**H i v e**

The Apache Hive ™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive.

Apache Hive is a data warehouse system for data summarization and analysis and for querying of large data systems in the open-source Hadoop platform. It converts SQL-like queries into MapReduce jobs for easy execution and processing of extremely large volumes of data.

**Hive Architecture: -- https://cwiki.apache.org/confluence/display/Hive/Design**

**Create Database Statement** - Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables.

Hive contains a default database named **default**.

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

hive> CREATE DATABASE IF NOT EXISTS userdb; OR

hive> CREATE SCHEMA userdb;

hive> SHOW DATABASES;

default  
userdb

**Programmatically to create Database in hive – Example1**

import java.sql.SQLException;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.Statement;  
import java.sql.DriverManager;

public class **HiveCreateDB** {  
 private static String driverName = "**org.apache.hadoop.hive.jdbc.HiveDriver**";   
 public static void main(String[] args) throws SQLException {

// Register driver and create driver instance  
 Class.forName(driverName);

// get connection – default is DB/schema name  
 Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/default", "", "");

// Create statement  
 Statement stmt = con.createStatement();

// Execute Query  
 stmt.executeQuery("CREATE DATABASE userdb");  
 System.out.println(“Database userdb created successfully.”);

//Close the connection  
 con.close();

}   
}

**Drop Database Statement** - Drop Database is a statement that drops all the tables and deletes the database.

DROP (DATABASE|SCHEMA) [IF EXISTS] database\_name [RESTRICT|CASCADE];

-> Drops the database using CASCADE means dropping respective tables before dropping the database.

hive> DROP DATABASE IF EXISTS userdb CASCADE;

hive> DROP SCHEMA userdb;

**// JDBC Execute Query Statement use this statement in above Example1 program 😊**

stmt.executeQuery("DROP DATABASE userdb");

**Create Table Statement** - Create Table is a statement used to create a table in Hive in current DB. There are two types of table in hive: managed and external.

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db\_name.] table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)]

[COMMENT table\_comment]

[ROW FORMAT row\_format]

[STORED AS file\_format]

hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,

salary String, destination String)

COMMENT ‘Employee details’

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ‘\t’

LINES TERMINATED BY ‘\n’

STORED AS TEXTFILE;

// JDBC Execute Query Statement

stmt.executeQuery("CREATE TABLE IF NOT EXISTS employee ( eid int, name String, salary String, destignation String) "

+" COMMENT ‘Employee details’ ROW FORMAT DELIMITED"

+" FIELDS TERMINATED BY ‘\t’ LINES TERMINATED BY ‘\n’ STORED AS TEXTFILE;");

**Complex Data Types**

arrays: ARRAY<data\_type>  
maps: MAP<primitive\_type, data\_type>  
structs: STRUCT<col\_name : data\_type [COMMENT col\_comment], …>

**ARRAY:**

Dataset\_Temperature

1/2/17 Karnataka 23.2,22.3,20.5,25.5,24.5,20.3  
1/2/17 Maharastra 25.2,23.3,22.5,24.5,24.5,26.3  
1/2/17 Jharkhand 25.2,23.3,20.5,22.5,23.5,20.3

hive> create table Temperature(date string,city string, **MyTemp ARRAY<DOUBLE>**) row format delimited fields terminated by ‘\t’ **COLLECTION ITEMS TERMINATED BY ‘,’;**

hive> describe Temperature

OK

date string  
city string  
mytemp array<double>

hive> **select \* from Temperature**

1/2/17 Karnataka [23.2,22.3,20.5,25.5,24.5,20.3]  
1/2/17 Maharastra [25.2,23.3,22.5,24.5,24.5,26.3]  
1/2/17 Jharkhand [25.2,23.3,20.5,22.5,23.5,20.3]

To select a column and a value from the table we can use **index 0 to N-1** as the below command.

hive>select city,**MyTemp[0]** from Temperature;

Karnataka 23.2  
Maharastra 25.2  
Jharkhand 25.2

**Map**   
SecondarySchool Assam Male 2015:56897,2016:575757,2017:585858  
SecondarySchool Assam Female 2015:19947,2016:20287,2017:33552  
SecondarySchool Jharkhand Male 2015:456987,2016:415263,2017:362514  
SecondarySchool Jharkhand Female 2015:12453,2016:52146,2017:63254

