**Hive**

The [Apache Hive™](http://hive.apache.org/) data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage and queried using SQL syntax

1. Hive is a data warehousing infrastructure based on [Apache Hadoop](http://hadoop.apache.org/).
2. Hive is not designed for online transaction processing.  It is best used for traditional data warehousing tasks
3. Hive Stores its Metadata in a Relational Database .The purpose of storing this metadata information into relational databases because if we store this information at file level then performance of hive will be down, logically file reading will take more time as compared to relational storage that is why the purpose came to store this information in separate database.

**Hive Architecture**

**UI**

User interface – CLI

**Metadata -** data about the data

If we store the metadata in HDFS it will be available in Flat files, reading the flat files and parse the flat file is huge and complex task rather than reading the same information from the RDBMS table. So we are storing the metadata in the RDBMS.

**Driver**

We need drivers to talk with the database

**Complier**

It will check for syntax error, it will generate the execution plan

**Execution Engine**

The component which executes the execution plan created by the complier.

**Metastore**

The Component that stores all the structured information of the various tables and partition in the warehouse including column and column type information serialized, deserializer necessary to read and write the data.

* Metastore will store only Meta data information. Real Data will not be stored and it will be available in HDFS.
* By default Meta store will be running in the same process as the Hive service. Derby is the default Meta store. We are going for External Database like Mysql.
* Data Abstraction and Data Discovery( used to discover and explore the specific data)

**Different Modes of Meta Store**

**1. Embedded Metastore:**



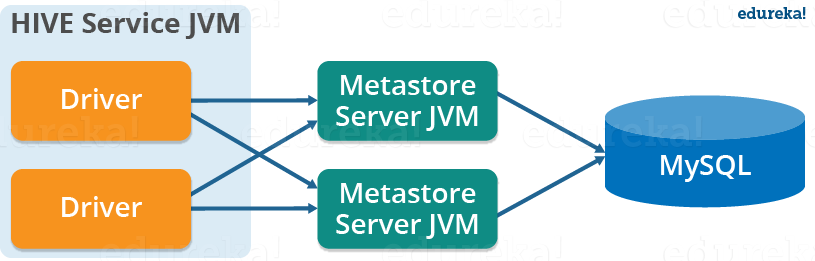
Both the metastore service and the Hive service runs in the same JVM by default using an embedded Derby Database instance where metadata is stored in the local disk. This is called embedded metastore configuration. In this case, only one user can connect to metastore database at a time. If you start a second instance of Hive driver, you will get an error. This is good for unit testing, but not for the practical solutions.

**2. Local Metastore:**

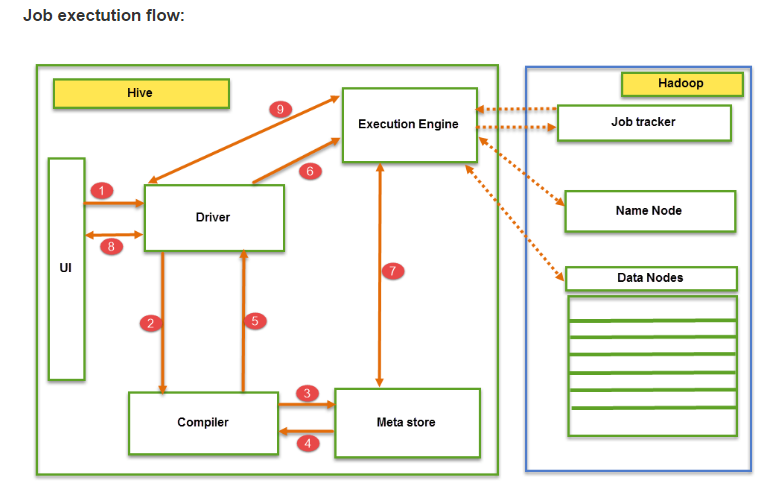


This configuration allows us to have multiple Hive sessions i.e. Multiple users can use the metastore database at the same time. This is achieved by using any JDBC compliant database like My SQL which runs in a separate JVM or a different machine than that of the Hive service and metastore service which are running in the same JVM as shown above. In general, the most popular choice is to implement a My SQL server as the metastore database.

**3. Remote Metastore:**



In the remote metastore configuration, the metastore service runs on its own separate JVM and not in the Hive service JVM. Other processes communicate with the metastore server using Thrift Network APIs. You can have one or more metastore servers in this case to provide more availability. The main advantage of using remote metastore is you do not need to share JDBC login credential with each Hive user to access the metastore database.

****

The data flow in Hive behaves in the following pattern

* Executing Query from the UI( User Interface)
* The driver is interacting with Compiler for getting the plan. (Here plan refers to query execution) process and its related metadata information gathering
* The compiler creates the plan for a job to be executed. Compiler communicating with Meta store for getting metadata request
* Meta store sends metadata information back to compiler
* Compiler communicating with Driver with the proposed plan to execute the query
* Driver sending execution plans to Execution engine
* Execution Engine (EE) acts as a bridge between Hive and Hadoop to process the query. For DFS Operations.
* EE should first contact Name Node and then to Data nodes to get the values stored in tables.
* EE is going to fetch desired records from Data Nodes. The actual data of tables resides in data node only. While from Name Node it only fetches the metadata information for the query.
* It collects actual data from data nodes related to mentioned query
* Execution Engine (EE) communicates bi-directionally with Meta store present in Hive to perform DDL (Data Definition Language) operations. Here DDL operations like CREATE, DROP and ALTERING tables and databases are done. Meta store will store information about database name, table names and column names only. It will fetch data related to query mentioned.
* Execution Engine (EE) in turn communicates with Hadoop daemons such as Name node, Data nodes, and job tracker to execute the query on top of Hadoop file system
* Fetching results from driver
* Sending results to Execution engine. Once the results fetched from data nodes to the EE, it will send results back to driver and to UI (front end)

**Why apache hive called schema on read not schema on write?**

* In relation database systems, let’s say we perform any insert or update operations in this case database has full control over the storage and database can enforce the schema as data is written, this statement in relation databases called schema on write.
* However, hive has no control over the storage (Ex. HDFS), there are many possible ways in Hadoop to damage the hive warehouse data and that’s why hive can only apply query on read which called schema on read

**Hive Data Types**

Hive supports both primitive and Complex data types

|  |  |
| --- | --- |
| **Primitive Data type** |  |
| TINYINT |  |
| SMALLINT |  |
| INT |  |
| BIGINT |  |
| FLOAT |  |
| DOUBLE |  |
| STRING |  |
| VARCHAR |  |
| CHAR |  |
| TIMESTAMP |  |
| DATE |  |
| BINARY |  |

|  |  |  |
| --- | --- | --- |
| **Complex Data Types** | **Declaration** | **Accessing Elements** |
| STRUCTS | c {a INT; b INT} | c.a |
| MAPS | M('group' -> gid ) | M['group'] |
| ARRAYS | ['a', 'b', 'c'] | A[1] |

**Hive Table Creation**

*Hive Table =* ***Data Stored in HDFS*** *+* ***Metadata (Schema of the table) stored in RDBMS***

* *Hive Metastore is not stored in HDFS*
* *Hive Metastore store the metadata information in any of the Traditional RDBMS System like MYSQL Oracle*
* *Hive by default uses derby database to store the metadata. But it allows only one user connection at a time. So in Real Time environment we go for Databases like MYSQL ORACLE*
* *Default Location is /user/hive/warehouse/*
* *In HDFS Database Name and Table is a Directory.*

**Hive Table TYPES**

**Managed Tables**

**External Tables**

**Temporary Tables**

**Skewed Tables**

**Managed Tables Default table type in Hive**

* By default Hive creates managed tables, where files, metadata and statistics are managed by internal Hive processes
* A Managed Table is stored under the data warehouse Directory, which we specified in the hime-site.xml
  + hive.metastore.warehouse.dir
* If a Managed table is Dropped , the data and metadata information associated with the table are also deleted .
* Use managed tables when Hive should manage the lifecycle of the table, or when generating temporary tables.

**External Table**

* These tables are not managed or owned by Hive.
* Table data will not be copied into hive warehouse directory but maintained at external location.
* If External Tables are dropped only the Schema from the metastore will be deleted. Data from the external data sources will be available.
* It provides the convenience to share the table data with other application like pig Hbase etc.
* Location” Clause is mandatory to create an external table otherwise table will be managed by Hive only even if we create it with “External” keyword.

**Temporary Tables**

* These table will be available till the end of the current session.
* Useful in case of creating intermediate tables to copy data records from one table to another but can be deleted after our copy operation.
* Table’s Data will be stored in the user’s scratch directory configured by ***hive.exec.scratchdir***, and deleted at the end of the session.
* Temporary tables don’t support Partitioning & Indexing.
* If a temporary table is created with same name of a permanent table which already exists in the database, then, original table can’t be accessed in that session until we drop the temporary table.

**Skewed Tables**

* To improve performance of tables with one or more columns having skewed (repeated) values.
* Hive will split the skewed (very often) values records into separate files, and the same will be considered into account at the time of querying this table, so that it can skip (or include) the whole file based on the input criteria.
* These are not separate table types, but can be managed or external. If we partition a table by country and there 200 countries in input file, but 80% records are from only US, UK, IN, JPN, then it is better to go by Skewing by country for four values. In skewing, it will create only 5 separate files/directories (4 for US, UK, IN, JPN and 1 for remaining all) where as partitioning will create 200 directories making the structure very complex.

***CTAS – Create Table As Select***

*Lets Create a Table like Below*

*CREATE TABLE new\_key\_value\_store*

*ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"*

*STORED AS RCFile*

*AS*

*SELECT (key % 1024) new\_key, concat(key, value) key\_value\_pair*

*FROM key\_value\_store*

*SORT BY new\_key, key\_value\_pair;*

* The above CTAS statement creates the target table new\_key\_value\_store with the schema (new\_key DOUBLE, key\_value\_pair STRING) derived from the results of the SELECT statement.
* If the SELECT statement does not specify column aliases, the column names will be automatically assigned to \_col0, \_col1, and \_col2 etc.
* In addition, the new target table is created using a specific SerDe and a storage format independent of the source tables in the SELECT statement.

