CDW Workshop Student Guide

Revision History

Revision	Date	Author	Comments
1		Vinay Rayker	Initial Version
1.1		Puneet Joshi	Updated as per CDP Version

Introduction

In this workshop, we will introduce you to the key capabilities of Cloudera Data Warehouse experience. Cloudera Data Warehouse is an auto-scaling, highly concurrent and cost effective analytics service that ingests high scale data anywhere, from structured, unstructured and edge sources. It supports hybrid and multi-cloud infrastructure models by seamlessly moving workloads between on-premises and any cloud for reports, dashboards, ad-hoc and advanced analytics, including AI, with consistent security and governance. Cloudera Data Warehouse offers zero query wait times, reduced IT costs and agile delivery.

We are going to set up Cloudera Data Warehouse on CDP Public Cloud with AWS as backend infrastructure for storage and compute. At a high level, we are going to cover below items during our workshop:

- Controlling Cost, Matching DW Supply to Demand via autosuspend, autostart
- Protecting Workloads via isolation
- Scaling Up Usage via concurrency optimizations
- Keeping Up with the Needs of Business via fast provisioning
- Meeting SLAs via automated caching
- Secure & Governance via SDX

Labs summary

1. Set-up Virtual Warehouse
2. Data Exploration
3. Auto-scaling, Auto-suspend & Caching
4. Security & Governance with Ranger & Audit
5. Data Visualization

Lab 0 - Initial setup

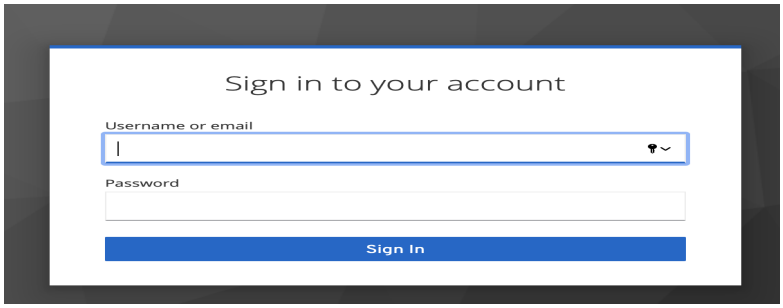
Pre-requisites

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

Getting Connected

URL:	Will be shared by the instructor
User:	Will be shared by the instructor
Password:	Will be shared by the instructor
Project:	https://github.com/vrayker/cdw_workshop

Open the URL provided by the instructor on your browser. You will get the following . Please inform your instructor and share your machine's IP in case you are NOT able to get the following page.



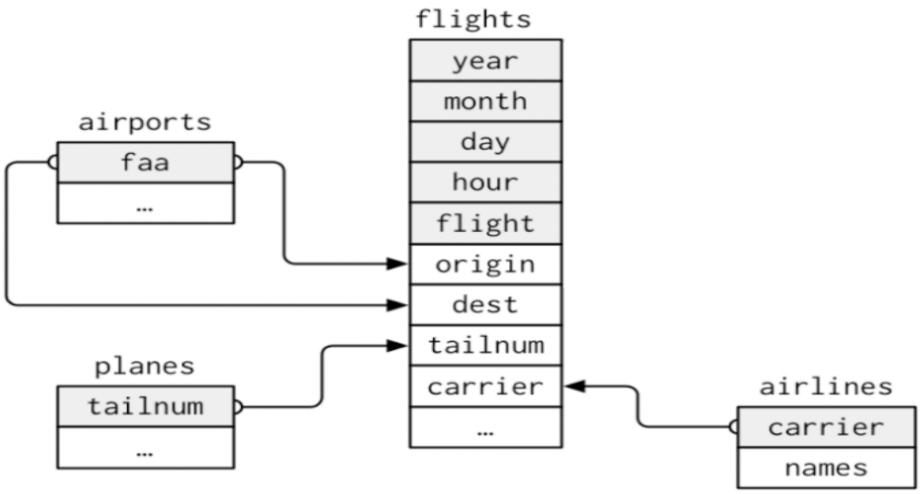
Enter the username and password shared by your instructor. You should be able to get the following home page of CDP Public Cloud.

Dataset for the workshop

In this exercise, we will work with an existing data set containing airline data. The data consists of flight arrival and departure details for all commercial flights within the USA, from October 1987 to April 2008. This is a large dataset: there are nearly 120 million records in total, and it takes up 1.6 GB of space compressed, and 12 GB uncompressed. The objective is to show how you can easily work with your data in a secure, scalable and cost-effective data warehouse: Cloudera Data Warehouse

The goal is to build a DW application that mirrors the applications that our customers run on our platform (or that we want them to migrate to our platform). We will navigate through the data and use some DW concepts like SCD (Slow Change Dimension), Facts, Materialized Views, etc.

Data Model



*faa ⇒ iata
*carrier ⇒ code

Tables

The database consists of four tables:

Dimensions:

- **Airports** - Table with airport information ("carrier" column name is changed to "code" in this exercise).
- **Planes** - Table with planes details.
- **Airlines** - Table with airline information ("faa" column name is changed to "iata" in this exercise).

Fact:

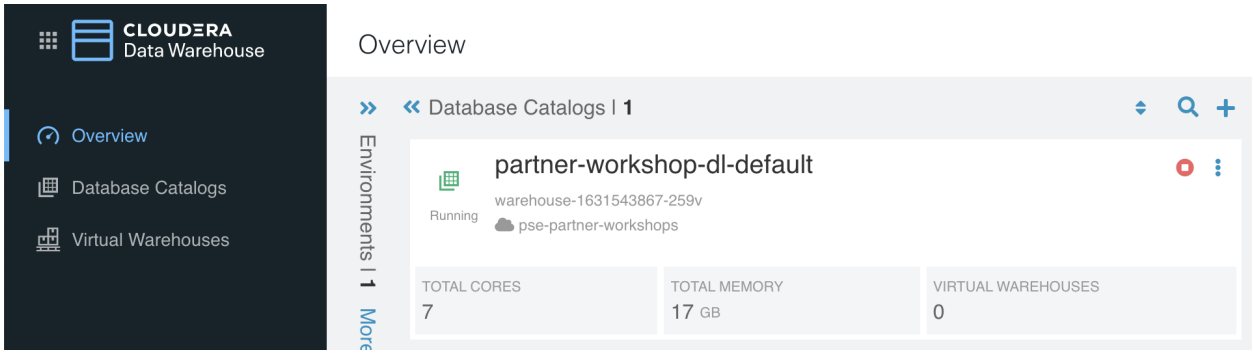
- **Flights** - Table containing flights between a period and with details of cancelled flights.

Initial Setup

To make this workshop efficient, we have already created necessary prerequisites for users to start working on features of Cloudera Datawarehouse. Here are steps that are already done:

1. Workshop data is already loaded in S3 bucket
2. CDP environment is already created
3. The necessary Database Catalog (kind of Hive metastore) has also been created.

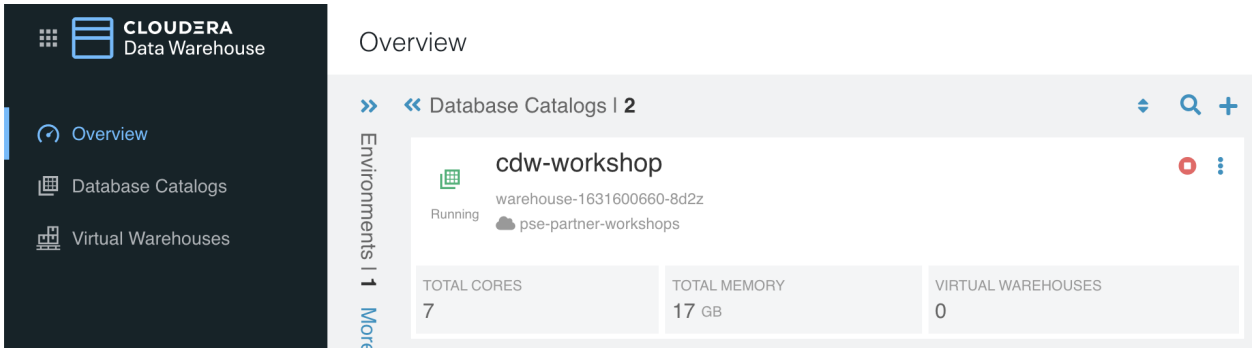
******** Screenshots of environment & Data Catalogue *********

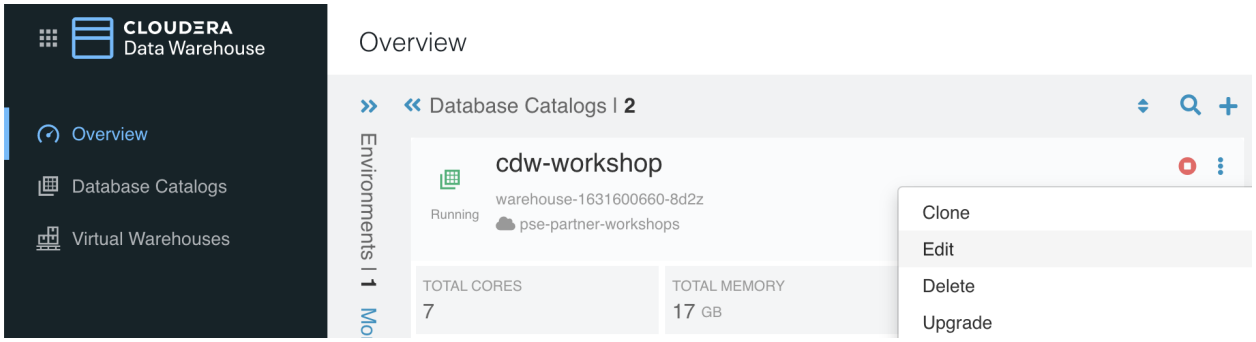


Lab 1 – Setup Virtual Warehouse

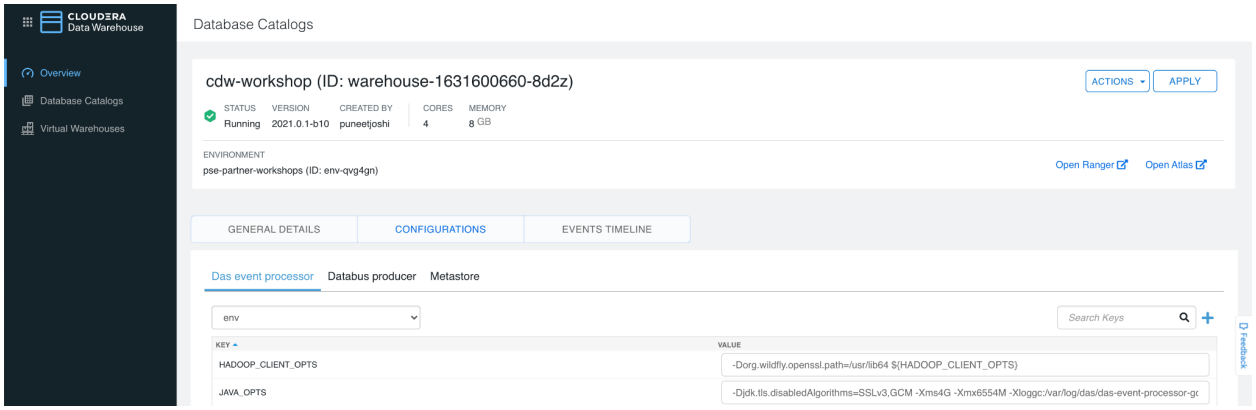
In this lab, you will create a Virtual Warehouse in CDP with an existing Database Catalog.

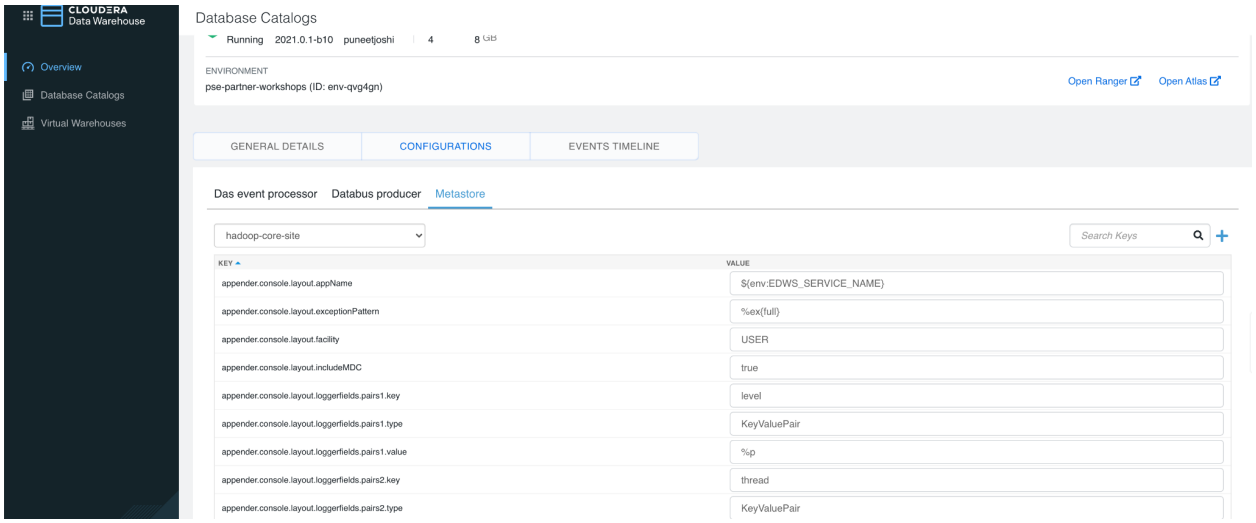
STEP 1 : Once you login into CDP, goto Datawarehouse tab. You will see an existing Database catalog already running. Click on the Database Catalog and edit to browse through all key attributes of it.



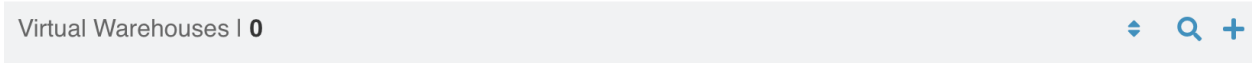


STEP 2 : Spend some time to go through the properties. Click on “Das event processor”, “Databus producer” and “metastore” and familiarize with its properties. It is almost similar to what you typically configure for Hive metastore.

