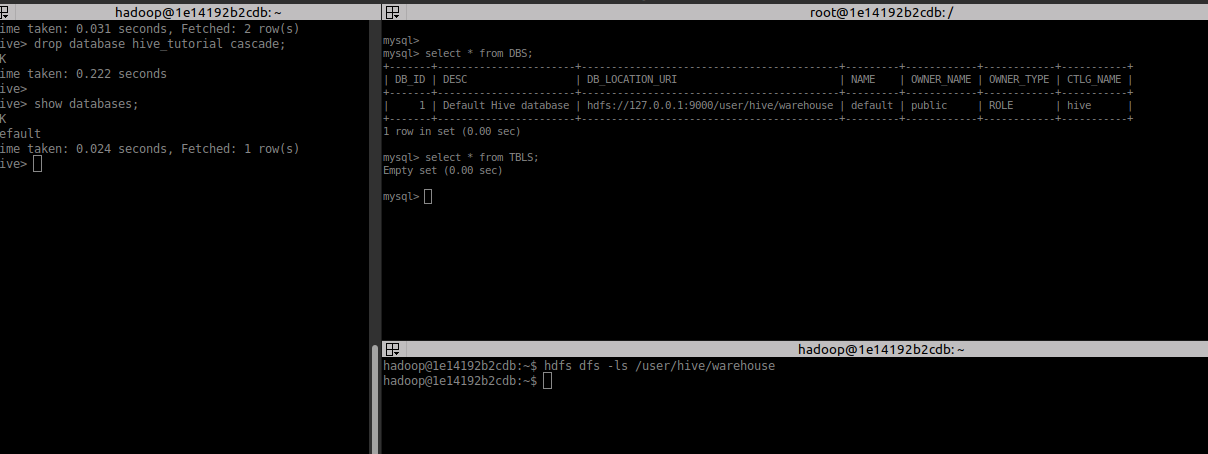
# Hive DDL Commands

<http://hadooptutorial.info/hive-database-commands/>

When a new DB and hdfs are created the data in mysql metastore and hdfs are empty



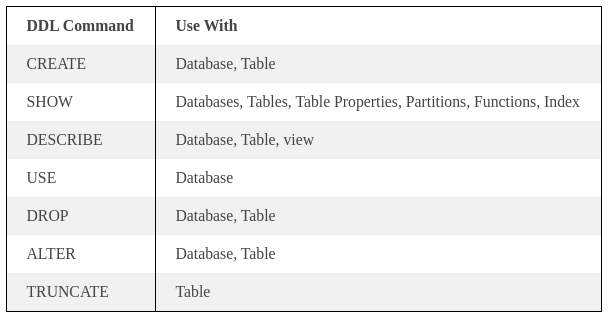
Hive DDL commands are the statements used for defining and changing the structure of a table or database in Hive. It is used to build or modify the tables and other objects in the database.

The several types of Hive DDL commands are:

1. CREATE
2. SHOW
3. DESCRIBE
4. USE
5. DROP
6. ALTER
7. TRUNCATE

Before moving forward, note that the Hive commands are ****case-insensitive****.

CREATE DATABASE is the same as create database.



## DDL Commands On Databases in Hive

### 1. CREATE DATABASE in Hive

The ****CREATE DATABASE**** statement is used to create a database in the Hive. The DATABASE and SCHEMA are interchangeable. We can use either DATABASE or SCHEMA.

****Syntax:****

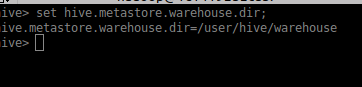
**CREATE** (DATABASE|SCHEMA) [IF NOT EXISTS] database\_name

[COMMENT database\_comment]

[LOCATION hdfs\_path]

[WITH **DBPROPERTIES** (property\_name=property\_value, ...)];

****Description of Arguments:****

* ****IF NOT EXISTS**** – This optional but recommended to use, so that, if a database with same name already exists, then it will not try to create it again and will not show any error message.
* ****COMMENT**** – It is also optional. It can be used for providing short description or any comment to database
* ****LOCATION**** – It is also optional. By default all the hive databases will be created under default warehouse directory (set by the property ****hive.metastore.warehouse.dir****) as ****/user/hive/warehouse/*****database\_name.db* . But if we want to specify our own location then this option can be specified.
* 
* ****DBPROPERTIES**** – Optional but used to specify any properties of database in the form of (key, value) separated pairs.

