

CDW Workshop Student Guide

Revision History	2
Introduction	3
Labs summary	3
Lab 0 - Initial setup	4
Lab 1 – Setup Virtual Warehouse	7
Lab 2 - Data Exploration	10
Lab 3 – Auto-Scaling, Auto-Suspend & Caching Cloudera DataWarehouse	18
Lab 4 – Security & Governance with Ranger & Atlas	27
Lab 5 – Data Visualization	41

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

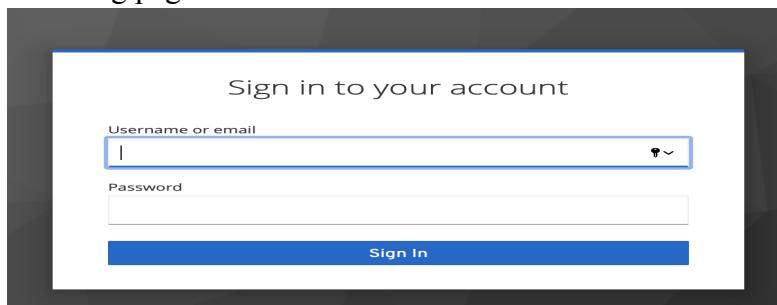
URL: Will be shared by the instructor

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.

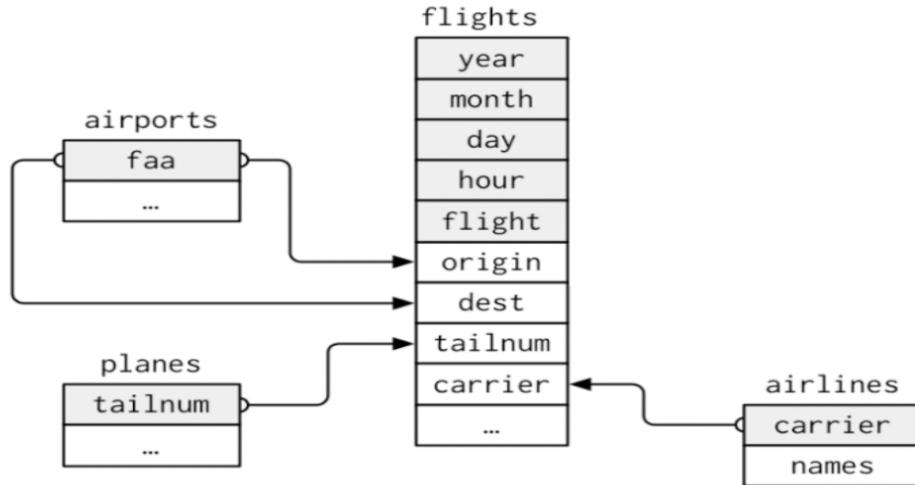


Dataset for the workshop

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 \Rightarrow iata

*carrier \Rightarrow 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.

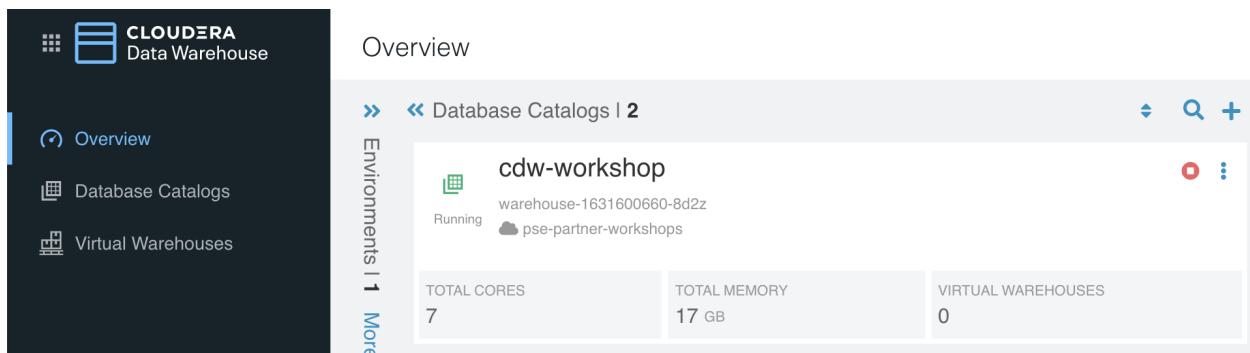
***** Screenshots of environment & Data Catalogue *****

The screenshot shows the Cloudera Data Warehouse interface. On the left, there is a sidebar with the following options: Overview (selected), Database Catalogs, and Virtual Warehouses. The main area is titled "Overview" and shows a list of "Database Catalogs | 1". There is one entry: "partner-workshop-dl-default" (warehouse-1631543867-259v). It is marked as "Running" and associated with "pse-partner-workshops". Below this, there are three summary metrics: "TOTAL CORES" (7), "TOTAL MEMORY" (17 GB), and "VIRTUAL WAREHOUSES" (0).

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.



The screenshot shows the Cloudera Data Warehouse interface. On the left, there's a sidebar with 'Overview', 'Database Catalogs', and 'Virtual Warehouses'. The main area is titled 'Overview' and shows a list of 'Database Catalogs'. One catalog, 'cdw-workshop', is listed with details: 'warehouse-1631600660-8d2z', 'Running', and 'pse-partner-workshops'. Below this, resource statistics are shown: 'TOTAL CORES' (7), 'TOTAL MEMORY' (17 GB), and 'VIRTUAL WAREHOUSES' (0). On the right, there are navigation icons (back, forward, search, plus) and a three-dot menu. A secondary screenshot below it shows the same interface but with a context menu open over the 'cdw-workshop' entry. The menu options are 'Clone', 'Edit' (which is highlighted in grey), 'Delete', and 'Upgrade'.

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.

