BIG DATA ANALYTICS

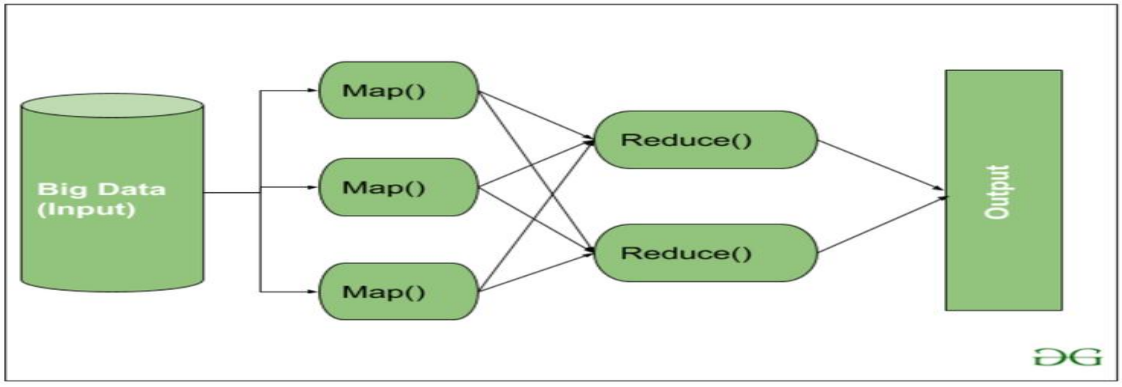
UNIT – II

Anatomy of Hadoop map-reduce(Input Splits, map phase, shuffle, sort, combiner, reduce phase) (theory)

Hive: Introduction to Hive, data types and file formats, HiveQL data definition(Creating Databases and Tables),HiveQL for Data loading, HiveQL data manipulation, Logical joins, Window functions, Optimization, Table partitioning, Bucketing, Indexing, Join Strategies.

Hadoop Map-Reduce :

- **Hadoop MapReduce** is a programming model or framework which is used for for data processing. It processes the huge amount of structured and unstructured data stored in HDFS.Hadoop can run Map Reduce programs written in Java, Ruby and Python. MapReduce processes data in parallel by dividing the job into the set of independent tasks. So, parallel processing improves speed and reliability.In Map Reduce programming, Jobs(applications) are split into a set of **map tasks and reduce tasks.**



- **MAP TASK:-**Map task takes care of loading, parsing, transforming and filtering data . It Converts input into key-value pairs. Each map task is broken down into the following phases:

  1. Record Reader

  2. Mapper

  3. Combiner

  4.Partitioner.

- **REDUCE TASK:-**The responsibility of reduce task is grouping and aggregating data that is produced by map tasks to generate final output. It Combines output of mappers as the input to Reduce task and produces a reduced result set. The **reduce tasks are broken down into the following phases:**

  1. Shuffle

  2. Sort

  3. Reducer

  4. Output format.

- There are two **Daemons** associated with Map-Reduce Programming:
  - Job Tracker and
  - Task Tracer

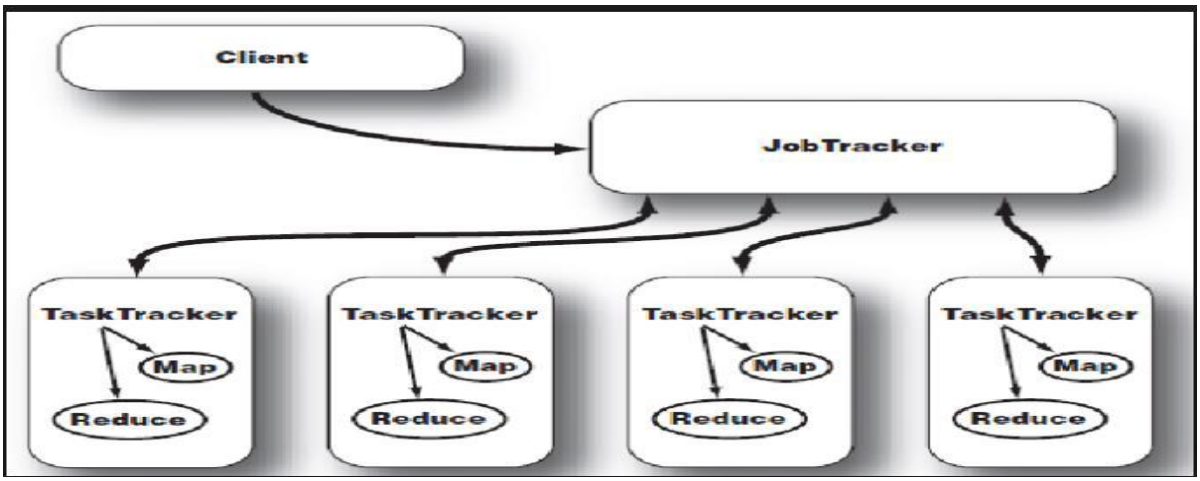
Job Tracker: Job Tracker is a **master**daemon. The work of Job tracker is to manage all the resources and all the jobs across the cluster and also to schedule each map on the Task Tracker running on the same data node since there can be hundreds of data nodes available in the cluster.

- It provides connectivity between Hadoop and application.

- Whenever code submitted to a cluster, Job Tracker creates the execution plan by deciding which task to assign to which node.
- It also monitors all the running tasks. When task fails it automatically re-schedules the task to a different node after a predefined number of retires.
- There will be one job Tracker process running on a single Hadoop cluster.

Task Tracker: Task Tracker is known as **Slave** daemon.

- The Task Tracker can be considered as the actual slaves that are working on the instruction given by the Job Tracker. This Task Tracker is deployed on each of the nodes available in the cluster that executes the Map and Reduce task as instructed by Job Tracker.
- This is responsible for executing individual tasks that is assigned by the Job Tracker.
- Task Tracker continuously sends **heartbeat** message/signal to job tracker. **Heartbeat** is the signal that is sent by the Task tracker to the Job Tracker after the regular interval to time to indicate its presence, i.e. to indicate that it is alive.
- When a job tracker fails to receive a heartbeat message from a Task Tracker, the Job Tracker assumes that the Task Tracker has failed and resubmits the task to another available node in the cluster.



Map Reduce Framework

Phases:

Map: Converts input into key-value pairs.

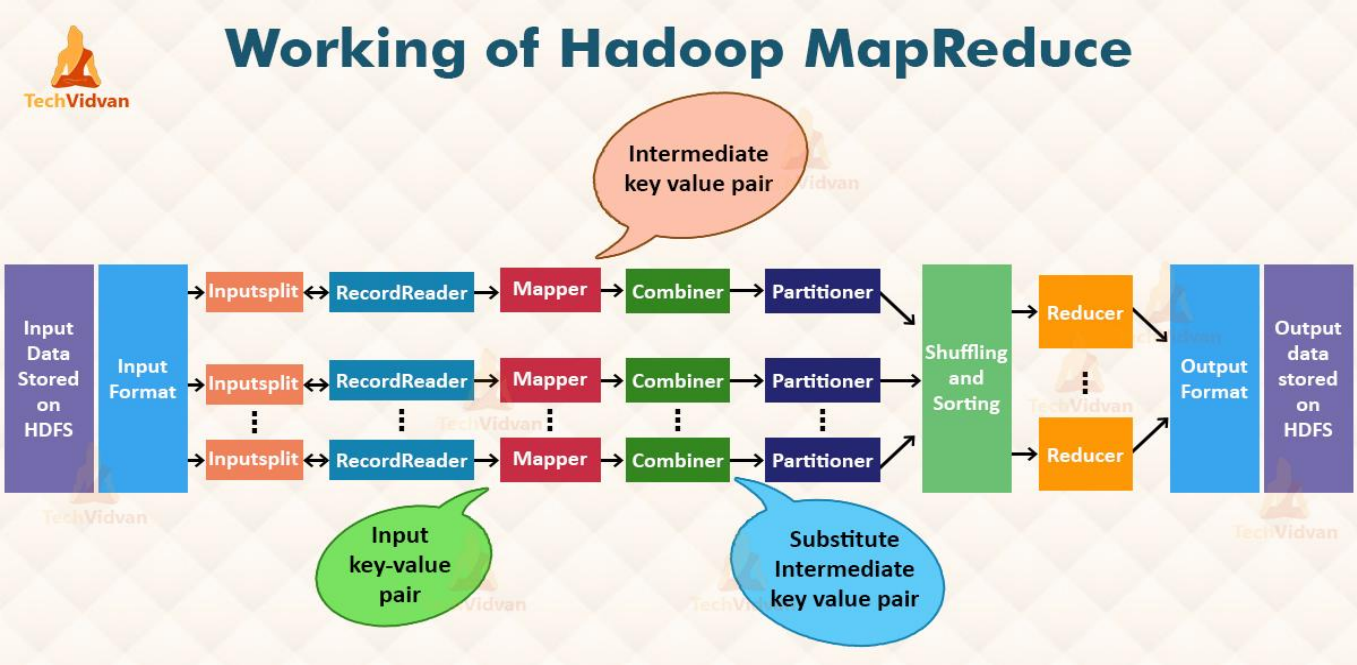
Reduce: Combines output of mappers and produces a reduced result set.

