Lecture Notes

Hive and Querying

Introduction to Hive

Hive was introduced to enable analysts to easily write high-level programming languages in Java or Python in order to execute MapReduce programs. Hive uses an SQL-like language known as the Hive Query Language (HQL), which makes it easy for analysts to execute MapReduce tasks on the HDFS.

What is Hive?

- **A software tool:** Hive is a software tool that is used for processing the data available in the HDFS. It runs over the Hadoop system to summarise big data and enables analysts to easily query and perform analysis.
- **SQL-like syntaxes:** Hive supports simple SQL-like syntaxes to query into the HDFS.
- **A lens between the HDFS and MapReduce:** Hive acts as a lens between the HDFS and MapReduce. When you write queries in Hive, it triggers the MapReduce job to convert these queries into MapReduce codes that can operate on the data available in the HDFS.
- **Multiple storage formats:** An interesting aspect of Hive is that it allows you to fetch or query into any type of file format available in the HDFS.

What is Hive not?

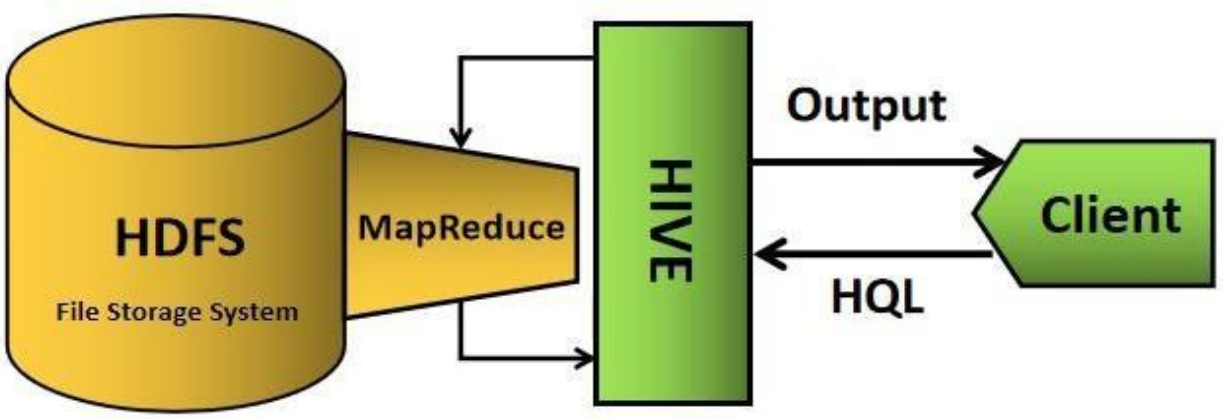
- **An analytical tool:** As Hive only supports OLAP systems, you can use it only for analytical purposes. It does not support real-time transactions in a database.
- **No record-level updates:** Generally, Hive allows you to only append data to a database; it does not support record-level updates such as delete, update and alter.
- **Limited by SQL syntaxes:** Since Hive supports only simple SQL-like syntaxes, you cannot perform analyses that are too complex.

Hive in OLA and Pinterest

You learnt how large companies like **Ola** and **Pinterest** are using Hive to derive valuable insights from the massive volumes of data that they collect from their customers. These companies use Hive for processing huge batches of data such as data over a period of a month, year and so on. Analytical processing involves generating summary reports, analysing historical data for finding trends and patterns, etc.

Features of Hive

- **Hive works on the HDFS:** Clients interact with the HDFS using Hive tools. They use the Hive Query Language (HQL) to query into the HDFS, which ultimately initiates a MapReduce job to fetch the required data. Note that Hive is not a storage system; it is merely a software tool that runs over the HDFS to query into the file storage system.



- **Single write, Multi-read:** As you learnt previously, Hive is an OLAP system that is designed to read data for analytical purposes. Hence, you can read the data as many times as you want, but you can perform write operations into the HDFS only once.
- **Hive Query Language:** Hive supports HQL for querying into the HDFS, although HQL is ultimately translated into MapReduce jobs internally. Hence, internally, Hive uses MapReduce and the HDFS.
- **Metadata:** Hive fetches data from the HDFS in the form of tables and stores the schema of the data in its metastore. Metadata is the data or information related to some given data.
- **Tabular structure:** In Hive, you read data into tables after creating their schema using HQL. There are two types of tables in Hive: internal and external.
- **Aggregation functions**: Since Hive is an OLAP system used for analytical purposes, it has a variety of aggregation functions such as sorting, average, sum, count, min and max.
- **User-defined functions (UDFs):** One of the dynamic features of Hive is user-defined functions. In addition to the inbuilt functions, Hive allows you to customise functions for processing records or groups of records. In many industries, it is necessary to perform various tasks using UDFs, which are more extensible than the existing functions.
- **Batch processing:** Consider a ride-hailing company, such as Uber, that needs to process the data of all the rides availed and distribute the revenue earned among the drivers accordingly at the end of each day. Now, processing such data at the end of each day, rather than after every ride, is an example of batch processing This is an example of batch processing.

  Batch processing in Hadoop refers to processing a group of transactions that have been collected

over a particular period of time. As in the case of the aforementioned example, the period of time is one day. Multiple rides are availed throughout the day, with money being transacted between riders and drivers/Uber, and at the end of the day, the money received from the day's transactions is finally allocated to the drivers.

