

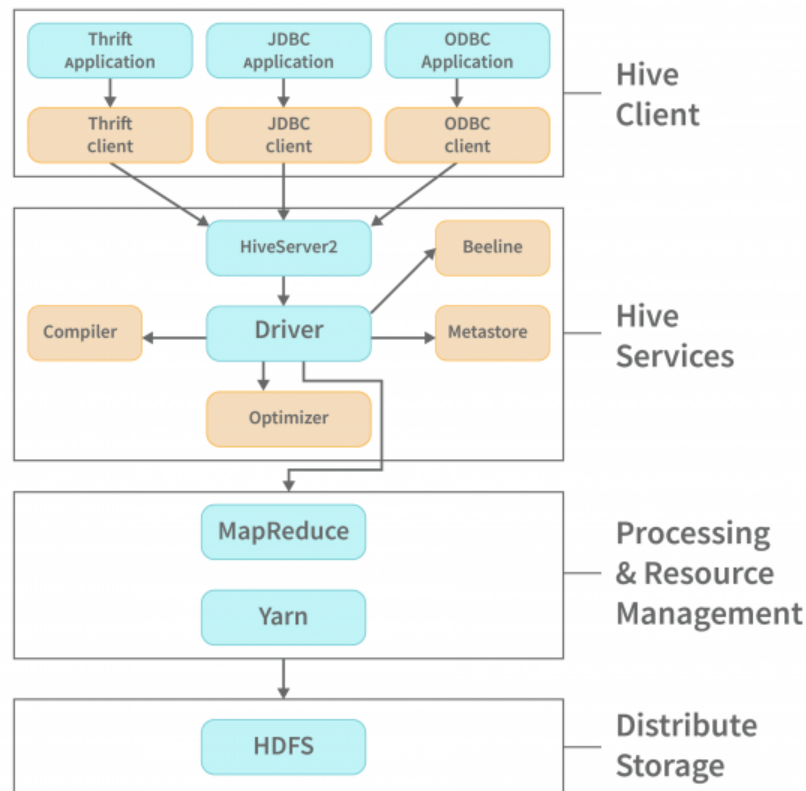
Apache Hive

The **Apache Hive™** data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage and queried using SQL syntax.

Built on top of **Apache Hadoop™**, Hive provides the following features:

- Tools to enable easy access to data via SQL, thus enabling data warehousing tasks such as extract/transform/load (ETL), reporting, and data analysis.
- A mechanism to impose structure on a variety of data formats
- Access to files stored either directly in **Apache HDFS™** or in other data storage systems such as **Apache HBase™**
- Query execution via Apache Tez™, Apache Spark™, or MapReduce
- Procedural language with HPL-SQL
- Sub-second query retrieval via Hive LLAP, Apache YARN and Apache Slider.
- Large-scale data analysis
- Perform Ad-hoc queries
- Perform Data encapsulation

Hive Architecture & Its Components



Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives an SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

Here are some key points about Hive:

- 1. SQL-like Query Language:** Hive provides a query language called HiveQL (HQL) that is similar to SQL. This makes it easier for people with SQL knowledge to write queries without needing to learn a new language.
- 2. Data Warehousing:** Hive is designed to handle large datasets, making it ideal for data warehousing tasks. It allows for easy storage, querying, and analysis of large volumes of data.

