**Hive**

* 1. **What is a map join in hive**

If one of the tables in the join query is small enough to fit in memory, we recommend you to use map join. Map join is a special type of join where the smaller table is loaded in memory and the join is performed in the map phase of the MapReduce job. Since no reducers are required, map joins are much faster compared to regular joins.  
  
hive> SELECT /\*+ MAPJOIN(t1) \*/

t1.id, t1.name, t1.age, t2.name, t2.age FROM

t1 JOIN t2 ON (t1.id = t2.id)

In the above example, if t1 is small enough to fit in memory, the map join query would look like:

|  |  |
| --- | --- |
|  |  |

Starting Hive 0.7, there is no need to specify the query hint (i.e. /\*+ MAPJOIN(t1) \*/) for map joins. Set the **hive.auto.convert.join** property to true in hive-site.xml or your Hive CLI session, and Hive will use a conditional task to determine if it is possible to run a map join instead of a regular join. If so, it will run a map-join; if not, a regular join is run.  
  
Another (better, in my opinion) way to turn on mapjoins is to let Hive do it automatically. Simply set hive.auto.convert.join to true in your config, and Hive will automatically use mapjoins for any tables smaller than hive.mapjoin.smalltable.filesize (default is 25MB).

Mapjoins have a limitation in that the same table or alias cannot be used to join on different columns in the same query. (This makes sense because presumably Hive uses a HashMap keyed on the column(s) used in the join, and such a HashMap would be of no use for a join on different keys).

The workaround is very simple - do not use the same aliases in your query.

* 1. **UDF?**
* package com.hive.example.util;  
    
  import java.util.Date;  
  import java.text.DateFormat;  
    
  import org.apache.hadoop.hive.ql.exec.UDF;  
  import org.apache.hadoop.io.Text;  
    
  public class UnixtimeToDate extends UDF{  
      public Text evaluate(Text text) {  
          if(text == null) return null;  
          long timestamp = Long.parseLong(text.toString());  
          return new Text(toDate(timestamp));  
      }  
        
      private String toDate(long timestamp) {  
          Date date = new Date (timestamp \* 1000);  
          return DateFormat.getInstance().format(date).toString();  
      }  
  }
* Pack this class file into a jar:
* $jar -cvf convert.jar com.hive.example.util.UnixtimeToDate
* Verify jar using command : *$jar -tvf convert.jar*
* add this jar in hive prompt
* hive>create temporary function userdate as 'com.hive.example.util.UnixtimeToDate';
* Example:
* Normally without the function query:
* hive>select id, unixtime from table;
* 12     879959583
* Then use function 'userdate' in sql command
* hive>select id, userdate(unixtime) from table;
* 12     19/11/97 10:43 PM

**package org.learn.hive;**  
 **import org.apache.hadoop.hive.ql.exec.UDF;**  
**import org.apache.hadoop.io.Text;**  
 **class ToUpper extends UDF {**  
  
**public Text evaluate(Text input) {**  
**if(input == null) return null;**  
**return new Text(input.toString().toUpperCase());**  
**}**  
**}**  
  
  
7. Now export this project as jar file and name it hive-to-upper-udf.jar.  
  
8. copy this jar file in <hive-installation-dir>/lib/ directory.  
  
9. Now go to hive shell and type following command.  
  
**ADD JAR /home/hduser/hive/lib/ hive-to-upper-udf.jar;**  
 **CREATE TEMPORARY FUNCTION toUpper as 'org.learn.hive.ToUpper';**  
  
 **select toUpper(name) from user\_table limit 1000;**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**3. Distinct- how is it used in hive? Is there any other way to get the ‘distinct’ done without using the key word?**select count(distinct user)

from some\_table;

This query does the count on the map side. Each mapper emits one value, the count. Then all values have to be aggregated to produce the total count, and that is the job of one single reducer.

select count(\*) from

(select user

from some\_table

group by user) q;

This query has two stages. On stage 1 the GROUP BY aggregates the users on the map side and emits one value *for each user*. The output has to be aggregated then on the reduce side, **but it can use many reducers**. On stage 2 the the COUNT is performed, on the map side, and then the final result is aggregated using one single reducer.

So if you have a very large number of map side splits then the first query will have to aggregate a very large number of one value results. The second query can use many reducers at the reduce side of stage 1 and then, at stage 2, will have a smaller task for the lone reducer at the end.

This would normally not be an optimization. You would have to have a significant number of map splits for the query 1 reducer to become a problem. The second query has two stages and that alone would be slower than query 1 (stage 2 cannot start until stage 1 is completely done). So, while I can see*some* reasoning for the advice you got, I would be skeptical unless proper measurement is done and shows improvement.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This query looks familier to SQL users, but this query is very slow because only one reducer is used to process the request.

SELECT count(DISTINCT field) FROM tbl

So please rewrite the query like below to leverage multiple reducers.

SELECT

count(1)

FROM (

SELECT DISTINCT field FROM tbl

) t

**4.What is metastore?**  
Hive is query engine.

Once you install Hive it comes along with in memory database called Derby is a RDBMS.

Hive’s inmemory is Derby.

Hive use to store meta data of a table .Actual data will be stored in HDFS but metdata of the table will be stored in Derby i.e. schema  
Derby is not centralized. Derby cannot be used as master slave architecture

You cannot take backup of your meta store because it’s in-memory and not cetralized.  
That’s why mysql came.

There are 3 types of metastore service :

* 1. Embedded :: if you use Derby as metastore
  2. Local :: If you are configuring MySQL as metastore and mysql reside in the same machine whre Hive installed.
  3. Distributed:: If mysql is installed in different machine and storing the meatstore over there.

Create hive-site.xml file in $HIVE\_HOME/conf directory and add the following configurations:

<configuration>

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value>jdbc:mysql://localhost/metastore\_db?createDatabaseIfNotExist=true</value>

<description>metadata is stored in a MySQL server</description>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

<description>MySQL JDBC driver class</description>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>hiveuser</value>

<description>user name for connecting to mysql server </description>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>hivepassword</value>

<description>password for connecting to mysql server </description>

</property>

**<property>**

**<name>hive.metastore.local</name>**

**<value>true</value>**

**</property>**

</configuration>

**Ref:: \_https://archanaschangale.wordpress.com/2013/09/05/changing-default-metastore-derby-of-hive-to-mysql/**

[Download](http://dev.mysql.com/downloads/connector/j) *mysql-connector-java-5.0.5.jar* file and copy it to $HIVE\_HOME/lib directory.  
  
hive> exit;

hive> bin/hive  
hive> show tables;  
hive> create table emp(eid INT, ename STRING) row format delimited fields terminated by ‘,’ store as textfile;

hive> load data local inpath ‘/home/ubutu/emp.txt’ into table emp;  
hive> select \* from emp;  
  
  
MySql console:

There are 2 ways to access metastore\_db

***1.      mysql -u root -p***

**Enter password:**

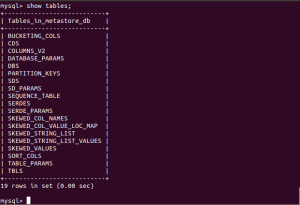
***mysql> use metastore\_db;***

***mysql> show tables ;***

***2.     mysql -u hiveuser -p metastore\_db***

**Enter password:**

***mysql> show tables;***

[](https://archanaschangale.files.wordpress.com/2013/09/metastore.png)

You can query the metastore schema in your MySQL database. Something like:

***mysql> select \* from TBLS;***

**Where we can get the information of a metastore.**

**What is parameter in hive-site.xml which corresponds to metastore configuration?  
mentioned above**

**If you have 10 different user and you want to change the metastore for one user, how you can do that?**

HIVE\_CONF\_DIR

if [ -f "${HIVE\_CONF\_DIR}/hive-env.sh" ]; then

  . "${HIVE\_CONF\_DIR}/hive-env.sh"

fi

* Create the Database and User

Create a metastore\_db database in MySQL database using root user

***$ mysql -u root -p***

**Enter password:**

***mysql> CREATE DATABASE metastore\_db;***

Create a User [hiveuser] in MySQL database using root user

***mysql> CREATE USER 'hiveuser'@'%' IDENTIFIED BY 'hivepassword';***

***mysql> GRANT all on \*.\* to 'hiveuser'@localhost identified by 'hivepassword';***

[here the user **hiveuser** is same as ConnectionUserName in hive-site.xml file.]

***mysql>  flush privileges;***

**Suppose you have data from Jan to Dec and one month data is corrupted, what strategies would you use so that your existing clean data need not be uploading again?**

Use partition to achieve this.

-- Set following two properties for your Hive session:

SET hive.exec.dynamic.partition=true;

SET hive.exec.dynamic.partition.mode=nonstric;

-- Create an Historical table with partition as below

hive> create table Historical (Id int, Date String) partitioned by (year\_part string, month\_no int) row format delimited fields terminated by ',';

--Load the Data into Historical table and insert from Results table such that data is partitioned in Historical table depending upon the year from date and dynamically found week number based on the date in Results table. --Do make sure that the column on which you want to partition should come last in select statements. If there are series of column then there order in partition(col3,col4) should match in select statement

hive> insert overwrite table Historical partition(year\_part, month\_no) select id, date, year(date) as year\_part, month(date) as month\_no from Results;

\_Ref:: \_http://ksssblogs.blogspot.in/2014/03/dynamic-partition-inserts-in-hive.html  
*INSERT OVERWRITE TABLE invoice\_details\_hive\_partitioned PARTITION(pay\_location=’USA’)*

*SELECT idh.Invoice\_Id, idh.Invoice\_Date, idh.Invoice\_Amount, idh.Paid\_Date FROM invoice\_details\_hive idh WHERE pay\_location=’USA’;*

load data local inpath ‘/home/local/bang’ into table emp

* + parttion(loc=’Bangalore’)

**What is dynamic partitioning?**

<property>

<name>hive.exec.dynamic.partition</name>

<value>true</value>

<description>Whether or not to allow dynamic partitions in DML/DDL.</description>

</property>

<property>

**<name>hive.exec.dynamic.partition.mode</name>**

**<value>nonstrict</value>**

**<description>**

**In strict mode, the user must specify at least one static partition**

**in case the user accidentally overwrites all partitions.**

**In nonstrict mode all partitions are allowed to be dynamic.**

**</description>**

**</property>**

<property>

<name>hive.exec.max.dynamic.partitions</name>

<value>1000</value>

<description>Maximum number of dynamic partitions allowed to be created in total.</description>

</property>

<property>

<name>hive.exec.max.dynamic.partitions.pernode</name>

<value>1000</value>

<description>Maximum number of dynamic partitions allowed to be created in each mapper/reducer node.</description>

</property>

<property> **Ref::\_** **http://hadooptutorial.info/partitioning-in-hive/**

**What is the restriction on dynamic partitioning (strict mode - non-strict mode?).**

**Problem:** I have a huge table in my EDW that holds 5 billion rows. Every record has a column pay\_location which has 300 distinct values across the table. Need to do some processing on the same within Hadoop environment and at a time my processing involves data only from certain pay\_locations.

Table Schema in DWH

|  |  |
| --- | --- |
| **Invoice\_Details** | |
| ***Column Name*** | ***Data Type*** |
| Invoice\_Id | double |
| Invoice\_Date | Date |
| Invoice\_Amount | double |
| Pay\_Location | VarChar |
| Paid\_Date | Date |

**Solution:** We can accomplish the same in 2 easy steps

Step 1: SQOOP import the table into hive from DWH

Step 2: Analyze the data using Hive QL

                Fist we need to SQOOP import the data into hive table ‘*invoice\_details’*using the basic SQOOP import command as follows.

sqoop import --driver *<driver name>* --connect *<connection string>* --username*<username>* -P --table *Invoice\_Details* --split-by *Invoice\_Id* --num-mappers *<num of mappers>* --warehouse-dir *<hdfs dir>* --hive-import --hive-table *invoice\_details\_hive*

We’d look into the second part here in more detail. How to effectively and efficiently analyze the same in hive. In our requirement it is clear that we can go ahead with a partitioned table approach for our data set, as the data analysis is made pay\_location by pay\_location we can do the partition based on pay\_location itself.

Now how can we do the partition? Simple, need to create a partitioned table and load data into each partition. So first we can create an equivalent partitioned table in hive

*CREATE TABLE invoice\_details\_hive\_partitioned(Invoice\_Id double, Invoice\_Date string, Invoice\_Amount double,Paid\_Date string)PARTITIONED BY(pay\_location string);*

Once table creation is completed we need to load data into the partitions on invoice\_details\_hive \_partitioned from invoice\_details\_hive . How can we do it? Can we go ahead for individual insert statements for each pay\_location like?

*INSERT OVERWRITE TABLE invoice\_details\_hive\_partitioned PARTITION(pay\_location=’USA’)*

*SELECT idh.Invoice\_Id, idh.Invoice\_Date, idh.Invoice\_Amount, idh.Paid\_Date FROM invoice\_details\_hive idh WHERE pay\_location=’USA’;*

If we follow this approach we may have to go in for 300 insert statements as there are 300 distinct values for pay\_location in invoice\_details\_hive table. **This type of implementation can be called as STATIC PARTIONS.** But in our scenario static partitions won’t serve the purpose or rather it is too tedious. We’d have to implement the concept of **DYNAMIC PARTITIONS introduced from hive 0.6 onwards**. With Dynamic partitions we just need a single Insert Overwrite statement to create and load data into all the partitions.

*INSERT OVERWRITE TABLE invoice\_details\_hive\_partitioned PARTITION(pay\_location)*

*SELECT idh.Invoice\_Id, idh.Invoice\_Date, idh.Invoice\_Amount, idh.Paid\_Date, idh.pay\_location FROM invoice\_details\_hive idh;*

This Single Query would implement dynamic partition for you, when you use dynamic partitions the last column from the select query on the source table should be column used for partitioning in the destination table (*idh.pay\_location*)

When you try executing the query you can see hive throwing some fatal errors, like dynamic partition mode is strict and dynamic partition not enabled. So we need to set the following parameters in hive shell

*1.*       *set hive.exec.dynamic.partition=true;*

To 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.**

*3.*       *set hive.exec.max.dynamic.partitions.pernode=300;*

The default value is 100, we have to modify the same according to the possible no of partitions that would come in your case

*4.*       *set hive.exec.max.created.files=150000;*

The default values is 100000 but for larger tables it can exceed the default, so we may have to update the same

In practical scenarios I did find the Dynamic Partition not working with the above query on really large tables and shooting a java print error after completion of first map process. This could be due to the larger number of files created on the first map process. However a slight modification of the job can help you overcome the same, group the records in your hive query on the map process and process them on the reduce side. ie use a map reduce process to achieve your goal rather than two map process. You can implement the same in your hive query itself with the usage of DISTRIBUTE BY, so the modified query would be

FROM invoice\_details\_hive idh

INSERT OVERWRITE TABLE invoice\_details\_hive\_partitioned PARTITION(pay\_location)

SELECT idh.Invoice\_Id, idh.Invoice\_Date, idh.Invoice\_Amount, idh.Paid\_Date, idh.pay\_location

DISTRIBUTE BY pay\_location;

With this approach you don’t need to overwrite the hive.exec.max.created.files parameter.

**Default value per node of number of dynamic partition supported.**

*set hive.exec.max.dynamic.partitions.pernode=300;*

The default value is 100, we have to modify the same according to the possible no of partitions that would come in your case

Lets consider TCS employee data . I want to fetch the employees whose salary above 5 lakh in Bangalore.

Select \* from emp where loc=’Bangalore’;

To avoid the query time we can partition the data

TCS

|----- Bangalore

|----- Kolkata

Now once the data is partitioned. The query will only be applied to Bangalore partition only not in Kolkata partition.  
Partition Bangalore can also be taken as a colum in the table.  
  
Let’s create partition:

Create table emp (ipaddr STRING, name STRING)

* + partitioned by (loc STRING)
  + row format delimited
  + fields terminated by ‘\t’

load data local inpath ‘/home/local/bang’ into table emp

* + parttion(loc=’Bangalore’)

TCS

|----- loc = Bangalore

|---- Bang named file

When table is partitioned then hive will not generate mapreduce job.

When table needs to be full scan then mapreduce job will not be genearated

Select \* from table --🡪 It will not generate full table scan

Parttioning is physical.

\_ <http://hadooptutorial.info/partitioning-in-hive/>

\_ <http://kickstarthadoop.blogspot.in/2011/06/how-to-speed-up-your-hive-queries-in.html>

**What is rlike?**

**LIKE and RLIKE**

You have probably seen LIKE before, a standard SQL operator. For example, the following three queries select the employee names and addresses where the street ends with Ave., the city begins with O, and the street contains Chicago:

|  |  |
| --- | --- |
| 1  2  3 | hive> SELECT name, address.street FROM employees WHERE address.street LIKE '%Ave.';  John Doe 1 Michigan Ave.  Todd Jones 200 Chicago Ave. |

|  |  |
| --- | --- |
| 1  2  3 | hive> SELECT name, address.city FROM employees WHERE address.city LIKE 'O%';  Todd Jones Oak Park  Bill King Obscuria |

|  |  |
| --- | --- |
| 1  2 | hive> SELECT name, address.street FROM employees WHERE address.street LIKE '%Chi%';  Todd Jones 200 Chicago Ave. |

A Hive extension is the RLIKE clause, which lets us use Java regular expressions, a more powerful minilanguage for specifying matches.

|  |  |
| --- | --- |
| 1  2  3  4  5 | hive> SELECT name, address.street  > FROM employees WHERE address.street RLIKE '.\*(Chicago|Ontario).\*';    Mary Smith 100 Ontario St.  Todd Jones 200 Chicago Ave. |

The string after the RLIKE keyword has the following interpretation. A period . matches any character and a star \* means repeat the “thing to the left” (period, in the two cases shown) zero to many times. The expression x|y means match either x or y. Hence, there might be no characters before “Chicago” or “Ontario” and there might be no characters after them. Of course, we could have written this particular example with two LIKE clauses:

|  |  |
| --- | --- |
| 1  2 | SELECT name, address FROM employees  WHERE address.street LIKE '%Chicago%' OR address.street LIKE '%Ontario%'; |

**External table and manage table.**

**Check above.**

**What if you have to create a manage table from external table, what would happen to the data (data will move or copy).**

\_http://www.slideshare.net/PrashantPandey37/hive-tutoloading-data-into-hive-tables

CREATE EXTERNAL TABLE SALESVER2\_MANAGED(STOREID INT,SMONTH INT, SALES FLOAT,TIME TIMESTAMP)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

hive> insert overwrite table salesver2\_managed select \* from salesver2\_external;

Total MapReduce jobs = 3

Launching Job 1 out of 3

Number of reduce tasks is set to 0 since there's no reduce operator

Starting Job = job\_201504211610\_0002, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job\_201504211610\_0002

Kill Command = /usr/local/hadoop/bin/../bin/hadoop job -kill job\_201504211610\_0002

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0

2015-04-21 16:28:06,642 Stage-1 map = 0%, reduce = 0%

2015-04-21 16:28:09,669 Stage-1 map = 100%, reduce = 0%

2015-04-21 16:28:12,695 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_201504211610\_0002

Stage-4 is selected by condition resolver.

Stage-3 is filtered out by condition resolver.

Stage-5 is filtered out by condition resolver.

Moving data to: hdfs://localhost:54310/tmp/hive-ubuntu/hive\_2015-04-21\_16-28-01\_734\_2450912268369376436-1/-ext-10000

Loading data to table default.salesver2\_managed

Table default.salesver2\_managed stats: [num\_partitions: 0, num\_files: 1, num\_rows: 0, total\_size: 391, raw\_data\_size: 0]

MapReduce Jobs Launched:

Job 0: Map: 1 HDFS Read: 391 HDFS Write: 391 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

Time taken: 11.202 seconds

hive> select \* from salesver2\_managed;

OK

1001 1 35000.0 1985-09-25 15:32:30.008

1002 2 35000.0 2001-07-25 11:01:30.009

1001 2 25000.0 2001-11-25 07:20:30.005

1002 3 110000.0 1991-05-25 01:15:30.005

1001 3 40000.0 1988-05-25 23:45:30.005

1002 1 40000.0 1985-02-25 20:55:30.005

1004 1 48000.0 1989-02-25 15:35:30.005

1005 1 22000.0 2005-04-12 14:42:30.005

1006 2 41000.0 2004-08-14 15:35:30.005

1007 2 21000.0 2011-04-08 14:42:30.005

Time taken: 0.033 seconds, Fetched: 10 row(s)

hive> select \* from salesver2\_external;

OK

1001 1 35000.0 1985-09-25 15:32:30.008

1002 2 35000.0 2001-07-25 11:01:30.009

1001 2 25000.0 2001-11-25 07:20:30.005

1002 3 110000.0 1991-05-25 01:15:30.005

1001 3 40000.0 1988-05-25 23:45:30.005

1002 1 40000.0 1985-02-25 20:55:30.005

1004 1 48000.0 1989-02-25 15:35:30.005

1005 1 22000.0 2005-04-12 14:42:30.005

1006 2 41000.0 2004-08-14 15:35:30.005

1007 2 21000.0 2011-04-08 14:42:30.005

Time taken: 0.028 seconds, Fetched: 10 row(s)

Use EXTERNAL tables when:

* The data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files.
* Data needs to remain in the underlying location even after a DROP TABLE. This can apply if you are pointing multiple schemas (tables or views) at a single data set or if you are iterating through various possible schemas.
* You want to use a custom location such as ASV.
* Hive should not own data and control settings, dirs, etc., you have another program or process that will do those things.
* You are not creating table based on existing table (AS SELECT).

Use INTERNAL tables when:

* The data is temporary.
* You want Hive to completely manage the lifecycle of the table and data.

Load some data (hadoop file system put) and then verify it loaded (hadoop file system list recursively):   
  
root@mahesh:/home/mahesh/Desktop/Hadoop\_Guide/sample DB/bacon# hadoop fs -put bacon.txt /user/demo/food/  
   
 root@mahesh:/data# hadoop fs -ls /user/demo/food  
Warning: $HADOOP\_HOME is deprecated.  
  
