Unit 4 Hadoop Ecosystem Components

1. What is Hive?

Hive [Query Data Engine]

Hive is a data warehouse system (tool) which is used to analyze structured data in Hadoop. It is built on the top of Hadoop. It was developed by Facebook and is currently maintained by the Apache Software Foundation as Apache Hive

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), Data Manipulation Language (DML), and User Defined Functions (UDF).

Hive may be used to process both structured and sem-istructured data.

It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

File Format	Description
TextFile	Flat file with data in comma-, tab-, or space-separated value format or JSON notation.
CSV	
SequenceFile	Flat file consisting of binary key/value pairs.
RCFile	Record columnar data consisting of binary key/value pairs; high row compression rate.
ORC	Optimized row columnar data with stripe, footer, and postscript sections; reduces data size.
Parquet	Compressed columnar data representation.
Avro	Serialization system with a binary data format.
JSON	
HBase	

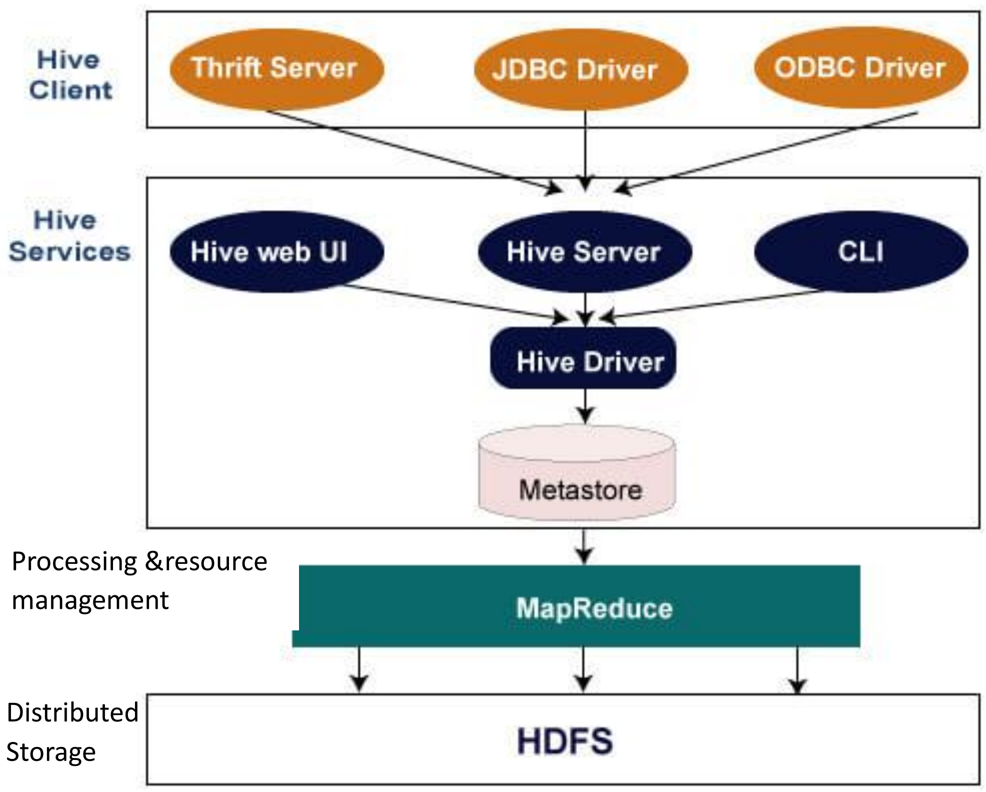
1. Hive Architecture

Apache Hive is a data warehouse software that facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At its core, Hive is designed to enable easy data summarization, ad-hoc querying, and analysis of large volumes of data.

The architecture of Hive is built on top of Hadoop and consists of several key components that work together to enable the processing of large datasets:

The major components of Apache Hive are: Hive Client, Hive Services, Processing and Resource Management, Distributed Storage.

The following architecture explains the flow of submission of query into Hive.



Hive Client

Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-

- Thrift Server - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
- JDBC Driver - It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
- ODBC Driver - It allows the applications that support the ODBC protocol to connect to Hive.

Hive Services

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 metastore is a component that stores metadata for Hive tables and partitions, including table schemas, column definitions, and partition locations.
- **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. Concept of session handles is implemented by driver
- **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.
  Execution plan with the help of the table in the database and partition metadata observed from the metastore are generated by the compiler eventually.
- **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.