Daemons:

Job Tracker: Master, Schedules Task

Task Tracker: Slave, Execute task

Working of Hadoop Map Reduce:(Input Splits, map phase, shuffle, sort, combiner, reduce phase)



1. Input Files:

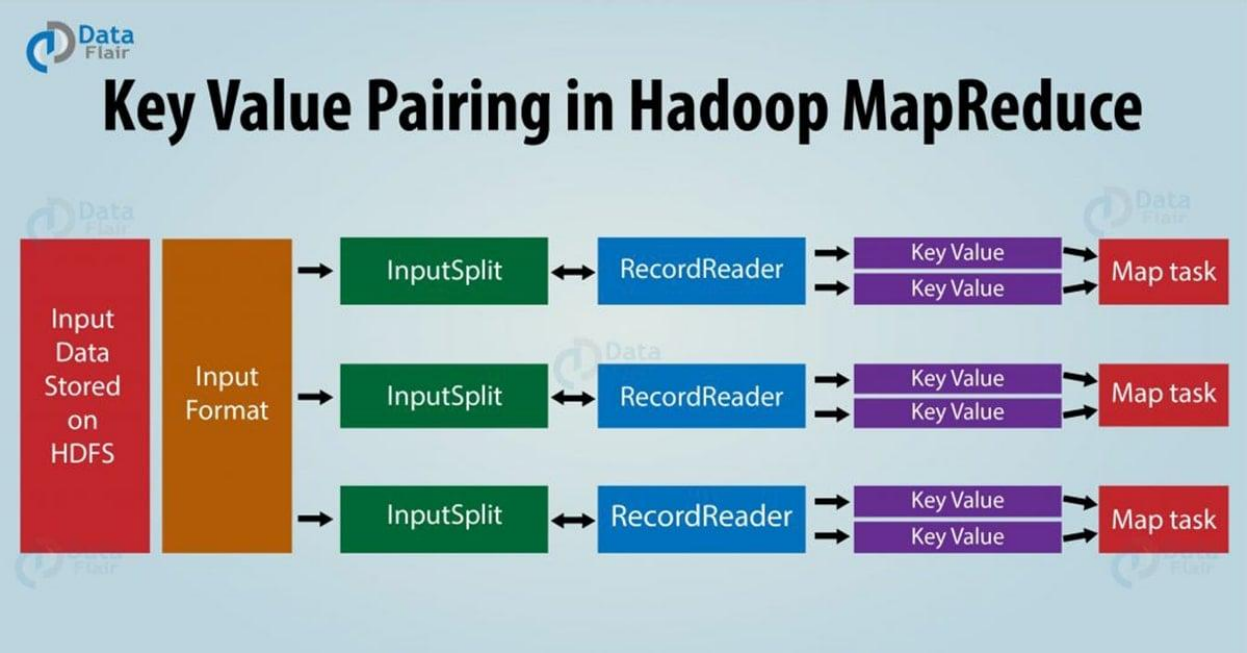
In input files data for MapReduce job is stored. In **HDFS**, input files reside. Input files format is arbitrary.

2. InputFormat:

After that InputFormat defines how to split and read these input files. It selects the files or other objects for input. InputFormat creates InputSplit.

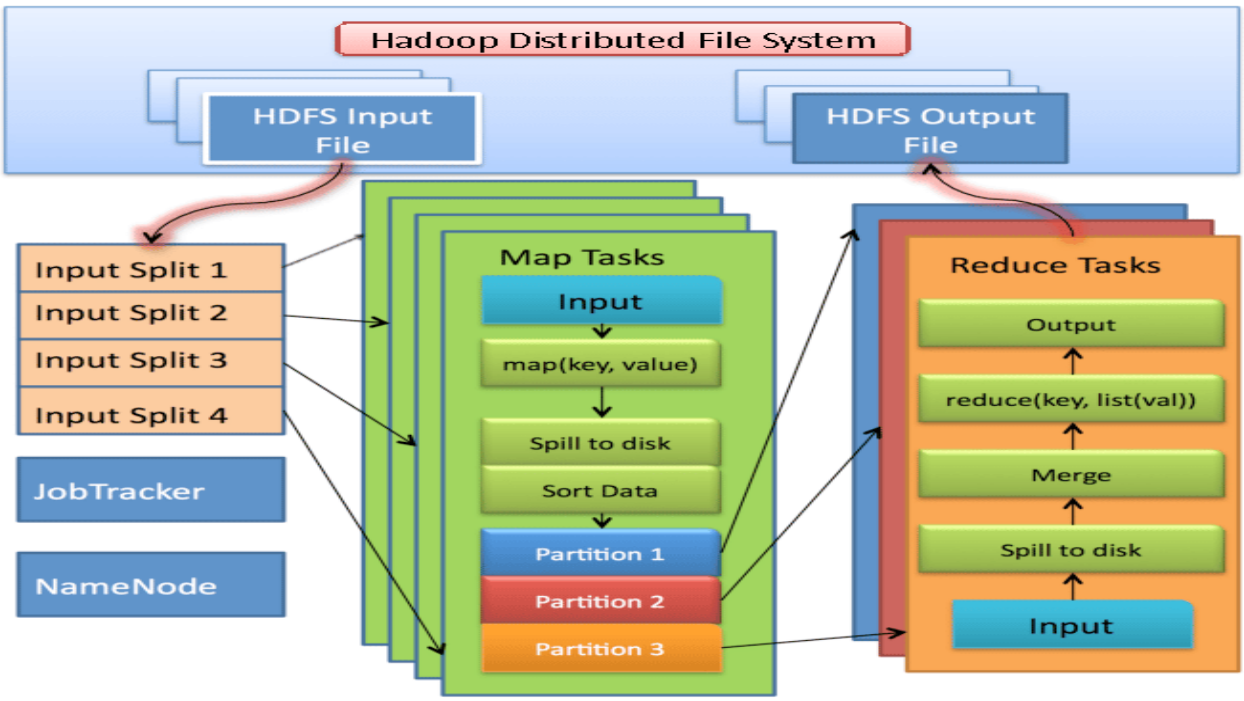
3. InputSplits:

It represents the data which will be processed by an individual **Mapper**. For each split, one map task is created. Thus the number of map tasks is equal to the number of InputSplits. Framework divide split into records, which mapper process.



Key-value pair Generation in MapReduce

- In MapReduce process, before passing the data to the mapper data should be first converted into key-value pairs.
- key-value pairs in Hadoop MapReduce is generated as follows:
- **InputSplit** – Input Split in Hadoop Map Reduce is the logical representation of data. It describes a unit of work that contains a single map task in a Map Reduce program.
- Hadoop Input Split represents the data which is processed by an individual Mapper.
- The split is divided into records. Hence, the mapper process each record (which is a key-value pair).
- The important thing to notice is that Input split does not contain the input Data it is just a reference to the data.
- As a user, we don't need to deal with Input Split directly, because they are created by an Input Format(Input Format creates the Input split and divide into records).



4. RecordReader:

- It communicates with the inputSplit. It converts the Split into **records** which are in form of key-value pairs that are suitable for reading by the mapper.In Hadoop terminology, each line in a text is termed as a '**record'.**
- **Record reader** converts this text into (key, value) pair depends on the format of the file.
- It defines Record Reader using **createRecordReader()**
- RecordReader by default uses TextInputFormat to convert data into a key-value pair.
- RecordReader communicates with the InputSplit until the file reading is not completed.
- Then, the key-value pairs are further sent to the **mapper** for further processing.

5.Mapper:

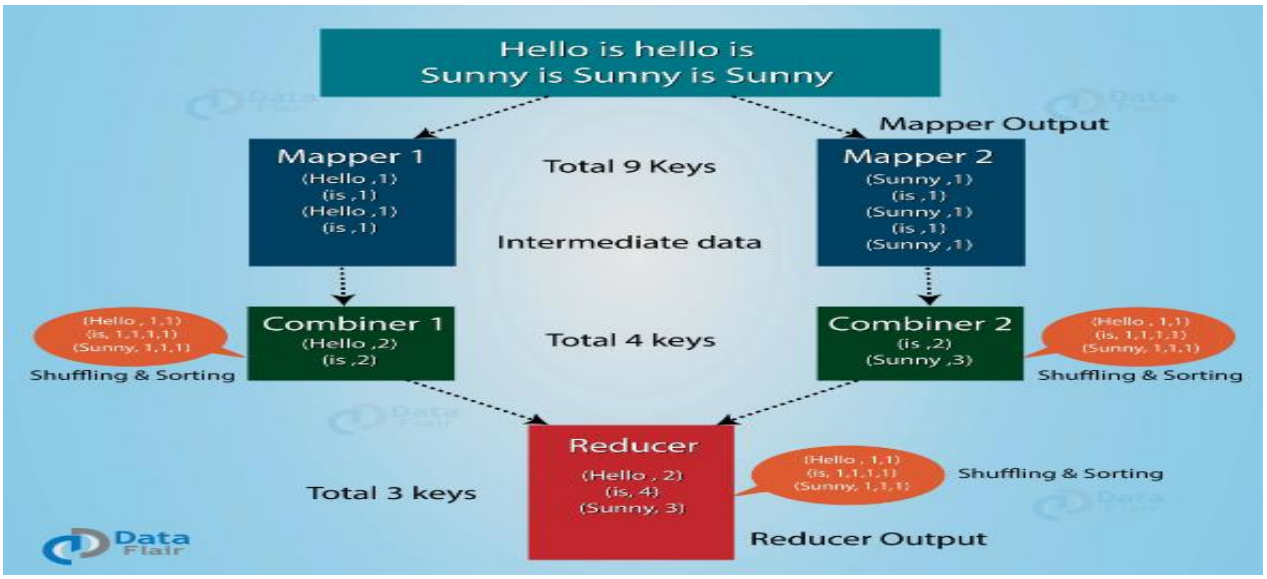
- Mapper task is the first phase of processing that processes each input record produced (from RecordReader) and generates an intermediate key-value pair. The intermediate output is

completelydifferent from the input pair.