The screenshot shows the Cloudera Data Warehouse interface for managing database catalogs. A specific catalog named 'cdw-workshop' is selected. The 'CONFIGURATIONS' tab is active, showing environment variables for the 'Das event processor' environment. Key configurations include:

KEY	VALUE
HADOOP_CLIENT_OPTS	-Dorg.wildfly.openssl.path=/lib64 \${HADOOP_CLIENT_OPTS}
JAVA_OPTS	-Djdk.tls.disabledAlgorithms=SSLv3,GCM -Xms4G -Xmx6554M -Xloggc:/var/log/das/das-event-processor-gc

The screenshot shows the Cloudera Data Warehouse interface for managing database catalogs. A specific catalog named 'cdw-workshop' is selected. The 'CONFIGURATIONS' tab is active, showing environment variables for the 'Metastore' environment under 'hadoop-core-site'. Key configurations include:

KEY	VALUE
appender.console.layout.appName	\$env:EDWS_SERVICE_NAME
appender.console.layout.exceptionPattern	%ex{full}
appender.console.layout.facility	USER
appender.console.layout.includeMDC	true
appender.console.layout.loggerFields.pairs1.key	level
appender.console.layout.loggerFields.pairs1.type	KeyValuePair
appender.console.layout.loggerFields.pairs1.value	%p
appender.console.layout.loggerFields.pairs2.key	thread
appender.console.layout.loggerFields.pairs2.type	KeyValuePair

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:

Virtual Warehouses | 0 🔍 +

- 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

New Virtual Warehouse

Name *

Type *

HIVE IMPALA

Database Catalog *

cdw-workshop

Tagging ⓘ

Only alphanumeric and _-@.: are allowed

Size *

xsmall - 2 Executor Nodes

Disable AutoSuspend

AutoSuspend Timeout (in seconds): 300

Concurrency Autoscaling ⓘ

Nodes: Min:2, Max:6

HEADROOM WAIT TIME

Desired Free Capacity: 1

Feedback

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

Virtual Warehouses

puser01-cdw (ID: compute-1631601390-n6kj)

ACTIONS **APPLY**

STATUS	VERSION	CREATED BY	SIZE	NODE COUNT	CORES	MEMORY	TYPE
Stopped	2021.0.1-b10	puneetjoshi	xsmall	0	11	48 GB	HIVE

ENVIRONMENT DATABASE CATALOG
pse-partner-workshops (ID: env-qvg4gn) cdw-workshop (ID: warehouse-1631600660-8d22z) [Open DAS](#)

SIZING AND SCALING CONFIGURATIONS DIAGNOSTIC BUNDLE EVENTS TIMELINE

Cluster Shape

Auto scale

Disable AutoSuspend

AutoSuspend Timeout (in seconds): 300

Concurrency Autoscaling

Nodes: Min:2, Max:6

HEADROOM **WAIT TIME**

Desired Free Capacity: 1

?

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**

Amazon S3 > pse-airline-demo1

pse-airline-demo1 [Info](#)

[Objects](#) [Properties](#) [Permissions](#) [Metrics](#) [Management](#) [Access Points](#)

Objects (2)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	airlines_new_orc.db/	Folder	-	-	-
<input type="checkbox"/>	airlines_new_parquet.db/	Folder	-	-	-

Amazon S3 > pse-airline-demo1 > airlines_new_orc.db/

airlines_new_orc.db/

[Copy S3 U](#)

[Objects](#) [Properties](#)

Objects (5)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	airlines/	Folder	-	-	-
<input type="checkbox"/>	airlines1/	Folder	-	-	-
<input type="checkbox"/>	airports/	Folder	-	-	-
<input type="checkbox"/>	flights/	Folder	-	-	-
<input type="checkbox"/>	planes/	Folder	-	-	-

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

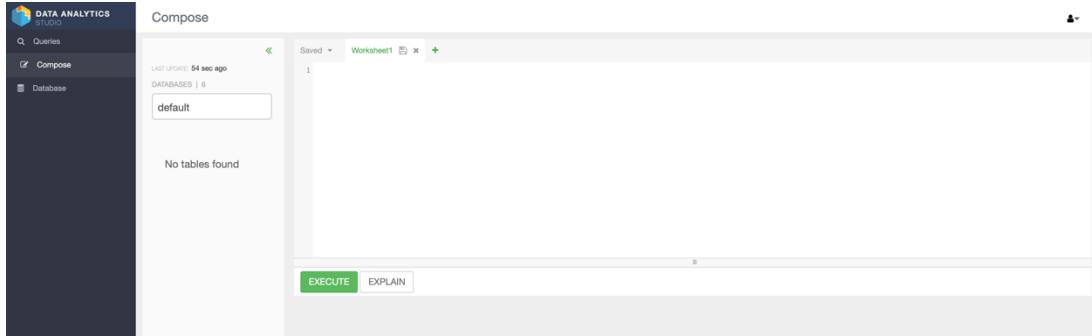
The screenshot shows the Amazon S3 console interface. The path is: Amazon S3 > pse-airline-demo1 > airlines_new_orc.db/ > airlines/. The 'Objects' tab is selected. There is one object listed: '000000_0'. The object was last modified on September 14, 2021, at 02:04:29 (UTC+05:30) and has a size of 15.8 KB. The storage class is Standard. Action buttons include Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, and Upload.

STEP 2 : Click on and “Open DAS”

The screenshot shows the Virtual Warehouses page in the AWS Glue console. A warehouse named 'puser01-cdw' is selected. The details shown are: compute-1631601390-n6kj, Stopped, and cdw-workshop. Below these details are metrics: NODE COUNT (0), TOTAL CORES (12), TOTAL MEMORY (56 GB), and TYPE (HIVE UNIFIIE). A context menu is open for the 'puser01-cdw' warehouse, listing options: Clone, Edit, Delete, Upgrade, Copy JDBC URL, Download JDBC Jar, Open DAS, Open HUE, Open Data Visualization, Set Compactor, Run AutoScaling Demo, Collect Diagnostic Bundle, and Feedback. The 'Open DAS' option is highlighted.

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

Compose

LAST UPDATED 21 sec ago

Saved Worksheet! +

DATABASES | 4

default

No tables found

```

1 Select * from puser01.airlines_ext
2 Select * from puser01.airports_ext
3 Select * from puser01.planes_ext
4 Select * from puser01.flights_ext limit 10;

```

EXECUTE EXPLAIN

Results

FLIGHTS_EXTMONTH	FLIGHTS_EXTDAYOFMONTH	FLIGHTS_EXTDAYOFWEEK	FLIGHTS_EXTDDEPTHME	FLIGHTS_EXTCRSDEPTIME	FLIGHTS_EXTRARITIME	FLIGHTS_EXTCRSARRTIME	FLIGHTS_EXTRUNIQUECARRIER	FLIGHTS_EXTFLIGHTNUM	FLIGHTS_E
10	14	3	741	730	912	849	PS	1451	NA
10	15	4	729	730	903	849	PS	1451	NA
10	17	6	741	730	918	849	PS	1451	NA
10	18	7	729	730	847	849	PS	1451	NA
10	19	1	749	730	922	849	PS	1451	NA
10	21	3	728	730	848	849	PS	1451	NA
10	22	4	728	730	852	849	PS	1451	NA
10	23	5	731	730	902	849	PS	1451	NA
10	24	6	744	730	908	849	PS	1451	NA
10	25	7	729	730	851	849	PS	1451	NA