Found 1 items  
-rw-r--r--   1 root supergroup        124 2013-05-05 18:27 /user/demo/food/bacon.txt

Enter the Hive

root@mahesh:/data# hive

Create an INTERNAL table in Hive and point it to the directory with the bacon.txt file:   
  
hive> create table internal1(col1 string) LOCATION '/user/demo/food';  
OK  
Time taken: 0.996 seconds

That will return the time taken but no other result. Now let's look at the schema that was created:. Note that the table type is MANAGED\_TABLE.

hive> describe formatted internal1;  
OK  
col\_name    data\_type    comment  
# col\_name                data\_type               comment              
            
col1                    string                  None                 
            
# Detailed Table Information            
Database:               employees                  
Owner:                  root                       
CreateTime:             Sun May 05 18:18:53 IST 2013       
LastAccessTime:         UNKNOWN                    
Protect Mode:           None                       
Retention:              0                          
Location:               hdfs://localhost:54310/user/demo/food       
Table Type:             MANAGED\_TABLE              
Table Parameters:            
    transient\_lastDdlTime    1367758133           
            
# Storage Information            
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe       
InputFormat:            org.apache.hadoop.mapred.TextInputFormat       
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat       
Compressed:             No                         
Num Buckets:            -1                         
Bucket Columns:         []                         
Sort Columns:           []                         
Storage Desc Params:            
    serialization.format    1                    
Time taken: 0.724 seconds  
   
 hive> select \* from internal1;  
OK  
HDInsight\_Bacon  
SQL\_Bacon  
PASS\_bacon  
Summit\_BACON  
Database\_Bacon  
NoSQL\_Bacon  
BigData\_Bacon  
Hadoop\_Bacon  
Hive\_Bacon  
Time taken: 1.25 seconds

What happens if we don't specify a directory for an INTERNAL table?

**hive> CREATE TABLE internaldefault(col1 string);**

It is created in the default Hive directory, which by default is in /hive/warehouse (dfs shells back out to Hadoop fs):

**dfs -lsr /user/hive/warehouse;**

*We can see that Hive has created a subdirectory with the same name as the table. If we were to load data into the table Hive would put it in this directory: drwxr-xr-x   - root supergroup          0 2013-05-05 18:37 /user/hive/warehouse/employees.db/internaldefault*

However, we won't use this table for the rest of the demo so let's drop it to avoid confusion. The drop also removes the subdirectory.

**DROP TABLE internaldefault;**

**dfs -lsr /user/hive/warehouse;**

Once we dropped the internaldefault table the directory that Hive created was automatically cleaned up. Now let's add a 2nd file to the first internal table and check that it exists:

root@mahesh:/home/mahesh/Desktop/Hadoop\_Guide/sample DB/bacon# hadoop fs -put bacon2.txt /user/demo/food/  
  
hive> dfs -ls /user/demo/food;                    
Found 2 items  
-rw-r--r--   1 root supergroup        124 2013-05-05 18:27 /user/demo/food/bacon.txt  
-rw-r--r--   1 root supergroup         31 2013-05-05 18:47 /user/demo/food/bacon2.txt

Since the CREATE TABLE statement points to a directory rather than a single file any new files added to the directory are immediately visible (remember that the column name col1 is only showing up because we enabled showing headers in the output - there is no row value of col1 in the data as headers are not generally included in Hadoop data):    
  
hive> select \* from internal1;  
OK  
HDInsight\_Bacon  
SQL\_Bacon  
PASS\_bacon  
Summit\_BACON  
Database\_Bacon  
NoSQL\_Bacon  
BigData\_Bacon  
Hadoop\_Bacon  
Hive\_Bacon  
More\_BaCoN  
AndEvenMore\_bAcOn  
Time taken: 0.457 seconds

Now let's create an EXTERNAL table that points to the same directory and look at the schema:

hive> CREATE EXTERNAL TABLE extenal1(col1 STRING) LOCATION '/user/demo/food';  
OK  
Time taken: 0.306 seconds

hive> DESCRIBE FORMATTED external1;                                             
OK  
# col\_name                data\_type               comment               
            
col1                    string                  None                  
            
# Detailed Table Information            
Database:               employees                  
Owner:                  root                       
CreateTime:             Sun May 05 19:13:36 IST 2013       
LastAccessTime:         UNKNOWN                    
Protect Mode:           None                       
Retention:              0                          
Location:               hdfs://localhost:54310/user/demo/food       
Table Type:             EXTERNAL\_TABLE             
Table Parameters:            
    EXTERNAL                TRUE                  
    transient\_lastDdlTime    1367761416            
            
# Storage Information            
SerDe Library:          org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe       
InputFormat:            org.apache.hadoop.mapred.TextInputFormat       
OutputFormat:           org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat       
Compressed:             No                         
Num Buckets:            -1                         
Bucket Columns:         []                         
Sort Columns:           []                         
Storage Desc Params:            
    serialization.format    1                     
Time taken: 0.163 seconds

This time the table type is EXTERNAL\_TABLE. You can see that the location was expanded to include the default settings which in this case are the localhost machine using the default HDFS

View the Table data,

hive> SELECT \* FROM external1;  
OK  
HDInsight\_Bacon  
SQL\_Bacon  
PASS\_bacon  
Summit\_BACON  
Database\_Bacon  
NoSQL\_Bacon  
BigData\_Bacon  
Hadoop\_Bacon  
Hive\_Bacon  
More\_BaCoN  
AndEvenMore\_bAcOn  
  

You may create multiple tables for the same data set if you are experimenting with various structures/schemas.

Add another data file to the same directory and see how it's visible to all the tables that point to that directory:

root@mahesh:/home/mahesh/Desktop/Hadoop\_Guide/sample DB/bacon# hadoop fs -put veggies.txt /user/demo/food/

**SELECT \* FROM internal1;**  
**SELECT \* FROM external1;**  
**SELECT \* FROM external2;**

Each table will return the same results:

*HDInsight\_Bacon   
SQL\_Bacon   
PASS\_bacon   
Summit\_BACON   
Database\_Bacon   
NoSQL\_Bacon   
BigData\_Bacon   
Hadoop\_Bacon   
Hive\_Bacon   
More\_BaCoN   
AndEvenMore\_bAcOn   
SQL\_Apple   
NoSQL\_Pear   
SQLFamily\_Kiwi   
Summit\_Mango   
HDInsight\_Watermelon   
SQLSat\_Strawberries   
Raspberrylimelemonorangecherryblueberry 123 456*

Now drop the INTERNAL table and then look at the data from the EXTERNAL tables which now return only the column name:

**DROP TABLE internal1;**

**SELECT \* FROM external1;**

**h\ive> dfs -lsr /user/demo/food;  
lsr: Cannot access /user/demo/food: No such file or directory.**

Because the INTERNAL (managed) table is under Hive's control, when the INTERNAL table was dropped it removed the underlying data. The other tables that point to that same data now return no rows even though they still exist!

Clean up the demo tables and directory:

**DROP TABLE external1;**  
  
**exit;**

This should give you a very introductory level understanding of some of the key differences between INTERNAL and EXTERNAL Hive tables. If you want full control of the data loading and management process, use the EXTERNAL keyword when you create the table.

**SAMPLE FILES**

**Create new text file named bacon.txt , add the following content**

HDInsight\_Bacon   
SQL\_Bacon   
PASS\_bacon   
Summit\_BACON   
Database\_Bacon   
NoSQL\_Bacon   
BigData\_Bacon   
Hadoop\_Bacon   
Hive\_Bacon

**Create another new text file name bacon2.txt , add the following content**

More\_BaCoN   
**AndEvenMore\_bAcOn**

**Veggies.txt**

SQL\_Apple   
NoSQL\_Pear   
SQLFamily\_Kiwi   
Summit\_Mango   
HDInsight\_Watermelon   
SQLSat\_Strawberries

**Can we change the table name, how?**

ALTER TABLE old\_table\_name RENAME TO new\_table\_name;

**Can we change column name, how?**

hive> alter table test change userVisit userVisit2 STRING;

OK

Time taken: 0.26 seconds

hive> describe test;

OK

uservisit2 string

category string

uuid string

Time taken: 0.213 seconds, Fetched: 3 row(s)

**Map join how that works, is there any setting that we need to do or not?**

Use small tables on the left side of your joins and enable auto optimisation:

set hive.auto.convert.join=true;

This will cache small enough tables and allow for mapper-only joins which are much faster. You can also tweak the size of what HIve considers small enough to cache:

set hive.mapjoin.smalltable.filesize=50000000;

Make sure your JVM has enough memory though.

**Something on serde?**SerDe is short for Serializer/Deserializer. Hive uses the SerDe interface for IO. The interface handles both serialization and deserialization and also interpreting the results of serialization as individual fields for processing.

A SerDe allows Hive to read in data from a table, and write it back out to HDFS in any custom format. Anyone can write their own SerDe for their own data formats.

Whenever data needs to be read and write from Hive table we have to use the serde.

So whenever you are firing select query hive is using the inputformat based on fileformat. Eg. your file format is text input format. So it will use. textinputformat . Hive will read textformat and give it to record reader. That recordRead will give you the record. This record is actually the serialized record.  
This serialized record are stored in Hive and this serialized record need to be converted into actual row so that Hive can represent the data and use it. So whenever you fire the select query record goes to the serde.deserialize method with the help of object inspector class. This objectInspector class is for entire row or each and individual field.So it does the mapping for each field and deserialize the field and give it to end user.  
  
So whenevr you fire insert query to insert file in Hive table, hive loads the record & convert the record into Javaobject which it can understand. This object will be given to the SerDe.Serialized method.During serialization method what it does is it will seralize each and every object/field with the help of objecrInspector. After serialization it will be stored in Hive Database table.

So steps are :

* 1. Initialization for entire record for once and it is done for only one time. Because it needs to find table properties during intialization phase.like column type, name
  2. Define Deserialization method
  3. Serialize method

We have to use custom serde for semistructured(xml) and unstructured(video) data  
Sample usage  
add jar path/to/csv-serde.jar;

create table my\_table(a string, b string, ...)

row format serde 'com.bizo.hive.serde.csv.CSVSerde'

stored as textfile

;

**https://github.com/ogrodnek/csv-serde/blob/master/src/main/java/com/bizo/hive/serde/csv/CSVSerde.java**

**Rc file format?**

Hive uses by default RCFileformat which is a columner thing. This is more advantageous compare to other file format.  
  
In normal RDBMS case whe you try to read say 4 column value from DB then generally normal DB read entire row then discard the columns which are not require and return the remaining columns.  
Compression is not possible. It’s took Higer I/O.

RC file format advantage –

* 1. Fast data loading
  2. Fast query processing
  3. Efficient Storage Space

Consider a file/table has 5 records and each row has 5 columns .

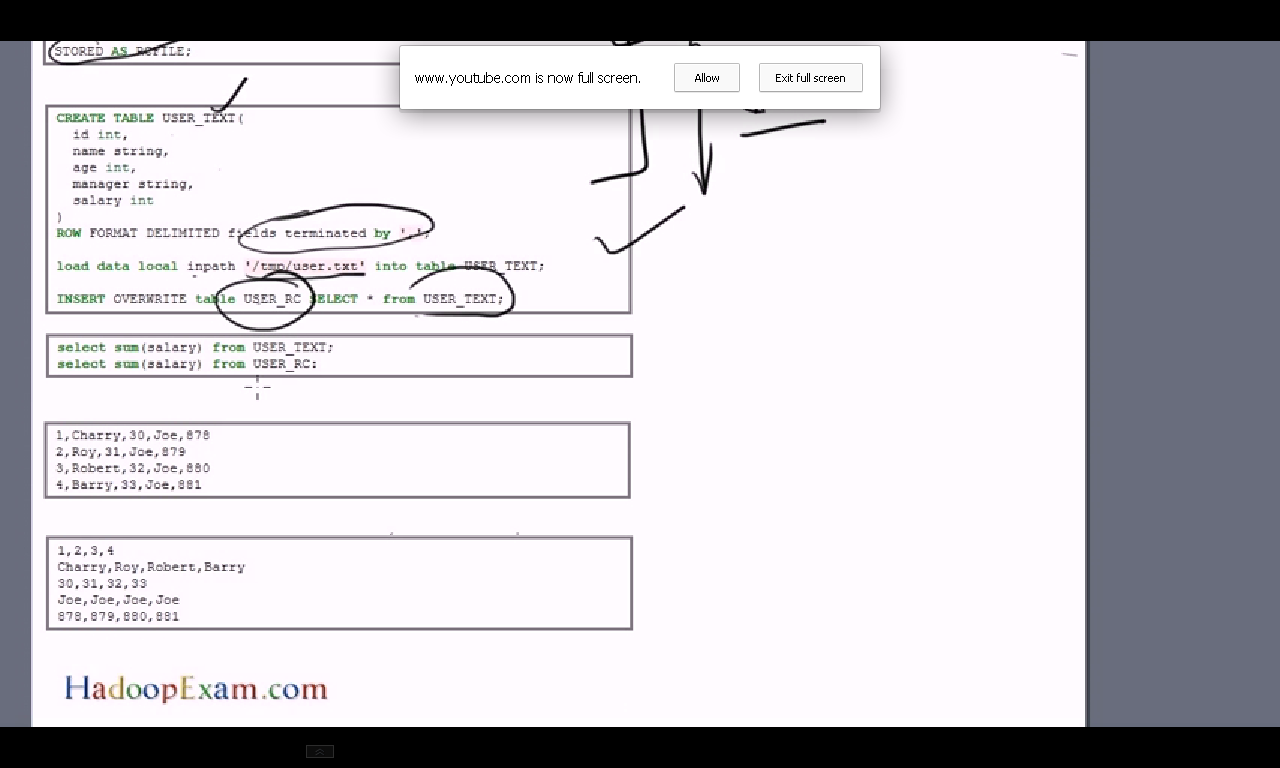
In RC file format horizontal as well vertical partitioning is done.  
In Horizontal partitioning eg 3 rows will go in one partiontion and 2 will go to another partition.

In Vertical partitioning eg 3 columns will go in one partiontion and 2 will go to another partition.  
  


USER\_RC is RC file example and USER\_TEXT is nonRC file.

There is a huge difference in data read when u use RC and nonRC table. In HDFS there is a property call HDFS\_Bytes\_read in log.

You can execute hive query how much data is read when u use RC file and NonRC file



In case of USER\_TEXT file when we want to get sum of my salary, it reads entire rows of the tuple discard all the 4 columns which are not needed. Then do the summation off the salary. Then return the record.

In case of RC (Consider horizontal file format), it convert entire column into row. So when we need sum of salary it does not need to read entire touple. It just go to the particular row pickup all the value do the summation.

Use RC file when table has huge column and few coluumns need to be access.

**how to restrict your query if you have to see some specific records?**

Consider the CUSTOMERS table having the following records:

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Following is an example on SQL server, which would fetch top 3 records from CUSTOMERS table:

SQL> SELECT TOP 3 \* FROM CUSTOMERS;

This would produce the following result:

+----+---------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+---------+-----+-----------+---------+

**Problem:**

Using limit might give the same set of rows every time you run it, if you need a true random records every time you execute the query you need to use rand() function.

**Solution:**

hive> select dept\_no  
> from departments  
> order by rand() limit 5;

**dept\_no**  
d008  
d004  
d003  
d007  
d005

hive> select dept\_no  
> from departments  
> order by rand() limit 5;

**dept\_no**  
d001  
d002  
d003  
d004  
d005

**Discussion:**  
Using rand() along with limit, we can generate a random sequence of records and also a sample.

Let’s say you have a Hive table with ten billion rows, but you want to efficiently randomly sample a fixed number- maybe ten thousand. The most obvious (and obviously wrong) way to do this is simply:

**select** \* **from** my\_table  
limit 10000;

Hive makes no guarantees about the order of records if you don’t sort them, but in practice, they come back in the same order in which they’re in the file, so this is far from truly random. As a next attempt:

**select** \* **from** my\_table  
order **by** rand()  
limit 10000;

This does actually give you truly random data, but performance is not so good. In order to impose a total ordering, Hive must force all data to one single reducer. That reducer will sort the entire dataset. Not good. Luckily, Hive has a non-standard-SQL “sort by” clause that only sorts within a single reducer, and gives no guarantees that data will be sorted across reducers:

**select** \* **from** my\_table  
sort **by** rand()  
limit 10000;

This is much better, but I’m not convinced it’s truly random. The issue is that Hive’s method of splitting data into multiple reducers is undefined. It might be truly random, it might be based on file order, it might be based on some value in the data. How Hive implements the limit clause across reducers is also undefined. Maybe it takes data from the reducers in order- i.e., all data from reducer 0, then all from reducer 1, etc. Maybe it round-robins through them and mixes everything together.

http://www.joefkelley.com/?p=736

**Extended describe.**describe extended employees;  
  
  
name    string  
salary  float  
subordinates    array<string>  
deductions      map<string,float>  
address struct<street:string,city:string,state:string,zip:int>  
  
Detailed Table Information   
    
Table  
(  
  tableName:employees  
, dbName:saurav  
, owner:training  
, createTime:1359348369  
, lastAccessTime:0  
, retention:0  
, sd:StorageDescriptor(  
                            cols:[  
                                  FieldSchema(name:name, type:string, comment:null)  
                                , FieldSchema(name:salary, type:float, comment:null)  
         , FieldSchema(name:subordinates, type:array<string>, comment:null)  
         , FieldSchema(name:deductions, type:map<string,float>, comment:null)  
         , FieldSchema(name:address, type:struct<street:string,city:string,state:string,zip:int>, comment:null)  
          ]  
    , location:hdfs://0.0.0.0:8020/user/hive/warehouse/saurav.db/employees  
    , inputFormat:org.apache.hadoop.mapred.TextInputFormat  
    , outputFormat:org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat  
    , compressed:false  
    , numBuckets:-1  
    , serdeInfo:SerDeInfo( name:null  
                          ,serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe  
                          ,parameters:{colelction.delim=, mapkey.delim=, serialization.format=, field.delim=, line.delim=}  
  )  
    , bucketCols:[]  
    , sortCols:[]  
    , parameters:{}  
 )  
, partitionKeys:[]  
, parameters:{transient\_lastDdlTime=1359348610}  
, viewOriginalText:null  
, viewExpandedText:null  
, tableType:MANAGED\_TABLE)

**DESCRIBE TABLE** shows the list of columns including partition columns for the given table. If the **EXTENDED**keyword is specified then it will show all the metadata for the table in Thrift serialized form. This is generally only useful for debugging and not for general use.  
If a table has complex column then you can examine the attributes of this column by specifying table\_name.complex\_col\_name (and '$elem$' for array element, '$key$' for map key, and '$value$' for map value). You can specify this recursively to explore the complex column type.  
For a view, **DESCRIBE TABLE EXTENDED** can be used to retrieve the view's definition. Two relevant attributes are provided: both the original view definition as specified by the user, and an expanded definition used internally by Hive.

### Describe Partition

**DESCRIBE** [EXTENDED] table\_name partition\_spec

This statement lists metadata for a given partition. The output is similar to that of **DESCRIBE TABLE**. Presently, the column information associated with a particular partition is not used while preparing plans.  
Example:

**DESCRIBE** EXTENDED page\_view PARTITION (ds='2008-08-08');

1) Given a hive database name, how can I get the list of external tables in that database?  
  
You can try this command:

SHOW TABLES [IN database\_name] [identifier\_with\_wildcards];

It will give you all tables. As far as I know there is no direct command to know all the tables of type external/internal. For that you have use JDBC connection to connect to HiveMetastore and get the required info.  
  
Given a hive table name, how can I find that whether the table is external or internal table ?  
You can try any of this commands:

describe formatted table\_name  
  
describe extended table\_name

It show all the detail info of a table. Along with:

Table Type: EXTERNAL\_TABLE   
Table Parameters: EXTERNAL=TRUE

**Query to find max salary department wise with condition of salary >20000.**

select d.deptid,d.deptname, max(e.salary) from dept d on employee e join (d.deptid= e.deptid) group by d.deptid,d.deptname having max(e.salary)>250;

Note:: Employee is large table so that one should come on the last of the query

**Join Optimizations**

When joining three or more tables, if every ON clause uses the same join key, a single MapReduce job will be used.

Hive also assumes that the last table in the query is the largest. It attempts to buffer the other tables and then stream the last table through, while performing joins on individual records. Therefore, you should structure your join queries so the largest table is last. Recall our previous join between stocks and dividends. We actually made the mistake of using the smaller dividends table last:

|  |  |
| --- | --- |
|  | SELECT s.ymd, s.symbol, s.price\_close, d.dividend   FROM stocks s JOIN dividends d ON s.ymd = d.ymd AND s.symbol = d.symbol   WHERE s.symbol = 'AAPL';   We should switch the positions of stocks and dividends:    SELECT s.ymd, s.symbol, s.price\_close, d.dividend   FROM dividends d JOIN stocks s ON s.ymd = d.ymd AND s.symbol = d.symbol   WHERE s.symbol = 'AAPL'; |

Fortunately, you don’t have to put the largest table last in the query. Hive also provides a “hint” mechanism to tell the query optimizer which table should be streamed:

|  |  |
| --- | --- |
|  | SELECT /\*+ STREAMTABLE(s) \*/ s.ymd, s.symbol, s.price\_close, d.dividend   FROM stocks s JOIN dividends d ON s.ymd = d.ymd AND s.symbol = d.symbol   WHERE s.symbol = 'AAPL'; |

Now Hive will attempt to stream the stocks table, even though it’s not the last table in the query.

