What is HIVE

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.

Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs. Hive supports Data Definition Language (DDL)Ex: ALTER,TRUNCATE ,CREATE Data Manipulation Language (DML)Ex: INSERT,UPDATE,DELETE, and User Defined Functions (UDF) Ex: CtoF (converts Celsius to Fahrenheit).

Features of Hive

These are the following features of Hive:

- Hive is fast and scalable.
- It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
- It is capable of analyzing large datasets stored in HDFS.
- It allows different storage types such as plain text, RCFile(RCFile is a data placement structure that determines how to store relational tables on computer clusters), and HBase.
- It uses indexing to accelerate queries.
- It can operate on compressed data stored in the Hadoop ecosystem.
- It supports user-defined functions (UDFs) where user can provide its functionality.

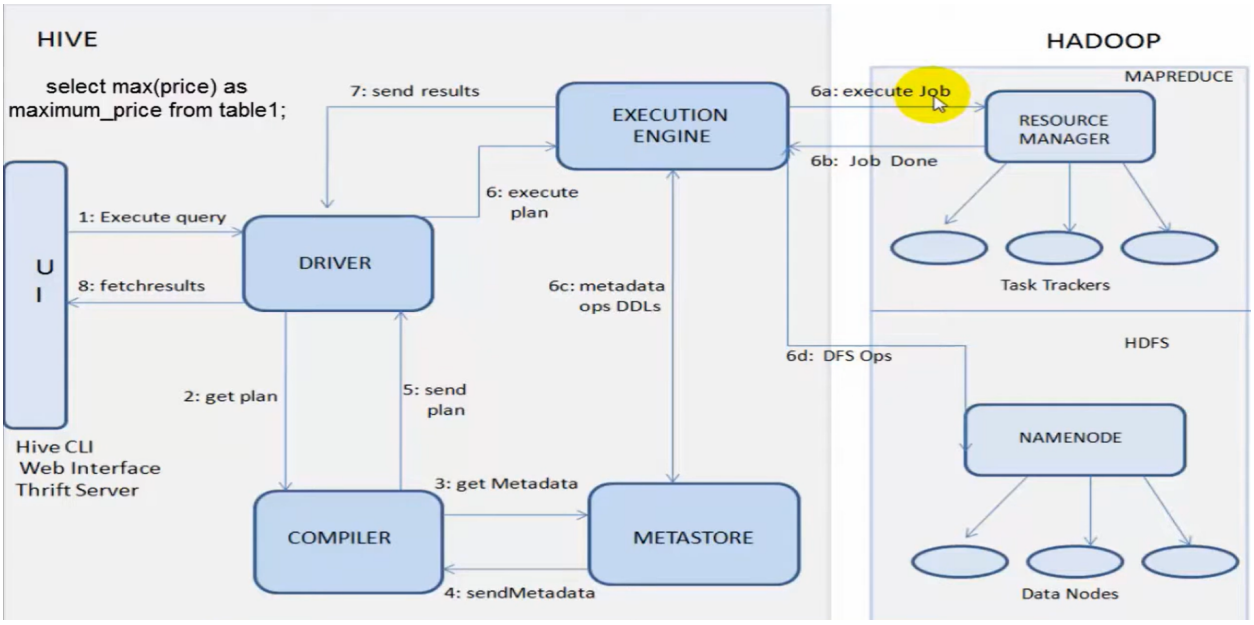
Limitations of Hive

- Hive is not capable of handling real-time data.
- It is not designed for online transaction processing.
- Hive queries contain high latency.

Differences between Hive and Pig

Hive	Pig
Hive is commonly used by Data Analysts.	Pig is commonly used by programmers.
It follows SQL-like queries.	It follows the data-flow language.
It can handle structured data.	It can handle semi-structured data.
It works on server-side of HDFS cluster.	It works on client-side of HDFS cluster.
Hive is slower than Pig.	Pig is comparatively faster than Hive.