3. **Schema on Read:** Unlike traditional databases that use "schema on write," Hive uses "schema on read," which means that the data can be structured, semi-structured, or unstructured, and the schema is applied when the data is read.
4. **Integration with Hadoop:** Hive integrates seamlessly with Hadoop, utilizing HDFS (Hadoop Distributed File System) for storage and MapReduce for processing. This allows it to leverage the distributed computing power of Hadoop clusters.
5. **Extensibility:** Hive can be extended with custom UDFs (User-Defined Functions), UDAFs (User-Defined Aggregation Functions), and UDTFs (User-Defined Table-Generating Functions).
6. **Partitioning and Bucketing:** Hive supports partitioning and bucketing to optimize query performance. Partitioning allows for dividing the data into segments based on a column, which can speed up queries that filter on that column. Bucketing further divides the data in each partition into buckets, which helps in more efficient sampling and query performance.
7. **Metastore:** Hive maintains a metastore, which is a central repository for storing metadata about the data stored in Hive. This includes information about the schemas, tables, columns, and partitions.
8. **Support for Various File Formats:** Hive supports a variety of file formats including plain text, RCFile, ORC (Optimized Row Columnar), Parquet, and Avro.

Hive is widely used in scenarios where there is a need to perform data analysis on large datasets, such as in data lakes, data warehouses, and big data analytics platforms. It allows users to manage and query big data efficiently without needing to write complex MapReduce programs.

Hive provides standard SQL functionality, including many of the later SQL:2003, SQL:2011, and SQL:2016 features for analytics.

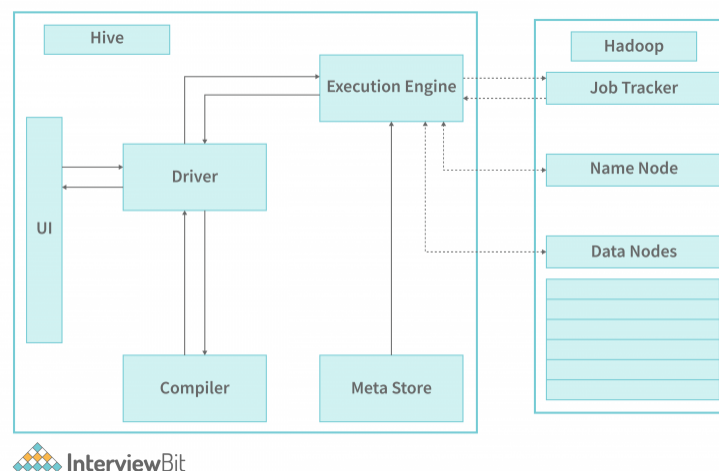
Hive's SQL can also be extended with user code via user defined functions (UDFs), user defined aggregates (UDAFs), and user defined table functions (UDTFs).

There is not a single "Hive format" in which data must be stored. Hive comes with built in connectors for comma and tab-separated values (CSV/TSV) text files, Apache Parquet™, Apache ORC™, and other formats. Users can extend Hive with connectors for other formats. Please see File Formats and Hive SerDe in the Developer Guide for details.

Hive is not designed for online transaction processing (OLTP) workloads. It is best used for traditional data warehousing tasks.

Hive is designed to maximize scalability (scale out with more machines added dynamically to the Hadoop cluster), performance, extensibility, fault-tolerance, and loose-coupling with its input formats.

Hive Execution



It have SQL like Command we will step...

Hive Data Types

Numeric Types

- TINYINT (1-byte signed integer, from 128 to 127)
- SMALLINT (2-byte signed integer, from 32,768 to 32,767)

- `INT` / `INTEGER` (4-byte signed integer, from -2,147,483,648 to 2,147,483,647)
- `BIGINT` (8-byte signed integer, from 9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)
- `FLOAT` (4-byte single precision floating point number)
- `DOUBLE` (8-byte double precision floating point number)
- `DOUBLE PRECISION` (alias for `DOUBLE`, only available starting with Hive [2.2.0](#))
- `DECIMAL`
 - Introduced in Hive [0.11.0](#) with a precision of 38 digits
 - Hive [0.13.0](#) introduced user-definable precision and scale
- `NUMERIC` (same as `DECIMAL` , starting with [Hive 3.0.0](#))

Date/Time Types

- `TIMESTAMP` (Note: Only available starting with Hive [0.8.0](#))
- `DATE` (Note: Only available starting with Hive [0.12.0](#))
- `INTERVAL` (Note: Only available starting with Hive [1.2.0](#))

