Big Data Tutorial 2: Hive

**What is Apache Hive?**

Apache Hive is a Data Warehouse software that facilitates querying and managing large datasets residing in a distributed storage (Example: HDFS). Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. The Hive Query Language (HiveQL or HQL) for MapReduce to process structured data using Hive.

It also provides:
- Tools to enable easy data extract/transform/load (ETL)
- A mechanism to impose structure on a variety of data formats
- Access to files stored either directly in Apache HDFS or in other data storage systems such as Apache HBase
- Query execution via MapReduce.

**What is Hadoop?**

Hadoop is an open-source software framework for storing and processing big data in a distributed/parallel fashion on large clusters of commodity hardware. Essentially, it accomplishes two tasks: massive data storage and faster processing. The core Hadoop consists of HDFS and Hadoop's implementation of MapReduce.

**What is HDFS?**

HDFS stands for Hadoop Distributed File System. HDFS is a highly fault-tolerant file system and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

**What is dumbo?**

Dumbo is the stand alone Hadoop cluster running on Cloudera Enterprise (CDH 5.9.0). Cloudera Enterprise(CDH) combines Apache Hadoop with a number of other open source projects to create a single, massively scalable system where you can unite storage with an array of powerful processing and analytic frameworks.

**To access dumbo the Hadoop cluster**

- Please follow the instructions on this link: [https://wikis.nyu.edu/display/NYUHPC/Clusters++-Dumbo#Clusters-Dumbo-Dumbo-Hadoop](https://wikis.nyu.edu/display/NYUHPC/Clusters++-Dumbo#Clusters-Dumbo-Dumbo-Hadoop)

  Note: Make sure to follow the instructions for Web UI access using the above link before following the steps below

- Now on the Desktop provided to you please follow the instructions below: (Mac OS X)

Steps to connect to hadoop cluster ie., dumbo.

**Windows:**

Step - 1: User hostname "gw.hpc.nyu.edu" with port 22 in putty. Provide your credentials.
Step - 2: Then, use "ssh <net id>@dumbo.es.its.nyu.edu". It will connect to dumbo cluster.

**Mac:**

Step - 1: From terminal, give "ssh <net id>@gw.hpc.nyu.edu"
Step - 2: Then, use "ssh <net id>@dumbo.es.its.nyu.edu". It will connect to dumbo cluster.
Components of Hive:

- HCatalog is a component of Hive. It is a table and storage management layer for Hadoop that enables users with different data processing tools — including Pig and MapReduce — to more easily read and write data on the grid.
- WebHCat provides a service that you can use to run Hadoop MapReduce (or YARN), Pig, Hive jobs or perform Hive metadata operations using an HTTP (REST style) interface.

Hive is not
- A relational database
- A design for On Line Transaction Processing (OLTP)
- A language for real-time queries and row-level updates.

All the data types in Hive are classified into four types, given as follows:

- Column Types: Integrals (INT), String (CHAR), TimeStamp, Dates, Decimals & Union.
- Literals: Floating Point
- Null Values: NULL
- Complex Types: Arrays, Maps & Structs

Basic Commands in HiveQL:

To access Hive there are two ways:

- For Hive Grunt Shell: Type “Hive” on dumbo

Using Hive Grunt Shell
### Hive Statements:

Provide "HIVE" at prompt to work with hive grunt shell.

```bash
-bash-4.1$ hive
hive>
```

Create database statement:

```bash
CREATE DATABASE example;
```
Create a database if not exists:

```
hive> CREATE DATABASE [IF NOT EXISTS] userdb;
```

Example:

```
hive> create database <net_id>;
```

The following query is used to verify a databases list:

```
hive> SHOW DATABASES;
```

Use statement:

```
hive> USE <database name>;
```

Example:

```
hive> use <net_id>;
```

Drop Database Statement:

```
hive> DROP DATABASE IF EXISTS userdb;
```

Example:

```
hive> drop database <net_id>;
```

Now lets try an example:

Before we start working with examples, please copy the Tutorial2 directory from `/share/apps/Tutorials/Tutorial2/` to `/home/<net_id>/`

```
cp -r /share/apps/Tutorials/Tutorial2/ /home/<net_id>/
```

**Example 1:**

Step 1: Create a table "messages" with columns user, post and time.

```
create table messages (user STRING, post STRING, time BIGINT, country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

Step 2: Use 'describe' to display the list of columns in the table.

