* **Hive**
* **Hive Tutorial.**

1. **What is Hive**?

Data warehouse software project built on top of Apache Hadoop.

Using Hive we can query and analyze the data stored in HDFS.

Hive is used to analyse structured data.

HQL queries implicitly translated to one or more Hadoop Map reduce job(s) for execution.

1. **What is the hive version used for DMP?**

1.3.0

hive --version

1. **Features of hive? What hive gives us?**

It provides SQL type language for querying called HiveQL or HQL.

It is designed for OLAP (OnLine Analytical Processing).

It stores schema in a database and processed data in HDFS

It is familiar, fast, scalable, and extensible.

1. **Hive is not?**

Hive is not a relational database

Hive is not designed for OnLine Transaction Processing (OLTP)

Hive is not a language for realtime queries and row-level updates

1. **Hive modules/Components?**

The main components of Hive are:

1. **Metastore**:

This component is used to store Schema, metadata of tables, databases and HDFS mapping.

Meta store component can be folder like database called as DERBY or it can be MYSQL.

**Which is the metastore used in DMP?**

**postgresql**

1. **Driver**:

Hive Driver consists of Compiler, Optimizer and Executor.

**Compiler:** Upon receiving a HiveQL statement driver invokes compiler.

The compiler then translates this statement into a plan which consists of a DAG (directed acyclic graph) of map-reduce jobs.

**Optimizer:** Enhances HiveQL for faster execution

**Executor:** The driver submits the individual map-reduce jobs from the DAG to the execution engine in a topological order.

**Which is the execution engine of hive?**

The execution engine used by Hive currently is Hadoop.

Hive manages just life cycle of HiveQL statement during compilation, optimization and execution. The execution engine is Hadoop

On receiving the HiveQL statement, from the Thrift server or some other interfaces, it creates a session handle using session handle we can keep track of statistics like execution time, number of output rows, etc.

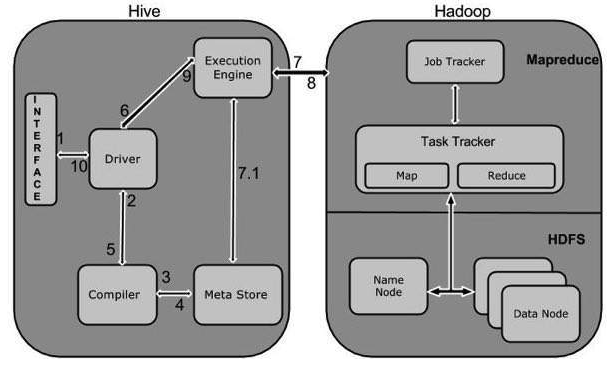
1. **Interfaces to use Hive**

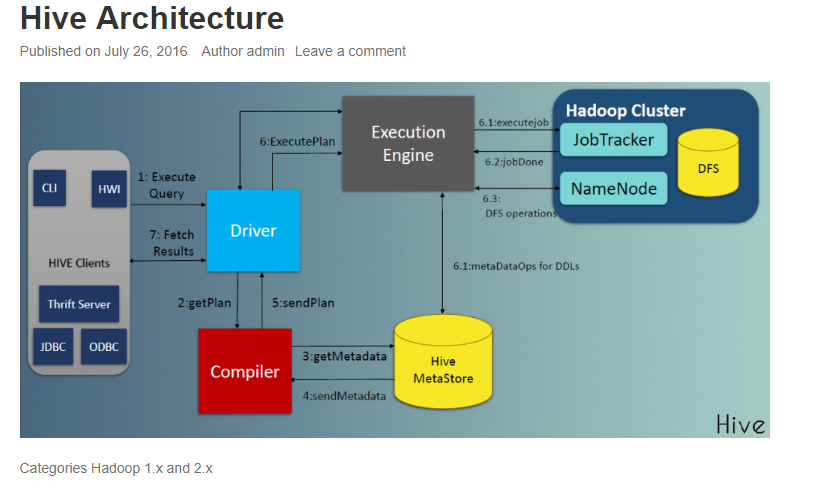
CLI: **C**ommand **L**ine **I**nterface(beeline) like MySQL shell connected to database. **HWI** **H**ive **W**eb **I**nterface

JDBC/ODBC client for programmatic access.

1. **Thrift server:** Interface between hive driver and other applications. This server exposes a very simple client API to execute HiveQL statements
2. **Working of Hive?**

**Select count(\*) from TABLE\_NAME;**

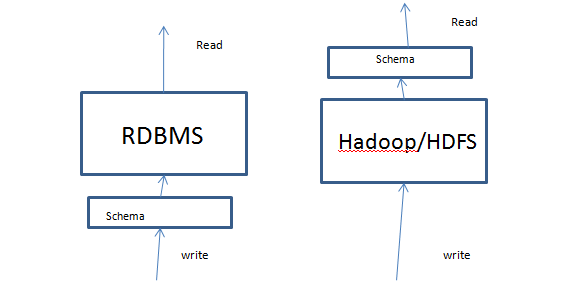




1. **Execute Query:** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2. **Get Plan:** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3. **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.

