What is Hive?

Apache Hive is considered the defacto standard for interactive SQL queries over petabytes of data in Hadoop.

Hadoop was built to organize and store massive amounts of data of all shapes, sizes and formats. Because of Hadoop's "schema on read" architecture, a Hadoop cluster is a perfect reservoir of heterogeneous data, structured and unstructured, from a multitude of sources.

Data analysts use Hive to query, summarize, explore and analyze that data, then turn it into actionable business insight.

Hive also provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL.

HiveQL also allows traditional map/reduce programmers to plug in their custom mappers and reducers.

Install Hive

1**. We can download the Hive: https://hive.apache.org/downloads.html**

```
$ wget http://www-us.apache.org/dist/hive/hive-2.1.0/apache-hive-2.1.0-bin.tar.gz

$ sudo tar xvzf apache-hive-2.1.0-bin.tar.gz  -C /home/mamoon
```

2. Open ~/.bashrc and set the environment variable HIVE_HOME to point to the installation directory and PATH:

```
export HIVE_HOME=/home/mamoon/apache-hive-2.1.0-bin
export HIVE_CONF_DIR=/home/mamoon/apache-hive-2.1.0-bin/conf
export PATH=$HIVE_HOME/bin:$PATH
export CLASSPATH=$CLASSPATH:/usr/local/hadoop/lib/*:.
export CLASSPATH=$CLASSPATH:/usr/local/apache-hive-2.1.0-bin/lib/*:.
```

3. Activate the new setting for Hive:

```
apache-hive-2.1.0-bin$ source ~/.bashrc
```

4. Creating Hive warehouse directory

Hive uses Hadoop, so we must have Hadoop in our path:

```
$ echo $HADOOP_INSTALL
```

/home/mamoon/hadoop-3.1.1

In addition, we must use below HDFS commands to create /tmp and /user/hive/warehouse (aka hive.metastore.warehouse.dir) and set them chmod g+w before we can create a table in Hive:

```
$ hdfs dfs -ls /
drwxr-xr-x   - hduser supergroup          0 2016-11-23 11:17 /hbase
drwx------   - hduser supergroup          0 2016-11-18 16:04 /tmp
drwxr-xr-x   - hduser supergroup          0 2016-11-18 09:13 /user

$ hdfs dfs -mkdir /user/hive/warehouse
$ hdfs dfs -chmod g+w /tmp
$ hdfs dfs -chmod g+w /user/hive/warehouse

$ hdfs dfs -ls /
drwxr-xr-x   - hduser supergroup          0 2016-11-23 11:17 /hbase
drwx-w----   - hduser supergroup          0 2016-11-18 16:04 /tmp
drwxr-xr-x   - hduser supergroup          0 2016-11-23 17:18 /user

$ hdfs dfs -ls /user
drwxr-xr-x   - hduser supergroup          0 2016-11-18 23:17 /user/hduser
drwxr-xr-x   - hduser supergroup          0 2016-11-23 17:18 /user/hive
```

The directory warehouse is the location to store the table or data related to hive, and the temporary directory tmp is the temporary location to store the intermediate result of processing.

5. Configuring Hive

To configure Hive with Hadoop, we need to edit the hive-env.sh file, which is placed in the $HIVE_HOME/conf directory. The following commands redirect to Hive conf folder and copy the template file:

```
cd $HIVE_HOME/conf
apache-hive-2.1.0-bin/conf$ sudo cp hive-env.sh.template hive-env.sh
```

6. Edit the hive-env.sh file by appending the following line:

```
export HADOOP_INSTALL=/home/mamoon/hadoop-3.1.1
```

Hive installation is completed successfully. Now we need an external database server to configure Metastore. We use Apache Derby database.

7. Downloading Apache Derby

The following command is used to download Apache Derby:

```
$ cd /tmp

$ wget http://archive.apache.org/dist/db/derby/db-derby-10.13.1.1/db-derby-10.13.1.1-bin.tar.gz

$ sudo tar xvzf db-derby-10.13.1.1-bin.tar.gz -C /home/mamoon
```

8. Let's set up the Derby environment by appending the following lines to ~/.bashrc file:

```
export DERBY_HOME=/home/mamoon/db-derby-10.13.1.1-bin
export PATH=$PATH:$DERBY_HOME/bin
export CLASSPATH=$CLASSPATH:$DERBY_HOME/lib/derby.jar:$DERBY_HOME/lib/derbytools.jar
```

9. We need to create a directory named data in $DERBY_HOME directory to store Metastore data.

```
$ sudo mkdir $DERBY_HOME/data
```

Now we completed Derby installation and environmental setup.

10 Configuring Hive Metastore

Configuring Metastore means specifying to Hive where the database is stored. We want to do this by editing the hive-site.xml file, which is in the $HIVE_HOME/conf directory.