Hive Architecture



The following are the services provided by Hive:-

- **Hive CLI** - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
- **Hive Web User Interface** - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- **Hive MetaStore** - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
- **Hive Server** - It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
- **Hive Driver** - It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
- **Hive Compiler** - The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
- **Hive Execution Engine** - Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Hive - Data Types

Column Types

Column type are used as column data types of Hive. They are as follows:

Integral Types

Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

Type	Postfix	Example
TINYINT	Y	10Y
SMALLINT	S	10S
INT	-	10
BIGINT	L	10L

String Types

String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

The following table depicts various CHAR data types:

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

Timestamp

It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format YYYY-MM-DD HH:MM:SS.fffffffff and format yyyy-mm-dd hh:mm:ss.fffffffff.

Dates

DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Decimals

The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

DECIMAL(precision, scale)

decimal(10,0)

Union Types

Union is a collection of heterogeneous data types. You can create an instance using **create union**. The syntax and example is as follows:

UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>

{0:1}

{1:2.0}

{2:["three","four"]}

{3:{"a":5,"b":"five"}}

{2:["six","seven"]}

{3:{"a":8,"b":"eight"}}

{0:9}

{1:10.0}

Literals

The following literals are used in Hive:

Floating Point Types

Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

Decimal Type

Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10^{-308} to 10^{308}.

Null Value

Missing values are represented by the special value NULL.

Complex Types

The Hive complex data types are as follows:

Arrays

Arrays in Hive are used the same way they are used in Java.

Syntax: ARRAY<data_type>

Maps

Maps in Hive are similar to Java Maps.

Syntax: MAP<primitive_type, data_type>

Structs

Structs in Hive is similar to using complex data with comment.

Syntax: STRUCT<col_name : data_type [COMMENT col_comment], ...>

Hive - Create Database

Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it. This chapter explains how to create Hive database. Hive contains a default

Create Database Statement

Create Database is a statement used to create a database in Hive. A database in Hive is a **namespace** or a collection of tables. The **syntax** for this statement is as follows:

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**:

userdb:

hive> CREATE DATABASE [IF NOT EXISTS] userdb;

or

hive> CREATE SCHEMA userdb;

The following query is used to verify a databases list:

hive> SHOW DATABASES;

default

userdb

Drop Database Statement

Drop Database is a statement that drops all the tables and deletes the database. Its syntax is as follows:

DROP DATABASE StatementDROP (DATABASE|SCHEMA) [IF EXISTS] database_name

[RESTRICT|CASCADE];

The following queries are used to drop a database. Let us assume that the database name is **userdb**.

hive> DROP DATABASE IF EXISTS userdb;

The following query drops the database using **CASCADE**. It means dropping respective tables before dropping the database.

hive> DROP DATABASE IF EXISTS userdb CASCADE;

The following query drops the database using **SCHEMA**.

hive> DROP SCHEMA userdb;

This clause was added in Hive 0.6.

Create Table

Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]

[COMMENT table_comment]

[ROW FORMAT row_format]

[STORED AS file_format]

Example

Let us assume you need to create a table named **employee** using **CREATE TABLE** statement. The following table lists the fields and their data types in employee table:

Sr.No	Field Name	Data Type
1	Eid	int
2	Name	String
3	Salary	Float
4	Designation	string

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

COMMENT Employee details

FIELDS TERMINATED BY \t

LINES TERMINATED BY \n

STORED IN TEXT FILE

The following query creates a table named **employee** using the above data.

hive> CREATE TABLE IF NOT EXISTS employee (eid int, name String,

salary String, destination String)

COMMENT Employee details

ROW FORMAT DELIMITED

FIELDS TERMINATED BY \t

LINES TERMINATED BY \n

STORED AS TEXTFILE;

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

On successful creation of table, you get to see the following response:

OK

Time taken: 5.905 seconds

hive>

Alter Table Statement

It is used to alter a table in Hive.

Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

ALTER TABLE name RENAME TO new_name

ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])

ALTER TABLE name DROP [COLUMN] column_name

ALTER TABLE name CHANGE column_name new_name new_type

ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])

Rename To Statement

The following query renames the table from **employee** to **emp**.

hive> ALTER TABLE employee RENAME TO emp;

Drop Table

Drop Table Statement

The syntax is as follows:

DROP TABLE [IF EXISTS] table_name;

The following query drops a table named **employee**:

hive> DROP TABLE IF EXISTS employee;

On successful execution of the query, you get to see the following response:

OK

Time taken: 5.3 seconds

hive>

Partitioning

Hive organizes tables into partitions. It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion of the data.