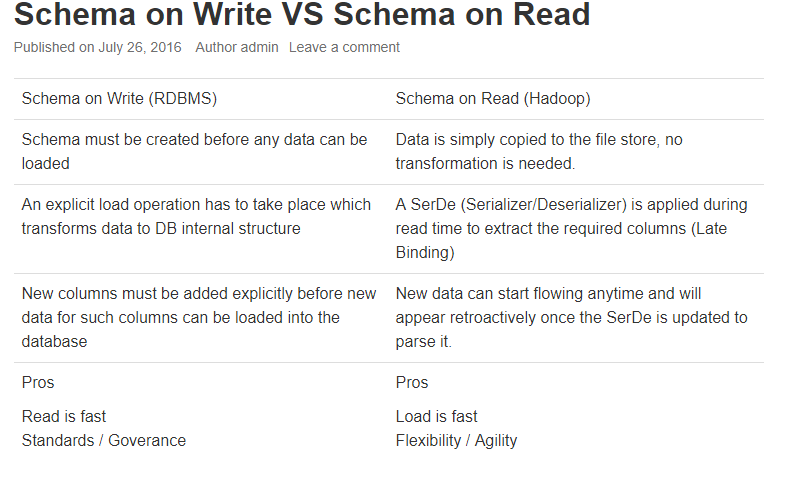
**7.1 Metadata Ops:** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.

1. **Fetch Result:** The execution engine receives the results from Data nodes.
2. **Send Results:** The execution engine sends those resultant values to the driver.
3. **Send Results:** The driver sends the results to Hive Interfaces.
4. **Schema on Write and schema on read**

****

**In case of RDBMS we write data once and read regularly. Write frequency is less and read frequency is more. So we need good prformace for read so RDBMS uses schema on write.**

**In case of Hadoop as it is for bigdata. Huge data flow will be there daily and we need to save it so we need more performance during writing. Where as reading frequency is less say for example we read one day’s data once for analysis. So hive uses schema on read.**

****

1. **What happens when there is data type mismatch in case of hive?**

As hive follows schema on read null values will be displayed for the fields when data type mismatch happens q

**Data Units of Hive:**

**Databases:**

In hive databases are created to avoid naming conflicts for tables, views, partitions, columns, and so on.

Also creating database provides security so that we can provide access to one user or group of user.

**Tables:**

Homogeneous units of data which have the same schema

**Partitions:**

Each Table can have one or more partition Keys which determines how the data is stored. Partitions—apart from being storage units—also allow the user to efficiently identify the rows that satisfy a specified criteria; for example, a date\_partition of type STRING and country\_partition of type STRING. Each unique value of the partition keys defines a partition of the Table. For example, all "US" data from "2009-12-23" is a partition of the page\_views table. Therefore, if you run analysis on only the "US" data for 2009-12-23, you can run that query only on the relevant partition of the table, thereby speeding up the analysis significantly. Note however, that just because a partition is named 2009-12-23 does not mean that it contains all or only data from that date; partitions are named after dates for convenience; it is the user's job to guarantee the relationship between partition name and data content! Partition columns are virtual columns, they are not part of the data itself but are derived on load.

**Buckets (or Clusters):**

 Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of the Table. For example the page\_views table may be bucketed by userid, which is one of the columns, other than the partitions columns, of the page\_view table. These can be used to efficiently sample the data.

**Loading Data into Hive table:**

* **Language Manual**
  + **Data manipulate statement**
    - **Data retrival queries**
      * **UDF**

**String functions:**

|  |  |  |
| --- | --- | --- |
| **Return type** | **Name(signature)** | **Description** |
| int | **instr**(string str, string substr) | This is like indexOf() of java.  Returns the position of the first occurrence of substr in str. Returns null if either of the arguments are null and returns 0 if substr could not be found in str. Be aware that this is not zero based. The first character in str has index 1 |
| String | **get\_json\_object**(string json\_string, string path) | Extracts json object from a json string based on json path specified, and returns json string of the extracted json object. It will return null if the input json string is invalid. NOTE: The json path can only have the characters [0-9a-z\_], i.e., no upper-case or special characters. Also, the keys \*cannot start with numbers.\* This is due to restrictions on Hive column names. |
|  |  |  |

**Built in table generating functions: (UDTF)**