Let's copy the template file using the following command:

```
$ cd $HIVE_HOME/conf

apache-hive-2.1.0-bin/conf$ sudo cp hive-default.xml.template hive-site.xml
```

11. Make sure the following lines are between the <configuration> and </configuration> tags of hive-site.xml:

```
<property>
   <name>javax.jdo.option.ConnectionURL</name>
   <value>jdbc:derby:;databaseName=metastore_db;create=true</value>
   <description>
     JDBC connect string for a JDBC metastore.
     To use SSL to encrypt/authenticate the connection, provide database-specific SSL flag in the connection URL.
     For example, jdbc:postgresql://myhost/db?ssl=true for postgres database.
   </description>
 </property>
```

12. Create a file named jpox.properties and add the following lines into it:

```
javax.jdo.PersistenceManagerFactoryClass =

org.jpox.PersistenceManagerFactoryImpl
org.jpox.autoCreateSchema = false
org.jpox.validateTables = false
org.jpox.validateColumns = false
org.jpox.validateConstraints = false
org.jpox.storeManagerType = rdbms
org.jpox.autoCreateSchema = true
org.jpox.autoStartMechanismMode = checked
org.jpox.transactionIsolation = read_committed
javax.jdo.option.DetachAllOnCommit = true
javax.jdo.option.NontransactionalRead = true
javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver
javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore_db;create = true
javax.jdo.option.ConnectionUserName = APP
javax.jdo.option.ConnectionPassword = mine
```

13. We need to set permission to Hive folder:

$ sudo chown -R mamoon:mamoon apache-hive-2.1.0-bin

14. Metastore schema initialization

Starting from Hive 2.1, we need to run the schematool command below as an initialization step. In our case, we use derby as db type:

apache-hive-2.1.0-bin/bin$ schematool -dbType derby –initSchema

15. Verifying Hive Installation by running Hive CLI

To use the Hive command line interface (CLI) from the shell, issue bin/hive command to verify Hive

$ echo $HIVE_HOME
/home/mamoon/apache-hive-2.1.0-bin

$ $HIVE_HOME/bin/hive

We may get couple of errors when we try to start hive via bin/hive command. The followings are the errors and corresponding fixes:

1. Error #1:

Exception in thread "main" java.lang.RuntimeException: Couldn't create directory ${system:java.io.tmpdir}/${hive.session.id}_resources

Fix #1: edit hive-site.xml:

```
<property>
   <name>hive.downloaded.resources.dir</name>
   <!--
   <value>${system:java.io.tmpdir}/${hive.session.id}_resources</value>
   -->
   <value>/home/mamoon/apache-hive-3.1.0-bin/tmp/${hive.session.id}_resources</value>
   <description>Temporary local directory for added resources in the remote file system.</description>
  </property>
```

Error #2:

java.net.URISyntaxException: Relative path in absolute URI: ${system:java.io.tmpdir%7D/$%7Bsystem:user.name%7D

Fix #2: replace}/${system:user.name} by /tmp/mydir in hive-site.xml (see Confluence - AdminManual Configuration):${system:java.io.tmpdir

```
<property>
   <name>hive.exec.local.scratchdir</name>
   <!--
   <value>${system:java.io.tmpdir}/${system:user.name}</value>
   -->
   <value>/tmp/mydir</value>
   <description>Local scratch space for Hive jobs</description>
 </property>
```

16. Now that we fixed the errors, let's start Hive CLI:

apache-hive-2.1.0-bin/bin$ hive

17. To display all the tables:

```
hive> show tables;
OK
Time taken: 4.603 seconds
```

18. We can exit from that Hive shell by using exit command:

```
hive> exit;
mamoon@mamoon-VirtualBox:apache-hive-2.1.0-bin/bin$
```

Hive - Create Database:

Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it.

Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables. The syntax for this statement is as follows:

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named tempdb:

hive> CREATE DATABASE [IF NOT EXISTS] tempdb;
or

hive> CREATE SCHEMA tempdb;
The following query is used to verify a databases list:

- The following query is used to verify a databases list:

```
hive> SHOW DATABASES;
default
tempdb
```

Hive - Drop Database:

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database_name [RESTRICT|CASCADE];

- The following queries are used to drop a database. Let us assume that the database name is userdb.

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

- The following query drops the database using CASCADE. It means dropping respective tables before dropping the database.

```
hive> DROP DATABASE IF EXISTS temp CASCADE;
```

The following query drops the database using SCHEMA.

```
hive> DROP SCHEMA temp;
```

Hive - Create Table:

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax
CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]

Example
Let us assume you need to create a table named employee using CREATE TABLE statement. The following table lists the fields and their data types in employee table:

Sr.No	Field Name	Data Type
1	Eid	int
2	Name	String
3	Salary	Float
4	Designation	string

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

```
COMMENT ‘Employee details’
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED IN TEXT FILE
```

The following query creates a table named employee using the above data.

```
hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String, salary String, destination String)
COMMENT ‘Employee details’
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ‘\t’
LINES TERMINATED BY ‘\n’
STORED AS TEXTFILE;
```

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

Load Data Statement:

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system.