****Example:****

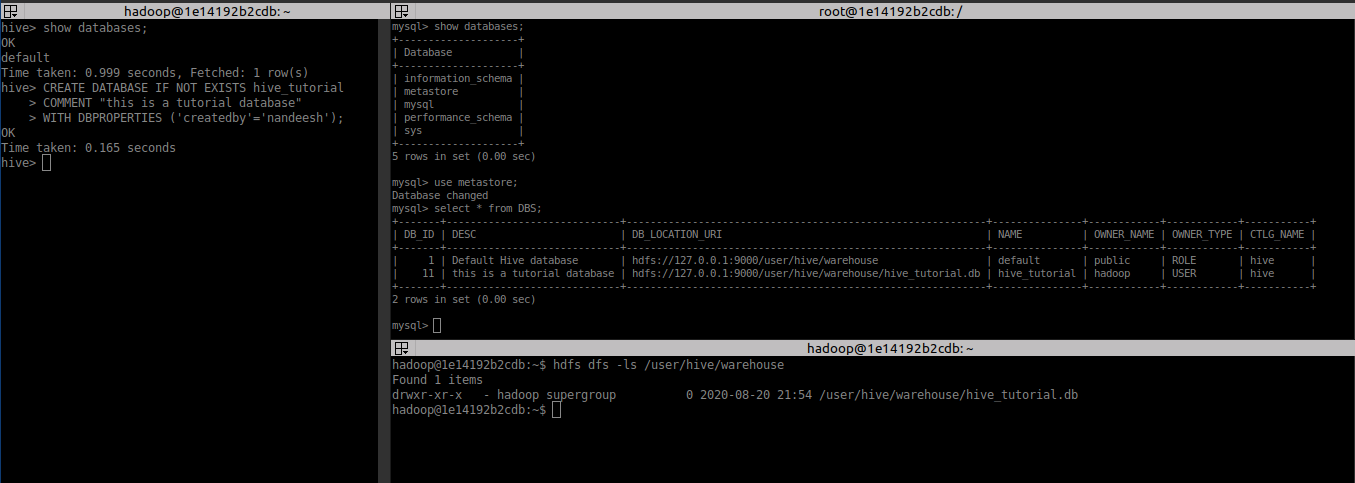
CREATE DATABASE IF NOT EXISTS hive\_tutorial

COMMENT "this is a tutorial database"

LOCATION '/user/hive/warehouse/hive\_tutorial'

WITH DBPROPERTIES ('createdby'='nandeesh');

The databse is created in the spedified hdfs location

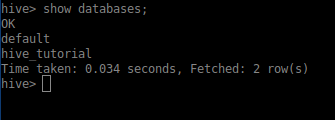


### 2. SHOW DATABASE in Hive

The****SHOW DATABASES**** statement lists all the databases present in the Hive.

****Syntax:****

**SHOW** (DATABASES|SCHEMAS) [**LIKE** identifier\_with\_wildcards];

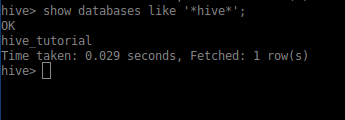


By default, SHOW DATABASES lists all of the databases defined in the metastore.

* ****LIKE**** – It is optional. But it allows us to filter the database names using a regular expression.

Wild cards in the regular expression can only be ****”**** (single quotes) for any character(s) or ****‘|’**** for a choice.

Examples are ****’employees’, ’emp’, ‘emp\*|\*ees‘****,  (emp\* or \*ees), all of which will match the database named ’employees’.



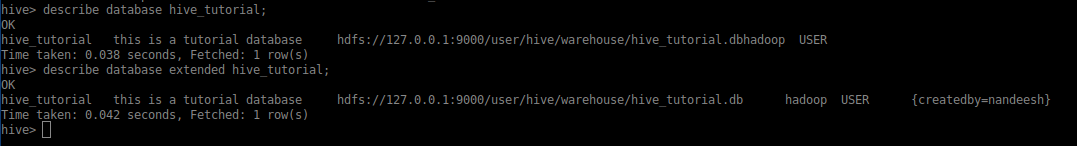
### 3. DESCRIBE DATABASE in Hive

The ****DESCRIBE DATABASE**** statement in Hive shows the name of Database in Hive, its comment (if set), and its location on the file system.