Map is a **collection of key-value pairs** where fields are accessed using array notation of keys Eg: [‘Key’]

hive> create table MySchools(schooltype string,state string,gender string, **TOTAL MAP<INT,INT>**) row format delimited fields terminated by ‘\t’ **COLLECTION ITEMS TERMINATED BY ‘,’ MAP KEYS TERMINATED BY ‘:’;**

hive> describe MySchools;

OK  
schooltype string  
state string  
gender string  
TOTAL MAP<INT,INT>

hive> load data local inpath ‘/home/acadgild/Desktop/School\_Data.txt’ into table MySchools;

hive> **select \* from MySchools;**

SecondarySchool Assam Male {2015:56897,2016:575757,2017:585858}  
SecondarySchool Assam Female 2{015:19947,2016:20287,2017:33552}  
SecondarySchool Jharkhand Male {2015:456987,2016:415263,2017:362514}  
SecondarySchool Jharkhand Female {2015:12453,2016:52146,2017:63254}

hive> select **total[2016]** from MySchools where state=’Assam’;

575757  
20287

hive> select **total[2017]** from MySchools where state=’Jharkhand’ and gender=’Female’;

63254

**Struct**

Struct is a record type which encapsulates a set of named fields that can be any primitive data type. An element in STRUCT type can be accessed using the DOT (.) notation.

Yahama RZ Aircooled,149.0,14.0,0  
Hero Mestro Aircooled,155.0,14.8,0  
Honda Dio Fule-injection,223.0,20.25,0

hive> create table MyBikes(name string,   
**BikeFeatures** **struct<EngineType:string, cc:float,power:float,gears:int>** )   
row format delimited fields terminated by ‘\t’ **collection items terminated by ‘,’**;

hive> describe MyBikes;

name string  
BikeFeatures struct<EngineType:string,cc:float,power:float,gears:int>

hive> load data local inpath ‘/home/acadgild/Desktop/Bikes.txt’ into table MyBikes;

hive> **select \* from MyBikes;**

Yahama RZ {"EngineType":"Aircooled","cc":149.0,"power":14.0,"gears":0}  
Hero Mestro {"EngineType":"Aircooled","cc":155.0,"power":14.8,"gears":0}  
Honda Dio {"EngineType":"Fule-injection","cc":223.0,"power":20.25,"gears":0}

hive> select BikeFeatures.EngineType from MyBikes;

Aircooled  
Aircooled  
Fule-injection

hive> select **BikeFeatures.EngineType** from MyBikes where name=’Yahama RZ’;

Aircooled

**Load Data Statement**

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement. While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is **from local file system** and second is **from Hadoop file system**.

The syntax for load data is as follows:

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename   
[PARTITION (partcol1=val1, partcol2=val2 ...)]

**LOCAL** is identifier to specify the local path. It is optional.  
**OVERWRITE** is optional to overwrite the data in the table.  
**PARTITION** is optional.

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

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

The following query loads the given text into the table.

hive> **LOAD DATA LOCAL INPATH** '/home/user/sample.txt' **OVERWRITE INTO TABLE** employee;

// JDBC Execute Query Statement  
stmt.executeQuery("LOAD DATA LOCAL INPATH '/home/user/sample.txt'" + "OVERWRITE INTO TABLE employee;");

**Alter Table Statement** - It is used to alter or change a table attributes in Hive.

**The statement takes any of the following syntaxes based on what attributes we wish to modify in a table. (RENAME TO, ADD COLUMNS, DROP [COLUMN], REPLACE COLUMNS, CHANGE)**

1. ALTER TABLE table\_name **RENAME TO** new\_name

hive> **ALTER TABLE employee RENAME TO emp;**

stmt.executeQuery("ALTER TABLE employee RENAME TO emp;");

2. ALTER TABLE table\_name **ADD COLUMNS** (col\_spec[, col\_spec ...])

hive> ALTER TABLE employee ADD COLUMNS (dept STRING COMMENT 'Department name');

3. ALTER TABLE name **DROP [COLUMN]** column\_name

We cannot drop column directly from a table using command. The only way to drop column is using replace command. Let’s say, I have a table TEST with id, name and case column.

