HIVE

What is Hive?

- Hive is an Extract-Transform-Load (ETL) and data warehouse tool which resides on top of Hadoop to summarize Big Data, and it makes querying and analyzing easy.
- The three important functionalities for which Hive is deployed are data summarization, data analysis and data query and is used for structured data.
- It is a platform used to develop SQL type scripts to do MapReduce operations.

Features of Hive

- It stores schema in a database and processed data into HDFS.
- It is designed for Online Analytical Processing (OLAP). Hive is not designed for Online transaction processing (OLTP)
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Architecture of Hive

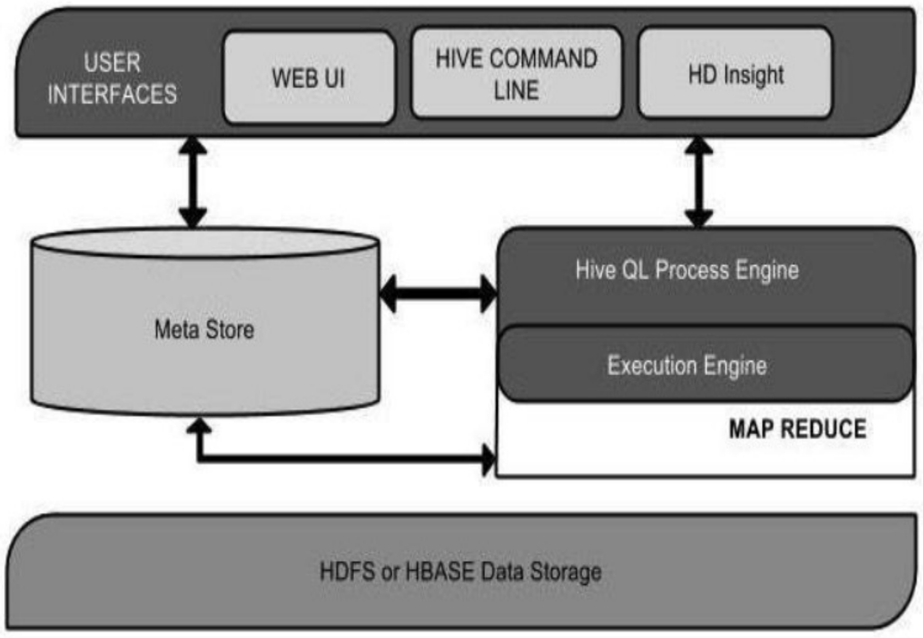
- **User Interface** – Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight.

- **Meta Store** –Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types and HDFS mapping.

- **HiveQL Process Engine** – HiveQL is similar to SQL for querying on schema info on the Megastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.

- **Execution Engine** – The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results.

- **HDFS or HBASE** – Hadoop distributed file system or HBASE are the data storage techniques to store data into the file system.



Working of Hive

1) **Execute Query**: The Hive interface such as Command Line or Web UI sends query to Driver to execute.

2) **Get Plan**: The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.

3) **Get Metadata**: The compiler sends metadata request to Metastore (any database).

4) **Send Metadata**: Metastore sends metadata as a response to the compiler.

5) **Send Plan**: The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.

6) **Execute Plan**: The driver sends the execute plan to the execution engine.

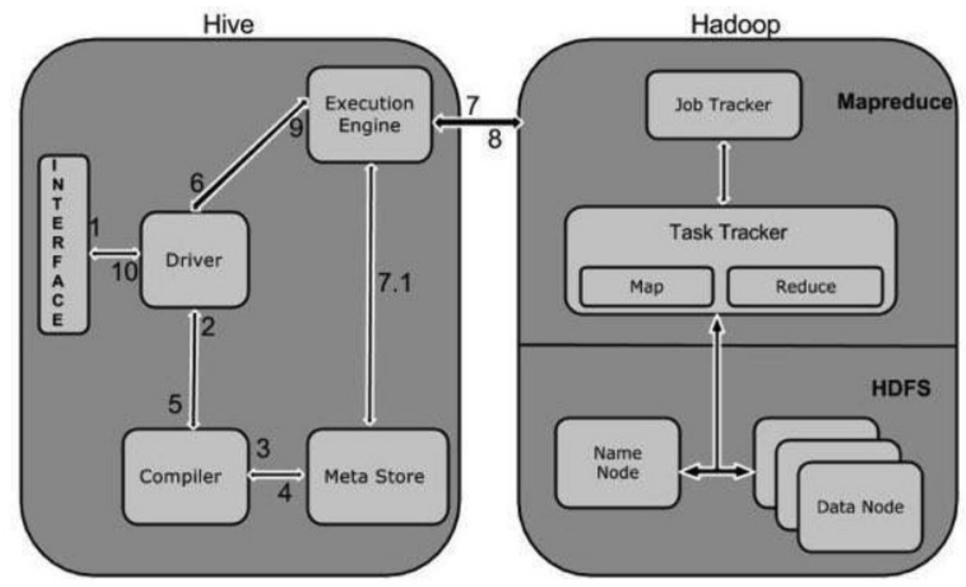
7) **Execute Job**: Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.

