**HIVE DATA DEFINITIONS**

Hadoop stores structured data with the help of HIVE in form of databases and tables, in order to facilitate querying the data stored at HDFS in form of databases, tables and views.

**DATABASES:**

Data is stored in databases in form of tabular format in tables. For classification of data, different databases are created. If no database is mentioned, tables will be created in default databases.

If we need to categorize tables in different databases, we will create the database first and then create corelated tables inside that database.

PFB the syntax to create DATABASE:

hive> CREATE DATABASE [IF NOT EXISTS] database\_name;

We can list the databases with below command:

hive> SHOW DATABASES like ‘d%’;

Above command will list all the databases in hive starting with ‘d’.

To create tables into a database we need to use below command:

hive> USE database\_name;

hive> SET hive.cli.print.current.db = true;

Databases which have been created, will be stored at below default location as below:

/user/hive/warehouse/database\_name.db

To change the directory of a database, below command will be used:

Hive> CREATE DATABASE [IF NOT EXISTS] database\_name

LOCATION ‘new\_hdfs\_location’;

To drop the database, below command will be used:

Hive> DROP DATABASE [IF EXISTS] database\_name;

**Tables:**

Tables in a HIVE Database can be created with below command:

hive(database\_name)> CREATE TABLE [IF NOT EXISTS] table\_name (

Col1 datatype,

Col2 datatype,

……

)

ROW FORMAT FIELD DELIMITED BY ‘delimiter’

[STORED AS file\_format]

;

Partitioning of Table can be performed with below command:

hive(database\_name)> CREATE TABLE [IF NOT EXISTS] table\_name

(

Col1 datatype,

Col2 datatype,

Col3 complex\_datatype,

……

)

ROW FORMAT FIELD DELIMITED BY ‘delimiter’

COLLECTION ITEMS TERMINATED BY ‘delimiter\_for\_complex\_datatype’

;

Bucketing can be performed by below syntax.

hive(database\_name)> CREATE TABLE [IF NOT EXISTS] table\_name (

Col1 datatype,

Col2 datatype,

…………………………

)

ROW FORMAT FIELD DELIMITED BY ‘delimiter’

CLUSTERED BY (column\_name) INTO number\_of\_buckets BUCKETS

STORED AS file\_format;

File\_format values can be any value mentioned below:

1.TEXTFILE

2.SEQUENCEFILE

3.RCFILE

4.ORCFILE

5.PARQUET

Tables in a particular database can be listed with below commands:

hive(database\_name)> SHOW TABLES;

Below command lists all the databases which starts with character s.

hive(database\_name)> SHOW TABLES like ‘s%’;

Below command lists all the databases which ends with character s.

hive(database\_name)> SHOW TABLES ‘.\*s';

Below Command lists all the columns which are present in a table which has been created successfully.

hive(database\_name)> DESCRIBE table\_name;

Table name can be altered with below command:

hive(database\_name)> ALTER TABLE table\_name RENAME TO table\_new\_name;

New columns can be added with below command:

hive(database\_name)> ALTER TABLE table\_name ADD COLUMNS (new\_col1 datatype, new\_col2 datatype);

Below commands drops a table and deletes data stored in corresponding table:

hive(database\_name)> DROP TABLE [IF EXISTS] table\_name;

**Views:**

Below command can be used to create a view for restricting the result rows based on the value of one or more columns.

hive(database\_name)> CREATE VIEW [IF NOT EXISTS] view\_name AS SELECT\_QUERY;

User can write any complex select query based on the usecase.

User can query the data from view with below command:

hive(database\_name)> SELECT \* FROM view\_name;

Below command will drop a view:

hive(database\_name)> DROP VIEW [IF EXISTS] view\_name;

**HQL MANIPULATIONS**

**Load Operation**

Below command will load and overwrite data into Hive table from local file system:

hive(database\_name)> LOAD DATA LOCAL INPATH 'local\_file\_directory' OVERWRITE INTO TABLE table\_name;

Below command will load and append data into Hive table from local file system:

hive(database\_name)> LOAD DATA LOCAL INPATH 'local\_file\_directory' INTO TABLE table\_name;

Below command will load and overwrite data into Hive table from HDFS file system:

hive(database\_name)> LOAD DATA INPATH 'hdfs\_file\_directory' OVERWRITE INTO TABLE table\_name;

Below command will load and append data into Hive table from HDFS file system :

hive(database\_name)> LOAD DATA INPATH 'hdfs\_file\_directory' INTO TABLE table\_name;

Below command will load and overwrite a partitioned table based on the partitioned column value from local file system:

hive(database\_name)> LOAD DATA LOCAL INPATH 'local\_file\_directory’

OVERWRITE INTO TABLE table\_name

PARTITION (partitioned\_column\_name ='partitioned\_column\_value');

Below command will load and overwrite a partitioned table based on the partitioned column value from HDFS file system:

hive(database\_name)> LOAD DATA INPATH 'Hdfs\_file\_directory’

OVERWRITE INTO TABLE table\_name

PARTITION (partitioned\_column\_name ='partitioned\_column\_value');

