Big Data Tutorial 2: Hive

Slides for this tutorial can be found [here](#).

**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 5.11.1). 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-Hadooppcluster

  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.

<table>
<thead>
<tr>
<th>Components of Hive:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- 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.</td>
</tr>
<tr>
<td>- 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.</td>
</tr>
</tbody>
</table>
Hive is not

- A relational database
- Designed for Online 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 “beeline” on dumbo
- Hue UI: Provides a Hive interface -> http://demo.gethue.com/

Using Hive Grunt Shell

<table>
<thead>
<tr>
<th>Function</th>
<th>MySQL</th>
<th>Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieving Information (General)</td>
<td>SELECT from_columns FROM table WHERE conditions;</td>
<td>SELECT from_columns FROM table WHERE conditions;</td>
</tr>
<tr>
<td>Retrieving All Values</td>
<td>SELECT * FROM table;</td>
<td>SELECT * FROM table;</td>
</tr>
</tbody>
</table>
Beeline: Hive CLI is deprecated and migration to Beeline is recommended.

```
-bash-4.1$ beeline

beeline> !connect jdbc:hive2://babar.es.its.nyu.edu:10000/
Enter username for jdbc:hive2://babar.es.its.nyu.edu:10000/: <net_id>
Enter password for jdbc:hive2://babar.es.its.nyu.edu:10000/: ************

0: jdbc:hive2://babar.es.its.nyu.edu:10000/> use <netid> ;
```

Hive Statements:
User databases are precreated for users on Dumbo. The following query is used to verify list of databases you have access to:

**Retrieving Some Values**
- `SELECT * FROM table WHERE rec_name = "value";`
- `SELECT * FROM table WHERE rec_name = "value";`

**Retrieving With Multiple Criteria**
- `SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";`
- `SELECT * FROM TABLE WHERE rec1 = "value1" AND rec2 = "value2";`

**Retrieving Specific Columns**
- `SELECT column_name FROM table;`
- `SELECT column_name FROM table;`

**Retrieving Unique Output**
- `SELECT DISTINCT column_name FROM table;`
- `SELECT DISTINCT column_name FROM table;`

**Sorting**
- `SELECT col1, col2 FROM table ORDER BY col2;`
- `SELECT col1, col2 FROM table ORDER BY col2;`

**Sorting Reverse**
- `SELECT col1, col2 FROM table ORDER BY col2 DESC;`
- `SELECT col1, col2 FROM table ORDER BY col2 DESC;`

**Counting Rows**
- `SELECT COUNT(*) FROM table;`
- `SELECT COUNT(*) FROM table;`

**Grouping With Counting**
- `SELECT owner, COUNT(*) FROM table GROUP BY owner;`
- `SELECT owner, COUNT(*) FROM table GROUP BY owner;`

**Maximum Value**
- `SELECT MAX(col_name) AS label FROM table;`
- `SELECT MAX(col_name) AS label FROM table;`

**Selecting from multiple tables (Join same table using alias w/"AS")**
- `SELECT pet.name, comment FROM pet, event WHERE pet.name = event.name;`
- `SELECT pet.name, comment FROM pet JOIN event ON (pet.name = event.name) ;`

---

<table>
<thead>
<tr>
<th>Command Line</th>
<th>Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td><strong>Hive</strong></td>
</tr>
<tr>
<td>Run Query</td>
<td>beeline -u jdbc:hive2://babar.es.its.nyu.edu:10000/&lt;NetID&gt; -n NetID -w /home/&lt;netID&gt;/&lt;passwordfile&gt; -e 'select * from &lt;table&gt;'</td>
</tr>
<tr>
<td>Run Non-Interactive Script</td>
<td>beeline -u jdbc:hive2://babar.es.its.nyu.edu:10000/&lt;NetID&gt; -n NetID -w /home/&lt;netID&gt;/&lt;passwordfile&gt; -f /home/&lt;netID&gt;/st.hql &gt;&gt; out.txt</td>
</tr>
</tbody>
</table>
Now let's try an example:

Before we start working with examples, please copy the Tutorial2 directory from '/share/apps/Tutorials/Tutorial2/' to '/home/netid/'

Example 1:

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

```java
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.

```java
describe <table_name>;
or
describe extended <table_name>;
```

Step-3: Load data into hive table.

```java
load data inpath 'hdfs://babar.es.its.nyu.edu:8020/user/<net_id>/user_posts.txt' overwrite into table messages;
```

Step-4: Usage of 'select' statement.

```java
select count(*) from messages;
select * from messages where country='UK';
select * from messages where user = 'user1';
select * from messages where user = 'user1' or user='user2';
select distinct user from messages;
select count(distinct user) from messages;
select * from messages order by time;
select user,count(user) from messages group by user;
```

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

```java
drop table <table_name>;
```
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.

```sql
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/<net_id>/messages2';
```

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

```bash
hadoop fs -copyFromLocal /home/<net_id>/Tutorial2/user_posts.txt /user/<net_id>/messages2
```

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

```bash
-bash-4.1$ beeline
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> select * from messages2;
```

**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.

```sql
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.

```sql
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> INSERT INTO TABLE messages3 PARTITION(country='US')
SELECT user, post, time FROM messages WHERE country='US';
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> INSERT INTO TABLE messages3 PARTITION(country='IND')
SELECT user, post, time FROM messages WHERE country='IND';
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> 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.

```bash
cp -r /share/apps/Tutorials/Tutorial2/can_posts.txt /home/<net_id>/
hdfs dfs -put $HOME/Tutorial2/can_posts.txt /user/<net_id>/
```

0: jdbc:hive2://babar.es.its.nyu.edu:10000/> load data inpath 'hdfs://babar.es.its.nyu.edu:8020/user/<net_id>/can_posts.txt' overwrite into table messages3 PARTITION (country='CAN');

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

```sql
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> select * from messages3;
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> select * from messages3 where country='CAN' or country='IND';
```

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

```bash
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 COMPILATION 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.

```sql
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.

```sql
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> SET hive.exec.dynamic.partition = true;
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> SET hive.exec.dynamic.partition.mode = nonstrict;
```

Step-3: Load the data into partitioned table 'messages4' from table 'messages' which was created in previous examples.

```sql
0: jdbc:hive2://babar.es.its.nyu.edu:10000/> FROM messages
INSERT OVERWRITE TABLE messages4 PARTITION(country)
SELECT user.post.time.country DISTIBUTE BY country;
```

Step-4: Now, You can query the table '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;

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

**Step-2:** Create bucketed table.

```
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.

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

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

```
select * from messages5;
selct * from messages5 where country = 'UK';
selct * from messages5 limit 5;
selct count(*) from messages5;
```

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