7.1) **Metadata Ops**: Meanwhile in execution, the execution engine can execute metadata operations with Metastore.

8) **Fetch Result**: The execution engine receives the results from Data nodes.

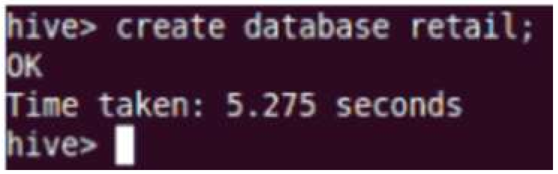
9) **Send Results**: The execution engine sends those resultant values to the driver.

10) **Send Results**: The driver sends the results to Hive Interfaces.

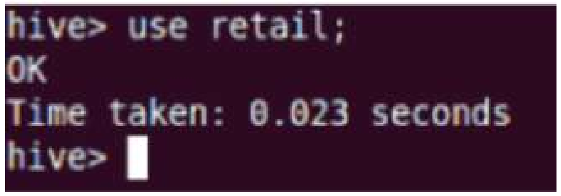


Hive Commands :
Data Definition Language (DDL)

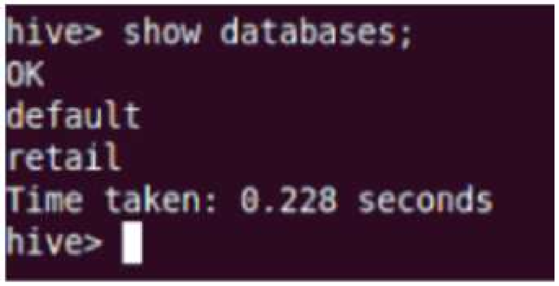
DDL statements are used to build and modify the tables and other objects in the database.



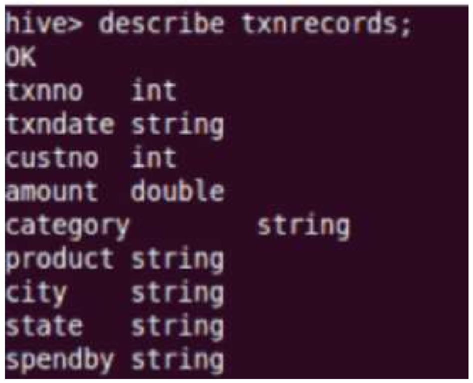
```
hive> create database retail;
OK
Time taken: 5.275 seconds
hive>
```



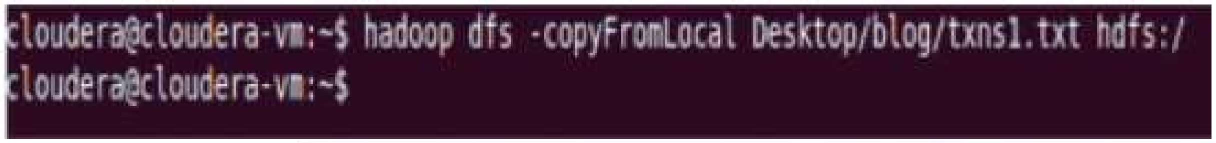
```
hive> use retail;
OK
Time taken: 0.023 seconds
hive>
```



