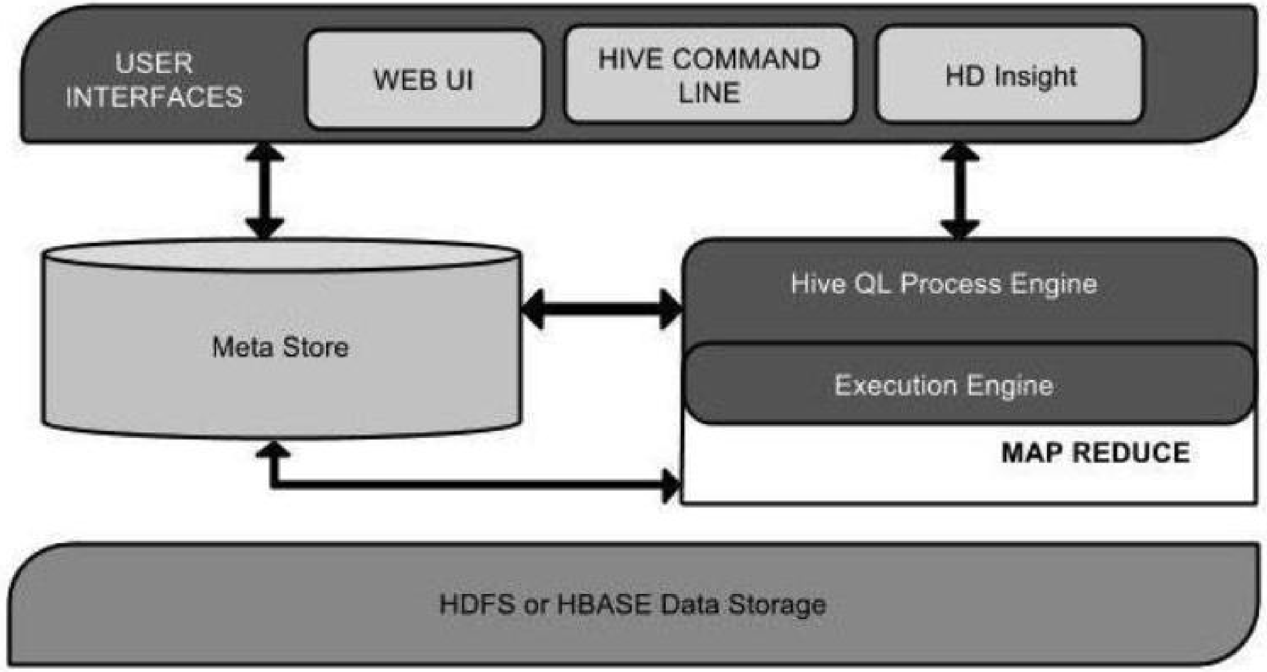
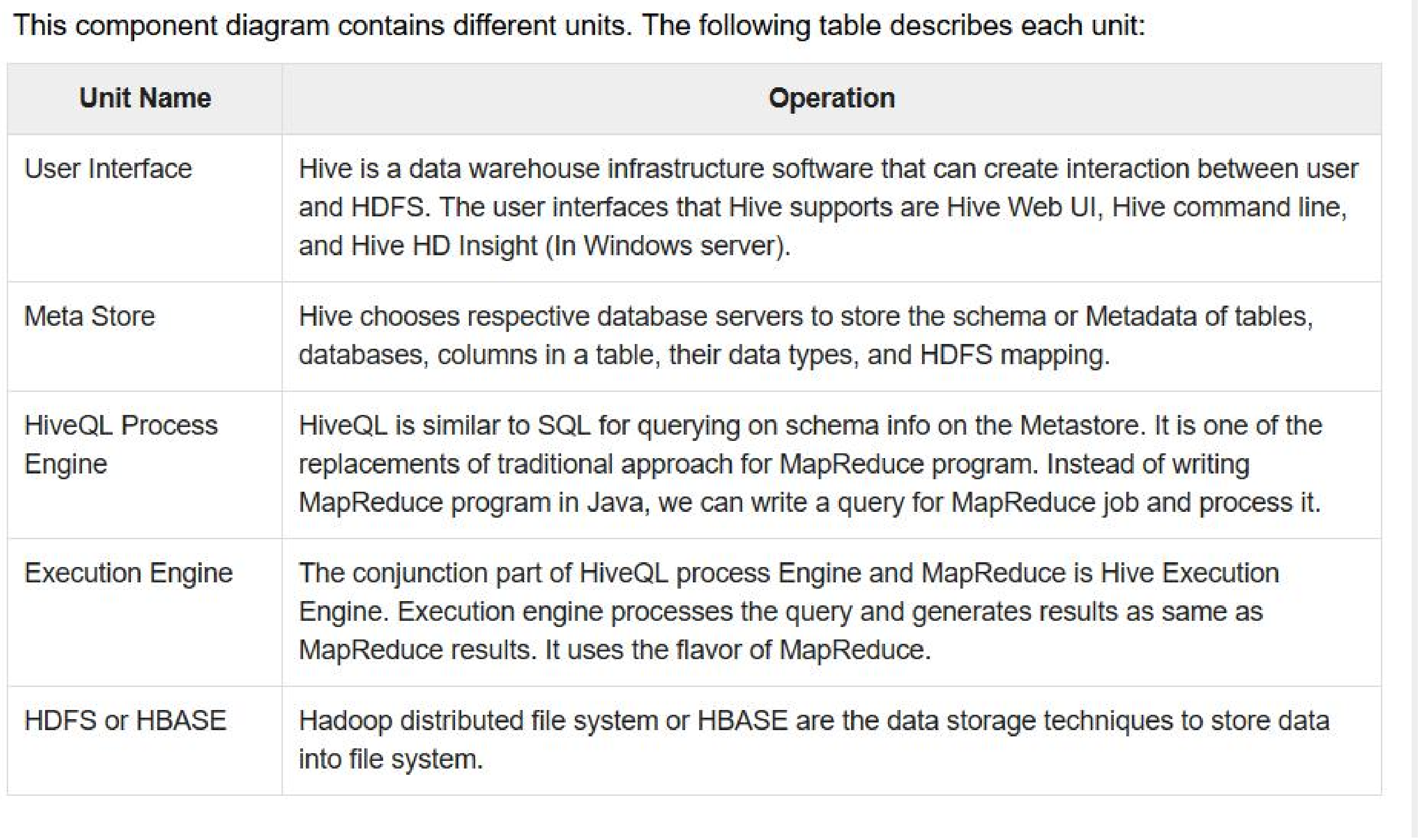
Architecture of Hive

