay under Measures > flights, Drag & Drop this Field in the Y Axis Shelf
- So the two (2) options for adding items to Shelves is 1) select Shelf and click to add Field; or 2) Drag & Drop Field to Shelf
- Modify Properties for Dep Delay - click on the > in the Y Axis Shelf to the right of avg(Dep Delay) to access the Properties for this Shelf



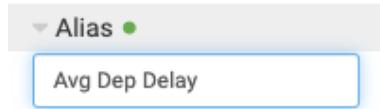
- Click on the down arrow to the left of Aggregates - in CDV you have control over any behavior, in the Dataset we set the default aggregation for this Field to be Average, however, on any visual you can override this by changing it here. For now, leave this alone.



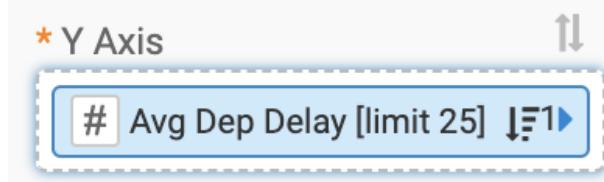
- Click the down arrow to the left of Order and Top K - to show the 25 Routes with the highest Average Dep Delay enter 25 into the box for "Top K"



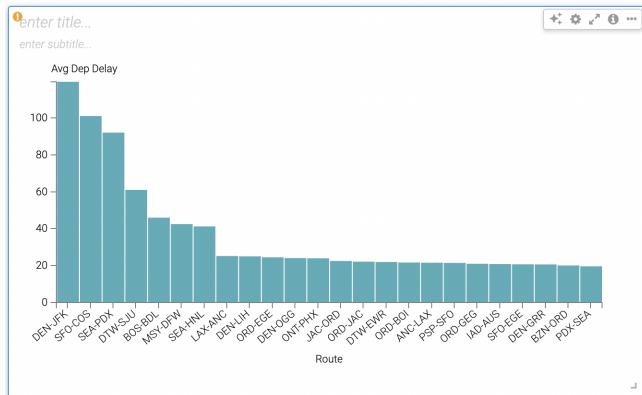
- Click the down arrow to the left of Alias - to change the display name for this Field. In the box enter Avg Dep Delay



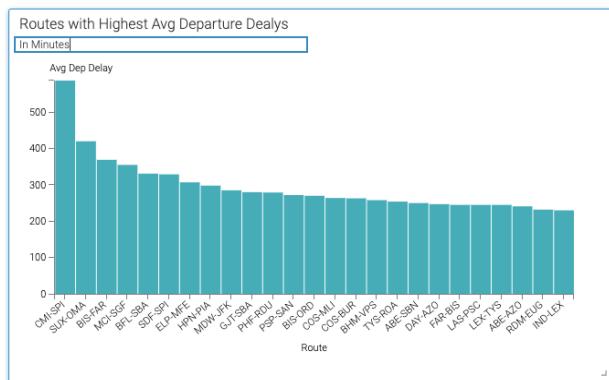
- The finished Y Axis Shelf should look like this



- Click the button to update the Visual



- Change the Title & Subtitle for this Visual
  - Click in the enter Title above the chart and enter - Routes with Highest Avg Departure Delays, hit enter
  - Click in the Subtitle and enter - In Minutes, hit enter

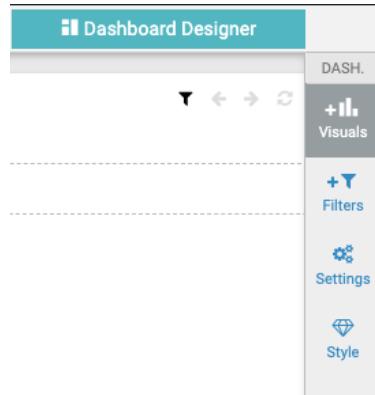


- Add Title & Subtitle for the Dashboard
  - Click in the enter Title right below the banner and enter - <user\_id> Logistics Dashboard
  - Click in the Subtitle and enter - CDW Workshop