**Second highest salary.**

select min(salary) from (select distinct(salary) from employee order by salary desc limit 2) t;

**Query to display cumulative salary like if a =10, b=20 ,c =30 result is a=10,b=30,c=60.**

input::

1001 1 35000.0

1002 2 35000.0

1001 2 25000.0

1002 3 110000.0

1001 3 40000.0

1002 1 40000.0

output:::

1001 1 35000.0 35000.0

1001 2 25000.0 60000.0

1001 3 40000.0 100000.0

1002 1 40000.0 40000.0

1002 2 35000.0 75000.0

1002 3 110000.0 185000.0

CREATE EXTERNAL TABLE CUMULATIVESUM(STOREID INT,SMONTH INT, SALES FLOAT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

hadoop fs -copyFromLocal /home/ubuntu/Documents/hive/cumulativesum.txt /user

LOAD DATA INPATH '/user/cumulativesum.txt' INTO TABLE CUMULATIVESUM;

SELECT storeid, smonth, sales, SUM(sales) OVER (PARTITION BY storeid ORDER BY smonth) FROM CUMULATIVESUM;

**How can display content of a table of a particular database from another.**Using DB Link

**Basic of spring batch.**

**How sqoop works.**

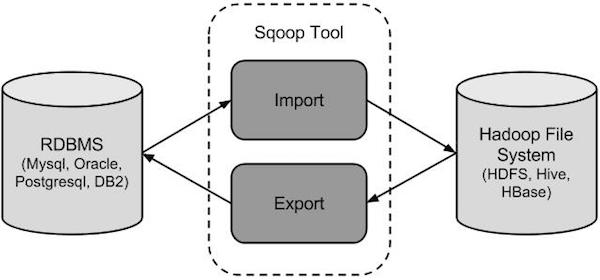
In an [**earlier blog post**](http://blog.cloudera.com/blog/2009/03/06/database-access-with-hadoop/), I described the [**DBInputFormat**](http://hadoop.apache.org/core/docs/current/api/org/apache/hadoop/mapred/lib/db/DBInputFormat.html), a connector that allows Hadoop MapReduce programs to read rows from SQL databases. DBInputFormat allows Hadoop to read input from [**JDBC**](http://en.wikipedia.org/wiki/Jdbc): a Java interface to databases that most popular database vendors (Oracle, MySQL, Postgresql, etc.) implement.

In order to use DBInputFormat you need to write a class that deserializes the columns from the database record into individual data fields to work with. This is pretty tedious—and entirely algorithmic. Sqoop auto-generates class definitions to deserialze the data from the database. These classes can also be used to store the results in Hadoop’s SequenceFile format, which allows you to take advantage of built-in compression within HDFS too. The classes are written out as .java files that you can incorporate in your own data processing pipeline later. The class definition is created by taking advantage of JDBC’s ability to read metadata about databases and tables.

**When Sqoop is invoked, it retrieves the table’s metadata, writes out the class definition for the columns you want to import, and launches a MapReduce job to import the table body proper.**

Hadoop users know that moving large volumes of data can be a time-intensive operation. While it provides a reliable implementation-independent mechanism to read database tables, using a MapReduce JDBC job to import data from a remote database is often inefficient. Database vendors usually provide an export tool that exports data in a more high-performance manner. Sqoop is capable of using alternate import strategies as well. By examining the connect string URL that tells Sqoop which database to connect to, Sqoop will choose alternate import strategies as appropriate to the database. We’ve already implemented the ability to take advantage of MySQL’s export tool called mysqldump. We’ll add support for other systems as soon as we can.

The following image describes the workflow of Sqoop.



## Sqoop Import

The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

## Sqoop Export

The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

**Basics of udf.**Check above

**Metastore with MySQL.**

Check above **Why would you use Pig vs. Hive? Advantages**1) Hive Hadoop Component is used mainly by data analysts whereas Pig Hadoop Component is generally used by Researchers and Programmers.

2) Hive Hadoop Component is used for completely structured Data whereas Pig Hadoop Component is used for semi structured data.

3) Hive Hadoop Component has a declarative SQLish language (HiveQL) whereas Pig Hadoop Component has a procedural data flow language (Pig Latin)

4) Hive Hadoop Component is mainly used for creating reports whereas Pig Hadoop Component is mainly used for programming.

5) Hive Hadoop Component operates on the server side of any cluster whereas Pig Hadoop Component operates on the client side of any cluster.

6) Hive Hadoop Component is helpful for ETL whereas Pig Hadoop is great for ETL because of its powerful transformation and processing capabilities.

7) Hive can start an optional thrift based server that can send queries from any nook and corner directly to the Hive server which will execute them whereas this feature is not available with Pig.

8) Hive directly leverages SQL expertise and thus can be learnt easily whereas Pig is also SQL-like but varies to a great extent and thus it will take some time efforts to master Pig.

9) Hive makes use of exact variation of the SQL DLL language by defining the tables beforehand and storing the schema details in any local database whereas in case of Pig there is no dedicated metadata database and the schemas or data types will be defined in the script itself.

10) The Hive Hadoop component has a provision for partitions so that you can process the subset of data by date or in an alphabetical order whereas Pig Hadoop component does not have any notion for partitions though might be one can achieve this through filters.

11) Pig supports Avro whereas Hive does not.

12) Pig can be installed easily over Hive as it is completely based on shell interaction

13) Pig Hadoop Component renders users with sample data for each scenario and each step through its “Illustrate” function whereas this feature is not incorporated with the Hive Hadoop Component.

14) Hive has smart inbuilt features on accessing raw data but in case of Pig Latin Scripts we are not pretty sure that accessing raw data is as fast as with HiveQL.

15) You can join, order and sort data dynamically in an aggregated manner with Hive and Pig however Pig also provides you an additional COGROUP feature for performing outer joins.

**What HIVE can do which is not possible in PIG?**

Partitioning can be done using HIVE but not in PIG, it is a way of bypassing the output.

**What PIG can do which is not possible in HIVE?**

Positional referencing - Even when you dont have field names, we can reference using the position like $0 - for first field, $1 for second and so on.

And another fundamental difference is, PIG doesn't need a schema to write the values but HIVE does need a schema.

You can connect from any external application to HIVE using JDBC and others but not with PIG.

Note: Both runs on top of HDFS (hadoop distributed file system) and the statements are converted to Map Reduce programs.

For example, Hive is commonly used at Facebook for analytical purposes.  Facebook promotes the Hive language and their employees frequently [**speak about Hive**](http://www.hadoopwizard.com/top-10-presentations-for-learning-hadoop-on-slideshare) at Big Data and [Hadoop conferences](http://www.hadoopwizard.com/events/categories/conference/).

However, Yahoo! is a big advocate for Pig Latin.  Yahoo! has one of the [biggest Hadoop clusters](http://www.hadoopwizard.com/which-big-data-company-has-the-worlds-biggest-hadoop-cluster/) in the world.  Their data engineers use Pig for data processing on their Hadoop clusters.

Alternatively, you may have a choice of Pig or Hive at your organization, especially if no standards have yet been established, or perhaps multiple standards have been set up.

If you know SQL, then Hive will be very familiar to you.  Since Hive uses SQL, you will feel at home with all the familiar **select**, **where**, **group by**, and **order by** clauses similar to SQL for relational databases.  You do, however, lose some ability to optimize the query, by relying on the Hive optimizer.  This seems to be the case for any implementation of SQL on any platform, Hadoop or traditional RDBMS, where hints are sometimes ironically needed to teach the automatic optimizer how to optimize properly.

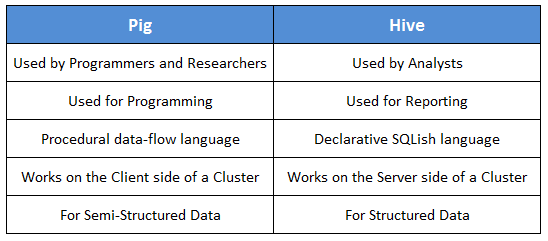
However, compared to Hive, Pig needs some mental adjustment for SQL users to learn.  Pig Latin has many of the usual data processing concepts that SQL has, such as filtering, selecting, grouping, and ordering, but the syntax is a little different from SQL (particularly the **group by** and **flatten** statements!).  Pig requires more verbose coding, although it’s still a fraction of what straight Java MapReduce programs require.  Pig also gives you more control and optimization over the flow of the data than Hive does.

Personally, I use both Pig Latin and Hive, although for different purposes.  I learned Pig Latin first, and have used it to construct dataflows, where I typically have a scheduled job to periodically crunch the massive data from HDFS and to transfer the summarized data into a relational database for reporting, dashboarding, and ad-hoc analyses.  I also use Hive for some simple ad-hoc analytical queries into the data in HDFS, as Hive queries are a lot faster to write for those types of queries.  However, I don’t use Hive for the automated batch jobs that move data between HDFS and other systems.  I find that I can tune the dataflow process better using Pig than with Hive.  Additionally, some of the datasets that I need in Hadoop have not yet been structured with metadata schemas for use with Hive.  In those cases, Pig is much more flexible in reading those datasets than Hive is.

Hadoop expert Alan Gates has an excellent write-up comparing [the differences between Pig Latin and Hive](http://developer.yahoo.com/blogs/hadoop/comparing-pig-latin-sql-constructing-data-processing-pipelines-444.html) and when to use each of them.

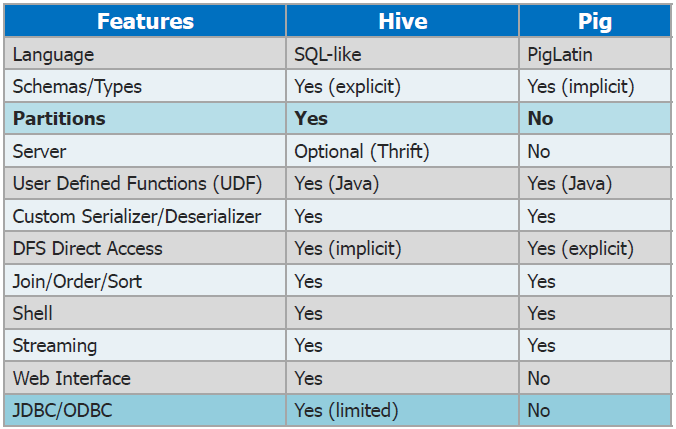
## If you are a data engineer, then you’ll likely feel like you’ll have better control over the dataflow (ETL) processes when you use Pig Latin, especially if you come from a procedural language background.  If you are a data analyst, however, you will likely find that you can ramp up on Hadoop faster by using Hive, especially if your previous experience was more with SQL than with a procedural programming language.  If you really want to become a Hadoop expert, then you should learn both Pig Latin and Hive for the ultimate flexibility. Pig Vs Hive

Here are some basic difference between Hive and Pig which gives an idea of which to use depending on the type of data and purpose.



## Why Go for Hive When Pig is There?

So why go for Hive when Pig is there. The tabular column below gives a comprehensive comparision between the two. The Hive can be used in places where partitions are necessary and when it is essential to define and create cross-language services for numerous languages.

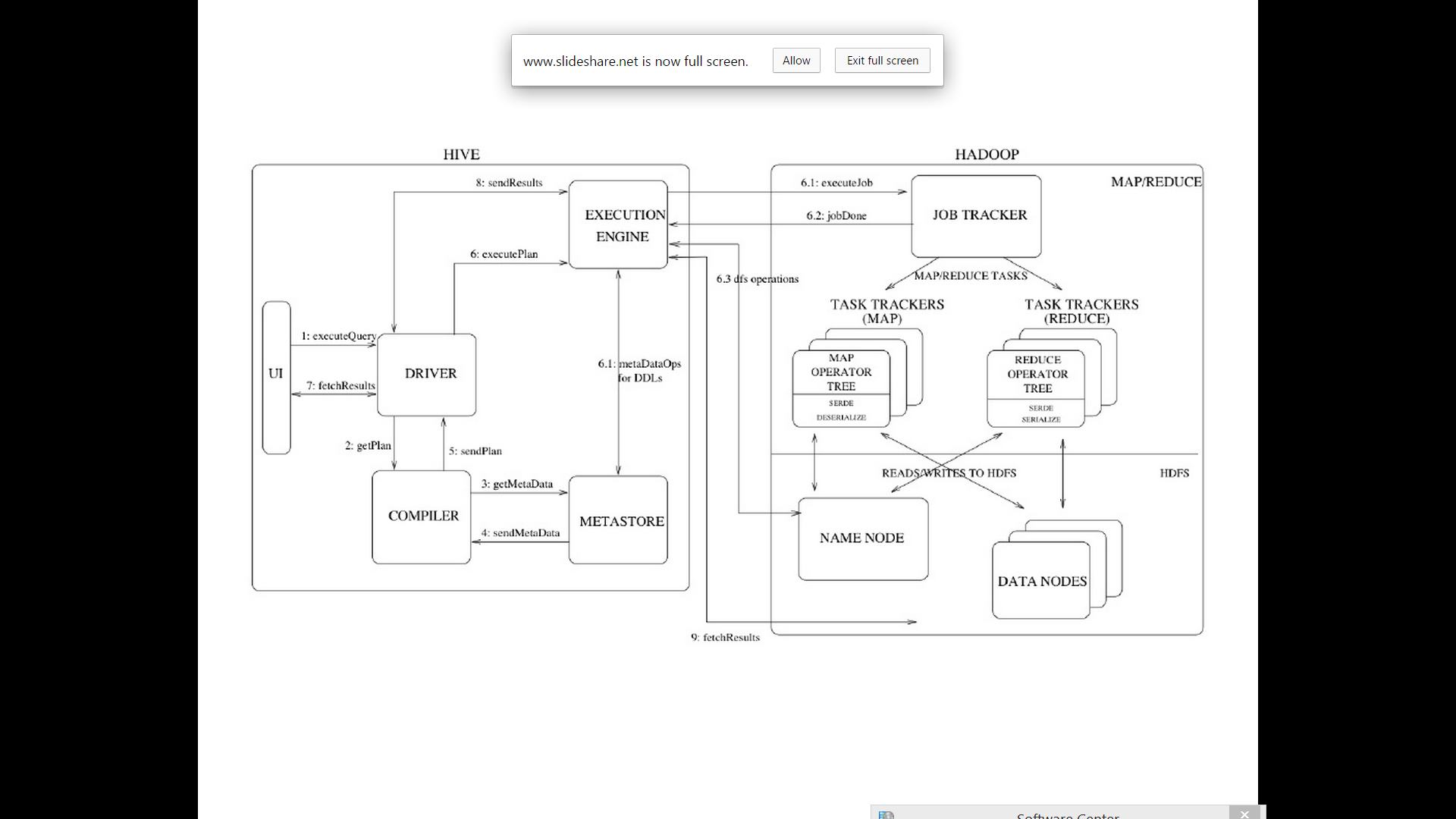


Got a question for us? Mention them in the comments section and we will get back t

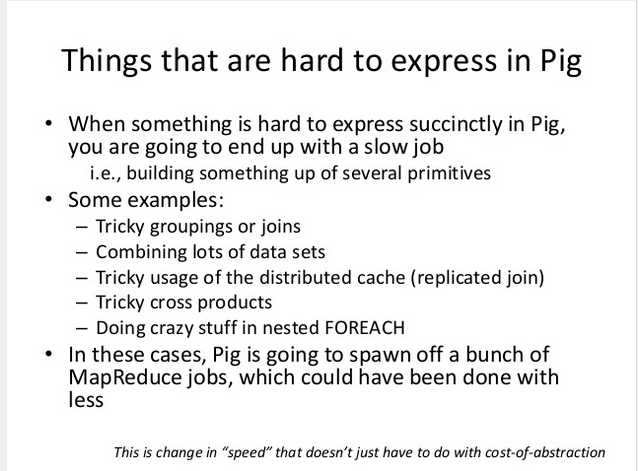
**How to monitor health of such a system?**

**How to do map side join in Hive. Is there any particular syntax for it?**Check above

**How Hive query works (more interested in knowing query‘s conversion to map-reduce).**



**Does using Pig/Hive have performance benefit over MapReduce?**Hypothetically mapreduce faster than pig

**  
  
What are the ways to implement hive optimizations?**

## Leveraging Time-based Partitioning

All imported data is automatically partitioned into hourly buckets, based on the **‘time’** field within each data record. **By specifying the time range to query, you avoid reading unnecessary data and can thus speed up your query significantly.**

### 1) WHERE time <=> Integer

**When the ‘time’ field within the WHERE clause is specified**, the query parser will automatically detect which partition(s) should be processed. Please note that this auto detection will not work if you specify the time with*float*instead of*int*.

[GOOD]: SELECT field1, field2, field3 FROM tbl WHERE time > 1349393020

[GOOD]: SELECT field1, field2, field3 FROM tbl WHERE time > 1349393020 + 3600

[GOOD]: SELECT field1, field2, field3 FROM tbl WHERE time > 1349393020 - 3600

[BAD]: SELECT field1, field2, field3 FROM tbl WHERE time > 13493930200 / 10

[BAD]: SELECT field1, field2, field3 FROM tbl WHERE time > 1349393020.00

[BAD]: SELECT field1, field2, field3 FROM tbl WHERE time BETWEEN 1349392000 AND 1349394000

### 2) TD\_TIME\_RANGE

**An easier way to slice the data is to use**[**TD\_TIME\_RANGE UDF**](http://docs.treasuredata.com/articles/udfs#tdtimerange).

SELECT ... WHERE TD\_TIME\_RANGE(time, "2013-01-01 PDT")

SELECT ... WHERE TD\_TIME\_RANGE(time, "2013-01-01", NULL, "PDT")

SELECT ... WHERE TD\_TIME\_RANGE(time, "2013-01-01",

TD\_TIME\_ADD("2013-01-01", "1day", "PDT"))

However, if you use TD\_TIME\_FORMAT UDF or division in TD\_TIME\_RANGE, time partition opimization doesn’t work. For instance, the following conditions disable optimization.

SELECT ... WHERE TD\_TIME\_RANGE(time, TD\_TIME\_FORMAT(TD\_SCHEDULED\_TIME(), 'yyyy-MM-dd'))

SELECT ... WHERE TD\_TIME\_RANGE(time, TD\_TIME\_FORMAT(1356998401, 'yyyy-MM-dd'))

SELECT ... WHERE TD\_TIME\_RANGE(time, TD\_SCHEDULED\_TIME() / 86400 \* 86400))

SELECT ... WHERE TD\_TIME\_RANGE(time, 1356998401 / 86400 \* 86400))

## Set Custom Schema

As explained in the [Schema Management](http://docs.treasuredata.com/articles/schema) article, all tables have two fields: ‘v’ and ‘time’. In addition to these, you can set[custom schema](http://docs.treasuredata.com/articles/schema#setting-custom-schema) on the tables.

$ td schema:set testdb www\_access action:string user:int

$ td query -w -d testdb "SELECT user, COUNT(1) AS cnt

FROM www\_access

WHERE action='login'

GROUP BY user ORDER BY cnt DESC"

After setting the schema, **queries issued with named columns instead of ‘v’** will use the schema information to achieve a more optimized execution path. In particular, GROUP BY performance will improve significantly.

## Use Hive’s Mapjoin

If all but one of the tables being joined are small, Hive can perform the join on the mapper side in memory. This can greatly reduce the query time by cutting down on the number of mappers and/or stages in the mapreduce pipeline.

SELECT /\*+ MAPJOIN(tbl2) \*/ ... FROM tbl1 join tbl2 on tbl1.key = tbl2.key

In this example, we’re specifying tbl2 as a Mapjoin table that is much smaller than tbl1 and should fit in memory. For more information on this optimization, please refer to the [Hive documentation](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins#LanguageManualJoins-Mapjoinrestrictions).

## DISTRIBUTE BY…SORT BY v. ORDER BY

In Hive, ORDER BY is not a very fast operation because it forces all the data to go into the same reducer node. By doing this, Hive ensures that the entire dataset is totally ordered.

However, sometimes we do not require total ordering. For example, suppose you have a table called user\_action\_tablewhere each row has user\_id, action, and time. Your goal is to order them by time **per user\_id**.

If you are doing this with ORDER BY, you would run

SELECT time, user\_id, action FROM user\_action\_table

ORDER BY user\_id, time

However, you can achieve the same result with

SELECT time, user\_id, action FROM user\_action\_table

DISTRIBUTE BY user\_id SORT BY time

This is because all the rows belonging to the same user\_id go to the same reducer (“DISTRIBUTE BY user\_id”) and in each reducer, rows are sorted by time (“SORT BY time”). This is faster than the other query because it uses multiple reducers as opposed to a single reducer.

You can learn more about the differences between ORDER BY and SORT BY [here](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy#LanguageManualSortBy-DifferencebetweenSortByandOrderBy).

## Avoid “SELECT count(DISTINCT field) FROM tbl”

This query looks familier to SQL users, but this query is very slow because **only one reducer** is used to process the request.

SELECT count(DISTINCT field) FROM tbl

So please rewrite the query like below to leverage multiple reducers.

SELECT

count(1)

FROM (

SELECT DISTINCT field FROM tbl

) t

## Considering the Cardinality within GROUP BY

There’s a probability where GROUP BY becomes a little bit faster, by carefully ordering a list of fields within GROUP BY in an order of high cardinality.

good: SELECT GROUP BY uid, gender

bad: SELECT GROUP BY gender, uid

If this article is incorrect or outdated, or omits critical information, please [let us know](http://www.treasuredata.com/contact.php). For all other issues, please see our[support channels](http://docs.treasuredata.com/articles/support-channels). [Live chat](javascript:olark('api.box.expand');) with our staff also works well.

**Difference b/w sort by and order by.**

In Hive, ORDER BY is not a very fast operation because it forces all the data to go into the same reducer node. By doing this, Hive ensures that the entire dataset is totally ordered.

However, sometimes we do not require total ordering. For example, suppose you have a table called user\_action\_tablewhere each row has user\_id, action, and time. Your goal is to order them by time **per user\_id**.

If you are doing this with ORDER BY, you would run

SELECT time, user\_id, action FROM user\_action\_table

ORDER BY user\_id, time

However, you can achieve the same result with

SELECT time, user\_id, action FROM user\_action\_table

DISTRIBUTE BY user\_id SORT BY time

This is because all the rows belonging to the same user\_id go to the same reducer (“DISTRIBUTE BY user\_id”) and in each reducer, rows are sorted by time (“SORT BY time”). This is faster than the other query because it uses multiple reducers as opposed to a single reducer.

The longer version:

* ORDER BY x: guarantees global ordering, but does this by pushing all data through just one reducer. This is basically unacceptable for large datasets. You end up one sorted file as output.
* SORT BY x: orders data at each of N reducers, but each reducer can receive overlapping ranges of data. You end up with N or more sorted files with overlapping ranges.
* DISTRIBUTE BY x: ensures each of N reducers gets non-overlapping ranges of x, but doesn't sort the output of each reducer. You end up with N or unsorted files with non-overlapping ranges.
* CLUSTER BY x: ensures each of N reducers gets non-overlapping ranges, then sorts by those ranges at the reducers. This gives you global ordering, and is the same as doing (DISTRIBUTE BY x and SORT BY x). You end up with N or more sorted files with non-overlapping ranges.