We want to drop id column of table TEST. So**, provide all those columns which you want to be the part of table in replace columns clause**

Hive> **ALTER TABLE TEST REPLACE COLUMNS( name string, case string);**

4. ALTER TABLE table\_name **REPLACE COLUMNS** (col\_spec[, col\_spec ...])  
hive> **ALTER TABLE employee REPLACE COLUMNS (eid INT empid Int, ename STRING name String);**

The above query deletes all the columns from the employee table and replaces it with empid and name columns:

5. ALTER TABLE table\_name **CHANGE** ***column\_name*** **new\_col\_name new\_col\_name\_datatype**

hive> ALTER TABLE employee CHANGE ***name*** **ename String**;  
hive> ALTER TABLE employee CHANGE ***salary*** **salary Double**;

**Drop Table Statement**   
Drops the table and all the data associated with it in the Hive metastore. When we drop a table from Hive Metastore, it removes the table/column data and their metadata. It can be a normal table (stored in Metastore) or an external table (stored in local file system); Hive treats both in the same manner, irrespective of their types.

**DROP TABLE [IF EXISTS] table\_name;**

Query drops a table named employee:  
hive> DROP TABLE IF EXISTS employee;

// JDBC execute statement  
stmt.executeQuery("DROP TABLE IF EXISTS employee;");

**Hive - Partitioning and bucketing**

**What is Partitions?**

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

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

It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion of the data.

**Tables or partitions are sub-divided into buckets**, to provide extra structure to the data that may be used for more efficient querying. Bucketing works based on the value of **hash function** of some column of a table.

For example, a table named Tab1 contains employee data such as id, name, dept, and yoj (i.e., year of joining). Suppose you need to retrieve the details of all employees who joined in 2012. A query searches the whole table for the required information. However, if you partition the employee data with the year and store it in a separate file, it reduces the query processing time. The following example shows how to partition a file and its data

id, name, dept, yoj  
1, gopal, TP, 2012  
2, kiran, HR, 2012  
3, kaleel,SC, 2013  
4, Prasanth, SC, 2013

The above data is partitioned into two files using year.

/tab1/employeedata/2012/file2  
1, gopal, TP, 2012  
2, kiran, HR, 2012

/tab1/employeedata/2013/file3  
3, kaleel,SC, 2013  
4, Prasanth, SC, 2013

**Creating partition**

hive> CREATE TABLE STD(SNAME STRING, SID INT) **PARTITIONED BY (YOP INT)** ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

Note: When we are creating partitions then the fileds used for creating partition is not the part of table fileds. here YOP is partition fileds is not used in STD(SNAME STRING, SID INT) table fileds.

**Adding a Partition**

We can add partitions to a table by altering the table. Let us assume we have a table called employee with fields such as Id, Name, Salary, Designation, Dept, and yoj.

Syntax:

**ALTER TABLE table\_name ADD [IF NOT EXISTS] PARTITION partition\_spec [LOCATION 'location1'] partition\_spec [LOCATION 'location2'] ...;**

**PARTITION\_SPEC: (p\_column = p\_col\_value, p\_column = p\_col\_value, ...)**

The following query is used to add a partition to the employee table.

hive> **ALTER TABLE employee ADD PARTITION (year=’2012’) location '/2012/part2012';**

**Renaming a Partition**

The syntax of this command is as follows.

ALTER TABLE table\_name PARTITION partition\_spec RENAME TO PARTITION partition\_spec;

The following query is used to rename a partition:

hive> **ALTER TABLE employee PARTITION (year=’1203’) RENAME TO PARTITION (Yoj=’1203’);**

**Dropping a Partition**

The following syntax is used to drop a partition:

ALTER TABLE table\_name DROP [IF EXISTS] PARTITION partition\_spec, PARTITION partition\_spec,...;

The following query is used to drop a partition:

hive> **ALTER TABLE employee DROP [IF EXISTS] PARTITION (year=’1203’);**

**Types of partitions:**

Hive supports two types of partitions:  
1. Static Partition 2. Dynamic partition