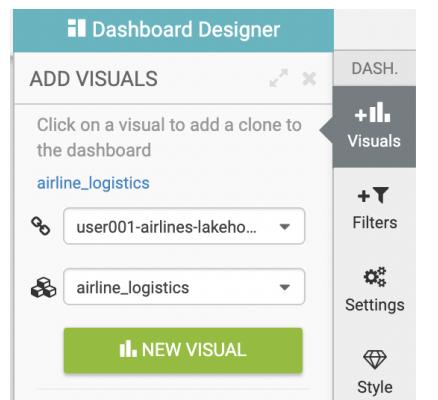
The dashboard interface includes a top bar with 'LAYOUT', 'SAVE', and 'PRIVATE' buttons. The main area displays the title 'user001 Logistics Dashboard' and subtitle 'CDW Workshop'. Below the title is a note: 'Add dashboard filters by selecting dataset fields from the sidebar.' The visual itself is titled 'Routes with Highest Avg Departure Delays' and shows the detailed list of routes and their average departure delays.

- Click on the SAVE button to save this Dashboard, this Dashboard is now named <user\_id> Logistics Dashboard
- 2nd Visual - Relationship between flight Cancellation reason and Carrier; are there Carriers or Reasons that need to be addressed?

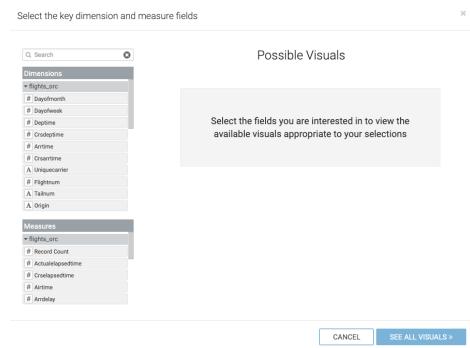
- Click on +Visuals - on the far right of the screen you will see the DASH. menu. This menu allows you to add new visuals and change settings for the Dashboard or Visual such as formatting, style, and anything related to the presentation of the data.



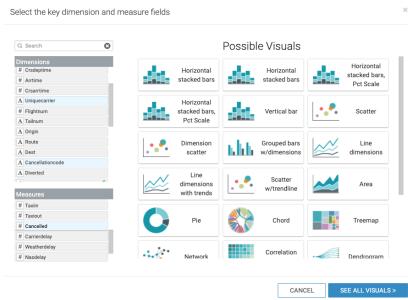
- Under ADD VISUALS - leave the (Connection) as <user\_id>-airlines-lakehouse and (Dataset) as airlines\_logistics. However, a Dashboard can have any number of connections and Datasets for various visuals on a Dashboard.



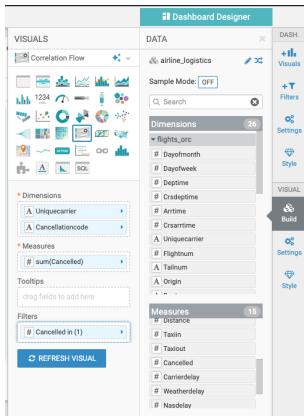
- This will add a default visual to the canvas
- Click on the button, next to the Table . Instead of manually building this visual, we will use the Visual helper



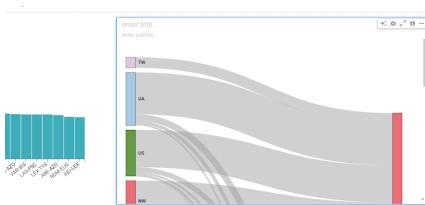
- Select Uniquecarrier and Cancellationcode under Dimensions; select Cancelled under Measures. As you select items from the Fields you will see the Possible Visuals change - helping you quickly select the visual based on what you have selected.