- Hadoop Mapper store intermediate-output on the local disk.
- It doesn't store, as data on HDFSas data is temporary and writing on HDFS will create unnecessary multiple copies.
- The output of the **Mappers** is then passed to the **Combiner** for further processing.

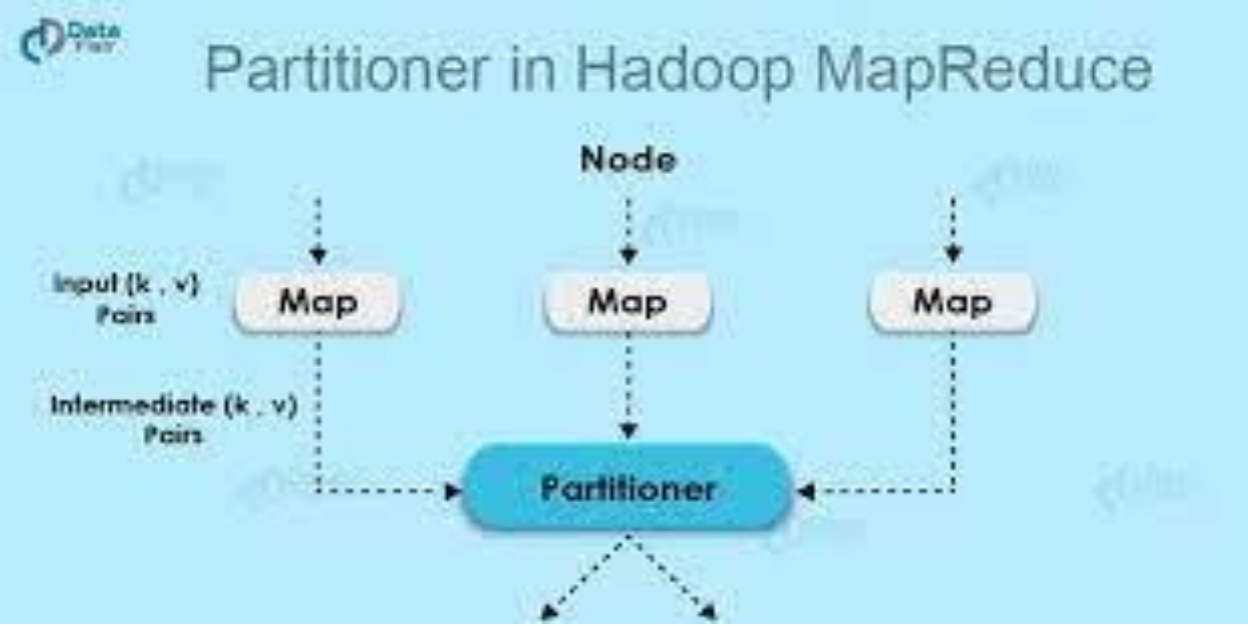
6. Combiner: **Combiner** always works in between Mapper and Reducer.

- The output produced by the Mapper is the intermediate output in terms of key-value pairs which is massive in size. If we directly feed this huge output to the Reducer, then that will result in increasing the **Network Congestion**. So, to minimize this Network congestion we have to put combiner in between Mapper and Reducer. These combiners are also known as **semi-reducer**.
- It takes intermediate <keys, value> pairs provided by mapper and applies user specific aggregate function to only one mapper. It is also known as local Reducer.
- We can optionally specify a combiner using Job.setCombinerClass (ReducerClass) to perform local aggregation on intermediate outputs.



7. Partitioner:

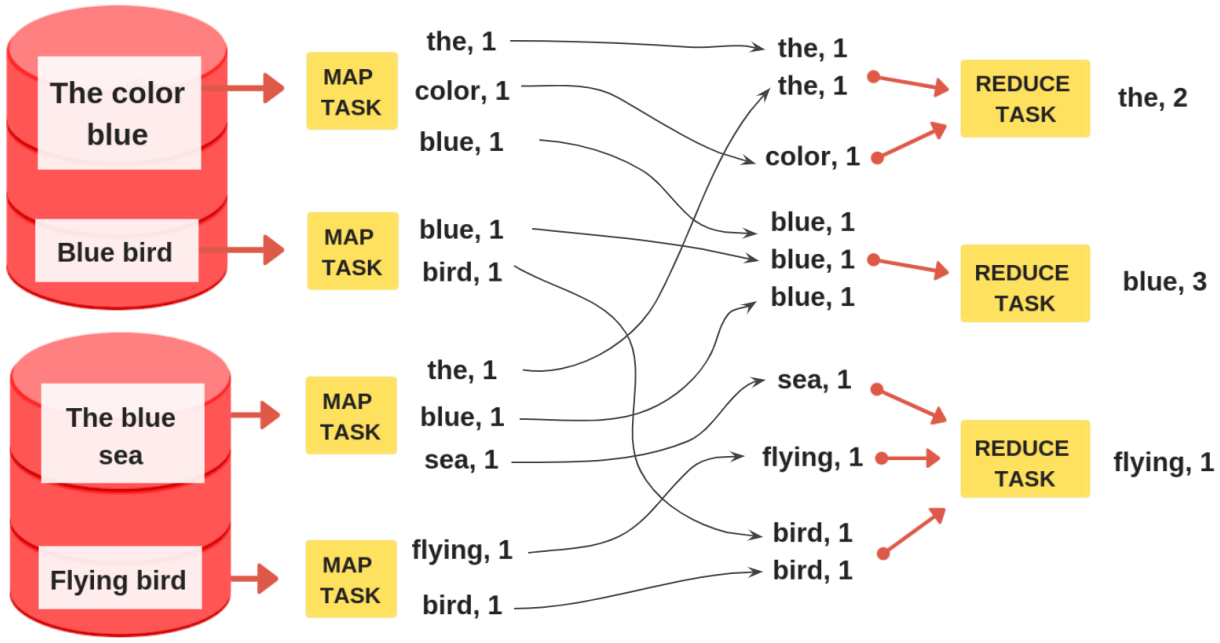
- Partitioner comes into the existence if we are working with more than one reducer. It takes the output of the combiner and performs partitioning. Partitioning of output takes place on the basis of the key in MapReduce. On the basis of key value in MapReduce, partitioning of each combiner output takes place. And then the record having the same key value goes into the same partition. After that, each partition is sent to a reducer.
- Values of the same key are sent to the same reducer. This helps to determine which reducer is responsible for which key.
- In MapReduce, the number of partitioners and reducers are the same. The output of a single partitioner is sent to a single reducer. If there is only one reducer no partitioner will be created.
- Take intermediate <keys, value> pairs produced by the mapper, splits them into partitions the data using a user-defined condition.
- Partitioning in MapReduce execution allows even distribution of the map output over the reducer.



8. Shuffling and Sorting:

- Before passing this intermediate data to the reducer, it is first passed through two more stages, called ***Shuffling and Sorting*.**
- **Shuffling Phase:** The process of transferring data from the mappers to reducers is known as shuffling.
- This phase combines all values associated to an identical key. For eg, if (Blue, 1) is there three times in the input file. So after the shuffling phase, the output will be like (Blue, [1,1,1]).
- **Sorting Phase:** Once shuffling is done, the output is sent to the sorting phase where all the (key, value) pairs are sorted automatically. In Hadoop sorting is an automatic process because of the presence of an inbuilt interface called **WritableComparableInterface.**
- This is then provided as input to reduce phase.

Example:



9.Reducer:

- Reduce is the second phase of processing where the user can specify his own custom business logic as per the requirements.
- Reducer then takes set of intermediate key-value pairs produced by the mappers as the input.
- After that runs a reducer function on each of them to generate the output.
- The output of the reducer is the final output. Then framework stores the output on HDFS.

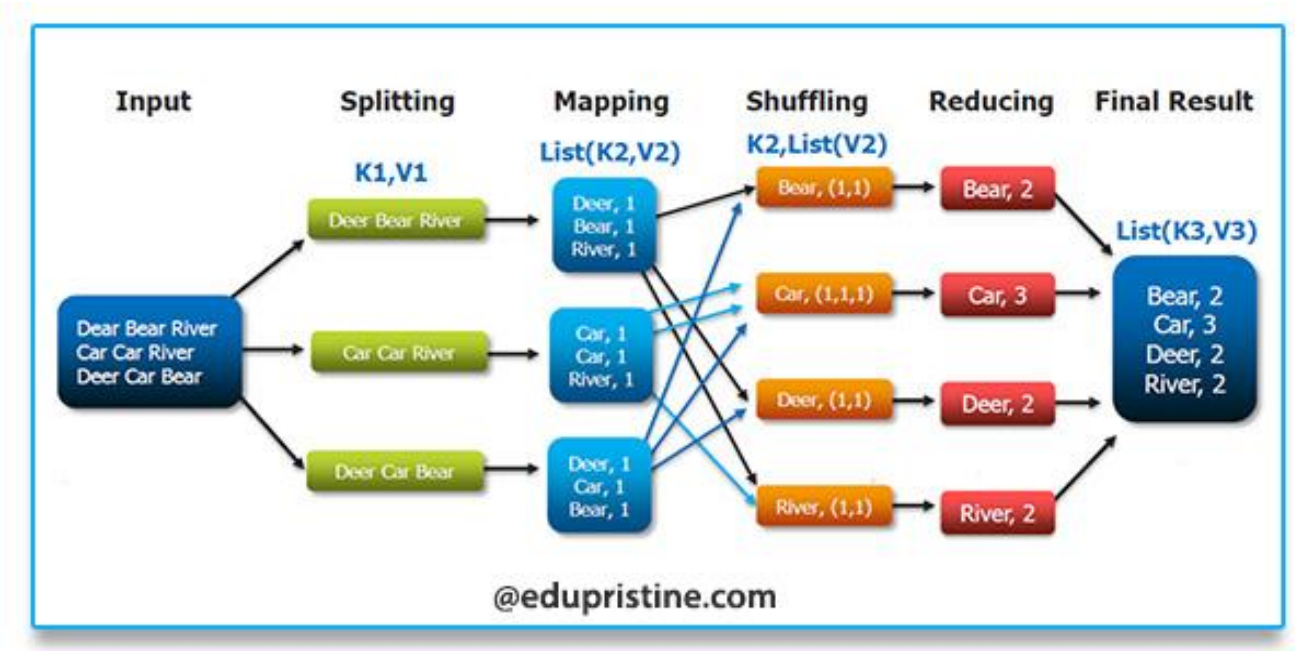
RecordWriter:

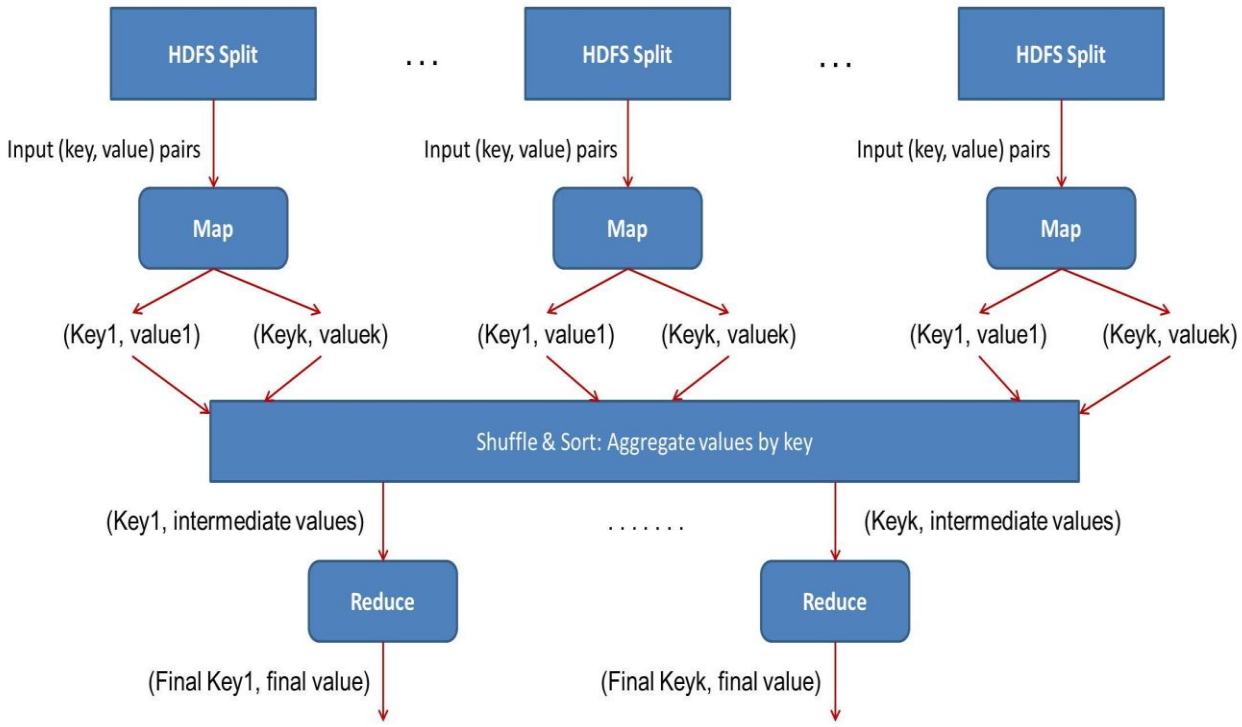
- The output format translate the final key/value pair from the reduce function and writes it out to a file by a record writer. It writes these output key-value pair from the Reducer phase to the output files.

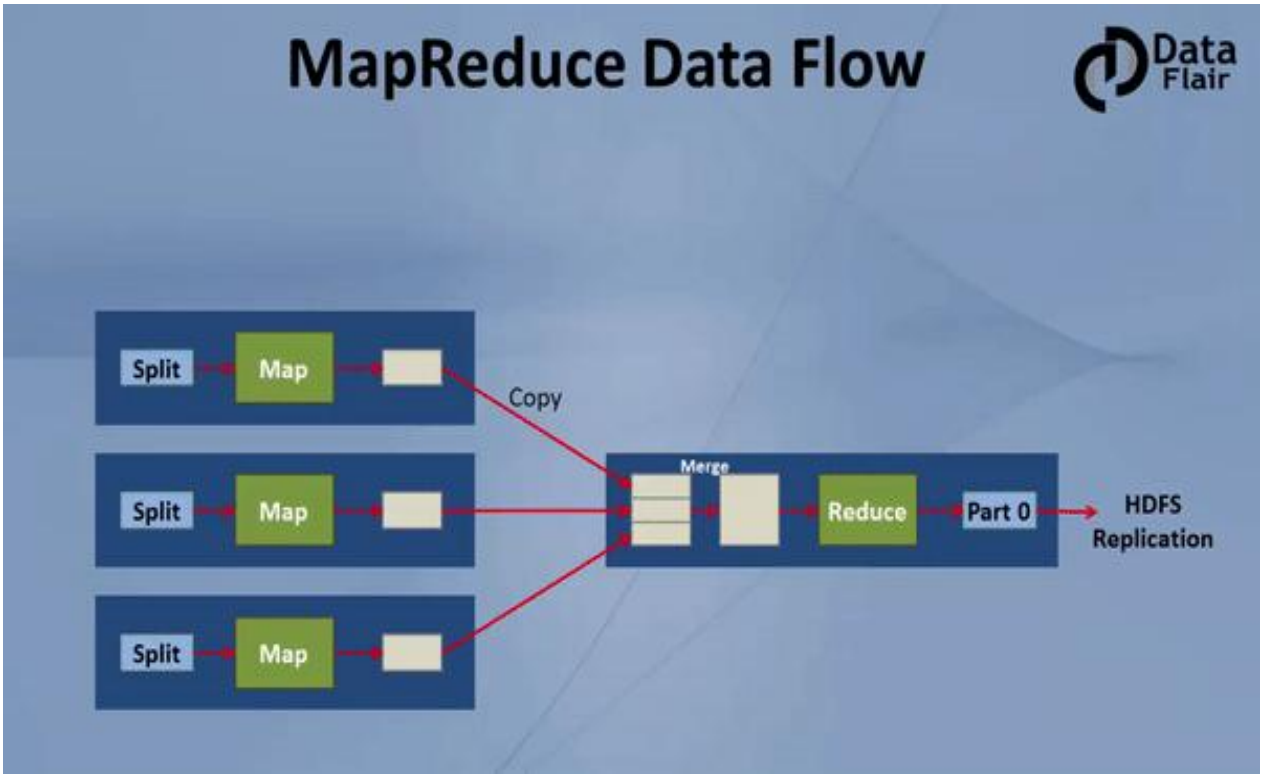
10.OutputFormat:

- OutputFormat defines the way how it writes these output key-value pairs in output files. So, its instances provided by the Hadoop write files in HDFS.
- Thus OutputFormat instances write the final output of reducer on HDFS.

MAPREDUCE WORD COUNT EXAMPLE







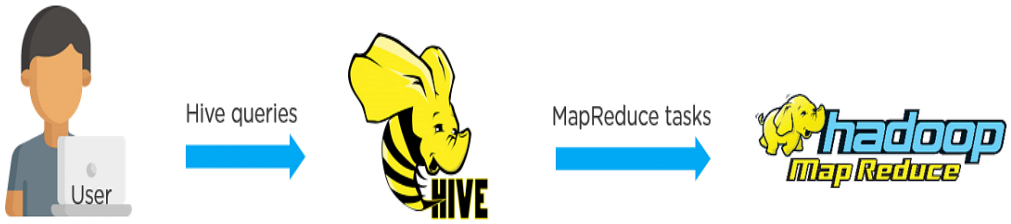
Hive – Introduction to Hive:

- The Hadoop ecosystem contains different sub-projects (tools) such as Sqoop, Pig, and Hive that are used to help Hadoop modules.
- There are various ways to execute MapReduce operations:

1. The traditional approach using **Java MapReduce program** for structured, semi-structured, and unstructured data.
2. The **scripting approach** for MapReduce to process structured and semi structured data using Pig.
3. The **Hive Query Language** (HiveQL or HQL) for MapReduce to process structured data using Hive.