**1. Static Partition**

In case, of static partitions, We need to have separate files for each partitions. While creating tables, partition columns should be mentioned at the end in partitioned by clause. Every time we load, we need to specify the partition value and file name.

hive> **CREATE TABLE STD (SNAME STRING, SID INT) PARTITIONED BY (YOP INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;**

hive> **LOAD DATA LOCAL INPATH '/HOME/TRAINING/DVS/STD\_2011.CSV' INTO TABLE STD PARTITION (YOP='2011');**

hive> **LOAD DATA LOCAL INPATH '/HOME/TRAINING/DVS/STD\_2012.CSV' INTO TABLE STD PARTITION (YOP='2012');**

hive> **LOAD DATA LOCAL INPATH '/HOME/TRAINING/DVS/STD\_2013.CSV' INTO TABLE STD PARTITION (YOP='2013');**

**Drawback with static partitions:**

1. If no. of files are more, than no. of load commands we need to write more.

2. User should be very careful while specifying the partition values and file name. Else wrong data will be loaded to the partitions

3. IF we have mix all partitions data in one file then static partitions will not work.

**2. Dynamic partition**

Steps to creating dynamic partitions table:

1. Create a Temporary (non-partitioned) table.  
2. Load data to Temporary table.  
3. Create partition table.  
4. Enable dynamic partitions using   
 **hive.exec.dynamic.partition = true** and **hive.exec.dynamic.partition.mode = strict**  
5. Insert data to partition table from temporary table.  
6. Drop temporary table.

We need to set the following properties For Dynamic Partitions:   
**hive.exec.dynamic.partition=true  
hive.exec.dynamic.partition.mode=strict**

1. hive> CREATE TABLE STUD\_TEMP(SNAME STRING, SID INT, YEAR INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

2. hive> LOAD DATA LOCAL INPATH '/HOME/TRAINING/DVS/STD/STD\_ALL.CSV' INTO TABLE STUD\_TEMP;

3. hive> CREATE TABLE STUD\_DP(SNAME STRING, SID INT) PARTITIONED BY (YEAR INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

4. hive> hive.exec.dynamic.partition=true hive> set hive.exec.dynamic.partition.mode=strict;

5. hive> **INSERT INTO TABLE STUD\_DP PARTITION(YEAR) SELECT \* FROM STUD\_TEMP;**

6. hive> DROP TABLE STUD\_TEMP;

**Note**: If there are any null values for year column, there will be a default hive partition created in warehouse directory under the table directory.

**Nested Partitions:**

CREATE TABLE STD\_TEMP2(SNAME STRING, SID INT, YEAR INT, MONTH INT, DAY INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

LOAD DATA LOCAL INPATH '/HOME/TRAINING/DVS/STD/STD\_ALL.CSV' INTO TABLE STUD\_TEMP2;

CREATE TABLE STUD\_DP2(SNAME STRING, SID INT) PARTITIONED BY (YEAR INT, MONTH INT, DAY INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;

hive> INSERT INTO TABLE STUD\_DP2 PARTITION(YEAR, MONTH, DAY) SELECT \* FROM STUD\_TEMP2;

**???** What will happen if partition columns are not in sequence:

\* Create temporary table with same column sequence as input file.

\* Load data in temporary table.

\* Create partition table in the sequence that we wanted like year, month and day.

\* Insert data from temp table with same column sequence as partition table column sequence like

**CREATE TABLE STD\_TEMP3(SNAME STRING, SID INT, DAY INT, MONTH INT, YEAR INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;**

LOAD DATA LOCAL INPATH '/HOME/TRAINING/DVS/STD/STUD.TXT' INTO TABLE STUD\_TEMP3;

**CREATE TABLE STUD\_DP3(SNAME STRING, SID INT) PARTITIONED BY (YEAR INT, MONTH INT, DAY INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;**

INSERT INTO TABLE STUD\_DP3 PARTITION(YEAR, MONTH, DAY) SELECT SNAME, SID, YEAR,MOTH,DAY FROM STUD\_TEMP3;