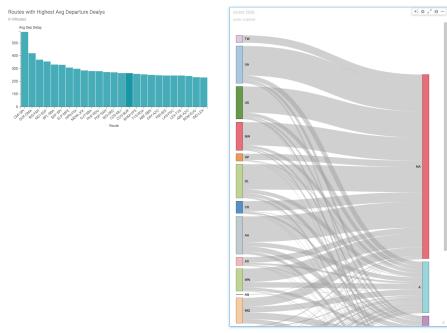
- Click on SEE ALL VISUALS> button - to preview the Possible Visuals with actual data plotted
- Click on the Correlation flow visual - Scroll thru until you see the Correlation flow visual tile
- Drag & Drop Cancelled from DATA menu under Measures > flights to the Filter Shelf - There are several ways to apply filters to visuals within the Dashboard, this is one
  - When the dialog box comes up



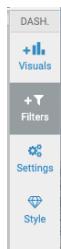
- This will filter this chart to only return flights that have been cancelled, and will not return any other data
- Click the REFRESH VISUAL button



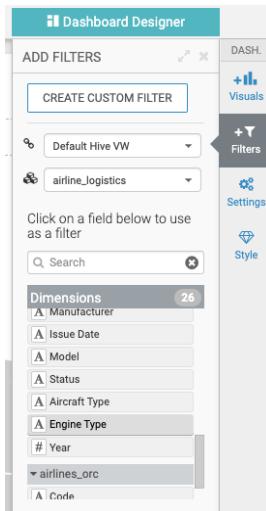
- **NOTICE:** Since there is a security policy still in effect, you will only see "UA" showing up in this visual. To see all of the Carriers, you could disable the security Policy and Refresh the Visual.
- Click ↗ in the bottom right corner of the Correlation flow visual and drag it down to resize this chart (make it a bit larger to your liking)



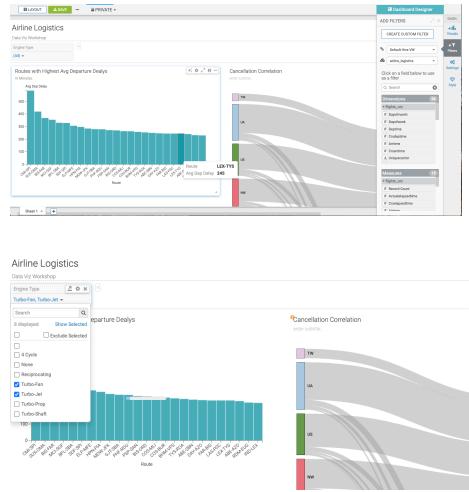
- Click the enter Title and enter Cancellation Correlation, and hit enter
- Click the SAVE button to save the Dashboard
- Add prompts - allow dashboard to be sliced & diced; this is another way to filter on multiple charts at the same time in your Dashboard
  - On the DASH. menu (far right) click on +Filters button



- Click on Engine Type under Dimensions > planes, to add this Field as a filter; you continue to select other Fields that you want to create Filters for and these will also be added



- Click the drop down arrow next to Engine Type and select any value - the filter(s) are added to the Filter shelf for the Dashboard which is between the Dashboard Title and the Visuals you have been creating. When you are selecting values from this Filter, the visuals will change to reflect information for just flights where this Engine Type was in the plane for the flight



- Click the SAVE button to save the Dashboard
  - [optional] View Dashboard from Visuals page
    - Click Visuals in the top banner
    - Dashboards can be shared with other users - use Workspaces to organize and secure content that is created
    - Create Applications - combine multiple Dashboards to produce an application that allows users to make better decisions
  -
- 

## Optional Lab 5 - Data Security & 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.

Robust Data Catalog (Governance) capability that:

- Enables you to understand, manage, secure, and govern data assets across
- Provides a 360 degree view of Assets - Lineage, Metadata, Security Policy applied to the asset
  - There are many assets - everything created in CDP is captured as an Asset
  - For this Workshop we have been working with Assets, like - databases, views, tables, etc.