Make sense? So CLUSTER BY is basically the more scalable version of ORDER BY.

**Ref\_::** [**http://stackoverflow.com/questions/13715044/hive-cluster-by-vs-order-by-vs-sort-by**](http://stackoverflow.com/questions/13715044/hive-cluster-by-vs-order-by-vs-sort-by)Hive would normally use the Hadoop built-in sort mechanism, that is to say in-memory quick sort along with merge sorts to combine several sorted pieces. Hive only defines Java and binary comparators which are used inside sorting phases.  
  
Following Mark's question, I can add that Hadoop uses a quicksort in the recursion is not too deep. In this particular case, HeapSort is used (see org.apache.hadoop.util.QuickSort).  
  
Moreover, the sorting algorithm class can be modified/re-implemented by overriding map.sort.class in Hadoop configuration.

**What is metastore in hive and how to configure warehouse directory.**Check above

**Query to find max amount spend from credit card month wise with amount spent greater than 20000.**

hive> select max(sales), month(time) from SALESVER2\_EXTERNAL group by month(time) having max(sales)>30000;

**How to delete a particular file from hive table?**

Before delete ::

hive> select \* from employee;

OK

11 Vinay 200 1

12 Bala 400 2

13 Eve 100 3

14 Ram 150 1

15 Adam 900 3

16 kris 600 3

17 Rahul 700 2

11 Vinay 200 1

12 Bala 400 2

13 Eve 100 3

14 Ram 150 1

15 Adam 900 3

16 kris 600 3

17 Rahul 700 2

11 Vinay 200 1

12 Bala 400 2

13 Eve 100 3

14 Ram 150 1

15 Adam 900 3

16 kris 600 3

17 Rahul 700 2

Time taken: 0.303 seconds, Fetched: 21 row(s)

ubuntu@ubuntu:/usr/local/hadoop$ hadoop fs -ls /user/hive/warehouse

Found 9 items

drwxr-xr-x - ubuntu supergroup 0 2015-04-08 23:06 /user/hive/warehouse/cumulativesum

drwxr-xr-x - ubuntu supergroup 0 2015-04-08 14:59 /user/hive/warehouse/dept

drwxr-xr-x - ubuntu supergroup 0 2015-04-20 19:47 /user/hive/warehouse/employee

drwxrwxr-x - hadoop hive 0 2015-04-13 12:52 /user/hive/warehouse/had.db

drwxr-xr-x - ubuntu supergroup 0 2015-04-10 12:36 /user/hive/warehouse/sales

drwxr-xr-x - ubuntu supergroup 0 2015-04-10 13:00 /user/hive/warehouse/salesver2

drwxr-xr-x - ubuntu supergroup 0 2015-04-20 19:18 /user/hive/warehouse/salesver2\_external

drwxr-xr-x - ubuntu supergroup 0 2015-04-21 16:28 /user/hive/warehouse/salesver2\_managed

drwxr-xr-x - ubuntu supergroup 0 2015-04-09 01:31 /user/hive/warehouse/testdb.db

ubuntu@ubuntu:/usr/local/hadoop$ hadoop fs -ls /user/hive/warehouse/employee

Found 3 items

-rw-r--r-- 1 ubuntu supergroup 98 2015-04-07 16:32 /user/hive/warehouse/employee/employee.txt

-rw-r--r-- 1 ubuntu supergroup 98 2015-04-08 14:58 /user/hive/warehouse/employee/employee\_copy\_1.txt

-rw-r--r-- 1 ubuntu supergroup 98 2015-04-20 19:44 /user/hive/warehouse/employee/employee\_copy\_2.txt

ubuntu@ubuntu:/usr/local/hadoop$ hadoop fs -rm /user/hive/warehouse/employee/employee\_copy\_1.txt

Deleted hdfs://localhost:54310/user/hive/warehouse/employee/employee\_copy\_1.txt

ubuntu@ubuntu:/usr/local/hadoop$ hadoop fs -ls /user/hive/warehouse/employee

Found 2 items

-rw-r--r-- 1 ubuntu supergroup 98 2015-04-07 16:32 /user/hive/warehouse/employee/employee.txt

-rw-r--r-- 1 ubuntu supergroup 98 2015-04-20 19:44 /user/hive/warehouse/employee/employee\_copy\_2.txt

ubuntu@ubuntu:/usr/local/hadoop$

hive> select \* from employee;

OK

11 Vinay 200 1

12 Bala 400 2

13 Eve 100 3

14 Ram 150 1

15 Adam 900 3

16 kris 600 3

17 Rahul 700 2

11 Vinay 200 1

12 Bala 400 2

13 Eve 100 3

14 Ram 150 1

15 Adam 900 3

16 kris 600 3

17 Rahul 700 2

Time taken: 0.053 seconds, Fetched: 14 row(s)

To remove files from HDFS using hadoop, run command

***hadoop fs -rmr <directory/files>- From hdfs console***

**dfs –rmr <directory/file> - From Hive console**

Now use the below hadoop command to delete all the files belonging to that table from hdfs. This should basically truncate the table in Hive.  
  
hadoop fs -rm /user/hive/warehouse/mydatabase/mytable/\*  
  
  
If for instance you want to delete only few files from dhfs then you can use the same command where instead of the wildcard you will use the file name. This is equivalent to deleting records in Hive.  For example:  
  
hadoop fs -rm /user/hive/warehouse/mydatabase/mytable/file1.txt

**Can we change column name in hive, how?**CREATE TABLE test\_change (a int, b int, c int);

// will change column a's name to a1

ALTER TABLE test\_change CHANGE a a1 INT;

**Can we give location in hive managed tables?**By default the hive warehouse directory is located at  the hdfs location ***/user/hive/warehouse***

If you want to change this location, you can add the following property to hive-site.xml.

Everyone using hive should have appropriate read/write permissions to this warehouse directory.

|  |  |
| --- | --- |
|  | <property>     <name>hive.metastore.warehouse.dir</name>     <value>/user/hivestore/warehouse </value>     <description>location of the warehouse directory</description>   </property> |

**Is Hive a database?**Hive is a data warehousing package/infrastructure built on top of Hadoop.It provides an SQL dialect, called Hive Query Language(HQL) for querying data stored in a Hadoop cluster.HQL is the Hive query language. Like all SQL dialects in widespread use, it doesn’t fully conform to any particular revision of the ANSI SQL standard. It is perhaps closest to MySQL’s dialect, but with significant differences. Hive offers no support for rowlevel inserts, updates, and deletes. Hive doesn’t support transactions.So we can't compare it with RDBMS. Hive adds extensions to provide better performance in the context of Hadoop and to integrate with custom extensions and even external programs. It is well suited for batch processing data like: **Log processing, Text mining, Document indexing, Customer-facing business intelligence, Predictive modeling, hypothesis testing etc.**

Hive is not designed for online transaction processing and does not offer real-time queries

[**Can I change a table from internal to external in hive?**](http://stackoverflow.com/questions/11839989/can-i-change-a-table-from-internal-to-external-in-hive)

ALTER TABLE <table> SET TBLPROPERTIES('EXTERNAL'='TRUE')

Note: EXTERNAL and TRUE need to caps or it will not work

**Can we update or delete the data from dataset?**There is no operation supported for deletion or update of a particular record or particular set of records, and to me this is more a sign of a poor schema.

You can delete rows from a table using a workaround, in which you overwrite the table by the dataset you want left into the table as a result of your operation.

insert overwrite table your\_table

select \* from your\_table

where id <> 1

;

**What are the best practices for using Hive? What settings should we enable most of the time?**

**Compression**  
Generally, use block compression over value compression since it is more efficient (see mapred.output.compression.type setting). Try and use splittable compression algorithms provided by Hadoop & Hive like Snappy (see mapred.output.compression.codec setting). Avoid Gzip - not splittable and CPU intensive - unless you ingest pre-split Gzipped files from a secondary process.  
  
Ensure your final Hive output is always compressed:

set hive.exec.compress.output=true;

And also your intermediate output should be compressed:

set hive.exec.compress.intermediate=true;

**Map Joins**  
Use small tables on the left side of your joins and enable auto optimisation:

set hive.auto.convert.join=true;

This will cache small enough tables and allow for mapper-only joins which are much faster. You can also tweak the size of what HIve considers small enough to cache:

set hive.mapjoin.smalltable.filesize=50000000;

Make sure your JVM has enough memory though.  
  
**Tune Hadoop**  
Lastly, optimise Hadoop of course and keep an eye on the jobtracker and logs of the mapreduce jobs like spills. You usually a good idea is to increase io.sort.mb - default 100 (MB), try 250 or 500 depending on your JVM size -, increase mapred.min.split.size and mapred.max.split.size to ensure that you don't create countless tiny map tasks - size depends on many factors, I had good experience with 250,000,000 to 2,000,000,000 (bytes).

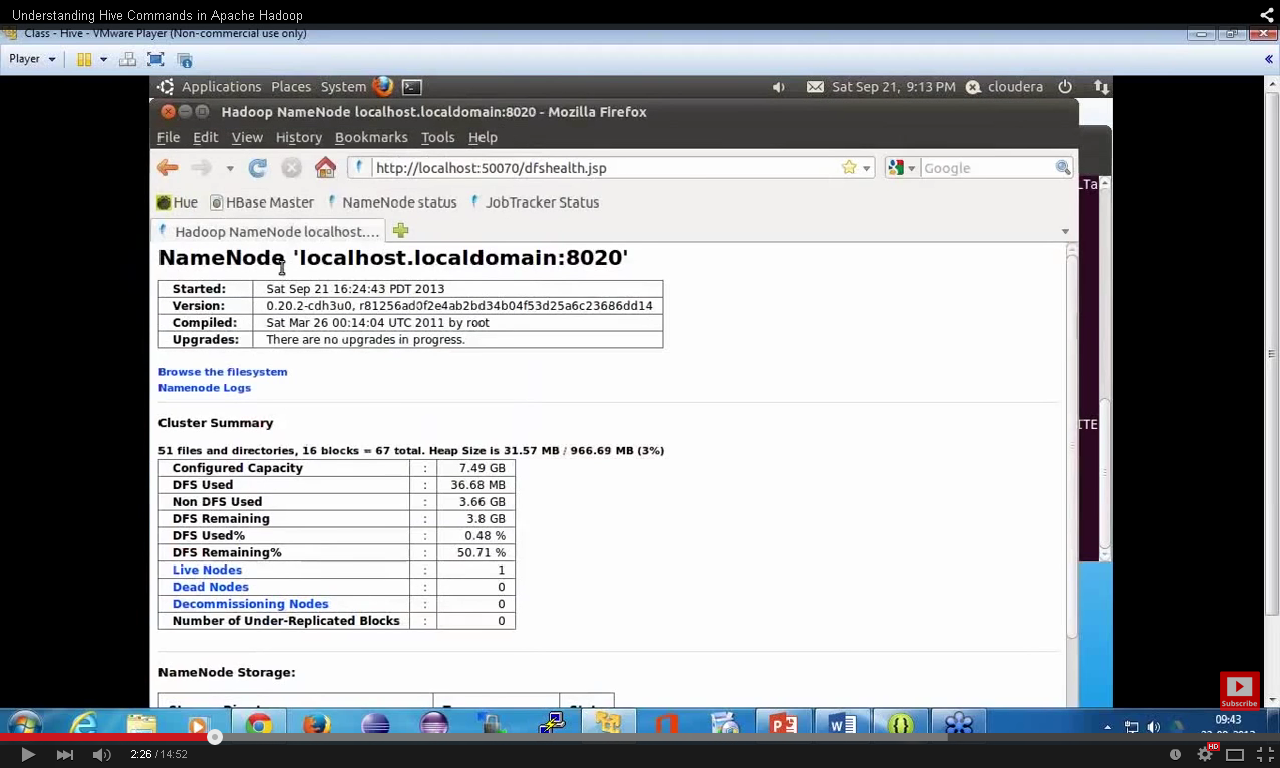
\_http://www.qubole.com/blog/big-data/hive-best-practices/  
  
  
**Hive usage**::

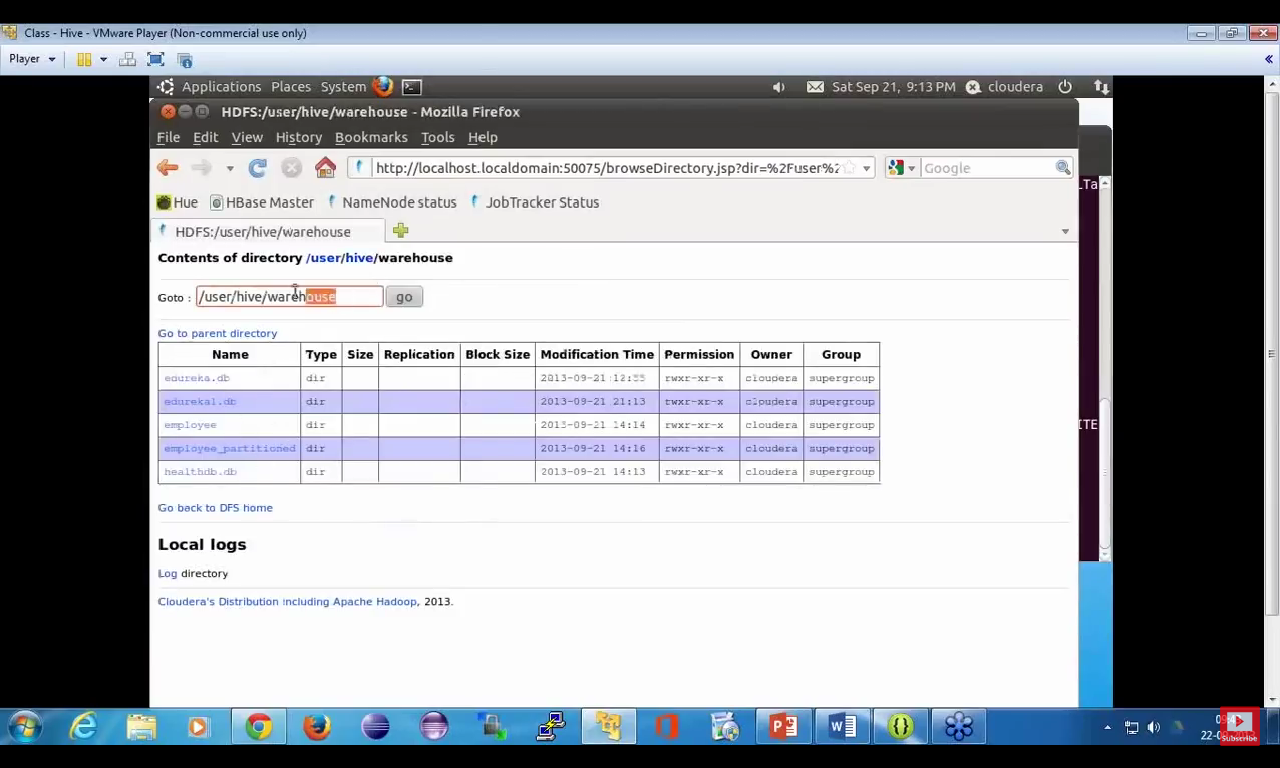
Data Mining  
Log Processing

Document Indexing – Partition your document and then do index the data

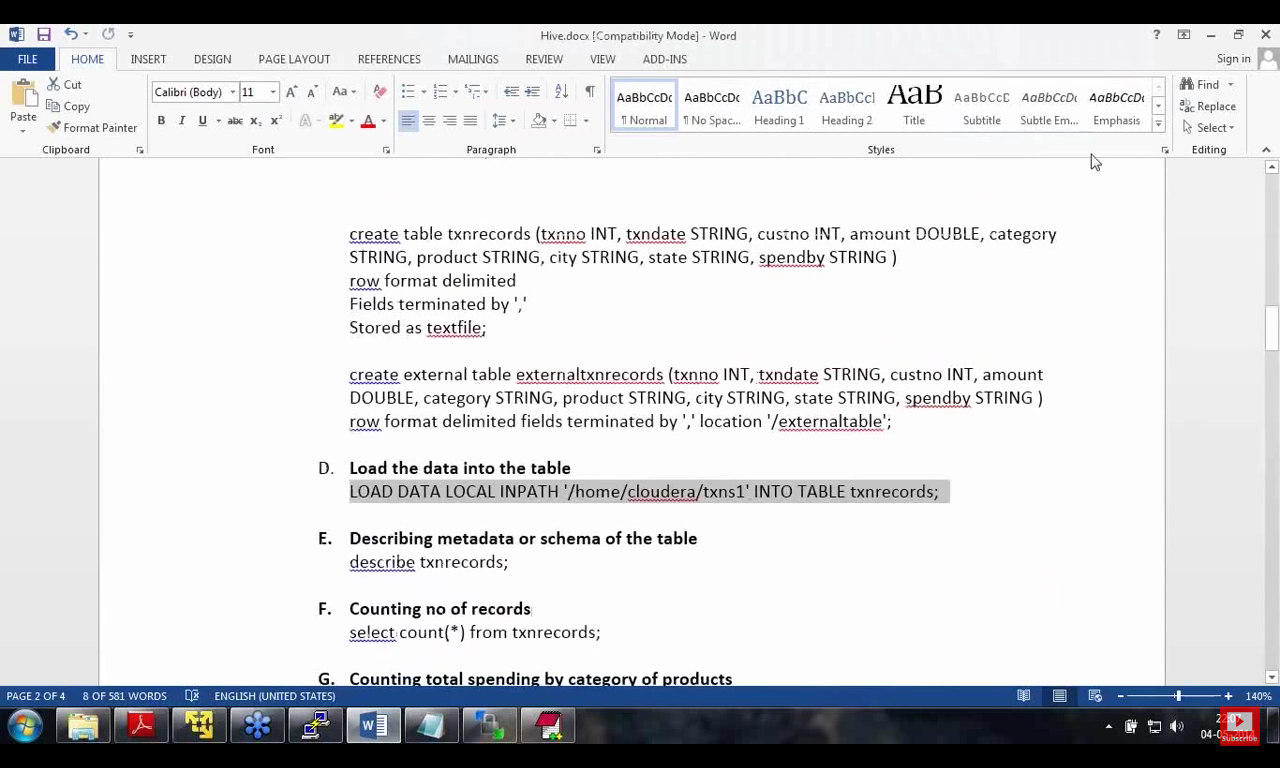
Customer facing business Intelligence – facebook analysis  
Predictive Modelling

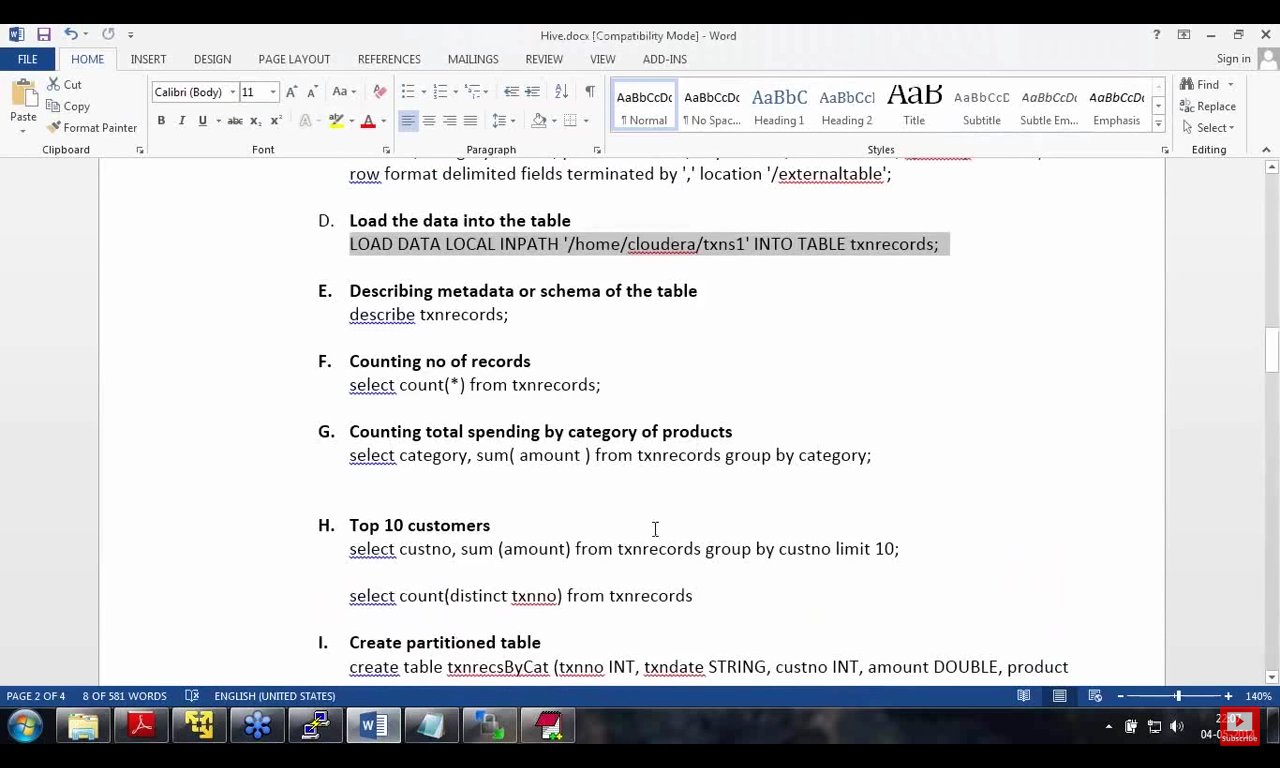
To check Hive created table open namenode URL.  
<http://localhost:50070>. Then click Browse the file system.