3. Features of Hive

These are the following features of Hive:

- Hive is fast and scalable.
- It provides SQL-like queries (called HiveQL) 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, 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.
- In Hive, tables are used that are similar to RDBMS, hence easier to understand
- By using HiveQL, multiple users can simultaneously query data
- Hive supports a variety of data formats
- For running queries on the Hive, Hive supports applications written in any language, including Python, Java, C++, Ruby, and others, using JDBC, ODBC, and Thrift drivers.

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.

4. Operators in Hive

4.1Arithmetic Operators in Hive

Arithmetic operator accepts any numeric type. Following is the list of Operators:

Operator	Example	Description
+	A + B	This is used to add A and B.
-	A – B	This is used to subtract B from A.
*	A * B	This is used to multiply A and B.
/	A / B	This is used to divide A and B and returns the quotient of the operands.
%	A % B	This returns the remainder of A / B.
|	A | B	This is used to determine the bitwise OR of A and B.
&	A & B	This is used to determine the bitwise AND of A and B.
^	A ^ B	This is used to determine the bitwise XOR of A and B.
~	~A	This is used to determine the bitwise NOT of A.

Examples of Arithmetic Operator:

(i) It display salary by increasing 100

```
select salary + 100 from emp;
```

4.2 Relational Operators

In Hive, the relational operators are generally used with clauses like Join and Having to compare the existing records. The commonly used relational operators are: -

Operator		Description
= (equal to)	A=B	It returns true if A equals B, otherwise false.
<>, !- (not equal to)	A <> B, A !=B	It returns null if A or B is null; true if A is not equal to B, otherwise false.
< (less than)	A<B	It returns null if A or B is null; true if A is less than B, otherwise false.

> (grater than)	A>B	It returns null if A or B is null; true if A is greater than B, otherwise false.
<=	A<=B	It returns null if A or B is null; true if A is less than or equal to B, otherwise false.
>=	A>=B	It returns null if A or B is null; true if A is greater than or equal to B, otherwise false.
Is NULL	A IS NULL	It returns true if A evaluates to null, otherwise false.
IS NOT NULL	A IS NOT NULL	It returns false if A evaluates to null, otherwise true.

Examples of Relational Operator

(i) Display details of the employee having salary>=2000.

```
select * from emp where salary >= 2000;
```

5. Function in hive

5.1 **Mathematical Functions:**

Functions	**Return type**	**Description**	**example**	**output**
round(num)	BIGINT	It returns the BIGINT for the rounded value of DOUBLE num.		
floor(num)	BIGINT	It returns the largest BIGINT that is less than or equal to num.		
ceil(num)	BIGINT	It returns the smallest BIGINT that is greater than or equal to num.		
exp(num)	DOUBLE	It returns exponential of num.		
ln(num)	DOUBLE	It returns the natural logarithm of num.		
log10(num)	DOUBLE	It returns the base-10 logarithm of num.		
sqrt(num)	DOUBLE	It returns the square root of num.		

abs(num)	DOUBLE	It returns the absolute value of num.		
sin(d)	DOUBLE	It returns the sin of num, in radians.		
asin(d)	DOUBLE	It returns the arcsin of num, in radians.		
cos(d)	DOUBLE	It returns the cosine of num, in radians.		
acos(d)	DOUBLE	It returns the arccosine of num, in radians.		
tan(d)	DOUBLE	It returns the tangent of num, in radians.		
atan(d)	DOUBLE	It returns the arctangent of num, in radians.		

Example of Mathematical Functions:

(i) sqrt()

Fetch the square root of each employee's salary.

select Id, Name,sqrt(Salary) from empl ;

(ii) round() function

SELECT round(2.6) from temp;

On successful execution of query it will return:

3.0

(iii) select min(column_name) from table_name;

(iv) select max(Salary) from table_name;

(v) select floor(2.6); => 2

(vi) select count(*) from table_name;