***Create Table Like***

*The LIKE form of CREATE TABLE allows you to copy an existing table definition exactly (without copying its data). In contrast to CTAS, the statement below creates a new empty\_key\_value\_store table whose definition exactly matches the existing key\_value\_store in all particulars other than table name. The new table contains no rows.*

*CREATE TABLE empty\_key\_value\_store LIKE key\_value\_store;*

**Hive Table Creation**

**Ex1**

**http://hadooptutorial.info/hive-table-creation-commands/**

**Create a Simple Hive table for the below specified Records , Create the table using .hql File**

*(C:\Users\1532894\Documents\Spark\Hive\Data\User\_Records.txt)*

Rebbecca,Didio,Brandt Jonathan F Esq,171 E 24th St,AU,Leith,TA,7315,03-8174-9123 0458-665-290 0427-885-282,email:rebbecca.didio@didio.com.au,http://www.brandtjonathanfesq.com.au

Stevie,Hallo,Landrum Temporary Services,22222 Acoma St,US,Proston,QL,4613,07-9997-3366 0497-622-620,email:stevie.hallo@hotmail.com,http://www.landrumtemporaryservices.com.au

Mariko,Stayer,Inabinet Macre Esq,534 Schoenborn St #51,UK,Hamel,WA,6215,08-5558-9019 0427-885-282,email:mariko\_stayer@hotmail.com,http://www.inabinetmacreesq.com.au

Gerardo,Woodka,Morris Downing & Sherred,69206 Jackson Ave,AU,Talmalmo,NS,2640,02-6044-4682 0443-795-912,email:gerardo\_woodka@hotmail.com,http://www.morrisdowningsherred.com.au

Mayra,Bena,Buelt David L Esq,808 Glen Cove Ave,AU,Lane Cove,NS,1595,02-1455-6085 0453-666-885 0427-885-282,email:mayra.bena@gmail.com,http://www.bueltdavidlesq.com.au

first\_name,last\_name,company\_name,address,country,city,state,post,phone\_nos,email,web

**EX2**

**Create Hive External Table for the Above Exercise**

*Move the File to the Hdfs directory*

**EX03**

**External Table with Create Like Command**

**Ex04**

**External Table with ORC File Fomat & Snappy Compressed**

**EX05**

**Create Skewed Tables Stored in SequenceFile**

**EX06**

* CREATE A TEMPORARY FILE which is having fname and mark
* Insert 5 sample records
* Describe the table
* Display all the records from the table

**EX07**

**Create a Sample External Hive Table for the below data and query It .**

12345|John,Smith|123 Main St,New York, NY,00000|45,40,17,13|weekly\_update:true,special\_clearance:true,birthday\_greeting:false

**Ex08**

**Practice the above CTAS TABLE**

**Ex09**

**Practice Run Hive Script File Passing Parameter**

**EX10**

**Create a Simple partition Table for the new\_user\_records.txt file and store it as Sequence File**

**EX11**

**lets create the dynamic partitioned table for the user records**

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.

**EX12**

1. **SHOW THE PARTIONS**
2. **DESCRIBE THE PARTITIONS**
3. **ALTER THE PARTITIONS** 
   1. **Adding Partitions**
   2. **Changing Partitions**
   3. **Drop Partitions**

**EX13**

**LOAD A SIMPLE CSV FILE INTO HIVE AND CREATE IT AS A TABLE**

**EX14**

[**http://hadooptutorial.info/hive-use-case-example-for-json-data/**](http://hadooptutorial.info/hive-use-case-example-for-json-data/)

**EX15**

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

**EX16**

**Truncate any above table**

**Ex17**

**Alter the Table name**

**Alter the Table Properties**

**Alter the Column Name and Change the Data Type**

**Ex18**

**Make a Hql by using Shell Script**

**Ex19:**

**Create Tables for Orders order\_items products categories**

**Ex20:**

**Import the Orders Tables and create View Based on the Orderstatus**

**For closed items we should have a view**

**NOTES**

* *Table names and column names are case insensitive*
* *Comments are single quoted string literals*
* *CREATE-TABLE-AS-SELECT cannot create external tables, partitioned tables and bucketed tables.*
* *Default Storage format is TEXTFILE.*
* ***Default field delimiter is not tab (‘\t’) but it is Ctrl+A (‘\001’)***
* *Table can be made freeze so that no other changes are allowed on this with table property* ***TBLPROPERTIES (“immutable”=”true”).***
* *Table and column comments are string literals (single-quoted).*
* *The TBLPROPERTIES clause allows you to tag the table definition with your own metadata key/value pairs.*
  + *TBLPROPERTIES ("comment"="table\_comment")*
  + *TBLPROPERTIES ("immutable"="true") or ("immutable"="false")*
  + *TBLPROPERTIES ("orc.compress"="ZLIB") or ("orc.compress"="SNAPPY") or ("orc.compress"="NONE") and other ORC properties*
  + TBLPROPERTIES ("transactional"="true") or ("transactional"="false")
  + *TBLPROPERTIES ("NO\_AUTO\_COMPACTION"="true") or ("NO\_AUTO\_COMPACTION"="false")*
  + *TBLPROPERTIES ("auto.purge"="true") or ("auto.purge"="false")* If the PURGE option is not specified, the data is moved to a trash folder for a defined duration
* *If the structure or partitioning of an external table is changed, an MSCK REPAIR TABLE table\_name statement can be used to refresh metadata information.*
* *CTAS – Create Table As Select has dome restrictions*
  + *The target table cannot be a partitioned table.*
  + *The target table cannot be an external table.*
  + *The target table cannot be a list bucketing table.*

**A Simple Hive SQL Script**

Create a Hql File

* sudo vim sample.sql
* create table product ( productid: int, productname: string, price: float, category: string) rows format delimited fields terminated by ',' ;
* load data local inpath '/home/cloudera/input.txt' into table product ;
* select \* from product ;

run the Hql File

* hive -f sample.sql

**Passing parameters to Hive scripts**

Create a Hive Sql File which does the Following

Select \* from user\_records where country = <parameterize >

Hive provides you with the ability to create parameterized scripts – greatly increasing the re-usability of the scripts

**Select \* from user\_records where country = ‘${hiveconf:USER\_COUNTRY}’**

While Hive Execution use it like

**Hive –f hivescript.hql -hiveconf USER\_COUNTRY=’AU’**

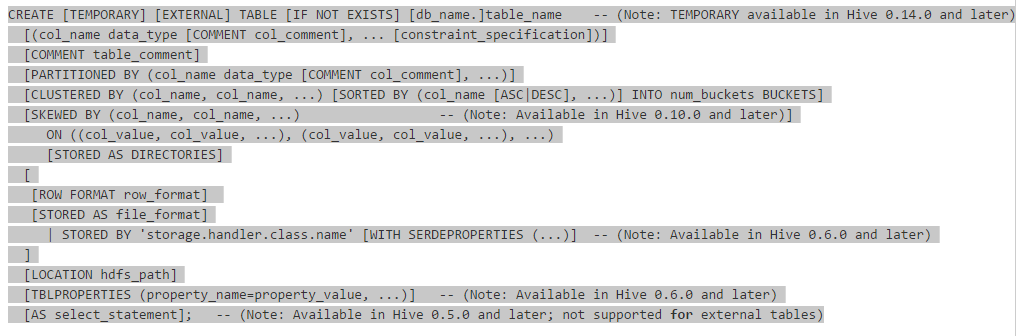
*If you define the parameter in the script but fail to specify a value at run-time, you won’t get any error like you would with Pig.*

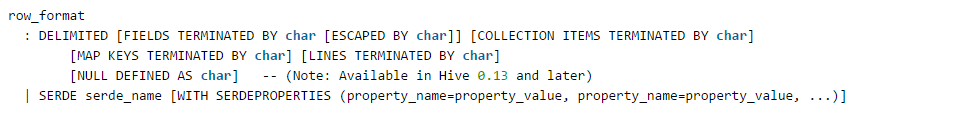
*Instead, the restriction effectively becomes “where teamid = ””. If you have blanks then you might get a result back; if not, you’ll go through all the necessary mechanics of executing the script sans the results.*

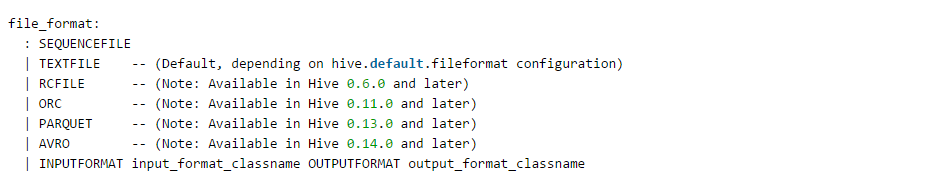
We can run the above command from the Unix shell itself

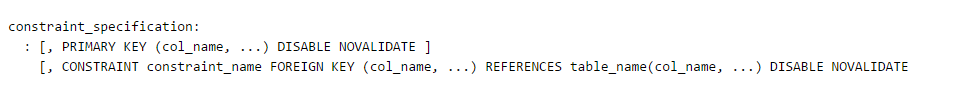
**hive -hiveconf USER\_COUNTRY=’AU’ –e ‘Select \* from user\_records where country =’${hiveconf:USER\_COUNTRY}’**

**HIVE TABLE CREATION SYNTAX**









**Data Storage Formats in Hive**

**Text File Format** – Default file format in hive if we don’t specify anything it will be used

CREATE TABLE Employee(

ID BIGINT,

NAME STRING,

AGE INT,

SALARY BIGINT

)

COMMENT 'This is Employee table stored as textfile'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

#To overwrite the data in the table use -

LOAD DATA INPATH '/home/hadoop/data/employee.csv' OVERWRITE INTO TABLE Employee;

#To append the data in the table use -

LOAD DATA INPATH '/home/hadoop/data/employee.csv' INTO TABLE Employee;

**ORC File Format**

ORC (Optimized Row Columnar) file format provides a highly efficient way to store Hive data. Using ORC format improves performance when reading, writing, and processing data in Hive. We can specify compression to further compress data files. It could result in a small performance loss while writing, but there will be huge performance gain in reading. Compression available are SNAPPY, ZLIB, NONE.