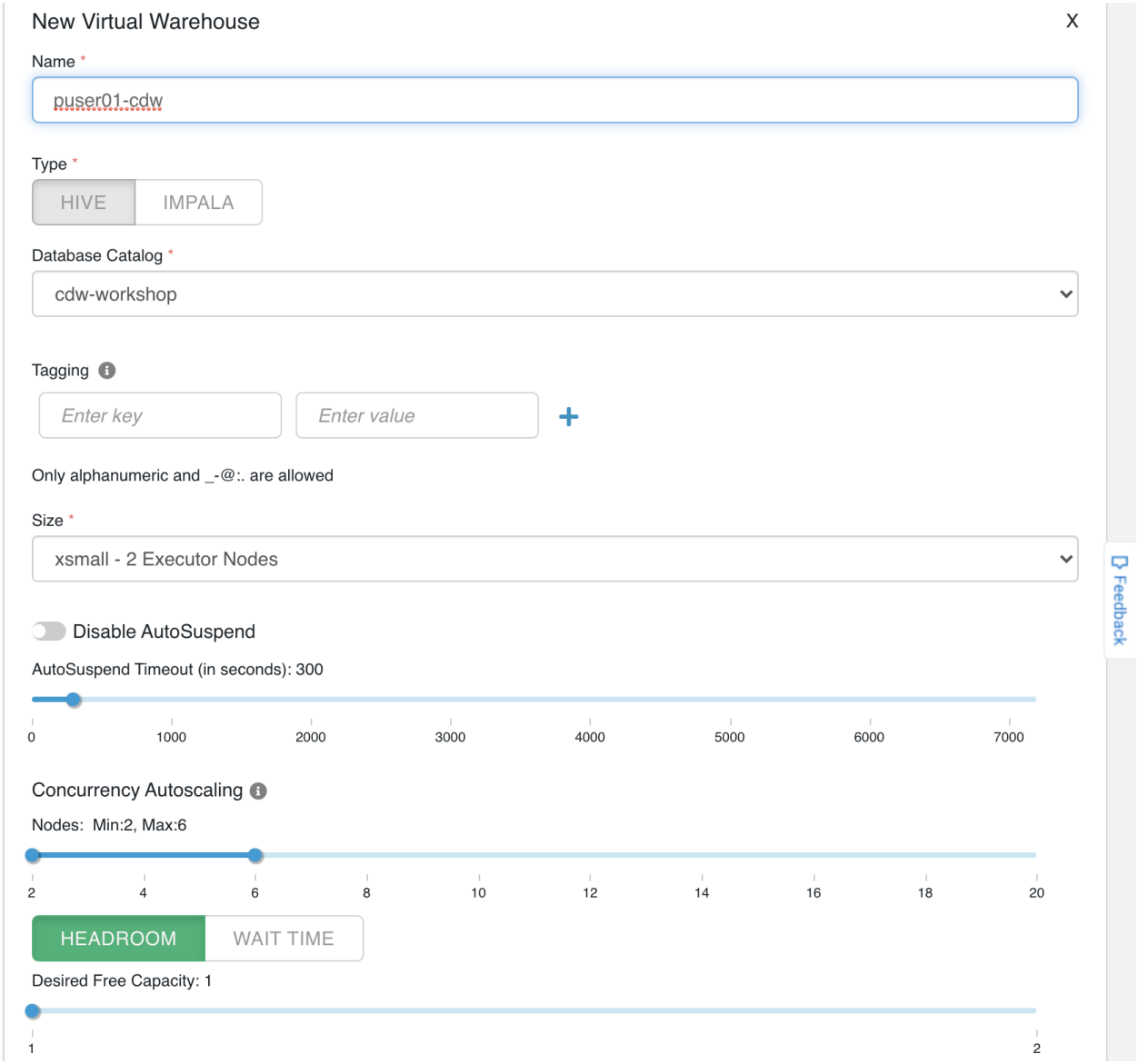




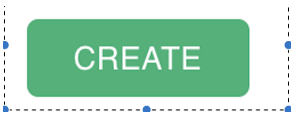
STEP 3 : In this step, we will create a new Virtual Warehouse. Click on + on Virtual Warehouse side and it will open up a new area for you to fill-in necessary values for your new Virtual Warehouse. Please key-in below values:



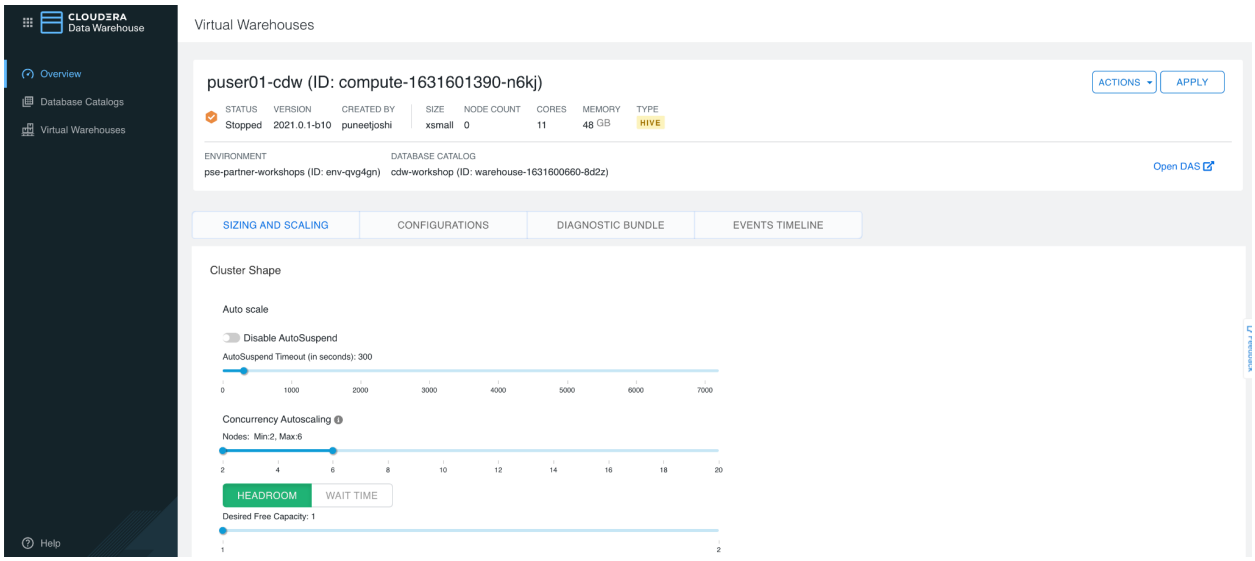
- Name: username-cdw (prefix with your username)
- Type: Hive
- Database Catalog: **cdw-workshop**
- Virtual Warehouse size: **xsmall - 2 Executor Nodes**
- AutoSuspend Timeout (in seconds): 300
- Nodes: Min:2, Max:6 (move the slide bar to adjust maximum to 6)
- Rest all properties can be left with default values



Hit on Create to create your Virtual Warehouse Compute. It usually takes less than 5 minutes for your Virtual Warehouse to be available for consumption.



STEP 4 : Spend some time going through the properties of the newly created Virtual Warehouse. Click on edit button, which will take you to view all properties linked to your virtual Warehouse.

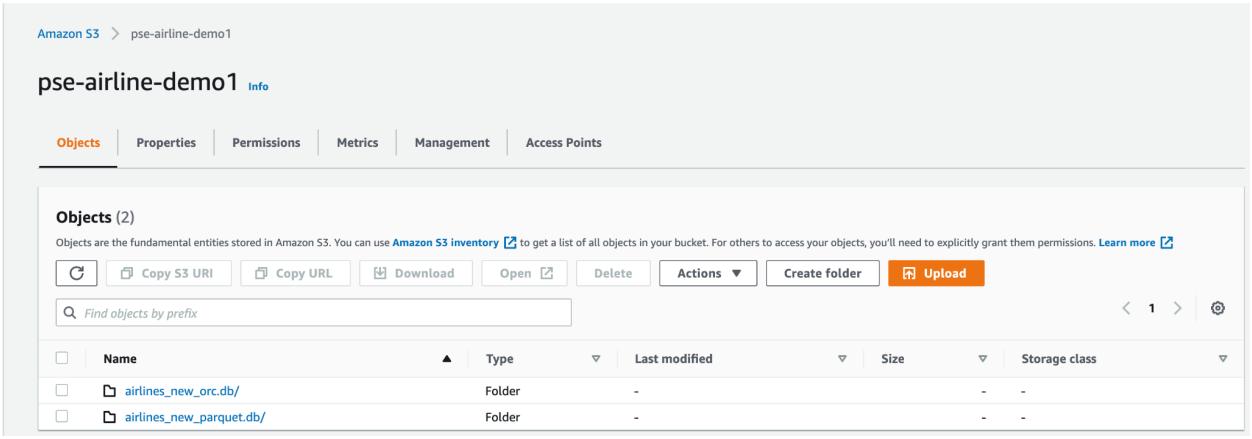

CLOUDERA
Data Warehouse
Overview
Database Catalogs
Virtual Warehouses
Help
Virtual Warehouses
puser01-cdw (ID: compute-1631601390-n6kj)
ACTIONS
APPLY
STATUS
Stopped
VERSION
2021.0.1-b10
CREATED BY
puneetjoshi
SIZE
xsmall
NODE COUNT
0
CORES
11
MEMORY
48 GB
TYPE
HIVE
ENVIRONMENT
pse-partner-workshops (ID: env-qvg4gn)
DATABASE CATALOG
cdw-workshop (ID: warehouse-1631600660-8d2z)
Open DAS
SIZING AND SCALING
CONFIGURATIONS
DIAGNOSTIC BUNDLE
EVENTS TIMELINE
Cluster Shape
Auto scale
Disable AutoSuspend
AutoSuspend Timeout (in seconds): 300
0
1000
2000
3000
4000
5000
6000
7000
Concurrency Autoscaling
Nodes: Min:2, Max:6
2
4
6
8
10
12
14
16
18
20
HEADROOM
WAIT TIME
Desired Free Capacity: 1
1
2

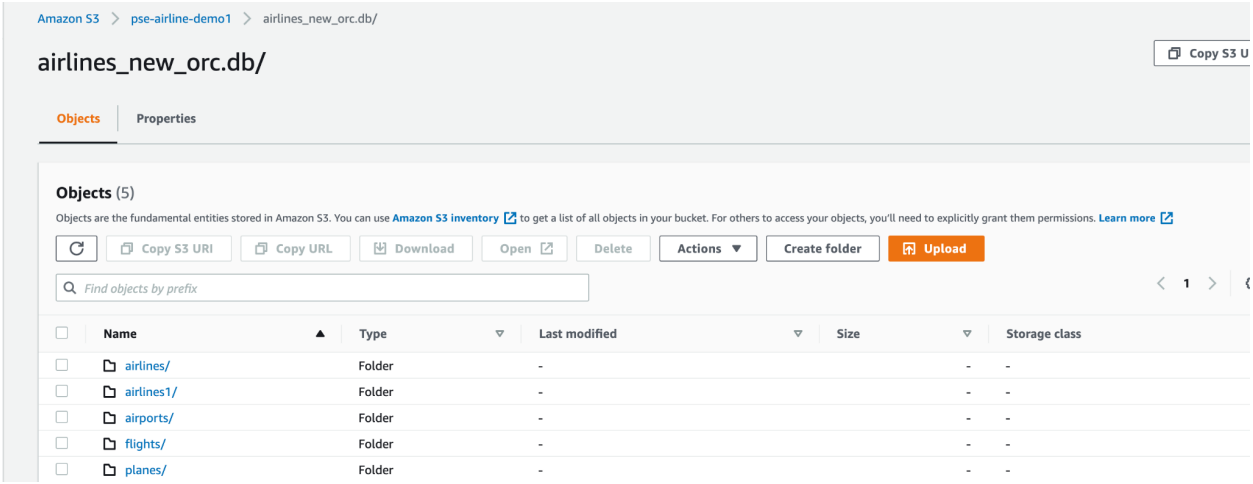
Lab 2 - Data Exploration

In this lab, you will explore some dataset using a different editor available within CDP

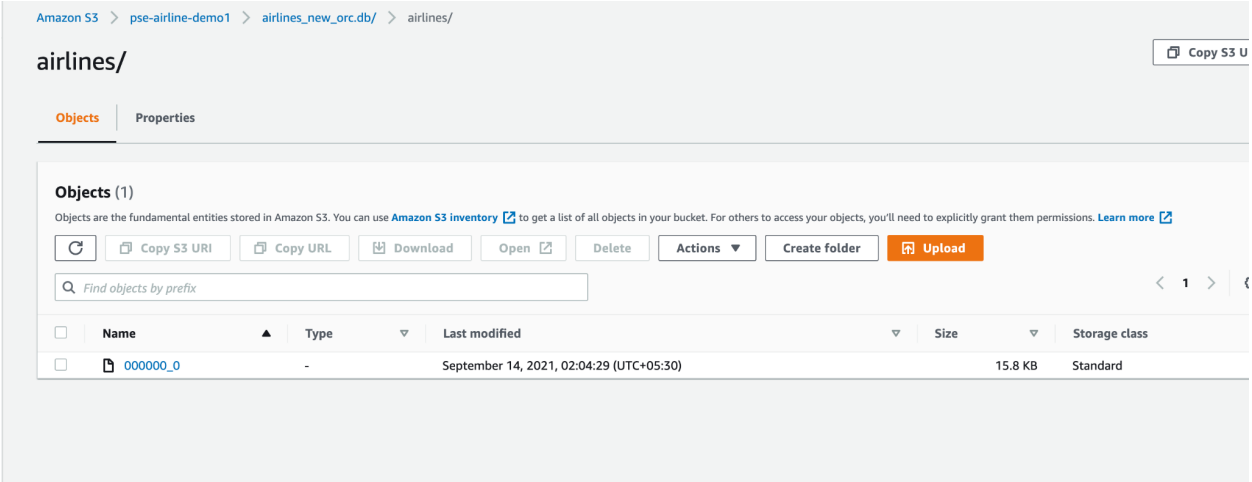
STEP 1 : Information on S3 files

You will be using a dataset that's already loaded into S3 buckets as ORC files. Here is how these data is placed in S3 bucket named **pse-airline-demo1**

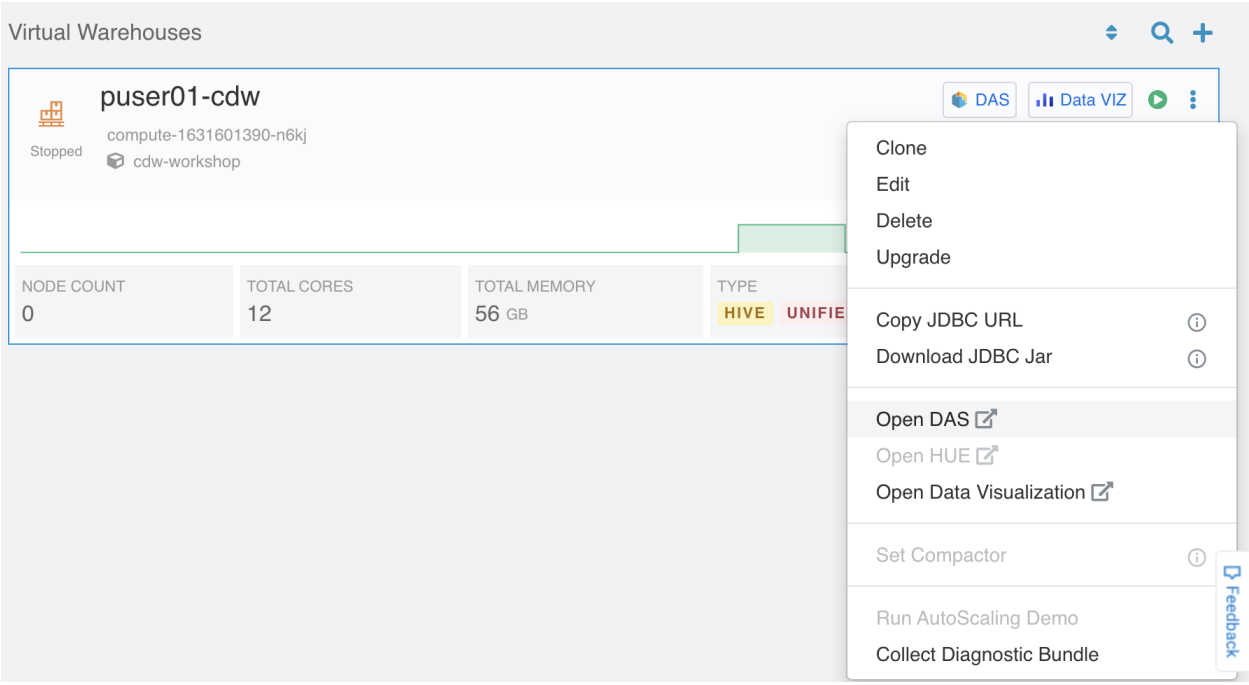




Since the files are stored in ORC format for faster querying & access, you won't be able to read these files as standard ASCII

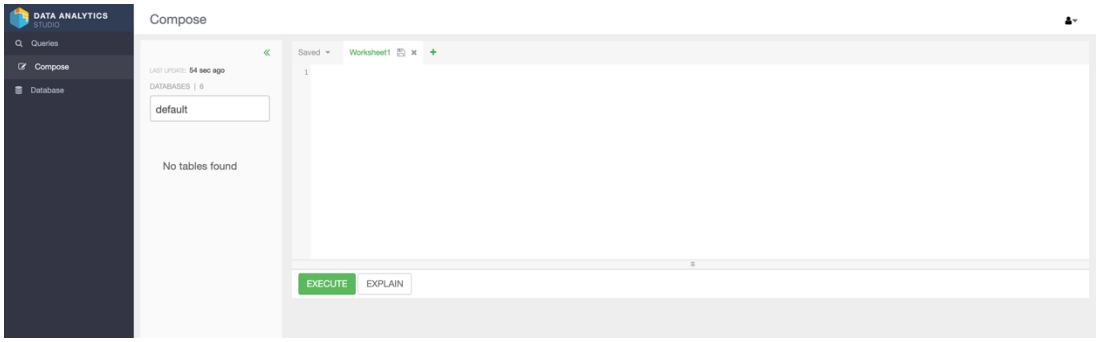


STEP 2 : Click on and "Open DAS"



This opens-up DAS editor for you to work on your queries and do necessary analysis of your data. Provide your assigned username and password.

In DAS, click on Compose option to see an editor for writing SQLs



STEP 3 : Create external tables for airlines, airports, planes & flights. Please follow below syntax & change username to your assigned username.