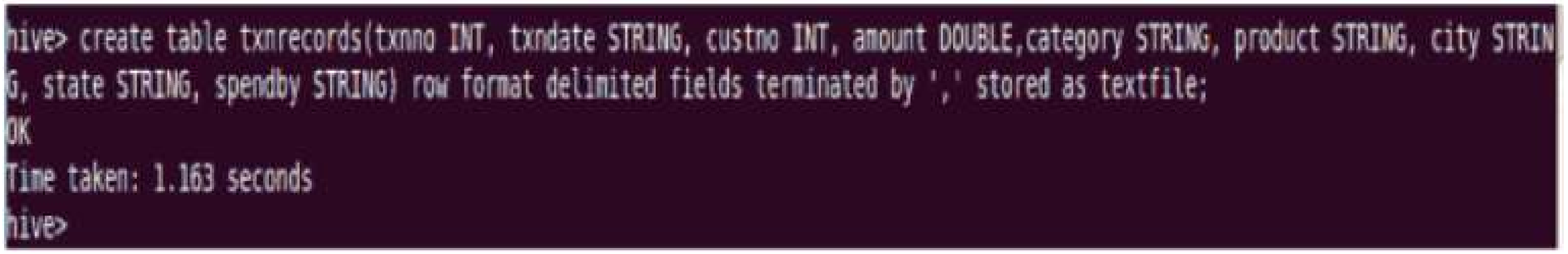
```
hive> show databases;
OK
default
retail
Time taken: 0.228 seconds
hive>
```



```
hive> describe txnrecords;
OK
txnno   int
txndate string
custno  int
amount  double
category        string
product string
city    string
state   string
spendby string
```



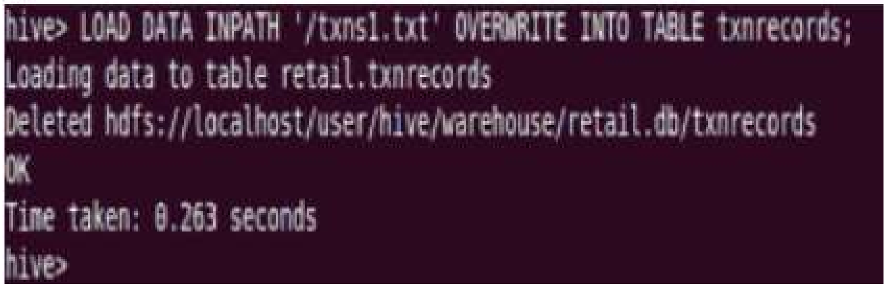
```
cloudera@cloudera-vm:~$ hadoop dfs -copyFromLocal Desktop/blog/txns1.txt hdfs:/
cloudera@cloudera-vm:~$
```



```
hive> create table txnrecords(txnno INT, txndate STRING, custno INT, amount DOUBLE,category STRING, product STRING, city STRIN
G, state STRING, spendby STRING) row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 1.163 seconds
hive>
```

Data Manipulation Language (DML)