CREATE TABLE Employee(

ID BIGINT,

NAME STRING,

AGE INT,

SALARY BIGINT

)

COMMENT 'This is Employee table in ORC file format'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS ORC tblproperties ("orc.compress"="ZLIB");

#Overwrite data from result of a select query into the table

INSERT OVERWRITE TABLE Employee SELECT id, name, age, salary from Employee\_text;

#Append data from result of a select query into the table

INSERT INTO TABLE Employee SELECT id, name, age, salary from Employee\_text;

**SEQUENCE FILE FORMAT**

CREATE TABLE Employee(

ID BIGINT,

NAME STRING,

AGE INT,

SALARY BIGINT

)

COMMENT 'This is Employee table stored as sequencefile'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS SEQUENCEFILE;

#Overwrite data from result of a select query into the table

INSERT OVERWRITE TABLE Employee SELECT id, name, age, salary from Employee\_text;

#Append data from result of a select query into the table

INSERT INTO TABLE Employee SELECT id, name, age, salary from Employee\_text

RC FILE

PARQUET

AVRO

**Row Formats & SerDe**

**If we have multiple Delimiter as below**

[NYSE@#B7J@#2009-12-16@#8.80@#8.84@#8.77@#8.83@#69700@#8.83](mailto:NYSE@#8.84@)

CREATE EXTERNAL TABLE IF NOT EXISTS stocks\_multi (

exch STRING,

symbol STRING,

ymd STRING,

price\_open FLOAT,

price\_high FLOAT,

price\_low FLOAT,

price\_close FLOAT,

volume INT,

price\_adj\_close FLOAT)

ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.MultiDelimitSerDe'

WITH SERDEPROPERTIES ("field.delim"="@#")

LOCATION '/user/hirw/stocks-multi';

This SerDe works for most CSV data, but does not handle embedded newlines. To use the SerDe, specify the fully qualified class name org.apache.hadoop.hive.serde2.OpenCSVSerde.

Documentation is based on original documentation at <https://github.com/ogrodnek/csv-serde>.

Create table, specify CSV properties

|  |
| --- |
| CREATE TABLE my\_table(a string, b string, ...)  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'  WITH SERDEPROPERTIES (     "separatorChar" = "\t",     "quoteChar"     = "'",     "escapeChar"    = "\\"  )  STORED AS TEXTFILE; |

Default separator, quote, and escape characters if unspecified

|  |
| --- |
| DEFAULT\_ESCAPE\_CHARACTER \  DEFAULT\_QUOTE\_CHARACTER  "  DEFAULT\_SEPARATOR        , |

**Read Xml Files**

**XML data**

hadoop fs -cat /sourcedata/bookdata.xml

<CATALOG>

<BOOK>

<TITLE>Hadoop Defnitive Guide</TITLE>

<AUTHOR>Tom White</AUTHOR>

<COUNTRY>US</COUNTRY>

<COMPANY>CLOUDERA</COMPANY>

<PRICE>24.90</PRICE>

<YEAR>2012</YEAR>

</BOOK>

<BOOK>

<TITLE>Programming Pig</TITLE>

<AUTHOR>Alan Gates</AUTHOR>

<COUNTRY>USA</COUNTRY>

<COMPANY>Horton Works</COMPANY>

<PRICE>30.90</PRICE>

<YEAR>2013</YEAR>

</BOOK>

</CATALOG>

**Creating Hive Table**

DROP TABLE IF EXISTS BOOKDATA;

CREATE EXTERNAL TABLE BOOKDATA (TITLE STRING, PRICE FLOAT)

ROW FORMAT SERDE 'com.ibm.spss.hive.serde2.xml.XmlSerDe'

WITH SERDEPROPERTIES (

"column.xpath.TITLE"="/BOOK/TITLE/text()",

"column.xpath.PRICE"="/BOOK/PRICE/text()")

STORED AS INPUTFORMAT 'com.ibm.spss.hive.serde2.xml.XmlInputFormat'

OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.IgnoreKeyTextOutputFormat'

LOCATION '/user/it1/hive/xml'

TBLPROPERTIES ("xmlinput.start"="<BOOK","xmlinput.end"= "</BOOK>");

**Reading the Json Data**

**{**

"customerId":"0277ZGAX80PG6ZSJ04J5",

"age":23,

"services":[ {

"trips":[ {

"tripId":"A12-5678344-4097746",

"fare":24.0,

"distanceTravelled":3.2,

"referrals":{

"email\_campaign":{

"campaignA":{

"referralIds":["0ZK7V4HM5ZZNKJ0PRRR5"]

}

}

}

} ]

} ]

}

Now we have to create a table schema using the

**Org.openx.data.jsonserde.JsonSerDe**

create table cust\_trips (

customerId string,

age int,

services array<struct<

trips:array<struct<

tripId:string,

fare:double,

distanceTravelled:double,

referrals:struct<

email\_campaign:map<string, struct<referralIds:array<string>>>

>

>>

>>

) row format serde 'org.openx.data.jsonserde.JsonSerDe'

location '<location-of-files>';

Another Function

{

"DocId": "ABC",

"User": {

"Id": 1234,

"Username": "sam1234",

"Name": "Sam",

"ShippingAddress": {

"Address1": "123 Main St.",

"Address2": null,

"City": "Durham",

"State": "NC"

},

"Orders": [

{

"ItemId": 6789,

"OrderDate": "11/11/2012"

},

{

"ItemId": 4352,

"OrderDate": "12/12/2012"

}

]

}

}

CREATE TABLE complex\_json (

DocId string,

User struct<Id:int,

Username:string,

Name: string,

ShippingAddress:struct<Address1:string,

Address2:string,

City:string,

State:string>,

Orders:array<struct<ItemId:int,

OrderDate:string>>>

)

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe';

Load the data:

LOAD DATA LOCAL INPATH '/tmp/complex.json'

OVERWRITE INTO TABLE complex\_json;

SELECT DocId, User.Id, User.ShippingAddress.City as city,

User.Orders[0].ItemId as order0id,

User.Orders[1].ItemId as order1id

FROM complex\_json;

{"DocId":"ABC","User1":{"Id":1234,"Username":"sam1234","Name":"Sam","ShippingAddress":{"Address1":"123 Main St.","Address2":null,"City":"Durham","State":"NC"},"Orders":[{"ItemId":6789,"OrderDate":"11/11/2012"},{"ItemId":4352,"OrderDate":"12/12/2012"}]}}

Add the jar file in hive cli. You may have different version, I have 1611 jar file  
  
**add jar /opt/mapr/hive/hive-1.2/hcatalog/share/hcatalog/hive-hcatalog-core-1.2.0-mapr-1611.jar;**

This Jar contains all the implementations of handling complex json files.

3.  
Create the Table Definition as below.   
  
**CREATE EXTERNAL TABLE sample\_json3(**  
**DocId string,**  
**user1 struct<Id: int, username: string, name:string,shippingaddress:struct<address1:string,address2:string,city:string,state:string>, orders:array<struct<ItemId:int,orderdate:string>>>**  
**)**  
**ROW FORMAT SERDE**  
**'org.apache.hive.hcatalog.data.JsonSerDe'**  
**LOCATION**  
**'maprfs:/tmp/jsondir'**

. Now you can query your json data as below.  
  
Output of the query  
  
**hive> select DocId,user1.username,user1.name,user1.orders from sample\_json3;**  
**OK**  
**ABC     sam1234 Sam     [{"itemid":6789,"orderdate":"11/11/2012"},{"itemid":4352,"orderdate":"12/12/2012"}]**  
**Time taken: 0.098 seconds, Fetched: 1 row(s)**  
hive>

**Hive Functions**

## STRING FUNCTIONS

String Functions

The following are built-in String functions are supported in hive:

| **Return Type** | **Name(Signature)** | **Example** |
| --- | --- | --- |
| int | ascii(string str) | Returns the numeric value of the first character of str |
| string | concat(string|binary A, string|binary B…) | Returns the string or bytes resulting from concatenating the strings or bytes passed in as parameters in order. e.g. concat(‘foo’, ‘bar’) results in ‘foobar’. Note that this function can take any number of input strings. |
| array<struct  <string,double>> | context\_ngrams  (array<array>, array, int K, int pf) | Returns the top-k contextual N-grams from a set of tokenized sentences, given a string of “context”. See StatisticsAndDataMining for more information. |
| string | concat\_ws(string SEP, string A, string B…) | Like concat() above, but with custom separator SEP. |
| string | concat\_ws(string SEP, array) | Like concat\_ws() above, but taking an array of strings. (as of Hive 0.9.0) |
| int | find\_in\_set(string str, string strList) | Returns the first occurance of str in strList where strList is a comma-delimited string. Returns null if either argument is null. Returns 0 if the first argument contains any commas. e.g. find\_in\_set(‘ab’, ‘abc,b,ab,c,def’) returns 3 |
| string | format\_number  (number x, int d) | Formats the number X to a format like ‘#,###,###.##’, rounded to D decimal places, and returns the result as a string. If D is 0, the result has no decimal point or fractional part. (as of Hive 0.10.0) |
| string | get\_json\_object(string json\_string, string path) | Extract json object from a json string based on json path specified, and return json string of the extracted json object. It will return null if the input json string is invalid.NOTE: The json path can only have the characters [0-9a-z\_], i.e., no upper-case or special characters. Also, the keys \*cannot start with numbers.\* This is due to restrictions on Hive column names. |
| boolean | in\_file(string str, string filename) | Returns true if the string str appears as an entire line in filename. |
| int | instr(string str, string substr) | Returns the position of the first occurence of substr in str |
| int | length(string A) | Returns the length of the string |
| int | locate(string substr, string str[, int pos]) | Returns the position of the first occurrence of substr in str after position pos |
| string | lower(string A) lcase(string A) |  |
| string | lpad(string str, int len, string pad) | Returns str, left-padded with pad to a length of len |
| string | ltrim(string A) | Returns the string resulting from trimming spaces from the beginning(left hand side) of A e.g. ltrim(‘ foobar ‘) results in ‘foobar ‘ |
| array<struct  <string,double>> | ngrams(array<array >, int N, int K, int pf) | Returns the top-k N-grams from a set of tokenized sentences, such as those returned by the sentences() UDAF. See StatisticsAndDataMining for more information. |
| string | parse\_url(string urlString, string partToExtract [, string keyToExtract]) | Returns the specified part from the URL. Valid values for partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO. e.g. parse\_url(‘http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1’, ‘HOST’) returns ‘facebook.com’. Also a value of a particular key in QUERY can be extracted by providing the key as the third argument, e.g. parse\_url(‘http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1’, ‘QUERY’, ‘k1’) returns ‘v1’. |
| string | printf(String format, Obj… args) | Returns the input formatted according do printf-style format strings (as of Hive 0.9.0) |
| string | regexp\_extract(string subject, string pattern, int index) | Returns the string extracted using the pattern. e.g. regexp\_extract(‘foothebar’, ‘foo(.\*?)(bar)’, 2) returns ‘bar.’ Note that some care is necessary in using predefined character classes: using ‘\s’ as the second argument will match the letter s; ‘s’ is necessary to match whitespace, etc. The ‘index’ parameter is the Java regex Matcher group() method index. See docs/api/java/util/regex/Matcher.html for more information on the ‘index’ or Java regex group() method. |
| string | regexp\_replace(string INITIAL\_STRING, string PATTERN, string REPLACEMENT) | Returns the string resulting from replacing all substrings in INITIAL\_STRING that match the java regular expression syntax defined in PATTERN with instances of REPLACEMENT, e.g. regexp\_replace(“foobar”, “oo|ar”, “”) returns ‘fb.’ Note that some care is necessary in using predefined character classes: using ‘\s’ as the second argument will match the letter s; ‘s’ is necessary to match whitespace, etc. |
| string | repeat(string str, int n) | Repeat str n times |
| string | reverse(string A) | Returns the reversed string |
| string | rpad(string str, int len, string pad) | Returns str, right-padded with pad to a length of len |
| string | rtrim(string A) | Returns the string resulting from trimming spaces from the end(right hand side) of A e.g. rtrim(‘ foobar ‘) results in ‘ foobar’ |
| array<array> | sentences(string str, string lang, string locale) | Tokenizes a string of natural language text into words and sentences, where each sentence is broken at the appropriate sentence boundary and returned as an array of words. The ‘lang’ and ‘locale’ are optional arguments. e.g. sentences(‘Hello there! How are you?’) returns ( (“Hello”, “there”), (“How”, “are”, “you”) ) |
| string | space(int n) | Return a string of n spaces |
| array | split(string str, string pat) | Split str around pat (pat is a regular expression) |
| map<string,string> | str\_to\_map(text[, delimiter1, delimiter2]) | Splits text into key-value pairs using two delimiters. Delimiter1 separates text into K-V pairs, and Delimiter2 splits each K-V pair. Default delimiters are ‘,’ for delimiter1 and ‘=’ for delimiter2. |
| string | substr(string|binary A, int start) substring(string|binary A, int start) | Returns the substring or slice of the byte array of A starting from start position till the end of string A e.g. substr(‘foobar’, 4) results in ‘bar’ |
| string | substr(string|binary A, int start, int len) substring(string|binary A, int start, int len) | Returns the substring or slice of the byte array of A starting from start position with length len e.g. substr(‘foobar’, 4, 1) results in ‘b’ |
| string | translate(string input, string from, string to) | Translates the input string by replacing the characters present in the from string with the corresponding characters in the to string. This is similar to the translatefunction in PostgreSQL. If any of the parameters to this UDF are NULL, the result is NULL as well (available as of Hive 0.10.0) |
| string | trim(string A) | Returns the string resulting from trimming spaces from both ends of A e.g. trim(‘ foobar ‘) results in ‘foobar’ |
| string | upper(string A) ucase(string A) | Returns the string resulting from converting all characters of A to upper case e.g. upper(‘fOoBaR’) results in ‘FOOBAR’ |

**Creating Table based on Some Rules**

CREATE TABLE esakraw.orders\_all\_f (

order\_id INT,

order\_date DATE,

order\_status STRING,

order\_status\_code VARCHAR(8),

orderdate\_month INT,

orderdate\_day INT,

orderdate\_year INT,

juliandate INT,

currentday INT,

MONTHEND INT,

SinceOrderedDate BIGINT,

userdate DATE,

time TIMESTAMP,

evenorderid INT,

oddorderid INT,

orderid\_code VARCHAR(4)

);

INSERT OVERWRITE TABLE esakraw.orders\_all\_f

SELECT

order\_id,

TO\_DATE(from\_unixtime(UNIX\_TIMESTAMP(order\_date))),

order\_status,

CASE

WHEN order\_status='CANCELED' THEN 'CA'

WHEN order\_status='CLOSED' THEN 'CL'

WHEN order\_status='COMPLETE' THEN 'CP'

WHEN order\_status='ON\_HOLD' THEN 'OH'

WHEN order\_status='PAYMENT\_REVIEW' THEN 'PR'

WHEN order\_status='PENDING' THEN 'P'

WHEN order\_status='PENDING\_PAYMENT' THEN 'PP'

WHEN order\_status='PROCESSING' THEN 'PRO'

WHEN order\_status='SUSPECTED\_FRAUD' THEN 'SF'

END,

cast(month(order\_date) as INT),

cast(day(order\_date) as INT),

cast(year(order\_date) as INT),

from\_unixtime(unix\_timestamp(order\_date), 'DDD'),

cast(day(CURRENT\_DATE) as INT),

cast(day(last\_day(CURRENT\_DATE)) as INT)-cast(day(CURRENT\_DATE) as INT),

datediff(current\_date,last\_day(order\_date)),

CURRENT\_DATE,

CURRENT\_TIMESTAMP,

CASE WHEN order\_customer\_id % 2 == 0 THEN order\_customer\_id ELSE 0 END ,

CASE WHEN order\_customer\_id % 2 != 0 THEN order\_customer\_id ELSE 0 END ,

CASE WHEN order\_customer\_id % 2 == 0 THEN 'E' ELSE 'O' END

FROM esakraw.orders\_raw

limit 10;

CREATE TABLE PRODUCTS\_AGG(

PRODUCT\_CATEGORY\_ID INT ,

TOTAL INT ,

COUNT INT ,

Average DOUBLE,

Minimumm DOUBLE,

Maximum DOUBLE

);

INSERT OVERWRITE TABLE PRODUCTS\_AGG

SELECT

product\_category\_id,

SUM(new\_price),

COUNT(product\_category\_id),

AVG(new\_price),

MIN(new\_price),

MAX(new\_price)

FROM esakraw.products\_all\_f

group by product\_category\_id;

sqoop import --connect jdbc:mysql://nn01.itversity.com/retail\_db --username retail\_dba --password itversity --table products --hive-import --create-hive-table --hive-overwrite --hive-database esakraw --hive-table products\_raw --target-dir /user/esakkipillai/hadoop/hive/products -m 1

CREATE TABLE products\_all\_f(

product\_id INT,

product\_category\_id INT,

product\_name STRING,

Product\_NameLength INT,

product\_description STRING,

product\_price DOUBLE,

Price\_Range VARCHAR(12),

GST INT,

NEW\_PRICE DOUBLE,

product\_image STRING,

Product\_url\_s1 STRING,

Product\_url\_s2 STRING,

Product\_url STRING

);

INSERT OVERWRITE TABLE esakraw.products\_all\_f

SELECT

product\_id,

product\_category\_id,

product\_name,

length(product\_name),

CASE WHEN length(product\_description) > 0 then product\_description ELSE "NILL" END ,

product\_price,

CASE WHEN product\_price < 10 THEN 'CHEAPER'

WHEN product\_price BETWEEN 10 AND 100 THEN 'OKAY'

WHEN product\_price BETWEEN 100 AND 1000 THEN 'HIGHER'

WHEN product\_price > 1000 THEN 'LUXURY'

END,

CASE WHEN product\_price < 10 THEN 5

WHEN product\_price BETWEEN 10 AND 100 THEN 10

WHEN product\_price BETWEEN 100 AND 1000 THEN 20

WHEN product\_price > 1000 THEN 40

END,

CASE WHEN product\_price < 10 THEN ((0.05\* product\_price)+product\_price)

WHEN product\_price BETWEEN 10 AND 100 THEN ((0.1\* product\_price)+product\_price)

WHEN product\_price BETWEEN 100 AND 1000 THEN ((0.2\* product\_price)+product\_price)

WHEN product\_price > 1000 THEN ((0.4\* product\_price)+product\_price)

END,

product\_image ,

split(product\_image,'\\:')[0],

regexp\_replace(split(product\_image,'\\:')[1],"//",""),

concat(split(product\_image,'\\:')[0],split(product\_image,'\\:')[1] )

from products\_raw;

select

product\_image ,

split(product\_image,'\\:')[0],

regexp\_replace(split(product\_image,'\\:')[1],"//","")

from esakraw.products\_raw limit 10;