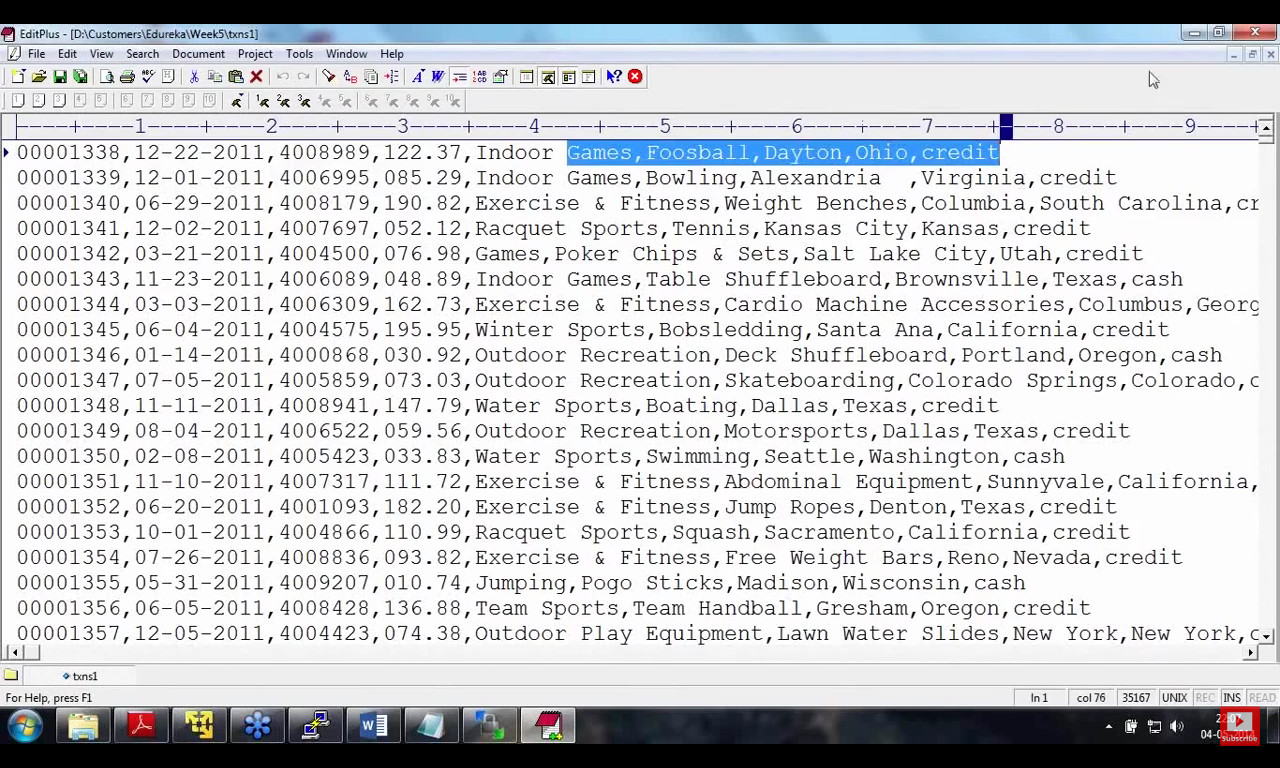
  
Below is the location for hive managed table.

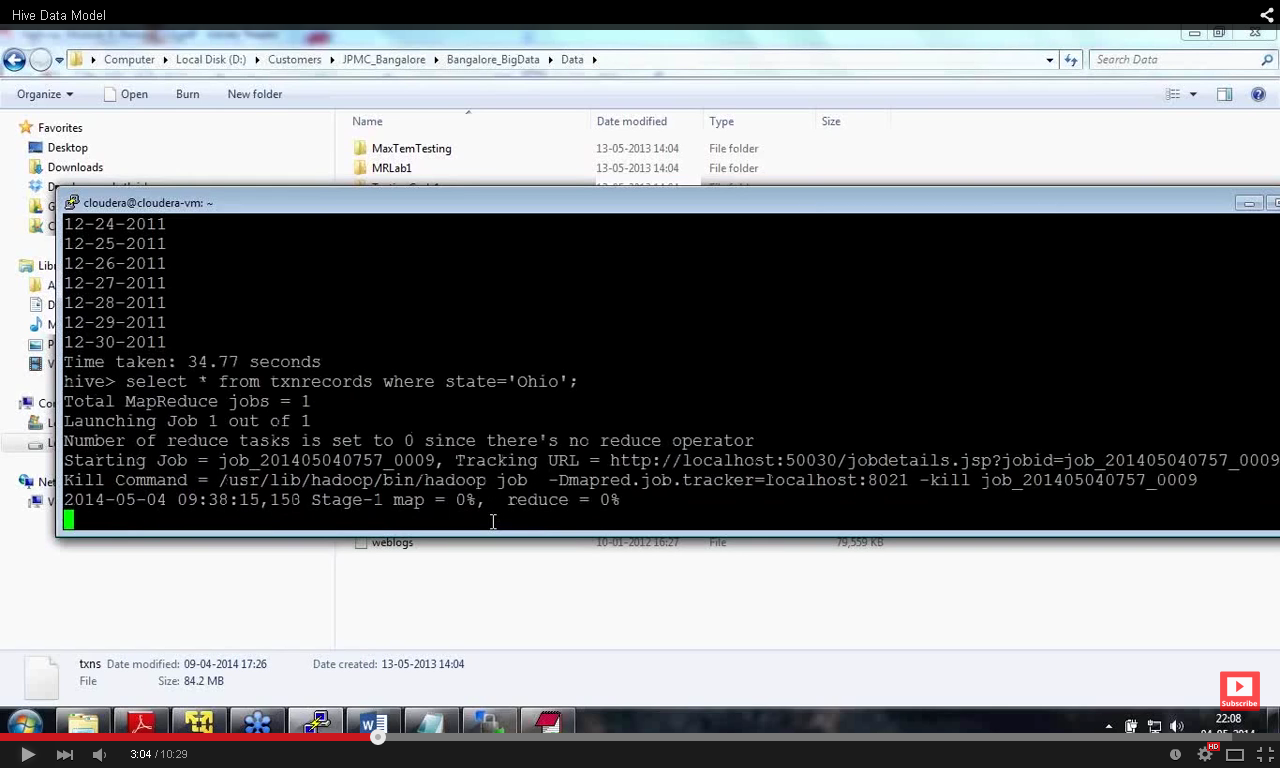


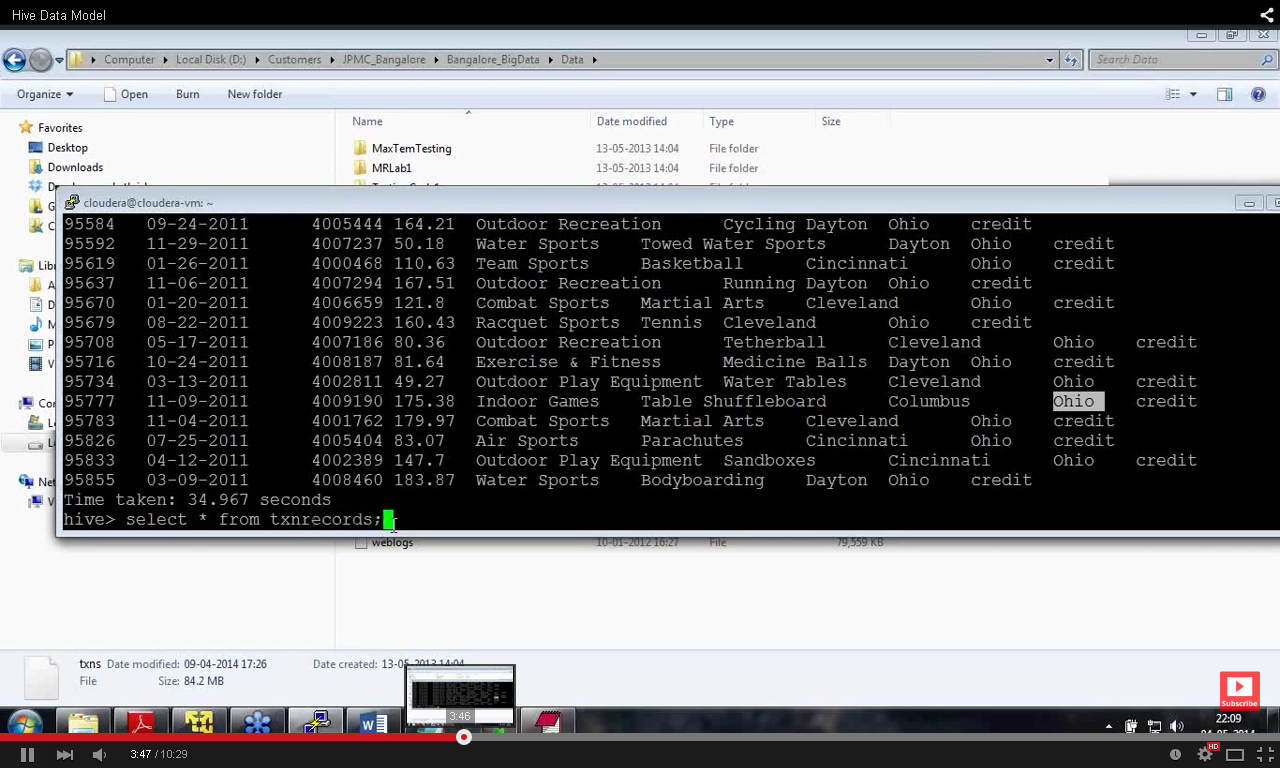
create table managedemp(col1 datatype,col2 datatype, ....) row format delimited fields terminated by 'delimiter character'  
location '/data/employee'  
  
but to create external table ,we use external keyword like below  
  
create external table managedemp(col1 datatype,col2 datatype, ....) row format delimited fields terminated by 'delimiter character'  
location '/data/employee'  
  
2. Differentiation  
  
How do you check wether existing table is managed or external table?  
  
To check that we use describe command like below  
  
describe formatted tablename;  
  
It displays complete meta data of a table.you will see one row called table type which will display either MANAGED\_TABLE OR EXTERNAL\_TABLE  
  
for example if it is managed table ,you will see  
  
Table Type:             MANAGED\_TABLE  
  
if it is external table ,you will see  
  
Table Type:             EXTERNAL\_TABLE

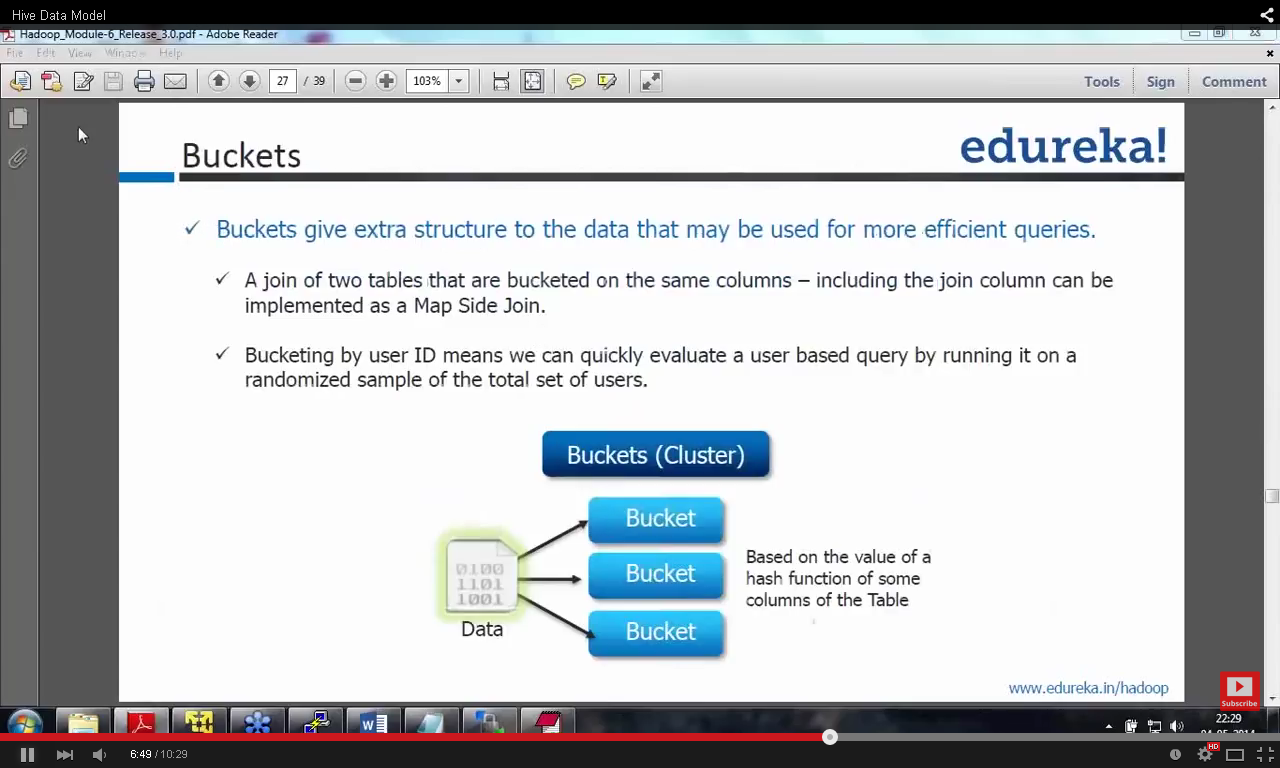


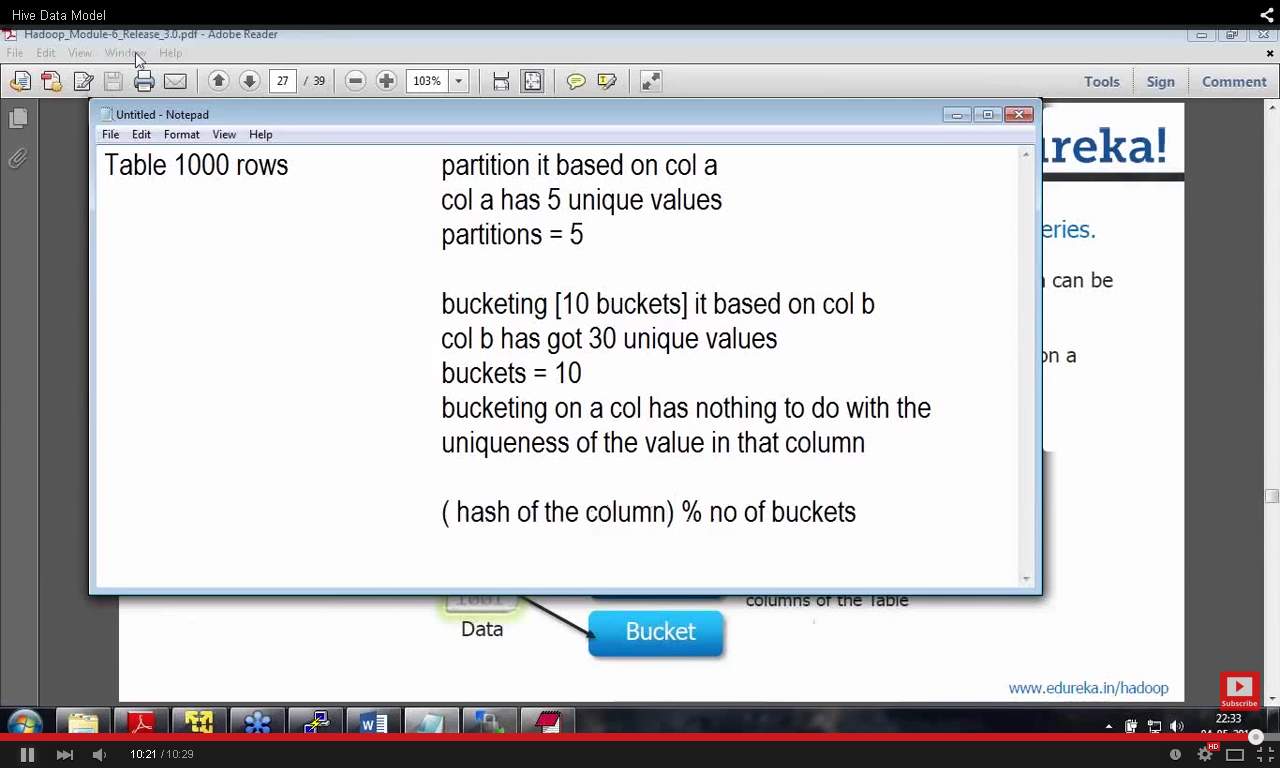






Since the table is not partitioned MapReduce Job is generating.  
Whenever full table scan is require then no need of Mapreduce job.  
  
ucke

No Map reduce code is require.  
Bucketing::  
When you want to break the data into bucket based on some column.  
Bucketing is always done for sampling.  
Say u want 10% of the data , so you break the data in 10 buckets and get 10% of the data from each bucket.  
  




## Which data will go to which bucket is determined by (hash of the column)% no of buckets Bucketing in Hive

Usually Partitioning in Hive offers a way of segregating hive table data into multiple files/directories. **But partitioning gives effective results when**,

* There are limited number of partitions
* Comparatively equal sized partitions

But this may not possible in all scenarios, like when are partitioning our tables based geographic locations like country, some bigger countries will have large partitions (ex: 4-5 countries itself contributing 70-80% of total data) where as small countries data will create small partitions (remaining all countries in the world may contribute to just 20-30 % of total data). So, In these cases Partitioning will not be ideal.

**To overcome the problem of over partitioning**, Hive provides Bucketing concept, another technique for decomposing table data sets into more manageable parts.

###### ****Features****

* Bucketing concept is based on **(hashing function on the bucketed column)** **mod** **(by total number of buckets)**. The hash\_function depends on the type of the bucketing column.
* Records with the same bucketed column will always be stored in the same bucket.
* We use **CLUSTERED BY** clause to divide the table into buckets.
* Physically, each bucket is just a file in the table directory, and Bucket numbering is 1-based.
* Bucketing can be done along with Partitioning on Hive tables and even without partitioning.
* Bucketed tables will create almost equally distributed data file parts.

###### ****Advantages****

* Bucketed tables offer efficient sampling than by non-bucketed tables. With sampling, we can try out queries on a fraction of data for testing and debugging purpose when the original data sets are very huge.
* As the data files are equal sized parts, map-side joins will be faster on bucketed tables than non-bucketed tables. In Map-side join, a mapper processing a bucket of the left table knows that the matching rows in the right table will be in its corresponding bucket, so it only retrieves that bucket (which is a small fraction of all the data stored in the right table).
* Similar to partitioning, bucketed tables provide faster query responses than non-bucketed tables.
* Bucketing concept also provides the flexibility to keep the records in each bucket to be sorted by one or more columns. This makes map-side joins even more efficient, since the join of each bucket becomes an efficient merge-sort.

###### ****Limitations****

* Specifying bucketing doesn’t ensure that the table is properly populated. Data Loading into buckets needs to be handled by our-self.

### ****Example Use Case****

Lets explore the remaining features of Bucketing in Hive with an example Use case, by creating buckets for sample user records provided in the previous post on partitioning –> [UserRecords](http://hadooptutorial.info/wp-content/uploads/2014/12/UserRecords.txt)

Let us create the table partitioned by country and bucketed by state and sorted in ascending order of cities.

#### ****Creation of Bucketed Tables****

We can create bucketed tables with the help of **CLUSTERED BY** clause and optional **SORTED BY**clause in CREATE TABLE statement. With the help of the below HiveQL we can create**bucketed\_user** table with above given requirement.

Bucketed Sorted Table Creation

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | CREATE TABLE bucketed\_user(          firstname VARCHAR(64),           lastname  VARCHAR(64),           address   STRING,           city   VARCHAR(64),          state     VARCHAR(64),           post      STRING,           phone1    VARCHAR(64),           phone2    STRING,           email     STRING,           web       STRING           )          COMMENT 'A bucketed sorted user table'           PARTITIONED BY (country VARCHAR(64))          CLUSTERED BY (state) SORTED BY (city) INTO 32 BUCKETS           STORED AS SEQUENCEFILE; |

Unlike partitioned columns (which are not included in table columns definition) , Bucketed columns are included in table definition as shown in above code for **state** and **city** columns.

#### ****Inserting data Into Bucketed Tables****

Similar to partitioned tables, we can not directly load bucketed tables with **LOAD DATA (LOCAL) INPATH** command, rather we need to use **INSERT OVERWRITE TABLE … SELECT …FROM** clause from another table to populate the bucketed tables. For this, we will create one temporary table in hive with all the columns in input file from that table we will copy into our target bucketed table.

Lets assume we have created **temp\_user** temporary table, and below is the HiveQL for populating bucketed table with **temp\_user** table.

To populate the bucketed table, we need to set the property **hive.enforce.bucketing = true,**so that Hive knows to create the number of buckets declared in the table definition.

Loading Data into Hive Bucketed Sorted Table

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | set hive.enforce.bucketing = true;    INSERT OVERWRITE TABLE bucketed\_user PARTITION (country)          SELECT  firstname ,           lastname  ,           address   ,           city      ,          state     ,           post      ,           phone1    ,           phone2    ,           email     ,           web       ,           country           FROM temp\_user; |

**Note:**

* The property **hive.enforce.bucketing = true**similar to **hive.exec.dynamic.partition=true**property in partitioning. By Setting this property we will enable dynamic bucketing while loading data into hive table.
* It will automatically **sets the number of reduce tasks** to be equal to the **number of buckets**mentioned in the table definition (for example 32 in our case) and automatically selects the clustered by column from table definition.
* If we do not set this property in Hive Session, we have to manually convey same information to Hive that, number of reduce tasks to be run (for example in our case, by using **set mapred.reduce.tasks=32**) and **CLUSTER BY (state) and SORT BY (city)** clause in the above INSERT …SELECT statement at the end.

Lets see the execution of these scripts and their output in the next page.

##### ****Solution For Example Use Case****

Below is the combined HiveQL along with script required for temporary hive table creation. Lets save this HiveQL into **bucketed\_user\_creation.hql** and save the input file provided in example use case section into **user\_table.txt** file in home directory.

bucketed\_user\_creation.hql

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | set hive.exec.dynamic.partition=true;  set hive.exec.dynamic.partition.mode=nonstrict;  set hive.exec.max.dynamic.partitions.pernode=1000;  set hive.enforce.bucketing = true;    DROP TABLE IF EXISTS bucketed\_user;    CREATE TEMPORARY TABLE temp\_user(         firstname VARCHAR(64),         lastname  VARCHAR(64),         address   STRING,         country   VARCHAR(64),         city      VARCHAR(64),         state     VARCHAR(64),         post      STRING,         phone1    VARCHAR(64),         phone2    STRING,         email     STRING,         web       STRING         )         ROW FORMAT DELIMITED         FIELDS TERMINATED BY ','         LINES TERMINATED BY '\n'         STORED AS TEXTFILE;  LOAD DATA LOCAL INPATH '/home/user/user\_table.txt' INTO TABLE temp\_user;    CREATE TABLE bucketed\_user(         firstname VARCHAR(64),         lastname  VARCHAR(64),         address   STRING,         city      VARCHAR(64),         state     VARCHAR(64),         post      STRING,         phone1    VARCHAR(64),         phone2    STRING,         email     STRING,         web       STRING         )         COMMENT 'A bucketed sorted user table'         PARTITIONED BY (country VARCHAR(64))         CLUSTERED BY (state) SORTED BY (city) INTO 32 BUCKETS         STORED AS SEQUENCEFILE;    set hive.enforce.bucketing = true;  INSERT OVERWRITE TABLE bucketed\_user PARTITION (country)         SELECT  firstname ,                 lastname  ,                 address   ,                 city      ,                 state     ,                 post      ,                 phone1    ,                 phone2    ,                 email     ,                 web       ,                 country           FROM temp\_user; |

###### ****Output****

Lets execute this script in hive and below is the output of the above script execution.