The ****EXTENDED**** can be used to get the database properties.

****Syntax:****

DESCRIBE DATABASE/SCHEMA [EXTENDED] db\_name;

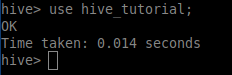


### 4. USE DATABASE in Hive

The ****USE**** statement in Hive is used to select the specific database for a session on which all subsequent HiveQL statements would be executed.

****Syntax:****

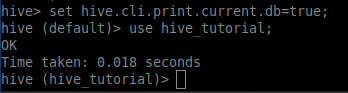
USE database\_name;



****Note****

To know the current database under which we are working we need to the below property in Hive shell. There is no command to show the current database, but this property will keep printing the current database name as suffix in the command line prompt as shown in below examples.

set hive.cli.print.current.db=true;



Updating the conf property via set hive.cli.print.current.db=true will show the current DB for current hive session only, but if we update the .hiverc file with the above property will keep showing the current db for all the sessions.

### **5. ALTER DATABASE in Hive**

The ****ALTER DATABASE**** statement in Hive is used to change the metadata associated with the database in Hive.

We can

Assign any new (key, value) pairs into DBPROPERTIES

Set owner user or role to the Database

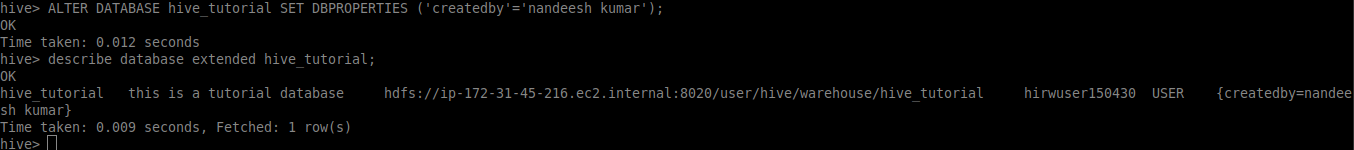
But below are its limitations

We can’t unset any property using Alter command

No other meta-data about the database can be changed, including its name and directory location.

****Syntax for changing Database Properties:****

**ALTER** (DATABASE|SCHEMA) database\_name SET **DBPROPERTIES** (property\_name=property\_value, ...);



****Syntax for changing Database owner:****

**ALTER** (DATABASE|SCHEMA) database\_name SET OWNER [USER|ROLE] user\_or\_role;

****Syntax for changing Database Location:****

**ALTER** (DATABASE|SCHEMA) database\_name SET LOCATION hdfs\_path;

****Note:**** The ALTER DATABASE … SET LOCATION statement does not move the database current directory contents to the newly specified location. This statement does not change the locations associated with any tables or partitions under the specified database. Instead, it changes the default parent-directory, where new tables will be added for this database.

No other metadata associated with the database can be changed.

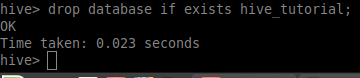
### 6. DROP DATABASE in Hive

The ****DROP DATABASE**** statement in Hive is used to Drop (delete) the database.

The default behavior is RESTRICT which means that the database is dropped only when it is empty. To drop the database with tables, we can use CASCADE.

****Syntax:****

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



hive error message *” InvalidOperationException(message:Database xxxx is not empty. One or more tables exist.)”* when trying to delete ****xxxx**** which has one ****yyyy**** with no CASCADE argument.

## **Introduction to Hive Tables**

In Hive, Tables are nothing but collection of homogeneous data records which have same schema for all the records in the collection.

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

Hive metadata is stored in hive metastore configured via any RDBMS (default is Derby but can be configured to any of these: MySQL, PostGreSQL, Oracle, MS SQL Server, etc…). So, **Hive metadata is not stored on HDFS**.

Why?

The storage system for the metastore should be optimized for online transactions with random accesses and updates. A file system like HDFS is not suited since it is optimized for sequential scans and not for random access. So, the metastore uses either a traditional relational database (like MySQL, Oracle) or file system (like local, NFS, AFS) and not HDFS. As a result, HiveQL statements which only access metadata objects are executed with very low latency. However, Hive has to explicitly maintain consistency between metadata and data.