```
describe <table_name>;
```

or

```
describe extended <table_name>;
```

Step 3: Load data into hive table.

```
load data local inpath '/home/<net_id>/Tutorial2/user_posts.txt' overwrite into table messages;
```
Step-4: Usage of ‘select’ statement.

<table>
<thead>
<tr>
<th>SQL Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>select count(*) from messages;</code></td>
</tr>
<tr>
<td><code>select * from messages where country='UK';</code></td>
</tr>
<tr>
<td><code>select * from messages where user = 'user1';</code></td>
</tr>
<tr>
<td><code>select * from messages where user = 'user1' or user='user2';</code></td>
</tr>
<tr>
<td><code>select distinct user from messages;</code></td>
</tr>
<tr>
<td><code>select count(distinct user) from messages;</code></td>
</tr>
<tr>
<td><code>select * from messages order by time;</code></td>
</tr>
<tr>
<td><code>select user,count(user) from messages group by user;</code></td>
</tr>
</tbody>
</table>

Step-5: Delete table. NOTE: Do not delete table ‘messages’, as the data from this table is used in next example.

<table>
<thead>
<tr>
<th>SQL Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>drop table &lt;table_name&gt;;</code></td>
</tr>
</tbody>
</table>

With Hive we are operating on the Apache Hadoop data store. Any query you make, table that you create, data that you copy persists from query to query. You can think of Hive as providing a data workbench where you can examine, modify and manipulate the data in Apache Hadoop. So when we perform our data processing task we will execute it one query or line at a time. Once a line successfully executes you can look at the data objects to verify if the last operation did what you expected. All your data is live, This kind of flexibility is Hive’s strength. You can solve problems bit by bit and change your mind on what to do next depending on what you find.

Hive is only pointing to the data on the HDFS file system but there is also an option of using “local inpath”. Hive only stores the structure of table not the data. Data is always accessed from HDFS or local machine.

**Example 2: External Tables**

Step-1: Create external table “messages2” with columns user, post and time.

<table>
<thead>
<tr>
<th>Hive Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hive&gt; CREATE EXTERNAL TABLE messages2(user STRING, post STRING, time BIGINT, country STRING) COMMENT 'Bigdata-Learning' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/user/&lt;net_id&gt;/messages2';</code></td>
</tr>
</tbody>
</table>

Step-2: Copy input data manually to HDFS location '/user/<net_id>/messages2' as given below.

<table>
<thead>
<tr>
<th>Hadoop Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hadoop fs -copyFromLocal /home/&lt;net_id&gt;/Tutorial2/user_posts.txt /user/&lt;net_id&gt;/messages2</code></td>
</tr>
</tbody>
</table>

Step-3: Now, You can query the table ‘messages2’ from hive.

<table>
<thead>
<tr>
<th>Hive Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-bash-4.1$ hive</code></td>
</tr>
<tr>
<td><code>hive&gt; use database;</code></td>
</tr>
<tr>
<td><code>hive&gt; select * from messages2;</code></td>
</tr>
</tbody>
</table>

**PARTITIONING**

**Example 3: Static Partition**

Hive organizes tables into partitions. It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion of the data.

Step-1: Create partitioned hive table. Here we are creating partition for ‘country’ by using `PARTITIONED BY` clause.
create table messages3 (user STRING, post STRING, time BIGINT) PARTITIONED BY (country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

Step-2: Load data from table 'messages' which was created in previous example.