**Windowing Function**

**Lead**

**Lag**

**FirstValue**

**LastValue**

**RANK**

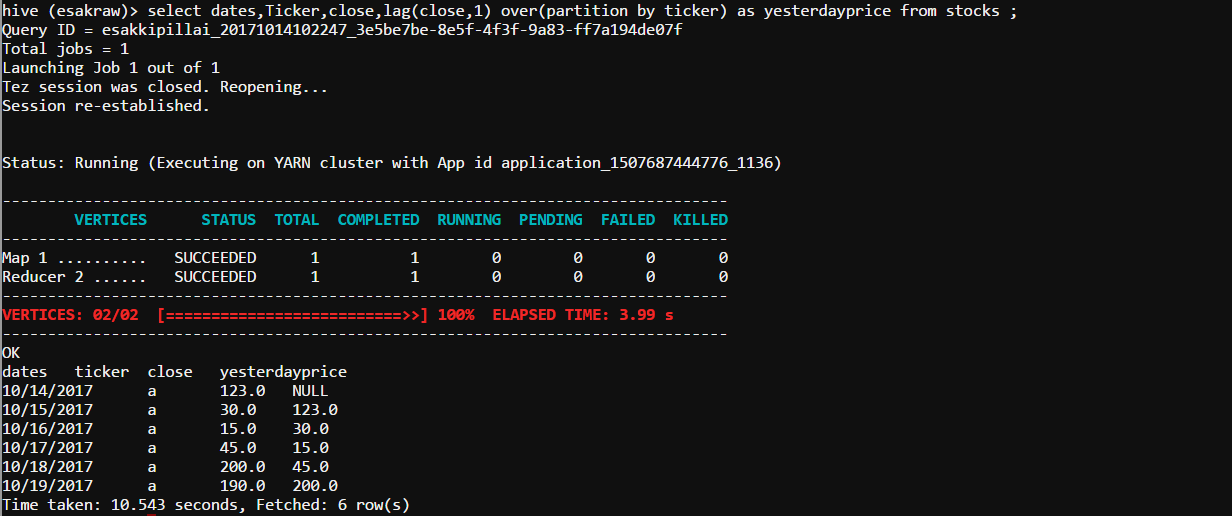
**ROW\_NUMBER**

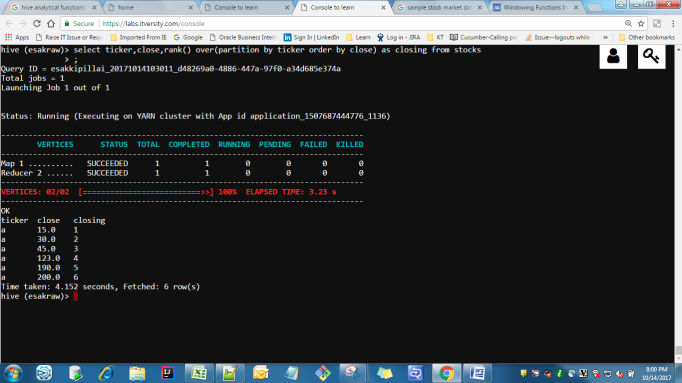
**DENSE\_RANK**

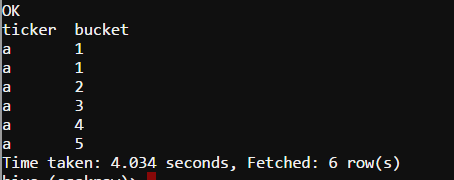
**NTILE**

**CUME\_DIST**

**PERCENT\_RANK**

****

****

****

create table esakraw.stocks (

dates String,

Ticker String,

Open Double,

High Double,

Low Double,

Close Double,

Volume\_for\_the\_day int

)

row format delimited

fields terminated by '\t'

stored as textfile;

LOAD DATA LOCAL INPATH '/home/esakkipillai/hadoop/hive/data/stock.csv' INTO TABLE stocks;

date Ticker Open High Low Close Volume\_for\_the\_day

10/14/2017 a 123 123 44 123 2222

10/15/2017 a 22 33 21 30 3333

10/16/2017 a 11 55 11 15 4444

10/17/2017 a 44 48 44 45 5555

10/18/2017 a 332 556 150 200 6666

10/19/2017 a 221 200 170 190 9999

LAG Value

select dates,Ticker,close,lag(close,1) over(partition by ticker) as yesterdayprice from stocks

Here using lag we can display the yesterday’s closing price of the ticker. Lag is to be used with over function, inside the over function you can use partition or order by classes.

we will find that whether the following day’s closing price is higher or lesser than today’s and that can be done as follows.

select dates,

Ticker,

close,

case lead(close,1) over(partition by ticker) - close)>0 when true THEN "higher" when false THEN "lower" as Changes

from stocks

select

Ticker,

first\_value(High) over (partition by Ticker) as highvalue

from stocks;

select dates,

Ticker,

last\_value(High) over (partition by Ticker) as Last\_highvalue

from stocks;

select ticker , count(\*) over(partition by ticker) as CNT

from stocks;

let us take if you want to get running total of the volume\_for\_the\_day for all the days for every ticker

then you can do this with the below query.

select

ticker ,dates, Volume\_for\_the\_day , sum(Volume\_for\_the\_day) over(partition by ticker,dates) as runningtotal

from stocks;

Finding the percentage of each row value

MIN

MAX

AVG

1

select ticker, avg(close) over(partition by ticker) as maximum from acadgild.stocks

RANK

select ticker,close,rank() over(partition by ticker order by close) as closing from stocks

DENSERANK

select ticker,close,dense\_rank() over(partition by ticker order by close) as closing from stocks

select ticker,close,row\_number() over(partition by ticker order by close) as closing from stocks

1

select ticker,ntile(5) over(partition by ticker order by close ) as bucket from stocks

**HOW TO DEPLOY / DROP UDFS**

**At start of each session:**

ADD JAR /full\_path\_to\_jar/YourUDFName.jar;

CREATE TEMPORARY FUNCTION YourUDFName AS 'org.apache.hadoop.hive.contrib.udf.example.YourUDFName';

**At the end of each session:**

DROP TEMPORARY FUNCTION IF EXISTS YourUDFName;

**HOW TO DEPLOY / DROP UDFS Permanently**

  ADD JAR /home/amal/hive/amaludf.jar

ADD JAR /home/amal/hive/amaludf2.jar

Instead of using the above commands in each session, you can define it for all sessions.

Create a location for storing these jars in the hiveserver host.

mkdir /var/lib/hive

Add all these jars to that directory

Set the property in *hive-site.xml*

<property>

<name>hive.aux.jars.path</name>

<value>/var/lib/hive</value>

</property>

Restart the hiveserver2 after doing this modification.

Instead of creating a directory and putting all the jars, you can specify paths of individual jars also. The only condition is that all these jars should be present in the hiveserver host.

<property>

<name>hive.aux.jars.path </name>

<value>file:///home/amal/hive/udf1.jar,file:///usr/lib/hive/lib/hive-hbase-handler.jar</value>

</property>

**Show Database**

SHOW DATABASES ‘e.\*’

**Creating Database in a specified Location**

CREATE DATABASE ESAk07 LOCATION ‘/user/esakkipillai/’

**Adding Comment**

CREATE DATABASE ESAk07 LOCATION ‘/user/esakkipillai/’ COMMENT ‘this is the Sample Database’

**ADDING DATABASE PROPERTY**

**CREATE DATABASE ESAK07 LOCATION ‘/USER/ESAKKIPILLAI/MYHIVE’ COMMENT ‘THIS IS THE SAMPLE DATABSE ‘WITH DBPROPERTIES (‘OWNER’=’ESAK’, ‘DATE’=’10-9-2017’)**

**DESCRIBE THE DATABASE**

**DESCRIBE DFATABASE EXTENDED ESAK07;**

**DELETE A DATABASE ( NO TABLE)**

**DROP DATABASE IF EXISTS ESAK07;**

**DELETE A DATABASE ( WITH TABLE)**

**DROP DATABASE IF EXISTS ESAK07 CASCADE;**

**ALTER DATABASE PROPERTIES**

**ALTER DATABASE ESAK07 SET DBPROPERTIES(‘OWNER’=’ESAKKIPILLAI’);**

**HOW TO COPY THE SCHEMA TO MAKE ANOTHER TABLE**

**CREATE TABLE IF NOT EXISTS ESAK\_COPY LIKE ESAK;**

**create external table mydb.employees3 like employees;**

**CHECK THE SCHEMA FOR ONLY ONE COLUMN**

**DESCRIBE DEFAULT.TABLENAME.COLUMN;**

**Compression in Hive**

First we have to check for available compression in hive

**set io.compression.codec;**

it will list the available compression codec in the hadoop cluster.

Some of them are namely

* org.apache.hadoop.io.compress.GzipCodec,
* org.apache.hadoop.io.compress.DefaultCodec,
* org.apache.hadoop.io.compress.BZip2Codec,
* org.apache.hadoop.io.compress.SnappyCodec

We need to set the Below Property in hive to enable the Compression

hive> set hive.exec.compress.output=true;

hive> set mapreduce.output.fileoutputformat.compress=true;

hive> set mapreduce.output.fileoutputformat.compress.codec = org.apache.hadoop.io.compress.GzipCodec;

hive> set mapreduce.output.fileoutputformat.compress.type=BLOCK;

hive> set hive.exec.compress.intermediate=true;

we are having the .gz file in HDFS in the location '/tmp/weblogs/20090603-access.log.gz'

CREATE TABLE raw (line STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n';

Table raw has been stored as textfile which is the default file storage format

However, in this case Hadoop will not be able to split your file into chunks/blocks and run multiple

maps in parallel. This can cause underutilization of your cluster's 'mapping' power.

CREATE TABLE raw\_sequence (line STRING)

STORED AS SEQUENCEFILE;

**ALTER TABLES**

**Partition Tables**

* Partition means dividing the data based on some column like date or country. partitioning can be done on multiple columns which impose multi-dimensional structure on directory storage.
* partition is used for distributing the load horizontally
* Partitions are defined at the time of table creation using PARTITIONED BY clause, with the list of column definitions for partitioning
* As the data is sliced/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
* In a large table if we partition the table , then select \* from table where country = 'In' will look for the data inside the Partition table In rather than the entire table employee

**Limitations**

* Having too many partition will create large number of files in the HDFS, which is an overhead for the Name Node , Since it will keep all the metadata in the memory
* Partition is optimised for some queries based on the where clause but may be less responsive for other important queries like group by clause
* IN MR Processing, Huge no of partition will lead to huge no of tasks in MR job thus creating lot of overhead to maintain start up and teardown the JVM

CREATE [EXTERNAL] TABLE table\_name (col\_name\_1 data\_type\_1, ....)

**PARTITIONED BY (col\_name\_n data\_type\_n [COMMENT col\_comment], ...);**

*PARTITIONED BY COLUMN SHOULD NOT BE AVAILABLE IN THE TABLE DEFINITION PART IT SHOULD BE IN PARTITION BY CLAUSE*

**Inserting DATA IN To Partitions**

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

* Static Partitioning
* Dynamic Partitioning

**STATIC PARTITION**

* 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)

Loading Data into Partition

INSERT OVERWRITE TABLE partitioned\_user

PARTITION (country = 'US', state = 'AL')

SELECT \* FROM another\_user au

WHERE au.country = 'US' AND au.state = 'AL';

Second Method

#Load employees data for partition having department as HR

LOAD DATA INPATH '/home/hadoop/hr\_employees.csv' INTO TABLE Employee PARTITION (department='HR');

#Load employees data for partition having department as BIGDATA

LOAD DATA INPATH '/home/hadoop/bigdata\_employees.csv' INTO TABLE Employee PARTITION (department='BIGDATA');

Overwriting Existing Partition

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 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.

* **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.**

#Insert a single row in a table partition

INSERT INTO table Employee PARTITION (department)

values(50000, 'Rakesh', 28, 57000,'HR');

#Insert Multiple rows in a table partition

INSERT INTO table Employee PARTITION (department)

values(60001, 'Sudip', 34, 62000, 'HR'),(70001, 'Suresh', 45, 76000, 'BIGDATA');

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.