Hive table data can be stored local filesystem as well, when running in local mode.

### Hive Table Types :

Hive supports two types of tables.

#### Managed Tables – Default table type in Hive

Tables data is manged by Hive by moving data into its warehouse directory configured by hive.metastore.warehouse.dir (by default /user/hive/warehouse).

If this table is dropped both data and metadata (schema) are deleted. I.e. these tables are owned by Hive.

Less convenient to share with other tools like Pig, HBase etc, as these are maintained by Hive and data can be deleted without informing these tools.

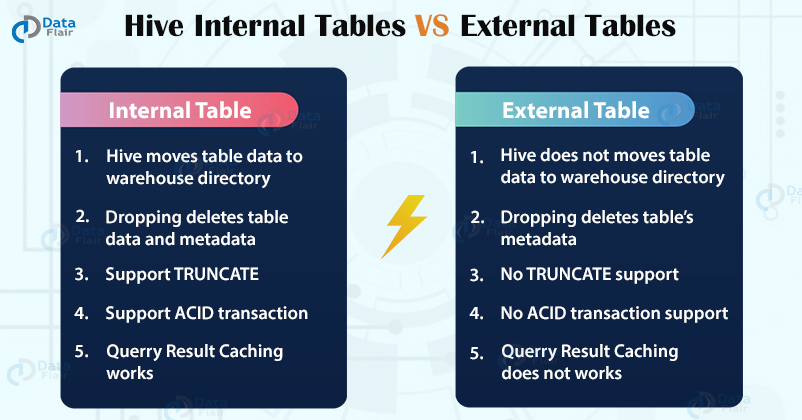
#### External Tables

These tables are not managed or owned by Hive. And tables data will not be copied into hive warehouse directory but maintained at external location

If these tables are dropped only the schema from metastore will be deleted but not the data files from external location.

Provides convenience to share the tables data with other tools 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

By the name itself, these are temporary and available till end of current session only.

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 doesn’t support Partitioning & Indexing.

****Warning:****

Be careful in naming temporary tables, As Hive doesn’t warn or error out if we use a name that already exist in the database.

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

These are introduced for first time in Hive-0.14.0 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.

Comparison With  Partitioned Tables and Skewed Tables

Skewing technique similar to Partitioning but it is recommended when only few values are occurring very often in input.

Like, 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.

One of the main disadvantage of Partitioning is that HDFS Scalability will be an issue more partitioning is done. ****For example****, if there are 1000 mappers and 1000 partitions, and each mapper gets at least 1 row for each key, we will end up in creating 1 million intermediate files, So Namenode’s memory will be in trouble to store metadata about all these files.

## DDL Commands on Tables in Hive

<http://hadooptutorial.info/hive-table-creation-commands/>

### 1. CREATE TABLE

The ****CREATE TABLE**** statement in Hive is used to [create](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL" \l "LanguageManualDDL-CreateTable) a table with the given name. If a table or view already exists with the same name, then the error is thrown. We can use ****IF NOT EXISTS**** to skip the error

****Syntax:****

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

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

      [**COMMENT** table\_comment]

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

      [CLUSTERED **BY** (col\_name, ...) [SORTED **BY** (col\_name [**ASC**|**DESC**], ...)] **INTO** num\_buckets BUCKETS]

      [SKEWED **BY** (col\_name, ...) **ON** ([(col\_value, ...), ...|col\_value, ...])

             [STORED **AS** DIRECTORIES] ]

      [

         [**ROW** **FORMAT** **row\_format**]

         [STORED **AS** file\_format]

         | STORED **BY** 'storage.handler.class.name' [**WITH** SERDEPROPERTIES (...)]

      ]

      [LOCATION hdfs\_path]

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

      [**AS** select\_statement];

Below  are the detailed descriptions of each option/argument to CREATE TABLE command.

****TEMPORARY**** – Specified for creation of temporary tables

****EXTERNAL**** – Specified only for external tables

****IF NOT EXISTS**** – Suppresses error messages when a table already exists with same name and ignores creation of table again even if there is a schema difference between existing table and new table.

****db\_name**** – This is also optional but can be used to specify the table under a particular target database, if we are not already working under it.