Tables or partitions are sub-divided into buckets, to provide extra structure to the data that may be used for more efficient querying. Bucketing works based on the value of hash function of some column of a table.

For example, a table named Tab1 contains employee data such as id, name, dept, and yoj (i.e., year of joining). Suppose you need to retrieve the details of all employees who joined in 2012. A query searches the whole table for the required information. However, if you partition the employee data with the year and store it in a separate file, it reduces the query processing time. The following example shows how to partition a file and its data:

The following file contains employeedata table.

/tab1/employeedata/file1

id, name, dept, yoj

1, gopal, TP, 2012

2, kiran, HR, 2012

3, kaleel,SC, 2013

4, Prasanth, SC, 2013

The above data is partitioned into two files using year.

/tab1/employeedata/2012/file2

1, gopal, TP, 2012

2, kiran, HR, 2012

/tab1/employeedata/2013/file3

3, kaleel,SC, 2013

4, Prasanth, SC, 2013

Adding a Partition

We can add partitions to a table by altering the table. Let us assume we have a table called employee with fields such as Id, Name, Salary, Designation, Dept, and yoj.

Syntax:

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, ...)

The following query is used to add a partition to the employee table.

hive> ALTER TABLE employee

> ADD PARTITION (year=2012)

> location '/2012/part2012';

Renaming a Partition

The syntax of this command is as follows.

ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION partition_spec;

The following query is used to rename a partition:

hive> ALTER TABLE employee PARTITION (year=1203)

> RENAME TO PARTITION (Yoj=1203);

Dropping a Partition

The following syntax is used to drop a partition:

ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec, PARTITION partition_spec,...;

The following query is used to drop a partition:

hive> ALTER TABLE employee DROP [IF EXISTS]

> PARTITION (year=1203);

View and Indexes

Creating a View

You can create a view at the time of executing a SELECT statement. The syntax is as follows:

CREATE VIEW [IF NOT EXISTS] view_name [(column_name [COMMENT column_comment], ...)]

[COMMENT table_comment]

AS SELECT ...

Example

Let us take an example for view. Assume employee table as given below, with the fields Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000. We store the result in a view named **emp_30000.**

+------+--------------+-------------+-------------------+--------+
| 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 |
+------+--------------+-------------+-------------------+--------+

The following query retrieves the employee details using the above scenario:

hive> CREATE VIEW emp_30000 AS

SELECT * FROM employee

WHERE salary>30000;

Dropping a View

Use the following syntax to drop a view:

DROP VIEW view_name

The following query drops a view named as emp_30000:

```
hive> DROP VIEW emp_30000;
```

Creating an Index

An Index is nothing but a pointer on a particular column of a table. Creating an index means creating a pointer on a particular column of a table. Its syntax is as follows:

```
CREATE INDEX index_name

ON TABLE base_table_name (col_name, ...)

AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

[IDXPROPERTIES (property_name=property_value, ...)]

[IN TABLE index_table_name]

[PARTITIONED BY (col_name, ...)]

[

  [ ROW FORMAT ...] STORED AS ...

  | STORED BY ...

]

[LOCATION hdfs_path]

[TBLPROPERTIES (...)]
```

Example

Let us take an example for index. Use the same employee table that we have used earlier with the fields Id, Name, Salary, Designation, and Dept. Create an index named index_salary on the salary column of the employee table.

The following query creates an index:

```
hive> CREATE INDEX inedx_salary ON TABLE employee(salary)

AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';
```

It is a pointer to the salary column. If the column is modified, the changes are stored using an index value.

Dropping an Index

The following syntax is used to drop an index:

DROP INDEX <index_name> ON <table_name>

The following query drops an index named index_salary:

hive> DROP INDEX index_salary ON employee;

HiveQL - Select-Joins

Syntax

join_table:

table_reference JOIN table_factor [join_condition]

| table_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table_reference

join_condition

| table_reference LEFT SEMI JOIN table_reference join_condition

| table_reference CROSS JOIN table_reference [join_condition]

Example

We will use the following two tables in this chapter. Consider the following table named CUSTOMERS..

