Hive -

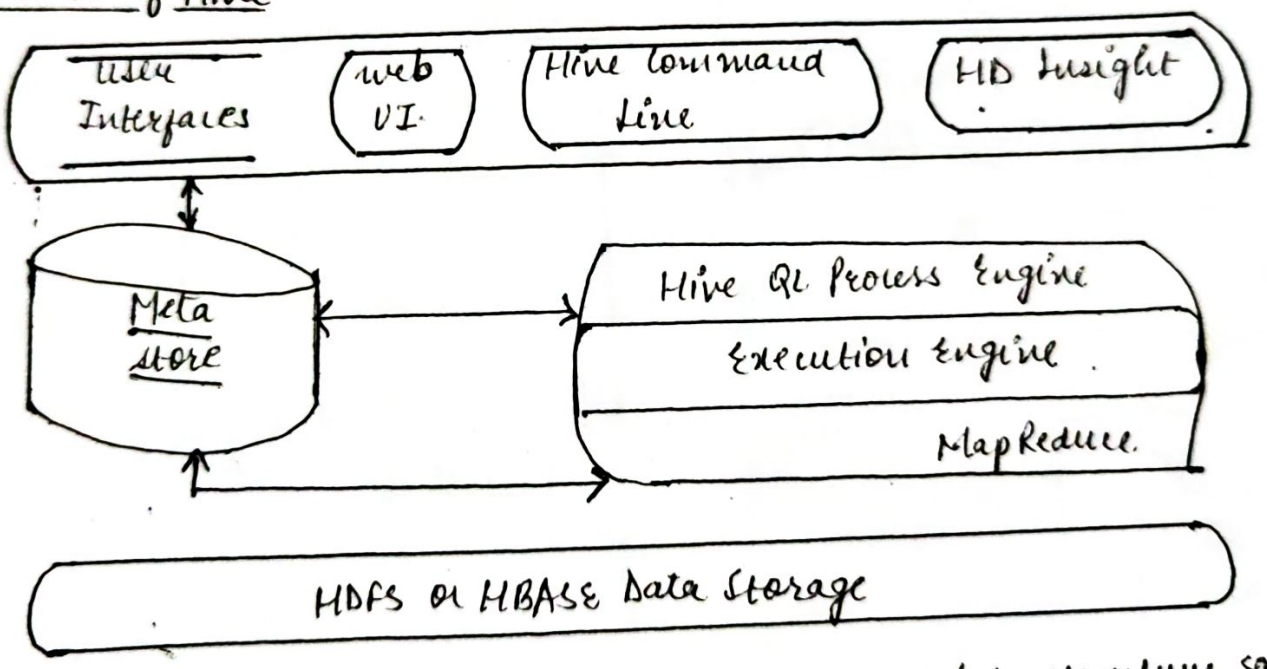
What is Hive

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on the top of Hadoop to summarize big data and makes querying and analyzing easy. Initially, Hive was developed by Facebook, later the Apache software foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. eg :- Amazon uses it in Amazon Elastic MapReduce.

Hive is not :- A relational Database
A design for Online Transaction Processing OLTP
A language for real-time queries and row-level updates

Features :- * It is designed for OLAP
* It provides SQL type language for querying called HiveQL or HQL
* It stores schema in database and processed data into HDFS
* It is familiar, fast, Scalable and extensible.

Architecture of Hive



User Interface : Hive is a data warehouse infrastructure software that can create interaction b/w user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command Line and Hive HD Insight In Windows