String Types

- `STRING`
- `VARCHAR` (Note: Only available starting with Hive [0.12.0](#))
- `CHAR` (Note: Only available starting with Hive [0.13.0](#))

Misc Types

- `BOOLEAN`
- `BINARY` (Note: Only available starting with Hive [0.8.0](#))

Complex Types

- arrays: `ARRAY<data_type>` (Note: negative values and non-constant expressions are allowed as of [Hive 0.14.](#))

- maps: `MAP<primitive_type, data_type>` (Note: negative values and non-constant expressions are allowed as of Hive 0.14.)
- structs: `STRUCT<col_name : data_type [COMMENT col_comment], ...>`
- union: `UNIONTYPE<data_type, data_type, ...>` (Note: Only available starting with Hive 0.7.0.)

Column Types

Integral Types (`TINYINT` , `SMALLINT` , `INT/INTEGER` , `BIGINT`)

Integral literals are assumed to be `INT` by default, unless the number exceeds the range of `INT` in which case it is interpreted as a `BIGINT`, or if one of the following postfixes is present on the number.

Type	Postfix	Example
TINYINT	Y	100Y
SMALLINT	S	100S
BIGINT	L	100L

Reference -

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types>

Note - There are the things Query Engine(Hive) and Data will be in any Distributed Storage like HDFS, S3, Gen2 etc and metadata in any RDBMS like MsSQL, MySQL , Derby etc.

Hive Installation with MySQL

Creating Table in Hive

In Hive there is two types of table

Managed Tables

- **Definition:** Managed tables, also known as internal tables, are the default table type in Hive. When you create a table without specifying it as external, Hive treats it as a managed table.
- **Data Storage:** Hive manages the data for these tables, meaning it controls where and how the data is stored in the Hadoop Distributed File System (HDFS).
- **Location:** By default, the data for a managed table is stored in a directory under the Hive warehouse directory, typically at `/user/hive/warehouse/<tablename>`.
- **Lifecycle Management:** Hive is responsible for the lifecycle of the data in managed tables. If you drop a managed table, Hive will delete both the metadata and the actual data stored in HDFS.
- **Use Case:** Managed tables are useful when you want Hive to handle the data storage and lifecycle management, including creation, maintenance, and deletion of data.

External Tables

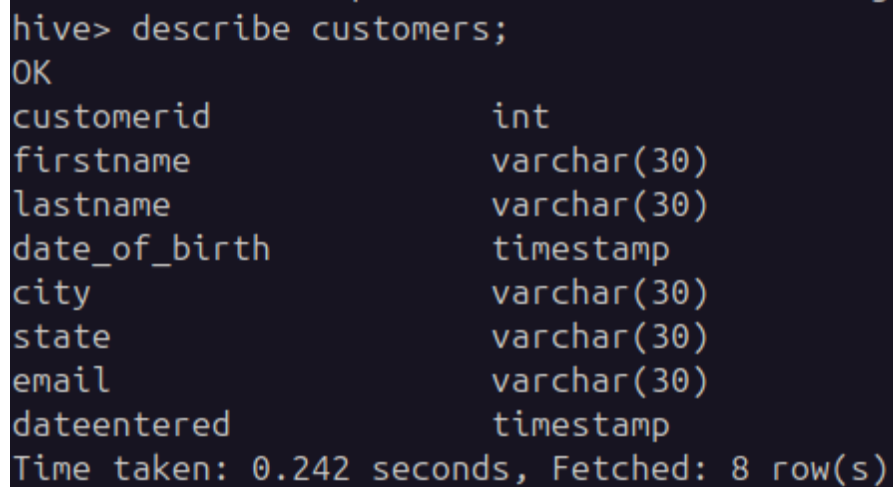
- **Definition:** External tables allow you to manage the data outside of Hive. When creating an external table, you explicitly specify that it is external and provide the location of the data.
- **Data Storage:** Hive does not control the storage of data for external tables. Instead, it references data that is stored externally, which could be in HDFS or any other storage system supported by Hadoop.
- **Location:** You must specify the location of the data when creating an external table. For example, `LOCATION '/path/to/data'`.
- **Lifecycle Management:** Hive does not manage the lifecycle of the data in external tables. If you drop an external table, Hive only deletes the metadata about the table, leaving the actual data untouched.
- **Use Case:** External tables are useful when you need to manage data outside of Hive or share the data between different systems or applications, ensuring that dropping a table in Hive does not delete the underlying data.