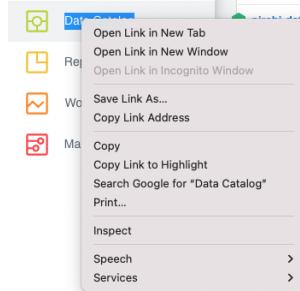
Powerful Security features like:

- Rule-based masking columns based on a user's role
- Group association or rule-based row filters
- Attribute-Based Access Control a.k.a. Tag-based security policies

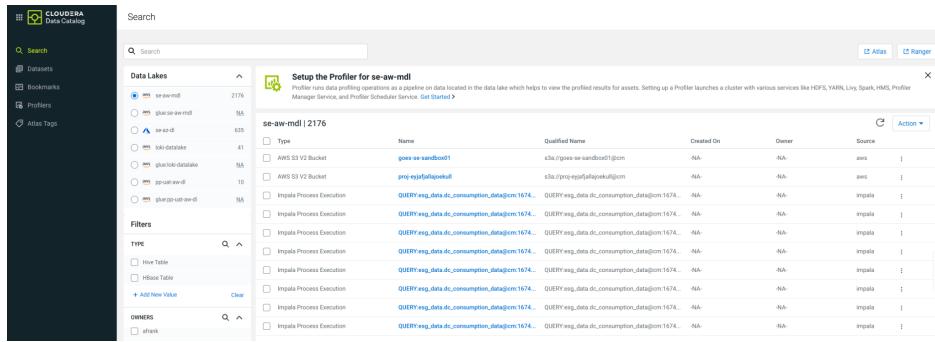
25. Data Catalog - Governance (Lineage, Metadata, etc.)
  - On the left navigation menu click the next to Data Warehouse



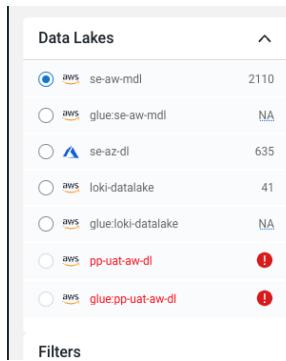
- Right click on the Data Catalog and select “Open Link in New Tab” option



- Open browser tab that just opened, should have Data Catalog in the tab title



- To the left, under Data Lakes, ensure the Data Lake provided in Lab Setup radio button is selected



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

## Filters

TYPE	
<input checked="" type="checkbox"/> Hive Table	<input type="checkbox"/>
<input type="checkbox"/> HBase Table	
<a href="#">+ Add New Value</a>	
<a href="#">Clear</a>	

- 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 two panels. The left panel displays a list of existing databases: reddit\_data, prescribing\_p\_e, hol\_rj, default, and information\_schema. Below this is a search bar and a 'Clear' button. The right panel shows a list of databases with a search bar at the top. A dropdown menu is open, showing 'user' and 'user001\_airlines\_raw'. The 'user001\_airlines' option is highlighted. At the bottom of the right panel are 'Cancel' and 'Add' buttons.

- 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 flights table.

The screenshot shows the Cloudera Manager interface for the 'se-aw-mdl' data lake. On the left, there's a sidebar with filters for Type (Hive Table selected), Owners (jingle), and Database (user001\_airlines selected). The main area shows a table of assets. The 'flights' table is highlighted with an orange border. The table has columns: Type, Name, Qualified Name, Created On, Owner, and Source. The 'flights' row shows the following details:

Type	Name	Qualified Name	Created On	Owner	Source
Hive Table	flights	user001_airlines.flights@com	Thu Jan 19 2023	jingle	hive

- 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 - hoover over the flights click on the "i" icon that appears to see more detail on this table

Asset Details

**flights**

**Properties**

Type: HIVE TABLE  
# of Columns: 29  
Data Lake: se-airline  
Description: 0  
Owner: jingalls  
Created On: Thu Jan 19 2023 13:49:17 GMT-0600 (Central Standard Time)  
Last Accessed: Thu Jan 19 2023 13:49:17 GMT-0600 (Central Standard Time)  
Table Type: EXTERNAL\_TABLE  
Database: user001\_airlines  
DB Creation: em  
Parent: user001\_airlines

**Classifications**

+ Add Classification

**Terms**

+ Add Terms

**Overview**   Schema   Metadata Audits   Policy   Access Audits

**Lineage**   Filter By: Depth: 3 x   Process Node: Hide +

The lineage diagram illustrates the data flow. It starts with three blue boxes labeled 'flights'. One arrow points from one flight box to a purple box labeled 'flights\_csv'. Another arrow points from the 'flights\_csv' box to an orange box labeled 'flights'. A third arrow points from the 'flights' box to a green box labeled 'traffic\_cancel\_airlines'. A legend at the bottom indicates: Lineage (green arrow), Impact (red arrow), Replication (blue arrow), and Current Entity (orange box).