****COMMENT**** – Similar to CREATE DATABASE statement comments, we can add comments to table as well as to columns (strings within single quotes) to provide descriptive information to users.

****PARTITIONED BY**** – This clause is useful to partition the tables based on particular columns.

****CLUSTERED BY**** – This clause is used to provide more structure to tables and partitions.

****SKEWED BY**** – This clause is useful to create skewed tables. Further details can be discussed during example for this clause.

Below is an example of syntax for a table with two skewed columns:

**CREATE** **TABLE** table\_name (col1 **STRING**, col2 **int**, col3 **STRING**)

  SKEWED **BY** (col1, col2) **ON** (('s1',1), ('s3',3), ('s10',20), ('s20',20)) [STORED **AS** DIRECTORIES];

****ROW FORMAT**** – This clause is used to specify the format of each row in the input data. If data fields are delimited by certain characters we can use ****DELIMITED**** sub-clause or we need to provide a ****SERDE**** that can serialize or deserialize the input data records.

Below are the default values/Syntax for DELIMITED clause.

**ROW** **FORMAT** DELIMITED

**FIELDS** **TERMINATED BY** '\001'

    COLLECTION ITEMS **TERMINATED BY** '\002'

    MAP **KEYS** **TERMINATED BY** '\003'

**LINES** **TERMINATED BY** '\n'

I.e. default field delimiter is ****Ctrl+A**** (octal representation – ****‘\001’****) (also represented as ****^A****), collection delimiter is ****Ctrl+B**** (****‘\002’**** or ****^B****) and map keys are terminated by ****Ctrl+C**** (****‘\003’**** or ****^C****) and lines terminator is new line feed ****‘\n’.****

****Syntax for SERDE clause:****

**ROW** **FORMAT** SERDE serde\_name

           [**WITH** SERDEPROPERTIES (prop\_name=prop\_value, ...)]

****STORED AS**** –  Storage file format can be specified in this clause. Below are the available file formats for hive table creation.

SEQUENCEFILE

          TEXTFILE

          RCFILE

          PARQUET

          ORC

          AVRO

          INPUTFORMAT input\_format\_classname OUTPUTFORMAT output\_format\_classname

****STORED BY class\_name [WITH SERDEPROPERTIES (…)]****

It is an alternative to above two clauses (ROW FORMAT & STORED AS) to provide custom row format handler class\_name and custom serde properties.

****LOCATION**** – Directory location for table data will be specified under this clause.

****TBLPROPERTIES****–Metadata key/value pairs can be tagged to the table. ****last\_modified\_user**** and ****last\_modified\_time**** properties are automatically added under table properties and managed by Hive. Some example predefined table properties are,

TBLPROPERTIES ("comment"="table\_comment")

TBLPROPERTIES ("hbase.table.name"="table\_name") //**for** hbase integration

TBLPROPERTIES ("immutable"="true") **or** ("immutable"="false")

TBLPROPERTIES ("orc.compress"="ZLIB") **or** ("orc.compress"="SNAPPY") **or** ("orc.compress"="NONE")

TBLPROPERTIES ("transactional"="true") **or** ("transactional"="false") **default** **is** "false"

TBLPROPERTIES ("NO\_AUTO\_COMPACTION"="true") **or** ("NO\_AUTO\_COMPACTION"="false"), the **default** **is** "fa

****AS select\_statement**** – AS clause is used to create table similar to the schema of the select\_statement (another query statement) and populated with the output records of select\_statement. It is also know as ****CTAS**** (Create Table AS) clause.

****Create Table Like****

Combination of LIKE clause with CREATE TABLE can be used to create tables similar to another existing tables or views. Its syntax is as follows.