Create a your first Managed Table

```
CREATE TABLE IF NOT EXISTS Customers (  
  CustomerID INT,  
  FirstName VARCHAR(30),  
  LastName VARCHAR(30),  
  Date_of_Birth TIMESTAMP,  
  City VARCHAR(30),  
  State VARCHAR(30),  
  Email VARCHAR(30),  
  DateEntered TIMESTAMP  
)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ','  
STORED AS TEXTFILE;
```

After Creating this table run

```
describe customers;
```



```
hive> describe customers;  
OK  
customerid          int  
firstname           varchar(30)  
lastname            varchar(30)  
date_of_birth       timestamp  
city                varchar(30)  
state               varchar(30)  
email               varchar(30)  
dateentered         timestamp  
Time taken: 0.242 seconds, Fetched: 8 row(s)
```

you will something like that.

```
describe extended customers;
```



```
hive> describe extended customers;
OK
customerid      int
firstname        varchar(30)
lastname         varchar(30)
date_of_birth    timestamp
city             varchar(30)
state            varchar(30)
email            varchar(30)
dateentered      timestamp

Detailed Table Information
Table(tableName:customers, dbName:dummy, owner:nooman, createTime:1720937447, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(name:customerid, type:int, comment:null), FieldSchema(name:firstname, type:varchar(30), comment:null), FieldSchema(name:lastname, type:varchar(30), comment:null), FieldSchema(name:date_of_birth, type:timestamp, comment:null), FieldSchema(name:city, type:varchar(30), comment:null), FieldSchema(name:state, type:varchar(30), comment:null), FieldSchema(name:email, type:varchar(30), comment:null), FieldSchema(name:dateentered, type:timestamp, comment:null)], location:hdfs://localhost:9000/user/hive/warehouse/dummy.db/customers, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:1, serdeInfo:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, parameters:{serialization.format=, field.delim=,}), bucketCols:[], sortCols:[], parameters:{}, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}), storedAsSubDirectories:false), partitionKeys:[], parameters:{totalSize=0, numRows=0, rawDataSize=0, COLUMN_STATS_ACCURATE={"BASIC_STATS":"true"}, numFiles=0, transient_lastDdlTime=1720937447}, viewOriginalText:null, viewExpandedText:null, tableType:MANAGED_TABLE, rewriteEnabled:false)
Time taken: 0.221 seconds, Fetched: 10 row(s)
```

Insert from local File System

let insert the some data

```
hive> LOAD DATA INPATH 'hdfs:///Ecommerce/Customers/Customers' ;
```

```
hive> SELECT * FROM Customers limit 5;
```

```
hive> SELECT * FROM Customers limit 5;
OK
57081  James  Smith  1987-03-26 00:00:00  New York  New York  United States  NULL
57082  Robert Downey Jr  1973-05-24 00:00:00  New York  New York  United States  NULL
57083  John   Williams  1990-04-14 00:00:00  Chicago  Illinois  United States  NULL
57084  Michael Johnson 1953-03-25 00:00:00  Brisbane  Queensland  Australia  NULL
57085  Steve  Williams  1971-04-24 00:00:00  Bremen   Bremen   Germany  NULL
Time taken: 0.248 seconds, Fetched: 5 row(s)
```