- **Highly extensible:** Hive is highly extensible. If a particular application is written in Java or Python, then it can interact with Hive and can fetch the relevant data from the HDFS.

Use Cases of Hive

Some of the use cases of Hive are as follows:

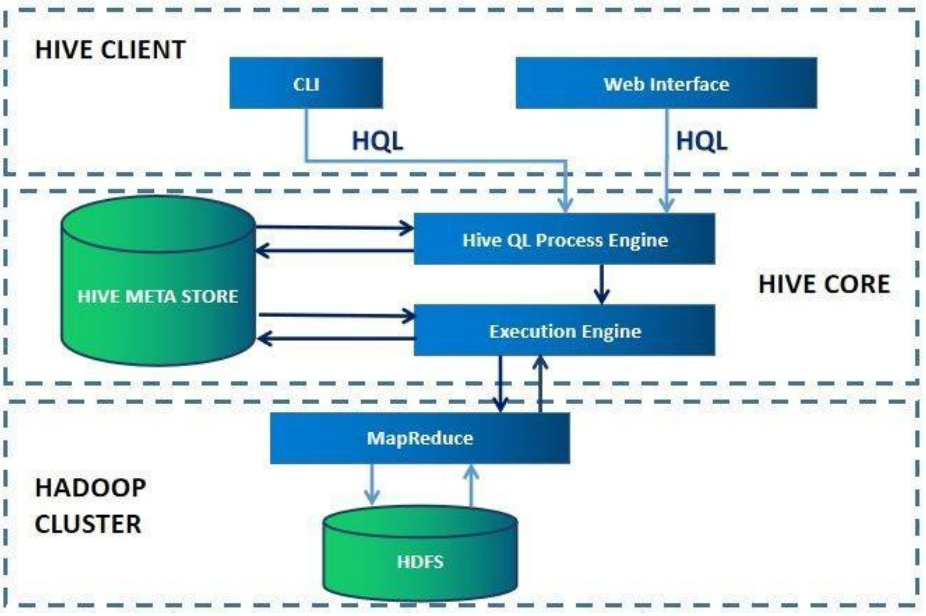
- **Reporting:** Consider a mutual funds company where the average, minimum or maximum price of stocks needs to be calculated at the end of each day. This process of running scheduled reports in a particular period of time is known as reporting.

- **Ad-hoc analysis:** Let's take the example of the mutual funds company again. Here, you are not required to present the report of stock prices at the end of each day. Now, suppose you want to know the current price of a group of shares and predict the price of the stocks after every hour, based on the rise or fall in the share price. This process of querying into a database for a particular ad-hoc demand is known as ad-hoc analysis.

- **Machine learning:** As you have learnt previously, Hive is an OLAP system where you conduct data analysis and build machine learning algorithms to derive meaningful insights from the given data.

Architecture of Hive

The diagram given below depicts the architecture of Hive.



The Hive ecosystem is divided into the following parts:

1. **Hive client:** A Hive client acts as the interface to the Hive system through which users interact with Hive and query into the HDFS. It consists of the following parts:
    a. **Command-line interface (CLI):** This is an interface between Hive and its users. You need to write queries in HQL to derive insights from the data available in the HDFS. The CLI acts as the command-line tool for the Hive server.
    b. **Web interface:** This is the graphical user interface of Hive. It is an alternative to the Hive CLI, wherein you query into the HDFS using the Hive tool.

2. **Hive core:** The Hive core is the core part of the Hive architecture; it links the Hive client and the Hadoop cluster. It consists of the following three parts:
    a. **Hive QL process engine:** When you write a query in HQL via the CLI or the web interface, it moves into the Hive QL process engine, which checks the syntax of the query and analyses it. HQL is similar to SQL in terms of querying, and it uses the schema information available in the metastore.

    b. **Hive metastore:** The Hive metastore can be considered the heart of the Hive ecosystem. Let's try to understand this with the help of an example. Suppose an e-commerce company wants to manage some of its units such as employees, customers and orders. Now, the data about these aspects is available in the HDFS and you need to perform analytics on it.

    Information about such tables and the database is known metadata.

    It is important to note that Hive is not a database, and the metastore acts as a storage space for Hive for storing the schema information of all the tables that have been created. The metastore does not store the actual data; instead, it stores the metadata of the tables that have been created. It has its own embedded RDBMS for storing relational tables and uses Apache's 'Derby' RDBMS for this purpose.

    c. **Execution engine:** An execution engine is used for converting Hive queries to MapReduce jobs. The engine processes the queries and generates results that match with those generated by MapReduce.