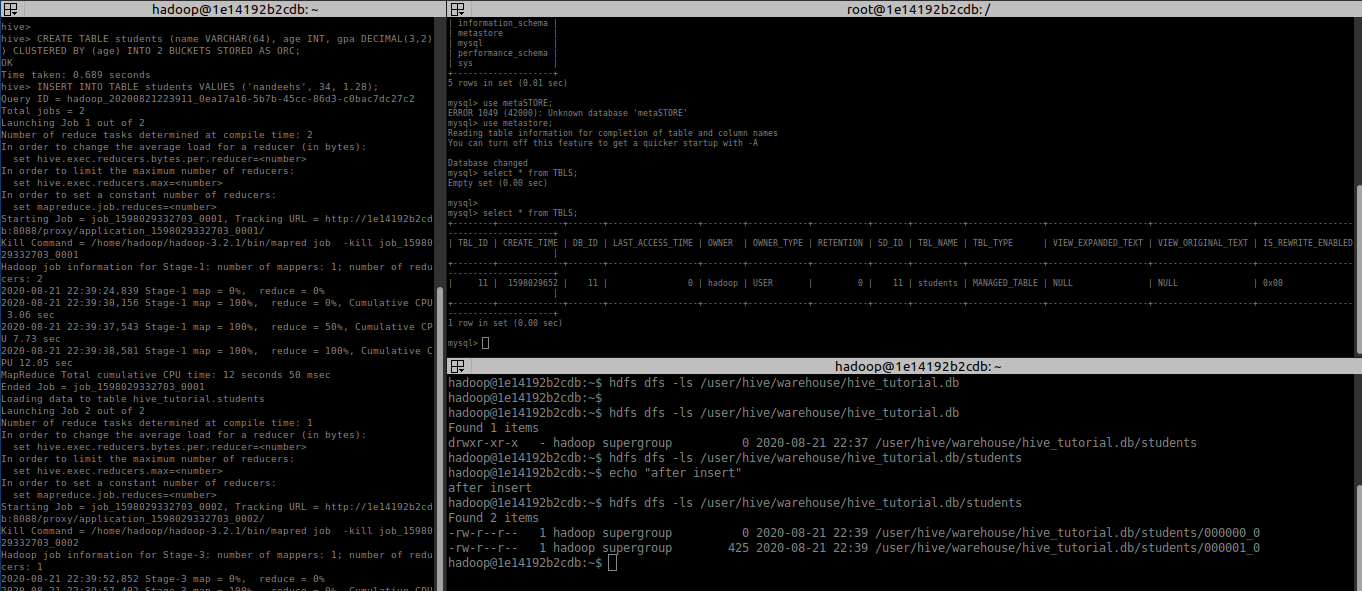
**CREATE** [**TEMPORARY**] [EXTERNAL] **TABLE** [**IF NOT EXISTS**] [db\_name.]table\_name

**LIKE** existing\_table\_or\_view\_name

       [LOCATION hdfs\_path];

Unlike ****CTAS**** clause, Only the schema is copied from existing table or view but not the records.