Syntax
The syntax for load data is as follows:

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)]
LOCAL is identifier to specify the local path. It is optional.
OVERWRITE is optional to overwrite the data in the table.
PARTITION is optional.

Example
We will insert the following data into the table. It is a text file named emp.txt in /home/mamoon directory.

```
1201    Gopal            45000       Technical manager
1202   Manisha         45000       Proof reader
1203    Masthanvali   40000        Technical writer
1204    Kiran            40000        Hr Admin
1205    Kranthi          30000        Op Admin
```

The following query loads the given text into the table.

hive> LOAD DATA LOCAL INPATH '/home/mamoon/emp.txt‘ OVERWRITE INTO TABLE employee;

Hive Partitions:

Hive Partitions is a way to organizes tables into partitions by dividing tables into different parts based on partition keys.

Partition is helpful when the table has one or more Partition keys. Partition keys are basic elements for determining how the data is stored in the table.

For Example: -
"Client having Some E –commerce data which belongs to India operations in which each state (38 states) operations mentioned in as a whole. If we take state column as partition key and perform partitions on that India data as a whole, we can able to get Number of partitions (38 partitions) which is equal to number of states (38) present in India. Such that each state data can be viewed separately in partitions tables.

1. Creation of Table all states

```
create table allstates(state string, District string,Enrolments string)

row format delimited

fields terminated by ',';
```

2. Loading data into created table allstates:

```
Load data local inpath '/home/mamoon/test.csv' into table allstates;
```

3. Creation of partition table

```
create table state_part(District string,Enrolments string) PARTITIONED BY(state string);
```

4. For partition we have to set this property

```
set hive.exec.dynamic.partition.mode=nonstrict
```

5. Loading data into partition table

```
INSERT OVERWRITE TABLE state_part PARTITION(state)
SELECT district,enrolments,state from  allstates;
```

6. Actual processing and formation of partition tables based on state as partition key

7. There are going to be 38 partition outputs in HDFS storage with the file name as state name. We will check this in this step

Bucketing:

- Partition helps in increasing the efficiency when performing a query on a table. Instead of scanning the whole table, it will only scan for the partitioned set and does not scan or operate on the unpartitioned sets, which helps us to provide results in lesser time and the details will be displayed very quickly because of Hive Partition.

- At times, even after partitioning on a particular field or fields, the partitioned file size doesn't match with the actual expectation and remains huge and we want to manage the partition results into different parts. To overcome this problem of partitioning, Hive provides Bucketing concept, which allows user to divide table data sets into more manageable parts.

- Bucketing helps user to maintain parts that are more manageable and user can set the size of the manageable parts or Buckets too.

- Hive partition divides table into number of partitions and these partitions can be further subdivided into more manageable parts known as Buckets or Clusters.

- The Bucketing concept is based on Hash function, which depends on the type of the bucketing column.

- Records which are bucketed by the same column will always be saved in the same bucket.

- ***CLUSTERED BY*** clause is used to divide the table into buckets.

- In Hive Partition, each partition will be created as directory. But in Hive Buckets, each bucket will be created as file.

- Bucketing can also be done even without partitioning on Hive tables.

Bucketing Example:

1. Input Dataset to Perform Bucketing Operation.
2. Creating a New Input Table.
3. Load the Input Dataset.
4. Set hive.enforce.bucketing = true
5. Creating Bucket Table
6. Query to Retrieve Data from Bucketed Table

What is a View?

Views are similar to tables, which are generated based on the requirements.

We can save any result set data as a view in Hive
Usage is similar to as views used in SQL
All type of DML operations can be performed on a view

Creation of View:

Syntax:

Create VIEW < VIEWNAME> AS SELECT

Example:

Hive>Create VIEW Sample_View AS SELECT * FROM employees WHERE salary>25000

In this example, we are creating view Sample_View where it will display all the row values with salary field greater than 25000.

What is Index?

Indexes are pointers to particular column name of a table.

The user has to manually define the index
Wherever we are creating index, it means that we are creating pointer to particular column name of table
Any Changes made to the column present in tables are stored using the index value created on the column name.
Syntax:

Create INDEX < INDEX_NAME> ON TABLE < TABLE_NAME(column names)>

Example:

Create INDEX sample_Index ON TABLE emp_table(id)

Here we are creating index on table emp_table for column name id.

Hive Queries: Order By, Group By, Distribute By, Cluster By Examples

Hive provides SQL type querying language for the ETL purpose on top of Hadoop file system.

Hive Query language (HiveQL) provides SQL type environment in Hive to work with tables, databases, queries.

We can have a different type of Clauses associated with Hive to perform different type data manipulations and querying. For better connectivity with different nodes outside the environment. HIVE provide JDBC connectivity as well.

Hive queries provides the following features:

Data modeling such as Creation of databases, tables, etc.
ETL functionalities such as Extraction, Transformation, and Loading data into tables.
Joins to merge different data tables.
User specific custom scripts for ease of code.
Faster querying tool on top of Hadoop.