Create a database for your username.

```
create database if not exists username;
```

```
e.g.
create database if not exists puser01;
```

Airlines(external):

```
create external table if not exists username.airlines_ext (
  code string,
  description string
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS ORC
LOCATION 's3a://pse-airline-demo1/airlines_new_orc.db/airlines/'
;
```

Airports(external):

```
create external table if not exists username.airports_ext (
  iata string,
  airport string,
  city string,
  state double,
  country string,
  lat double,
  lon double
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS ORC
LOCATION 's3a://pse-airline-demo1/airlines_new_orc.db/airports'
;
```

Planes(external):

```
create external table if not exists username.planes_ext (
  tailnum string,
  owner_type string,
  manufacturer string,
  issue_date string,
  model string,
  status string,
  aircraft_type string,
  engine_type string,
  year int
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS ORC
LOCATION 's3a://pse-airline-demo1/airlines_new_orc.db/planes'
;
```

Flights(external):

```
create external table if not exists username.flights_ext (month int, dayofmonth int,
dayofweek int, deptime int, crsdeptime int, arrtime int, crsarrtime int, uniquecarrier
string, flightnum int, tailnum string, actualelapsedtime int, crselapsedtime int,
airtime int, arrdelay int, depdelay int, origin string, dest string, distance int,
taxiin int, taxiout int, cancelled int, cancellationcode string, diverted string,
carrierdelay int, weatherdelay int, nasdelay int, securitydelay int, lateaircraftdelay
int,year int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
STORED AS ORC
LOCATION 's3a://pse-airline-demo1/airlines_new_orc.db/flights'
;
```

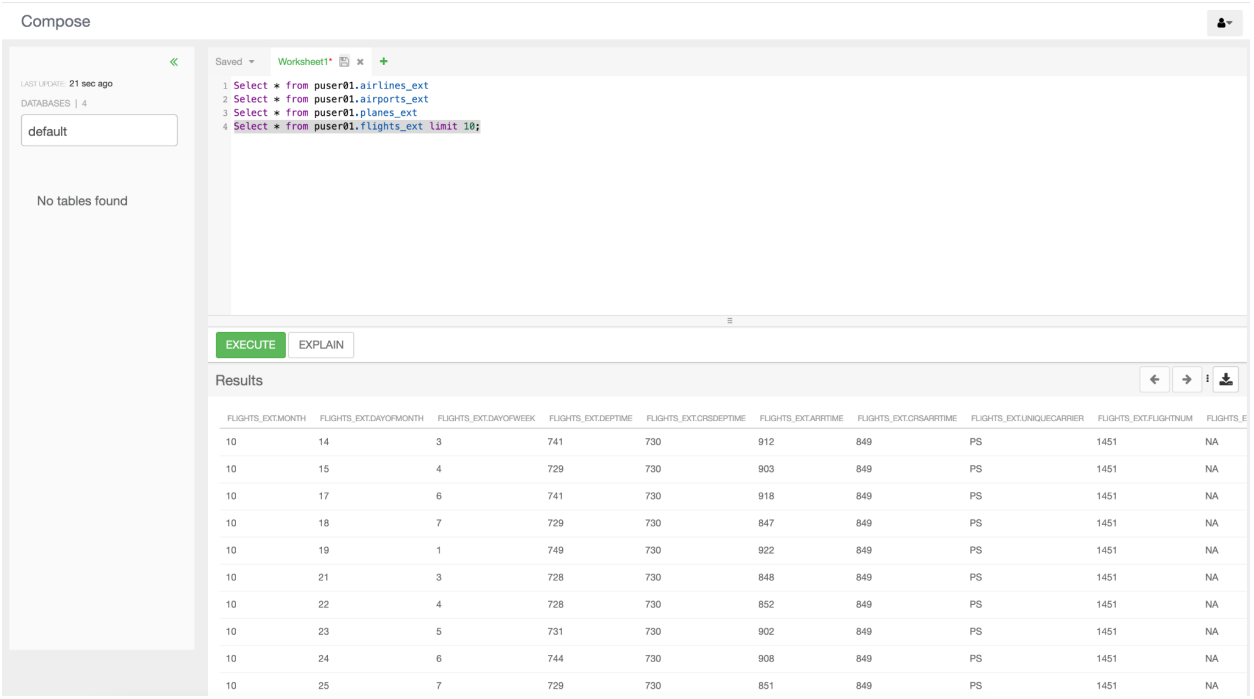
Check if these tables are created with necessary data. Run below commands one after another and check for results

a) `Select * from username.airlines_ext`

b) `Select * from username.airports_ext`

c) `Select * from username.planes_ext`

d) `Select * from username.flights_ext limit 10`



STEP 4 : Create managed tables

```
-- Airlines (Managed)
create table if not exists username.airlines (
  code string,
  description string
);


-- Airports (Managed)
create table if not exists username.airports (
  iata string,
  airport string,
  city string,
  state double,
  country string,
  lat double,
  lon double
);


-- Planes (Managed)
create table if not exists username.planes (
  tailnum string,
  owner_type string,
  manufacturer string,
  issue_date string,
  model string,
```

```
  status string,
  aircraft_type string,
  engine_type string,
  year int
);
```

```
-- Flights (Managed)

create table if not exists username.flights (month int, dayofmonth int, dayofweek int,
deptime int, crsdeptime int, arrtime int, crsarrtime int, uniquecarrier string,
flightnum int, tailnum string, actualelapsedtime int, crselapsedtime int, airtime int,
arrdelay int, depdelay int, origin string, dest string, distance int, taxiin int,
taxiout int, cancelled int, cancellationcode string, diverted string, carrierdelay
int, weatherdelay int, nasdelay int, securitydelay int, lateaircraftdelay int,year
int);
```

STEP 5 : Load data into managed tables from external tables

```
-- Airlines (Insert Managed)
insert overwrite table username.airlines
select code, description from username.airlines_ext;
```

```
-- Airports (Insert Managed)
insert overwrite table username.airports
select  iata, airport, city, state, country, lat, lon from username.airports_ext;
```

```
-- Planes (Insert Managed)
insert overwrite table username.planes
select  tailnum, owner_type, manufacturer, issue_date, model, status, aircraft_type,
engine_type, year from username.planes_ext;
```

```
-- Flights (Insert Managed) (Please note, this insert query may take some time due to
big data set that needs to be inserted into managed table)
INSERT OVERWRITE TABLE username.flights SELECT * FROM username.flights_ext;
```

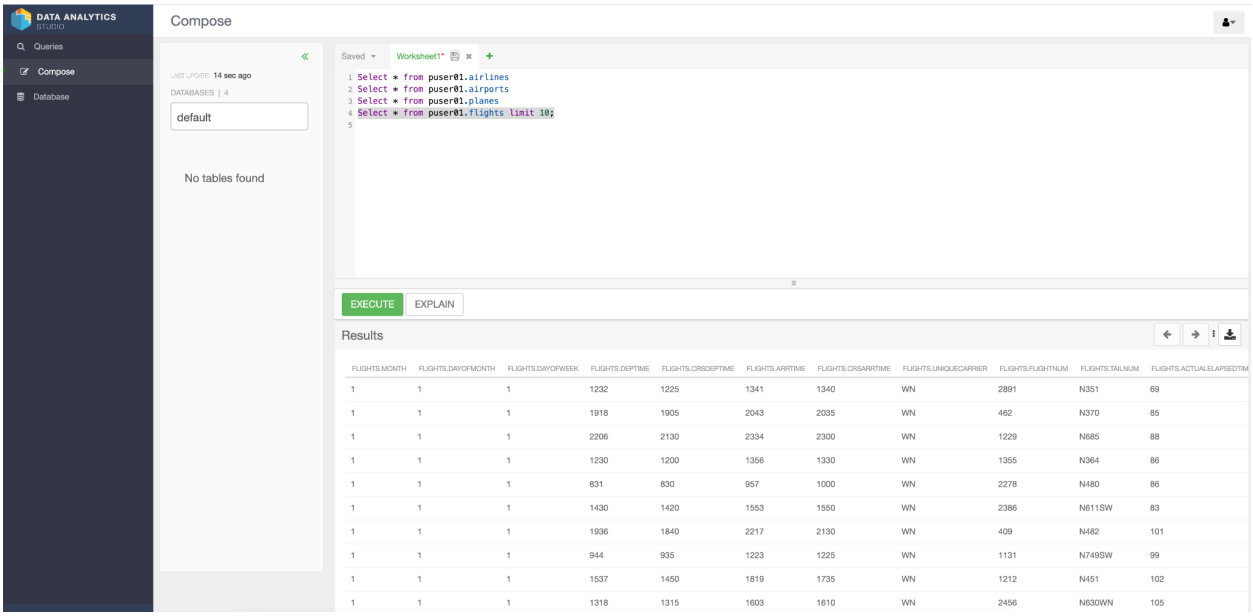
Check if these managed tables are created with necessary data. Run below commands one after another and check for results

a) `Select * from username.airlines`

b) `Select * from username.airports`

c) `Select * from username.planes`

d) `Select * from username.flights limit 10`



STEP 6 : Create primary keys and foreign keys

This is done so that Materialized View rewriting can be done properly later on. Also, in general, specifying these constraints helps the query planner to build more effective plans. But note that these constraints are not actually enforced (i.e. no referential integrity checks are done). Be sure to replace all "username" instances with your assigned username.

```
ALTER TABLE username.planes ADD CONSTRAINT planes_pk_username PRIMARY KEY (tailnum)
DISABLE NOVALIDATE;
ALTER TABLE username.flights ADD CONSTRAINT planes_fk_username FOREIGN KEY (tailnum)
REFERENCES username.planes(tailnum) DISABLE NOVALIDATE RELY;

ALTER TABLE username.airlines ADD CONSTRAINT airlines_pk_username PRIMARY KEY (code)
DISABLE NOVALIDATE;
ALTER TABLE username.flights ADD CONSTRAINT airlines_fk_username FOREIGN KEY
(uniquecarrier) REFERENCES username.airlines(code) DISABLE NOVALIDATE RELY;

ALTER TABLE username.airports ADD CONSTRAINT airports_pk_username PRIMARY KEY (iata)
DISABLE NOVALIDATE;
ALTER TABLE username.flights ADD CONSTRAINT airports_orig_fk_username FOREIGN KEY
(origin) REFERENCES username.airports(iata) DISABLE NOVALIDATE RELY;
ALTER TABLE username.flights ADD CONSTRAINT airports_dest_fk_username FOREIGN KEY
(dest) REFERENCES username.airports(iata) DISABLE NOVALIDATE RELY;
```

STEP 6 : Materialized View in Hive Virtual Warehouse – Initial setup Step

Now that our dataset is ready, from this place it is common to do things like build complex reports, do ad hoc analysis, run operational dashboards, and create downstream data marts for subject specific uses. This scenario is commonly referred to as an Enterprise Data Warehouse - all the enterprise data is cleaned, in one place, and trustworthy.

In this layer it is common to use Materialized Views to accelerate BI performance. Hive supports Materialized Views and has the ability to rewrite queries automatically if they can take advantage of an existing materialized view. It utilizes Apache Calcite to perform partial and full rewrites. The rich SQL syntax support in Hive is also very useful here, as there is typically a very wide array of SQL needed in this layer. And finally, with dashboarding scenarios in particular, the Hive query result cache can provide a big performance boost.

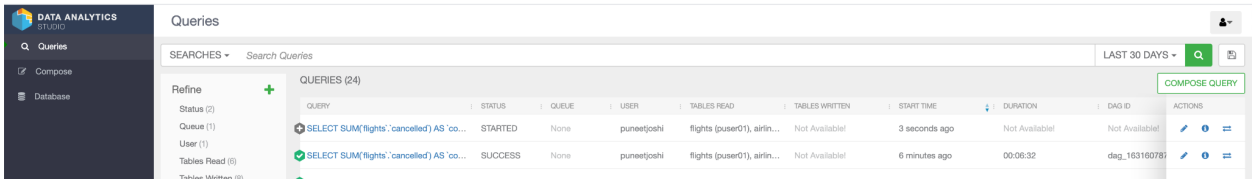
In this exercise we'll use a BI generated query and try to improve the performance using the Materialized View feature.

Open DAS in the Hive Virtual Warehouse that was used above, and do the following:

Execute the following BI query two times in a row:

```
SELECT
  SUM(`flights`.`cancelled`) AS `col_1`,
  SUM(1) AS `col_2`,
  MIN(`airlines`.`description`) AS `col_3`,
  `airlines`.`code` AS `code`
FROM
  `username`.`flights` `flights`
  JOIN `username`.`airlines` `airlines` ON (`flights`.`uniquecarrier` =
`airlines`.`code`)
WHERE
  (`flights`.`month` = 12)
GROUP BY
  `airlines`.`code`;
```

Go to DAS query history and view the details for the queries that you've just executed. You can see the tables read and a visual explain plan for the query. Note the path that was followed to execute the query and the time spent. Also, note that in the second query, the results were returned much faster and the visual explain plan is simple. This is because the second execution simply read from the query result cache.



STEP 7 : Materialized View in Hive Virtual Warehouse – Create step

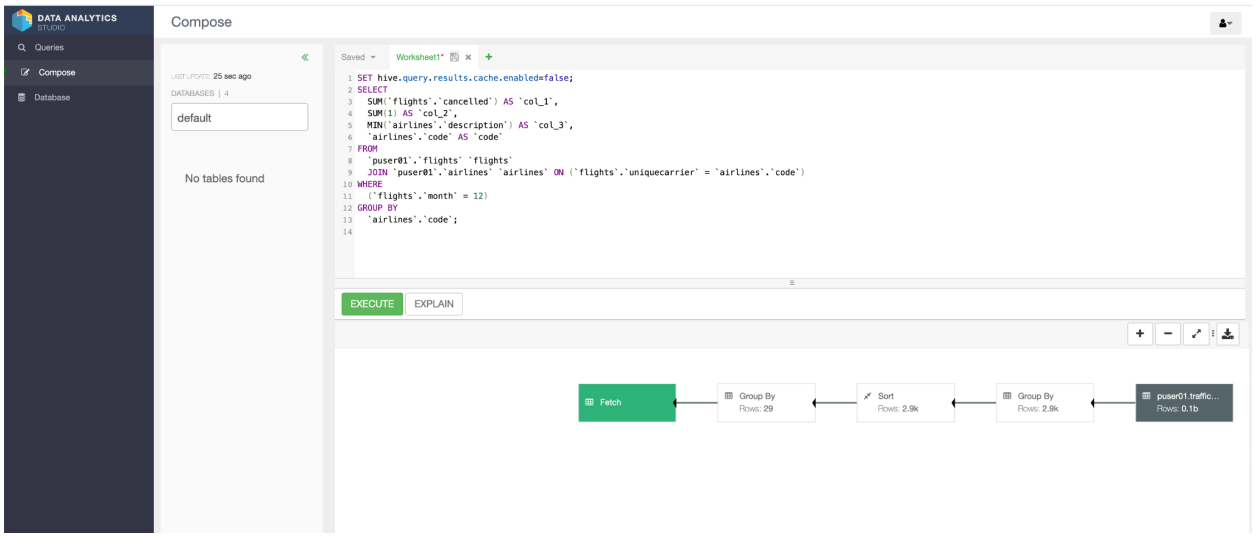
Create a Materialized View that selects the columns used by the query in STEP 6 and performs the same join, no need to group or perform any "group by" or "where" in this step

```
CREATE MATERIALIZED VIEW username.traffic_cancel_airport as
SELECT
  `flights`.`cancelled`,
  `airlines`.`description`,
  `airlines`.`code` AS `code`,
  `flights`.`month` AS month,
  `flights`.`year` AS year
FROM
  `username`.`flights` `flights`
  JOIN `username`.`airlines` `airlines` ON (`flights`.`uniquecarrier` =
`airlines`.`code`);
```

Execute the BI query again, this time disabling the query result cache to force usage of the materialized view:

```
SET hive.query.results.cache.enabled=false;
SELECT
  SUM(`flights`.`cancelled`) AS `col_1`,
  SUM(1) AS `col_2`,
  MIN(`airlines`.`description`) AS `col_3`,
  `airlines`.`code` AS `code`
FROM
  `username`.`flights` `flights`
  JOIN `username`.`airlines` `airlines` ON (`flights`.`uniquecarrier` =
`airlines`.`code`)
WHERE
  (`flights`.`month` = 12)
GROUP BY
  `airlines`.`code`;
```

In DAS, Click on EXPLAIN (next to EXECUTE button), to check the path and the visual plan for the last executed query. Check if the table read is the same and the path. Note that now it is using the materialized view (traffic_cancel_airport) and this means that all queries from the BI(Tableau, Qlik, etc) dashboard that performs the same join and use the same columns will follow this path, doesn't matter the month that comes in the filter.