**Partitions with External Table (With Location)**

CREATE EXTERNAL TABLE STUD\_ET3 (SNAME STRING, SID INT) PARTITIONED BY(YEAR INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/USER/TRAINING/DVS\_HDFS/STUD' ;

**Exchanging partitions:**

Exchange partitions of one table to another table ( Both table must have the same structure)

HIVE> ALTER TABLE STD1 EXCHANGE PARTITION(YOP=2015) WITH TABLE STD ;

HIVE> ALTER TABLE STD1 EXCHANGE PARTITION(YOP=2016) WITH TABLE STD ;

**Repairing or Recovering partitions:**

hive> msck repair table std\_dp;

Note: This feature add from hive version hive-0.14.0. It may not work in lower version of hive.

**Enable No Drop/OFF Line on Partition:**

ALTER TABLE STUD PARTITION(YEAR=2015) ENABLE NO\_DROP;

ENABLE NO\_DROP - Will not allow you to drop this partition

ALTER TABLE STUD PARTITION(YEAR=2015) DISABLE NO\_DROP;

DISABLE NO\_DROP - Will allow you to drop this partition

ALTER TABLE STUD PARTITION(YEAR=2015) ENABLE OFFLINE;

ENABLE OFFLINE - Hide the partition

ALTER TABLE STUD PARTITION(YEAR=2015) DISABLE OFFLINE;

DISABLE OFFLINE - Un-hide the partition

**Bucketing**

Buckets in hive is used in segregating of hive table-data into multiple files or directories. it is used for efficient querying.

😊 The data i.e. present in that partitions can be divided further into Buckets.

😊 The division is performed based on Hash of particular columns that we selected in the table.

😊 Buckets use some form of Hashing algorithm at back end to read each record and place it into buckets

Note: In Hive, we have to enable buckets by using the **set hive.enforce.bucketing=true**;

Bucketing is also one more way of decomposing a big data set into small and manageable data sets like partirioning.

**Bucketing is mainly used for 2 reasons**:

1. For faster joins (Join optimizatioin) 2. Table sampling

**Creating a bucked table**

hive> create table emp\_bk1(empno int, ename string, job string, sal float, comm float, DEPTNO int) CLUSTERED BY(DEPTNO) INTO 3 BUCKETS row format delimited fields terminated by ',' ;

**Steps involved in creating a bucketd tables**

1. Create a tempprary table.  
2. Load data in tempprary table.  
3. Create Bucketed Table.  
4. Enable bucketing by using set hive.enforce.bucketing=true  
5. Insert data into bucked table from tempprary table.  
6. Drop tempprary table.

**Enable bucketing:**

hive> set hive.enforce.bucketing=true

**Insert data into bucked table from tempprary table/table**.

hive> insert into table emp\_bk1 select \* from emp\_et\_wl;

😊 Note: It creates index based on the buckeded column for faster lookups (Joins)

**Getting bucket wise sampling**

hive> select \* from emp\_bk1 tablesample(1 out of 3 on deptno);

hive> select \* from emp\_bk1 tablesample( bucket 1 out of 3 on deptno);

hive> select \* from emp\_bk1 tablesample(3 out of 3 on deptno);

**Getting percentage wise sampling**

hive> select \* from emp\_bk1 tablesample(10 percent);

**Hive - View and Indexes**

Views are generated based on user requirements. You can save any result set data as a view. The usage of view in Hive is same as that of the view in SQL. It is a standard RDBMS concept. We can execute all DML operations on a view.

**Creating a View**

You can create a view at the time of executing a SELECT statement. The syntax is as follows:

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

[COMMENT table\_comment] AS SELECT ...

Assume employee table as given below, with the fields Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000. We store the result in a view named emp\_30000.