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, crsarrrtime int, uniquecarrier string,
flightnum int, tailnum string, actualemapsedtime 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`

The screenshot shows a Data Analytics Studio interface. On the left, there's a sidebar with 'Compose' selected under 'Queries'. The main area is titled 'Compose' and shows a 'Worksheet' tab. The code editor contains the following four queries:

```

1: Select * from puser01.airlines
2: Select * from puser01.airports
3: Select * from puser01.planes
4: Select * from puser01.flights limit 10;

```

Below the code editor, it says 'No tables found'. Under the 'Results' section, there are two tabs: 'EXECUTE' (which is highlighted in green) and 'EXPLAIN'. The 'Results' table has 10 rows and 13 columns, corresponding to the columns listed in the fourth query. The columns are:

FLIGHTS.MONTH	FLIGHTS.DAYOFMONTH	FLIGHTS.DAYOFWEEK	FLIGHTS.DEPTIME	FLIGHTS.CRSDEPTIME	FLIGHTS.ARTIME	FLIGHTS.CRSARTIME	FLIGHTS.UNIQUECARRIER	FLIGHTS.FLIGHTNUM	FLIGHTS.TAILNUM	FLIGHTS.ACUTUALDEPASSTIME
1	1	1	1232	1225	1341	1340	WN	2891	N351	69
1	1	1	1918	1905	2043	2035	WN	462	N370	85
1	1	1	2206	2130	2334	2300	WN	1229	N685	88
1	1	1	1230	1200	1356	1330	WN	1355	N364	86
1	1	1	831	830	957	1000	WN	2278	N480	86
1	1	1	1430	1420	1553	1550	WN	2386	N611SW	83
1	1	1	1936	1840	2217	2130	WN	409	N482	101
1	1	1	944	935	1223	1225	WN	1131	N749SW	99
1	1	1	1537	1450	1819	1735	WN	1212	N451	102
1	1	1	1318	1315	1603	1610	WN	2456	N830WN	105

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.

QUERY	STATUS	QUEUE	USER	TABLES READ	TABLES WRITTEN	START TIME	DURATION	DAG ID	ACTIONS
<code>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);</code>	STARTED	None	puneetjoshi	flights (puser01), airlin...	Not Available	3 seconds ago	Not Available	Not Available	edit refresh more
<code>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);</code>	SUCCESS	None	puneetjoshi	flights (puser01), airlin...	Not Available	6 minutes ago	00:06:32	dag_163180781	edit refresh more

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.

```

1 SET hive.query.results.cache.enabled=false;
2 SELECT
3   SUM('flights'.`cancelled`) AS `col_1`,
4   SUM(1) AS `col_2`,
5   MIN(`airlines`.`description`) AS `col_3`,
6   `airlines`.`code` AS `code`
7 FROM
8   `puser01`.`flights` `flights`
9   JOIN `puser01`.`airlines` `airlines` ON (`flights`.`uniquecarrier` = `airlines`.`code`)
10 WHERE
11   `flights`.`month` = 12
12 GROUP BY
13   `airlines`.`code`;
14

```

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:

New Virtual Warehouse

Name *

Type *

HIVE IMPALA

Database Catalog *

cdw-workshop

Tagging ⓘ

Only alphanumeric and _-@.: are allowed

Size *

xsmall - 2 Executor Nodes

Disable AutoSuspend

AutoSuspend Timeout (in seconds): 300

Concurrency Autoscaling ⓘ

Nodes: Min:2, Max:6

HEADROOM	WAIT TIME
----------	-----------

Desired Free Capacity: 1

Feedback

STEP 1 : Auto-Suspended

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.

The screenshot shows the Cloudera Manager interface for managing Virtual Warehouses. At the top, it says "Virtual Warehouses | 1". Below that, there's a card for a warehouse named "partner-cdw". The card includes the following details:

- Icon: A small icon representing the warehouse.
- Name: partner-cdw
- Status: Stopped
- Compute ID: compute-1631552161-9wsc
- Dataset: partner-workshop-dl-default

Below the card, there are summary statistics:

NODE COUNT	TOTAL CORES	TOTAL MEMORY	TYPE
0	12	56 GB	HIVE UNIFIED ANALYTICS COMPACTOR

At the bottom right of the card, there are three icons: DAS, Data VIZ, and a more options icon (three dots). There is also a search and filter icon at the very top right of the page.

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

Virtual Warehouses

puser01-cdw

compute-1631601390-n6kj
cdw-workshop

Stopped

NODE COUNT: 0 TOTAL CORES: 12 TOTAL MEMORY: 56 GB TYPE: HIVE UNIFI

Clone Edit Delete Upgrade

Copy JDBC URL Download JDBC Jar

Open DAS ↗ Open HUE ↗ Open Data Visualization ↗

Set Compactor

Run AutoScaling Demo Collect Diagnostic Bundle

Feedback

This screenshot shows the Cloudera Manager interface for managing virtual warehouses. It displays a list of warehouses, with 'puser01-cdw' selected. A context menu is open over this entry, listing options like Clone, Edit, Delete, Upgrade, and various links to external tools. The 'Open DAS' option is highlighted.

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

Compose

LAST UPDATE: 11 sec ago

DATABASES | 6

puser01

TABLES | 11

Search Tables

- flights_ext (29)
- airports_ext (7)
- airlines_ext1 (2)
- airlines_ext2 (2)
- planes_ext (9)
- airlines (2)
- airports (7)
- planes (9)
- flights (29)
- airlines_ext (2)
- traffic_cancel...

This screenshot shows the 'Compose' DAS window. It lists the 'puser01' database and its 11 tables. The tables are listed in a hierarchical tree view, with some entries showing their count in parentheses.