5.2 String function:

function name	Return Type	Description
concat(string A, string B,...)	string	It returns the string resulting from concatenating B after A.
substr(string A, int start)	string	It returns the substring of A starting from start position till the end of string A.
substr(string A, int start, int length)	string	It returns the substring of A starting from start position with the given length.
upper(string A)	string	It returns the string resulting from converting all characters of A to upper case.
ucase(string A)	string	Same as above.
lower(string A)	string	It returns the string resulting from converting all characters of B to lower case.
lcase(string A)	string	Same as above.
trim(string A)	string	It returns the string resulting from trimming spaces from both ends of A.
ltrim(string A)	string	It returns the string resulting from trimming spaces from the beginning (left hand side) of A.
rtrim(string A)	string	rtrim(string A) It returns the string resulting from trimming spaces from the end (right hand side) of A.
ascii(string str)		Returns ASCII numeric value of the first character of the input argument.
character_length()		returns length of the string

Example

selectascii("ABC"); ==> Returns 65

selectcharacter_length("ABC"); ==> Returns 3

selectconcat("ABC","DEF"); ==> Returns ABCDEF

selectsubstr('ABCD',1,2);   on successful execution it will display : AB

6.0 Difference Between Hive and RDBMS

There are several differences between Apache Hive and a traditional relational database management system (RDBMS), including:

(i) Data validation
Hive follows the schema-on-read rule, which means that there is no data validation, checking, or parsing. In traditional databases, a schema is applied to a table that enforces a schema on a write rule.

(ii) Data access
Hive works on write once, read many times, while RDBMS functions work on read and write many times.

(iii) Data size
Hive is designed to work quickly on petabytes of data.

(iv) File formats
Hive supports various types of file formats such as textfile, ORC, Parquet, LZO Compression, etc.

(v) Architecture
Hive is built on top of Apache Hadoop, an open-source framework used to efficiently store and process large datasets.

(vi) Features

RDBMS typically includes features such as ACID support, multi-user access, data durability, data consistency, data flexibility, and hierarchical relationship

Here is a comparison between traditional RDBMS and Apache Hive:

Feature	Traditional RDBMS	Apache Hive
Data Model	Relational (Structured)	Schema-on-Read (Supports structured, semi-structured, unstructured)
Data Storage	Stored in tables with predefined schemas	Data stored in HDFS, supports large datasets
Query Language	SQL (Structured Query Language)	HiveQL (similar to SQL, but specific to Hive)
Schema Enforcement	Schema-on-Write (strict schema enforcement)	Schema-on-Read (flexible schema enforcement)
Transaction Management	ACID (Atomicity, Consistency, Isolation, Durability)	Limited ACID support (fully supported in newer versions)
Data Size	Typically for smaller datasets (terabyte)	Designed for very large datasets (petabyte-scale)
Performance	High performance for transactional queries	Optimized for read-heavy queries, batch processing, not real-time
Processing Engine	Local processing engines (like MySQL, Oracle, etc.)	Uses MapReduce, Tez, or Spark for distributed processing
Concurrency Control	Strong, supports many concurrent users	Limited concurrency compared to RDBMS

Data Partitioning	Manual partitioning	Supports automatic partitioning and bucketing
Use Case	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Scalability	Limited vertical scaling	Horizontally scalable across distributed systems
Fault Tolerance	Limited fault tolerance, dependent on hardware	Highly fault-tolerant due to HDFS
Data Latency	Low latency for real-time transactions	Higher latency due to batch processing
Joins	Efficient joins on structured data	Joins are expensive and slow due to MapReduce
Data Input	Requires structured and cleaned data	Can handle raw data, no strict data formatting
Storage Cost	Expensive due to use of high-end databases	Low-cost storage due to HDFS
Indexing Support	Strong indexing and optimization features	Basic indexing; less emphasis on indexing
Security	Advanced security features (authentication, authorization)	Basic security (Kerberos support, improving over time)

7. Data types:

Primitive Data Types:

1. Numeric Data types - Data types like integral, float, decimal
2. String Data type - Data types like char, string
3. Date/ Time Data type - Data types like timestamp, date, interval
4. Miscellaneous Data type - Data types like Boolean and binary

Complex Data Types:

1. Arrays - A collection of the same entities. The syntax is: array<data_type>
2. Maps - A collection of key-value pairs and the syntax is map<primitive_type, data_type>
3. Structs - A collection of complex data with comments. Syntax: struct<col_name :data_type [COMMENT col_comment],…..>
4. Units - A collection of heterogeneous data types. Syntax: uniontype<data_type, data_type,..