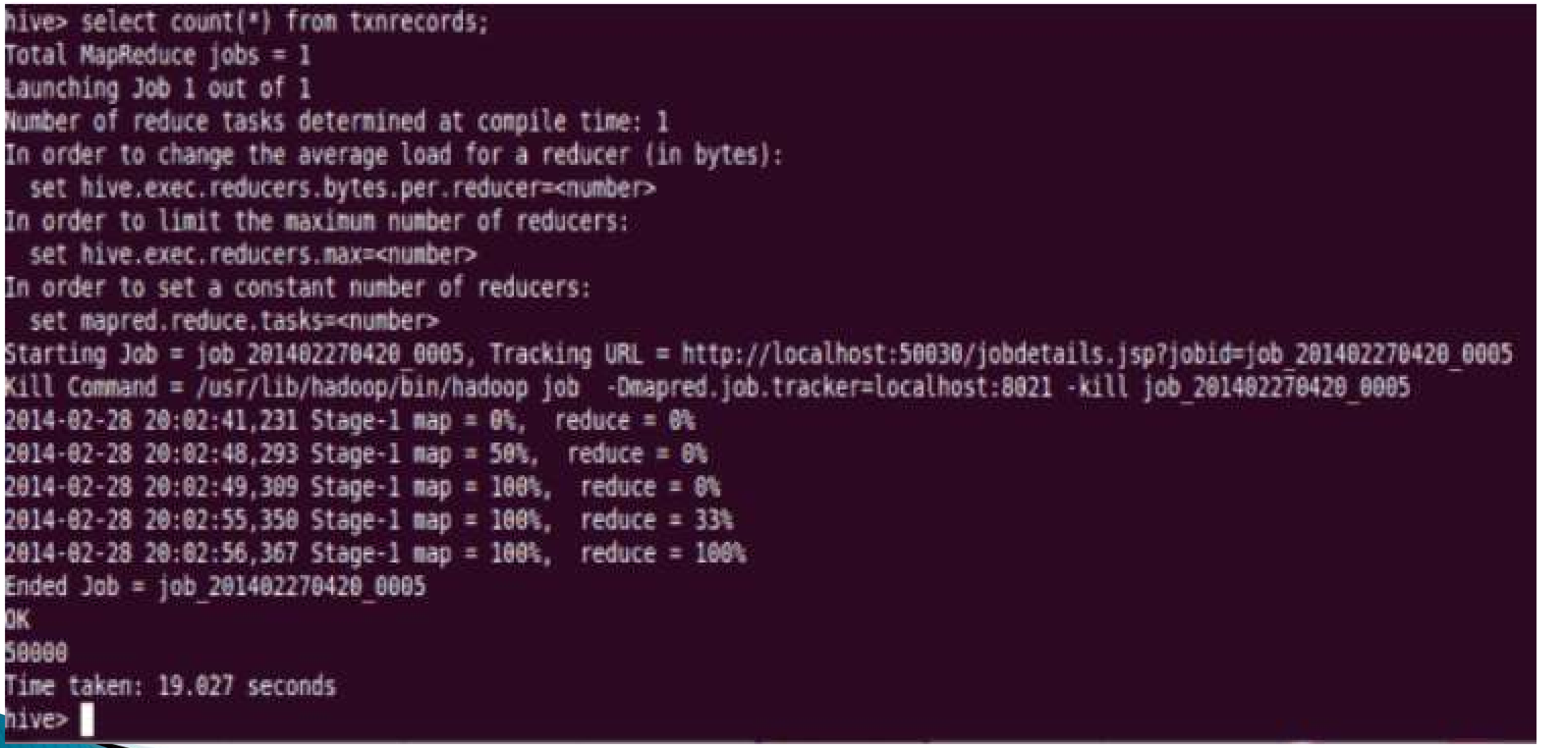
DML statements are used to retrieve, store, modify, delete, insert and update data in the database.



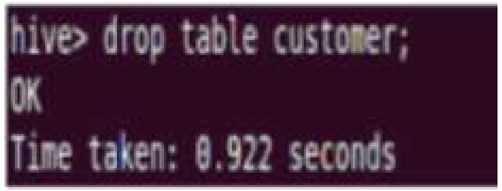
```
hive> LOAD DATA INPATH '/txns1.txt' OVERWRITE INTO TABLE txnrecords;
Loading data to table retail.txnrecords
Deleted hdfs://localhost/user/hive/warehouse/retail.db/txnrecords
OK
Time taken: 0.263 seconds
hive>
```

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

Once you load the data into the table, aggregate function can be done using the command: **Select count(*) from table_name**



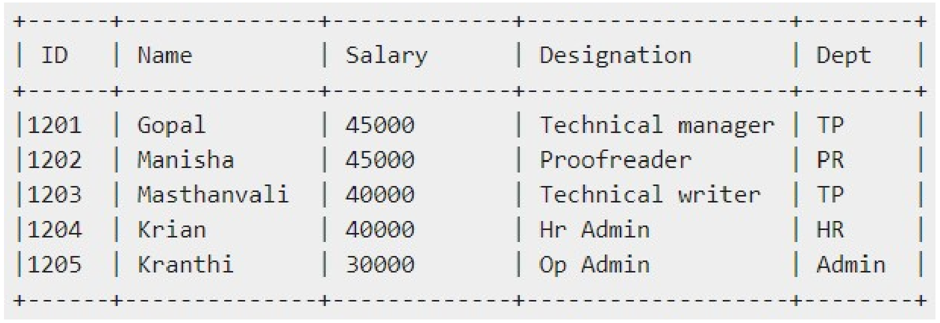
```
hive> select count(*) from txnrecords;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201402270420_0005, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201402270420_0005
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201402270420_0005
2014-02-28 20:02:41,231 Stage-1 map = 0%,  reduce = 0%
2014-02-28 20:02:48,293 Stage-1 map = 50%,  reduce = 0%
2014-02-28 20:02:49,309 Stage-1 map = 100%,  reduce = 0%
2014-02-28 20:02:55,350 Stage-1 map = 100%,  reduce = 33%
2014-02-28 20:02:56,367 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201402270420_0005
OK
50000
Time taken: 19.027 seconds
hive> 
```