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
```

The screenshot shows the 'Compose' query editor in DAS. The left sidebar lists 'Queries' and 'Compose'. Under 'Compose', there's a 'Database' section with 'puser01' selected. Below it is a 'TABLES' section with 11 entries, including 'flights_ext', 'airlines', 'airports', 'planes', and 'traffic_cancel...'. The main area is titled 'Compose' and contains a code editor with the following SQL query:

```
1 SELECT AVG(CAST(flights.depdelay AS DOUBLE)) AS avg_depdelay_ok, flights.year AS year
2 FROM flights flights
3 JOIN airlines airlines ON (flights.uniquecarrier = airlines.code)
4 WHERE (airlines.description = 'American Airlines Inc.')
5 GROUP BY flights.year
6
7
8
```

Below the code editor are 'EXECUTE' and 'EXPLAIN' buttons. The results section shows a single row:

AVG_DEPEND_DELAY_OK	YEAR
7.862321254420546	null

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`;
```

```

1 SELECT
2   SUM(`flights`.`cancelled`) AS `col_1`,
3   SUM(1) AS `col_2`,
4   MIN(`airlines`.`description`) AS `col_3`,
5   `airlines`.`code` AS `code`
6 FROM
7   `puser01`.`flights` `flights`
8   JOIN `puser01`.`airlines` `airlines` ON (`flights`.`uniquecarrier` = `airlines`.`code`)
9 WHERE
10  (`flights`.`month` = 12)
11 GROUP BY
12   `airlines`.`code`;
13
14

```

COL_1	COL_2	COL_3	CODE
3040	44354	Pinnacle Airlines Inc.	9E
28212	1290312	American Airlines Inc.	AA
170	13800	Aloha Airlines Inc.	AQ
1498	48113	Independence Air	DH
27445	1422017	Delta Air Lines Inc.	DL
1861	107825	Eastern Air Lines Inc.	EA
4267	138861	Atlantic Southeast Airlines	EV

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.

NODE COUNT	TOTAL CORES	TOTAL MEMORY	TYPE
2	38	292 GB	HIVE UNIFIED ANALYTICS COMPACTOR

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.

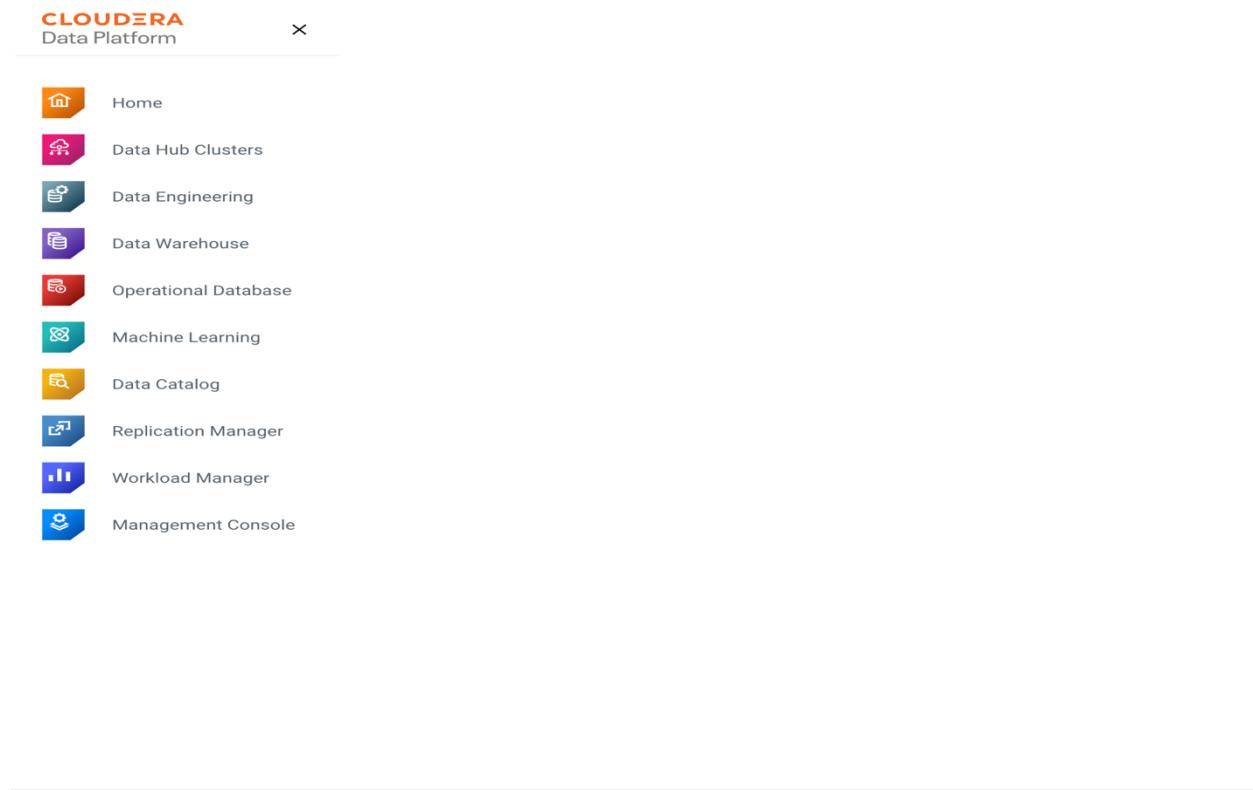
Lab 4 – Security & Governance with Ranger & Atlas

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

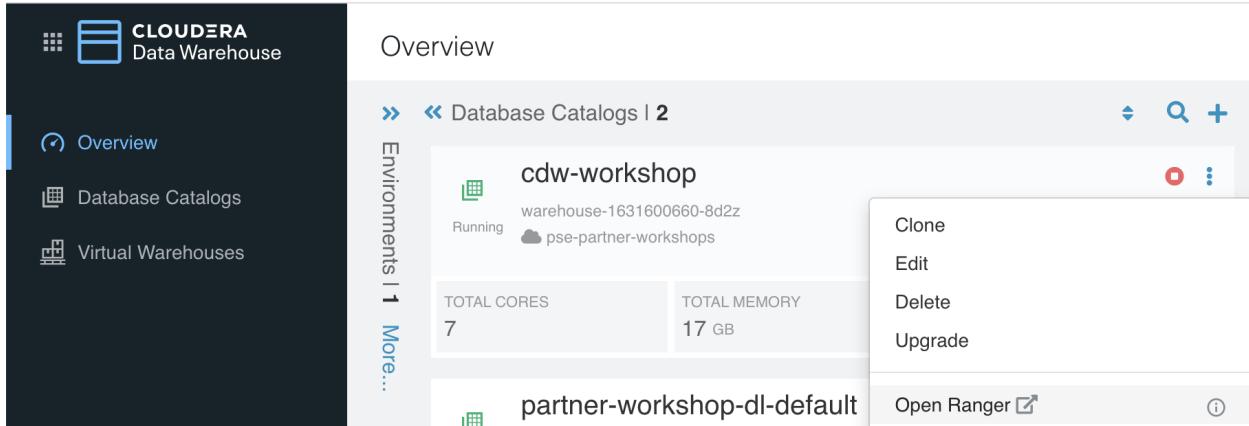
conditions such as time or geolocation can also be added to an existing policy rule. The Ranger authorization model is pluggable and can be easily extended to any data source using a service-based definition.