The following component diagram depicts the architecture of Hive:



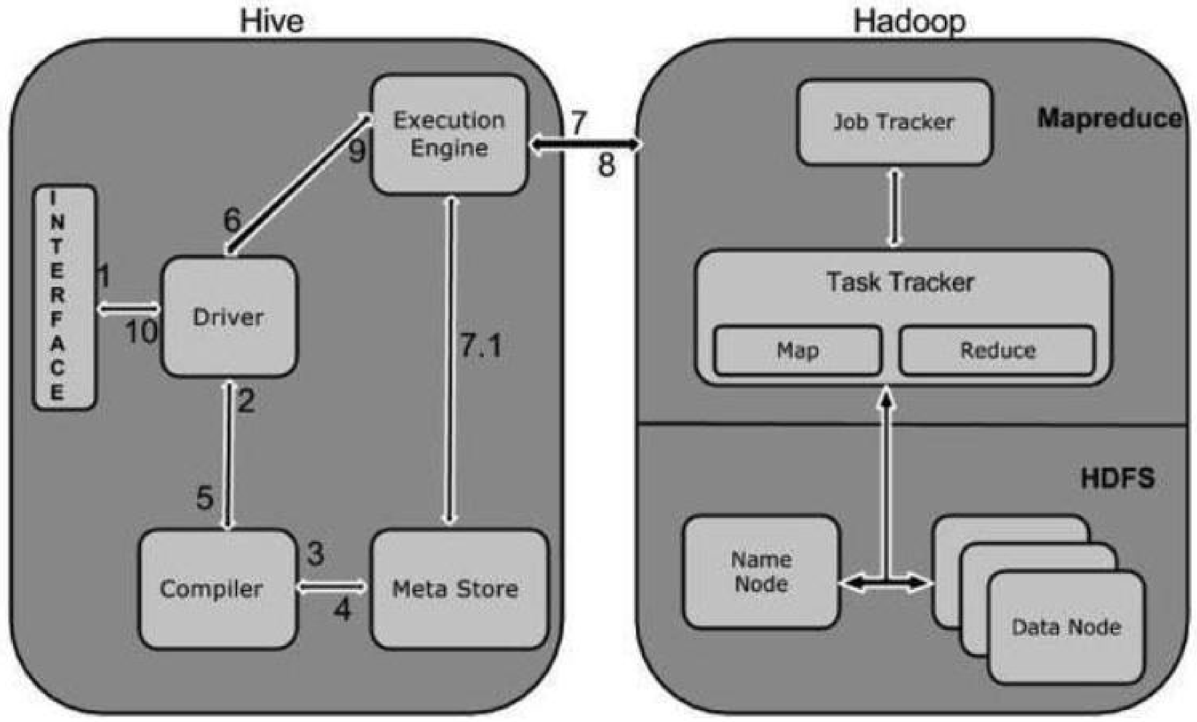