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

| ID | Name | Salary | Designation | Dept |

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

|1201 | Gopal | 45000 | Technical manager | TP |  
|1202 | Manisha | 45000 | Proofreader | PR |  
|1203 | Masthanvali | 40000 | Technical writer | TP |  
|1204 | Krian | 40000 | Hr Admin | HR |  
|1205 | Kranthi | 30000 | Op Admin | Admin |  
+------+--------------+-------------+-------------------+--------+

The following query retrieves the employee details using the above scenario:

hive> CREATE VIEW emp\_30000 AS SELECT \* FROM employee WHERE salary>30000;

**Dropping a View**

Use the following syntax to drop a view:

**DROP VIEW view\_name**

The following query drops a view named as emp\_30000:

hive> DROP VIEW emp\_30000;

**Creating an Index**

An Index is nothing but a pointer on a particular column of a table. Creating an index means creating a pointer on a particular column of a table. Its syntax is as follows:

CREATE INDEX index\_name ON TABLE base\_table\_name (col\_name, ...)

AS **'index.handler.class.name'** [WITH DEFERRED REBUILD]

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

[IN TABLE index\_table\_name]

[PARTITIONED BY (col\_name, ...)]

[

[ ROW FORMAT ...] STORED AS ...

| STORED BY ...

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

Example: Let us take an example for index. Use the same employee table that we have used earlier with the fields Id, Name, Salary, Designation, and Dept. Create an index named index\_salary on the salary column of the employee table.

The following query creates an index:

hive> **CREATE INDEX inedx\_salary ON TABLE employee(salary) AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';**

It is a pointer to the salary column. If the column is modified, the changes are stored using an index value.

**Dropping an Index**

The following syntax is used to drop an index:

DROP INDEX <index\_name> ON <table\_name>

The following query drops an index named index\_salary:

hive> DROP INDEX index\_salary ON employee;

**FILE FORMATS**

File formats are mainly used foe dast retrieval, faster writing and compression. Below is some of file formats:

TEXTFILE

RC

PARQUET

ORC and many more

hive> create table emp\_text(empno int, ename string, sal float, comm float, deptno int) row format delimited fields terminated by ',' **STORED AS textfile**;

hive> load data local inpath '/home/training/dvs/emp.csv' into table emp\_text;

hive> create table emp\_rc(empno int, ename string, sal float, comm float, deptno int) row format delimited fields terminated by ',' **stored as rcfile**;

OR

hive> create table emp\_rc **stored as rcfile** as select \* from emp\_text;

hive> insert into table emp\_rc select \* from emp\_text;

😊 Note: Try with other formats wiht some more data so that you can feel the difference the file formats. You can try all the fole formats wiht latest version of cloudera.

hive> create table emp\_orc(empno int, ename string, sal float, comm float, deptno int) row format delimited fields terminated by ',' **stored as orcfile**;

hive> create table emp\_pr(empno int, ename string, sal float, comm float, deptno int) row format delimited fields terminated by ',' **stored as parquetfile**;

hive> create table emp\_avro(empno int, ename string, sal float, comm float, deptno int) row format delimited fields terminated by ',' **stored as avrofile**;

Goto to warehouse directory and see the file formats which may not be user understandable/readable format.

**USE CASES:**

a. If your data is delimited by some parameters then you can use TESTFILE format.

b. If your data is in small files whose size is less than the block size then you can use SEQUENCEFILE format.

c. If you want to perform analytics on your data and you want to store your data efficiently for that then you can use RCFILE format.

d. If you want to store your data in an optimized way which lessens your storage and increase your performance then you use ORCFILE format.

**MISCELIANEOUS**

Create a temporary table (Supports from hive-0.14.0 onwards)

hive> create TEMPORARY table std\_temp(sname string, sid int, yop int) row format delimited fields terminated by ',';

The above table will have existence till the end of the session and will be deleted once we come out of the session

**ENABLING AUTO PURGE:**

If trash is enabled then the data will go to trash, in case of auto purge the data will not go to trash and will be permanently deleted.

hive> set TBLPROPERTIES("auto.purse" = "true")

**SKIPPING 'N' LINES OF DATA WHILE LOADING THE DATA**

hive> create table emp3(empno int, ename string, sal double, comm float, deptno int) row format delimited fields terminated by ',' TBLPROPERTIES('skip.header.line.count'='1','Createor'='Shesh', 'Date'='2020-01-22',auto.purge=true) ;

By using skip.header.line.count option, We use skip header of the file whiling loading data to the table if the file is comming with column headers.