Creating Table in Hive:

We are going to create table "employees_guru" with 6 columns.

```
hive> create table employees_guru(Id INT, Name STRING, Age INT, Address STRING, Salary FLOAT, Department STRING)
    > Row format delimited
    > Fields terminated by ',';
OK
Time taken: 0.194 seconds
hive> load data local inpath '/home/hduser/Employees.txt' into TABLE employees_guru;
Loading data to table default.employees_guru
Table default.employees_guru stats: [numFiles=1, totalSize=
OK
Time taken: 0.278 seconds
```

1 "Employees_guru" table creation

2 Loading data into "employeees_guru"

From the above screen shot,

We are creating table "employees_guru" with 6 column values such as Id, Name, Age, Address, Salary, Department, which belongs to the employees present in organization "guru."
Here in this step we are loading data into employees_guru table. The data that we are going to load will be placed under Employees.txt file

Order by query:

The ORDER BY syntax in HiveQL is similar to the syntax of ORDER BY in SQL language.
Order by is the clause we use with "SELECT" statement in Hive queries, which helps sort data. Order by clause use columns on Hive tables for sorting particular column values mentioned with Order by. For whatever the column name we are defining the order by clause the query will selects and display results by ascending or descending order the particular column values.

If the mentioned order by field is a string, then it will display the result in lexicographical order. At the back end, it has to be passed on to a single reducer.

```
hive> SELECT * FROM employees_guru ORDER BY Department;
Query ID = hduser_201511171701 70229_6e8af657-126b-470f-8b88
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks de
In order to change the av                                    n by
  set hive.exec.reducers
In order to limit the max
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201511171701_0001, Tracking URL = htt
etails.jsp?jobid=job_201511171701_0001
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/ha
11171701_0001
```

order by query on "Employees_guru" table

From the Above screen shot, we can observe the following

It is the query that performing on the "employees_guru" table with the ORDER BY clause with Department as defined ORDER BY column name.
"Department" is String so it will display results based on lexicographical order.

This is actual output for the query. If we observe it properly, we can see that it get results displayed based on Department column such as ADMIN, Finance and so on in orderQuery to be perform.

Query:
SELECT * FROM employees_guru ORDER BY Department;

Group by query:
Group by clause use columns on Hive tables for grouping particular column values mentioned with the group by. For whatever the column name we are defining a "groupby" clause the query will selects and display results by grouping the particular column values.

For example, in the below screen shot it's going to display the total count of employees present in each department. Here we have "Department" as Group by value.

```
hive> SELECT Department, count(*) FROM employees_guru GROUP BY Department;
Query ID = hduser_201511051553 07_1574cd2b-866e-437a-8d14-657e02b7e513
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not sp if                                    size: 1
In order to change the average l
  set hive.exec.reducers.bytes.pe
In order to limit the maximum num
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0005, Tracking URL = http://localhost:5003
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop job  -kill jo
Hadoop job information for Stage-1: number of mappers: 1; number of reduce
2015-11-05 15:53:19,229 Stage-1 map = 0%,  reduce = 0%
2015-11-05 15:53:21,235 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1
2015-11-05 15:53:28,277 Stage-1 map = 100%,  reduce
2015-11-05 15:53:29,281 Stage-1 map = 100%,  reduc                    PU
MapReduce Total cumulative CPU time: 2 seconds 130
Ended Job = job_201511051442_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU 2.13 sec   HDFS Read: 7
Total MapReduce CPU Time Spent: 2 seconds 130 msec
OK
ADMIN    2
FINANCE  1
HR       2
IT       4
PR       2
Time taken: 23.057 seconds, Fetched: 5 row(s)
```

1

Groupby query on "employees_guru"

2

Group by query output

From the above screenshot, we will observe the following

It is the query that is performed on the "employees_guru" table with the GROUP BY clause with Department as defined GROUP BY column name.
The output showing here is the department name, and the employees count in different departments. Here all the employees belong to the specific department is grouped by and displayed in the results. So the result is department name with the total number of employees present in each department.

Query:

SELECT Department, count(*) FROM employees_guru GROUP BY Department;

Sort by:
Sort by clause performs on column names of Hive tables to sort the output. We can mention DESC for sorting the order in descending order and mention ASC for Ascending order of the sort.

In this sort by it will sort the rows before feeding to the reducer. Always sort by depends on column types.