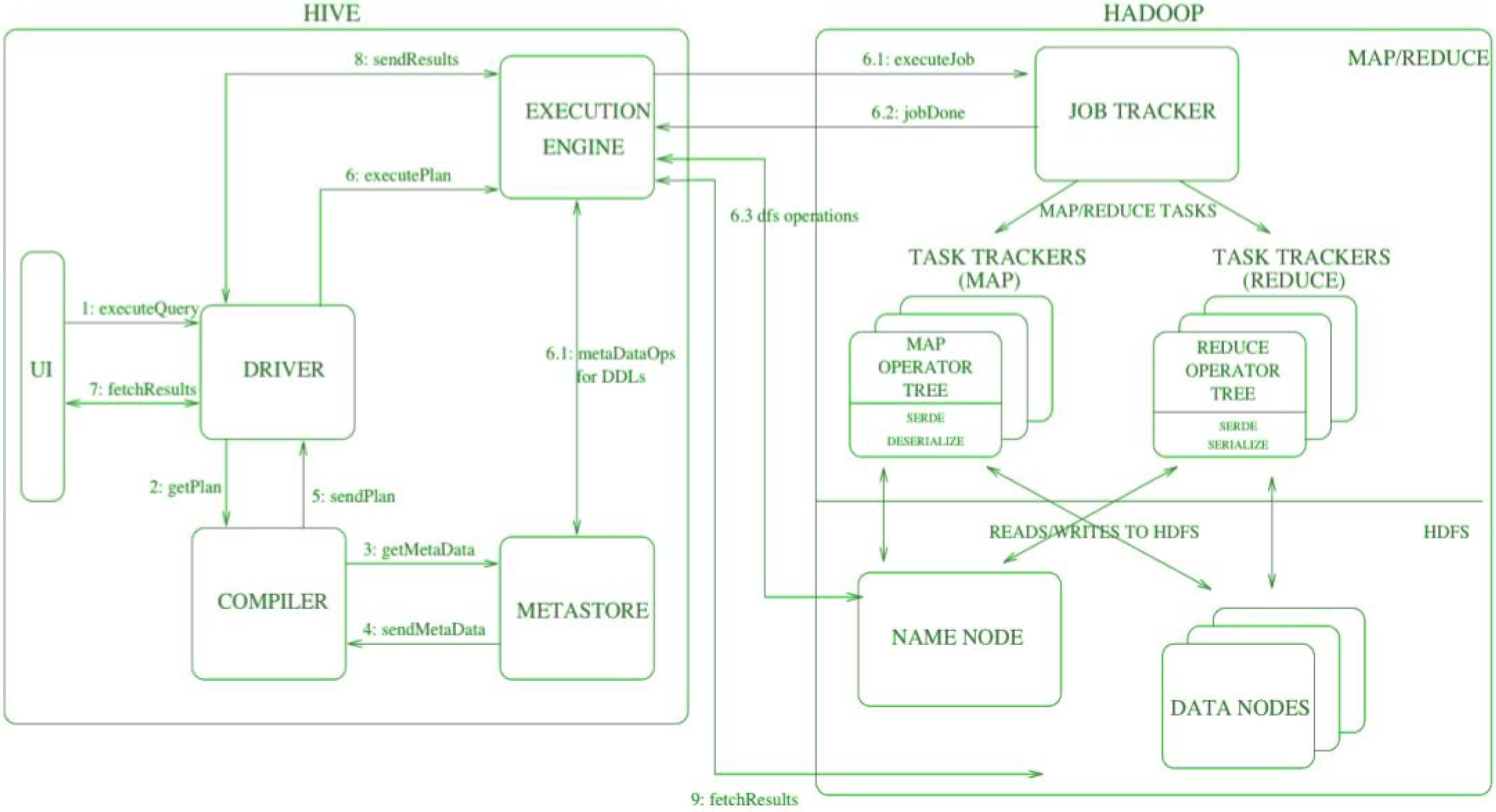
This component diagram contains different units. The following table describes each unit:

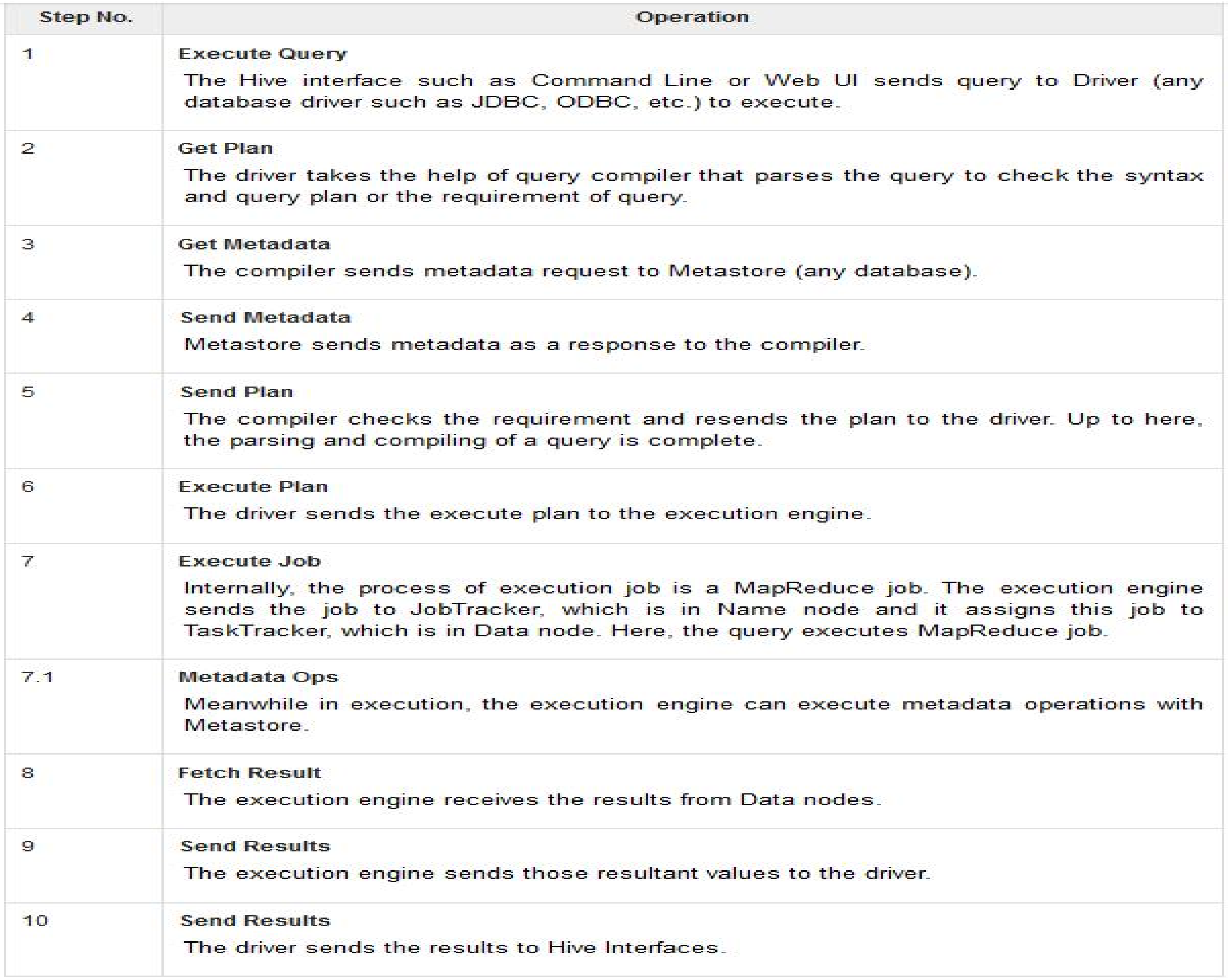
Unit Name	Operation
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers 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 Metastore. 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. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.