Script Execution log messages and Output

Shell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | user@tri03ws-386:~$ hive -f bucketed\_user\_creation.hql  Logging initialized using configuration in jar:file:/home/user/bigdata/apache-hive-0.14.0-bin/lib/hive-common-0.14.0.jar!/hive-log4j.properties  OK  Time taken: 12.144 seconds  OK  Time taken: 0.146 seconds  Loading data to table default.temp\_user  Table default.temp\_user stats: [numFiles=1, totalSize=283212]  OK  Time taken: 0.21 seconds  OK  Time taken: 0.5 seconds  Query ID = user\_20141222163030\_3f024f2b-e682-4b08-b25c-7775d7af4134  Total jobs = 1  Launching Job 1 out of 1  Number of reduce tasks determined at compile time: 32  In order to change the average load for a reducer (in bytes):    set hive.exec.reducers.bytes.per.reducer=<number>  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 mapreduce.job.reduces=<number>  Starting Job = job\_1419243806076\_0002, Tracking URL = http://tri03ws-386:8088/proxy/application\_1419243806076\_0002/  Kill Command = /home/user/bigdata/hadoop-2.6.0/bin/hadoop job  -kill job\_1419243806076\_0002  Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 32  2014-12-22 16:30:36,164 Stage-1 map = 0%,  reduce = 0%  2014-12-22 16:31:09,770 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.66 sec  2014-12-22 16:32:10,368 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.66 sec  2014-12-22 16:32:28,037 Stage-1 map = 100%,  reduce = 13%, Cumulative CPU 3.19 sec  2014-12-22 16:32:36,480 Stage-1 map = 100%,  reduce = 14%, Cumulative CPU 7.06 sec  2014-12-22 16:32:40,317 Stage-1 map = 100%,  reduce = 19%, Cumulative CPU 7.63 sec  2014-12-22 16:33:40,691 Stage-1 map = 100%,  reduce = 19%, Cumulative CPU 12.28 sec  2014-12-22 16:33:54,846 Stage-1 map = 100%,  reduce = 31%, Cumulative CPU 17.45 sec  2014-12-22 16:33:58,642 Stage-1 map = 100%,  reduce = 38%, Cumulative CPU 21.69 sec  2014-12-22 16:34:52,731 Stage-1 map = 100%,  reduce = 56%, Cumulative CPU 32.01 sec  2014-12-22 16:35:21,369 Stage-1 map = 100%,  reduce = 63%, Cumulative CPU 35.08 sec  2014-12-22 16:35:22,493 Stage-1 map = 100%,  reduce = 75%, Cumulative CPU 41.45 sec  2014-12-22 16:35:53,559 Stage-1 map = 100%,  reduce = 94%, Cumulative CPU 51.14 sec  2014-12-22 16:36:14,301 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 54.13 sec  MapReduce Total cumulative CPU time: 54 seconds 130 msec  Ended Job = job\_1419243806076\_0002  Loading data to table default.bucketed\_user partition (country=null)  Time taken for load dynamic partitions : 2421  Loading partition {country=AU}  Loading partition {country=country}  Loading partition {country=US}  Loading partition {country=UK}  Loading partition {country=CA}  Time taken for adding to write entity : 17  Partition default.bucketed\_user{country=AU} stats: [numFiles=32, numRows=500, totalSize=78268,rawDataSize=67936]  Partition default.bucketed\_user{country=CA} stats: [numFiles=32, numRows=500, totalSize=76564,rawDataSize=66278]  Partition default.bucketed\_user{country=UK} stats: [numFiles=32, numRows=500, totalSize=85604,rawDataSize=75292]  Partition default.bucketed\_user{country=US} stats: [numFiles=32, numRows=500, totalSize=75468,rawDataSize=65383]  Partition default.bucketed\_user{country=country} stats: [numFiles=32, numRows=1, totalSize=2865,rawDataSize=68]  MapReduce Jobs Launched:  Stage-Stage-1: Map: 1  Reduce: 32   Cumulative CPU: 54.13 sec   HDFS Read: 283505 HDFS Write:316247 SUCCESS  Total MapReduce CPU Time Spent: 54 seconds 130 msec  OK  Time taken: 396.486 seconds  user@tri03ws-386:~$ |

From the above box, we can see that mapreduce job initiated 32 reduce tasks for 32 buckets and four partitions are created by country.

#### ****Table Sampling in Hive****

Table Sampling in hive is nothing but extraction small fraction of data from the original large data sets. It is similar to LIMIT operator in Hive.

But below are the **difference between LIMIT and TABLESAMPLE in Hive**.

* In many cases a LIMIT clause executes the entire query, and then only returns a limited results.
* But Sampling will only select a portion of data to perform query.

Now we will do sampling on these bucketed tables to see the performance difference between bucketed and non-bucketed tables. Lets pull the records present in the last bucket of**bucketed\_user** table created above.

TABLESAMPLE example

Shell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | hive> SELECT firstname, country, state, city FROM bucketed\_user      > TABLESAMPLE(BUCKET 32 OUT OF 32 ON state);  OK  Carman    CA NL St. Johns  Chuck     CA NL St. Johns  Kristal   CA NL Paradise  Micah     CA NL St. Johns  ..............................  Man       UK Greater London Mildmay Ward  Lovetta   UK Greater London High Barnet Ward  Evette    UK Leicester Stone ygate Ward  Eulah     UK Greater London Bunhill Ward  Selene    UK Greater London West Wickham Ward  Kenda     UK Greater London Custom House Ward  ...........................  Abraham   UK Greater London Aldborough Ward  Dustin    UK Greater London Brockley Ward  Craig     UK Greater London East Putney Ward  ................  Lindsey   US CA Ontario  Justine   US CA Pomona  Tarra     US CA San Francisco  Kiley     US CA Los Angeles  ........................  Dorothy   US CA San Diego  Refugia   US CA Hayward  Time taken: 0.89 seconds, Fetched: 129 row(s)  hive> |

In the above sampling we can see the sample records from various countries and covering many states and cities, But if we use LIMIT operator on non-bucketed tables it will return either all the 129 records from first country CA or last country US but we can’t evenly distributed sample records from all countries and states. This can be seen in the below screen.

Sampling with LIMIT operator

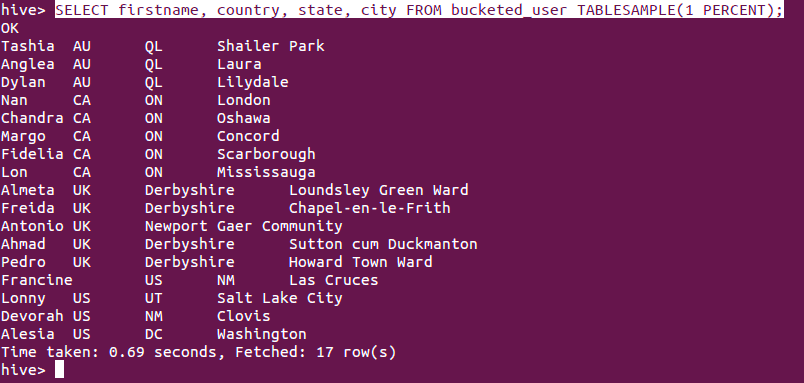
Shell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | hive> SELECT firstname, country, state, city FROM temp\_user LIMIT 129 ;  OK  first\_name     country   state   city  Rebbecca       AU        TA      Leith  Stevie         AU       QL     Proston  Mariko         AU        WA     Hamel  Gerardo        AU        NS      Talmalmo  Mayra          AU        NS     Lane Cove  Idella         AU       WA     Cartmeticup  Sherill        AU       WA      Nyamup  Ena            AU       NS     Bendick Murrell  Vince          AU       QL     Purrawunda  Theron         AU        SA      Blanchetown  Amira         AU       QL     Rockside  ...............  Louann         AU        QL     Wyandra  William        AU        QL     Goondi Hill  Time taken: 0.232 seconds, Fetched: 129 row(s)  user@tri03ws-386:~$ |

We can also perform random sampling with Hive with below syntax.

MySQL

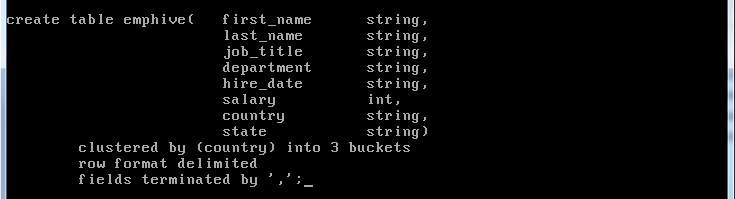
|  |  |
| --- | --- |
| 1  2 | hive> SELECT firstname, country, state, city FROM bucketed\_userTABLESAMPLE(1 PERCENT); |

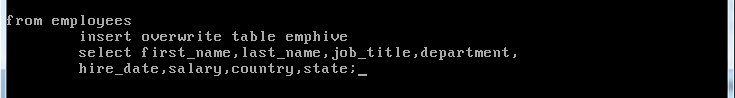


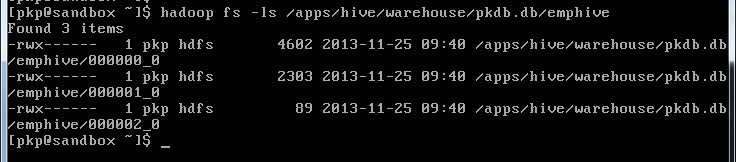
**Thus bucketing is most suitable Sampling purpose and also adds some optimization to query performance.**

### Hive Bucketed Tables and Sampling

Bucketing is a simple idea if you are already aware. You create multiple buckets. You read each record and place it into one of the buckets based on some logic mostly some kind of hashing algorithm. This allows you to organize your data by decomposing it into multiple parts. You might think that we do achieve same thing using partitioning then why bother about bucketing. There is one difference. When we do partitioning, we create a partition for each unique value of the column. This may burst into a situation where you might need to create thousands of tiny partitions. But if you use bucketing, you can limit it to a number which you choose and decompose your data into those buckets. In hive a partition is a directory but a bucket is a file. We will see it in action.

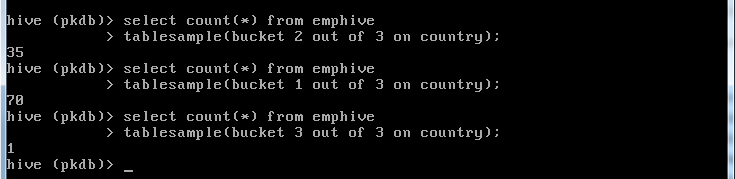
In hive, bucketing does not work by default. You will have to set following variable to enable bucketing. **set hive.enforce.bucketing=true;**Once you ensured that above variable is set, lets create a bucketed table. In my previous post, we had an external table named employees. We will create a bucketed hive table named emphive and load data from employees table. Once data is loaded into bucketed table, we will go back to our warehouse to check if we have multiple data files created which should be equal to the number of buckets in our table.  
Here is command to create a bucketed file. I have bucketed data on country column into 3 buckets.

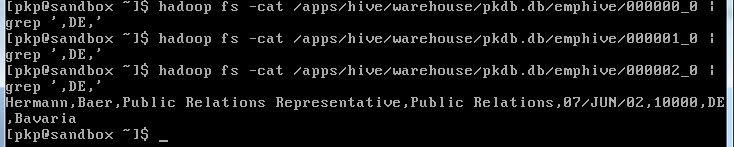
I have executed this command on hive prompt and now it’s time to load data, below screen shows command to load data.

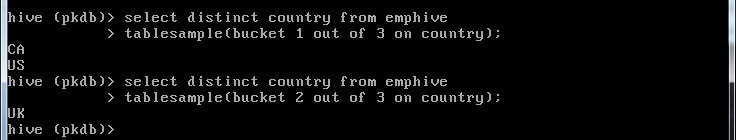
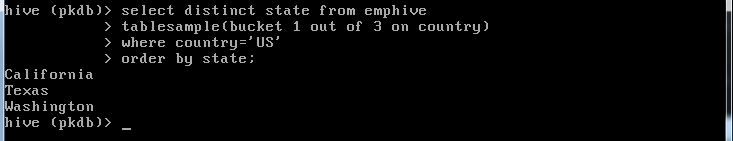
I have executed this command on hive prompt, now it’s time to go and check how many data files are created in our warehouse. If you have read all my hive articles, you already know where these files will be created. Check below screen and you will realize three files names as 000000\_0, 000001\_0 and 000002\_0 are created these are our data files.   
  


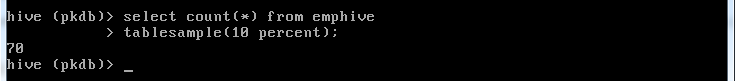
You might have question in your mind, how can we select data only from one bucket. I will answer that question. We can use sampling queries. Keep reading.

There are two benefits of bucketing. First, it is more efficient for certain types of queries in hive particularly join operation on two tables which are bucketed on same column. Second reason is your sampling queries are more efficient if they are performed on bucketed columns.   
Do you understand sampling? No worries let me explain it. You might have a situation when accuracy is not very important and you just need run your queries on a smaller set of data instead of entire table. One use case for such situation is testing scenarios. You have huge data in table which takes lot of time to read and return your queries and you want to test it on just a smaller sample of you data instead of entire set. Another use case is when you need to draw some patterns or need some aggregation like average and accuracy is not your top priority. A general sense with a predictable degree of error is fine. You may just need to draw your patterns on a smaller sample set of data instead of full data set. Sampling can also be used to obtain a random subset of the actual rows in a table for auditing purposes. Hive supports two types of sampling which we will explore through examples.

Let’s count how many records we have in our different buckets. We will use hive sampling query for this.

You noticed that 3rd bucket has just one record. Let’s check which country this record belongs to and does my first and second files contain data for this country. 

What does this mean, a specific country does not go into two buckets, that’s what we expect. Let’s check which countries bucket 1 and 2 contains?  
  
  
  
So, bucket 1 contains data for CA and US whereas bucket 2 contains data for UK. I hope you understood what hive has done when we inserted data into this bucketed table. Let me summarize. Hive created three buckets as I instructed it to do so in create table statement. When I loaded data into this table, hive has used some hashing technique for each country to generate a number in range of 1 to 3. Based on the outcome of hashing, hive has placed data row into appropriate bucked. This hashing technique is deterministic so that when hive has to read it back or write more data, it goes in correct bucket.  
One thing about tablesample clause is worth mentioning, it’s part of from clause so you can have where clause or other clauses like order by etc as usual. Check below example if you have any doubt.

Hive supports another type of sampling. That’s called block sampling. In this sampling we specify percentage of data to be sampled. Let’s see an example.

Oops! It doesn’t seem to be working correctly. I have total 106 records and I asked to sample 10% so I expect count to be something close to 10. How come it is 70? Answer is straight, because it is block sampling. When I asked hive to sample 10%, I actually asked to read approximately 10% blocks but I just have two blocks for my data into this table and minimum hive can read is one block. Yes, granularity of block sampling is at block level. For example, if your HDFS block size is 256MB, even if n% of input size is only 100MB, you get 256MB of data.

Keep Reading and Keep Learing.

## Partitioning in Hive

Table partitioning means dividing table data into some parts based on the values of particular columns like date or country, segregate the input records into different files/directories based on date or country.

Partitioning can be done based on more than column which will impose multi-dimensional structure on directory storage. For Example, In addition to partitioning log records by date column, we can also sup divide the single day records into country wise separate files by including country column into partitioning. We will see more about this in the examples.

Partitions are defined at the time of table creation using the PARTITIONED BY clause, with a list of column definitions for partitioning.

###### Syntax

MySQL

|  |  |
| --- | --- |
| 1  2  3 | CREATE [EXTERNAL] TABLE table\_name (col\_name\_1 data\_type\_1, ....)  PARTITIONED BY (col\_name\_n data\_type\_n [COMMENT col\_comment], ...); |

As shown in syntax, we can also add comments to partitioned columns.

##### ****Advantages****

* Partitioning is used for distributing execution load horizontally.
* As the data is stored as slices/parts, query response time is faster to process the small part of the data instead of looking for a search in the entire data set.
* For example, In a large user table where the table is partitioned by country, then selecting users of country ‘IN’ will just scan one directory ‘country=IN’ instead of all the directories.

##### ****Limitations****

* Having too many partitions in table creates large number of files and directories in HDFS, which is an overhead to NameNode since it must keep all metadata for the file system in memory only.
* Partitions may optimize some queries based on Where clauses, but may be less responsive for other important queries on grouping clauses.
* In Mapreduce processing, Huge number of partitions will lead to huge no of tasks (which will run in separate JVM) in each mapreduce job, thus creates lot of overhead in maintaining JVM start up and tear down. For small files, a separate task will be used for each file. In worst scenarios, the overhead of JVM start up and tear down can exceed the actual processing time.

##### ****Example Scenarios****

* Partitioning is used in real-time log files analysis to segregate the records based on time stamp or date value to see the results day wise quickly.
* Another real-time use is that, Customer/user details are partitioned by country/state or department for fast retrieval of subset data pertaining to some category.
* Sales records by-product type, country, year and month is another commonly used scenario.

In this post we will try examples of use case 2.

### ****Sample Use Case****

Lets explore the other features of partitions with the help of sample use case of Loading User records into Hive and performing some queries.

Sample User Records file for testing in this post –> [UserRecords](http://hadooptutorial.info/wp-content/uploads/2014/12/UserRecords.txt)

XHTML

|  |  |
| --- | --- |
| 1  2  3 | first\_name,last\_name,address,country,city,state,post,phone1,phone2,email,web  Rebbecca,Didio,171 E 24th St,AU,Leith,TA,7315,03-8174-9123,0458-665-290,rebbecca.didio@didio.com.au,http://www.brandtjonathanfesq.com.au |

###### Observation of Input Data

Input data has below fields or columns.

* First Name
* Last Name
* Address
* Country
* City
* State
* Postal Code
* Phone Number
* Alternative Phone Number
* Email Id
* Website URL

Easiest part is that, each field is separated by **‘,’** and no field contains the same **‘,’** in its values. Lets Assume we need to create Hive Table **partitioned\_user**partitioned by **Country** and **State** and load these input records into table is our requirement.

#### ****Creation of Partition Table****

##### ****Managed Partitioned Table****

Below is the HiveQL to create **managed partitioned\_user** table as per the above requirements.

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | CREATE TABLE partitioned\_user(  firstname VARCHAR(64),  lastname  VARCHAR(64),  address   STRING,  city   VARCHAR(64),  post      STRING,  phone1    VARCHAR(64),  phone2    STRING,  email     STRING,  web       STRING  )  PARTITIONED BY (country VARCHAR(64), state VARCHAR(64))  STORED AS SEQUENCEFILE; |

Note that we didn’t include **country** and **state** columns in table definition but included in partition definition. If we include them, then we will encounter [error scenario 1](http://hadooptutorial.info/partitioning-and-bucketing-in-hive/#error1). We can verify the partition columns of the table with the help of below command.

MySQL

|  |  |
| --- | --- |
| 1  2 | hive> DESCRIBE FORMATTED partitioned\_user; |

Partitioned columns **country** and **state** can be used in Query statements WHERE clause and can be treated regular column names even though there is actual column inside the input file data.

##### ****External Partitioned Tables****

We can create external partitioned tables as well, just by using the **EXTERNAL** keyword in the **CREATE**statement, but for creation of External Partitioned Tables, we do not need to mention LOCATION clause as we will mention locations of each partitions separately while inserting data into table.

### Inserting Data Into Partitioned Tables

Data insertion into partitioned tables can be done in two modes.

* Static Partitioning
* Dynamic Partitioning

### ****Static Partitioning in Hive****

In this mode, input data should contain the columns listed only in table definition (for example, firstname, lastname, address, city, post, phone1, phone2, email and web) but not the columns defined in partitioned by clause (country and state).

If our input column layout is according to the expected layout and we already have separate input files for each partitioned key value pairs, like one separate file for each combination of country and state values (country=US and state=CA), then these files can be easily loaded into partitioned tables with below syntax.

#### Loading Data into Managed Partitioned Table From Local FS

#### Example

For example, lets take below 3 records, which are not containing partitioned columns and save into**staticinput.txt** file on home directory. And assume that all these records belongs to**country=US and State=CA.**

XHTML

|  |  |
| --- | --- |
| 1  2  3  4  5 | first\_name,last\_name,address,city,post,phone1,phone2,email,web  Rebbecca,Didio,171 E 24th St,Leith,7315,03-8174-9123,0458-665-290,rebbecca.didio@didio.com.au,http://www.brandtjonathanfesq.com.au  Stevie,Hallo,22222 Acoma St,Proston,4613,07-9997-3366,0497-622-620,stevie.hallo@hotmail.com,http://www.landrumtemporaryservices.com.au  Mariko,Stayer,534 Schoenborn St #51,Hamel,6215,08-5558-9019,0427-885-282,mariko\_stayer@hotmail.com,http://www.inabinetmacreesq.com.au |

Now this file can be loaded into partitioned table with below syntax by specifying the country and statevalue at load time itself.

Static Partition Loading Syntax & Example

MySQL

|  |  |
| --- | --- |
| 1  2  3  4 | hive> LOAD DATA LOCAL INPATH '${env:HOME}/staticinput.txt'        INTO TABLE partitioned\_user        PARTITION (country = 'US', state = 'CA'); |

This will create separate directory under the default warehouse directory in HDFS.