    <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>

We can set the Same Property through Hive Shell also

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;

**Bucketing Tables**

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.

**Create a Bucketing Table**

CREATE TABLE Employee (

ID BIGINT,

NAME STRING,

AGE INT,

SALARY BIGINT,

DEPARTMENT STRING

)

COMMENT 'This is Employee table stored as textfile clustered by id into 5 buckets'

CLUSTERED BY(ID) INTO 5 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

We are Creating the Table as Text File and it is Clustered By 5 Buckets which it will have 5 bucket.

**Insert the Data**

INSERT OVERWRITE TABLE Employee SELECT \* from Employee\_old;

Below Files Will be Generated in HDFS

Name        Type

000000\_0    file

000001\_0    file

000002\_0    file

000003\_0    file

000004\_0    file

Create a Bucketed and Sorted Table

We have to use SORTBY Along with the Buckets to sort the data based on the column inside the table.

CREATE TABLE Employee(

ID BIGINT,

NAME STRING,

AGE INT,

SALARY BIGINT,

DEPARTMENT STRING

)

COMMENT 'This is Employee table clustered by id sorted by age into 5 buckets'

**CLUSTERED BY(ID) SORTED BY(AGE) INTO 5 BUCKETS**

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

#Insert data into Bucketed table employee

0: jdbc:hive2://localhost:10000> INSERT OVERWRITE TABLE Employee SELECT \* from Employee\_old;

* Bucketing is a method to evenly distribute the data across many files. files are placed in the Buckets based on the Hash Algorithm
* 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.
* Bucketing Feature of Hive can be used to distribute/organise the table/partition data into multiple files such that similar records are present in the same file.
* Physically, each bucket is just a file in the table directory, and Bucket numbering is 1-based
* Data for each bucket is stored in a separate HDFS file under the table directory on HDFS. Inside each bucket, we can define the arrangement of data by providing the SORT BY column while creating the table
* **Bucketed columns are included in table definition as shown in above code for state and city columns.**
* 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.
* To enable Bucketing in hive
  + set hive.enforce.bucketing= true;

lets say there are two tables user and user\_visits and both table data is bucketed using user\_id in 4 buckets . It means bucket 1 of user will contain rows with same user ids as that of bucket 1 of user\_visits. And if a join is performed on these two tables on user\_id columns, if it is possible to send bucket 1 of both tables to same mapper then good amount of optimization can be achived. This is exactly done in bucketed map join.

**Prerequisites for bucket map join:**

* Tables being joined are bucketized on the join columns,
* The number of buckets in one table is a multiple of the number of buckets in the other table; the buckets can be joined with each other

If the tables being joined are bucketized on the join columns, and the number of buckets in one table is a multiple of the number of buckets in the other table, the buckets can be joined with each other. If table A has 4 buckets and table B has 4 buckets, the following join

SELECT /\*+ MAPJOIN(b) \*/ a.key, a.valueFROM 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 on the join columns, and they have the same number of buckets, 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;**

## Advantages

* Fast Map side Joins – If two tables are bucketed by the same column(s) into same number of buckets and the join is performed on the bucketed column(s), then hive can do efficient map side join by reading the same bucket from both the tables and performing a join, as all the data for similar records will be present in the corresponding bucket from both the tables. If the records are sorted inside each bucket, then hive can join the data using merge, which is a linear time operation. Bucketing will help only when the join key and bucketing key are the same.
* Efficient Group by – If the group by is performed on the bucketed column(s), then aggregations can be performed in the combiner. This will reduce network traffic by sending less data to reducers.
* Sampling – Using Bucketing we can run queries on a sample of data from the table. This is beneficial while testing, so that we need not run our queries on whole data.

**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

SELECT firstname, country, state, city FROM bucketed\_user

TABLESAMPLE(BUCKET 32 OUT OF 32 ON state);

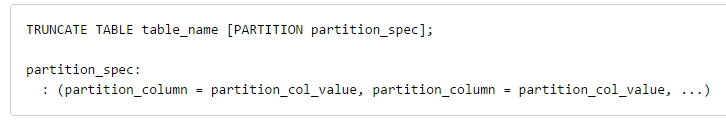
SELECT firstname, country, state, city FROM bucketed\_user TABLESAMPLE(1 PERCENT);

**Drop Tables**



* DROP TABLE removes metadata and data for this table. The data is actually moved to the .Trash/Current directory if Trash is configured (and PURGE is not specified). The metadata is completely lost.
* When dropping an EXTERNAL table, data in the table will NOT be deleted from the file system.
* When dropping a table referenced by views, no warning is given (the views are left dangling as invalid and must be dropped or recreated by the user).
* If PURGE is specified, the table data does not go to the .Trash/Current directory and so cannot be retrieved in the event of a mistaken DROP. The purge option can also be specified with the table property auto.purge

**Truncate Tables**



* Removes all rows from a table or partition(s). The rows will be trashed if the file system Trash is enabled, otherwise they are deleted

**Alter Partitions**

ALTER TABLE table\_name ADD [IF NOT EXISTS] PARTITION partition\_spec [LOCATION 'location'][, PARTITION partition\_spec [LOCATION 'location'], ...];

partition\_spec:

  : (partition\_column = partition\_col\_value, partition\_column = partition\_col\_value, ...)

ALTER TABLE page\_view ADD PARTITION (dt='2008-08-08', country='us') location '/path/to/us/part080808'

                          PARTITION (dt='2008-08-09', country='us') location '/path/to/us/part080809';

**To Recover the Partitions**

Hive stores a list of partitions for each table in its metastore. If, however, new partitions are directly added to HDFS (say by using hadoop fs -put command), the metastore (and hence Hive) will not be aware of these partitions

MSCK REPAIR TABLE table\_name;

* it will add any partitions that exist on HDFS but not in metastore to the metastore

**Drop Partitions**

ALTER TABLE page\_view DROP PARTITION (dt='2008-08-08', country='us');

**Archive Partition**

ALTER TABLE table\_name ARCHIVE PARTITION partition\_spec;

ALTER TABLE table\_name UNARCHIVE PARTITION partition\_spec;

Archiving is a feature to moves a partition's files into a Hadoop Archive (HAR). Note that only the file count will be reduced; HAR does not provide any compression.

**Alter Table/Partition File Format**

|  |
| --- |
| ALTER TABLE table\_name [PARTITION partition\_spec] SET FILEFORMAT file\_format; |
| ALTER TABLE table\_name [PARTITION partition\_spec] SET LOCATION "new location";  **TABLE / PARTITION Protection** |

ALTER TABLE table\_name [PARTITION partition\_spec] ENABLE|DISABLE NO\_DROP [CASCADE];

ALTER TABLE table\_name [PARTITION partition\_spec] ENABLE|DISABLE OFFLINE;

* Protection on data can be set at either the table or partition level. Enabling NO\_DROP prevents a table from being [dropped](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropPartitions). Enabling OFFLINE prevents the data in a table or partition from being queried, but the metadata can still be accessed.
* If any partition in a table has NO\_DROP enabled, the table cannot be dropped either. Conversely, if a table has NO\_DROP enabled then partitions may be dropped, but with NO\_DROP CASCADE partitions cannot be dropped either unless the [drop partition command](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-DropPartitions) specifies IGNORE PROTECTION.

**Alter Table/Partition CONCATENATE**

* If the table or partition contains many small RCFiles or ORC files, then the above command will merge them into larger files. In case of RCFile the merge happens at block level whereas for ORC files the merge happens at stripe level thereby avoiding the overhead of decompressing and decoding the data.

**ALTER TABLE table\_name [PARTITION (partition\_key = 'partition\_value' [, ...])] CONCATENATE;**

**ALTER TABLE**

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

ALTER TABLE table\_name SET TBLPROPERTIES table\_properties;

LTER TABLE table\_name SET TBLPROPERTIES ('comment' = new\_comment);

**ALTER COLUMNS**

**Change Column Type comment**

ALTER TABLE table\_name [PARTITION partition\_spec] CHANGE [COLUMN] col\_old\_name col\_new\_name column\_type

  [COMMENT col\_comment] [FIRST|AFTER column\_name] [CASCADE|RESTRICT];

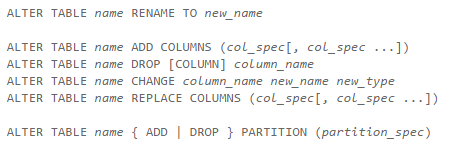
ALTER TABLE table\_name

  [PARTITION partition\_spec]                 -- (Note: Hive 0.14.0 and later)

  ADD|REPLACE COLUMNS (col\_name data\_type [COMMENT col\_comment], ...)

  [CASCADE|RESTRICT]

**Alter Tables**

****

|  |  |
| --- | --- |
| **Name** | **command** |
| Rename Table | ALTER TABLE TABLE\_NAME RENAME TO NEW\_TABLE\_NAME |
| ALTER TABLE PROPERTIES | ALTER TABLE table\_name SET TBLPROPERTIES ('comment' = new comment); |
| ADD SERDE PROPERTIES | ALTER TABLE TABLE\_NAME SET SERDE org.apache.hadoop.hive.serde2.OpenCSVSerde;  ALTER TABLE table\_name SET SERDEPROPERTIES ('field.delim' = ','); |
| ALTER PARTITION | alter table p1 add partition (month=1, day=1); |
| ALTER TABLE PROPERTIES | ALTER TABLE table\_name SET TBLPROPERTIES ('key1'='value1', 'key2'='value2'[, ...]); ALTER TABLE table\_name SET SERDEPROPERTIES ('key1'='value1', 'key2'='value2'[, ...]); |
| ALTER COLUMNS | ALTER TABLE table\_name ADD COLUMNS (column\_defs); ALTER TABLE table\_name REPLACE COLUMNS (column\_defs); ALTER TABLE table\_name CHANGE column\_name new\_name new\_type; ALTER TABLE table\_name DROP column\_name; |
| ALTER CONSTRAINTS | ALTER TABLE table\_name DROP CONSTRAINT constraint\_name; ALTER TABLE table\_name ADD CONSTRAINT constraint\_name PRIMARY KEY (column, ...) DISABLE NOVALIDATE; |
| SET FILE FORMAT | ALTER TABLE table\_name [PARTITION partition\_spec] SET FILEFORMAT file\_format; |
| ALTER COLUMNS | CREATE TABLE test\_change (a int, b int, c int); // First change column a's name to a1. ALTER TABLE test\_change CHANGE a a1 INT; // Add a comment to column a1 ALTER TABLE test\_change CHANGE a1 a1 INT COMMENT 'this is column a1'; |
| ALTER COLUMNS | // Next change column a1's name to a2, its data type to string, and put it after column b. ALTER TABLE test\_change CHANGE a1 a2 STRING AFTER b; // The new table's structure is: b int, a2 string, c int. |
| ALTER COLUMNS | // Then change column c's name to c1, and put it as the first column. ALTER TABLE test\_change CHANGE c c1 INT FIRST; // The new table's structure is: c1 int, b int, a2 string. |
| ALTER COLUMNS | ALTER TABLE foo PARTITION (ds, hr) CHANGE COLUMN dec\_column\_name dec\_column\_name DECIMAL(38,18); |

**Hive namespaces (Variables and Properties)**

* **Hivevar**
* **Hiveconf**
* **System**
* **Env**

**Hivevar**

This is useful in defining our own custom variables for example

*Hive> set hivevar:toodey=hello;*

*Hive> set toodey;*

Now we can use this in any Hive query statement as we have seen in my previous tutorial about create table.