STEP 1 : Ranger Lab

Click on left top to choose DataWarehouse



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



The screenshot shows the Cloudera Data Warehouse interface. On the left sidebar, there are links for Overview, Database Catalogs, and Virtual Warehouses. The main area is titled 'Overview' and shows 'Database Catalogs | 2'. One catalog is listed: 'cdw-workshop' (warehouse-1631600660-8d2z). It is marked as 'Running' and associated with 'pse-partner-workshops'. Below this, resource statistics are shown: 'TOTAL CORES' (7) and 'TOTAL MEMORY' (17 GB). A sub-section titled 'partner-workshop-dl-default' is visible. On the right, a context menu is open for the 'cdw-workshop' catalog, with options like Clone, Edit, Delete, Upgrade, and 'Open Ranger' highlighted.

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



The screenshot shows the Ranger Service Manager interface. At the top, there are tabs for Ranger, Access Manager, Audit, Security Zone, and Settings. The user is puneetjoshi. The main area is titled 'Service Manager' and contains three sections: 'HDFS' (with 'cm_hdfs'), 'HBASE' (with 'cm_hbase'), and 'HADOOP SQL' (with 'Hadoop SQL' and 'dwx-warehouse-1631600660-8d2z_hive'). Each section has a set of icons for managing services.

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**

Policy ID	Policy Name	Policy Labels	Status	Audit Logging	Roles	Groups	Users	Action		
133	all - global	--	Enabled	Enabled	--	c_ranger_admins_59c582ca	hive rangerlookup impala			
134	all - database, table, column	--	Enabled	Enabled	--	c_ranger_admins_59c582ca	hive rangerlookup [OWNER] impala			
135	all - database, table	--	Enabled	Enabled	--	c_ranger_admins_59c582ca	hive rangerlookup [OWNER] impala			
136	all - database	--	Enabled	Enabled	--	c_ranger_admins_59c582ca public	hive rangerlookup [OWNER] impala			
137	all - hiveservice	--	Enabled	Enabled	--	c_ranger_admins_59c582ca	hive rangerlookup impala			
138	all - database, udf	--	Enabled	Enabled	--	c_ranger_admins_59c582ca	hive rangerlookup [OWNER] impala			
139	all - url	--	Enabled	Enabled	--	c_ranger_admins_59c582ca	hive rangerlookup impala			
140	default database tables columns	--	Enabled	Enabled	--	public	hive impala			
141	Information_schema database tables columns	--	Enabled	Enabled	--	public	hive impala			
142	airline_onetime_orc	--	Enabled	Enabled	--	public	--			
143	airline_onetime_parquet	--	Enabled	Enabled	--	public	--			

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****

Create Policy

Policy Details :

Policy Type: **Access**

Policy Name *: tailnum_restriction

Policy Label:

enabled normal

database *: puser01

table *: flights

column *: tailnum

Description:

Audit Logging: YES

Add necessary permission as per below screen-shot

Select Role	Select Group	Select User	Permissions	Delegate Admin	
<input type="button" value="Select Roles"/>	<input type="button" value="Select Groups"/>	<input type="text" value="puser01"/>	<input type="button" value="Add Permissions +"/> 	<input type="checkbox"/>	

add/edit permissions

select
 update
 Create
 Drop
 Alter
 Index
 Lock
 All
 Read
 Write
 ReplAdmin
 Service Admin
 Temporary UDF Admin
 Refresh
 Select/Deselect All

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

The screenshot shows a DAS Worksheet titled "Worksheet1". The code entered is "select * from flights". Below the code are two buttons: "EXECUTE" (green) and "EXPLAIN" (grey). A pink error message box is displayed, containing an exclamation icon, the word "Error", and the text "Error while compiling statement: FAILED: HiveAccessControlException Permission denied: user [vrayker] does not have [SELECT] privilege on [puser01/flights/*]". There is also a "More details" link.

STEP 3 : Creating Masking policy

Click on Masking tab and Add New Policy

The screenshot shows the Ranger Access Manager interface. The top navigation bar includes "Ranger", "Access Manager", "Audit", "Security Zone", "Settings", and a user icon "vrayker". Below the navigation is a breadcrumb path: "Service Manager > dwx-warehouse-1592811147-mn4b_hive Policies". Underneath the path are three tabs: "Access" (disabled), "Masking" (selected and highlighted with a red circle), and "Row Level Filter". A search bar at the top says "List of Policies : dwx-warehouse-1592811147-mn4b_hive". Below the search bar is a table header with columns: "Policy ID", "Policy Name", "Policy Labels", "Status", "Audit Logging", "Roles", "Groups", "Users", and "Action". A blue button labeled "Add New Policy" is located in the "Action" column of the first row of the table, and it is also highlighted with a red circle. The table body contains the message "No Policies found!".

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****

Policy Details :

Policy Type	Masking	Add Validity Period
Policy ID	196	
Policy Name *	Airport_Masking	0
	enabled	normal
Policy Label	Policy Label	
Hive Database *	X puser01	
Hive Table *	X airports	
Hive Column *	X lon	
Description		
Audit Logging	YES	

Mask Conditions :

Select Role	Select Group	Select User	Access Types	Select Masking Option	
Select Roles	Select Groups	X vrayker	select	Redact	X

Save the policy.

Now, go back to DAS and run the query and the masking live in action

Select * from airports

```
3 select * from airports  
4  
5
```

EXECUTE EXPLAIN

Results

AIRPORTS.IATA	AIRPORTS.AIRPORT	AIRPORTS.CITY	AIRPORTS.STATE	AIRPORTS.COUNTRY	AIRPORTS.LAT	AIRPORTS.LON
00M	Thigpen	Bay Springs	null	USA	31.95376472	null
00R	Livingston Municipal	Livingston	null	USA	30.68586111	null
00V	Meadow Lake	Colorado Springs	null	USA	38.94574889	null
01G	Perry-Warsaw	Perry	null	USA	42.74134667	null
01J	Hilliard Airpark	Hilliard	null	USA	30.6880125	null
01M	Tishomingo County	Belmont	null	USA	34.49166667	null
02A	Gragg-Wade	Clanton	null	USA	32.85048667	null
02C	Capitol	Brookfield	null	USA	43.08751	null
02G	Columbiana County	East Liverpool	null	USA	40.67331278	null
02P	Kennedia Memorial	Kennedia	null	USA	40.44795889	null

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

The screenshot shows the Cloudera Data Warehouse Overview page. On the left sidebar, there are links for Overview, Database Catalogs, and Virtual Warehouses. The main area displays 'Database Catalogs' with two entries: 'cdw-workshop' and 'partner-workshop-dl-default'. The 'cdw-workshop' entry is highlighted. A context menu is open over this entry, showing options: Clone, Edit, Delete, Upgrade, Open Ranger, and Open Atlas.