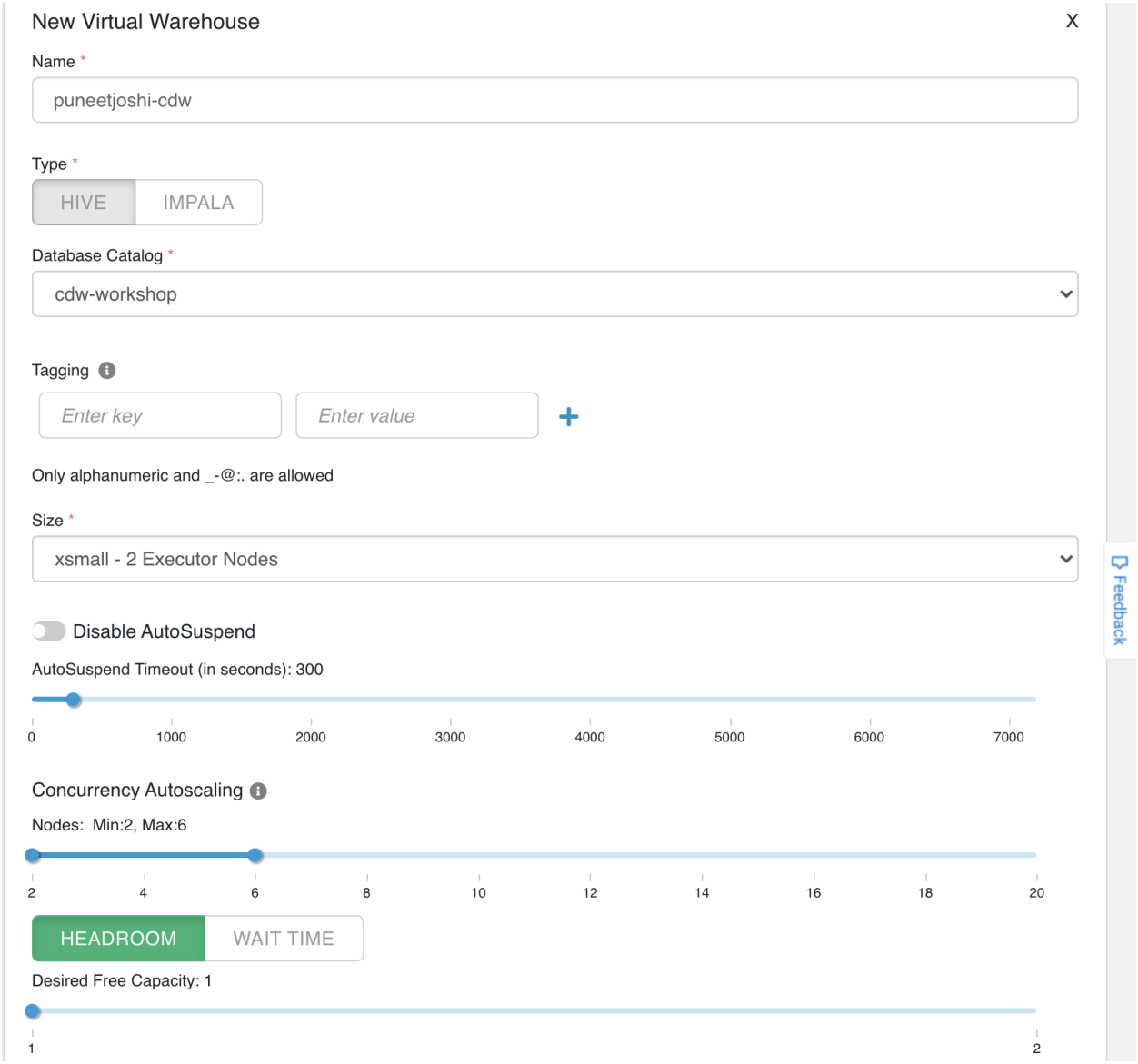
Lab 3 – Auto-Scaling, Auto-Suspend & Caching Cloudera DataWarehouse

In this lab, you will see autoscaling & auto-suspend capabilities of Cloudera DataWarehouse. Virtual Warehouses can use Hive or Impala as the underlying execution engine. Typically, Hive is used to support complex reports and enterprise dashboards. Impala is used to support interactive, ad-hoc analysis. When you create a Virtual Warehouse, you set auto-scaling to make sure you have adequate resources to meet increases in demand. Auto-scaling settings also insure that your Virtual Warehouse relinquishes resources when demand decreases to save costs.

When you create new Virtual Warehouse instances, you can set auto-scaling thresholds. These thresholds set limits on automatic cluster scaling to meet workload demands. Setting these limits prevents warehouses from consuming too many resources when workload demands increase or decrease.

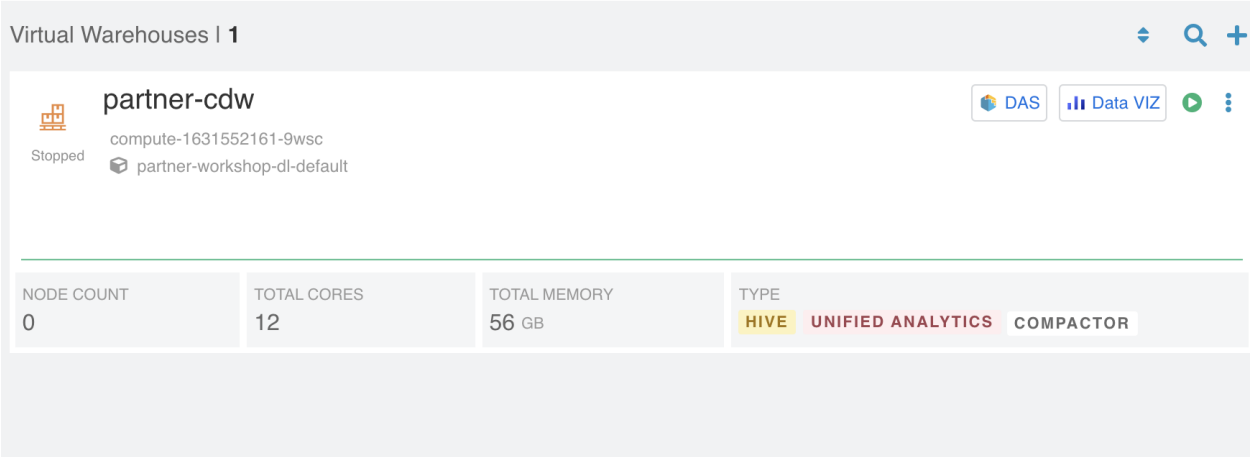
In Cloudera Data Warehouse (CDW) service, when you tune Hive-LLAP Virtual Warehouses, you set the auto-suspend timeout, the minimum and maximum number of nodes for your virtual cluster, when your cluster should scale up and down, and whether to use query isolation for scan-heavy, data-intensive queries.

Here is what we set as Auto-scale & Auto-suspend properties for our DataWarehouse:



STEP 1 : Auto-Susped

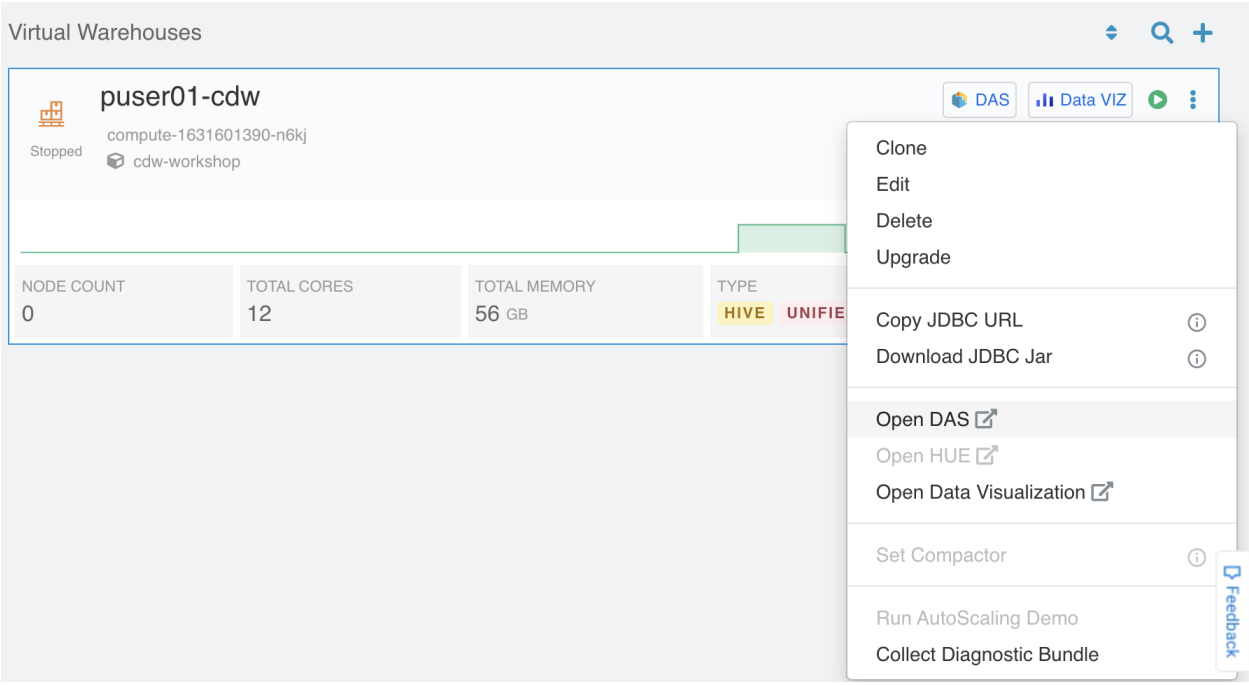
Since its been some time we ran our queries, Cloudera DataWarehouse has amazing capabilities to release off resources when its not in use. If you look at your Virtual Warehouse screen, you will notice that the current node count is 0. Which means, it has auto-suspended all the resources. As per our settings, if our Virtual Warehouse does not see any incoming queries for more than 300 seconds, it automatically suspends all the container nodes.



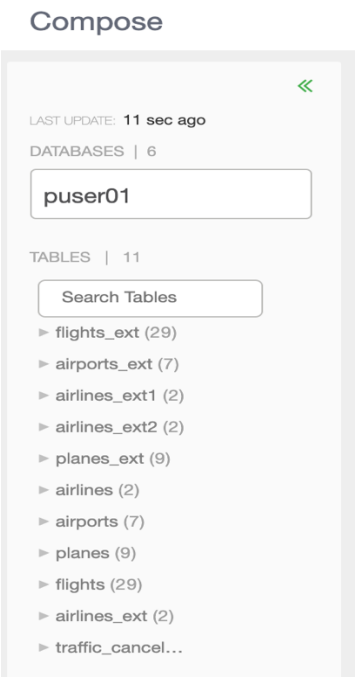
STEP 2 : Auto-Scale

Now, lets look at Auto-scale capabilities. Click on and “Open DAS” and open SQL editor by Clicking Compose on the left.

For us to show auto-scaling, open DAS in one another tab, so that we run complex queries parallelly and see how Cloudera Virtual Warehouse auto-scales

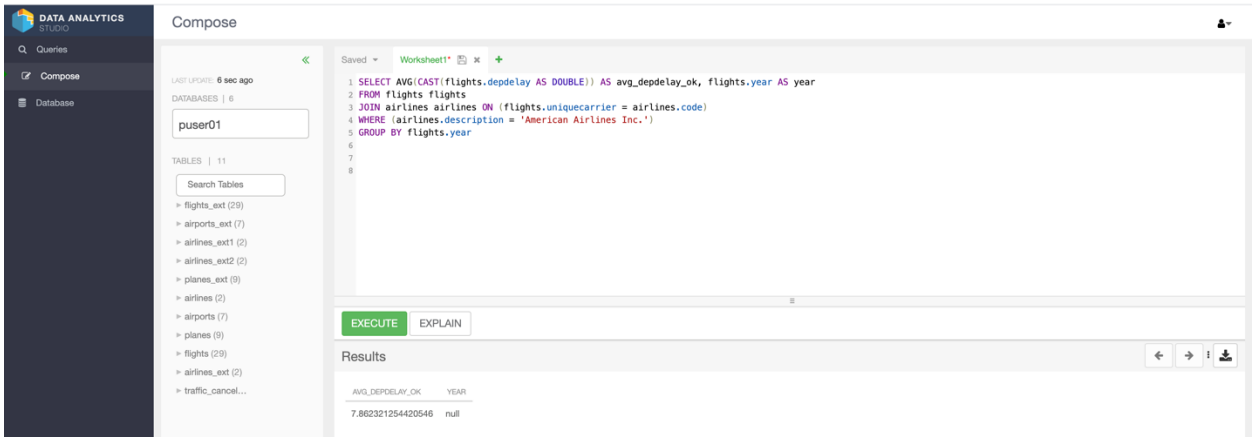


Choose your database in the databases section in both the DAS windows you have opened



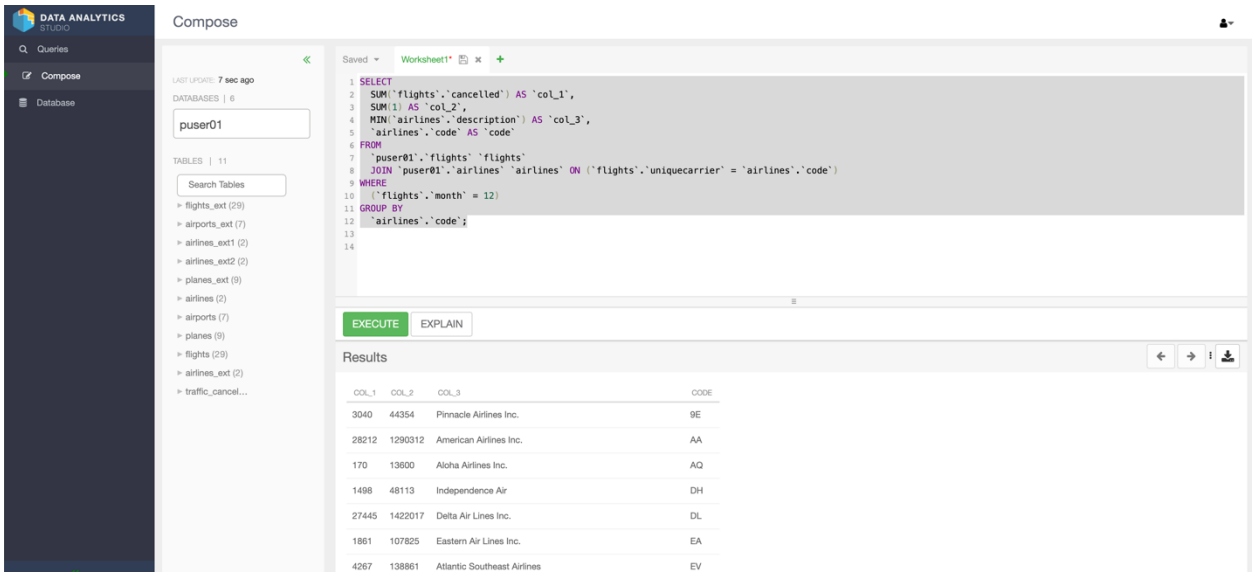
In the 1st DAS query editor, run the following command:

```
SELECT AVG(CAST(flights.depdelay AS DOUBLE)) AS avg_depdelay_ok, flights.year AS year
FROM flights flights
JOIN airlines airlines ON (flights.uniquecarrier = airlines.code)
WHERE (airlines.description = 'American Airlines Inc.')
GROUP BY flights.year
```

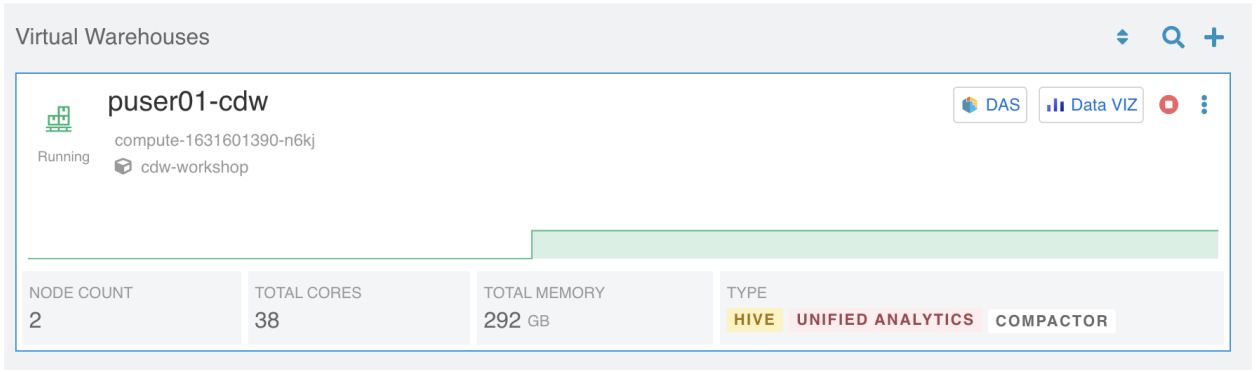


And in the 2nd DAS query editor, run the following command:

```
SELECT
  SUM(`flights`.`cancelled`) AS `col_1`,
  SUM(1) AS `col_2`,
  MIN(`airlines`.`description`) AS `col_3`,
  `airlines`.`code` AS `code`
FROM
  `username`.`flights` `flights`
  JOIN `username`.`airlines` `airlines` ON (`flights`.`uniquecarrier` =
`airlines`.`code`)
WHERE
  (`flights`.`month` = 12)
GROUP BY
  `airlines`.`code`;
```



As soon as you run both the queries, quickly switch back to the Virtual Data Warehouse screen of CDP and see an increase in the number of nodes.



