Lab # : 2; Lab Name : An Introduction to Hive ; Subject Name : Information Storage and Retrieval; Week #: 2; Lab Duration : 20 to 30 mins

# 

# **Intro**

In this lab, we will go over various Hive commands, which are useful for managing data in Hive. We will go over the following features:

* Load a data file into a Hive table
* Create a table using RC File Format
* Query a table using HQL
* Managed tables vs external tables
* Create a table using ORC File Format
* Partitioning a Table
* Bucketing a Table

# 

Here are a few points to get to know Hive :

* Hive data warehouse software facilitates loading, querying and transforming data on top of mainly Hadoop.
* Hive is used for ETL processing.
* Hive is very much SQL like.
* Hive query executes MapReduce processes on a hadoop cluster.
* Hive store it’s meta information in it’s metastore.
* Hive enables ad-hoc querying
* Hive works great for batch processing
* Latency for Hive queries is usually high

# **Let’s go!**

# **Step-1. Load a data file into a Hive table**

As hive stores data in HDFS, loading data into a Hive table means loading data files into HDFS and mapping the Hive table definition to the files’ content.

On your server, you can type “hive” and you will go to hive prompt as hive is already available on your instance.

Now, let’s create a table as follows:

# 

# CREATE TABLE Web\_Session\_Log

# (DATETIME varchar(500),

# USERID varchar(500),

# SESSIONID varchar(500),

# PRODUCTID varchar(500),

# REFERERURL varchar(500))

# row format delimited

# fields terminated by '\t'

# **stored as textfile;**

# Now let’s load some data into the table.

# From the github code, pull the data file, named, ‘weblog.csv’ into one of your folders. After that run the following command to load data into the above Hive table.

# LOAD DATA INPATH '/mnt/weblog.csv'

# OVERWRITE INTO TABLE Web\_Session\_Log;

The above command loaded the data file which is stored in your local unix directory into the hive table.

# **Step-2. Create a table using RC(Record Columnar) file format**

# As Hive reads datafiles stored in Hadoop, to optimize the storage of data format, there are various ways. RC File format is one format which has been very much used in Hive warehouses. RC File format is a structure which is a combination of data storage format, data compression approach, and optimization techniques for data reading. RC File format has the advantage of row oriented structure and column oriented structure both. Basically, you can slice and dice and table horizontally and vertically for storing. This gives the maximum flexibility to design your table based on access patterns.

# 

# 

# 

# Now, let’s create a table with RC File Format:

# CREATE TABLE Web\_Session\_Log\_RC

# (DATETIME varchar(500),

# USERID varchar(500),

# SESSIONID varchar(500),

# PRODUCTID varchar(500),

# REFERERURL varchar(500))

# row format delimited

# fields terminated by '\t'

# **STORED AS RCFILE;**

# 

# Let’s load data from the previously built Hive Table which was in Text File format.

# 

# INSERT OVERWRITE TABLE Web\_Session\_Log\_ORC

# select \* from Web\_Session\_Log;

# **Step-3. Query a tables**

# **SELECT DISTINCT SESSIONID,count(\*) as count from Web\_Session\_Log\_ORC GROUP BY SESSIONID ORDER BY count;**

# **Step-4. Managed tables vs External tables**

# 

# **Managed:**

# 

# **CREATE TABLE Web\_Session\_Log\_Managed(DATETIME varchar(500), USERID varchar(500), SESSIONID varchar(500), PRODUCTID varchar(500), REFERERURL varchar(500))**

# **row format delimited**

# **fields terminated by '\t'**

# **stored as textfile;**

# 

# **External:**

# 

# **CREATE EXTERNAL TABLE IF NOT EXIST Web\_Session\_Log\_External(DATETIME varchar(500), USERID varchar(500), SESSIONID varchar(500), PRODUCTID varchar(500), REFERERURL varchar(500))**

# **row format delimited**

# **fields terminated by '\t'**

# **stored as textfile;**

# 

# **Note:**

# **When you drop a Managed table, it deletes the data as well alongwith metadata.**

# **When you drop an External table, it only deletes the metadata.**

# **Step-5. ORC format**

# **CREATE TABLE ORCFileFormatExample(**

# **DATETIME varchar(500), USERID varchar(500), SESSIONID varchar(500), PRODUCTID varchar(500), REFERERURL varchar(500))**

# **COMMENT 'This is the Web Session Log data'**

# **ROW FORMAT DELIMITED**

# **FIELDS TERMINATED BY '\t'**

# **STORED AS ORC tblproperties ("orc.compress"="GLIB");**

# **Note:**

# **Optimized Row Columnar (ORC) File format**

# **It provides efficient way to store Hive data and**

# **Improves performance when Hive is reading, writing, and processing data**

# 

# **Step-6.PARTITIONED Table**

# **CREATE TABLE Web\_Session\_Log\_Partitioned(**

# **DATETIME varchar(500), USERID varchar(500), SESSIONID varchar(500), PRODUCTID varchar(500), REFERERURL varchar(500))**

# **COMMENT 'This is the Twitter streaming data'**

# **PARTITIONED BY(DATETIME STRING)**

# **ROW FORMAT DELIMITED**

# **FIELDS TERMINATED BY '\t'**

# **STORED AS TEXTFILE;**

# 

# **FROM Web\_Session\_Log**

# **INSERT OVERWRITE TABLE Web\_Session\_Log\_Partitioned PARTITION (DATETIME="2014-01-02 00:00:06 GMT") SELECT \* where PRODUCTID='/product/MT65XF2YA' limit 100;**

# **Note:**

# **Partitions are horizontal slices of data which allow large sets of data to be segmented into more manageable blocks. Partioning creates Folder at HDFS level.**

# 

# **Step-7. Bucketing a table**

# **CREATE TABLE Web\_Session\_Log\_Bucketing(DATETIME varchar(500), USERID varchar(500), SESSIONID varchar(500), PRODUCTID varchar(500), REFERERURL varchar(500))**

# **COMMENT 'This is the Web Session Log data'**

# **PARTITIONED BY( PRODUCTID STRING)**

# **CLUSTERED BY(USERID) INTO 2 BUCKETS**

# **ROW FORMAT DELIMITED**

# **FIELDS TERMINATED BY '\t'**

# **STORED AS TEXTFILE;**

# 

# **set hive.enforce.bucketing = true;**

# 

# **FROM Web\_Session\_Log**

# **INSERT OVERWRITE TABLE Web\_Session\_Log\_Bucketing PARTITION (PRODUCTID="/product/MT65XF2YA")**

# **SELECT \* where PRODUCTID='/product/MT65XF2YA' limit 100;**

# 

# **Note :**

# **Bucketing is a technique that allows you to cluster or segment large sets of data to optimize query performance**

# 