Name	Status	Type
cdw-workshop	Running	warehouse-1631600660-8d2z pse-partner-workshops
partner-workshop-dl-default	Running	warehouse-1631543867-259v pse-partner-workshops

This opens-up Atlas page for you

The screenshot shows the Apache Atlas search interface. At the top, there are tabs for SEARCH, CLASSIFICATION, and GLOSSARY. Below the tabs, there are four search input fields: 'Search By Type' (set to '_ALL_ENTITY_TYPES'), 'Search By Classification' (dropdown menu), 'Search By Term' (dropdown menu), and 'Search By Text' (text input field). There are also 'Clear' and 'Search' buttons. On the right side, there is a section titled 'Favorite Searches' with 'Save' and 'Save As' buttons, and a message stating 'You don't have any favorite search.'

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

The screenshot shows the Apache Atlas search interface. At the top, there are navigation links for 'SEARCH', 'CLASSIFICATION', and 'GLOSSARY'. Below these are two search modes: 'Basic' (selected) and 'Advanced'. A search icon is also present. The main search area has a 'Search By Type' section with a dropdown menu. The menu lists 'hive_table (99)' at the top, followed by 'hive_table' (which is highlighted with a blue border), 'hive_table (99)', 'hive_table_ddl (78)', and a 'Search Term' input field. Below this is a 'Search By Text' section with a 'Search by text' input field, a 'Clear' button, and a 'Search' button.

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

The screenshot shows a table of search results. At the top, it says 'Showing 25 records From 1 - 25'. There are checkboxes for selecting rows. The table has six columns: 'Name', 'Owner', 'Description', 'Type', 'Classifications', and 'Term'. Each row represents a different Hive table, with icons corresponding to the owner's name. The 'Type' column for all rows is 'hive_table'. The 'Classifications' and 'Term' columns each have a '+' icon. The table includes a header row with column names and a footer row with checkboxes for 'Exclude sub-types', 'Exclude sub-classifications', and 'Show historical entities', along with a 'Columns' dropdown.

	Name	Owner	Description	Type	Classifications	Term
<input type="checkbox"/>	test1	rjamsal		hive_table	+	+
<input type="checkbox"/>	flights	chris		hive_table	+	+
<input type="checkbox"/>	flights	chris		hive_table	+	+
<input type="checkbox"/>	employees_encrypted	chris		hive_table	+	+
<input type="checkbox"/>	t1	arawat		hive_table	+	+
<input type="checkbox"/>	t1	arawat		hive_table	+	+
<input type="checkbox"/>	flights1a	ss		hive_table	+	+
<input type="checkbox"/>	flights			hive_table	+	+
<input type="checkbox"/>	ww_customers	chris		hive_table	+	+

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

The screenshot shows a table of search results, similar to the one above, but with a circled '3' at the bottom right of the page navigation area. The columns are the same: Name, Owner, Description, Type, Classifications, and Term. The 'Type' column for all rows is 'hive_table'. The 'Classifications' and 'Term' columns each have a '+' icon. The table includes a header row with column names and a footer row with checkboxes for 'Exclude sub-types', 'Exclude sub-classifications', and 'Show historical entities', along with a 'Columns' dropdown.

	Name	Owner	Description	Type	Classifications	Term
<input type="checkbox"/>	column_privileges	hive		hive_table	+	+
<input type="checkbox"/>	tables	hive		hive_table	+	+
<input type="checkbox"/>	table_privileges	hive		hive_table	+	+
<input type="checkbox"/>	views	hive		hive_table	+	+
<input type="checkbox"/>	scheduled_executions	hive		hive_table	+	+

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

	Name	Owner	Description	Type	Classifications	Term
<input type="checkbox"/>	scheduled_queries	hive		hive_table	+	+
<input type="checkbox"/>	puser01.flights_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	flights	vrayker		hive_table	+	+
<input type="checkbox"/>	planes	vrayker		hive_table	+	+
<input type="checkbox"/>	puser01.planes_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	puser01.airports_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	airports	vrayker		hive_table	+	+
<input type="checkbox"/>	airlines	vrayker		hive_table	+	+
<input type="checkbox"/>	puser01.airlines_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	traffic_cancel_airport	vrayker		hive_table	+	+

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

	Name	Owner	Description	Type	Classifications	Term
<input type="checkbox"/>	scheduled_queries	hive		hive_table	+	+
<input type="checkbox"/>	puser01.flights_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	flights	vrayker		hive_table	+	+
<input type="checkbox"/>	planes	vrayker		hive_table	+	+
<input type="checkbox"/>	puser01.planes_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	puser01.airports_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	airports	vrayker		hive_table	+	+
<input type="checkbox"/>	airlines	vrayker		hive_table	+	+
<input type="checkbox"/>	puser01.airlines_ext@dwx-warehouse-1592811147-mn4b			hive_table	+	+
<input type="checkbox"/>	traffic_cancel_airport	vrayker		hive_table	+	+

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

traffic_cancel_airport (hive_table)

Classifications: [+](#)

Term: [+](#)

[Properties](#) [Lineage](#) [Relationships](#) [Classifications](#) [Audits](#) [Schema](#)

[▼ Technical properties](#) [Toggle](#)

columns (5)	cancelled description code month year
createTime	Fri Jul 17 2020 13:47:59 GMT+0530 (India Standard Time)
db	puser01
lastAccessTime	Fri Jul 17 2020 13:47:59 GMT+0530 (India Standard Time)
name	traffic_cancel_airport
owner	vrayker
parameters	{ totalSize: "5975042", transactional_properties: "default", numFiles: "22", transient_lastDdlTime: "1594973879", bucketing_version: "2", }
qualifiedName	puser01.traffic_cancel_airport@dwx-warehouse-1592811147-mn4b
retention	0
sd	puser01.traffic_cancel_airport@dwx-warehouse-1592811147-mn4b_storage
tableType	MATERIALIZED_VIEW
temporary	false
viewExpandedText	SELECT `flights`.`cancelled`, `airlines`.`description`, `airlines`.`code` AS `code`, `flights`.`month` AS `month`, `flights`.`year` AS `year` FROM `puser01`.`flights` JOIN `puser01`.`airlines` ON (`flights`.`uniquecarrier` = `airlines`.`code`)
viewOriginalText	SELECT flights.cancelled, airlines.description, airlines.code AS code, flights.month AS month, flights.year AS year FROM puser01.flights flights JOIN puser01.airlines airlines ON (flights.uniquecarrier = airlines.code)