HIVE
HADOOP
MAP/REDUCE
8: sendResults
EXECUTION ENGINE
6.1: executeJob
6.2: jobDone
JOB TRACKER
6: executePlan
6.3 dfs operations
MAP/REDUCE TASKS
TASK TRACKERS (MAP)
TASK TRACKERS (REDUCE)
MAP OPERATOR TREE
SERDE DESERIALIZE
REDUCE OPERATOR TREE
SERDE SERIALIZE
UI
1: executeQuery
DRIVER
7: fetchResults
6.1: metaDataOps for DDLs
2: getPlan
5: sendPlan
READS/WRITES TO HDFS
HDFS
COMPILER
3: getMetaData
METASTORE
4: sendMetaData
NAME NODE
DATA NODES
9: fetchResults



Step No.	Operation
1	**Execute Query** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) 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.

The file name says file1 contains client data table:

tab1/clientdata/file1
id, name, dept, yoj
1, sunny, SC, 2009
2, animesh, HR, 2009
3, sumeer, SC, 2010
4, sarthak, TP, 2010

Now, let us partition above data into two files using years
tab1/clientdata/2009/file2
1, sunny, SC, 2009
2, animesh, HR, 2009
tab1/clientdata/2010/file3
3, sumeer, SC, 2010
4, sarthak, TP, 2010

- CREATE TABLE studentTab (id INT, name STRING, dept STRING, yoj INT) PARTITIONED BY (year STRING);

- LOAD DATA LOCAL INPATH tab1’/clientdata/2009/file2’OVERWRITE INTO TABLE studentTab PARTITION (year='2009');

- LOAD DATA LOCAL INPATH tab1’/clientdata/2010/file3’OVERWRITE INTO TABLE studentTab PARTITION (year='2010');

Adding a Partition

```
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, ...)
```

```
hive> ALTER TABLE StudentTab > ADD PARTITION (year=’2009’) > location '/2009/part2009';
```

Renaming a Partition

```
ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION partition_spec;
```

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

HBase Commands

A list of HBase commands are given below.

Create: Creates a new table identified by 'table1' and Column Family identified by 'colf'.

Put: Inserts a new record into the table with row identified by 'row‘.