For instance, if column types are numeric it will sort in numeric order if the columns types are string it will sort in lexicographical order.

```
hive> Select * from employees_guru SORT BY id DESC;
Query ID = hdus 1 20151105164027_55bf2f64-6f5b-4764-b94e-e03f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not spe                                  put da
In order to change the average                                  ytes) :
  set hive.exec.reducers.bytes
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0007, Tracking URL = http://l
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop
Hadoop job information for St                   ppers: 1; num
2015-11-05 16:40:34,093 Stage                   e = 0%
2015-11-05 16:40:36,098 Stage                   uce = 0%, Cum
2015-11-05 16:40:44,145 Stage                   uce = 100%, C
MapReduce Total cumulative CP                   20 msec
Ended Job = job_201511051442_
MapReduce Jobs Launched:
Stage-Stage-1: Map 1  Reduce: 1   Cumulative CPU: 1.62 sec
Total MapReduce CPU Time Spent: 1 seconds 620 msec
OK
111     Syam     33      bangalore        25000.0 PR
110     Ravi     28      bangalore        20000.0 IT
109     Suresh   32      Kolkata 20000.0 IT
108     Sravan   31      Mumbai   60000.0 IT
107     Sravanthi        32      Chennai 20000.0 IT
106     Ramesh   30      Goa      24000.0 FINANCE
105     Santosh 33       bangalore        25000.0 PR
104     Anirudh 27       bangalore        27000.0 ADMIN
103     Animesh 26       Bangalore        25000.0 ADMIN
102     Rajiv    28      Delhi    30000.0 HR
101     Rajesh   27      Bangalore        20000.0 HR
Time taken: 18.088 seconds, Fetched: 11 row(s)
```

sort by query

sort by output on "employees_guru" table

From the above screen shot we can observe the following:

It is the query that performing on the table "employees_guru" with the SORT BY clause with "id" as define SORT BY column name. We used keyword DESC.
So the output displayed will be in descending order of "id".

Query:

SELECT * from employees_guru SORT BY Id DESC;

Cluster By:
Cluster By used as an alternative for both Distribute BY and Sort BY clauses in Hive-QL.

Cluster BY clause used on tables present in Hive. Hive uses the columns in Cluster by to distribute the rows among reducers. Cluster BY columns will go to the multiple reducers.

It ensures sorting orders of values present in multiple reducers
For example, Cluster By clause mentioned on the Id column name of the table employees_guru table. The output when executing this query will give results to multiple reducers at the back end. But as front end it is an alternative clause for both Sort By and Distribute By.
This is actually back end process when we perform a query with sort by, group by, and cluster by in terms of Map reduce framework. So if we want to store results into multiple reducers, we go with Cluster By.

```
hive> Select Id,Name from employees guru CLUSTER BY Id;
Query ID = hd_er_20151105165000_72cedc06-a797-48b1-a120-
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not                                  rom inpu
In order to change the ave                                  (in byt
  set hive.exec.reducers.b                                  >
In order to limit the maxi
  set hive.exec.reducers.m
In order to set a constant
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0009, Tracking URL = http
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/had
Hadoop job information for Stage-1: number of mappers: 1;
2015-11-05 16:50:08,541 Stage-1 map = 0%,  reduce = 0%
2015-11-05 16:50:10,546                    reduce = 0%,
2015-11-05 16:50:17,563                    reduce = 100
MapReduce Total cumulat                    nds 600 msec
Ended Job = job_201511
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.6 se
Total MapRe e CPU Time Spent: 1 seconds 600 msec
OK
101     Rajesh
102     Rajiv
103     Animesh
104     Anirudh
105     Santosh
106     Ramesh
107     Sravanthi
108     Sravan
109     Suresh
110     Ravi
111     Syam
Time taken: 18.941 seconds, Fetched: 11 row(s)
```

1

cluster by query

cluster by query output

2

From the above screen shot we are getting the following observations:

It is the query that performs CLUSTER BY clause on Id field value. Here it's going to get a sort on Id values.
It displays the Id and Names present in the guru_employees sort ordered by

Query:

SELECT Id, Name from employees_guru CLUSTER BY Id;

Distribute By:

Distribute BY clause used on tables present in Hive. Hive uses the columns in Distribute by to distribute the rows among reducers. All Distribute BY columns will go to the same reducer.

It ensures each of N reducers gets non-overlapping ranges of column
It doesn't sort the output of each reducer

```
hive> Select Id,Name from employees guru DISTRIBUTE BY Id;
Query ID = hduser_20151105165433_65088a93-2ec2-4878-985d-cb
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not                    from input
In order to change the ave                    r (in bytes
  set hive.exec.reducers.b                    er>
In order to limit the max                     :
  set hive.exec.reducers.m
In order to set a constant
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0010, Tracking URL = http:/
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoo
Hadoop job information for Stage-1: number of mappers: 1; n
2015-11-05 16:54:42,456 Stage-1 map = 0%,  reduce = 0%
2015-11-05 16:54:43,4                 00%,  reduce = 0%, C
2015-11-05 16:54:51,4                 0%,  reduce = 100%,
MapReduce Total cumul                 econds 790 msec
Ended Job = job_20151
MapReduce Jobs Launche
Stage-Stage-1: Map:                   educe: 1   Cumulative CPU: 1.79 sec
Total Map                 Time Spent: 1 seconds 790 msec
OK
101     Rajesh
102     Rajiv
103     Animesh
104     Anirudh
105     Santosh
106     Ramesh
107     Sravanthi
108     Sravan
109     Suresh
110     Ravi
111     Syam
Time taken: 19.584 seconds, Fetched: 11 row(s)
hive>
```