Shell

|  |  |
| --- | --- |
| 1  2 | /user/hive/warehouse/partitioned\_user/country=US/state=CA/ |

Similarly we have to add other partitions, which will create corresponding directories in HDFS. Or else we can load the entire directory into Hive table with single command and can add partitions for each file with [**ALTER**](http://hadooptutorial.info/partitioning-and-bucketing-in-hive/#alter) command.

Static Partition Loading Syntax & Example

MySQL

|  |  |
| --- | --- |
| 1  2  3 | hive> LOAD DATA LOCAL INPATH '${env:HOME}/inputdir'        INTO TABLE partitioned\_user; |

###### Loading Partition From Other Table

We can load or add partitions with query results from another table as shown below.

Loading Partition from another table

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5 | hive> INSERT OVERWRITE TABLE partitioned\_user        PARTITION (country = 'US', state = 'AL')        SELECT \* FROM another\_user au        WHERE au.country = 'US' AND au.state = 'AL'; |

Overwriting Existing Partition

We can overwrite an existing partition with help of OVERWRITE INTO TABLE partitioned\_user clause.

##### Loading Data into External Partitioned Table From HDFS

There is alternative for bulk loading of partitions into hive table. As data is already present in HDFS and should be made accessible by Hive, we will just mention the locations of the HDFS files for each partition.

If our files are on Local FS, they can be moved to a directory in HDFS and we can add partition for each file in that directory with commands similar to below.

MySQL

|  |  |
| --- | --- |
| 1  2  3 | hive> ALTER TABLE partitioned\_user ADD PARTITION (country = 'US', state ='CA')  LOCATION '/hive/external/tables/user/country=us/state=ca' |

Similarly we need to repeat the above alter command for all partition files in the directory so that a meta data entry will be created in metastore, mapping the partition and table.

### Dynamic Partitioning in Hive

Instead of loading each partition with single SQL statement as shown above, which will result in writing lot of SQL statements for huge no of partitions, Hive supports dynamic partitioning with which we can add any number of partitions with single SQL execution. Hive will automatically splits our data into separate partition files based on the values of partition keys present in the input files.

It gives the advantages of easy coding and no need of manual identification of partitions. This dynamic partition suits well for our example requirement on user records provided above.

For dynamic partition loading we will not provide the values for partition keys, as shown below for previously seen query.

Dynamic Partition Loading Example from Another table

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | hive>INSERT INTO TABLE partitioned\_user  PARTITION (country, state)          SELECT  firstname ,  lastname  ,  address   ,          city      ,  post      ,  phone1    ,  phone2    ,  email     ,  web       ,  country   ,          state  FROM temp\_user; |

We can also mix dynamic and static partitions by specifying it as **PARTITION(country = ‘US’, state)**. But static partition keys must come before the dynamic partition keys.

But by default, Dynamic Partitioning is disabled in Hive to prevent accidental partition creations. To use dynamic partitioning we need to set below properties either in **Hive Shell** or in **hive-site.xml** file.

Dynamic Partitioning Properties in hive-site.xml

XHTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <property>      <name>hive.exec.dynamic.partition</name>      <value>true</value>      <description>Whether or not to allow dynamic partitions in DML/DDL.</description>    </property>    <property>      <name>hive.exec.dynamic.partition.mode</name>      <value>nonstrict</value>      <description>        In strict mode, the user must specify at least one static partition        in case the user accidentally overwrites all partitions.        In nonstrict mode all partitions are allowed to be dynamic.      </description>    </property>    <property>      <name>hive.exec.max.dynamic.partitions</name>      <value>1000</value>      <description>Maximum number of dynamic partitions allowed to be created in total.</description>    </property>    <property>      <name>hive.exec.max.dynamic.partitions.pernode</name>      <value>1000</value>      <description>Maximum number of dynamic partitions allowed to be created in each mapper/reducer node.</description>    </property>    <property> |

We can set these through hive shell with below commands,

Shell

|  |  |
| --- | --- |
| 1  2  3  4  5 | set hive.exec.dynamic.partition=true;  set hive.exec.dynamic.partition.mode=nonstrict;  set hive.exec.max.dynamic.partitions=1000;  set hive.exec.max.dynamic.partitions.pernode=1000; |

Here, except dynamic partition mode property, remaining three are self explanatory. By default, dynamic partition mode is set to strict in Hive to specify at least one static partition column (key), which prevents generation huge no of partitions by a badly designed query.

Note that, this **dynamic partition strict mode** is different from **hive.mapred.mode=strict,** but it is right time to discuss about mapreduce strict mode also, because if this property is set to strict, then we cannot certain queries on partitioned tables as well.

###### ****Strict Mode in Hive****

In mapreduce strict mode (**hive.mapred.mode=strict**) , some risky queries are not allowed to run. They include:

* Cartesian Product.
* No partition being picked up for a query.
* Comparing bigints and strings.
* Comparing bigints and doubles.
* Orderby without limit.

According to point 2 and 5, we can not use SELECT statements without at least one partition key filter (like WHERE country=’US’) or ORDER BY clause without LIMIT condition on partitioned tables. But by default this property is set to **nonstrict**.

### Example Use Case

With the help of above concepts lets create the dynamic partitioned table for the user records provided on first page of this post –> [UserRecords](http://hadooptutorial.info/wp-content/uploads/2014/12/UserRecords.txt)

As this input file contains, partitions keys also as fields in each record, we need to create temporary user with the all the columns present in the input file and from that we need to extract the columns needed into partition table by keeping country and state columns as partition keys.

Lets also examine the performance of this temporary table and partitioned table for a SELECT statement with WHERE clause to confirm the performance improve on partitioned table.

We can create this temporary table and partitioned table and also load partitioned table dynamically with the help of below HiveQL. After going through all the above concepts, this HiveQL will be self explanatory, so we are not providing details/description of implementation but just the results.

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | set hive.exec.dynamic.partition=true;  set hive.exec.dynamic.partition.mode=nonstrict;  set hive.exec.max.dynamic.partitions.pernode=1000;    DROP TABLE IF EXISTS partitioned\_user;    CREATE TEMPORARY TABLE temp\_user(  firstname VARCHAR(64),  lastname  VARCHAR(64),  address   STRING,  country   VARCHAR(64),  city      VARCHAR(64),  state     VARCHAR(64),  post      STRING,  phone1    VARCHAR(64),  phone2    STRING,  email     STRING,  web       STRING  )  ROW FORMAT DELIMITED  FIELDS TERMINATED BY ','  LINES TERMINATED BY '\n'          STORED AS TEXTFILE;    LOAD DATA LOCAL INPATH '/home/siva/UserRecords.txt' INTO TABLEtemp\_user;    SELECT firstname, phone1, city  FROM temp\_user  WHERE country='US' AND state='CA'  ORDER BY city  LIMIT 5;    CREATE TABLE partitioned\_user(  firstname VARCHAR(64),  lastname  VARCHAR(64),  address   STRING,  city   VARCHAR(64),  post      STRING,  phone1    VARCHAR(64),  phone2    STRING,  email     STRING,  web       STRING  )  PARTITIONED BY (country VARCHAR(64), state VARCHAR(64))  STORED AS SEQUENCEFILE;    INSERT INTO TABLE partitioned\_user  PARTITION (country, state)          SELECT  firstname ,  lastname  ,  address   ,  city      ,  post      ,  phone1    ,  phone2    ,  email     ,  web       ,  country   ,  state  FROM temp\_user;    SELECT firstname, phone1, city  FROM partitioned\_user  WHERE country='US' AND state='CA'  ORDER BY city  LIMIT 5; |

#### Output of the above script execution

Shell

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106 | hadoop1@ubuntu-1:~$ hive -f partitioned\_table.hql  Logging initialized using configuration in jar:file:/usr/lib/hive/apache-hive-0.14.0-bin/lib/hive-common-0.14.0.jar!/hive-log4j.properties  OK  Time taken: 3.283 seconds  OK  Time taken: 0.191 seconds  Loading data to table default.temp\_user  Table default.temp\_user stats: [numFiles=1, totalSize=283212]  OK  Time taken: 0.24 seconds  Query ID = hadoop1\_20141209223333\_23e2d242-c70b-414b-856c-5b3b1e943691  Total jobs = 1  Launching Job 1 out of 1  Number of reduce tasks determined at compile time: 1  In order to change the average load for a reducer (in bytes):    set hive.exec.reducers.bytes.per.reducer=  In order to limit the maximum number of reducers:    set hive.exec.reducers.max=  In order to set a constant number of reducers:    set mapreduce.job.reduces=  Starting Job = job\_1418142386286\_0009, Tracking URL = http://localhost:8088/proxy/application\_1418142386286\_0009/  Kill Command = /usr/lib/hadoop/hadoop-2.3.0/bin/hadoop job  -kill job\_1418142386286\_0009  Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  2014-12-09 22:33:31,091 Stage-1 map = 0%,  reduce = 0%  2014-12-09 22:33:37,570 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec  2014-12-09 22:33:45,171 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.01 sec  MapReduce Total cumulative CPU time: 4 seconds 10 msec  Ended Job = job\_1418142386286\_0009  MapReduce Jobs Launched:  Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.01 sec   HDFS Read: 283510 HDFS Write: 157 SUCCESS  Total MapReduce CPU Time Spent: 4 seconds 10 msec  OK  Venita 714-523-6653 Anaheim  Merissa 562-579-6900 Bellflower  Joesph 510-677-9785 Berkeley  Louvenia 310-820-2117 Beverly Hills  Chau 310-560-8022 Beverly Hills  Time taken: 27.552 seconds, Fetched: 5 row(s)  OK  Time taken: 0.266 seconds  Query ID = hadoop1\_20141209223333\_c7b14643-0658-4b28-87a6-5172189e1299  Total jobs = 3  Launching Job 1 out of 3  Number of reduce tasks is set to 0 since there's no reduce operator  Starting Job = job\_1418142386286\_0010, Tracking URL = http://localhost:8088/proxy/application\_1418142386286\_0010/  Kill Command = /usr/lib/hadoop/hadoop-2.3.0/bin/hadoop job  -kill job\_1418142386286\_0010  Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0  2014-12-09 22:34:00,472 Stage-1 map = 0%,  reduce = 0%  2014-12-09 22:34:12,557 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 5.01 sec  MapReduce Total cumulative CPU time: 5 seconds 10 msec  Ended Job = job\_1418142386286\_0010  Stage-4 is selected by condition resolver.  Stage-3 is filtered out by condition resolver.  Stage-5 is filtered out by condition resolver.  Moving data to: hdfs://localhost:9000/tmp/mydir/hadoop1/7bebc4dd-f37c-44c2-8399-2acb84f7360d/hive\_2014-12-09\_22-33-47\_744\_8342706472640942260-1/-ext-10000  Loading data to table default.partitioned\_user partition (country=null, state=null)  Time taken for load dynamic partitions : 21527  Loading partition {country=UK, state=North Lanarkshire}  Loading partition {country=UK, state=Leicestershire}  Loading partition {country=US, state=UT}  Loading partition {country=US, state=MI}  Loading partition {country=CA, state=NL}  ................................................  Loading partition {country=UK, state=Neath Port Talbot}  Loading partition {country=UK, state=Moray}  Time taken for adding to write entity : 43  Partition default.partitioned\_user{country=AU, state=AC} stats: [numFiles=1, numRows=6, totalSize=951, rawDataSize=782]  Partition default.partitioned\_user{country=AU, state=NS} stats: [numFiles=1, numRows=125, totalSize=18603, rawDataSize=16640]  Partition default.partitioned\_user{country=AU, state=NT} stats: [numFiles=1, numRows=7, totalSize=1105, rawDataSize=922]  ...................................................................  Partition default.partitioned\_user{country=US, state=WY} stats: [numFiles=1, numRows=3, totalSize=517, rawDataSize=389]  Partition default.partitioned\_user{country=country, state=state} stats: [numFiles=1, numRows=1, totalSize=162, rawDataSize=62]  MapReduce Jobs Launched:  Stage-Stage-1: Map: 1   Cumulative CPU: 6.48 sec   HDFS Read: 283510 HDFS Write: 319558 SUCCESS  Total MapReduce CPU Time Spent: 6 seconds 480 msec  OK  Time taken: 62.443 seconds  Query ID = hadoop1\_20141209223434\_5d009092-312d-4ec8-abcf-e86c8dd5cfa4  Total jobs = 1  Launching Job 1 out of 1  Number of reduce tasks determined at compile time: 1  In order to change the average load for a reducer (in bytes):    set hive.exec.reducers.bytes.per.reducer=  In order to limit the maximum number of reducers:    set hive.exec.reducers.max=  In order to set a constant number of reducers:    set mapreduce.job.reduces=  Starting Job = job\_1418142386286\_0011, Tracking URL = http://localhost:8088/proxy/application\_1418142386286\_0011/  Kill Command = /usr/lib/hadoop/hadoop-2.3.0/bin/hadoop job  -kill job\_1418142386286\_0011  Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  2014-12-09 22:34:57,577 Stage-1 map = 0%,  reduce = 0%  2014-12-09 22:35:03,932 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.36 sec  2014-12-09 22:35:10,227 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.93 sec  MapReduce Total cumulative CPU time: 2 seconds 930 msec  Ended Job = job\_1418142386286\_0011  MapReduce Jobs Launched:  Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.93 sec   HDFS Read: 10863 HDFS Write: 157 SUCCESS  Total MapReduce CPU Time Spent: 2 seconds 930 msec  OK  Venita 714-523-6653 Anaheim  Merissa 562-579-6900 Bellflower  Joesph 510-677-9785 Berkeley  Louvenia 310-820-2117 Beverly Hills  Chau 310-560-8022 Beverly Hills  Time taken: 22.158 seconds, Fetched: 5 row(s) |

We can see the partitioned table query resulted in **22 sec** whereas temp\_user table resulted in **28 sec**for the same query. Thus we can confirm the speed of partitioned tables over regular tables.

###### ****Note****

When inserting data into a partition, it’s necessary to include the partition columns as the last columns in the query. The column names in the source query don’t need to match the partition column names, but they really do need to be last.

Refer next page for further commands on Partitions.

Below are a few more commands that are supported on Hive partitioned tables.

* Show Command
* Describe Command
* Alter Command

**Show Partitions**

We can see the partitions of a partitioned table with SHOW command as shown below.

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | hive> SHOW PARTITIONS partitioned\_user;  OK  country=AU/state=AC  country=AU/state=NS  country=AU/state=NT  country=AU/state=QL  country=AU/state=SA  country=AU/state=TA  country=AU/state=VI  country=AU/state=WA  country=CA/state=AB  country=CA/state=BC  country=CA/state=MB |

If we have a lot of partitions and want to see partitions for particular partition keys, we can further restrict the command with an optional PARTITION clause that specifies one or more of the partitions with specific values.

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | hive> SHOW PARTITIONS partitioned\_user PARTITION(country='US');  OK  country=US/state=AK  country=US/state=AR  country=US/state=AZ  country=US/state=CA  country=US/state=CO  country=US/state=CT  country=US/state=DC  country=US/state=FL  country=US/state=GA |

**Describe partitions**

As we already know how to see the descriptions of tables, Now we can see the descriptions of each partition with commands similar to below.

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | hive> DESCRIBE FORMATTED partitioned\_user PARTITION(country='US',state='CA');  OK  # col\_name             data\_type           comment  firstname           varchar(64)  lastname             varchar(64)  address             string  city                 varchar(64)  post                 string  phone1               varchar(64)  phone2               string  email               string  web                 string  # Partition Information  # col\_name             data\_type           comment  country             varchar(64)  state               varchar(64)  # Detailed Partition Information  Partition Value:     [US, CA]  Database:           default  Table:               partitioned\_user  CreateTime:         Tue Dec 09 22:34:30 IST 2014  LastAccessTime:     UNKNOWN  Protect Mode:       None  Location:          hdfs://localhost:9000/user/hive/warehouse/partitioned\_user/country=US/state=CA  Partition Parameters:  COLUMN\_STATS\_ACCURATE true  numFiles             1  numRows             72  rawDataSize         9358  totalSize           10527  transient\_lastDdlTime 1418144688  # Storage Information  SerDe Library:       org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe  InputFormat:         org.apache.hadoop.mapred.SequenceFileInputFormat  OutputFormat:       org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat  Compressed:         No  Num Buckets:         -1  Bucket Columns:     []  Sort Columns:       []  Storage Desc Params:  serialization.format 1  Time taken: 0.173 seconds, Fetched: 44 row(s) |

**Alter Partitions**

We can alter/change partitions (add/change/drop) with the help of below commands.

**Adding Partitions**

We can add partitions to an existing table with ADD PARTITION clause as shown below.

MySQL

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ALTER TABLE partitioned\_user ADD IF NOT EXISTS  PARTITION (country = 'US', state = 'XY') LOCATION'/hdfs/external/file/path1'  PARTITION (country = 'CA', state = 'YZ') LOCATION'/hdfs/external/file/path2'  PARTITION (country = 'UK', state = 'ZX') LOCATION'/hdfs/external/file/path2'  ...; |

**Changing Partitions**

We can change a partition location with commands like below. This command does not move the data from the old location and does not delete the old data but the reference to old data file will be lost.

MySQL

|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE partitioned\_user PARTITION (country='US', state='CA')  SET LOCATION '/hdfs/partition/newpath'; |

**Drop Partitions**

We can drop partitions of a table with DROP IF EXISTS PARTITION clause as shown below.

MySQL

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE partitioned\_user DROP IF EXISTS PARTITION(country='US',state='CA'); |

**Other rarely used alter commands on partitions**

The ARCHIVE PARTITION clause captures the partition files into a Hadoop archive (HAR) file.This only reduces the number of files in the filesystem, reducing the load on the NameNode, but doesn’t provide any space savings.

MySQL

|  |  |
| --- | --- |
| 1  2 | ALTER TABLE log\_messages ARCHIVE PARTITION(country='US',state='XZ'); |

we can un archive these with UNARCHIVE PARTITION clause.

The following statements prevent the partition from being dropped and queried.

MySQL

|  |  |
| --- | --- |
| 1  2  3 | ALTER TABLE partitioned\_user PARTITION(country='US',state='XY') ENABLENO\_DROP;  ALTER TABLE partitioned\_user PARTITION(country='US',state='XY') ENABLEOFFLINE; |

To reverse either operation, replace ENABLE with DISABLE.

Refer Next page for frequent error scenarios that occur in partitioned tables.

###### ****Error Scenario 1****

While creating Partitioned tables based on particular columns, then these columns should not be defined in the column types of the table as they will be included in PARTITION BY clause. If we declare then in the definition as well as shown below,

MySQL

|  |  |
| --- | --- |
| 1  2 | <span style="color: #000000;">CREATE TABLE emp(name VARCHAR, id INT, deptSTRING, salary DECIMAL(7,2)) PARTITION BY (dept STRING);</span> |

Then we will receive below error message, as we have included dept STRING table definition, by removing which, the issue will be solved.

Shell

|  |  |
| --- | --- |
| 1  2 | <span style="color: #000000;">"FAILED: Error in semantic analysis: Column repeated in partitioning columns,"</span> |

###### ****Error Scenario 2****

When we do not set **hive.exec.max.dynamic.partitions.pernode** to appropriate value then we will receive error messages like below.

XHTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Caused by: org.apache.hadoop.hive.ql.metadata.HiveException: Hive Runtime Error whileprocessing row {"firstname":"Eveline","lastname":"Brickhouse","address":"288 N 168th Ave #266","country":"AU","city":"Camberwell West","state":"VI","post":"3124","phone1":"03-9517-9800","phone2":"0463-242-525","email":"[[email protected]](http://hadooptutorial.info/cdn-cgi/l/email-protection)","web":"http://www.firstexpress.com.au"}  at org.apache.hadoop.hive.ql.exec.MapOperator.process(MapOperator.java:503)  at org.apache.hadoop.hive.ql.exec.mr.ExecMapper.map(ExecMapper.java:176)  ... 8 more  Caused by: org.apache.hadoop.hive.ql.metadata.HiveFatalException: [Error 20004]: Fatalerror occurred when node tried to create too many dynamic partitions. The maximum numberof dynamic partitions is controlled by hive.exec.max.dynamic.partitions andhive.exec.max.dynamic.partitions.pernode. Maximum was set to: 100  atorg.apache.hadoop.hive.ql.exec.FileSinkOperator.getDynOutPaths(FileSinkOperator.java:869)  at org.apache.hadoop.hive.ql.exec.FileSinkOperator.processOp(FileSinkOperator.java:649)  at org.apache.hadoop.hive.ql.exec.Operator.forward(Operator.java:815)  at org.apache.hadoop.hive.ql.exec.SelectOperator.processOp(SelectOperator.java:84)  at org.apache.hadoop.hive.ql.exec.Operator.forward(Operator.java:815)  at org.apache.hadoop.hive.ql.exec.TableScanOperator.processOp(TableScanOperator.java:95)  at org.apache.hadoop.hive.ql.exec.MapOperator$MapOpCtx.forward(MapOperator.java:157)  at org.apache.hadoop.hive.ql.exec.MapOperator.process(MapOperator.java:493)  ... 9 more |

###### ****Solution 2****

We need to override the values of below two properties to at least 1000 per each value. This can be done in **hive-site.xml** file or in **hive CLI shell** with the help of SET commands.

XHTML

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <property>      <name>hive.exec.max.dynamic.partitions</name>      <value>1000</value>      <description>Maximum number of dynamic partitions allowed to be created in total.</description>    </property>    <property>      <name>hive.exec.max.dynamic.partitions.pernode</name>      <value>100</value>      <description>Maximum number of dynamic partitions allowed to be created in each mapper/reducer node.</description>    </property> |

###### ****References****

# Skew Join. Reduce = 99% or Skewed Joins in Hive

Posted on [October 7, 2014](http://dmtolpeko.com/2014/10/07/reduce-99-or-skewed-joins-in-hive/) by [admin](http://dmtolpeko.com/author/admin/)

Often running a HQL query you may notice that it progresses to 99% reduce stage quite fast and then stucks:

...