Scan: returns the data stored in table

Get: Returns the records matching the row identifier provided in the table

Help: Get a list of commands

```
create 'table1', 'colf'
list 'table1'
      put 'table1', 'row1', 'colf:a', 'value1'
      put 'table1', 'row1', 'colf:b', 'value2'
      put 'table1', 'row2', 'colf:a', 'value3'
 scan 'table1'
 get 'table1', 'row1'
```

Using **put** command, you can insert rows into a table. Its syntax is as follows:

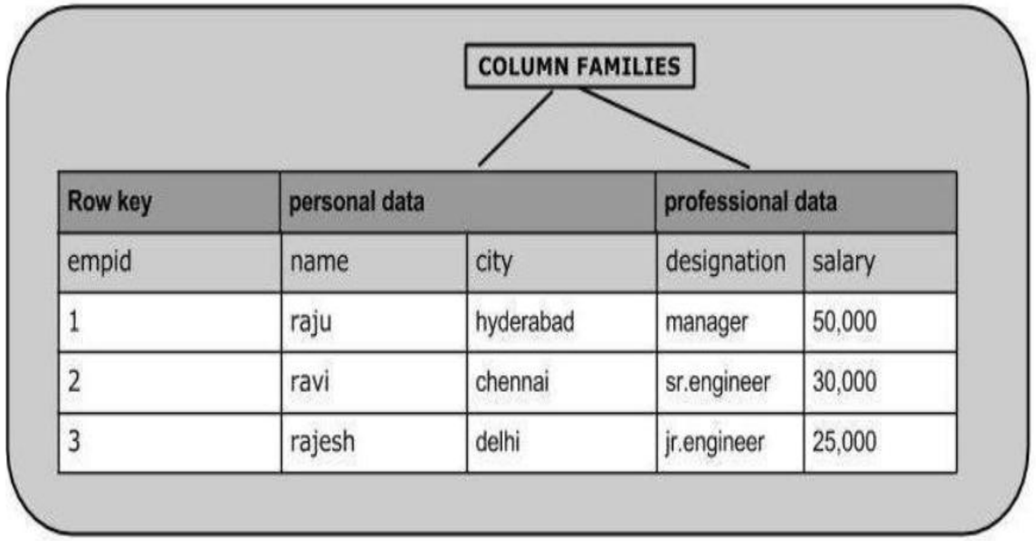
```
put '<table name>','row1','<colfamily:colname>','<value>'
```

Inserting the First Row

Let us insert the first row values into the emp table as shown below.

```
hbase(main):005:0> put 'emp','1','personal data:name','raju'
0 row(s) in 0.6600 seconds
hbase(main):006:0> put 'emp','1','personal data:city','hyderabad'
0 row(s) in 0.0410 seconds
hbase(main):007:0> put 'emp','1','professional
data:designation','manager'
0 row(s) in 0.0240 seconds
hbase(main):007:0> put 'emp','1','professional data:salary','50000'
0 row(s) in 0.0240 seconds
```

s an example, we are going to create the following table in HBase.



Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

```
hbase(main):022:0> scan 'emp'

   ROW                          COLUMN+CELL
1 column=personal data:city, timestamp=1417524216501, value=hyderabad

1 column=personal data:name, timestamp=1417524185058, value=ramu

1 column=professional data:designation, timestamp=1417524232601,

 value=manager

1 column=professional data:salary, timestamp=1417524244109, value=50000

2 column=personal data:city, timestamp=1417524574905, value=chennai

2 column=personal data:name, timestamp=1417524556125, value=ravi

2 column=professional data:designation, timestamp=1417524592204,

 value=sr:engg

2 column=professional data:salary, timestamp=1417524604221, value=30000

3 column=personal data:city, timestamp=1417524681780, value=delhi

3 column=personal data:name, timestamp=1417524672067, value=rajesh

3 column=professional data:designation, timestamp=1417524693187,

value=jr:engg
3 column=professional data:salary, timestamp=1417524702514,

value=25000
```