Both the queries for the first time may take some time. Now, for the 2nd time if you run the same queries with different filters, it returns results in a few seconds. Try running these queries and see how quickly your query results come out.

```
SELECT AVG(CAST(flights.depdelay AS DOUBLE)) AS avg_depdelay_ok, flights.year AS year
FROM flights flights
JOIN airlines airlines ON (flights.uniquecarrier = airlines.code)
WHERE (airlines.description = 'United Air Lines Inc.')
GROUP BY flights.year
```

```
SELECT
  SUM(`flights`.`cancelled`) AS `col_1`,
  SUM(1) AS `col_2`,
  MIN(`airlines`.`description`) AS `col_3`,
  `airlines`.`code` AS `code`
FROM
  `username`.`flights` `flights`
  JOIN `username`.`airlines` `airlines` ON (`flights`.`uniquecarrier` =
`airlines`.`code`)
WHERE
  (`flights`.`month` = 11)
GROUP BY
  `airlines`.`code`;
```

In CDP Data Warehouse service, frequently accessed data is cached in a storage layer on S3 so that it can be quickly retrieved for subsequent queries, which boosts data warehouse performance.

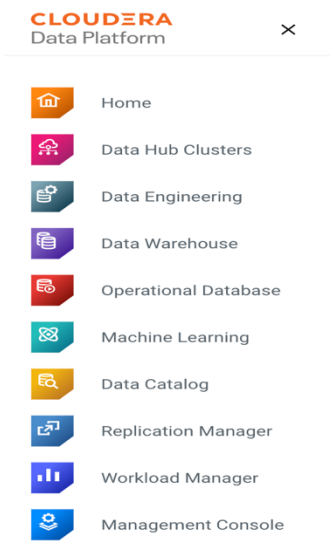
Lab 4 – Security & Governance with Ranger & Atlas

Apache Ranger manages access control through a user interface that ensures consistent policy administration across Cloudera Data Platform (CDP) components. Security administrators can define security policies at the database, table, column, and file levels, and can administer permissions for specific LDAP-based groups or individual users. Rules based on dynamic

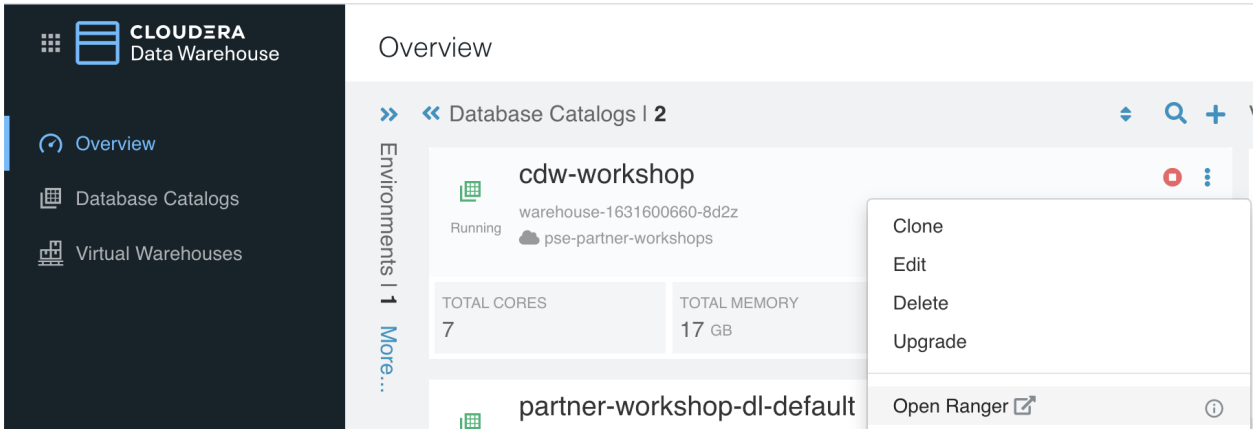
conditions such as time or geolocation can also be added to an existing policy rule. The Ranger authorization model is pluggable and can be easily extended to any data source using a service-based definition.

STEP 1 : Ranger Lab

Click on left top to choose DataWarehouse



Now, under your Database Catalog, Click on and choose open Ranger



Once you get into Ranger, you will get to see a high level view of Ranger policies for all major components of CDP.

In our case, we will create Ranger policies for our Data Warehouse access. Please choose

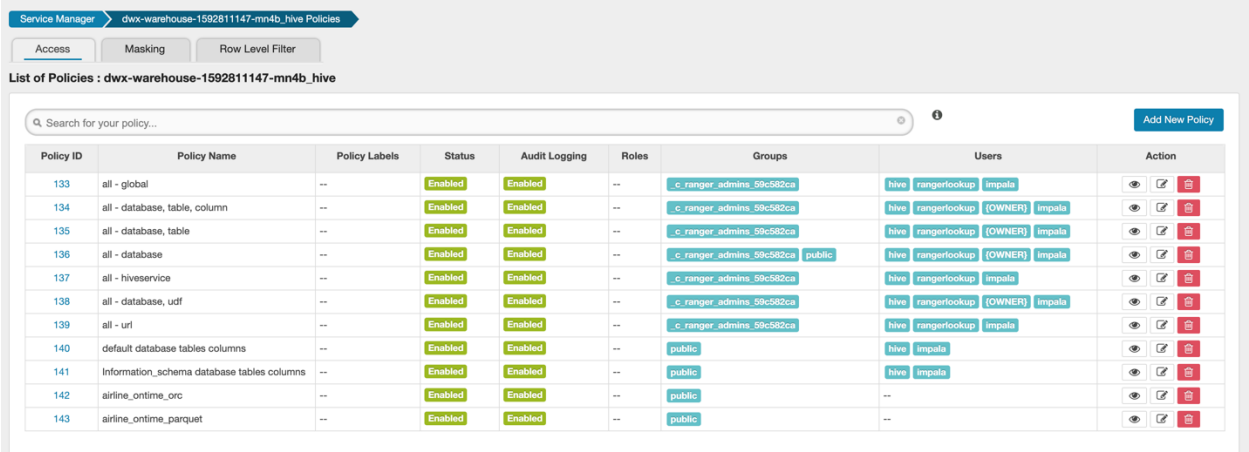
dwx-warehouse-1631600660-8d2z_hive

as shown in below screen-shot

Once you get into Ranger policies for your database, you will see a view of ranger policies for Access, Masking & Row-level filters. We will work on each of these categories

STEP 2 : Creating Ranger Access policy

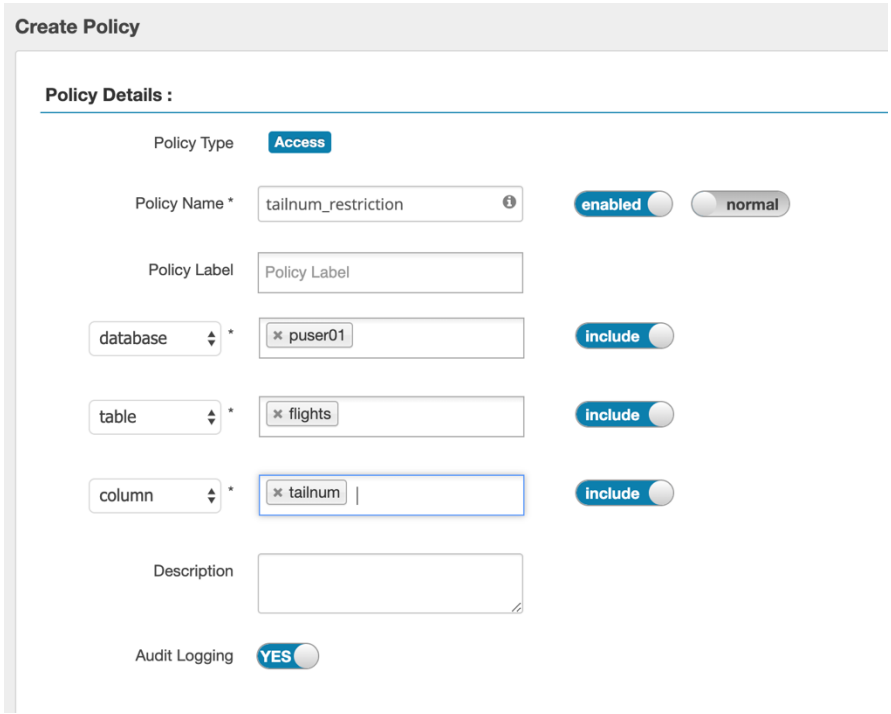
Under Access tab, click on **Add New Policy**



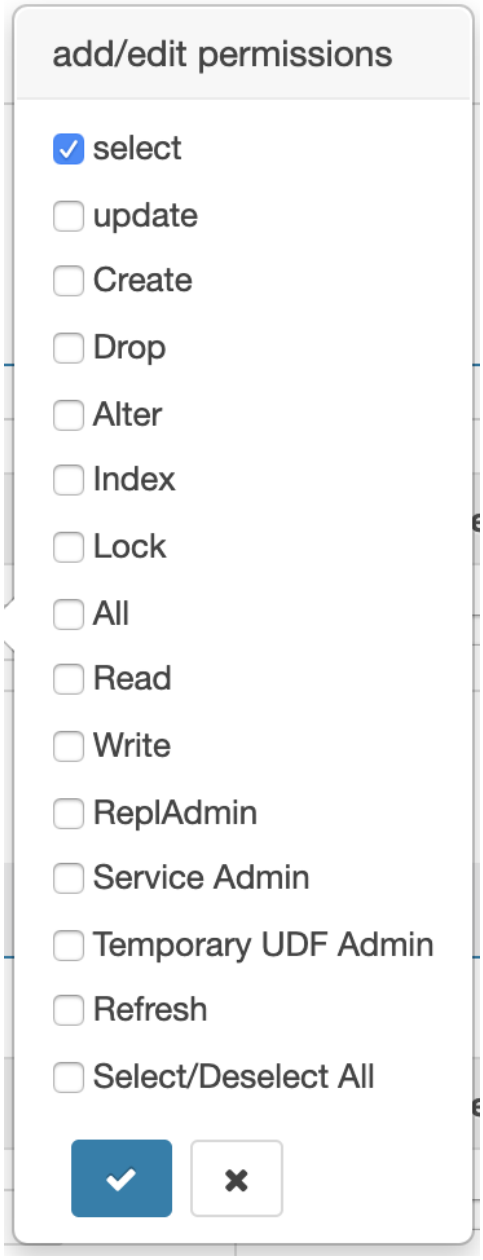
Now, we will create an Access policy for restricting access to **Flights** table and more specifically access restriction on a column named **tailnum**.

Do not add any **Allow Conditions**. We will add only **Deny Conditions**
Refer below screen-shot to create the access policy.

****Please use your database created in STEP3 of LAB2****



Add necessary permission as per below screen-shot



Save the policy.

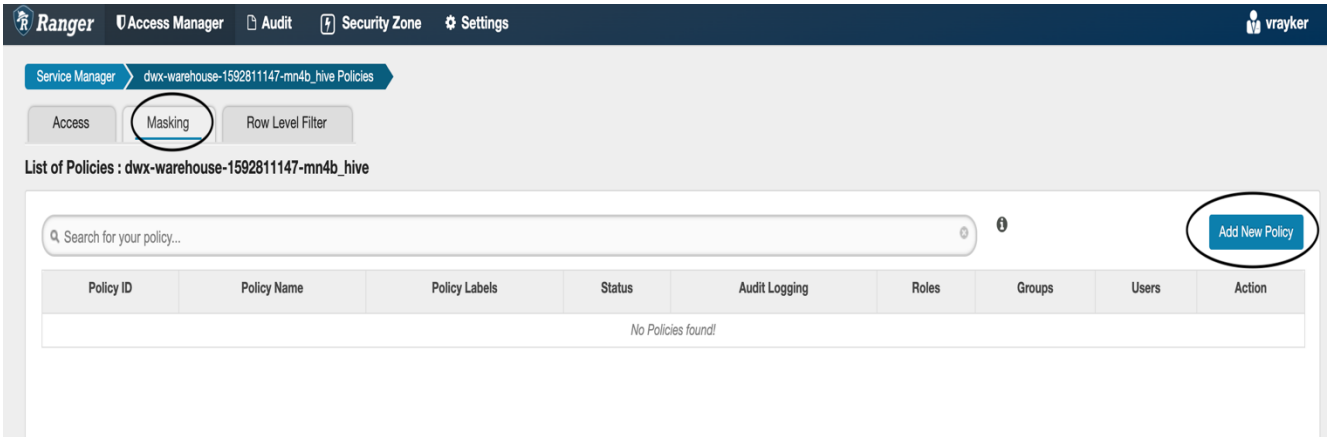
Now, go back to DAS screen and run this query

Select * from flights

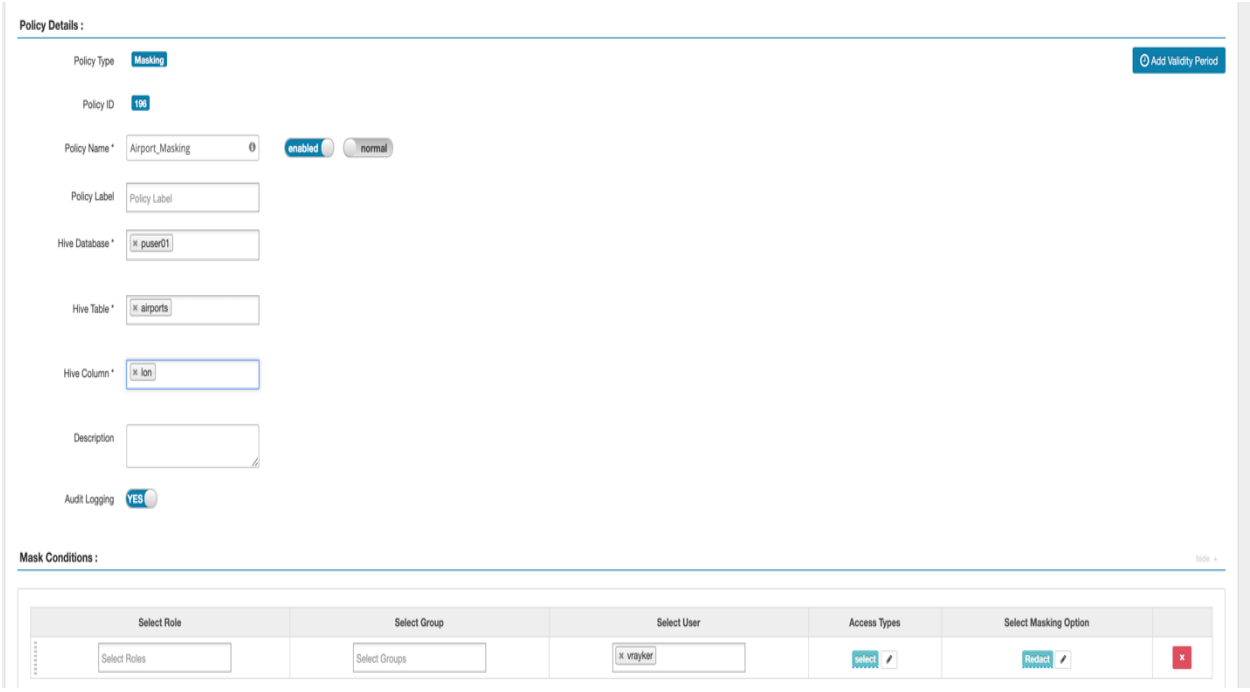
As you can see, you will see access restriction on flights column tailnum