7.1 Primitive Data Types:

7.1.1 Numeric

(I) Integer Types

Type	**Size**	**Range**
TINYINT	1-byte signed integer	-128 to 127
SMALLINT	2-byte signed integer	32,768 to 32,767
INT	4-byte signed integer	2,147,483,648 to 2,147,483,647
BIGINT	8-byte signed integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

(ii) Decimal Type

Type	**Size**	**Range**
FLOAT	4-byte	Single precision floating point number
DOUBLE	8-byte	Double precision floating point number

7.1.2Date/Time Types

TIMESTAMP

- It supports traditional UNIX timestamp with optional nanosecond precision.
- As Integer numeric type, it is interpreted as UNIX timestamp in seconds.
- As Floating point numeric type, it is interpreted as UNIX timestamp in seconds with decimal precision.
- As string, it follows java.sql.Timestamp format "YYYY-MM-DD HH:MM:SS.fffffffff" (9 decimal place precision)

(3) DATES

The Date value is used to specify a particular year, month and day, in the form YYYY--MM--DD. However, it didn't provide the time of the day. The range of Date type lies between 0000--01--01 to 9999--12--31.

7.1.3 String Types

(i) STRING

The string is a sequence of characters. It values can be enclosed within single quotes (') or double quotes (").

(ii) Varchar

The varchar is a variable length type whose range lies between 1 and 65535, which specifies that the maximum number of characters allowed in the character string.

(iii) CHAR

The char is a fixed-length type whose maximum length is fixed at 255.

7.2 Complex Type

Type	**Size**	**Range**
Struct	It is similar to C struct or an object where fields are accessed using the "dot" notation.	struct('college','my')
Map	It contains the key-value tuples where the fields are accessed using array notation.	map('first','jmp','last','desai')
Array	It is a collection of similar type of values that indexable using zero-based integers.	array('college','my')

== =

8. Working with Tables and Databases

8.1 Create Database

In Hive, the database is considered as a catalog or namespace of tables. So, we can maintain multiple tables within a database where a unique name is assigned to each table.

Hive also provides a default database with a name **default**.

Each database must contain a unique name. If we create two databases with the same name, It generates error.

Syntax :

create database database_name;

Example: create database "demo"

```
create database demo;
```

It will create database and display

```
OK
No rows affected (0.254 seconds)
```

If we want to suppress the warning generated by Hive on creating the database with the same name, follow the below command: -

hive> create a database if not exists demo;

List out all database:

```
show databases;
```

Command name	**purpose of command**	**Example**
show databases;	display list of existing databases	
create database database_name;	create new database "database_name"	create database demo;
use database_name;	Select or switch to database database_name	use demo; The command sets demo as the currently active database.

8.2Create Table

In Hive, we can create a table by using the conventions similar to the SQL. It supports a wide range of flexibility where the data files for tables are stored.

It provides two types of table: -

(i) Internal table
(ii) External table

(i) Internal Table

- The internal tables are also called managed tables as the lifecycle of their data is controlled by the Hive.
- By default, these tables are stored in a subdirectory under the directory defined by hive.metastore.warehouse.dir (i.e. /user/hive/warehouse).
- The internal tables are not flexible enough to share with other tools.
- If we try to drop the internal table, Hive deletes both table schema and data.

Syntax:

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

[(col_namedata_type[COMMENT col_comment],...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]
```

{{{ extra

1. We can add a comment to the table as well as to each individual column.

2. ROW FORMAT DELIMITED shows that whenever a new line is encountered the new record entry will start.

3. FIELDS TERMINATED BY ‘,’ shows that we are using ‘,’ delimiter to separate each column.

4. We can also override the default database location with the LOCATION option.

}}}

Example:

Let's create an internal table stud with three fields

(i) no with data type integer

(ii) Name with data type string

(iii) percentage of float data type.

```
hive> create table demo.stud(no int, Name string , percentage float)  ;
```

Create table so that data can be imported from table:

```
hive> create table empdb.regdetails(id int, name string,salary float)
    > row format delimited
    > fields terminated by ',';
```

Display structure of table:

The metadata or structure of the created table by using the following command:-

Syntax:

```
describe table_name;
```

Example:

```
hive> describe demo.stud;
```

Output:

```
OK
no int
name string
percentage float
3 rows selected (0.3 seconds)
```

Creating table from existing table:

like: Creating a new table by using the schema of an existing table.

syntax:

```
create table newtablename like oldtablename;
```

Example:

```
hive> create table demo.stud1
like demo.stud;
```

here New table stud1 is a copy of an existing table stud.

Command work on table

Command name	**purpose of command**
like	Creating a new table by using the schema of an existing table
quit;	Quit hive prompt
show tables;	check the list of existing tables in the corresponding database
drop table table_name;	It is used to drop table in current selected database. Drop table

	table_name

show functions;	
CREATE FUNCTION	Register function in the Hive environment

9. Data Loading and Manipulation

Data Manipulation in HiveQL involves performing actions such as inserting, updating, deleting, and querying data in Hive tables.

Key Data Manipulation Operations in HiveQL:

(i) Querying Data
(ii) Inserting Data
(iii) Updating Data
(iv) Deleting Data

Loading Data into Hive table:

Apache Hive DML stands for (Data Manipulation Language) which is used to insert, update, delete, and fetch data from Hive tables. Using DML commands we can load files into Apache Hive tables, write data into the file system from Hive queries, perform merge operation on the table, and so on.

Hive tables provide us the schema to store data in various formats (like CSV).

Hive provides multiple ways to add data to the tables.

We can use DML (Data Manipulation Language) queries in Hive to import or add data to the table. One can also directly put the table into the hive with HDFS commands.

We can add data to the Hive table in 2 different ways.

1. Using INSERT Command
2. Load Data Statement

(1) Using INSERT Command

Insert static values into a Table

Syntax:

```
INSERT INTO table_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

OR

INSERT INTO table_name VALUES (value1, value2, value3, ...);
```

Example:

INSERT INTO stud VALUES (3,'Alex',80.2) ;

INSERT INTO stud VALUES (2,'Perry',84.3) ;

(II) Insert Data from a SELECT Query

syntax:

INSERT INTO TABLE table_name SELECT columns FROM source_table WHERE condition;

example:

insert into table exam select id,name from stud where id=1;

Insert into partition

Syntax:

INSERT INTO TABLE table_name PARTITION (partition_column=value [, partition_column2=value2, ...])

SELECT columns

FROM source_table

WHERE condition;

Example:

(2) Load data statement

While inserting data into Hive, it is better to use LOAD DATA to store bulk records.

There are two ways to load data: one is from local file system and second is from Hadoop file system.

The LOAD DATA statement is used to load data into the hive table. Load operations are pure copy/move operations.

LOAD DATA [LOCAL] INPATH '<filepath>' [OVERWRITE]

INTO TABLE <tablename> [PARTITION (partcol1=val1, partcol2=val2 ...)]

[INPUTFORMAT '<inputformat>' SERDE 'serde'

LOCAL	LOAD command will check for the file path in the local filesystem. Use LOCAL if you have a file in the server where the beeline is running
OVERWRITE	OVERWRITE switch allows us to overwrite the table data.
filepath	Supports absolute and relative paths. filepath would be referred from the server where hive beeline is running otherwise it would use the HDFS path.

PARTITION	Loads data into specified partition

- If the LOCAL keyword is not mentioned, then the Hive will need the absolute URI of the file. The hive will consider the location as an HDFS path location such as hdfs://namenode:9000/user/hive/project/data1.
- If the OVERWRITE keyword is mentioned, then the contents of the target table/partition will be deleted and replaced by the files referred by the file path.
- If the OVERWRITE keyword is not mentioned, then the files referred to by the file path will be appended to the table.
- If we try to load unmatched data (i.e., one or more column data doesn't match the data type of specified table columns), it will not throw any exception. However, it stores the Null value at the position of unmatched tuple.

Example:

```
load data local

inpath 'file:///D:/tables/emp.csv'

into table empdb.regdetails;
```

(i) Querying Data

Syntax:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...

FROM table_reference

[WHERE where_condition]

[GROUP BY col_list]

[HAVING having_condition]

[ORDERBY column1 [ASC|DESC], column2 [ASC|DESC], ...]

[LIMIT number];
```

DISTINCT	remove duplicate rows
WHERE	Filters the result set based on a condition.
GROUP BY	group data
HAVING	After grouping rows, you can use the HAVING clause to filter groups based on aggregate conditions:
SORT BY	sort the results by one or more columns in ascending (ASC) or descending (DESC) order
LIMIT	Restrict the Number of Rows

Example:

SELECT * FROM stud WHERE percentage>80;

Retrieve detail of all students whose percentage is more than 80

Drop table:

DROP TABLE command in the hive is used to drop a table inside the hive. Hive will remove all of its data and metadata from the hive meta-store.