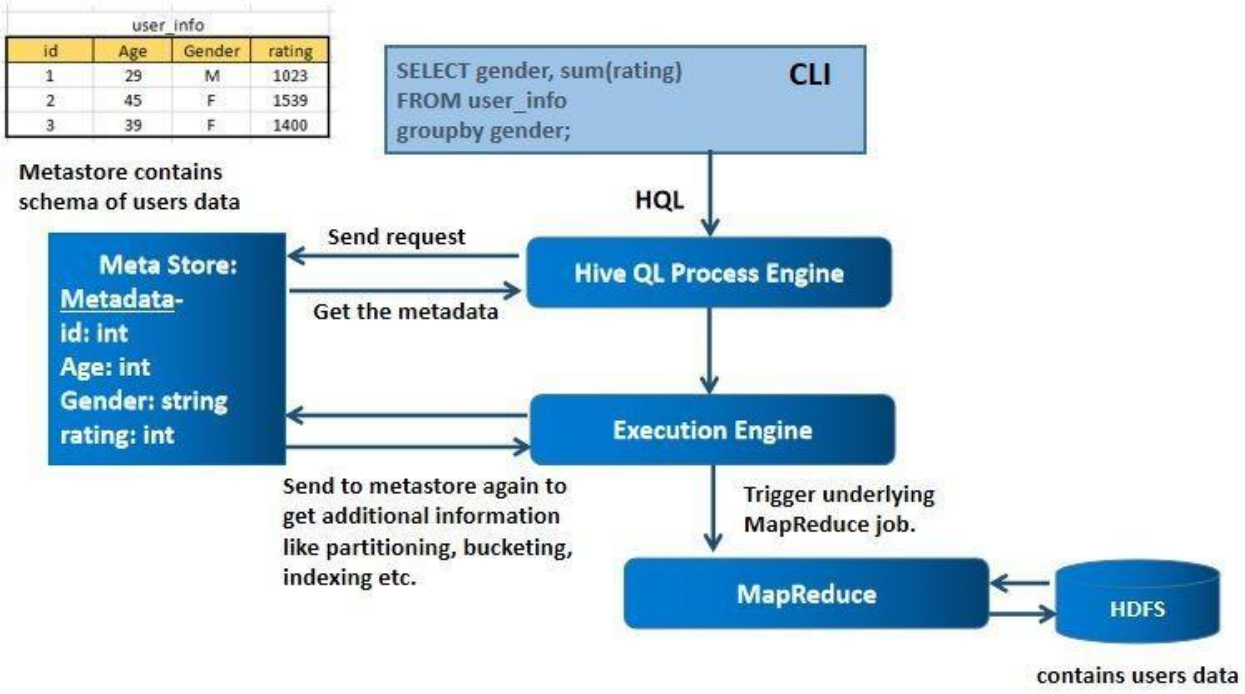
3. **Hadoop cluster:** The Hadoop cluster is a file storage system where users can store files in either a structured or an unstructured format. They can query into the HDFS using MapReduce.

It is the bottom-most block of the Hive architecture. It consists of the following two parts:

- **HDFS:** The Hadoop Distributed File System (HDFS) allows you to store large volumes of data across multiple nodes in a Hadoop cluster. It is used for performing distributed computing on structured or unstructured data.
- **MapReduce:** This is a programming framework that is used by Hadoop for processing data. After users store data in the HDFS, they use MapReduce to query into the HDFS to perform data

analysis. Note that even if you are running a Hive job, a MapReduce job is triggered in the background to query into the HDFS.

The example given below shows how a Hive query is executed and how it triggers a MapReduce job.



Hive	RDBMS
• Hive works on the '**Schema on READ**' principle. • In Hive, the entire data dump is available in the HDFS in a raw format. In order to read the data from the HDFS, you need to create a schema. • Since Hive is used for OLAP purposes, you need to create a schema every time you want to read the data stored in the HDFS. The schema is stored in the Hive metastore. Note that Hive is not a database. Here, data is stored only in the HDFS;Hive only stores the schema information of the tables that are defined by the user.	• An RDBMS works on the '**Schema on WRITE**' principle. • In an RDBMS, you need to define the schema of the tables before storing the data in the database. Hence, it is known as 'schema on write'.
• Since Hive supports OLAP systems, record-level updates are generally not possible. Also, Hive only allows you to append the data, whereas an RDBMS supports functions such as alters, delete and drop.	• An RDBMS supports record-level or row-level updates.
• Hive is used for dealing with petabytes of data.	• An RDBMS is used for dealing with terabytes of data.
• Due to the large data size, Hive takes a long time to compute. • Another reason for slow computation is that Hive supports 'schema on read', which means every time you want to read the given data, you need to match it with the schema.	• In an RDBMS, data is stored on the basis of the predefined schema; hence, computation takes less time.

Lecture Notes

Basic Hive Query

In this session, you learnt about some of the common Hive syntaxes and query optimisation techniques.

Database Creation

Note that all Hive demonstrations were performed on an CDH instance.

Since you worked with a data set in an S3 bucket, you can use the following two methods to load the data from S3:

1. You can load the data stored in S3 into Hive directly through the S3 link. However, any changes made to the bucket will cause changes while loading the data in Hive.
2. You can copy the data from S3 into the cluster and then load it into the Hive tables. This will help you keep your data safe within the cluster.

Some of the commonly used queries for creating databases are as follows:

- You can write the following query to create a database:
  create database if not exists demo ;.
  This 'demo' database is created in the Hive warehouse itself.
- You can write the following query with the keyword 'extended' to print the description of the database:
  describe database extended demo2 ;.
- If you want to view the database's headers, then you will need to set the headers to 'true' as follows:
  set hive.cli.print.header=true ;.

Internal and External Table I

There are two types of tables in Hive. These are as follows:

- Internal tables
- External tables

The properties of both internal and external tables are described below.

- **Creation of tables:** By default, any table in Hive is an internal table. To create an external table, you need to mention the keyword '**EXTERNAL**' explicitly while writing the query for creating the table.