2014-10-07 08:46:01,149 Stage-8 map = 100%, reduce = 99%, Cumulative CPU 6905.85 sec

2014-10-07 08:47:01,361 Stage-8 map = 100%, reduce = 99%, Cumulative CPU 6999.45 sec

2014-10-07 08:48:01,441 Stage-8 map = 100%, reduce = 99%, Cumulative CPU 7065.59 sec

2014-10-07 08:49:01,670 Stage-8 map = 100%, reduce = 99%, Cumulative CPU 7125.26 sec

2014-10-07 08:50:01,808 Stage-8 map = 100%, reduce = 99%, Cumulative CPU 7188.12 sec

The problem is that Hive estimates the progress depending on the number of reducers completed, and this does not always relevant to the actual execution progress. It is possible that a query can reach 99% in 1 minute and then execute remaining 1% during 1 hour.

The most typical reason of this behavior is skewed data. For example, assume that you have a table that tracks all visits to the specific sites and SITE.COM has 100M rows while there are a dozen of other sites SITE.ORG, SITE.NET etc. that have just 10K visitors each.

Then when you join this table with another by site name, one reducer has to process 100M rows while other reducers process just 10K rows each.

So if you have 99 sites having 10K visitors, single site having 100M visitors and specify 100 reducers then 99% of reducers will finish their work very quickly and you have to wait for a long time when the last reducer terminates.

**Not only joins**

Data skew issue can arise not only in joins. For example, if you perform a GROUP BY SITE\_NAME in our example then a single reducer has to deal with 100M rows while others have to process much smaller number of rows.

**Concepts::**

You need to use the special **hiveconf** for variable substitution. e.g.

hive> set CURRENT\_DATE='2012-09-16';  
hive> select \* from foo where day >= '${hiveconf:CURRENT\_DATE}'

[**Convert String to Integer**](http://stackoverflow.com/questions/12346750/hive-convert-string-to-integer)**::**

cast(str\_column as int)

**DateDiff::**

http://www.folkstalk.com/2011/11/date-functions-in-hive.html

datediff(from\_unixtime(unix\_timestamp(),'yyyy-MM-dd'),from\_unixtime(unix\_timestamp(f.birth\_dt,'yyyy-MM-dd'),'yyyy-MM-dd'))

**STDDEV\_POP::**  
Now, let’s do something more interesting with our dataset:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | hive&gt; SELECT              AVG(age),              STDDEV\_POP(age)            FROM              table1            WHERE              age &gt;= 21              AND entry\_date='2012-12-12'; |

The above query finds out the average age and standard deviation in our dataset by calling avg and stddev\_pop User Definition Functions (UDFs) on the age column for people whose age is greater than 21. The result looks like this:

|  |  |
| --- | --- |
| 1 | 27.5    6.5 |

# Diagnose sporadic SQL response time

1 - Monitor RAM and CPU with [vmstat](http://www.dba-oracle.com/unix_linux/sadc_utility.html):  
  
2 - Monitor for disk enqueues with [*iostat*](http://www.dba-oracle.com/art_disk_iostat.htm):  
  
3 - Monitor network with netstat

**Where to check Hive issue?**

HIVE

* Hive log
* Hive Plan

HADOOP

* Hadoop Job Log
* Mapreduce Monitoring

SYSTEM

**Distributed By Clause**

**SORT BY**

Hive uses the columns in SORT BY to sort the rows before feeding the rows to a reducer. The sort order will be dependent on the column types. If the column is of numeric type, then the sort order is also in numeric order. If the column is of string type, then the sort order will be lexicographical order.

Ordering : It orders data at each of ‘N’ reducers , but each reducer can have overlapping ranges of data.

Outcome : N or more sorted files with overlapping ranges.

Let’s understand with an example :-

SELECT key, value FROM src SORT BY key ASC, value DESC

The query had 2 reducers, and the output of each is:

Reducer 1 :

0 5

0 3

3 6

9 1

Reducer 2 :

0 4

0 3

1 1

2 5

As, we can see, each reducer output is ordered but total ordering is missing , since we end up with multiple outputs per reducer.

**ORDER BY**

This is similar to ORDER BY in SQL Language.

In Hive, ORDER BY guarantees total ordering of data, but for that it has to be passed on to a single reducer, which is normally unacceptable and therefore in strict mode, hive makes it compulsory to use LIMIT with ORDER BY so that reducer doesn’t get overburdened.

Ordering : Total Ordered data.

Outcome : Single output i.e. fully ordered.

For example :

SELECT key, value FROM src ORDER BY key ASC, value DESC

Reducer :

0 5

0 4

0 3

0 3

1 1

2 5

3 6

9 1

**DISTRIBUTE BY**

Hive uses the columns in Distribute By to distribute the rows among reducers. All rows with the same Distribute By columns will go to the same reducer.

It ensures each of N reducers gets non-overlapping ranges of column, but doesn’t sort the output of each reducer. You end up with N or more unsorted files with non-overlapping ranges.

Example ( taken directly from Hive wiki ):-

We are Distributing By x on the following 5 rows to 2 reducer:

x1

x2

x4

x3

x1

Reducer 1 got

x1

x2

x1

Reducer 2 got

x4

x3

Note that all rows with the same key x1 is guaranteed to be distributed to the same reducer (reducer 1 in this case), but they are not guaranteed to be clustered in adjacent positions.

**CLUSTER BY**

Cluster By is a short-cut for both Distribute By and Sort By.

CLUSTER BY x ensures each of N reducers gets non-overlapping ranges, then sorts by those ranges at the reducers.

Ordering : Global ordering between multiple reducers.

Outcome : N or more sorted files with non-overlapping ranges.

For the same example as above , if we use Cluster By x, the two reducers will further sort rows on x:

Reducer 1 got

x1

x1

x2

Reducer 2 got

x3

x4

Instead of specifying Cluster By, the user can specify Distribute By and Sort By, so the partition columns and sort columns can be different.

References : –

[1] http://stackoverflow.com/questions/13715044/hive-cluster-by-vs-order-by-vs-sort-by

[2] https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy#LanguageManualSortBy-SyntaxofOrderBy

**Explode Function**

hive> create table person(name string,work\_locations array<string>)

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '\t'

> COLLECTION ITEMS TERMINATED BY ',';

OK

Time taken: 0.832 seconds

hive> LOAD DATA LOCAL INPATH '/home/ubuntu/practicedata/hive\_array\_struct\_map.txt' OVERWRITE INTO TABLE person;

Copying data from file:/home/ubuntu/practicedata/hive\_array\_struct\_map.txt

Copying file: file:/home/ubuntu/practicedata/hive\_array\_struct\_map.txt

Loading data to table default.person

Table default.person stats: [num\_partitions: 0, num\_files: 1, num\_rows: 0, total\_size: 73, raw\_data\_size: 0]

OK

Time taken: 11.309 seconds

hive> select \* from person;

OK

biansutao ["beijing","shanghai","tianjin","hangzhou"]

linan ["changchu","chengdu","wuhan"]

Time taken: 0.301 seconds, Fetched: 2 row(s)

hive> select explode(work\_locations) from person;

Total MapReduce jobs = 1

Launching Job 1 out of 1

Number of reduce tasks is set to 0 since there's no reduce operator

Starting Job = job\_201505071017\_0001, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job\_201505071017\_0001

Kill Command = /usr/local/hadoop/bin/../bin/hadoop job -kill job\_201505071017\_0001

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0

2015-05-07 13:01:09,102 Stage-1 map = 0%, reduce = 0%

2015-05-07 13:01:12,125 Stage-1 map = 100%, reduce = 0%

2015-05-07 13:01:15,205 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_201505071017\_0001

MapReduce Jobs Launched:

Job 0: Map: 1 HDFS Read: 73 HDFS Write: 57 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

beijing

shanghai

tianjin

hangzhou

changchu

chengdu

wuhan

Time taken: 15.451 seconds, Fetched: 7 row(s)

hive>

hive> select name,work\_loc from person

> lateral view explode(work\_locations) adTable as work\_loc;

Total MapReduce jobs = 1

Launching Job 1 out of 1

Number of reduce tasks is set to 0 since there's no reduce operator

Starting Job = job\_201505071017\_0002, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job\_201505071017\_0002

Kill Command = /usr/local/hadoop/bin/../bin/hadoop job -kill job\_201505071017\_0002

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0

2015-05-07 13:10:31,505 Stage-1 map = 0%, reduce = 0%

2015-05-07 13:10:34,529 Stage-1 map = 100%, reduce = 0%

2015-05-07 13:10:37,539 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_201505071017\_0002

MapReduce Jobs Launched:

Job 0: Map: 1 HDFS Read: 73 HDFS Write: 115 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

biansutao beijing

biansutao shanghai

biansutao tianjin

biansutao hangzhou

linan changchu

linan chengdu

linan wuhan

Time taken: 10.532 seconds, Fetched: 7 row(s)

**Ref::\_ https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView**

**Using Hive Data types arrays, maps , structs and unions**

For now, hive supports following composite data type：

* **map**: (key1, value1, key2, value2, …). Creates a map with the given key/value pairs
* **struct**: (val1, val2, val3, …). Creates a struct with the given field values. Struct field names will be col1, col2, ...
* **named\_struct**: (name1, val1, name2, val2, …). Creates a struct with the given field names and values. (as of Hive 0.8.0)
* **array**: (val1, val2, …). Creates an array with the given elements
* **create\_union**:(tag, val1, val2, …). Creates a union type with the value that is being pointed to by the tag parameter

Here, I list some difference between map and struct

* For map, the type of key and values is unified. Struct are more flexible
* To access map using array like access ['key']; struct use "." access
* Struct is more like a table while map is more like a array with customized index

Below are samples of their usage

##### 1. Array

Create table using array data type

01. create table person(name string,work\_locations **array**<**string**>)

02. ROW FORMAT DELIMITED

03. FIELDS TERMINATED BY '\t'

04. COLLECTION ITEMS TERMINATED BY ',';

Prepared data as follows

01. cat person.txt

02. biansutao beijing,shanghai,tianjin,hangzhou

03. linan changchu,chengdu,wuhan

Load data

01. LOAD DATA LOCAL INPATH '/home/hadoop/person.txt' OVERWRITE INTO TABLE person;

Query the table

01. hive> select \* from person;

02. biansutao ["beijing","shanghai","tianjin","hangzhou"]

03. linan ["changchu","chengdu","wuhan"]

04. Time taken: 0**.355** seconds

05. hive> select name from person;

06. linan

07. biansutao

08. Time taken: 12**.397** seconds

09. hive> select work\_locations[0] from person;

10. changchu

11. beijing

12. Time taken: 13**.214** seconds

13. hive> select work\_locations from person;

14. ["changchu","chengdu","wuhan"]

15. ["beijing","shanghai","tianjin","hangzhou"]

16. Time taken: 13**.755** seconds

17. hive> select work\_locations[3] from person;

18. NULL

19. hangzhou

20. Time taken: 12**.722** seconds

21. hive> select work\_locations[4] from person;

22. NULL

23. NULL

24. Time taken: 15**.958** seconds

##### 2. Map

Create table using array data type

01. create table score(name string, score **map**<**string**,int>)

02. ROW FORMAT DELIMITED

03. FIELDS TERMINATED BY '\t'

04. COLLECTION ITEMS TERMINATED BY ','

05. MAP KEYS TERMINATED BY ':';

Prepared data as follows

01. cat score**.txt**

02. biansutao 'math':80,'chinese':89,'english':95

03. jobs 'chinese':60,'math':80,'english':99

Load data

01. LOAD DATA LOCAL INPATH '/home/hadoop/score.txt' OVERWRITE INTO TABLE score;

Query the table

01. hive> select \* from score;

02. biansutao {"math":80,"chinese":89,"english":95}

03. jobs {"chinese":60,"math":80,"english":99}

04. Time taken: 0.665 seconds

05. hive> select name from score;

06. jobs

07. biansutao

08. Time taken: 19.778 seconds

09. hive> select t.score from score t;

10. {"chinese":60,"math":80,"english":99}

11. {"math":80,"chinese":89,"english":95}

12. Time taken: 19.353 seconds

13. hive> select t.score['chinese'] from score t;

14. 2660

15. 2789

16. Time taken: 13.054 seconds

17. hive> select t.score['english'] from score t;

18. 3099

19. 3195

20. Time taken: 13.769 seconds

##### 3. Struct

Create table using array data type

01. create table test(id **int**,course **struct**<course:string,score:**int**>)

02. ROW FORMAT DELIMITED

03. FIELDS TERMINATED BY '\t'

04. COLLECTION ITEMS TERMINATED BY ',';

Prepared data as follows

01. cat test.txt

02. 1 english,80

03. 2 math,89

04. 3 chinese,95

Load data

01. LOAD DATA LOCAL INPATH '/home/hadoop/test.txt' OVERWRITE INTO TABLE test;

Query the table

01. hive> select \* from test;

02. OK

03. 1 {"course":"english","score":80}

04. 2 {"course":"math","score":89}

05. 3 {"course":"chinese","score":95}

06. Time taken: 0.275 seconds

07. hive> select course from test;

08. {"course":"english","score":80}

09. {"course":"math","score":89}

10. {"course":"chinese","score":95}

11. Time taken: 44.968 seconds

12. select t.course.course from test t;

13. english

14. math

15. chinese

16. Time taken: 15.827 seconds

17.hive> select t.course.score from test t;

18. 3080

19. 3189

20. 3295

21. Time taken: 13.235 seconds

##### 4. Composite

1. LOAD DATA LOCAL INPATH '/home/hadoop/test**.txt**' OVERWRITE INTO TABLE test;

2. create table test1(id int,a MAP<STRING,ARRAY<STRING>>)

3. row format delimited fields terminated by '\t'

4. collection items terminated by ','

5. MAP KEYS TERMINATED BY ':';

6. 1 english:80,90,70

7. 2 math:89,78,86

8. 3 chinese:99,100,82

9. LOAD DATA LOCAL INPATH '/home/hadoop/test1**.txt**' OVERWRITE INTO TABLE test1;

**Union::**

CREATE TABLE combined AS

SELECT unioned.id, unioned.var1, unioned.var2

FROM (

SELECT a.id, a.var1, a.var2

FROM table\_A a

UNION ALL

SELECT b.id, b.var1, b.var2

from table\_B b

) unioned;

INSERT OVERWRITE TABLE target\_table

  SELECT name, id, category FROM source\_table\_1

  UNION ALL

  SELECT name, id, "Category159" as category FROM source\_table\_2

SELECT key FROM src

UNION

SELECT key FROM src1

ORDER BY key LIMIT 10

SELECT key FROM (SELECT key FROM src ORDER BY key LIMIT 10)subq1

UNION

SELECT key FROM (SELECT key FROM src1 ORDER BY key LIMIT 10)subq2

**Ref::\_ http://www.cloudera.com/content/cloudera/en/documentation/cdh5/v5-1-x/Impala/Installing-and-Using-Impala/ciiu\_union.html**

The UNION keyword by itself is the same as UNION DISTINCT. Because eliminating duplicates can be a memory-intensive process for a large result set, prefer UNION ALL where practical. (That is, when you know the different queries in the union will not produce any duplicates, or where the duplicate values are acceptable.)

# Map Side Join

If all but one of the tables being joined are small, the join can be performed as a map only job. The query does not need a reducer. For every mapper a,b is read completely. A restriction is that a **FULL/RIGHT OUTER JOIN b** cannot be performed.

SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value

FROM a join b on a.key = b.key

# Bucketed Map Join

If the tables being joined are bucketized, and the buckets are a multiple of each other, the buckets can be joined with each other. If table A has 8 buckets are table B has 4 buckets, the following join:

SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value

FROM a join b on a.key = b.key

can be done on the mapper only. Instead of fetching B completely for each mapper of A, only the required buckets are fetched. For the query above, the mapper processing bucket 1 for A will only fetch bucket 1 of B. It is not the default behavior, and is governed by the following parameter

*set hive.optimize.bucketmapjoin = true*

If the tables being joined are sorted and bucketized, and the number of buckets are same, a sort-merge join can be performed. The corresponding buckets are joined with each other at the mapper. If both A and B have 4 buckets

SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value

FROM A a join B b on a.key = b.key

can be done on the mapper only. The mapper for the bucket for A will traverse the corresponding bucket for B. This is not the default behavior, and the following parameters need to be set:

set hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;

set hive.optimize.bucketmapjoin = true;

set hive.optimize.bucketmapjoin.sortedmerge = true;

# [How to set variables in HIVE scripts](http://stackoverflow.com/questions/12464636/how-to-set-variables-in-hive-scripts)

You need to use the special **hiveconf** for variable substitution. e.g.

hive> set CURRENT\_DATE='2012-09-16';

hive> select \* from foo where day >= '${hiveconf:CURRENT\_DATE}'

similarly, you could pass on command line:

% hive -hiveconf CURRENT\_DATE='2012-09-16' -f test.hql

Note that there are **env** and **system** variables as well, so you can reference ${env:USER} for example.

To see all the available variables, from the command line, run

% hive -e 'set;'

or from the hive prompt, run

hive> set;

**Update:** I've started to use *hivevar* variables as well, putting them into hql snippets I can include from hive CLI using the source command (or pass as -i option from command line). The benefit here is that the variable can then be used with or without the hivevar prefix, and allow something akin to global vs local use.

So, assume have some *setup.hql* which sets a tablename variable:

set hivevar:tablename=mytable;

then, I can bring into hive:

hive> source /path/to/setup.hql;

and use in query:

hive> select \* from ${tablename}

or

hive> select \* from ${hivevar:tablename}

I could also set a "local" tablename, which would affect the use of ${tablename}, but not ${hivevar:tablename}

hive> set tablename=newtable;

hive> select \* from ${tablename} -- uses 'newtable'

vs

hive> select \* from ${hivevar:tablename} -- still uses the original 'mytable'

**Q1. What is Hive Metastore?**  
**Answer:** Hive metastore is a database that stores metadata about your Hive tables (eg. table name, column names and types, table location, storage handler being used, number of buckets in the table, sorting columns if any, partition columns if any, etc.). When you create a table, this metastore gets updated with the information related to the new table which gets queried when you issue queries on that table.

**Q2. Wherever (Different Directory) I run hive query, it creates new metastore\_db, please explain the reason for it?**  
**Answer:** Whenever you run the hive in embedded mode, it creates the local metastore. And before creating the metastore it looks whether metastore already exist or not. This property is defined in configuration file hive-site.xml. Property is "javax.jdo.option.ConnectionURL" with default value "jdbc:derby:;databaseName=metastore\_db;create=true". So to change the behavior change the location to absolute path, so metastore will be used from that location.

**Q3. Is it possible to use same metastore by multiple users, in case of embedded hive?**  
**Answer:** No, it is not possible to use metastore in sharing mode. It is recommended to use standalone "real" database like MySQL or PostGresSQL.

**Q4. Is multiline comment supported in Hive Script ?**  
**Answer:** No.

**Q5. If you run hive as a server, what are the available mechanism for connecting it from application?**  
**Answer:** There are following ways by which you can connect with the Hive Server: 1. Thrift Client: Using thrift you can call hive commands from a various programming languages e.g. C++, Java, PHP, Python and Ruby. 2. JDBC Driver : It supports the Type 4 (pure Java) JDBC Driver 3. ODBC Driver: It supports ODBC protocol.

**Q6. What is SerDe in Apache Hive ?**  
**Answer:** A SerDe is a short name for a Serializer Deserializer. Hive uses SerDe (and FileFormat) to read and write data from tables. An important concept behind Hive is that it DOES NOT own the Hadoop File System (HDFS) format that data is stored in. Users are able to write files to HDFS with whatever tools/mechanism takes their fancy("CREATE EXTERNAL TABLE" or "LOAD DATA INPATH," ) and use Hive to correctly "parse" that file format in a way that can be used by Hive. A SerDe is a powerful (and customizable) mechanism that Hive uses to "parse" data stored in HDFS to be used by Hive.

**Q7. Which classes are used by the Hive to Read and Write HDFS Files**  
**Answer:** Following classes are used by Hive to read and write HDFS files

•TextInputFormat/HiveIgnoreKeyTextOutputFormat: These 2 classes read/write data in plain text file format.

•SequenceFileInputFormat/SequenceFileOutputFormat: These 2 classes read/write data in hadoop SequenceFile format.

**Q8. Give examples of the SerDe classes whihc hive uses to Serializa and Deserilize data ?**  
**Answer:** Hive currently use these SerDe classes to serialize and deserialize data:

• MetadataTypedColumnsetSerDe: This SerDe is used to read/write delimited records like CSV, tab-separated control-A separated records (quote is not supported yet.)

• ThriftSerDe: This SerDe is used to read/write thrift serialized objects. The class file for the Thrift object must be loaded first.

• DynamicSerDe: This SerDe also read/write thrift serialized objects, but it understands thrift DDL so the schema of the object can be provided at runtime. Also it supports a lot of different protocols, including TBinaryProtocol, TJSONProtocol, TCTLSeparatedProtocol (which writes data in delimited records).

**Q9. How do you write your own custom SerDe ?**  
**Answer:**

•In most cases, users want to write a Deserializer instead of a SerDe, because users just want to read their own data format instead of writing to it.

•For example, the RegexDeserializer will deserialize the data using the configuration parameter 'regex', and possibly a list of column names

•If your SerDe supports DDL (basically, SerDe with parameterized columns and column types), you probably want to implement a Protocol based on DynamicSerDe, instead of writing a SerDe from scratch. The reason is that the framework passes DDL to SerDe through "thrift DDL" format, and it's non-trivial to write a "thrift DDL" parser.

**Q10. What is ObjectInspector functionality ?**  
**Answer:** Hive uses ObjectInspector to analyze the internal structure of the row object and also the structure of the individual columns. ObjectInspector provides a uniform way to access complex objects that can be stored in multiple formats in the memory, including:

•Instance of a Java class (Thrift or native Java)

•A standard Java object (we use java.util.List to represent Struct and Array, and use java.util.Map to represent Map)

•A lazily-initialized object (For example, a Struct of string fields stored in a single Java string object with starting offset for each field) A complex object can be represented by a pair of ObjectInspector and Java Object. The ObjectInspector not only tells us the structure of the Object, but also gives us ways to access the internal fields inside the Object.

**Q11. What is the functionality of Query Processor in Apached Hive ?**  
**Answer:** This component implements the processing framework for converting SQL to a graph of map/reduce jobs and the execution time framework to run those jobs in the order of dependencies.

What is unconditionaltask = true in hive?