Syntax:

DROP TABLE [IF EXISTS] table_name [PURGE];

PURGE option indicate that the data will be completely lost and cannot be recovered later but if not mentioned then data will move to .Trash/current directory.

Example:

drop table edetals;

If we try to drop the database that doesn't exist, the following error generates:

database does not exist

However, if we want to suppress the warning generated by Hive on creating the database with the same name, follow the below command:-
hive> drop database if exists demo;

it is not allowed to drop the database that contains the tables directly. In such a case, we can drop the database either by dropping tables first or use Cascade keyword with the command.

Let's see the cascade command used to drop the database:-

hive> drop database if exists demo cascade;

This command automatically drops the tables present in the database first.

Alter table:

We can perform modifications in the existing table like changing the table name, column name, comments, and table properties. It provides SQL like commands to alter the table.

Adding columns to Table:

Syntax:

ALTER TABLE table_name ADD COLUMNS (new_column_name data_type [COMMENT 'column_comment']);

Example:

alter table employee add columns(salary int);

Rename a Table

If we want to change the name of an existing table, we can rename that table by using the following signature: -

Syntax:

Alter table old_table_name rename to new_table_name;

Example:

alter table pdetails rename to perdetails;

10. Partitioning table

The partitioning in Hive means dividing the table into some related parts based on the values of a particular column like date, course, city or country.

The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

Using partition, it is easy to query a portion of the data.

It allows a user working on the hive to query a small or desired portion of the Hive tables.

As we know that Hadoop is used to handle the huge amount of data, it is always required to use the best approach to deal with it. The partitioning in Hive is the best example of it.

Let's assume we have a data of 10 million students studying in an institute. Now, we have to fetch the students of a particular course. If we use a traditional approach, we have to go through the entire data. This leads to performance degradation. In such a case, we can adopt the better approach i.e., partitioning in Hive and divide the data among the different datasets based on particular columns.

Suppose we have a table student that contains 5000 records, and we want to only process data of students belonging to the 'A' section only. However, the student table contains student records belonging to all the sections (A, B, C, D) but with partitioning, we do not need to process all those 5000 records. Here, partitioning helps us in separating data of students according to their sections. By doing so the time to execute the query will be increased, and we do not need to scan all the other unnecessary data available inside the 'student' table

The partitioning in Hive can be executed in two ways –

1. Static partitioning
2. Dynamic partitioning

Static Partitioning

In static or manual partitioning, it is required to pass the values of partitioned columns manually while loading the data into the table. Hence, the data file doesn't contain the partitioned columns.

Features of Static Partitioning

- Partitions are manually added so it is also known as manual partition
- Data Loading in static partitioning is faster as compare to dynamic partitioning so static partitioning is preferred when we have massive files to load.
- In static partitioning individual files are loaded as per the partition we want to set.
- where clause is used in order to use limit in static partition
- Altering the partition in the static partition is allowed whereas dynamic partition doesn’t support Alter statement.

```
hive> create table student(id int,name string,age int)
    > partitioned by (course string)
    > row format delimited
    > fields terminated by ',';


hive> describe student;
```

```
2024-10-14T20:44:36,971 INFO [main] org.apache.hadoop.hive.conf.HiveConf - Using the default value passed in for log id: 59cd8928-a2ad-4c97-8cea-b0a602185c56

2024-10-14T20:44:36,972 INFO [main] org.apache.hadoop.hive.ql.session.SessionState - Updating thread name to 59cd8928-a2ad-4c97-8cea-b0a602185c56 main

OK
id              int
name            string
age             int
course          string

# Partition Information
# col_name          data_type           comment
course              string
Time taken: 0.296 seconds, Fetched: 8 row(s)
```

Dynamic Partitioning

In dynamic partitioning, the values of partitioned columns exist within the table. So, it is not required to pass the values of partitioned columns manually.

Example:

```
hive> create table empdynamic(id int,name string,salary float)
    > partitioned by (dept string)
    > row format delimited
    > fields terminated by ','
    > stored as textfile;
```

Loading data into table:

```
hive> LOAD DATA LOCAL INPATH 'file:///D:/tables/studJava.csv'
    > INTO TABLE student
    > partition(course='Hadoop');
```

Fetching records

```
hive> select * from student;
```

NOTE:

- The tables created in hive are stored as A subdirectory under the database directory
- Using the ALTER DATABASE command in an database you can change the dbproperties
- the results of a hive query can be stored as Local File & HDFS file