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

Showing 25 records From 1 - 25					
	Name	Owner	Description	Type	Classifications
<input type="checkbox"/>	test1	njamsal		hive_table	+
<input type="checkbox"/>	flights	chris		hive_table	+
<input checked="" type="checkbox"/>	flights	chris		hive_table	+
<input type="checkbox"/>	employees_encrypted	chris		hive_table	+
<input type="checkbox"/>	t1	arawat		hive_table	+
<input type="checkbox"/>	t1	arawat		hive_table	+
<input type="checkbox"/>	flights1a	ss		hive_table	+
<input type="checkbox"/>	flights			hive_table	+
<input type="checkbox"/>	ww_customers	chris		hive_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

 flights (hive_table)

Classifications: [+](#)

Term: [+](#)

Properties [Lineage](#) Relationships Classifications Audits Schema

Current Entity In Progress Lineage Impact



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.

The screenshot shows the 'Virtual Warehouses' page with one entry. The entry is for the 'cdw-workshop-default' warehouse, which is currently stopped. It has 12 total cores and 56 GB of total memory. The type is listed as 'HIVE' (highlighted in yellow), 'UNIFIED ANALYTICS', and 'COMPACTOR'. There are buttons for 'DAS' and 'Data VIZ' (highlighted in blue), and a three-dot menu icon.

NODE COUNT	TOTAL CORES	TOTAL MEMORY	TYPE
0	12	56 GB	HIVE UNIFIED ANALYTICS COMPACTOR

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.

The screenshot shows the Cloudera Data Visualization interface. At the top, there's a navigation bar with 'CLOUDERA Data Visualization' on the left, and 'HOME', 'VISUALS', and 'DATA' tabs. On the right, it shows the user 'Puneet Joshi' and some system status icons. Below the navigation bar, there's a 'Get Started' section with four cards: 'DASHBOARDS' (16), 'APPS' (1), 'DATASETS' (12), and 'TOTAL VIEWS' (1). To the right of this is a sidebar with a teal header containing 'NEW DASHBOARD' and 'NEW APP' buttons. Below the sidebar, a message says 'Over the last 7 days...' followed by three items: '16 Dashboards were created', '10 Apps were created', and '12 Datasets were created'. A 'Get Started' button is also in this sidebar. The main content area has sections for 'Last Viewed by You' (showing a thumbnail of 'Iris species w/ images'), 'Overall Most Popular' (also showing a thumbnail of 'Iris species w/ images'), and 'Sample Dashboards' (listing several thumbnails of various dashboards). At the bottom right of the main content area, it says 'Your last login was on: Sep. 14, 2021 at 02:44 PM'.

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.

The screenshot shows the Cloudera Data Visualization web interface. At the top, there's a navigation bar with links for HOME, VISUALS, and DATA. A search bar is also at the top. On the left, there's a sidebar with 'All Connections' and a section for 'Default Hive VV' containing 'samples'. The main area is titled 'Connection Explorer'. It has tabs for 'Datasets' (which is active) and 'Connection Explorer'. Below the tabs is a table with columns: Title/Table, Created, Last Updated, Modified By, and # Visuals. The table currently displays 'No data'. At the bottom of the table area, there are three buttons: 'NEW DATASET', 'ADD DATA', and '...'. The 'NEW DATASET' button is highlighted with a blue border.

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

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

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 *

Airlines_Dashboard

Dataset Source

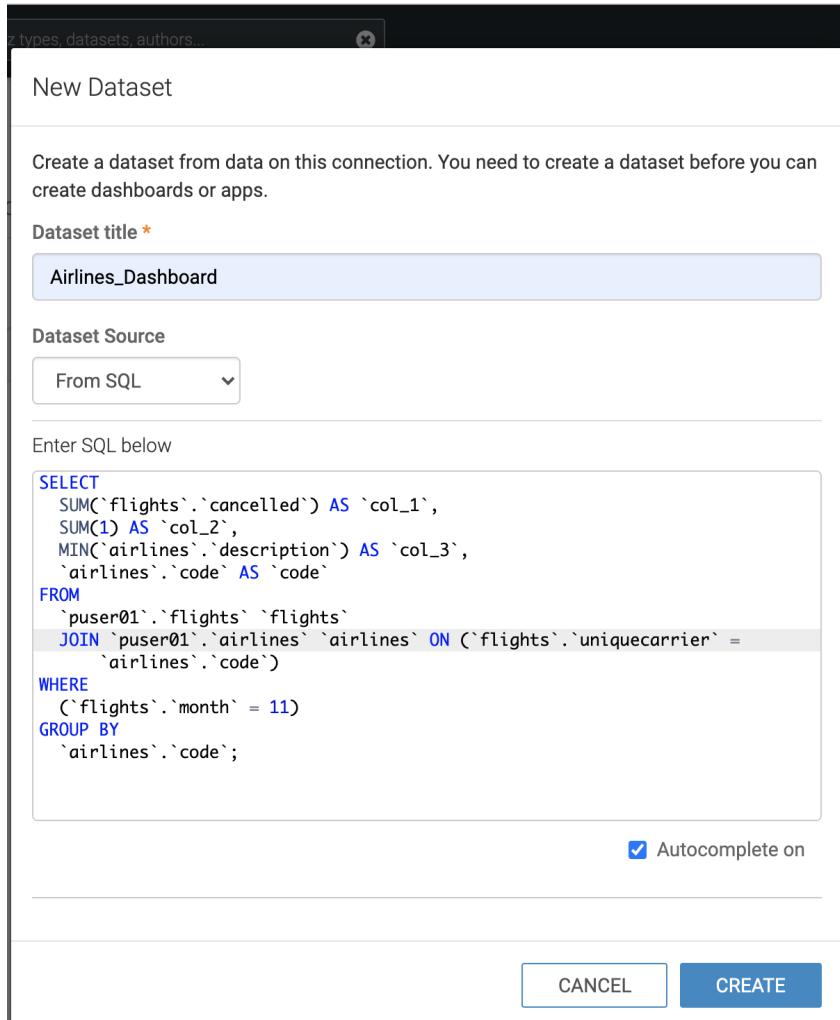
From SQL

Enter SQL below

```
SELECT
    SUM(`flights`.`cancelled`) AS `col_1`,
    SUM(1) AS `col_2`,
    MIN(`airlines`.`description`) AS `col_3`,
    `airlines`.`code` AS `code`
FROM
    `puser01`.`flights` `flights`
JOIN `puser01`.`airlines` `airlines` ON (`flights`.`uniquecarrier` =
    `airlines`.`code`)
WHERE
    (`flights`.`month` = 11)
GROUP BY
    `airlines`.`code`;
```

Autocomplete on

CANCEL CREATE



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.