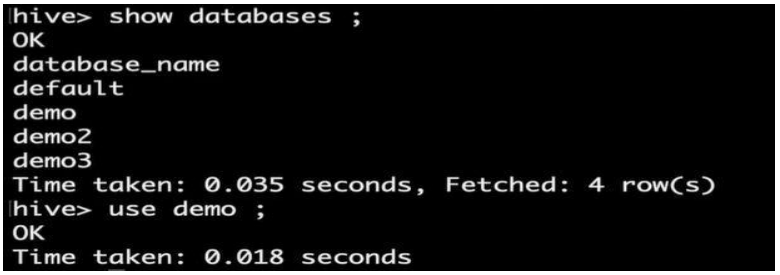
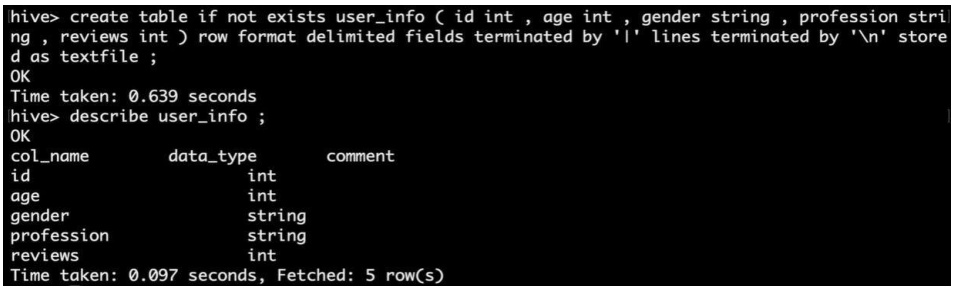
- **Internal table vs external table:** The basic difference between internal and external tables is that when you create an internal table, Hive itself is responsible for the data, and takes control of the entire life cycle of the metadata as well as the actual data.

  In contrast, when you create an external table, Hive is responsible for handling the table schema only, not the actual data.

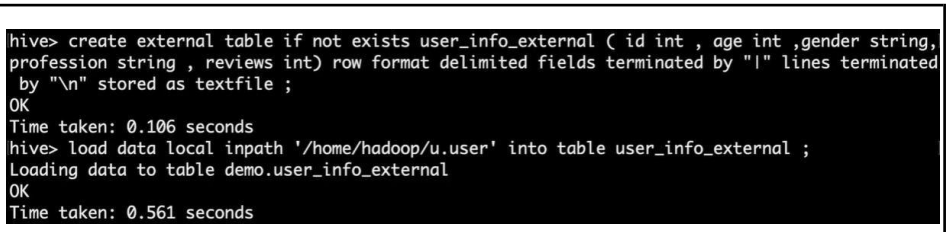
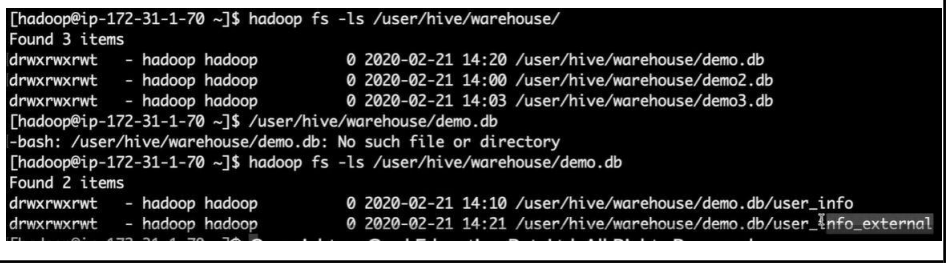
- **Dropping internal and external tables**: When you apply the 'drop' command on an internal table, the entire metadata available in the metastore, as well as the actual table data in the HDFS, gets deleted.

  However, when you apply the 'drop' command on an external table, the entire metadata available in the metastore gets deleted, but the actual table data in the HDFS remains intact.

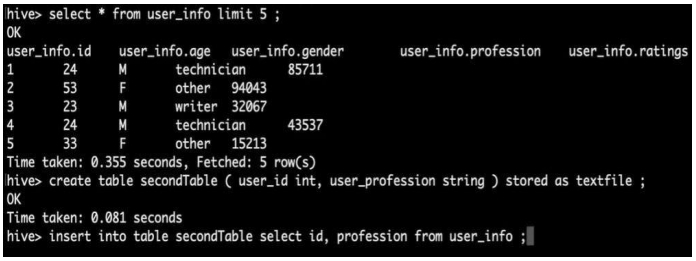
- **When to use internal and external tables:** Suppose there is a particular table that is quite common and can be used by multiple applications. In such cases, it is preferable to use an external table, because when you drop the table, the data will remain intact in the HDFS, and the external table can be used by some other applications. However, when you drop an internal table, the schema as well as the table data get deleted, and they become unavailable for other applications as well.