- The lineage shows
  - [blue box] flights data file residing in an s3 folder
  - [purple 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)
  - Traffic\_cancel\_airlines is a Materialized View that uses data from the flights Iceberg table.



- Click on the Schema Tab to see Metadata on this table. The Metadata includes basic information from the table such as: column names and data types for the columns. You can also run Profilers that come as part of CDP:
  - Hive Table Profiler - allows you to gather additional information from the table including Min/Max values in the column, # records with Null Values in a column, etc.
  - Sensitivity Profiler - identifies columns that may contain sensitive information such as PII data, SSN, credit card numbers, ID numbers, etc. These items will be "Tagged" in the Classification column of the Schema page.

The screenshot shows the 'Schema' tab of the Cloudera Data Platform interface. It displays a table with columns for Name, Type, Unique Values\*, Null Values, Max, Min, Mean, Comment, Classifications, and Terms. The table lists several columns from a table named 'flights': 'actualElapsedTime', 'airline', 'arrDelay', 'arrTime', 'cancellationCode', 'cancelled', 'carrierDelay', and 'crsArrTime'.

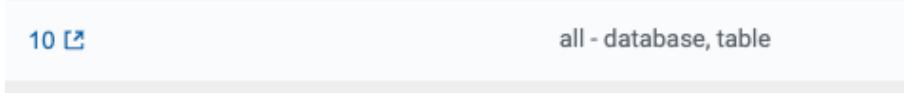
Chart Type	Name	Type	Unique Values*	Null Values	Max	Min	Mean	Comment	Classifications	Terms
	actualElapsedTime									
	airline									
	arrDelay									
	arrTime									
	cancellationCode									
	cancelled									
	carrierDelay									
	crsArrTime									

- Click on the Policy tab to see what security policies have been applied on this table. There are 2 policies that have been defined to allow some users & groups access via policy “all - database, table, column” and another policy “all - database, table” access.
  - The Access Audits tab allows Administrators to be able to see who, when, where, why either accessed this table or was denied access to this table. Feel free to switch to this tab to take a look and switch back to the Policy tab.

The screenshot shows the 'Policy' tab of the Cloudera Data Platform interface. It displays two resource-based policies:

Policy ID	Policy Name	Status	Audit Logging	Group	Users
9	all - database, table, column	ENABLED	ENABLED	..._ranger_admins_7925da73	hive, beacon, dpprofiler, hue, admin, impala, r...
10	all - database, table	ENABLED	ENABLED	..._ranger_admins_7925da73	hive, beacon, dpprofiler, hue, admin, impala, r...

- Click on the next to the “all - database, table” - to modify this policy or create a new policy



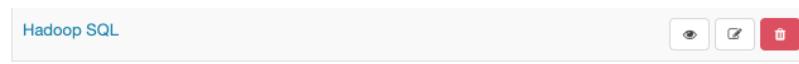
## 26. Security (Ranger) - modify and create security policies for the various CDP Data Services.

- View the Policy details and click the CANCEL button at the bottom.
- For this portion of the Lab let's restrict your access to a subset of data available in the “flights” table.