The screenshot shows the Cloudera Data Visualization interface. The top navigation bar includes 'HOME', 'VISUALS', 'DATA', and a user profile 'Puneet Joshi'. The left sidebar has sections for 'Dataset Detail', 'Related Dashboards', 'Fields', 'Data Model', 'Time Modeling', 'Segments', and 'Permissions', along with an 'Extract Job Logs' section. The main content area is titled 'Dataset: Airlines_Dashboard' and 'Detail'. It displays configuration details: 'Connection Type: Hive', 'Data Connection: Default Hive VW', 'Description: ', 'Join Elimination: Enabled', and 'Result Cache: From Connection'. Below this is a SQL query editor window containing the following code:

```
SELECT
    SUM(`flights`.`cancelled`) AS `col_1`,
    SUM(1) AS `col_2`,
    MIN(`airlines`.`description`) AS `col_3`,
    `airlines`.`code` AS `code`
FROM
    `puser01`.`flights` `flights`
JOIN `puser01`.`airlines` `airlines` ON (`flights`.`uniquecarrier` = `airlines`.`code`)
WHERE
    `flights`.`month` = 11
GROUP BY
    `airlines`.`code`
```

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.

Dashboard Designer

ADD VISUALS

Click on a visual to add a clone to the dashboard



Default Hive VW



Airlines_Dashboard

NEW VISUAL

DASH.

Visuals

Filters



Settings



Style

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.

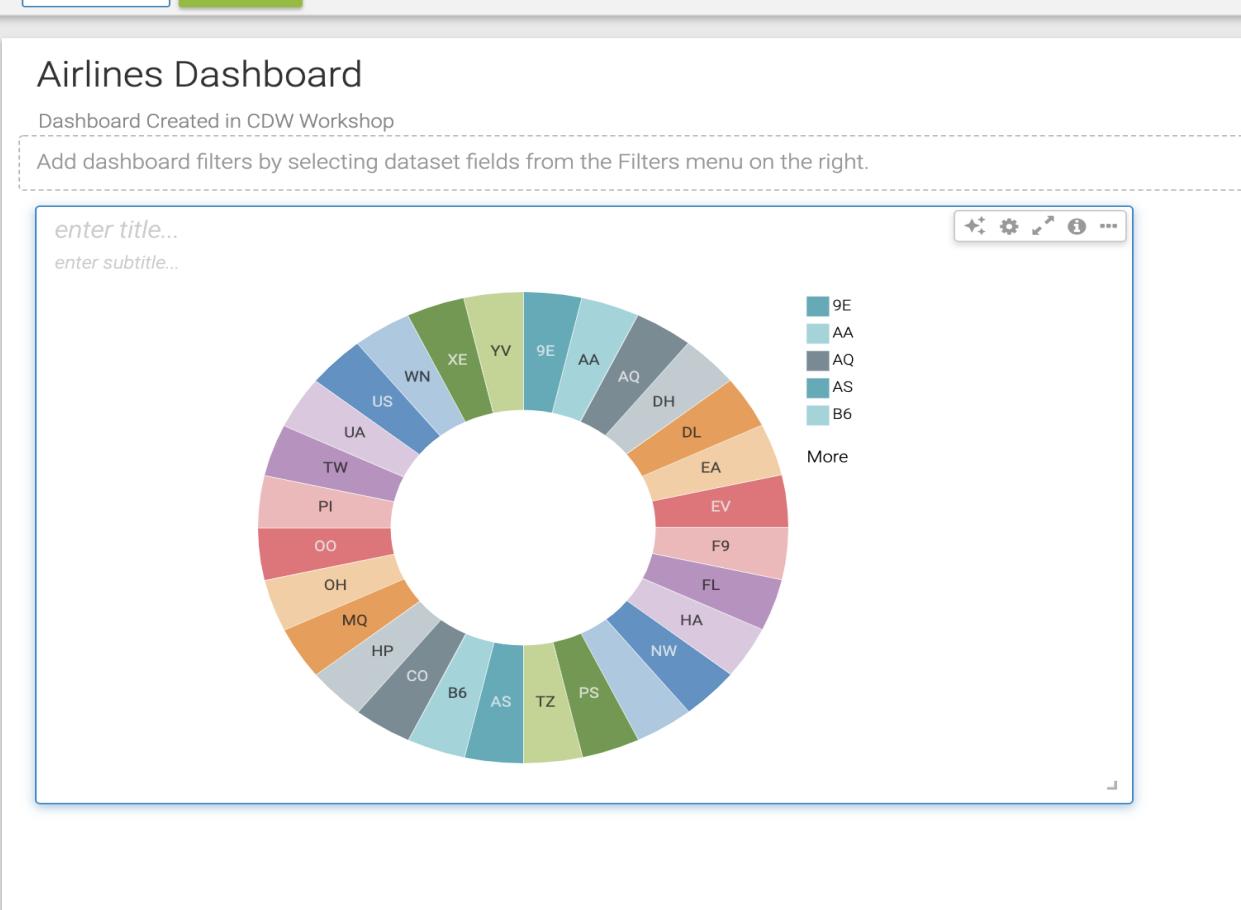
Step 7 : Enter details as per below screenshot

The screenshot shows the 'VISUALS' pane in Tableau. At the top, there is a 'Pie' icon and a dropdown menu. Below the header, there is a grid of icons representing different visual types: Bar, Line, Stacked Bar, Gantt, Map, Word Cloud, Scatter, Grid, Treemap, Network, Funnel, Heatmap, Gauge, Timeline, and Action. A 'WORD CLOUD' icon is highlighted with a blue border.

Below the grid, there are several configuration sections:

- * Dimensions:** A field containing 'A code' with a right-pointing arrow.
- * Measures:** A field containing '# Record Count' with a right-pointing arrow.
- Tooltips:** A dashed box with the placeholder text 'drag fields to add here'.
- X Trellis:** A dashed box with the placeholder text 'drag fields to add here'.
- Y Trellis:** A dashed box with the placeholder text 'drag fields to add here'.
- Filters:** A dashed box with the placeholder text 'drag fields to add here'.

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 *****