**CREATING DATABASE WITH DB PRPERTIES**

hive> create database kcdb comment 'SHESH DB' with DBPROPERTIES('Createor'='Minkesh', 'Date'='2020-01-22');

**CHECKING DB PROPERTIES:**

Create database shesh location '/user/training/warehouse/shesh.db' ;

hive> describe database extended kcdb;

**CREATING TABLE WITH TABLE PRPERTIES**

hive> create table emp(empno int, ename string) row format delimited fields terminated by ',' TBLPROPERTIES('Createor'='Shesh', 'Date'='2020-01-22') ;

**CHECKING TABLE PROPERTIES**

hive> describe formatted emp;

**SWITCH A TABLE FROM INTERNAL/MANAGED TO EXTERNAL**

hive> alter table table\_name SET TBLPROPERTIES('EXTERNAL' = 'TRUE');

**SWITCH A TABLE FROM EXTERNAL TO INTERNAL/MANAGED**

hive> alter table table\_name SET TBLPROPERTIES('EXTERNAL' = 'FALSE');

**CHANGEING FROM ONE DATA TYPE TO ANOTHER DATATYPE**

hive> select case(sal as int) from emp;

**IMPORTING STRUCTURE OF ANOTHER TABLE IN HIVE**

hive> create table emp3 as select \* from emp1 where 1=2;

hive> create table t2 like emp;

**IMPORTING STRUCTURE TABLE AND DATA OF ANOTHER TABLE IN HIVE**

hive> create table emp3 as select \* from emp1;

**TRUNCATING DATA OF HIVE TABLE WITHOUT USING TRUNCATE COMMAND**

hive> insert overwrite table emp1 select \* from emp where 1=2;

hive> insert into table emp1 select \* from emp;

**INSERTING DATA IN TO A HIVE TABLE FORM ANOTHER HIVE TABLE**

hive> insert into table empl2 select \* from emp1;

**WHAT HAPPEN IF WAREHOUSE PATH IS CHANGED?**

The default location of all the table and databses will be changed and existing databases and tables will still reside in the same location.

**RESTRICTING FULL TABLE SCAN ON PARTITIONED TABLES:**

set hive.mapred.mode=strict;

This property does not allow to query partitioned tables without where condition. This prevents very large job running for long time.  
Hadoop job -list;  
hadoop job -kill job\_id

**What is the significance of "IF EXISTS" clause while dropping a table?**

When we issue the command DROP TABLE TABLE\_NAME hive throws an error if the table being dropped does not exist. So to avoid we use "IF EXISTS" clause. i.e. DROP TABLE IF EXISTS TABLE\_NAME

**Create hive table:**

**drop table hive\_table;**

CREATE TABLE hive\_table( empno INT, ename string, designation string, sal Double, manager INT, deptno INT)ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LINES TERMINATED BY '\n' STORED AS TEXTFILE;

**load data to hive table:**

LOAD DATA LOCAL INPATH '/home/training/dvs/employee\_hive\_hbase.csv' INTO TABLE hive\_table;

**Create HBase-hive Mapping Table**

CREATE TABLE hbase\_table\_employee(empno INT, ename STRING, designation STRING, sal Double, manager INT, deptno INT ) **STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'   
WITH SERDEPROPERTIES** ("**hbase.columns.mapping" = ":key, cf:ename, cf:designation, cf:sal, cf:manager, cf:deptno**") **TBLPROPERTIES** **("hbase.table.name" = "employee\_hbase");**

INSERT INTO TABLE hbase\_table\_employee SELECT \* FROM hive\_table;

LOAD DATA LOCAL INPATH '/home/training/dvs/employee\_hive\_hbase.csv' INTO TABLE hbase\_table\_employee;

**SET Commands**

hive (default)> set <Hit EnterKey>

**To Print current db with Hive Cli propmpt**set hive.cli.print.current.db=true;

**RESTRICTING FULL TABLE SCAN ON PARTITIONED TABLES**

set hive.mapred.mode=strict;

**To enable bucketing**

set hive.enforce.bucketing=true

**For Dynamic Partitions**

hive.exec.dynamic.partition=true  
hive.exec.dynamic.partition.mode=strict  
hive.exec.max.dynamic.partitions=1000  
hive.exec.max.dynamic.partitions.pernode=100