Case 1: create table without location

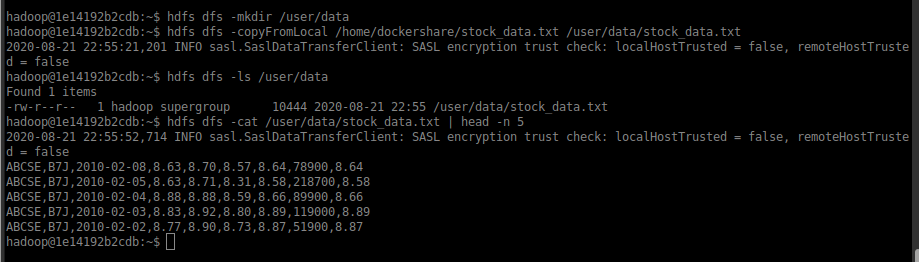


Metastore entry made in table TBLS

Folder with same table name created in hive default hdfs path

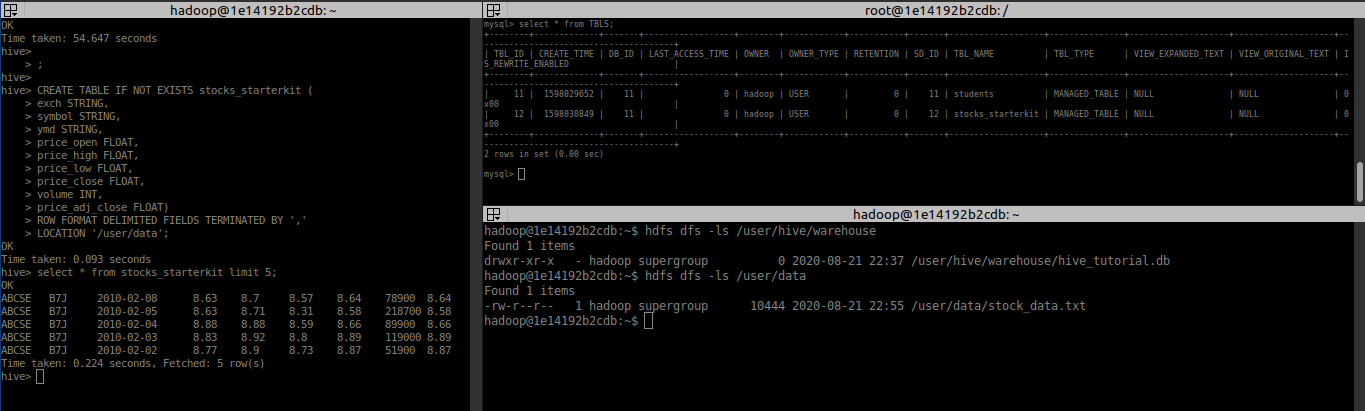
After record inserted to table, map reduce program is executed and data file created under newly created folder in hdfs.

Case 2: create table with existing data using location



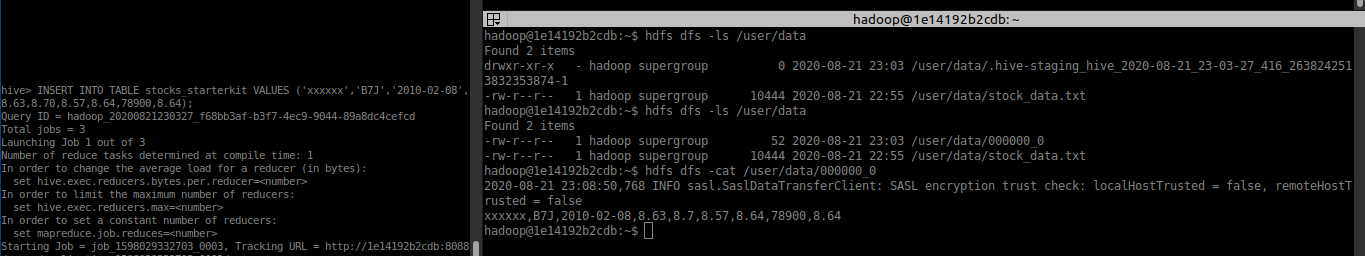
Copy the data from local to hdfs location

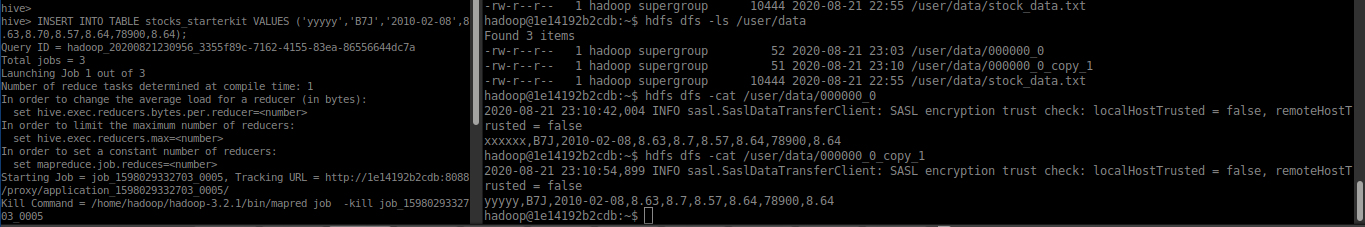
Create a table with existing file in HDFS

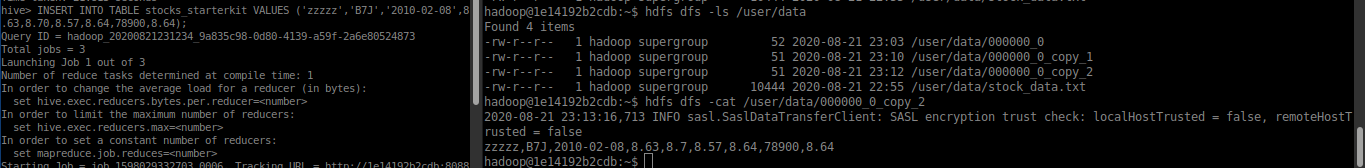


No any new directory is created in default location, hive read the file as table and process it.