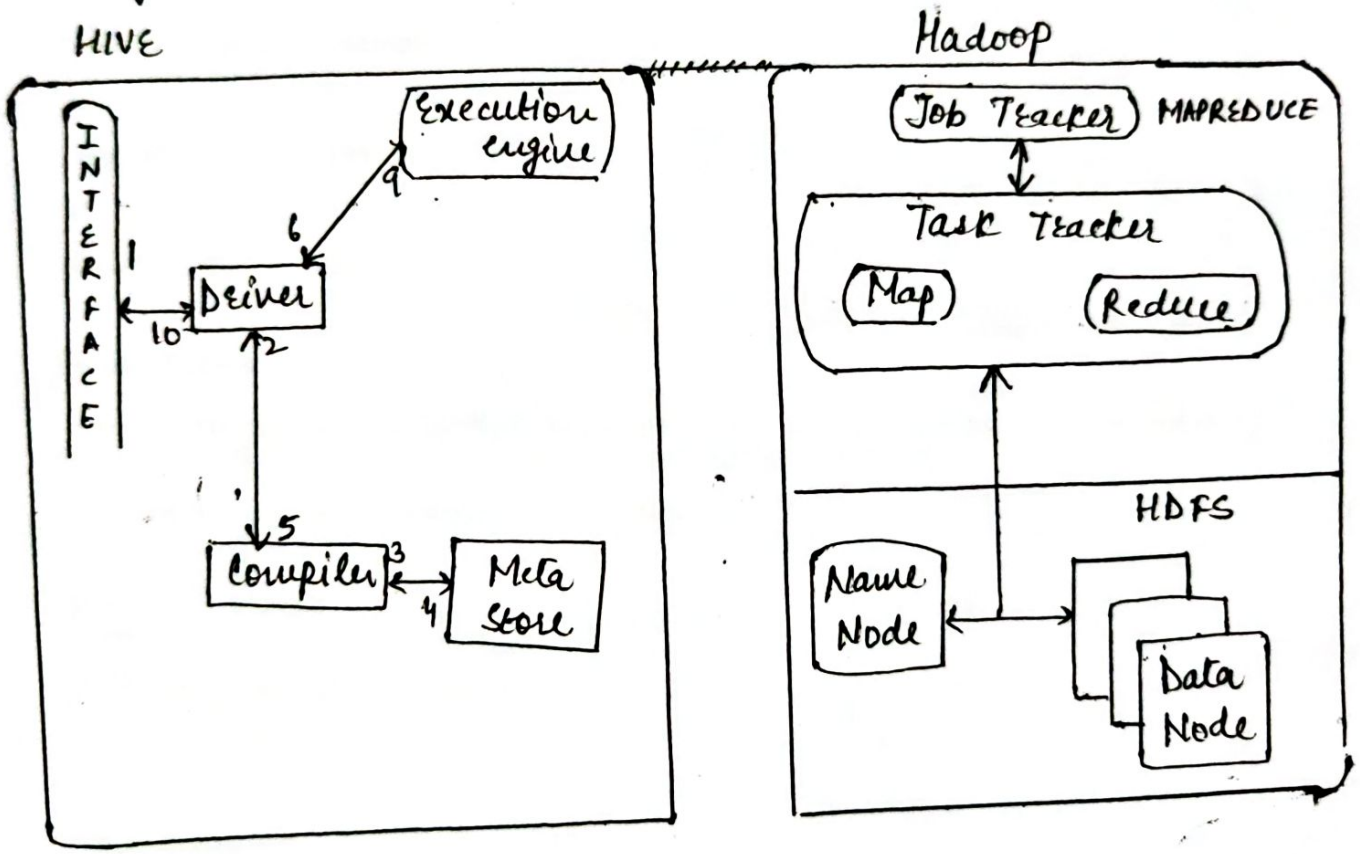
Meta store : Hive chooses respective database servers to store schema or meta data of tables, databases, columns in a table, their datatypes and HDFS mapping

HiveQL Process Engine : HiveQL is similar to SQL for querying on schema-info on 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 result as same as MapReduce results. It reduces the flavor of MapReduce.

HDFS or HBASE : HDFS or HBASE are the data storage technique to store data into file system.

Working of HIVE



The following diagram depicts the workflow b/w HIVE and Hadoop.

HIVE - DATA TYPES

This chapter takes you through the different data types in Hive, which are involved in the table creation. All the data types in Hive are classified into four types, given as follows:

- Column Types
- Literals
- Null Values
- Complex 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.

DDL commands.

Create Db statements

A db in hive is a namespace or collection of tables.

1) hive > create schema userdb;
2) hive > show DATABASES;

DROP database

1. hive> DROP DATABASE IF EXISTS userdb;

Creating Hive Tables

Creating a table called sonoo with columns, the first being an integer and other a string

1. hive> Create TABLE sonoo (foo INT, bar String);

Create a table called HIVE_TABLE with two columns and a partition column called ds. The partition column is a virtual column. It is not a part of the data itself but is derived from the partition that a particular dataset is loaded into. By default, tables are assumed to be a text i/p format and the delimiters are assumed to be ^A (ctrl-a).