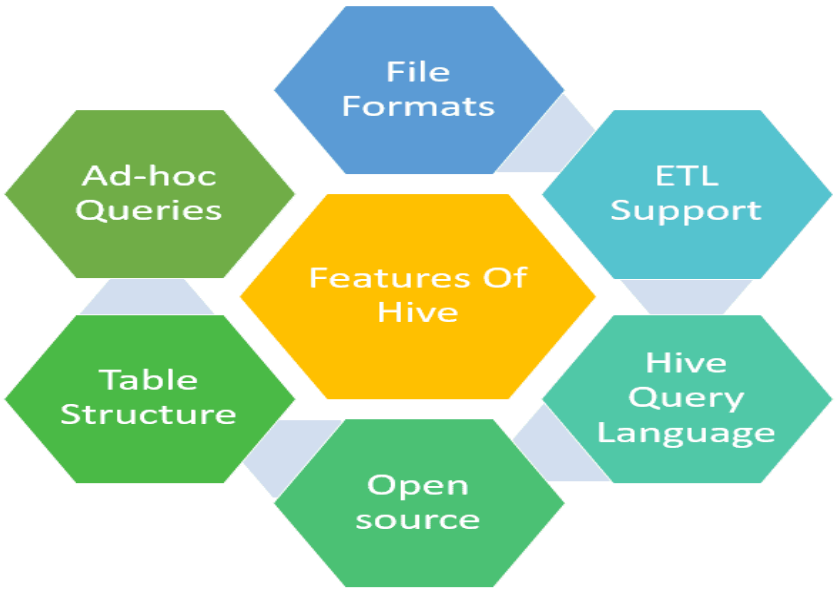
What is HIVE

- Hive is a data warehouse system or Tool which is used for Querying and Analyzing the structured data.
- It is built on the top of Hadoop.
- Initially Hive was developed by Facebookquery their huge amount of data each day (around 20TB) ,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. For example, Amazon uses it in Amazon Elastic MapReduce.
- 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).



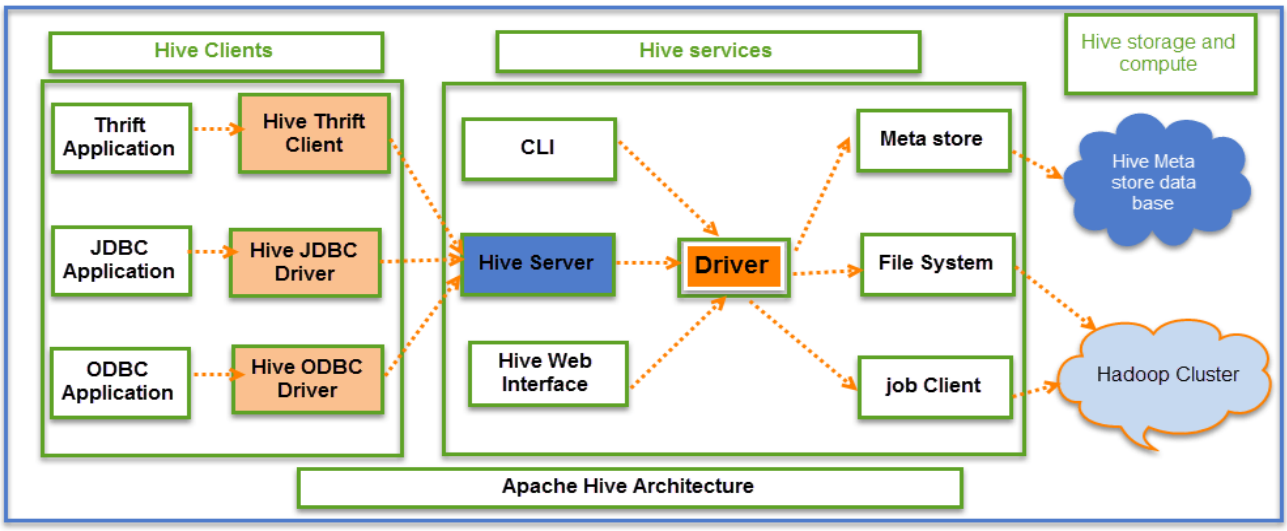
Data flow

Features of HIVE:



- **Open-Source:** It is an open-source tool so we can use it free of cost.
- **File Formats:** It supports various types of file formats such as textfile, ORC, Parquet, etc.
- **Hive-Query Language:** This language is similar to SQL. Only the basic knowledge of SQL is enough to work with Hive such as tables, rows, columns, and schema, etc.
- **Fast:** Hive is a Fast, scalable, extensible tool and uses familiar concepts.
- **Table Structure:** Hive as data warehouse designed for managing and querying only structured data that is stored in tables that is similar to RDBMS Tables.
- **Ad-hoc Queries:** Hive allows us to run ad-hoc queries which are the commands or queries whose value depends on some variable for the data analysis.
- **ETL Support:** ETL Functionalities such as Extraction, Transformation, and Loading data into tables coupled with joins, partitions, etc.

HIVEArchitecture:



Hive Consists of Mainly 3 core parts

1. **Hive Clients**
2. **Hive Services**
3. **Hive Storage and Computing**

Hive Client:

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

- **Hive Thrift Clients**- 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:-

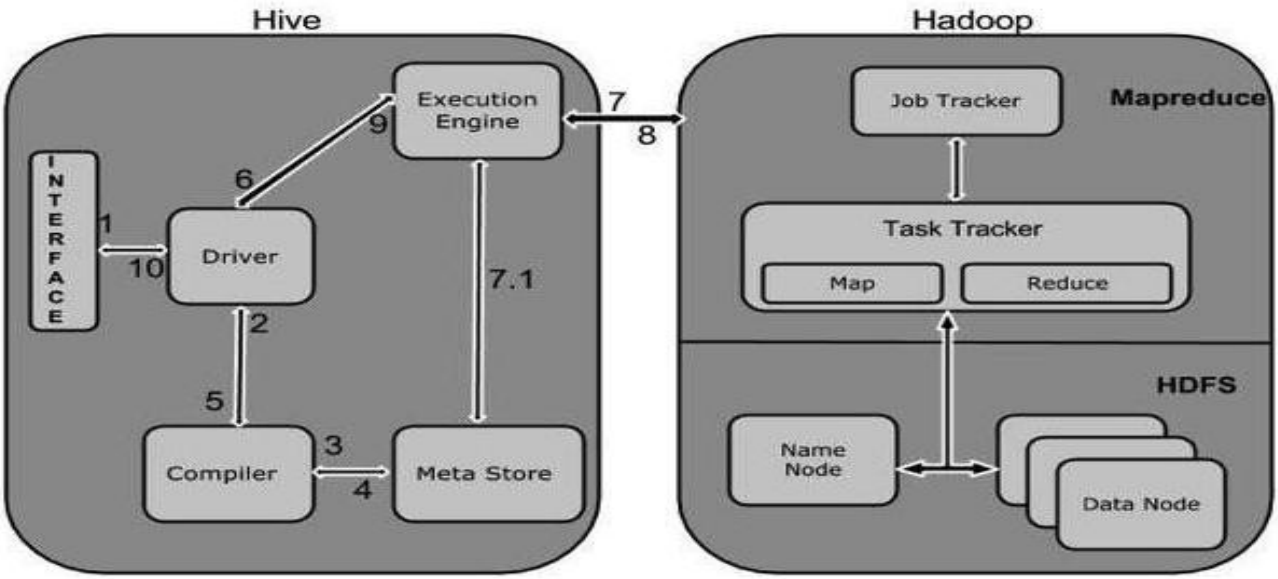
- **Hive CLI**- The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
- **Hive Web User Interface** - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- **Hive MetaStore**- It is a central 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 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. Execution Engine process the queries and generates the results same as mapreduce results. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Hive Storage and Computing:

Hive services, such as the Meta Store, the file system, and work clients, are also involved in and do the following for the Hive repository.

- Metadata tables created in Hive are stored in the "Meta storage database" in Hive.
- The results of the query and data loaded in the tables will be stored on HDFS in the Hadoop cluster.

Working of HIVE:



1. **Execute Query:** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2. **Get Plan :**The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3. **Get Metadata:** The compiler sends metadata request to Metastore (any database).
4. **Send Metadata :**Meta store sends metadata as a response to the compiler.
5. **Send Plan :**The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.

6. **Execute Plan :** The driver sends the execute plan to the execution engine.
7. **Execute Job :**Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.

   7.1 **Metadata Ops:**Meanwhile in execution, the execution engine can execute metadata operations with Metastore.

8. **Fetch Result:**The execution engine receives the results from Data nodes.