1

Distribute by query

Distribute by query output

2

From the above screenshot, we can observe the following:

DISTRIBUTE BY Clause performing on Id of "empoloyees_guru" table
Output showing Id, Name. At back end, it will go to the same reducer

Query:

SELECT Id, Name from employees_guru DISTRIBUTE BY Id;

Hive Join & SubQuery Tutorial with Examples:

Join queries:
Join queries can perform on two tables present in Hive. For understanding Join Concepts in clear here we are creating two tables overhere,

Sample_joins(Related to Customers Details)
Sample_joins1(Related to orders details done by Employees)

Step 1) Creation of table "sample_joins" with Column names ID, Name, Age, address and salary of the employees

```
hive> create table sample_joins(Id INT, Name STRING, Age INT, Address STRING, Salary FLOAT)
    > Row format delimited
    > Fields terminated by ',';
OK
Time taken: 0.748 seconds
```

Creation of "sample_joins" table

Step 2) Loading and Displaying Data

```
hive> load data local inpath '/home/hduser/Customers.txt' into TABLE sample_joins;
Loading data to table default.sample_joins
Table default.sample_joins stats: [numFiles=1, totalSize=451]
OK
Time taken: 0.357 seconds
hive> select *  from sample_joins;
OK
101     Rajesh  27      Bangalore       20000.0
102     Rajiv   28      Delhi   30000.0
103     Animesh 26      Bangalore       25000.0
104     Anirudh 27      bangalore       27000.0
105     Santosh 33      bangalore       25000.0
106     Ramesh  30      Goa     24000.0
107     Sravanthi       32      Chennai 20000.0
108     Sravan  31      Mumbai  60000.0
109     Suresh  32      Kolkata 20000.0
110     Ravi    28      bangalore       20000.0
111     Sravani 25      bangalore       2000.0
112     Kiran   34      bangalore       20000.0
113     Kishore 35      bangalore       27000.0
114     Prakash 27      bangalore       28000.0
115     Dheeraj 31      bangalore       35000.0
Time taken: 0.501 seconds, Fetched: 15 row(s)
```

1

loading data into "sample_joins" table

2

Displaying "sample_joins" table data

From the above screen shot

Loading data into sample_joins from Customers.txt
Displaying sample_joins table contents

Step 3) Creation of sample_joins1 table and loading, displaying data

```
hive> create TABLE sample_joins1(OrderId INT, Date1 TIMESTAMP, Id INT, Amount FLOAT)
    > Row format delimited
    > Fields terminated by ',';
OK
Time taken: 0.117 seconds
hive> load data local inpath '/home/hduser/orders.txt' into TABLE sample_joins1;
Loading data to table default.sample_joins1
Table default.sample_joins1 stats: [numFiles=1, totalSize=208]
OK
Time taken: 0.208 seconds
hive> SELECT * from sample_joins1;
OK
1100    2015-10-08 00:00:00     103     2000.0
1101    2015-10-15 00:00:00     105     1500.0
1102    2012-11-02 00:00:00     109     2000.0
1103    2014-12-08 00:00:00     112     1800.0
1104    2013-12-18 00:00:00     115     2500.0
1105    2014-12-25 00:00:00     107     3200.0
Time taken: 0.117 seconds, Fetched: 6 row(s)
```

Displaying "sample_joins 1" table data

From the above screenshot, we can observe the following

Creation of table sample_joins1 with columns Orderid, Date1, Id, Amount
Loading data into sample_joins1 from orders.txt
Displaying records present in sample_joins1
Moving forward we will see different types of joins that can be performed on tables we have created but before that you have to consider following points for joins.

Some points to observe in Joins:

Only Equality joins are allowed In Joins
More than two tables can be joined in the same query
LEFT, RIGHT, FULL OUTER joins exist in order to provide more control over ON Clause for which there is no match
Joins are not Commutative
Joins are left-associative irrespective of whether they are LEFT or RIGHT joins

Different type of joins:

Joins are of 4 types, these are

Inner join
Left outer Join
Right Outer Join
Full Outer Join

Inner Join:

The Records common to the both tables will be retrieved by this Inner Join.