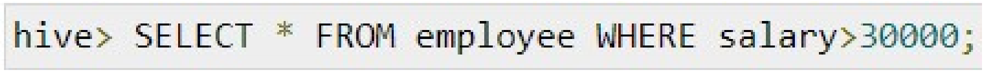
```
hive> drop table customer;
OK
Time taken: 0.922 seconds
```

DROP Table Command in Hive: Drops the table and all the data associated with it in the Hive metastore. DROP table command removes the metadata and data for a particular table. Data is usually moved to .

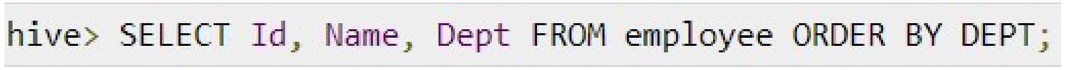
Example 2:



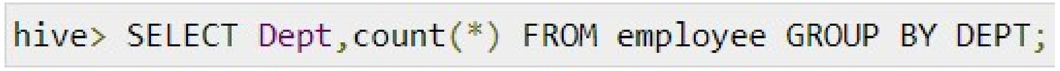
```
+------+--------------+------------+-------------------+--------+
| ID   | Name         | Salary     | Designation       | Dept   |
+------+--------------+------------+-------------------+--------+
|1201  | Gopal        | 45000      | Technical manager | TP     |
|1202  | Manisha      | 45000   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



```
hive> SELECT * FROM employee WHERE salary>30000;
```



```
hive> SELECT Id, Name, Dept FROM employee ORDER BY DEPT;
```

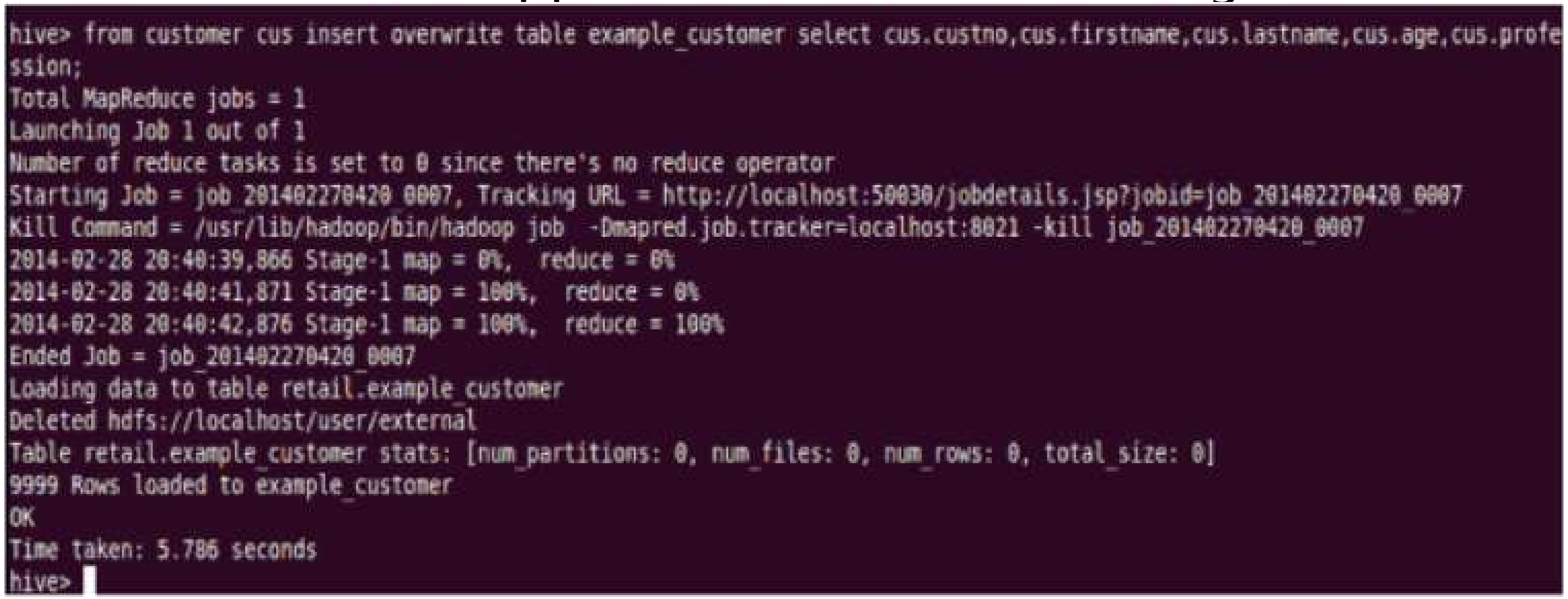