1. hive> CREATE TABLE HIVE_TABLE (foo INT, bar STRING)
PARTITIONED BY (ds STRING);

Browse the table

1. hive> show table

Altering And Deopping Table

1. hive> ALTER TABLE sonoo RENAME TO kafka;
2. hive> ALTER TABLE kafka ADD COLUMNS (col INT);
3. hive> ALTER TABLE HIVE_TABLE ADD COLUMN (COL 1 INT COMMEN 'a comment')
4. hive> ALTER TABLE HIVE_TABLE REPLACE COLUMN (col2 INT, weight STRING, baz INT COMMENT 'baz replaces new_col1'

HIVE DML COMMANDS

To understand DML commands, lets see the employee and employee_department table first.

D DATA

ive> LOAD DATA LOCAL INPATH '/usr/Desktop/kv1.txt' OVERWRITE INTO
TABLE employee;

SELECTS and FILTERS

1. hive> SELECT E.EMP_ID FROM employee E where E.Address = 'US';

GROUP BY

1. hive > hive > SELECT E.EMP_ID FROM Employee E GROUP BY E.Address;

HIVEQL - SELECT - WHERE

Syntax of SELECT query

```
SELECT [ALL | DISTINCT] SELECT_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GroupBY col_list]
[HAVING having_condition]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list]]
[LIMIT number];
```

HIVEQL - SELECT - ORDER BY

Syntax of ORDER BY clause

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[LIMIT number];
```

HIVE QL - SELECT - GROUP BY

Syntax of GROUP BY clause

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[HAVING having_condition]
[ORDER BY col_list]]
[LIMIT number];
```

QL - SELECT-JOINS

ntax

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]

JOIN

Join clause is used to combine and retrive 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.

```
hive > SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID)
```

The following query executes JOIN on the CUSTOMER and ORDER table.

EFT OUTER JOIN

The following query demonstrate LEFT OUTER JOIN b/w CUSTOMER and ORDER table.

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

IGHT OUTER JOIN

The following query demonstrate RIGHT OUTER JOIN b/w CUSTOMER & ORDER tab

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

Full OUTER JOIN

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

QL - SELECT-JOINS

ntax

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]

JOIN

Join clause is used to combine and retrive 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.

```
hive > SELECT c.ID, c.NAME, c.AGE, o.AMOUNT
FROM CUSTOMERS c JOIN ORDERS o
ON (c.ID = o.CUSTOMER_ID)
```

The following query executes JOIN on the CUSTOMER and ORDER table.

EFT OUTER JOIN

The following query demonstrate LEFT OUTER JOIN b/w CUSTOMER and ORDER table.

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

IGHT OUTER JOIN

The following query demonstrate RIGHT OUTER JOIN b/w CUSTOMER & ORDER t

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

Full OUTER JOIN

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

Database Statements

CREATE DATABASE/SCHEMA [IF NOT EXISTS] <database name>
ive > CREATE DATABASE [IF NOT EXISTS] userdb; Or
ive > CREATE ~~DATABASE~~ SCHEMA userdb;

HIVE - CREATE TABLE

Create Table Statement

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_nam
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]

LOAD DATA STATEMENT

Syntax: LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO
TABLE tablename
[PARTITION (partcol1 = val1, partcol2 = val2 ...)]

ALTER TABLE STATEMENT

Syntax: 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 ...])

Change statement

Syntax: hive > ALTER TABLE employee CHANGE name ename string;
hive > ALTER TABLE employee CHANGE salary salary Double;

ADD COLUMNS STATEMENT

Syntax: hive > ALTER TABLE employee ADD COLUMNS (
dept STRING COMMENT 'Department name');

REPLACE STATEMENT

syntax: hive > ALTER TABLE employee REPLACE COLUMNS (
eid INT empid INT,
ename STRING name String);

- DROP TABLE

Syntax : DROP TABLE [IF EXISTS] table_name;

HIVE - PARTITIONING

Hive organizes table 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 data.

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

ADDING a PARTITION

Syntax: ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec [LOCATION 'Location'] partition_spec [LOCATION 'location 2'] ...;

Partition_spec :

: (P_column = p_col_value, P_column = p_col_val, ...)

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

hive> ALTER TABLE employee
> ADD PARTITION (Year = '2013')
> location '/2012/part 2012';

RENAMING a PARTION

Syntax : 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');

DROPING 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');

BUILT-IN OPERATORS

There are 4 types of operators

* Relational operators
* Arithmetic operators
* Logical operators
* Complex operators