The screenshot shows the 'Service Manager' section of the Ranger interface. It lists security policies for various services:
 

- HDFS: om\_hdfs
- HBASE: om\_base, cod\_cod-test\_base, cod\_cod-workshop\_base, cod\_pavne-test-db\_base, cod\_nfc\_base
- HADOOP SQL: Hadoop SQL
- YARN: alink\_net\_sub\_yarn, amain\_hdfs\_yarn
- KNOX: om\_knox
- SOLR: om\_solr

- Click on the Hadoop SQL link in the upper left corner - to view the security policies in place for CDW. For this Workshop we 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 a table titled "List of Policies : Hadoop SQL". The columns are: Policy ID, Policy Name, Policy Labels, Status, Audit Logging, Roles, Groups, Users, and Action. The rows list various policies such as "all - global", "all - database, table, column", "all - database, table", "all - storage-type, storage-ut", "all - database", "all - hiveservice", "all - database, utf", and "all - set". Most policies have "Enabled" status and "Audit Logging" checked. Roles like "ranger\_jarvis", "ranger\_jarvis\_public", and "ranger\_jarvis\_hive" are assigned. Groups like "jimalls" and "hive" are listed. The "Action" column contains icons for edit, delete, and preview.

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

The screenshot shows a table titled "List of Policies : Hadoop SQL". The columns are: Policy ID, Policy Name, Policy Labels, Status, Audit Logging, Roles, Groups, Users, and Action. A message "No Policies found" is displayed at the bottom of the table area.

- Fill out the form as follows. For this policy let's say you are an Gate Agent for United Airlines and should not be able to see data for any other Airline. To setup the policy we need to apply a filter that is applied to any query that is run for your <user\_id> so you only see rows for flights run by United Airlines.
  - Policy Name: <user\_id> Workshop Row Level Filter
  - Hive Database: <user\_id>\_airlines (start typing, once you see this database in the list, select it)
  - Hive Table: flights (start typing, once you see this table in the list, select it)
  - Row Level Filtering Conditions
    - Select User - <user\_id> (start typing, once you see this user in the list, select it)
    - Row Level Filter - **uniquecarrier="UA"**
    - For this you could have any number of filter conditions (for this Lab we will only create 1) with many different filter configurations
  - Click **Add** button to accept this Policy

The screenshot shows the "Create Policy" page. The "Policy Type" is set to "Row Level Filter". The "Policy Name" is "workshop Row Level Filter", "Status" is "Enabled", and "Audit Logging" is checked. The "Policy Label" is "Policy Label". The "Hive Database" is "x.user001\_airlines" and the "Hive Table" is "x.flights". The "Description" field is empty. At the bottom, there is a "Row Filter Conditions" section with a table for selecting roles, groups, users, access types, and row level filters. A condition "uniquecarrier='UA'" is selected under "Row Level Filter".

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

The screenshot shows a table titled "List of Policies : Hadoop SQL". The columns are: Policy ID, Policy Name, Policy Labels, Status, Audit Logging, Roles, Groups, Users, and Action. A single row is shown for the policy "workshop Row Level Filter" with Policy ID 395, Status "Enabled", Audit Logging "Enabled", and User "jimalls".

- Test the policy is working - Open HUE for the CDW **Impala** Virtual Warehouse - **airlines-impala-vw** and execute the following query

```
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 HUE interface. In the top navigation bar, there is a dropdown menu with 'Impala' selected, followed by 'Add a name...', 'Add a description...', and a search bar. Below the navigation bar, the query editor contains the following SQL code:

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

Below the query editor, there is a dropdown menu set to 'user001'. The results pane shows the output of the query:

uniquecarrier	count(*)
UA	17642768

At the bottom of the results pane, it says 'Results(1)'. Above the results, there are four status messages from different queries:

- Query 93411c4cb83b:194.8de413ba00000000: 9% Complete (8 out of 55)
- Query 93411c4cb83b:194.8de413ba00000000: 100% Complete (55 out of 55)
- Query 93411c4cb83b:194.8de413ba00000000: 100% Complete (55 out of 55)
- Query 93411c4cb83b:194.8de413ba00000000: 100% Complete (55 out of 55)

## Optional Lab 6 - Iceberg Table Rollback

This lab will show you how to do a Rollback to a specific Snapshot. This is a key feature of Iceberg tables. There are other Maintenance Tasks that you can perform on Iceberg tables.

27. Iceberg Rollback [Table Maintenance] - sometimes data can be loaded incorrectly, due to many common issues - missing fields, only part of the data was 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

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

28. Iceberg Expire Snapshots [Table Maintenance] - 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

```
-- 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}`');
```