```
hive> SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

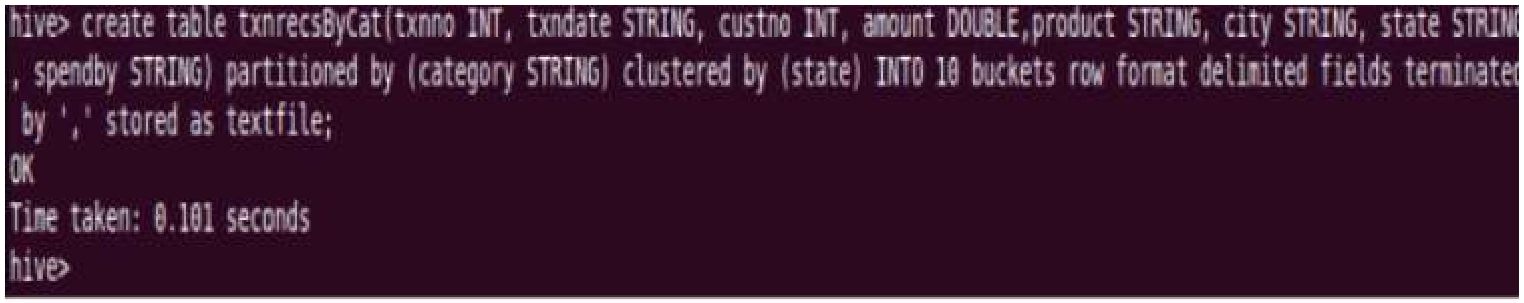
The insert command is used to load the data Hive table. Inserts can be done to a table or a partition.

• INSERT OVERWRITE is used to overwrite the existing data in the table or partition.