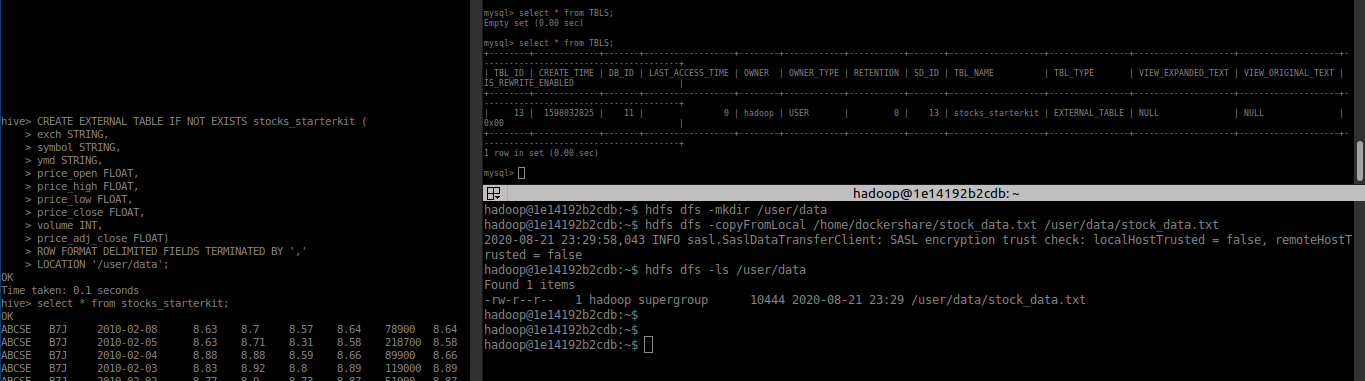
When a new record is inserted hive creates a new file for each record







Case 3:

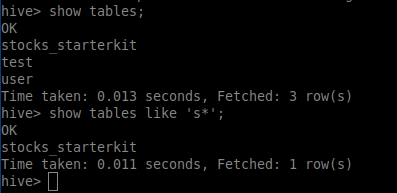


### 2. SHOW TABLES in Hive

The ****SHOW TABLES**** statement in Hive lists all the base tables and **[views](https://data-flair.training/blogs/hive-view-hive-index/)** in the current database.

****Syntax:****

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

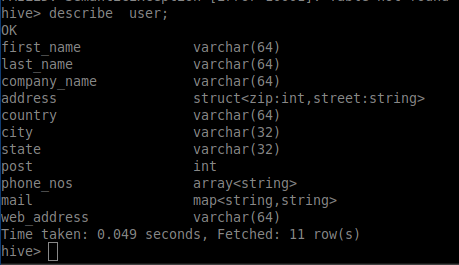


### 3. DESCRIBE TABLE in Hive

The ****DESCRIBE**** statement in Hive shows the lists of columns for the specified table.

****Syntax:****

DESCRIBE [EXTENDED|FORMATTED] [db\_name.] table\_name[.**col\_name** ( [.field\_name])];



### 4. DROP TABLE in Hive

The ****DROP TABLE**** statement in Hive deletes the data for a particular table and remove all metadata associated with it from Hive metastore. It also removes the hdfs file related to table

If ****PURGE**** is not specified then the data is actually moved to the .Trash/current directory. If ****PURGE**** is specified, then data is lost completely.

****Syntax:****

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

### 5. ALTER TABLE in Hive

The ****ALTER TABLE**** statement in Hive enables you to change the structure of an existing table. Using the ALTER TABLE statement we can rename the table, add columns to the table, change the table properties, etc.

****Syntax to Rename a table:****

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

****DDL ALTER TABLE name Example:****

In this example, we are trying to rename the ‘Employee’ table to ‘Com\_Emp’ using the ALTER statement.

****Syntax to Add columns to a table:****

ALTER TABLE table\_name ADD **COLUMNS** (column1, column2) ;

****Syntax to set table properties:****

ALTER TABLE table\_name SET **TBLPROPERTIES** (‘property\_key’=’property\_new\_value’);

### 6. TRUNCATE TABLE

****TRUNCATE TABLE**** statement in Hive removes all the rows from the table or partition. It also removes the hdfs file related to table

****Syntax:****

TRUNCATE TABLE table\_name;