*CREATE TABLE toodey(t STRING, $(toodey) STRING);*

**hiveconf**

This is useful when we need to set hive specific configuration parameters.

Example –

Let’s say we want to set our current working db to console or hive CLI, by default its set to FALSE.

*Hive> set hiveconf:hive.cli.print.current.db=true;*

*Hive (default)>*

**System**

This namespace provides read and write access to java system properties for example.

*Hive> set system:user.name=toodeyuser*

**Env**

It is useful in setting environment variable define by the shell (Bash)

Example

*Hive> set env:YEAR=2017*

*Hive> SELECT \* FROM toodey WHERE year = $ {env:YEAR};*

**Views**

CREATE VIEW [IF NOT EXISTS] [db\_name.]view\_name [(column\_name [COMMENT column\_comment], ...) ]

  [COMMENT view\_comment]

  [TBLPROPERTIES (property\_name = property\_value, ...)]

  AS SELECT ...;

* A view's schema is frozen at the time the view is created; subsequent changes to underlying tables (e.g. adding a column) will not be reflected in the view's schema. If an underlying table is dropped or changed in an incompatible fashion, subsequent attempts to query the invalid view will fail.
* A CREATE VIEW statement will fail if the view's defining SELECT expression is invalid.
* If no column names are supplied, the names of the view's columns will be derived automatically from the defining SELECT expression. (If the SELECT contains unaliased scalar expressions such as x+y, the resulting view column names will be generated in the form \_C0, \_C1, etc.

DROP VIEW [IF EXISTS] [db\_name.]view\_name;

ALTER VIEW [db\_name.]view\_name SET TBLPROPERTIES table\_properties;

table\_properties:

: (property\_name = property\_value, property\_name = property\_value, ...)

**Index**

* The short answer is no. Indexes in Hive are not recommended.
* The reason for this is **ORC**. ORC has built in Indexes which allow the format to skip blocks of data during read, they also support Bloom filters. Together this pretty much replicates what Hive Indexes did and they do it automatically in the data format without the need to manage an external table ( which is essentially what happens in indexes. ). I would rather spend my time to properly setup the ORC tables.

**Bee Line**

**Joins**

**0: jdbc:hive2://localhost:10000> select \* from employee;**

**+--------------+----------------+---------------+------------------+----------------------+--+**

**| employee.id | employee.name | employee.age | employee.salary | employee.department |**

**+--------------+----------------+---------------+------------------+----------------------+--+**

**| 80001 | Aarti | 25 | 37000 | BIGDATA |**

**| 80003 | Rajesh | 29 | 59000 | BIGDATA |**

**| 70001 | Suresh | 45 | 76000 | FINANCE |**

**| 80002 | Neha | 27 | 39000 | FINANCE |**

**| 60001 | Sudip | 34 | 62000 | HR |**

**| 80005 | Rahul | 24 | 35000 | HR |**

**+--------------+----------------+---------------+------------------+----------------------+--+**

**Refunds table containing refunds information stored along-with employee id:-**

**0: jdbc:hive2://localhost:10000> select \* from refunds;**

**+--------------------+-----------------+-----------------+--+**

**| refunds.refund\_id | refunds.emp\_id | refunds.amount |**

**+--------------------+-----------------+-----------------+--+**

**| 1 | 60001 | 10000 |**

**| 2 | 60001 | 15000 |**

**| 3 | 70001 | 25000 |**

**| 4 | 80001 | 12000 |**

**| 5 | 70001 | 13000 |**

**| 6 | 80003 | 17000 |**

**| 7 | 80002 | 21000 |**

**| 8 | 80001 | 31000 |**

**| 9 | 80003 | 3000 |**

**| 10 | 30001 | 30000 |**

**+--------------------+-----------------+-----------------+--+**

**Join using a where clause**

Select a.id , a.name,a.age,a.salary a.dept from employee a , refund b

Where a.id = b.emp\_id

Using a Join

select

e.id as emp\_id, e.name as emp\_name, r.amount as refund\_amount

from employee e join refunds r

on e.id=r.emp\_id;

* Inner join – Keywords to be used in query(join/inner join). Returns all the rows where there is match for join condition in both the tables. Default join used is the inner join.
* Left join – Keywords to be used in query(left join/left outer join). Returns all the rows from the left table and only those rows from the right table for which there is a match for join condition.
* Right join– Keywords to be used in query(right join/right outer join). Returns all the rows from the right table and only those rows from the left table for which there is a match for join condition.
* Full Join – Keywords to be used in query(full join/full outer join). Returns all the rows from the both the tables with nulls in place where there is no match for the join condition.

#Left Join

select e.id as emp\_id, e.name as emp\_name, r.amount as refund\_amount

from employee e left join refunds r

on e.id=r.emp\_id;

#Right Join

select e.id as emp\_id, e.name as emp\_name, r.amount as refund\_amount

from employee e right join refunds r

on e.id=r.emp\_id;

#Full join

select e.id as emp\_id, e.name as emp\_name, r.amount as refund\_amount

from employee e full join refunds r

on e.id=r.emp\_id;

**Join implementation with Map Reduce**

* Hive converts joins over multiple tables into a single map/reduce job if for every table the same column is used in the join clauses. The query below is converted into a single map/reduce job as only key1 column for b is involved in the join.

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

* It is very interesting to note that any number of tables can be joined in single map/reduce process as long as they fit the above criteria.
* However if the join columns are not the same for all tables the is converted into multiple map/reduce jobs

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

* In this case the first map/reduce job joins a with b and the results are then joined with c in the second map/reduce job.

**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;**

<http://www.openkb.info/2014/11/understanding-hive-joins-in-explain.html>

**Sub queries**

A sub query is a SELECT statement that is embedded in another SQL statement. Hive has limited support for sub queries, only permitting a sub query in the **FROM clause of a SELECT statement**.

**Tranactions In Hive**

**Row Level Updates are supported By Hive after 0.14 version with ORC FILE**

**Steps**

* Create An ORC TABLE with transactional Enabled
* Insert the Data

**CREATE TABLE IF NOT EXISTS esakraw.olympics\_txmplx(**

**Discipline STRING,**

**Athlete STRING,**

**Country STRING,**

**Event STRING,**

**Medal STRING,**

**City STRING**

**)**

**COMMENT 'SAMPLE OLYMPICS DATA with ORC and ZLIB'**

**PARTITIONED BY (Gender STRING , Sport STRING)**

**CLUSTERED BY (City ) into 16 BUCKETS**

**ROW FORMAT DELIMITED**

**FIELDS TERMINATED BY '\t'**

**LINES TERMINATED BY '\n'**

**STORED AS ORC**

**TBLPROPERTIES('transactional'='true');**

**INSERT INTO TABLE esakraw.olympics\_txmplx PARTITION (Gender,Sport)**

**SELECT Discipline,Athlete,Country,Event,Medal,City,Gender,Sport FROM esakraw.summerolympicsraw;**

**select \* from esakraw.olympics\_txmplx**

**where Gender='Men'**

**and Sport='Wrestling' and Country='USA';**

FAILED: SemanticException [Error 10295]: INSERT OVERWRITE not allowed on table with OutputFormat that implements AcidOutputFormat while transaction manager that supports ACID is in use

**Since the table is external and we enavle the ACID property it throws error**

FAILED: SemanticException [Error 10298]: ACID insert, update, delete not supported on tables that are sorted, table olympics\_txn

**Update The Data**

**UPDATE esakraw.olympics\_txmplx**

**SET Country='America Karan'**

**where Country='USA'**

* The referenced column must be a column of the table being updated.
* The value assigned must be an expression that Hive supports in the select clause.  Thus arithmetic operators, UDFs, casts, literals, etc. are supported.  Subqueries are not supported.
* Only rows that match the WHERE clause will be updated.
* Partitioning columns cannot be updated.
* Bucketing columns cannot be updated.
* In Hive 0.14, upon successful completion of this operation the changes will be auto-committed.

**INSERT**

* Each row listed in the VALUES clause is inserted into table tablename.
* Values must be provided for every column in the table.  The standard SQL syntax that allows the user to insert values into only some columns is not yet supported.  To mimic the standard SQL, nulls can be provided for columns the user does not wish to assign a value to.
* Dynamic partitioning is supported in the same way as for INSERT...SELECT.
* If the table being inserted into supports ACID and a transaction manager that supports ACID is in use, this operation will be auto-committed upon successful completion.

**DELETE**

* Only rows that match the WHERE clause will be deleted.
* In Hive 0.14, upon successful completion of this operation the changes will be auto-committed.

**Convert One File Format to Another**

*I have the Text File Format Table and i want to convert it Avro*

*Create table departments\_avro ( dept int , deptname String)*

*STORED AS AVRO*

*INSERT INTO TABLE DEPARTMENTS\_AVRO SELECT & FROM DEPARTMENTS;*

**1)How to control the number of Mappers and Reducers in Hive on Tez.**

**2) How to enable Fetch Task instead of MapReduce Job for simple query in Hive**

**Goal:**

Certain simple Hive queries can utilize fetch task, which can avoid the overhead of starting MapReduce job.