From the below screenshot, we can observe the following

Here we are performing join query using JOIN keyword between the tables sample_joins and sample_joins1 with matching condition as (c.Id= o.Id).
The output displaying common records present in both the table by checking the condition mentioned in the query
Query:

SELECT c.Id, c.Name, c.Age, o.Amount FROM sample_joins c JOIN sample_joins1 o ON(c.Id=o.Id);

```
hive> SELECT c.Id, c.Name, c.Age, o.Amount FROM sample_joins c
    > JOIN sample_joins1 o ON(c.Id=o.Id)
Query ID = hduser_20151105150626_d518db65-233d-4206-b2d7-2c704
Total jobs = 1
Execution log at: /tmp/hduser/hduser_20151105150626_d518db65-2
2015-11-05 15:06:33     Starting
2015-11-05 15:06:34     Dump the
5_15-06-26_406_7628437655443769 42-1/
2015-11-05 15:06:34     Uploaded 1 Fi
-10003/HashTable-Stage-3/MapJoin-mapf
2015-11-05 15:06:34     End of local
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce oper
Starting Job = job_201511051442_0001, Tracking URL = http://loc
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop jo
Hadoop job information for Stage-3: number of mappers: 1; numbe
2015-11-05 15:06:44,167 S                reduce = 0%
2015-11-05 15:06:46,180 %,  reduce = 0%, Cumul
2015-11-05 15:06:47,191 %,  reduce = 100%, Cu
MapReduce Total cumulati msec
Ended Job = job_201511
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 0.94 sec   HDFS Read:
Total MapReduce CPU Time Spent: 940 msec
OK
103     Animesh 26      2000.0
105     Santosh 33      1500.0
107     Sravanthi       32      3200.0
109     Suresh  32      2000.0
112     Kiran   34      1800.0
115     Dheeraj 31      2500.0
Time taken: 20.879 seconds, Fetched: 6 row(s)
```

1

Inner Join query

2

Inner Join output

Left Outer Join:

Hive query language LEFT OUTER JOIN returns all the rows from the left table even though there are no matches in right table
If ON Clause matches zero records in the right table, the joins still return a record in the result with NULL in each column from the right table

From the below screenshot, we can observe the following

Here we are performing join query using "LEFT OUTER JOIN" keyword between the tables sample_joins and sample_joins1 with matching condition as (c.Id= o.Id).
For example here we are using employee id as a reference, it checks whether id is common in right as well as left the table or not. It acts as matching condition.

The output displaying common records present in both the table by checking the condition mentioned in the query.
NULL values in the above output are columns with no values from Right table that is sample_joins1

Query:

SELECT c.Id, c.Name, o.Amount, o.Date1 FROM sample_joins c LEFT OUTER JOIN sample_joins1 o ON(c.Id=o.Id)

```
hive>  SELECT c.Id, c.Name, o.Amount, o.Date1 FROM sample_joins c
    > LEFT OUTER JOIN sample_joins1 o ON(c.Id=o.Id);
Query ID = hduser_20151105151057_1e2e2652-6f81-4a5e-b99c-29b15898
Total jobs = 1
Execution log at: /tmp/hduser/hduser_20151105151057_1e2e2652-6f81-
2015-11-05 15:11:02     Starting to launch local task to process
201     4     Dump the side-table for tag: 1 with group
5_1     1675585646620-1/-local-10003/HashTable-Stage-
201     4     Uploaded 1 File to: file:/tmp/hduser/7c10
-10     tage-3/MapJoin-mapfile11--.hashtable (422 bytes
201     4     End of local task; Time Taken: 2.115 sec.
Exe     d successfully
Mapr    succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operat
Starting Job = job_201511051442_0002, Tracking URL = http://locall
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop job
Hadoop job information for Stage-3: number of mapper
2015-11-05 15:11:12,191 Stage-3 map = 0%,  reduce =
2015-11-05 15:11:14,197 Stage-3 map = 100%,  reduce
2015-11-05 15:11:15,207 Stage-3 map = 100%,  reduce
MapReduce Total cumulative CPU time: 860 msec
Ended Job = job_201511051442_0002
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 0.86 sec    DFS Read: 712
Total MapReduce CPU Time Spent: 860 msec
OK
101     Rajesh  NULL    NULL
102     Rajiv   NULL    NULL
103     Animesh 2000.0  2015-10-08 00:00:00
104     Anirudh NULL    NULL
105     Santosh 1500.0  2015-10-15 00:00:00
106     Ramesh  NULL    NULL
107     Sravanthi       3200.0  2014-12-25 00:00:00
108     Sravan  NULL    NULL
109     Suresh  2000.0  2012-11-02 00:00:00
110     Ravi    NULL    NULL
111     Sravani NULL    NULL
112     Kiran   1800.0  2014-12-08 00:00:00
113     Kishore NULL    NULL
114     Prakash NULL    NULL
115     Dheeraj 2500.0  2013-12-18 00:00:00
Time taken: 17.759 seconds, Fetched: 15 row(s)
```

1

left outer join query

left outer join output

2

Right outer Join:

Hive query language RIGHT OUTER JOIN returns all the rows from the Right table even though there are no matches in left table
If ON Clause matches zero records in the left table, the joins still return a record in the result with NULL in each column from the left table
RIGHT joins always return records from a Right table and matched records from the left table. If the left table is having no values corresponding to the column, it will return NULL values in that place.