Selecting a database	
Creating an internal table	 By using the query given above, you are specifying that in the data available in the HDFS, fields are delimited by '|' and lines are terminated by '\n'.
Loading data into the tables	You can use the following query to load data into the table: **load data local** inpath '/root/movie_ratings/u.user' **into table** user_info;

Internal and External Table II

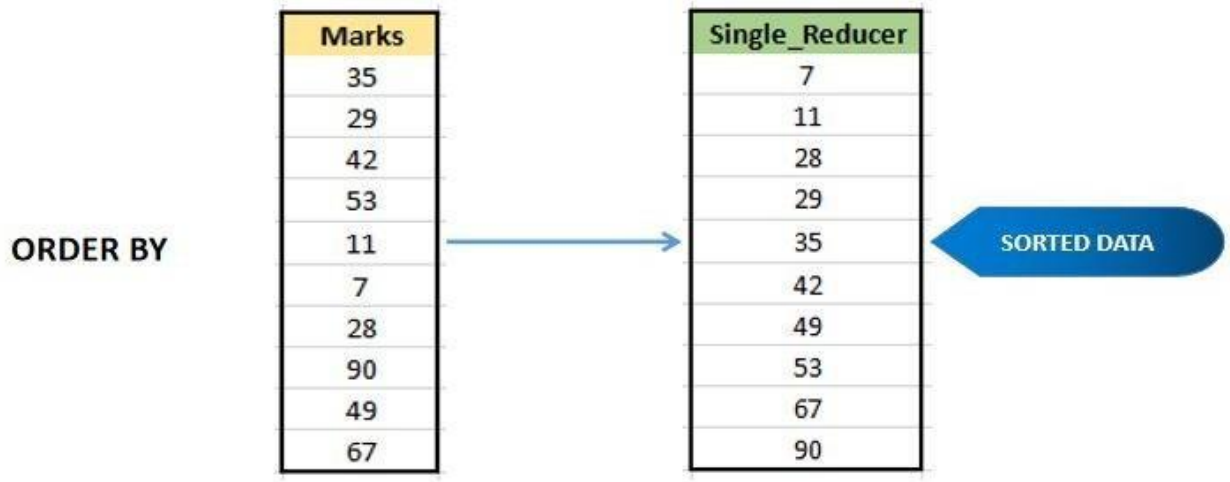
Creating an external table	
Location of the external table	

Operations on Tables

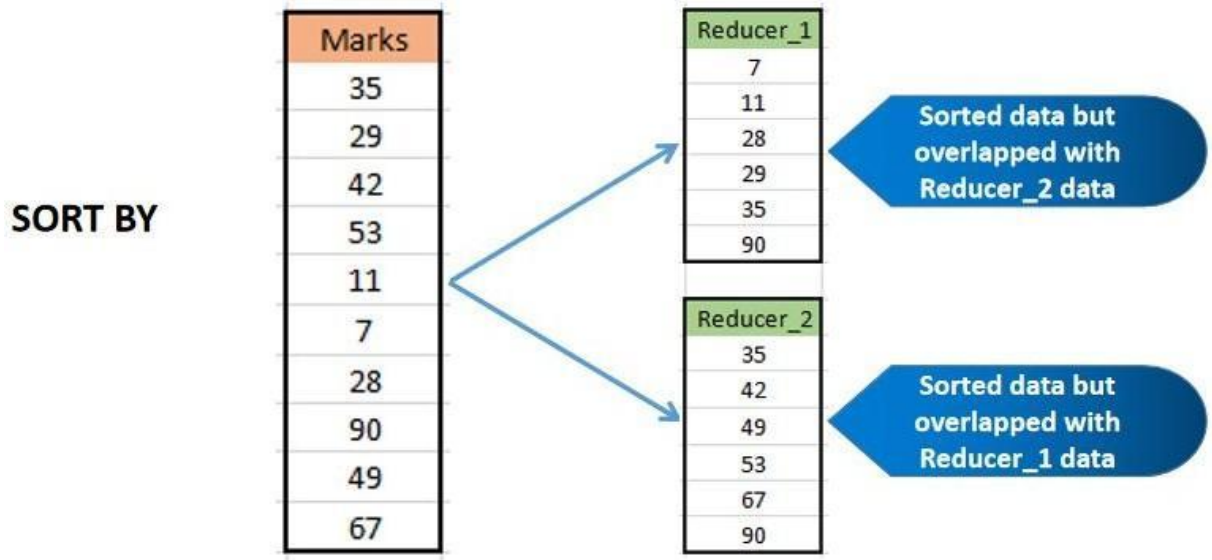
Inserting data from an existing table into a newly created table	
Changing the position of the 'gender' column from second to last by using the '**alter**' command	

ORDER BY and SORT BY

The ‘ORDER BY’ clause provides a complete sorted list of data and guarantees total ordering in the output. It may use multiple mappers, but it uses only one reducer for this purpose.