STEP 3 : Creating Masking policy

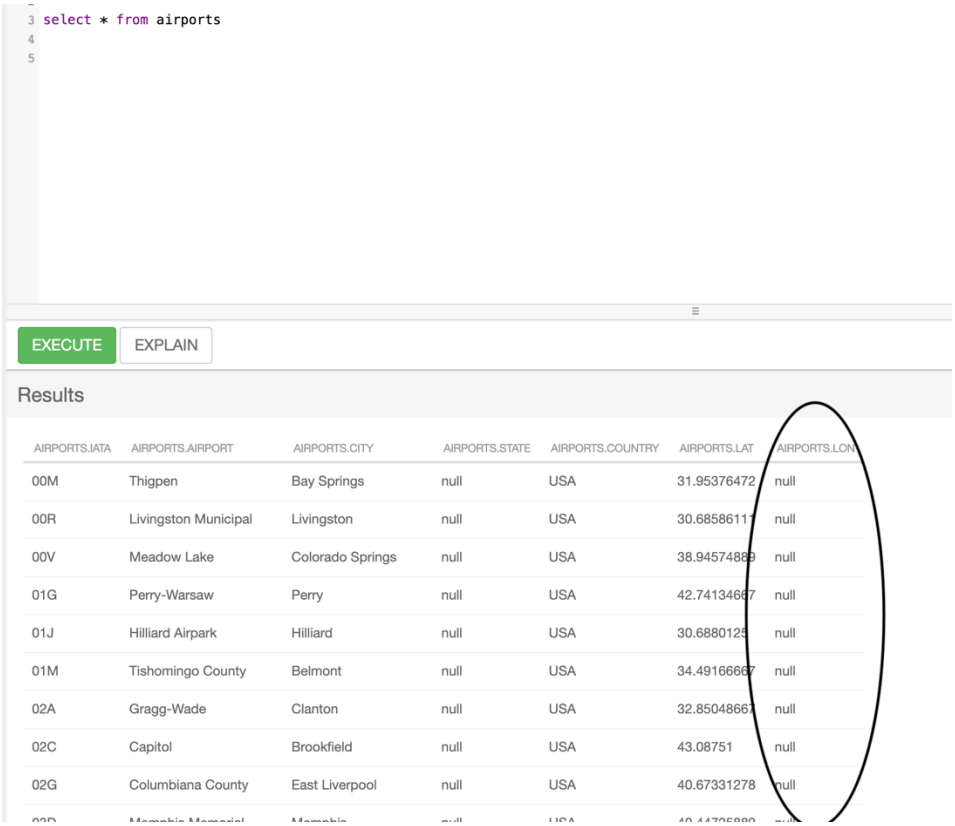
Click on Masking tab and Add New Policy



Now, we will create a masking policy to mask Longitude column in Airport table. Create the policy as per the screen-shot below. Select your username assigned. And also choose Access type and Masking option as mentioned below.
****Please use your Hive database created in STEP3 of LAB2****



Save the policy.
Now, go back to DAS and run the query and the masking live in action
Select * from airports

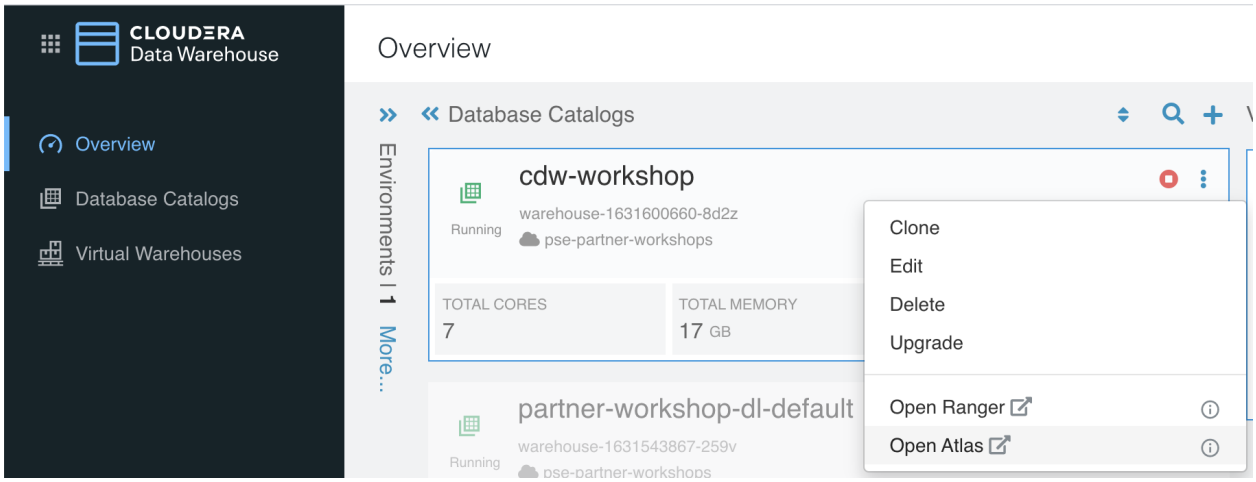


Similarly, we can also create Row-level filters to restrict access at a row-level.

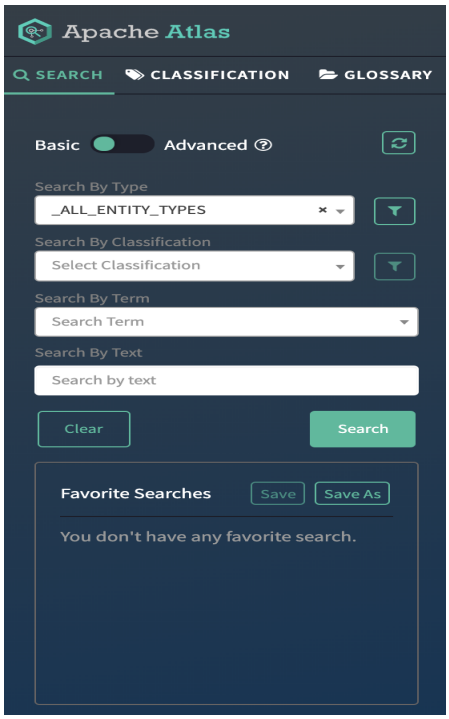
STEP 4 : Atlas lab for Governance

Apache Atlas provides scalable governance for Enterprises that is driven by metadata. Atlas, at its core, is designed to easily model new business processes and data assets with agility. This flexible type system allows exchange of metadata with other tools and processes within and outside of the Cloudera stack, thereby enabling platform-agnostic governance controls that effectively address compliance requirements

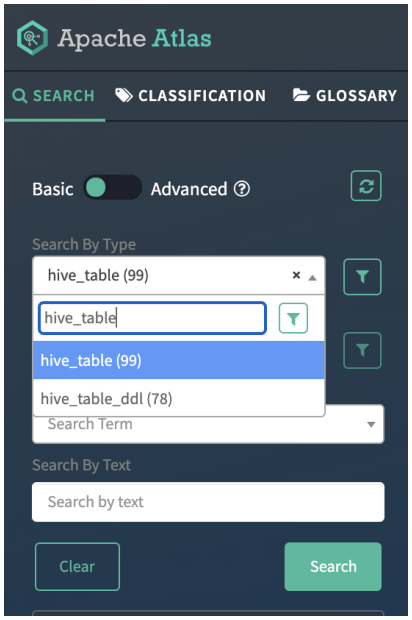
Now, under your Database Catalog, Click on and choose open Atlas



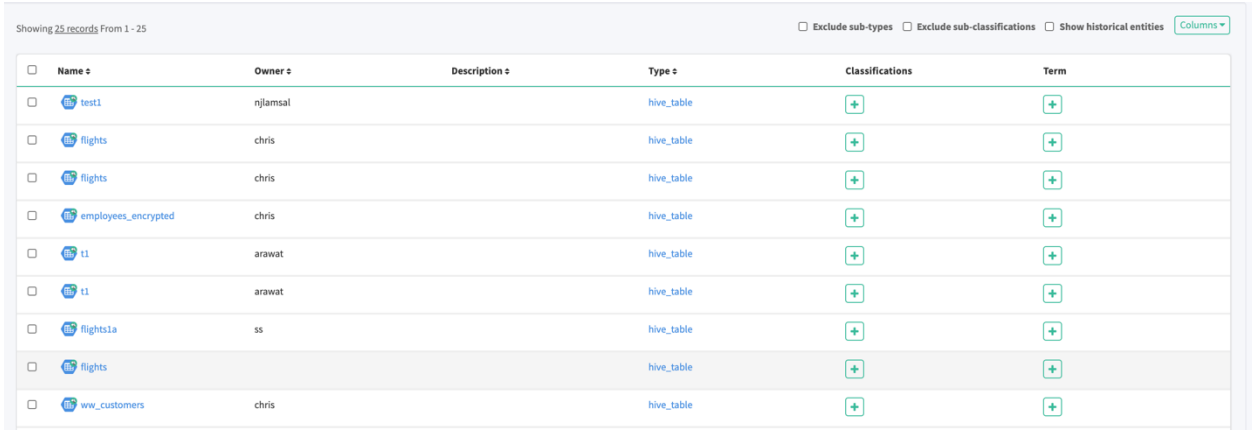
This opens-up Atlas page for you



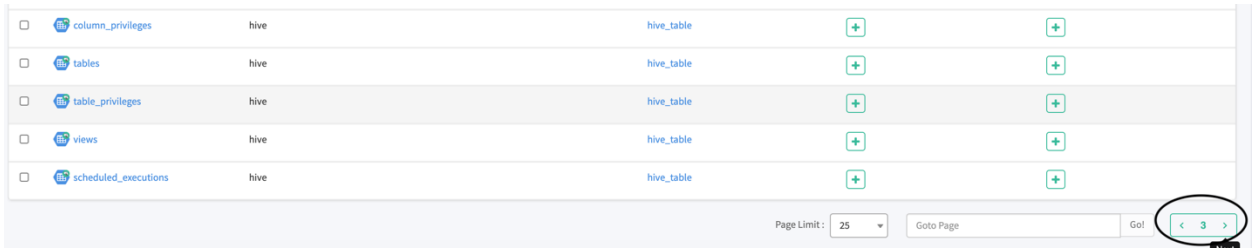
Now, in the search by type, select hive_table as per below screen-shot and hit search



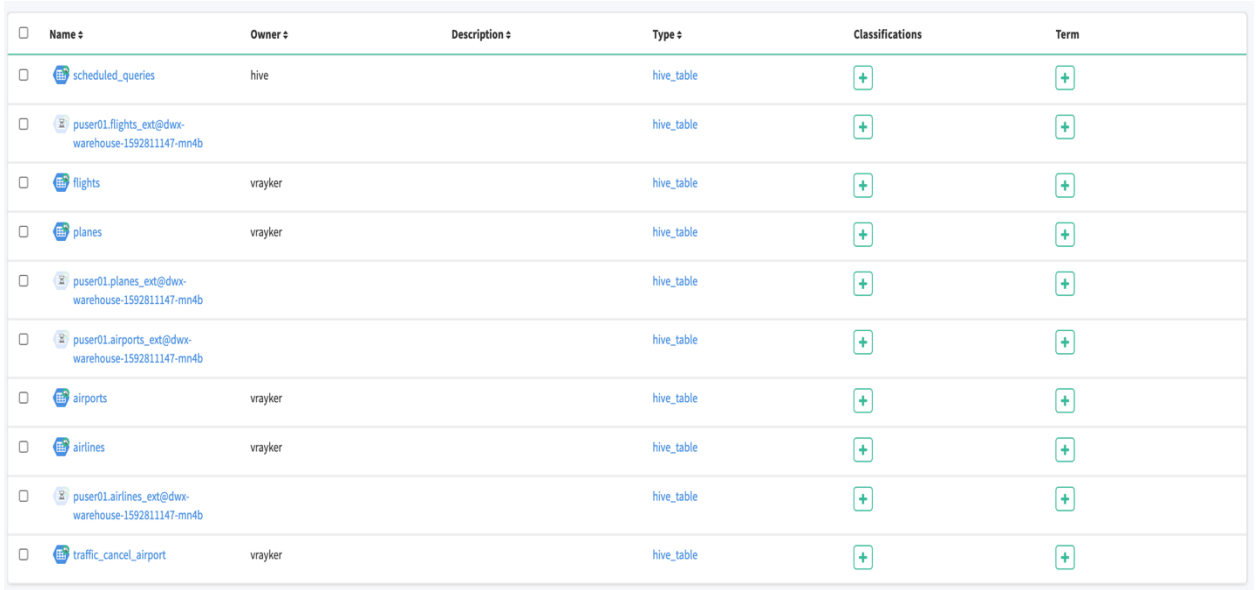
You will see search results coming now. All Hive tables in your CDP environment.



Now, bottom of the page, move forward to next pages to see all hive tables you created during previous labs

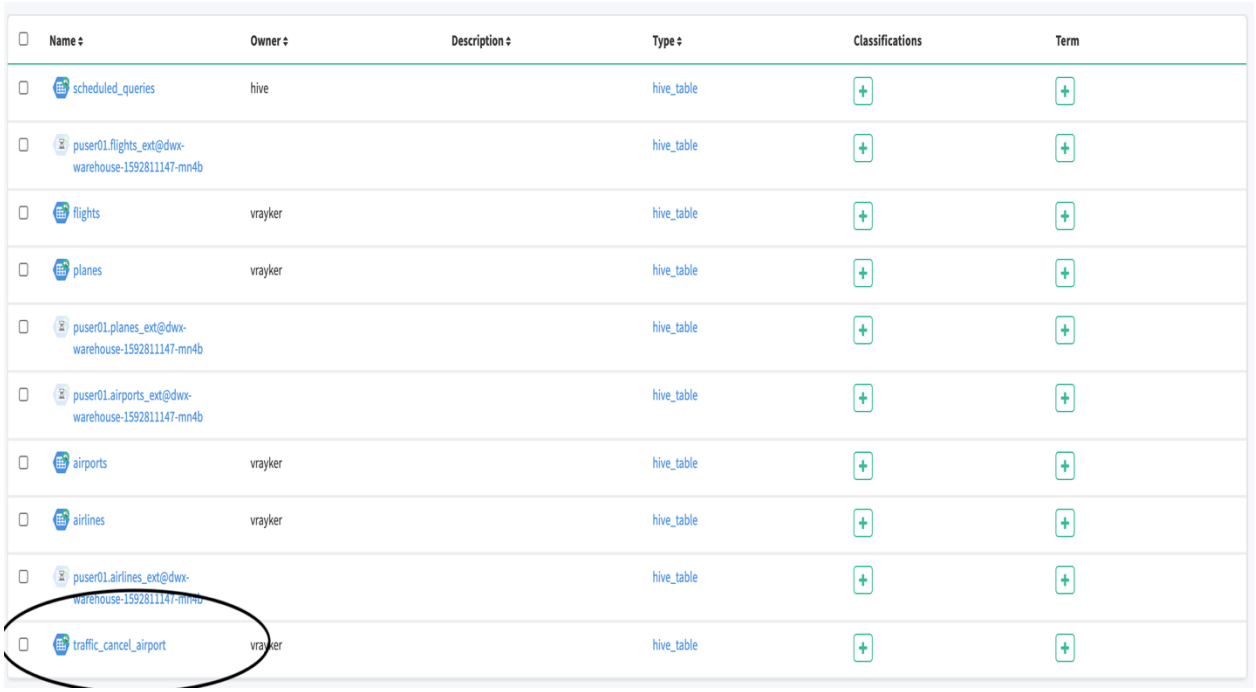


Against your username, you will see all hive tables you created during the lab.