<http://www.openkb.info/2015/01/how-to-enable-fetch-task-instead-of.html>

hive.fetch.task.conversion = none

hive.fetch.task.conversion = minimal

hive.fetch.task.aggr = false select count(\*) trigger MR

hive.fetch.task.aggr = true select count(\*) trigger FetchOperation

**3) How to add the Auxiliary Jar into Hive**

Modify the hive-site.xml

<property>

<name>hive.aux.jars.path</name>

<value>/opt/mapr/hive/hive-0.13/aux/hive-json-serde-0.2.jar</value>

</property>

**HIVE SHELL**

hive> ADD JAR /opt/mapr/hive/hive-0.13/aux/hive-json-serde-0.2.jar;

Added /opt/mapr/hive/hive-0.13/aux/hive-json-serde-0.2.jar to class path

Added resource: /opt/mapr/hive/hive-0.13/aux/hive-json-serde-0.2.jar

hive> list jars;

/opt/mapr/hive/hive-0.13/aux/hive-json-serde-0.2.jar

4) Hive Query Runs Out of heap Memory when shuffle in memory

Java.lang.Outofmemory

Increase mapred.reduce.child.java.opts after fully understanding the memory usage on the whole cluster.

Please refer to this article for details about Five Steps to Avoiding Java Heap Space Errors.  
Do not blindly increase this memory setting since it may cause other service or jobs running out of memory.  
In hive shell:

**set mapred.reduce.child.java.opts=-Xmx8192m;**

Decrease mapred.job.shuffle.input.buffer.percent from default 0.70 to 0.20 for example

**set mapred.job.shuffle.input.buffer.percent=0.20;**

**how to set all the Hive Properties in the CLI**

|  |  |
| --- | --- |
| set | Prints a list of configuration variables that are overridden by the user or Hive. |
| set -v | Prints all Hadoop and Hive configuration variables. |

|  |  |
| --- | --- |
| set hive.cli.print.current.db=true; | Shows the current database in the command prompt |
| set hive.cli.print.header=true; | Shows column names at top of query results |

**We can add Files jar**

**ADD FILES <filepath>**

**ADD JARS <jarpath>**

**Authentication**

In Hive, by default Authorization will not be enabled. But Hive provides three different types of Authorization models to enable security on the Hive data.

1. Hive Default Authorization
2. Hive – Storage Based Authorization (SBA)
3. Hive – SQL Standards Based Authorization (SSBA)

Misc

<http://bigdataprogrammers.com/parse-xml-data-in-hive/>

<https://community.hortonworks.com/questions/40979/hive-xml-parising-null-value-returned.html>

<https://community.hortonworks.com/content/kbentry/972/hive-and-xml-pasring.html>

**DISTRIBUTE BY**

Distribute By is used to distribute the rows to different reducers based on the value(s) of column(s). All rows with the same Distribute By columns will go to the same reducer. This is like partitioning in Map-Reduce, where all the records having same value of partition goes to the same reducer. Distribute By does not guarantee clustering the rows, based on the distributed By columns, in the reducers.

#set the number of reducers to 2

0: jdbc:hive2://localhost:10000> **set mapred.reduce.tasks=2**;

0: jdbc:hive2://localhost:10000> **select \* from employee distribute by id;**

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

| employee.id  | employee.name  | employee.age  | employee.salary  | employee.department  |

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

| 10002        | rahul          | 23            | 250000           | BIGDATA              |

| 2            | sakshi         | 22            | 60000            | HR                   |

| 10001        | rajesh         | 29            | 50000            | BIGDATA              |

| 10003        | dinesh         | 35            | 70000            | BIGDATA              |

| 1            | aarti          | 28            | 55000            | HR                   |

| 3            | mahesh         | 25            | 25000            | HR                   |

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

All rows in Red (with even id) were sent to one reducer and all rows in Blue (with Odd id)were went to second Reducer. Thus we can use distribute by to control the distribution of rows to different reducers.

**Distribute by with Sort by**

If we club Distribute by with Sort by, then we can control the clustering of rows inside each reducer based on the value of some columns.

#set the number of reducers to 2

0: jdbc:hive2://localhost:10000> **set mapred.reduce.tasks=2**;

0: jdbc:hive2://localhost:10000> **select \* from employee distribute by id sort by salary;**

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

| employee.id  | employee.name  | employee.age  | employee.salary  | employee.department  |

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

| 2            | sakshi         | 22            | 60000            | HR                   |

| 10002        | rahul          | 23            | 250000           | BIGDATA              |

| 3            | mahesh         | 25            | 25000            | HR                   |

| 10001        | rajesh         | 29            | 50000            | BIGDATA              |

| 1            | aarti          | 28            | 55000            | HR                   |

| 10003        | dinesh         | 35            | 70000            | BIGDATA              |

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

All rows in Red (with even id) were sent to one reducer and all rows in Blue (with Odd id)were went to second Reducer. Inside each reducer, the rows are sorted by salary.

**CLUSTER BY**

Cluster by is same as ‘Distribute by and Sort by’. But Cluster by does the distribution and sorting on same columns. If we want to distribute by some columns and then sort by some other columns, then we should use ‘ distribute by with sort by’ instead of ‘cluster by’. We will see what a query with ‘cluster by id‘ returns.

#set the number of reducers to 2

0: jdbc:hive2://localhost:10000> **set mapred.reduce.tasks=2**;

0: jdbc:hive2://localhost:10000> **select \* from employee cluster by id;**

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

| employee.id  | employee.name  | employee.age  | employee.salary  | employee.department  |

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

| 2            | sakshi         | 22            | 60000            | HR                   |

| 10002        | rahul          | 23            | 250000           | BIGDATA              |

| 1            | aarti          | 28            | 55000            | HR                   |

| 3            | mahesh         | 25            | 25000            | HR                   |

| 10001        | rajesh         | 29            | 50000            | BIGDATA              |

| 10003        | dinesh         | 35            | 70000            | BIGDATA              |

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

The above output is similar to ‘**distribute by id sort by id**‘ as shown below:-

#set the number of reducers to 2

0: jdbc:hive2://localhost:10000> **set mapred.reduce.tasks=2**;

0: jdbc:hive2://localhost:10000> **select \* from employee distribute by id sort by id;**

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

| employee.id  | employee.name  | employee.age  | employee.salary  | employee.department  |

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

| 2            | sakshi         | 22            | 60000            | HR                   |

| 10002        | rahul          | 23            | 250000           | BIGDATA              |

| 1            | aarti          | 28            | 55000            | HR                   |

| 3            | mahesh         | 25            | 25000            | HR                   |

| 10001        | rajesh         | 29            | 50000            | BIGDATA              |

| 10003        | dinesh         | 35            | 70000            | BIGDATA              |

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

**Some Mostly Used Configuration**

|  |
| --- |
| show conf "hive.metastore.warehouse.dir"; |
| set hive.execution.engine |
| set mapred.reduce.tasks; |
| set hive.exec.dynamic.partition.mode |
| set hive.exec.parallel |
| set hive.exec.parallel.thread.number |

**Configure Hive Metastore on MySQL**

We will see how to configure Hive metastore on MySQL.

Create User and Database for Hive Metastore

**Create User :-**

mysql> CREATE USER 'hiveuser'@'localhost' IDENTIFIED BY 'password';

Query OK, 0 rows affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON \*.\* TO 'hiveuser'@'localhost' WITH GRANT OPTION;

Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;

Query OK, 0 rows affected (0.00 sec

**Create Database :-**

mysql -u hiveuser –ppassword

mysql> create database hivemetastore;

mysql> use hivemetastore;

mysql> exit;

**Configure hive-site.xml**

Add the details about the Mysql connection and database to the hive-site.xml.

**<property>**

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

**<value>jdbc:mysql://localhost/hivemetastore?createDatabaseIfNotExist=true</value>**

**<description>JDBC connect string for a JDBC metastore</description>**

**</property>**

**<property>**

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

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

**<description>Driver class name for a JDBC metastore</description>**

**</property>**

**<property>**

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

**<value>hiveuser</value>**

**<description>Username to use against metastore database</description>**

**</property>**

**<property>**

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

**<value>password</value>**

**<description>password to use against metastore database</description>**

**</property>**

Now the Hive will point to the metastore DB on Mysql.

**Add Mysql connector jar**

To enable Hive to connect to Mysql DB, add mysql jdbc connector jar in the lib directory under Hive installation directory.

**Get all the TABLE Name and its structure in a file**

Steps to generate Create table ddls for all the tables in the Hive database and export into text file to run later:

**step 1)** create a .sh file with the below content ,say hive\_table\_ddl.sh

#!/bin/bash

rm -f tableNames.txt

rm -f HiveTableDDL.txt

hive -e "use $1; show tables;" > tableNames.txt

wait

cat tableNames.txt |while read LINE

do

hive -e "use $1;show create table $LINE" >>HiveTableDDL.txt

echo -e "\n" >> HiveTableDDL.txt

done

rm -f tableNames.txt

echo "Table DDL generated"

**step 2)** Run the above shell script by passing 'db name' as paramanter

>bash hive\_table\_dd.sh <<databasename>>

output :

All the create table statements of your DB will be written into the HiveTableDDL.txt

**Optimization Tech**

* if u join two tables and one table is small and another one is big we need to use map side join
  + set hive.auto.convert.join=true
* Order by use one reducer where as sort by uses multiple reducer so if u use large amount of data use sort by rather than order by
* To increase the Parallelism use the below command
  + set hive.exec.compress.output= true
  + set mapred.max.split.size=25600000;
  + set mapred.output.compression.type=BLOCK
  + set mapred.output.compression.codec=org.apache.hadoop.io.compress.SnappyCodec
* To decrese the Burden on namenode

add these to the property

<name>mapred.compress.map.output</name>

<value>true</value>

* SMB Join ==> Sort Merge Bucket Join is faster than the Map Join but it's used when u have sorted and bucketed the table
  + set hive.auto.convert.sortmerge.join = true
  + set hive.optimize.bucketmapjoin=true
  + set hive.optimize.bucketmapjoin.sortedmerge = true
* use skewed Table
* vectorization - it will improve the query Performance it combines multiple rows instead of single row each time use given code to enable it

* + set hive.vectorized.execution.enabled =true ;
  + set hive.vectorized.execution.reduce.enabled =true;
* ORC hive table will improve the performance. SNAPPY is the best compression technique use it with ORC