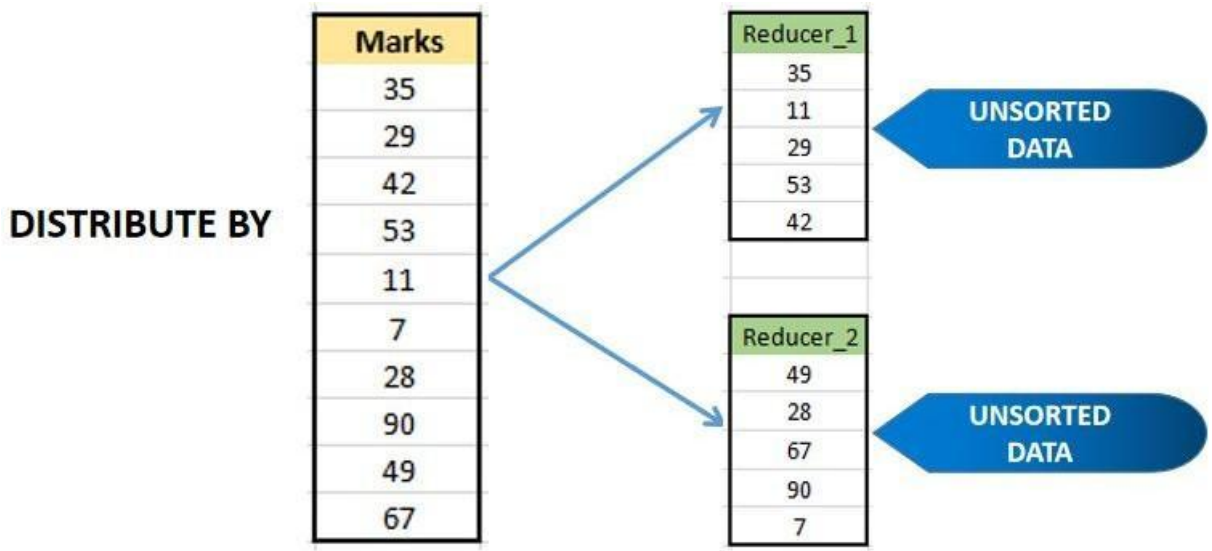


The ‘SORT BY’ clause sorts the data at the reducer level only. It does not guarantee total ordering of the data, as performed by the ‘ORDER BY’ clause. This command does not provide a complete sorted list. Also, there may be some overlapping data in the two reducers.

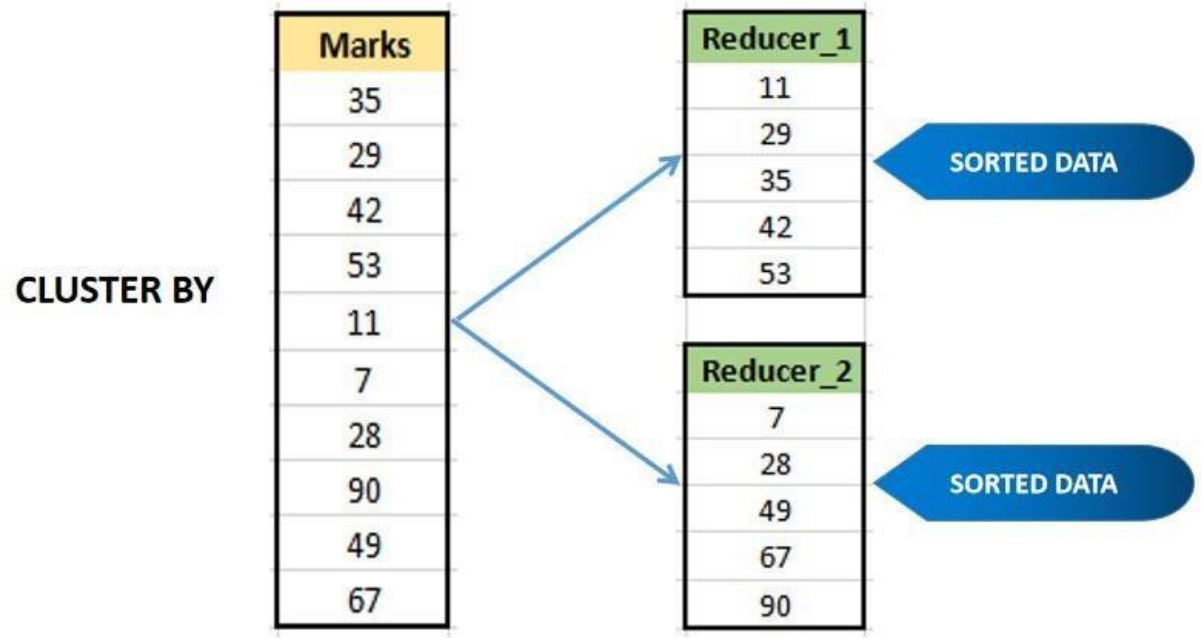


DISTRIBUTE BY and CLUSTER BY

The “DISTRIBUTE BY’ clause is used for distributing data among multiple reducers. Here, the data is not sorted into each reducer; the command merey distributes it among the reducers.