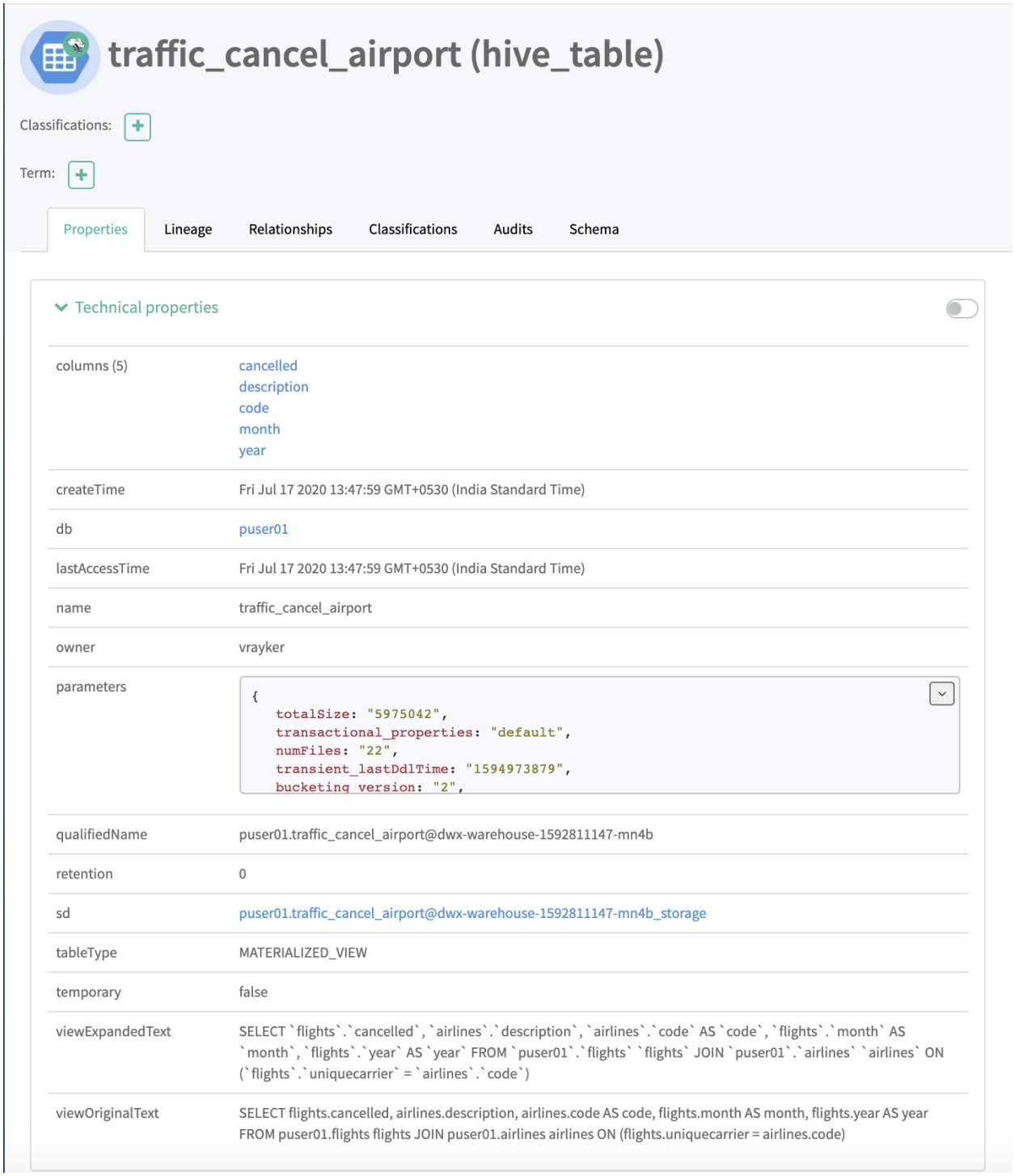
Name	Owner	Description	Type	Classifications	Term
scheduled_queries	hive		hive_table		
puser01.flights_ext@dwx-warehouse-1592811147-mn4b			hive_table		
flights	vrayker		hive_table		
planes	vrayker		hive_table		
puser01.planes_ext@dwx-warehouse-1592811147-mn4b			hive_table		
puser01.airports_ext@dwx-warehouse-1592811147-mn4b			hive_table		
airports	vrayker		hive_table		
airlines	vrayker		hive_table		
puser01.airlines_ext@dwx-warehouse-1592811147-mn4b			hive_table		
traffic_cancel_airport	vrayker		hive_table		

Now, click on the materialized view you created **traffic_cancel_airport**

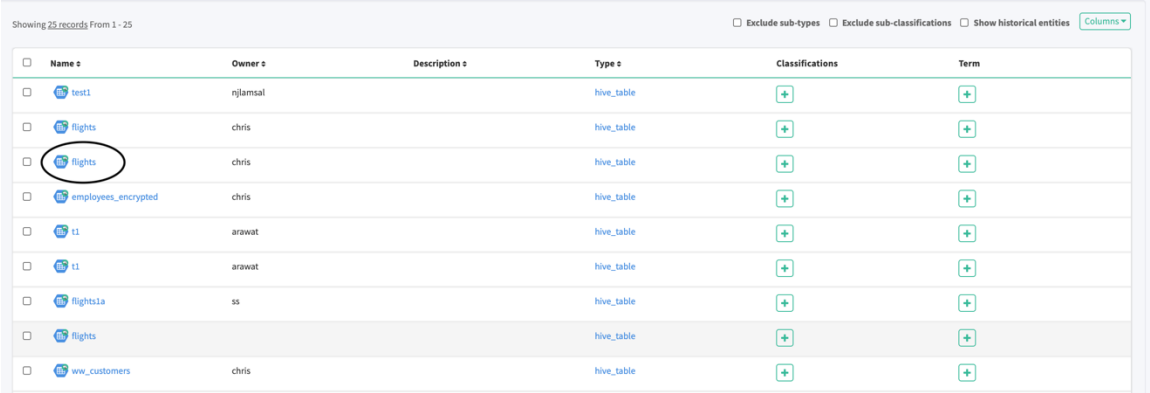


Name	Owner	Description	Type	Classifications	Term
scheduled_queries	hive		hive_table		
puser01.flights_ext@dwx-warehouse-1592811147-mn4b			hive_table		
flights	vrayker		hive_table		
planes	vrayker		hive_table		
puser01.planes_ext@dwx-warehouse-1592811147-mn4b			hive_table		
puser01.airports_ext@dwx-warehouse-1592811147-mn4b			hive_table		
airports	vrayker		hive_table		
airlines	vrayker		hive_table		
puser01.airlines_ext@dwx-warehouse-1592811147-mn4b			hive_table		
traffic_cancel_airport	vrayker		hive_table		

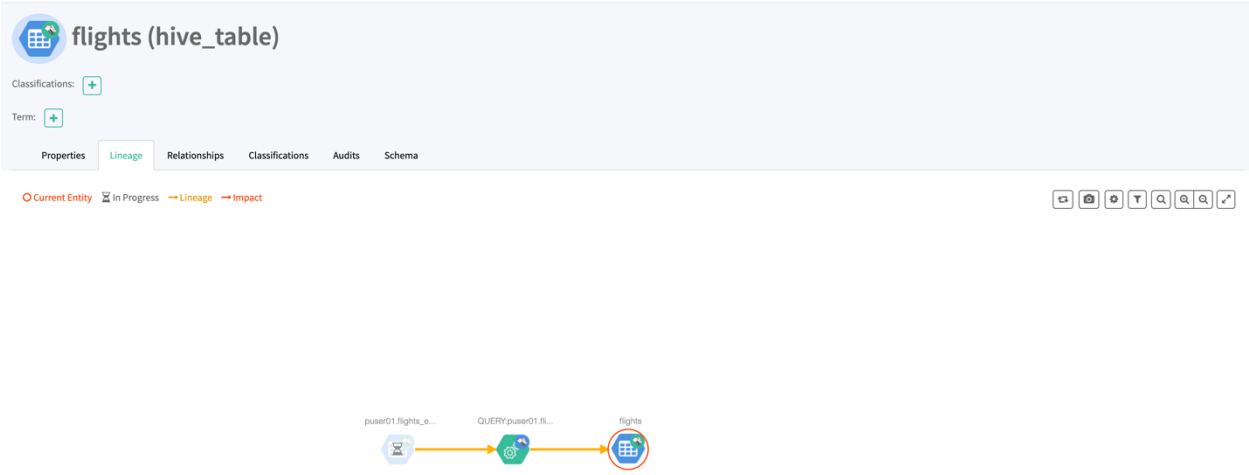
Go through the properties for this materialized view the Atlas has automatically captured.



Now, go back to your search results and choose flights table



Look at all the properties and click on Lineage. Spend some time to click on lineage results and how Atlas has captured lineage on how **flights** table got created

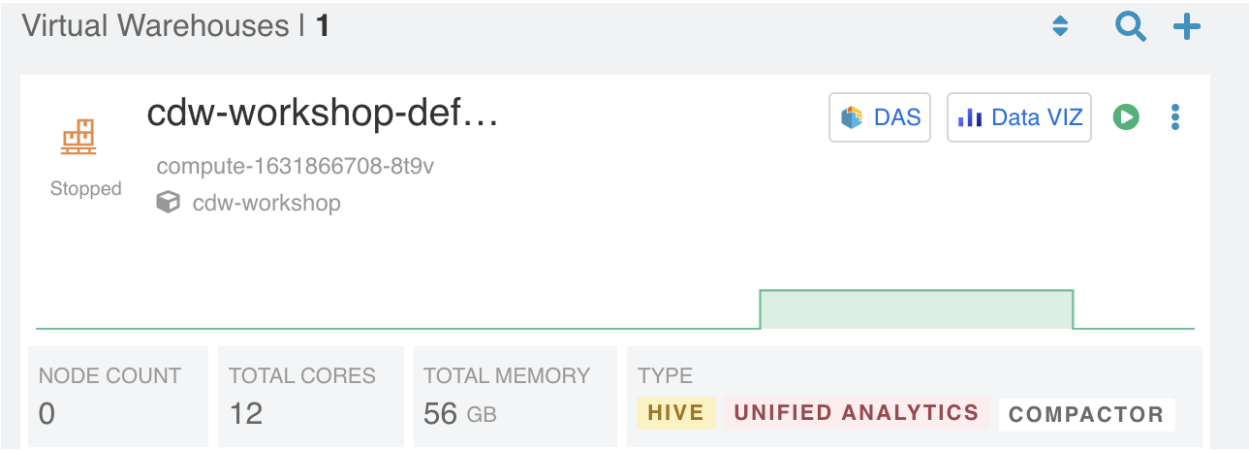


Lab 5 – Data Visualization

In this lab you will get an opportunity to explore and work with inbuilt Data Visualization capability of the CDP platform. To execute this lab we have created a default warehouse with Visualization enabled.

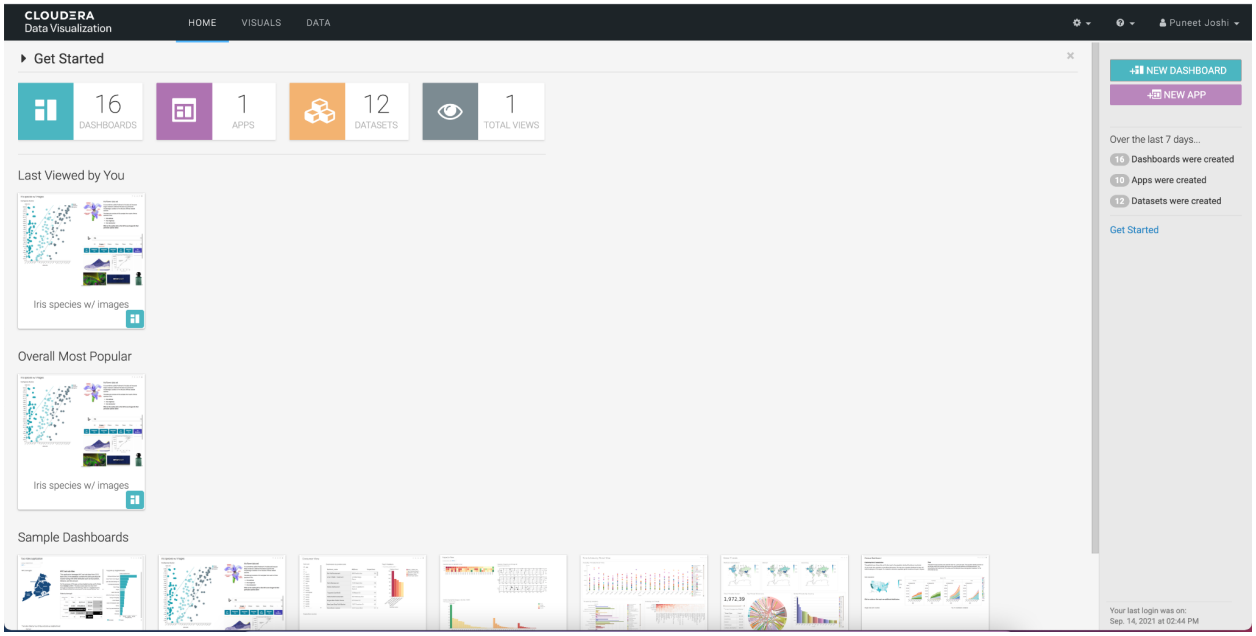
Step 1: Navigate back to Data Warehouse home page.

Search for “cdw-workshop-default” using the search option. Below is the screen shot.



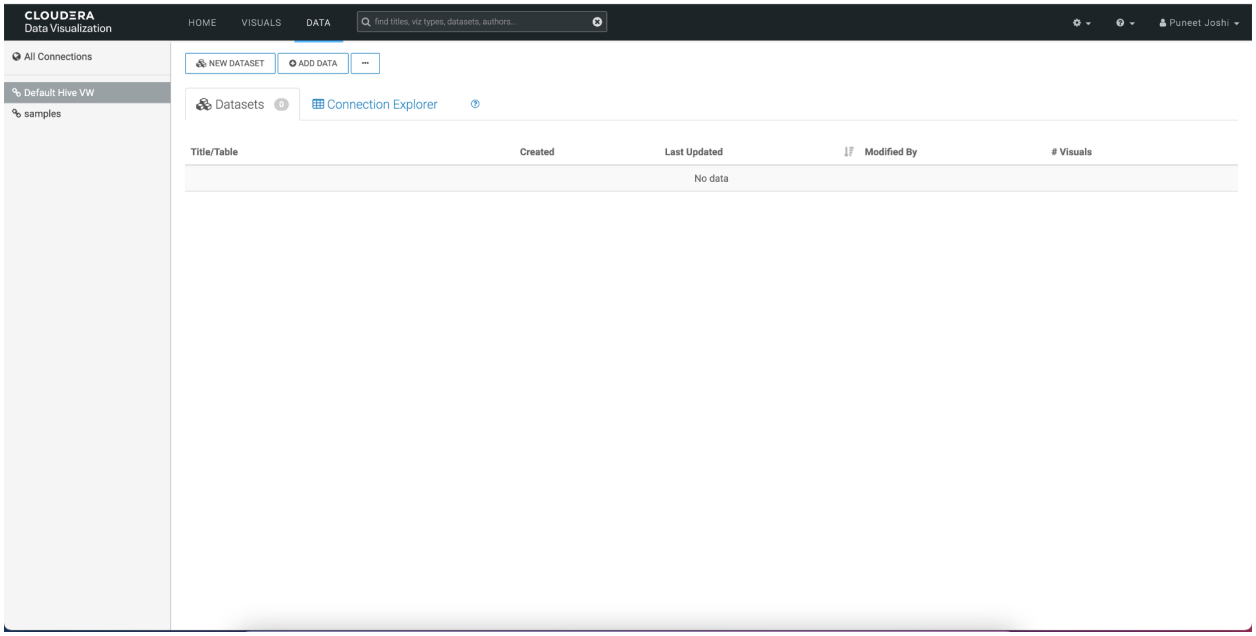
Click on Data Viz it will take you to the home page of data visualization offering.

Spend some time exploring the different options available . Especially the sample dashboards.