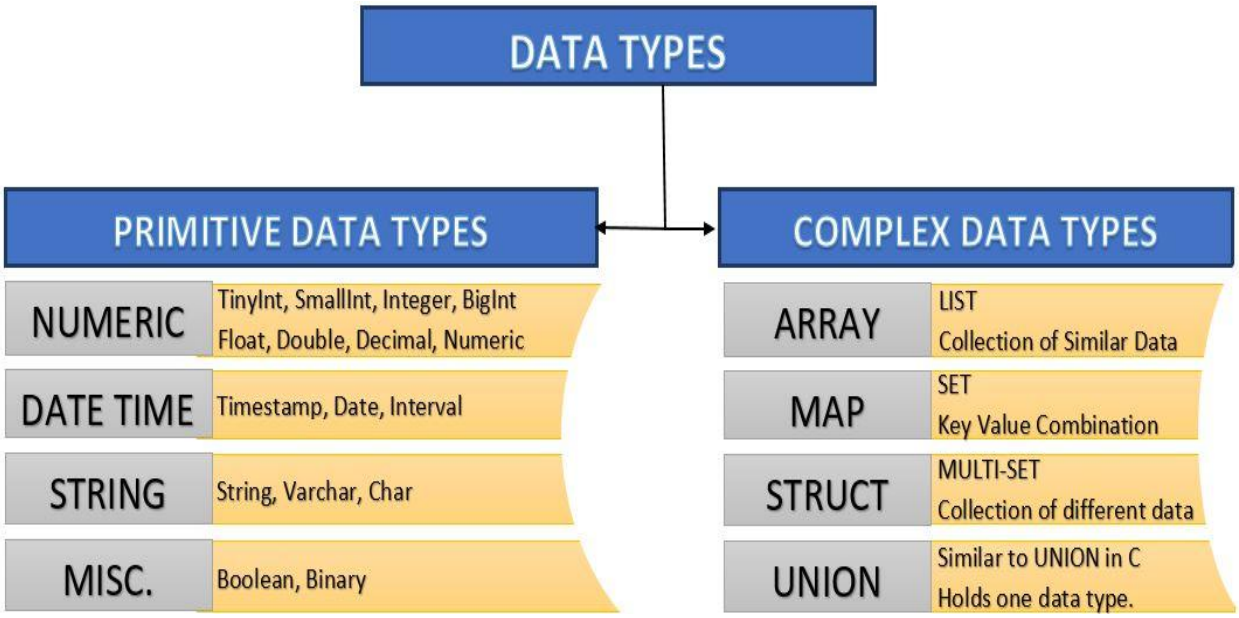
9. **Send Results :**The execution engine sends those resultant values to the driver.

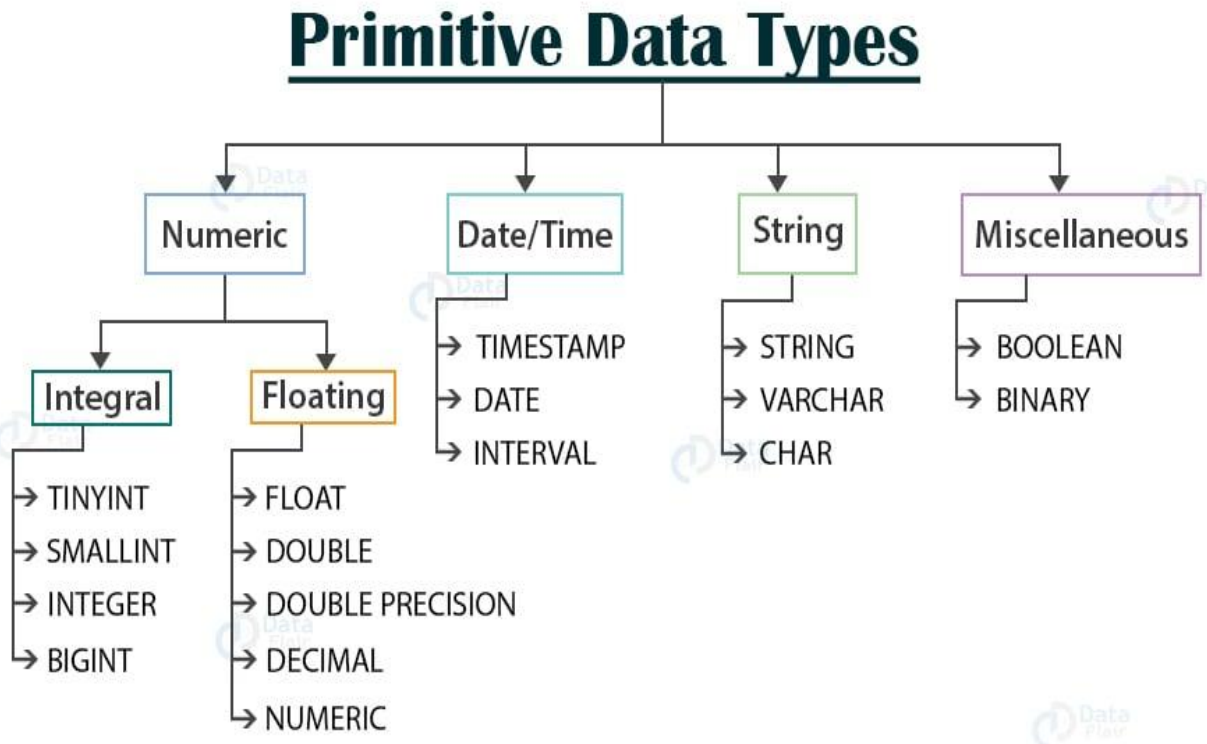
10. **Send Results :**The driver sends the results to Hive Interfaces.

Advantages of Hive Architecture:

- Scalability: Hive is a distributed system that can easily scale to handle large volumes of data by adding more nodes to the cluster.
- Data Accessibility: Hive allows users to access data stored in Hadoop without the need for complex programming skills. SQL-like language is used for queries and HiveQL is based on SQL syntax.
- Data Integration: Hive integrates easily with other tools and systems in the Hadoop ecosystem such as Pig, HBase, and MapReduce.
- Flexibility: Hive can handle both structured and unstructured data, and supports various data formats including CSV, JSON, andParquet.
- Security: Hive provides security features such as authentication, authorization, and encryption to ensure data privacy.

HIVE DATA TYPES:





PRIMITIVEDATATYPES

Numeric Data Types

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

Decimal Type:

The syntax and example is as follows:

DECIMAL(precision, scale)

Eg: decimal(10,0)

Type	Size	Range
FLOAT	4-byte	Single precision floating point number(32 Bytes)
DOUBLE	8-byte	Double precision floating point number(64 Bytes)

String Data Type: String Data is divided into three types:

- String
- Varchar
- char

String

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

Varchar

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

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

Char

- The char is a type of fixed length, with a maximum length of 255.

Date/Time Data Type: The Hive data types in this category are:

- TimeStamp
- Date

TimeStamp

- It is used for nanosecond precision and denoted by YYYY-MM-DD hh:mm:ss format.

Date

- The date value is used in the form 'YYYY-MM-DD' to determine a particular year, month, and day. It does not provide the time of the day.

Miscellaneous Data Type:

- Different types of data support both Boolean and binary types of data.
- Boolean
- Binary
- **Boolean:-**
- The Boolean stores the value either **true** or **false**.
- **Binary:-**
- It is defined as an array of bytes.

COMPLEX DATA TYPES : Complex Data is divide into three types:

- **Array**
- **Map**
- **Struct**
- **Array**
- Array is defined as the collection of similar data types. The value of such data types are indexable using the zero-based Integer.
- Arrays in Hive are used the same way they are used in Java.

Syntax: ARRAY<data_type>

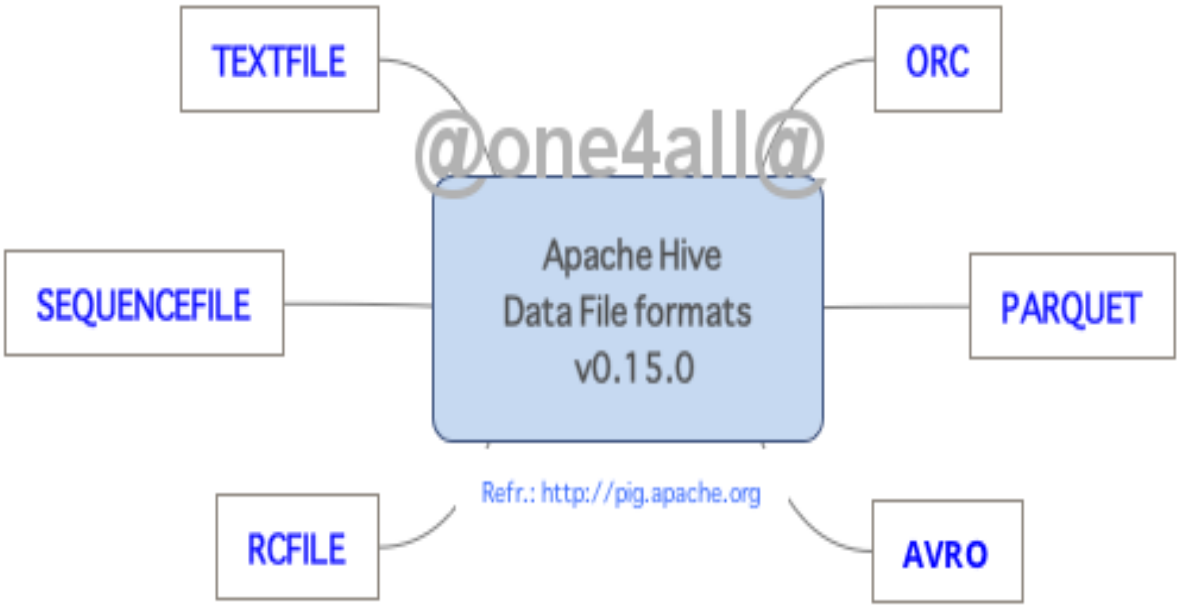
- **Map**
- It is a collection of key-value pairs. The Keys can be primitives values, or any data type. The keys and values for a specific map must be of the same type.

Syntax: MAP<primitive_type, data_type>

- **Struct**
- It is defined as the collection of named fields. The field can be of different types.
- The Struct is similar to the STRUCT present in C language).

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

Hive File Formats:



- Hive supports various file formats for data storage and retrieval. The choice of file format can impact the performance, compression, and compatibility of your Hive tables. Here are some commonly used file formats in Hive:

1. **TextFile**:

1. The default storage format in Hive is plain text, where each line in a

file represents a record, and fields within a record are delimited by a specified character (e.g., tab, comma, or custom delimiter).