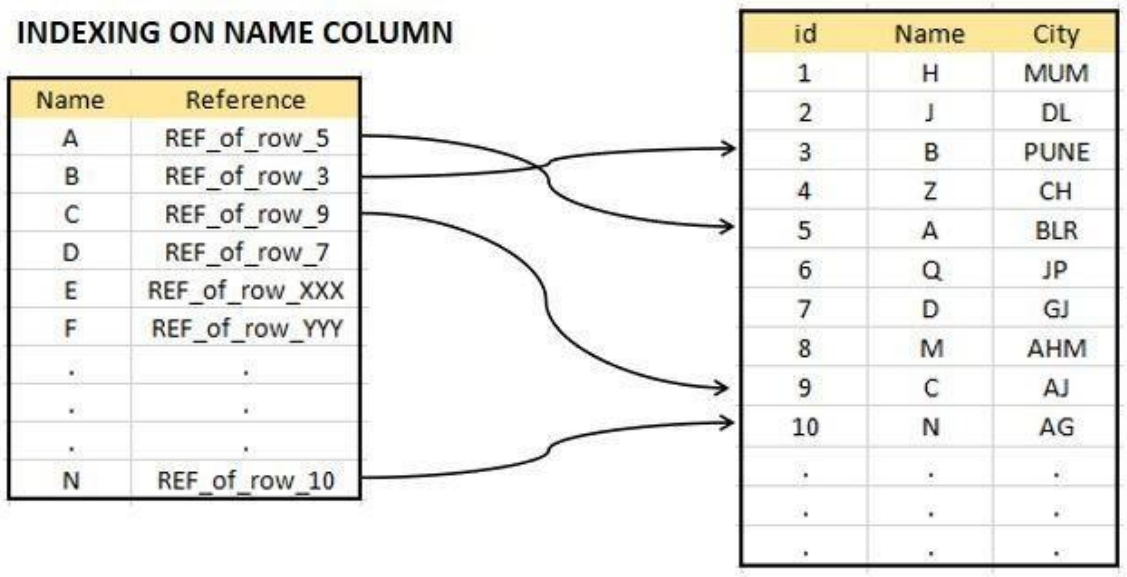


On the other hand, the ‘CLUSTER BY’ clause distributes data among multiple reducers, with each reducer containing sorted and non-overlapping data.



Thus, you can conclude that the ‘CLUSTER BY’ clause is a combination of the ‘DISTRIBUTE BY’ and “SORT BY’ clauses.

Indexing I



As you can see in the image given above, indexing has been done on the 'Name' column. Each name has been added in the sorted order to the separate data structure, and there is a reference corresponding to each name, which represents the rows in the original table.

So, when you search for a particular name, indexing uses the separate data structure and applies the binary search algorithm on it, and then returns the address of the row in the original table and returns the results.

The two types of indexing in Hive are as follows:

- Compact indexing
- Bitmap indexing

You can read more about the two types of indexing by referring to the link provided below.

Types of Indexing

You can use indexing in the following scenarios:

- When the data set is quite large
- When you want to reduce query execution time
- While applying indexing on some columns that are used more frequently than other columns
- For read-heavy applications, wherein you need to read data more frequently

Generally, it is not preferable to use indexing in write-heavy applications because all new updates have to be done in the separate data structure internally every time new data is inserted in the table. Simultaneously, a separate data structure must remain intact in the sorted state every time you add new data to the table, which is a time-consuming task.

Indexing II

- **COMPACT Keyword:** Using this keyword, you can specify the type of indexing that you want to to use for a particular column. For bitmap indexing, you need to use the BITMAP keyword.
- **Deferred rebuild:** The 'COMPACT' keyword is used to defer the process of indexing to a future time point, which means indexing is currently not active. To activate indexing, you need to write the following query separately: ***alter index i1 on user_info rebuild ;***

User-Defined Functions

The steps involved in creating user-defined functions (UDFs) are as follows:

- Create a Java program
- Save the Java file as a 'jar' file
- Add the 'jar' file to Hive. Use the following command for this purpose:
  add jar /home/hadoop/udfDemo.jar ;.
- Create a function of the 'jar' file that you have added
- Use that particular function in the Hive query

Advanced Hive Queries

In this session, you learnt about advanced Hive queries, which included map joins, partitioning, bucketing, etc.

Joins in Hive

In the case of joins, both the mapper and the reducer work simultaneously. If you can find a way to remove the reducer from the operation before performing the join using only the mapper, then you can decrease the query time. This entire process is known as ‘Map Join’.

Suppose there are two tables: ‘employee’ and ‘department’. Both of these tables contain columns related to the employees and departments of a company, respectively.

```
hive> desc employee ;
OK
emp_id                  int
emp_name                string
designation             string
salary                  int
mgr_id                  int
dept_id                 int
code                    string
Time taken: 0.02 seconds, Fetched: 7 row(s)
hive> desc department ;
OK
dep_id                  int
dep_name                string
dep_city                string
code                    string
Time taken: 0.025 seconds, Fetched: 4 row(s)
```

You can use the following query to perform a left join on the ‘employee’ and ‘department’ tables.

```
hive> select employee.emp_id , employee.emp_name , employee.designation , department.dep_id , department.dep_name , dep
artment.dep_city from employee left join department on ( employee.dept_id = department.dep_id ) ;
```

On executing the query, the table would look like this:

```
1281    Shawn   Architect       10      INVENTORY       HYDERABAD
1381    Jacob   Admin   20      Jacob   ACCOUNTS
1481    flink   Mgr     10      INVENTORY       HYDERABAD
1581    Richard Developer       NULL    NULL    NULL
1681    Mira    Mgr     10      INVENTORY       HYDERABAD
1781    John    Developer       10      INVENTORY       HYDERABAD
Time taken: 32.81 seconds, Fetched: 6 row(s)
```

Using a Map join, you can remove the reducer from the join operation and then perform the operation using only the mapper, thus reducing the query time. To perform a map join, you need to specify the hint to the above join query as shown in the query given below:

```
hive> select /*+ MAPJOIN (employee) */ employee.emp_id , employee.emp_name , employee.designation , department.dep_id ,
 department.dep_name , department.dep_city from employee left join department on ( employee.dept_id = department.dep_id
 ) ;
```

Now, without the map join, the query would take 32.81 seconds to execute, but after using map join, the query time would be reduced to 29.617 seconds.

```
1281    Shawn   Architect       10      INVENTORY       HYDERABAD
1381    Jacob   Admin   20      Jacob   ACCOUNTS
1481    flink   Mgr     10      INVENTORY       HYDERABAD
1581    Richard Developer       NULL    NULL    NULL
1681    Mira    Mgr     10      INVENTORY       HYDERABAD
1781    John    Developer       10      INVENTORY       HYDERABAD
Time taken: 29.617 seconds, Fetched: 6 row(s)
```

Static Partitioning

There are two types of partitioning in Hive. These are as follows:

- **Static partitioning:** In static partitioning, you need to load the data manually into the partitions.
- **Dynamic partitioning:** In dynamic partitioning, data is allocated to the partitions automatically.