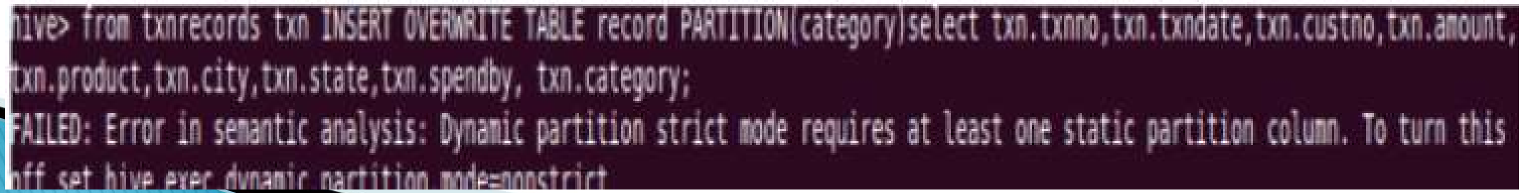
• INSERT INTO is used to append the data into existing data in a table.



```
hive> from customer cus insert overwrite table example_customer select cus.custno,cus.firstname,cus.lastname,cus.age,cus.profe
ssion;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201402270420_0007, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201402270420_0007
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201402270420_0007
2014-02-28 20:40:39,866 Stage-1 map = 0%,  reduce = 0%
2014-02-28 20:40:41,871 Stage-1 map = 100%,  reduce = 0%
2014-02-28 20:40:42,876 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201402270420_0007
Loading data to table retail.example_customer
Deleted hdfs://localhost/user/external
Table retail.example_customer stats: [num_partitions: 0, num_files: 0, num_rows: 0, total_size: 0]
9999 Rows loaded to example_customer
OK
Time taken: 5.786 seconds
hive>
```



```
hive> create table txnrecsByCat(txnno INT, txndate STRING, custno INT, amount DOUBLE,product STRING, city STRING, state STRIN
, spendby STRING) partitioned by (category STRING) clustered by (state) INTO 10 buckets row format delimited fields terminate
 by ',' stored as textfile;
OK
Time taken: 0.101 seconds
hive>
```



```
hive> from txnrecords txn INSERT OVERWRITE TABLE record PARTITION(category)select txn.txnno,txn.txndate,txn.custno,txn.amount,
txn.product,txn.city,txn.state,txn.spendby, txn.category;
FAILED: Error in semantic analysis: Dynamic partition strict mode requires at least one static partition column. To turn this
off set hive.exec.dynamic.partition.mode=nonstrict
```

Hive Data Model

- Table

Apache Hive tables are the same as the tables present in a Relational Database. The table in Hive is logically made up of the data being stored. And the associated metadata describes the layout of the data in the table. We can perform *filter, project, join* and *union* operations on tables. In Hadoop data typically resides in HDFS, although it may reside in any Hadoop filesystem, including the local filesystem. But Hive stores the metadata in a relational database and not in HDFS. Hive has two types of tables which are as follows:

Managed Table and External Table

- Partition

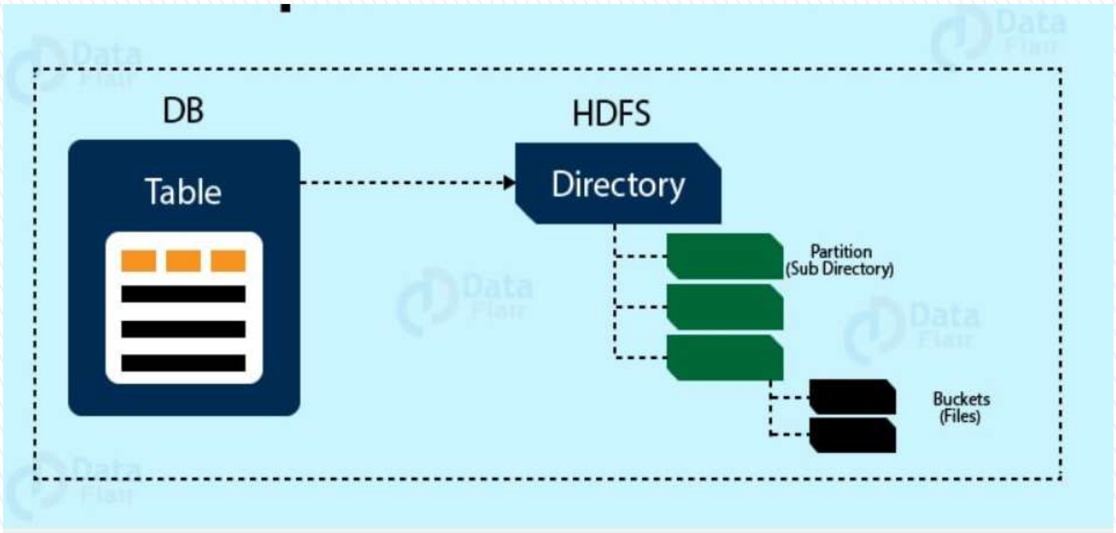
Apache Hive organizes tables into **partitions** for grouping same type of data together based on a column or partition key. Each table in the hive can have one or more partition keys to identify a particular partition. Using partition we can also make it faster to do queries on slices of the data.