From the below screenshot, we can observe the following

Here we are performing join query using "RIGHT OUTER JOIN" keyword between the tables sample_joins and sample_joins1 with matching condition as (c.Id= o.Id).
The output displaying common records present in both the table by checking the condition mentioned in the query
Query:

```
SELECT c.Id, c.Name, o.Amount, o.Date1 FROM sample_joins c RIGHT OUTER JOIN sample_joins1 o ON(c.Id=o.Id)
```

```
hive>  SELECT c.Id, c.Name, o.Amount, o.Date1 FROM sample_joins c
    >  RIGHT OUTER JOIN sample_joins1 o ON(c.Id=o.Id);
Query ID = hduser_20151105152248_d9[illegible]f8-dd32-4af8-8cda-919ffac22
Total jobs = 1
Execution log at: /tmp/hduser/hduser_201511[illegible]248_d90e7bf8-dd32-
2015-11-05 15:22:55     Starting to launc[illegible]ess m
2015-11-05 15:22:55     Dump the side-tab[illegible]roup
05_15-22-48_548_8469128603837222403-1/-lo[illegible]Stage
2015-11-05 15:22:55     Uploaded 1 File t[illegible]7c101
1-10003/HashTable-Stage-3/MapJoin-mapfile[illegible]bytes
2015-11-05 15:22:55     End of local task,[illegible]sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operato
Starting Job = job_201511051442_0003, Tracking URL = http://localh
Kill Command = /usr/local/hadoop-[illegible]n/hadoop job
Hadoop job information for Stage-[illegible]s: 1; number o
2015-11-05 15:23:04,190 Stage-3 m[illegible]0%
2015-11-05 15:23:14,218 Stage-3 m[illegible] = 0%, Cumulati
2015-11-05 15:23:15,221 Stage-3 m[illegible] = 100%, Cumula
MapReduce Total cumulative CPU ti[illegible]
Ended Job = job_201511051442_0003
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1  Cumulative CPU: 0.74 sec   HDFS Read: 685[illegible]
Total MapReduce CPU Time Spent: 740 msec
OK
103     Animesh 2000.0  2015-10-08 00:00:00
105     Santosh 1500.0  2015-10-15 00:00:00
109     Suresh  2000.0  2012-11-02 00:00:00
112     Kiran   1800.0  2014-12-08 00:00:00
115     Dheeraj 2500.0  2013-12-18 00:00:00
107     Sravanthi       3200.0  2014-12-25 00:00:00
Time taken: 26.834 seconds, Fetched: 6 row(s)
```

1

Right outer join query

2

Right outer join output

Full outer join:

It combines records of both the tables sample_joins and sample_joins1 based on the JOIN Condition given in query.

It returns all the records from both tables and fills in NULL Values for the columns missing values matched on either side.

From the below screen shot we can observe the following:

Here we are performing join query using "FULL OUTER JOIN" keyword between the tables sample_joins and sample_joins1 with matching condition as (c.Id= o.Id).
The output displaying all the records present in both the table by checking the condition mentioned in the query. Null values in output here indicates the missing values from the columns of both tables.
Query

SELECT c.Id, c.Name, o.Amount, o.Date1 FROM sample_joins c FULL OUTER JOIN sample_joins1 o ON(c.Id=o.Id)

```
hive>  SELECT c.Id, c.Name, o.Amount, o.Date1 FROM sample_joins c
    > FULL OUTER JOIN sample_joins1 o ON(c.Id=o.Id);
Query ID = hduser_20151105152525_c7bd728a-51ee-4d3d-8a36-6e030947
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data s
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.
In order to limit the max
  set hive.exec.reducers.
In order to set a constant
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0004, Tracking URL = http://local
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/hadoop job
Hadoop job information for Stage-1: number of mappers: 2; number
2015-11-05 15:25:34,585 Stage-1 map = 0%,  reduce = 0%
2015-11-05 15:25:37,594 Sta  e = 0%, Cumulat
2015-11-05 15:25:44,651 St  e = 33%, Cumulat
2015-11-05 15:25:45,657 St  e = 100%, Cumula
MapReduce Total cumulative  msec
Ended Job = job_20151105144
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 2.56 sec   HDF
Total MapRedu CPU time Spent: 2 seconds 560 msec
OK
101     Rajesh  NULL    NULL
102     Rajiv   NULL    NULL
103     Animesh 2000.0  2015-10-08 00:00:00
104     Anirudh NULL    NULL
105     Santosh 1500.0  2015-10-15 00:00:00
106     Ramesh  NULL    NULL
107     Sravanthi       3200.0  2014-12-25 00:00:00
108     Sravan  NULL    NULL
109     Suresh  2000.0  2012-11-02 00:00:00
110     Ravi    NULL    NULL
111     Sravani NULL    NULL
112     Kiran   1800.0  2014-12-08 00:00:00
113     Kishore NULL    NULL
114     Prakash NULL    NULL
115     Dheeraj 2500.0  2013-12-18 00:00:00
Time taken: 21.113 seconds, Fetched: 15 row(s)
```

1

Full outer join query

Full outer join output

2