```
hive> INSERT INTO TABLE messages3 PARTITION(country='US') SELECT user, post, time FROM messages WHERE country='US';
hive> INSERT INTO TABLE messages3 PARTITION(country='IND') SELECT user, post, time FROM messages WHERE country='IND';
hive> INSERT INTO TABLE messages3 PARTITION(country='UK') SELECT user, post, time FROM messages WHERE country='UK';
```

Step-3: Alternate way to load data into table.

```
cp -r /share/apps/Tutorials/Tutorial2/can_posts.txt /home/<net_id>/
load data local inpath '/home/<net_id>/Tutorial2/can_posts.txt' overwrite into table messages3 PARTITION (country='CAN');
```

Step-4: Now, You can query the table 'messages3' from hive.

```
hive> select * from messages3;
hive> select * from messages3 where country='CAN' or country='IND';
```

Step-5: Alternate way to see data i.e., Raw data display.

```
hadoop fs -ls /user/hive/warehouse/<net_id>.db/messages3
hadoop fs -cat /user/hive/warehouse/sk6404.db/messages3/country=IND/000000_0
```

NOTE: If we go for the above approach, if we have 50 partitions we need to do the insert statement 50 times. That is a tedious task and it is known as Static Partition.

**Example 4: Dynamic Partition**

- **Static Partition columns**: In DML/DDL involving multiple partitioning columns, the columns whose values are known at COMPILE TIME (given by user).
- **Dynamic Partition columns**: Columns whose values are only known at EXECUTION TIME.

Step-1: Create partitioned hive table. Here we are creating partition for 'country' by using **PARTITIONED BY** clause.

```
create table messages4 (user STRING, post STRING, time BIGINT) PARTITIONED BY (country STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
```

Step-2: In order to achieve dynamic partition we need to execute below SET commands in hive.

1. set hive.exec.dynamic.partition=true : This enable dynamic partitions, by default it is false.
2. set hive.exec.dynamic.partition.mode=nonstrict : We are using the dynamic partition without a static partition (A table can be partitioned based on multiple columns in hive) in such case we have to enable the non strict mode. In strict mode we can use dynamic partition only with a Static Partition.
Step-3: Load the data into partitioned table 'messages4' from table 'messages' which was created in previous examples.

```sql
hive> FROM messages
INSERT OVERWRITE TABLE messages4 PARTITION(country)
SELECT user,post,time,country DISTRIBUTE BY country;
```

Step-4: Now, you can query the table 'messages4'.

```sql
select * from messages4;
select * from messages4 where country = 'UK';
select * from messages4 limit 5;
select count(*) from messages4;
```

Example 5: Bucketing

Tables or partitions are sub-divided into buckets, to provide extra structure to the data that is used for more efficient querying. Bucketing works based on the value of hash function of some column of a table.

Step-1: In hive, bucketing does not work by default. You will have to set following variable to enable bucketing. set hive.enforce.bucketing=true;

```sql
hive> set hive.enforce.bucketing;
hive> set hive.enforce.bucketing=true;
```

Step-2: Create bucketed table.

```sql
hive> create table messages5 (user STRING, post STRING, time BIGINT, country STRING)
clustered by (country) into 3 buckets
row format delimited fields terminated by " ";
```

Step-3: Load data into bucketed table 'messages5' from table 'messages' created in previous example.

```sql
hive> FROM messages
INSERT into table messages5
SELECT user,post,time,country;
```

Step-4: Now, you can query the table 'messages5'.

```sql
select * from messages5;
select * from messages5 TABLESAMPLE (BUCKET 1 OUT OF 3);
select * from messages5 TABLESAMPLE (BUCKET 2 OUT OF 3);
select * from messages5 TABLESAMPLE (BUCKET 3 OUT OF 3);
```

(Please contact hpc@nyu.edu to learn more)