You can apply static partitioning to a table using the following syntax:

```
hive> create table if not exists part_user_info ( id int , age int , gender string , ratings int ) partitioned
 by ( profession string ) row format delimited fields terminated by '|' lines terminated by '\n' ;
OK
```

Now, you can manually add the data related to ‘profession’ equals to ‘engineer’ to the partitions using the following syntax:

```
hive> insert into table part_user_info partition( profession='engineer') select id , gender , age ,ratings fro
m user_info where profession='engineer' ;
```

Dynamic Partitioning and Drop the Partitions

In dynamic partitioning, data is allocated to the partitions automatically. You can apply static partitioning to a table using the following syntax:

- By default, Hive does not allow dynamic partitioning; you need to define 'dynamic partitioning' equals 'true' while writing the syntax, as shown below:

```
hive> set hive.exec.dynamic.partition=true ;
hive> set hive.exec.dynamic.partition.mode= nonstrict ;
```

- The code snapshot given below shows how a dynamically partitioned table named 'dyn_part_user_info' is created and how data is loaded into it using the 'insert' clause:

```
hive> create table if not exists dyn_part_user_info ( id int , age int , gender string, ratings int) partitioned by (pr
ofession string ) row format delimited fields terminated by '|' lines terminated by '\n' ;
OK
Time taken: 0.778 seconds
hive> insert into table dyn_part_user_info partition( profession) select id , age , gender , ratings , profession from
user_info ;
```

- Once you have loaded the data into the dynamically partitioned table 'dyn_part_user_info', 'profession' is allocated to multiple partitions automatically. These partitions are located in the '**user/hive/warehouse/dyn_part_user_info**' directory in Hadoop.

- You can view the partitions by running the following command:

```
hive> show partitions part_user_info ;
OK
profession=doctor
profession=engineer
```

- You can drop the partition 'doctor' using the following command:

```
hive> alter table part_user_info drop partition( profession='doctor');
Dropped the partition profession=doctor
OK
Time taken: 0.249 seconds
hive> show partitions part_user_info ;
OK
profession=engineer
```

Partitioning is ideal when:

1. The partitions are of comparable sizes, and
2. The number of partitions is limited.

Bucketing

Bucketing allows you to divide data into partitions of equal size. It is preferred to partitioning when the number of partitions is highly unbalanced, or there are too many partitions, resulting in over-partitioning. In general practice, you create the partitions first and then perform bucketing on the data.

You can apply bucketing on data using the following syntax:

- Consider the Movie Ratings data set that we discussed earlier. Initially, partitions were created on the basis of gender. Now, for each gender, you can create seven buckets on the basis of 'age' in the 'buck_user_info' table using the following syntax:

```
hive> create table if not exists buck_user_info ( id int , age int , profession string , ratings int )  partitioned by
( gender string ) clustered by ( age ) into 7 buckets row format delimited fields terminated by '|' lines terminated by
 "\n" stored as textfile ;
OK
Time taken: 0.156 seconds
```

- Once you have created the tables, the data gets loaded into them. You can copy the data from the 'user_info' table to the 'buck_user_info' table using the following query:

```
hive> insert into table buck_user_info partition(gender) select id , age , profession , ratings , gender from user_info
  ;
```

- Now, let's revise the memory locations in Hadoop where these buckets have been created.

    1. Let's first look at the partitions that were created on the basis of 'gender' in the following locations:

```
[hadoop@ip-172-31-81-61 ~]$ hadoop fs -ls /user/hive/warehouse/buck_user_info
Found 2 items
drwxrwxrwt   - hadoop hadoop          0 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F
drwxrwxrwt   - hadoop hadoop          0 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=M
```

    So, there are two partitions that have been created on the basis of 'gender': 'M' and 'F'.

    2. You can check the memory locations of the buckets for the partitions of female gender using the following commands:

```
[hadoop@ip-172-31-81-61 ~]$ hadoop fs -ls /user/hive/warehouse/buck_user_info/gender=F
Found 7 items
-rwxrwxrwt   1 hadoop hadoop       1029 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000000_0
-rwxrwxrwt   1 hadoop hadoop       1000 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000001_0
-rwxrwxrwt   1 hadoop hadoop        980 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000002_0
-rwxrwxrwt   1 hadoop hadoop        640 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000003_0
-rwxrwxrwt   1 hadoop hadoop        763 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000004_0
-rwxrwxrwt   1 hadoop hadoop        769 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000005_0
-rwxrwxrwt   1 hadoop hadoop        782 2020-02-22 08:12 /user/hive/warehouse/buck_user_info/gender=F/000006_0
```

    As you can see in the memory locations above, seven files were created in the folder

containing the partitions for female gender. These files are generated on the basis of bucketting over ‘age’.

One key difference between bucketing and partitioning is that when you create partitions, new folders are created for each partition; however, in the case of bucketing, multiple files (buckets) are created inside a single folder.