+----+----------+-----+-----------+----------+
| ID | NAME | AGE | ADDRESS | SALARY |
+----+----------+-----+-----------+----------+
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 25 | Mumbai | 6500.00 |
| 5 | Hardik | 27 | Bhopal | 8500.00 |
| 6 | Komal | 22 | MP | 4500.00 |
| 7 | Muffy | 24 | Indore | 10000.00 |
+----+----------+-----+-----------+----------+
Consider another table ORDERS as follows:
+-----+---------------------+-------------+--------+

```
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 | 3000   |
| 100 | 2009-10-08 00:00:00 |           3 | 1500   |
| 101 | 2009-11-20 00:00:00 |           2 | 1560   |
| 103 | 2008-05-20 00:00:00 |           4 | 2060   |
+-----+---------------------+-------------+--------+
```

There are different types of joins given as follows:

- **JOIN**
- **LEFT OUTER JOIN**
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**

JOIN

JOIN clause is used to combine and retrieve the records from multiple tables. JOIN is same as OUTER JOIN in SQL. A JOIN condition is to be raised using the primary keys and foreign keys of the tables.

The following query executes JOIN on the CUSTOMER and ORDER tables, and retrieves the records:

```
hive> SELECT c.ID, c.NAME, c.AGE, o.AMOUNT

FROM CUSTOMERS c JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
| 3  | kaushik  | 23  | 3000   |
| 3  | kaushik  | 23  | 1500   |
| 2  | Khilan   | 25  | 1560   |
| 4  | Chaitali | 25  | 2060   |
+----+----------+-----+--------+
```

LEFT OUTER JOIN

The HiveQL LEFT OUTER JOIN returns all the rows from the left table, even if there are no matches in the right table. This means, if the ON clause matches 0 (zero) records in the right

table, the JOIN still returns a row in the result, but with NULL in each column from the right table.

A LEFT JOIN returns all the values from the left table, plus the matched values from the right table, or NULL in case of no matching JOIN predicate.

The following query demonstrates LEFT OUTER JOIN between CUSTOMER and ORDER tables:

hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

LEFT OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);

On successful execution of the query, you get to see the following response:

```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
| 1  | Ramesh   | NULL   | NULL                |
| 2  | Khilan   | 1560   | 2009-11-20 00:00:00 |
| 3  | kaushik  | 3000   | 2009-10-08 00:00:00 |
| 3  | kaushik  | 1500   | 2009-10-08 00:00:00 |
| 4  | Chaitali | 2060   | 2008-05-20 00:00:00 |
| 5  | Hardik   | NULL   | NULL                |
| 6  | Komal    | NULL   | NULL                |
| 7  | Muffy    | NULL   | NULL                |
+----+----------+--------+---------------------+
```

RIGHT OUTER JOIN

The HiveQL RIGHT OUTER JOIN returns all the rows from the right table, even if there are no matches in the left table. If the ON clause matches 0 (zero) records in the left table, the JOIN still returns a row in the result, but with NULL in each column from the left table.

A RIGHT JOIN returns all the values from the right table, plus the matched values from the left table, or NULL in case of no matching join predicate.

The following query demonstrates RIGHT OUTER JOIN between the CUSTOMER and ORDER tables.

notranslate"> hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE FROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON (c.ID = o.CUSTOMER_ID);

On successful execution of the query, you get to see the following response:

+------+----------+--------+---------------------+
| ID | NAME | AMOUNT | DATE |
+------+----------+--------+---------------------+
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+

FULL OUTER JOIN

The HiveQL FULL OUTER JOIN combines the records of both the left and the right outer tables that fulfil the JOIN condition. The joined table contains either all the records from both the tables, or fills in NULL values for missing matches on either side.

The following query demonstrates FULL OUTER JOIN between CUSTOMER and ORDER tables:

```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE

FROM CUSTOMERS c

FULL OUTER JOIN ORDERS o

ON (c.ID = o.CUSTOMER_ID);
```

On successful execution of the query, you get to see the following response:

+------+----------+--------+---------------------+
| ID | NAME | AMOUNT | DATE |
+------+----------+--------+---------------------+
| 1 | Ramesh | NULL | NULL |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |
| 5 | Hardik | NULL | NULL |
| 6 | Komal | NULL | NULL |
| 7 | Muffy | NULL | NULL |
| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |
| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |
| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |
| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+