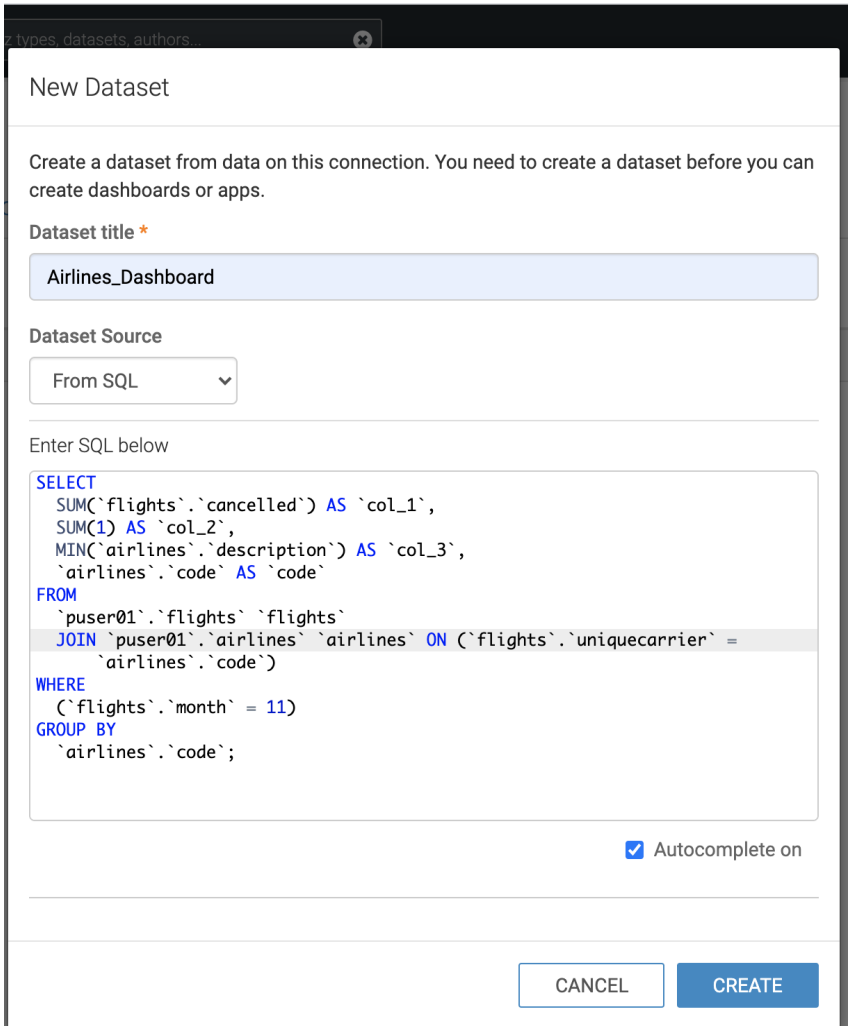
Let's create our own dashboard and see how simple it is to create a dashboard from the data which we created in our previous labs. Before proceeding with this we need to disable the ranger policy which we created to restrict access to the 'tailnum' column.

Step 2: Navigate to the "Data" tab on the screen.



Click on “New Dataset” to add source data for our dashboard.

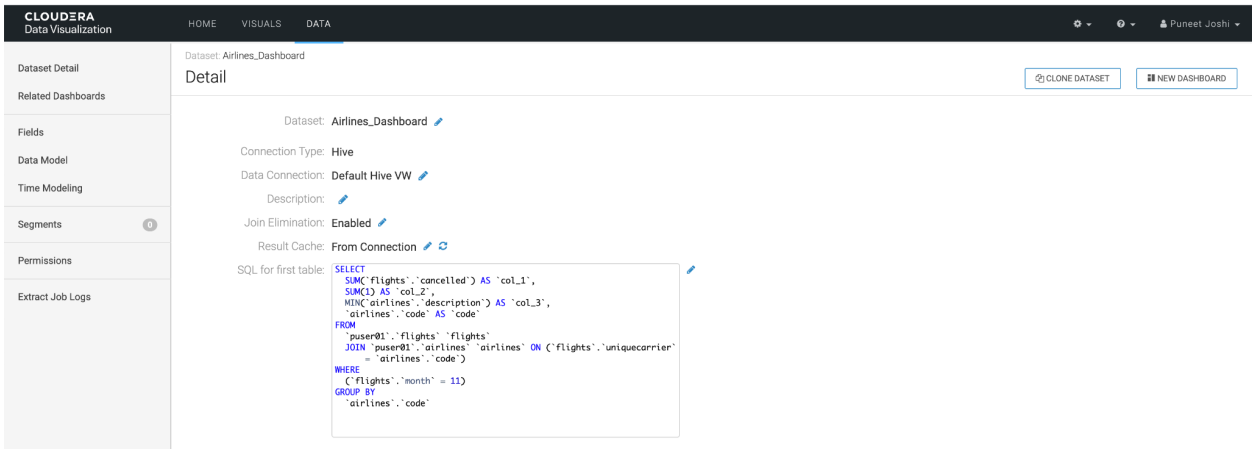
Step 3: Add the data set as per below screenshot.



Use the below SQL after replacing your username.

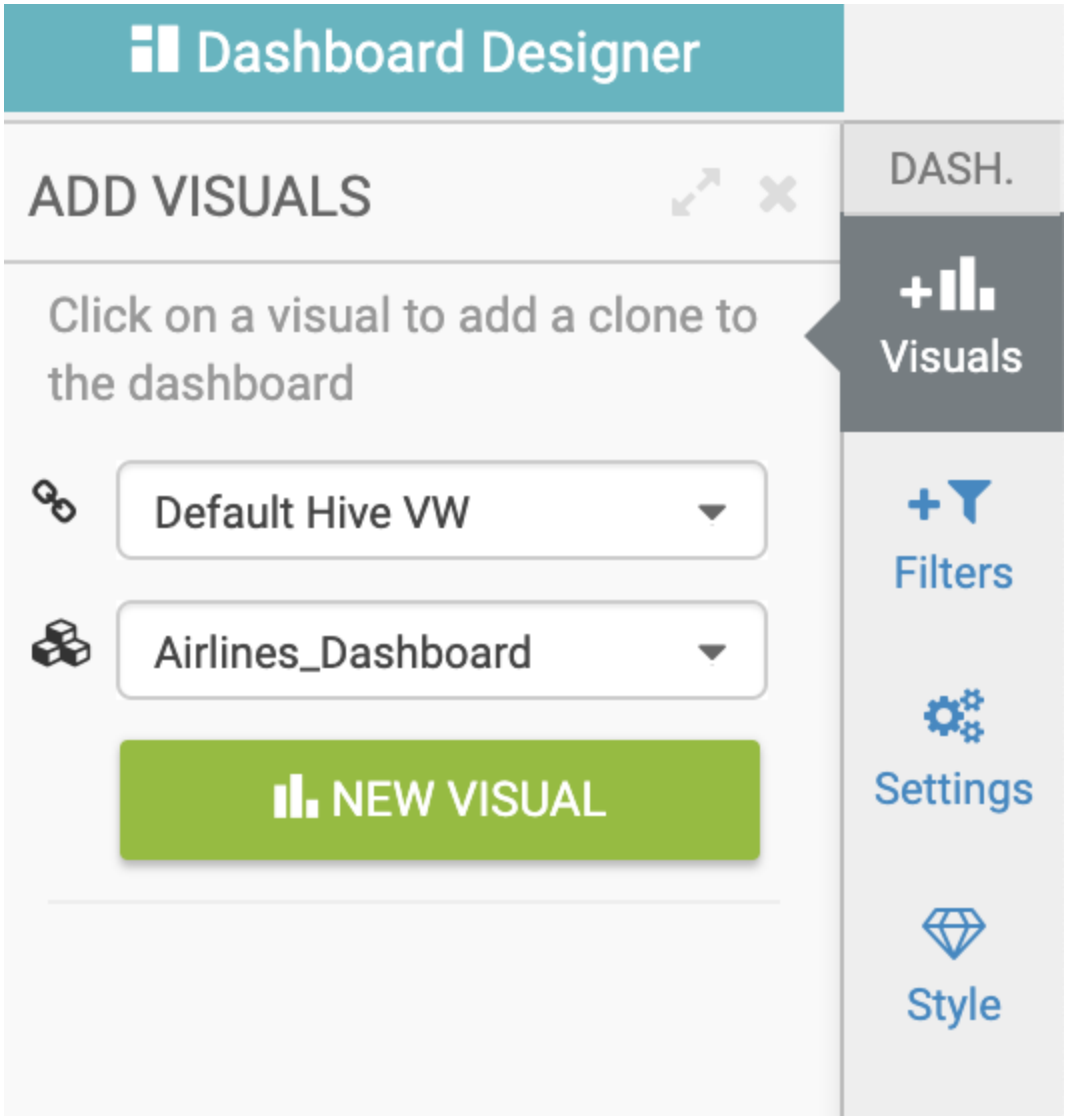
```
SELECT
  SUM(`flights`.`cancelled`) AS `col_1`,
  SUM(1) AS `col_2`,
  MIN(`airlines`.`description`) AS `col_3`,
  `airlines`.`code` AS `code`
FROM
  `username`.`flights` `flights`
  JOIN `username`.`airlines` `airlines` ON (`flights`.`uniquecarrier` =
`airlines`.`code`)
WHERE
  (`flights`.`month` = 11)
GROUP BY
  `airlines`.`code`;
```

Step 4: Once created click on the name of the newly created dataset to get the details page of the dataset.



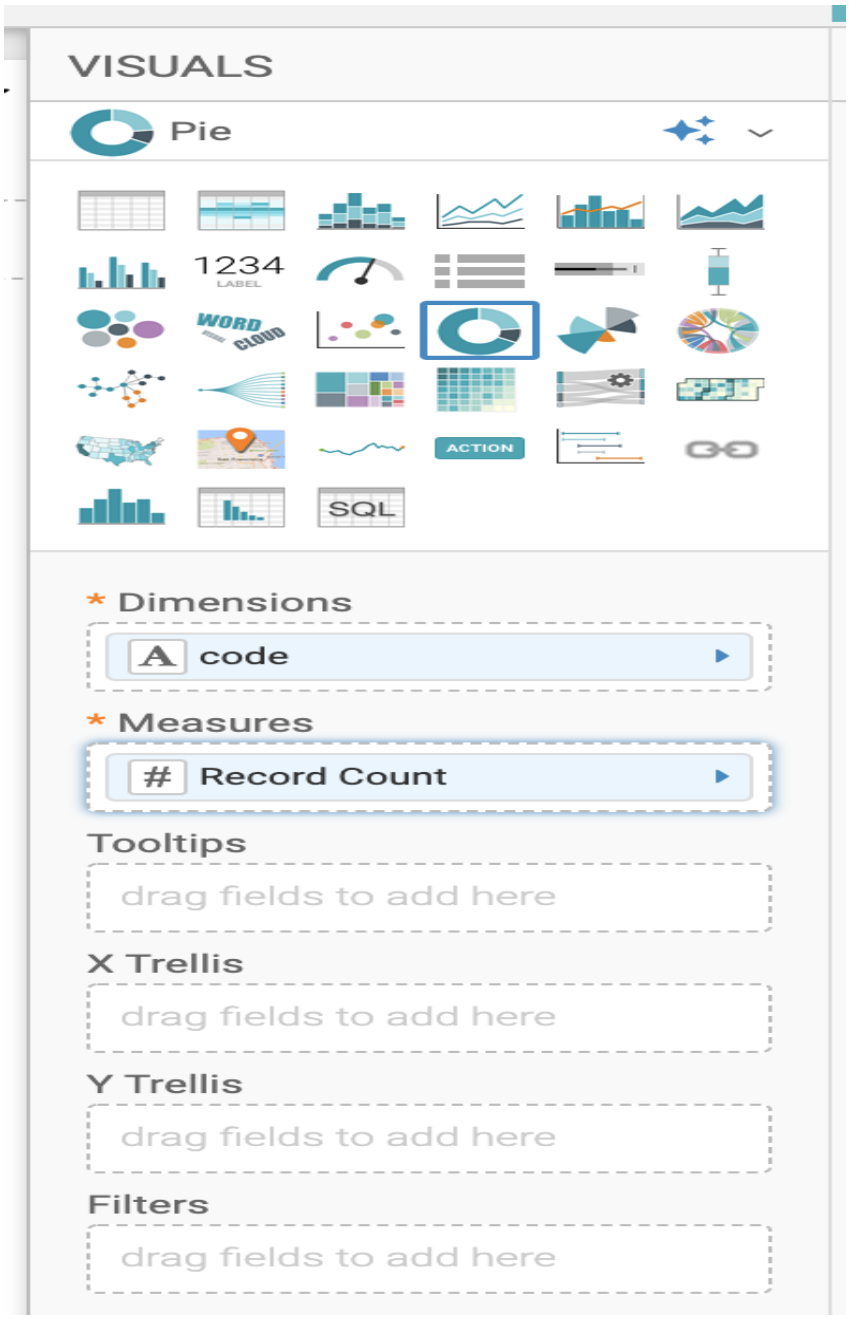
Step 5 :Click on  to create a dashboard from the output of the above SQL query.

Step 6 : After entering the name and sub title of the dashboard . Click on “Visuals” to add rich visualization in our dashboard.

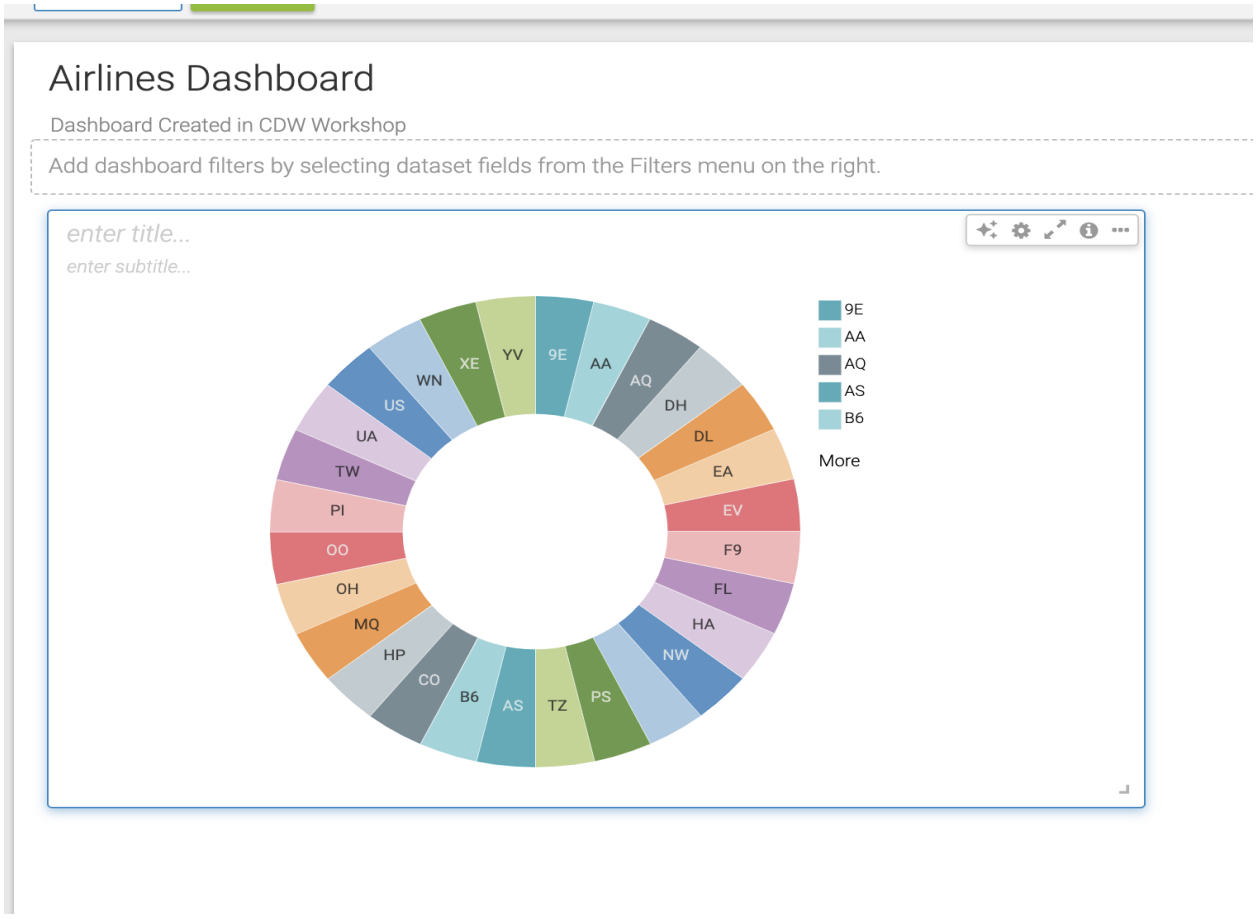


Add your desired visual to the dashboard from the available ones. For illustration purposes we are going with Pie Chart to display the output of our SQL.

Step 7 : Enter details as per below screenshot



Click on “Refresh Visual” to see output which will be similar to below.



Now you are aware how simple it is to create appealing dashboards . Spend some time on this interactive dashboard designer and try your hands on creating some impressive dashboards.

******This concludes Cloudera DataWarehouse Workshop Lab *******