CREATE TABLE table_name (column1 data_type, column2 data_type) PARTITIONED BY (partition1 data_type, partition2 data_type,….);

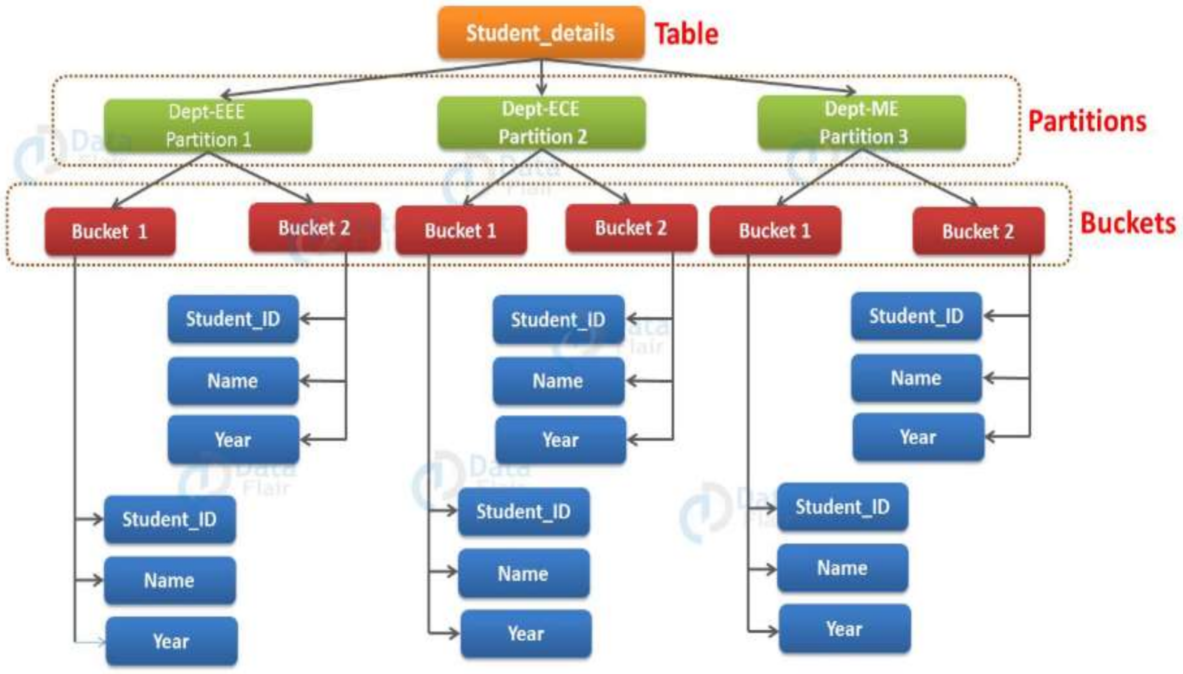
- Buckets

In Hive, Tables or partition are subdivided into **buckets** based on the hash function of a column in the table to give extra structure to the data that may be used for more efficient queries.

CREATE TABLE table_name PARTITIONED BY (partition1 data_type, partition2 data_type,….) CLUSTERED BY (column_name1, column_name2, …) SORTED BY (column_name [ASC|DESC], …)] INTO num_buckets BUCKETS;



Apache Hive Data Model



Managed Table

When we load data into a Managed table, Hive moves data into Hive warehouse directory.

- CREATE TABLE managed_table (dummy STRING);
- LOAD DATA INPATH '/user/tom/data.txt' INTO table managed_table;
- DROP TABLE managed_table

This will delete the table including its data and metadata. The data no longer exists anywhere. This is what it means for HIVE to manage the data. Hive solely controls the Managed table security. Within Hive, security needs to be managed; probably at the schema level.

External Table

We can control the creation and deletion of the data. The location of the external data is specified at the table creation time.

- CREATE EXTERNAL TABLE external_table (dummy STRING)
- LOCATION '/user/tom/external_table';
- LOAD DATA INPATH '/user/tom/data.txt' INTO TABLE external_table;

These tables' files are accessible to anyone who has access to HDFS file structure. So, it needs to manage security at the HDFS file/folder level.

We need data to remain in the underlying location even after a DROP TABLE. This may apply if we are pointing multiple schemas at a single data set.

Thank you