**SELECTS AND FILTERS OPERATION:**

Below command will display entire data of HIVE Table:

hive(database\_name)> SELECT \* FROM table\_name;

Below command will display a particular column on a given table:

hive(database\_name)> SELECT t.col1 FROM table\_name t;

Below command will filter records based on the condition mentioned with WHERE Clause:

hive(database\_name)> SELECT t.col1 FROM table\_name t

WHERE condition1

[,AND condition2]

[,OR condition3];

**GROUP BY OPERATION:**

Grouping can be performed based on one column or multiple column values:

hive(database\_name)> SELECT \* FROM table\_name t GROUP BY t.col1;

hive(database\_name)> SELECT \* FROM table\_name t GROUP BY t.col1,t.col2;

**ORDER BY OPERATION:**

Ordering can be performed based on one column or multiple column values in either ascending or descending order. By default, column values will be sorted in ascending order.

hive(database\_name)> SELECT \* FROM table\_name t ORDER BY t.col1;

hive(database\_name)> SELECT \* FROM table\_name t ORDER BY t.col1 ASC;

hive(database\_name)> SELECT \* FROM table\_name t ORDER BY t.col1 DESC;

**UPDATE/INSERT/DELETE OPERATIONS:**

There operations are available from HIVE 0.14.

**Update Syntax:** This command updates records in a table.

hive(database\_name)> UPDATE table\_name SET column = value [, column = value ...] [WHERE expression];

**Delete Syntax:** This command deletes records from a table.

hive(database\_name)> DELETE FROM table\_name [WHERE expression];

**Insert Syntax:** This command inserts records in a table.

hive(database\_name)> INSERT INTO TABLE table\_name [PARTITION (partcol1[=val1], partcol2[=val2] ...)] VALUES values\_row [, values\_row ...]

Where values\_row is:

(value [, value ...]);

where a value is either null or any valid SQL literal

**Merge Syntax:** This command allows actions to be performed on a target table based on the results of a join with a source table.

hive(database\_name)> MERGE INTO <target table> AS T USING <source expression/table> AS S

ON <boolean expression1>

WHEN MATCHED [AND <boolean expression2>] THEN UPDATE SET <set clause list>

WHEN MATCHED [AND <boolean expression3>] THEN DELETE

WHEN NOT MATCHED [AND <boolean expression4>] THEN INSERT VALUES<value list>

In order to perform above operations, set below properties:

hive>set hive.support.concurrency = true;

hive>set hive.enforce.bucketing = true;

hive>set hive.exec.dynamic.partition.mode = nonstrict;

hive>set hive.txn.manager = org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;

hive>set hive.compactor.initiator.on = true;

hive>set hive.compactor.worker.threads = a positive number on at least one instance of the Thrift metastore service;

**JOIN OPERATIONS:**

hive> SELECT a.\*, b.\* from table1 a JOIN table2 b ON (a.col1 = b.col2) ;

Below command will return all rows of table1.It will return records corresponding to b.value, provided that if col1 and col2 matches.

If there is no match of col1 and col2 , it will return NULL for b.value.

hive> SELECT a.value, b.value from table1 a LEFT OUTER JOIN table2 b ON (a.col1 = b.col2) ;

**Reduce-Side Join:**

If datasets are too large, reduce side join is used.

Hive> SET hive.auto.convert.join=false;

hive> SELECT \* FROM table1 u INNER JOIN table2 s

ON u.id = s.id;

number of mappers used: 2; number of reducers used: 1

**Map- Side Join:**

If datasets are small, map side join is used.

Hive> SET hive.auto.convert.join=true;

hive> SELECT \* FROM table1 u INNER JOIN table2 s

ON u.id = s.id;

number of mappers used: 1; number of reducers used: 0

**Bucket Map Join:**

Data must be bucketed on the keys, which has been used on the ON clause. Number of buckets of one table should be multiple of number of buckets of another table for this BUCKET MAP join to be utilized.

Hive> set hive.optimize.bucketmapjoin=true;

Hive> SET hive.auto.convert.join=true;

hive> SELECT \* FROM table1 u INNER JOIN buck\_table2 s

ON u.id = s.id;

**SMBM JOIN**

SORT-MERGE-BUCKET-MAP JOIN

These are map side joins which can be used on sorted and bucketed tables.

This join merge these tables.

Hive> set hive.enforce.sortmergebucketmapjoin=false;

hive> set hive.auto.convert.sortmerge.join=true;

hive> set hive.optimize.bucketmapjoin = true;

hive> set hive.optimize.bucketmapjoin.sortedmerge = true;

--to disable map-join

Hive> SET hive.auto.convert.join=false;

hive> SELECT \* FROM table1 u INNER JOIN table2 s

ON u.id = s.id;

No MapLocal Task to create hash table.

Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 0

**LEFT SEMI JOIN**

Left semi join is optimal as compared to INNER join/LEFT OUTERJOIN, as it returns from left table only for keys existing in the right table.

Select and where clause can’t reference columns from right hand table.

SELECT u.name FROM table1 u LEFT SEMI JOIN table2 s

ON u.id = s.id;