2. TextFile format is human-readable but not space-efficient or suitable for complex data structures.

2. **SequenceFile**:

1. SequenceFiles are a binary file format optimized for Hadoop's MapReduce and Hive. They can store key-value pairs efficiently.
2. When converting queries to MapReduce jobs, Hive chooses to use the necessary key-value pairs for a given record.
3. The key advantages of using a sequence file are that it incorporates two or more files into one file.
4. The sequence file format storage option is defined by specifying "**STORED AS SEQUENCEFILE**" at the end of the table creation.

3. ORC (Optimized Row Columnar)

1. The Optimized Row Columnar (ORC) file format provides a highly efficient way to store data in the Hive table.
2. This file system was actually designed to overcome limitations of the other Hive file formats.
3. ORC reduces I/O overhead by accessing only the columns that are required for the current query. It requires significantly fewer seek operations because all columns within a single group of row data are stored together on disk. The ORC file format storage option is defined by specifying "**STORED AS ORC**" at the end of the table creation.
4. ORC files are compressed and use lightweight compression algorithms, making them efficient for both storage and query performance.

RCFILE

RCFile = Record Columnar File

- RCFILE is used when we want to perform operations on multiple rows at a time.
- RCFILEs are flat files consisting of binary key/value pairs, which shares many similarities with SEQUENCEFILE.
- RCFILE stores columns of a table in form of record in a columnar manner. It first partitions rows horizontally into row splits and then it vertically partitions each row split in a columnar way.
- RCFILE first stores the metadata of a row split, as the key part of a record, and all the data of a row split as the value part. This means that RCFILE encourages column oriented storage rather than row oriented storage.

Hive Parquet File Format

- Parquet is a column-oriented binary file format.
- The parquet is highly efficient for the types of large-scale queries. Parquet is especially good for queries scanning particular columns within a particular table.
- The Parquet table uses compression Snappy, gzip; currently Snappy by default.
- Create Parquet file by specifying 'STORED AS PARQUET' option at the end of a CREATE TABLE Command.

HiveQL data definition(Creating Databases and Tables):

- Hive Query Language(HQL)
- Hive Query Language is a language used in Hive, similar to SQL, Hive Query Language is easy to use if you are familiar with SQL. The syntax of Hive QL is very similar to SQL with slight differences.

- Hive QL supports **DDL, DML, and user-defined functions**.
- In Hive, Data Definition Language (DDL) commands are used to define, manage, and manipulate the structure of database objects such as tables, views, and partitions.

DDL Commands: The following is the list of DDL statements that are supported in Apache Hive.

1. CREATE
2. DROP
3. TRUNCATE
4. ALTER
5. SHOW
6. DESCRIBE
7. USE

1. CREATE DATABASE in Hive

- The **CREATE DATABASE** statement is used to create a database in the Hive. A database in Hive is a collection of tables.

Syntax:

CREATE (DATABASE) [**IF** NOT EXISTS] database_name

[COMMENT database_comment]

[LOCATION hdfs_path]

[WITH DBPROPERTIES (property_name=property_value, ...)];

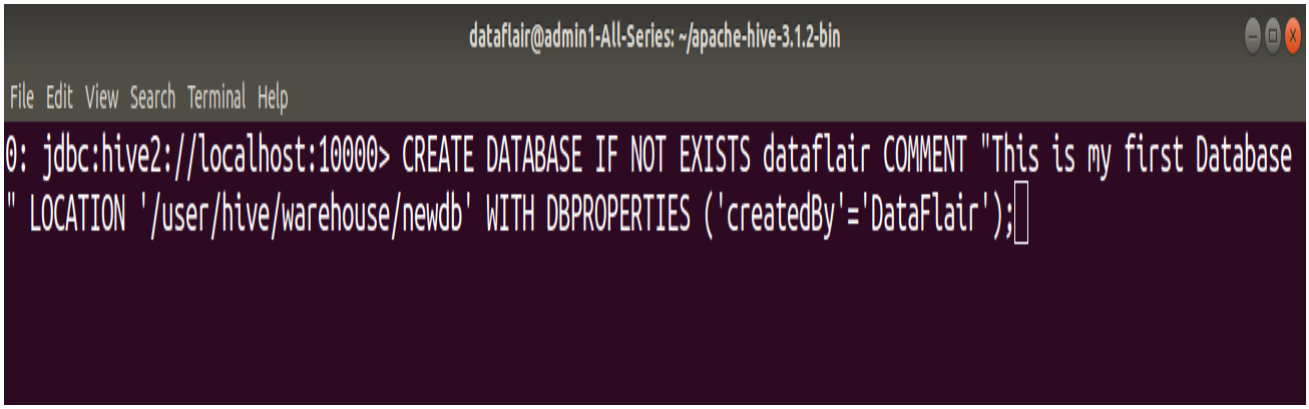
or

- CREATE SCHEMAdatabase_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.

Example:
Here in this example, we are creating a database ‘dataflair’.

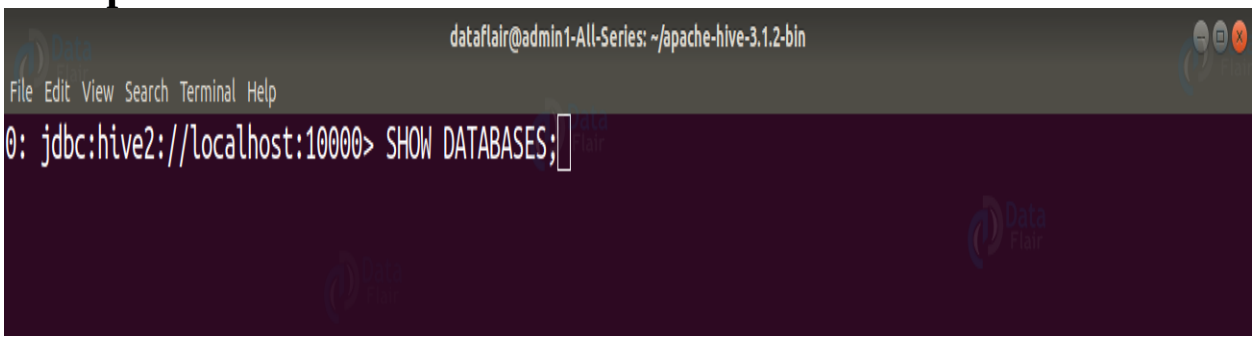


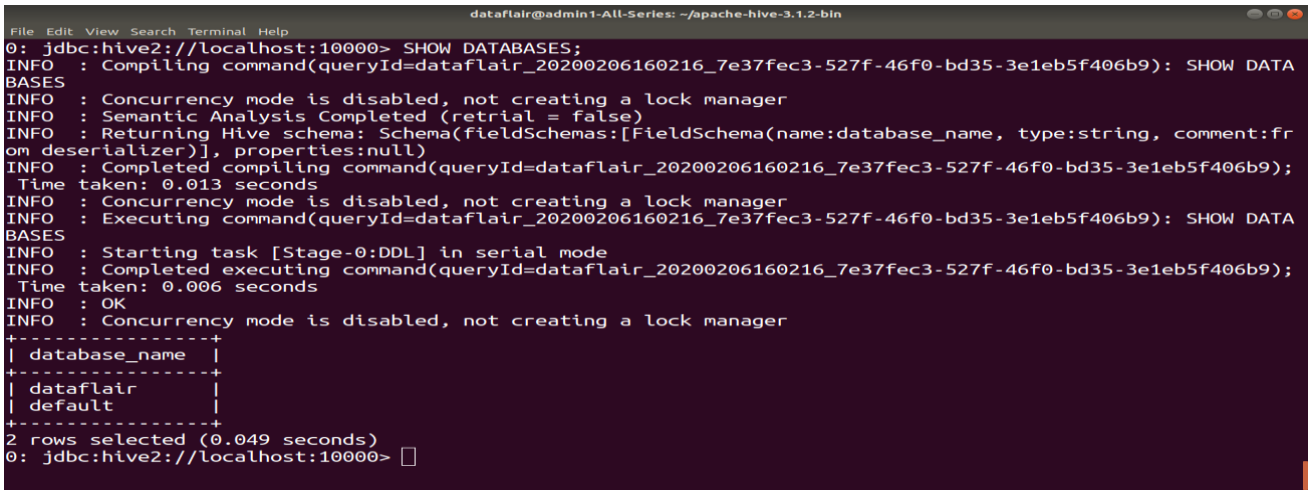
2. SHOW DATABASE in Hive

- The **SHOW DATABASES** statement lists all the databases present in the Hive.
- **Syntax:**

  SHOW (DATABASES);

Example:



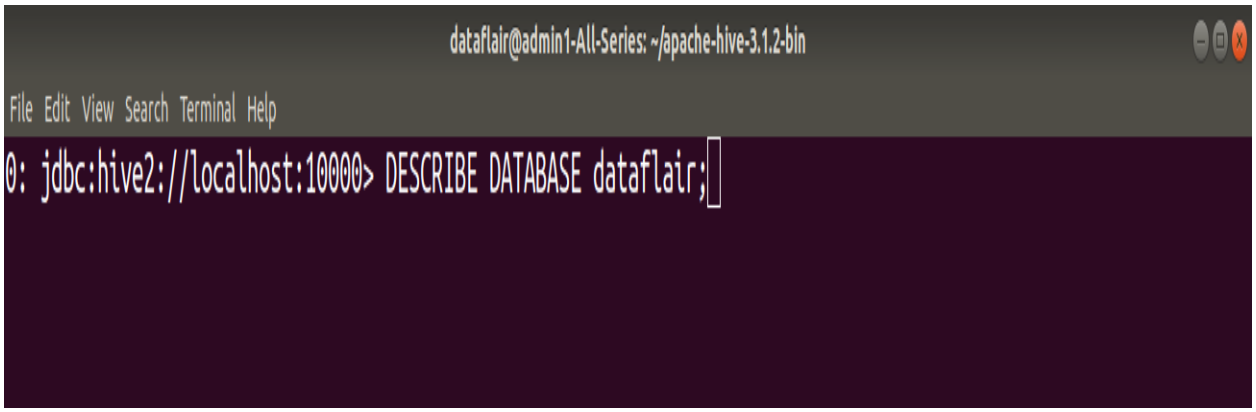


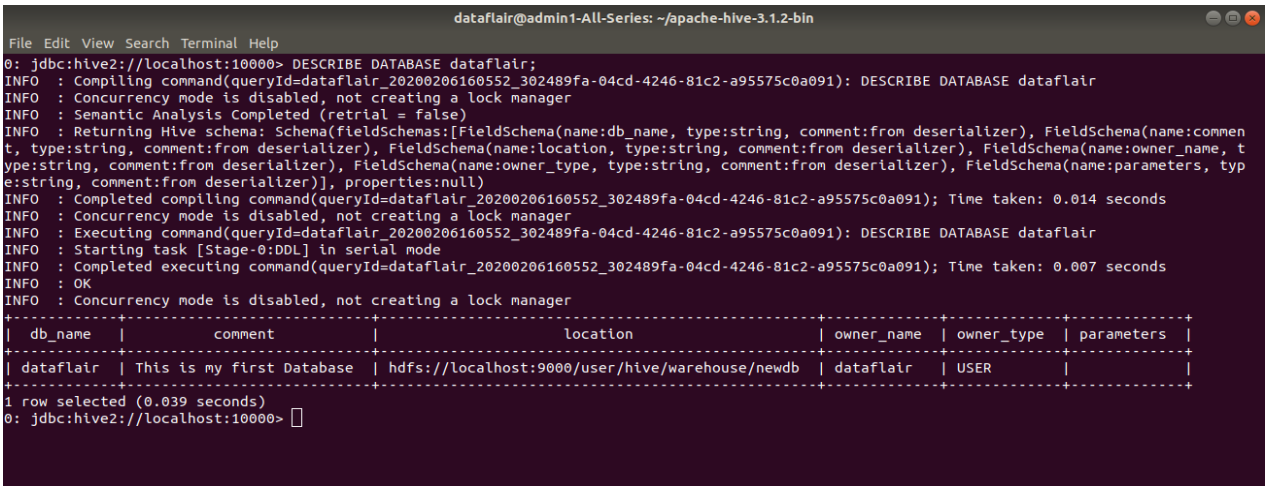
3. DESCRIBE DATABASE in Hive

- The DESCRIBE DATABASE statement in Hive shows the name of Database in Hive, its comment (if set), and its location on the file system.
- The EXTENDED can be used to get the database properties.

Syntax:

DESCRIBE DATABASE [EXTENDED] db_name;





4.DROP DATABASE in Hive

- The **DROP DATABASE** statement in Hive is used to Drop (delete) the database.
- The default behavior is RESTRICT which means that the database is dropped only when it is empty. To drop the database with tables, we can use CASCADE.
- **Syntax:**

DROP (DATABASE|SCHEMA) [**IF** EXISTS] database_name [RESTRICT|CASCADE];

Here in this example, we are dropping a database ‘dataflair’ using the DROP statement.

Creation of Tables:

The way of creating tables in the hive is very much similar to the way we create tables in SQL.

Syntax:

Hive>CREATE TABLE [IF NOT EXISTS] <table-name> (<column-name><data-type>,<column-name><data-type> COMMENT 'Your Comment',<column-name><data-type>)COMMENT 'Add if you want'

LOCATION 'Location On HDFS'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','STORED AS FILE FORMAT;

Here ROW FORMAT DELIMITED shows that whenever a new line is encountered the new record entry .

Example:

CREATE TABLE IF NOT EXISTS student_data(Student_Name STRING COMMENT 'This col. Store the name of student', Student_Rollno INT COMMENT 'This col. Stores the rollno of student',Student_Marks FLOAT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

```
hive> CREATE TABLE IF NOT EXISTS student_data (
    > Student_Name STRING COMMENT 'This col. Store the name of student',
    > Student_Rollno INT COMMENT 'This col. Stores the rollno of student',
    > Student_Marks FLOAT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 1.358 seconds
hive>
```

Show table:

Syntax:

```
SHOW TABLES [IN <database_name>];
```

Command:

```
SHOW TABLES IN student_detail;
```

```
hive> SHOW TABLES IN student_detail;
OK
student_data
Time taken: 0.056 seconds, Fetched: 1 row(s)
hive>
```

Hive DML Operations:

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.

The following list of DML statements is supported by Apache Hive.

1. LOAD
2. SELECT
3. INSERT
4. DELETE
5. UPDATE

HiveQL for Data loading:

In hive with DML statements, we **can Load data to the Hive table in 2 different ways**.

- Using INSERT Command
- Load Data Statement

1. Using INSERT Command

Syntax:

INSERT INTO TABLE <table_name> VALUES (<add values as per column entity>);

Example:

To insert data into the table let's create a table with the name ***student*** (By default hive uses its *default* database to store hive tables).

Command:

CREATE TABLE IF NOT EXISTS student(

Student_Name STRING,

Student_Rollno INT,

Student_Marks FLOAT)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';

EXAMPLE:

```
hive> CREATE TABLE IF NOT EXISTS student(
    > Student_Name STRING,
    > Student_Rollno INT,
    > Student_Marks FLOAT)
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ',';
OK
Time taken: 2.086 seconds
hive> show tables in default;
OK
student
Time taken: 0.268 seconds, Fetched: 1 row(s)
hive>
```

INSERT Query:

INSERT INTO TABLE student VALUES ('Dikshant',1,'95'),('Akshat', 2 , '96'),('Dhruv',3,'90');

```
hive> INSERT INTO TABLE student VALUES ('Dikshant',1,'95'),('Akshat', 2 , '96'),
('Dhruv',3,'90');
Query ID = dikshant_20201106121659_f5dfa694-f552-4b7a-a64b-4f3804213ab8
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
```

check the data of the ***student*** table with the help of the below command.

SELECT * FROM student;

```
hive> SELECT * FROM student;
OK
Dikshant        1       95.0
Akshat  2       96.0
Dhruv   3       90.0
Time taken: 0.162 seconds, Fetched: 3 row(s)
hive>
```

2. Load Data Statement:

Hive provides us the functionality to load pre-created table entities either from our local file system or from HDFS. The *LOAD DATA* statement is used to load data into the hive table.

Syntax:

LOAD DATA [LOCAL] INPATH '<The table data location>' [OVERWRITE] INTO TABLE <table_name>;

- The **LOCAL** Switch specifies that the data we are loading is available in our Local File System. If the LOCAL switch is not used, the hive will consider the location as an **HDFS** path location.
- The OVERWRITE switch allows us to overwrite the table data.

- LOAD DATA to the student hive table with the help of the below command.
- **LOAD DATA LOCAL INPATH '/home/dikshant/Documents/data.csv' INTO TABLE student;**

```
hive> LOAD DATA LOCAL INPATH '/home/dikshant/Documents/data.csv' INTO TABLE stud
ent;
Loading data to table default.student
OK
Time taken: 2.617 seconds
hive>
```

SELECT * FROM student;

```
hive> SELECT * FROM student;
OK
Dikshant        1       95.0
Akshat  2       96.0
Dhruv   3       90.0
Ganesh  4       85.0
Chandan 5       65.0
Bhavani 6       87.0
Time taken: 1.24 seconds, Fetched: 6 row(s)
hive>
```

Thus from above table We can observe that we have successfully added the data to the **student table**.

The Select statement project the records from the table.

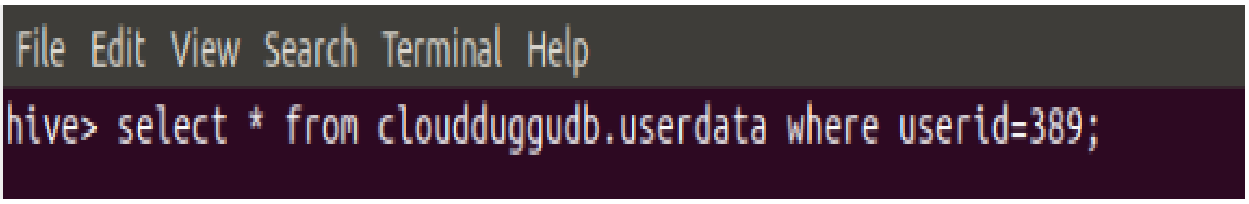
Select Command Syntax:

SELECT [ALL | DISTINCT] select_expr, select_expr, ...
FROM table_reference
[WHERE where_condition]
[GROUP BY col_list]
[ORDER BY col_list]
[CLUSTER BY col_list
| [DISTRIBUTE BY col_list] [SORT BY col_list]
]
[LIMIT [offset,] rows]

Select Command Statement:

SELECT * FROM cloudduggudb.userdata WHERE userid=389;

Example:



Update Command:

The Update command updates the existing records if where clause is supplied otherwise it will delete table data. We can't perform update command on Partitioning and Bucketing columns.

Update Command Syntax:

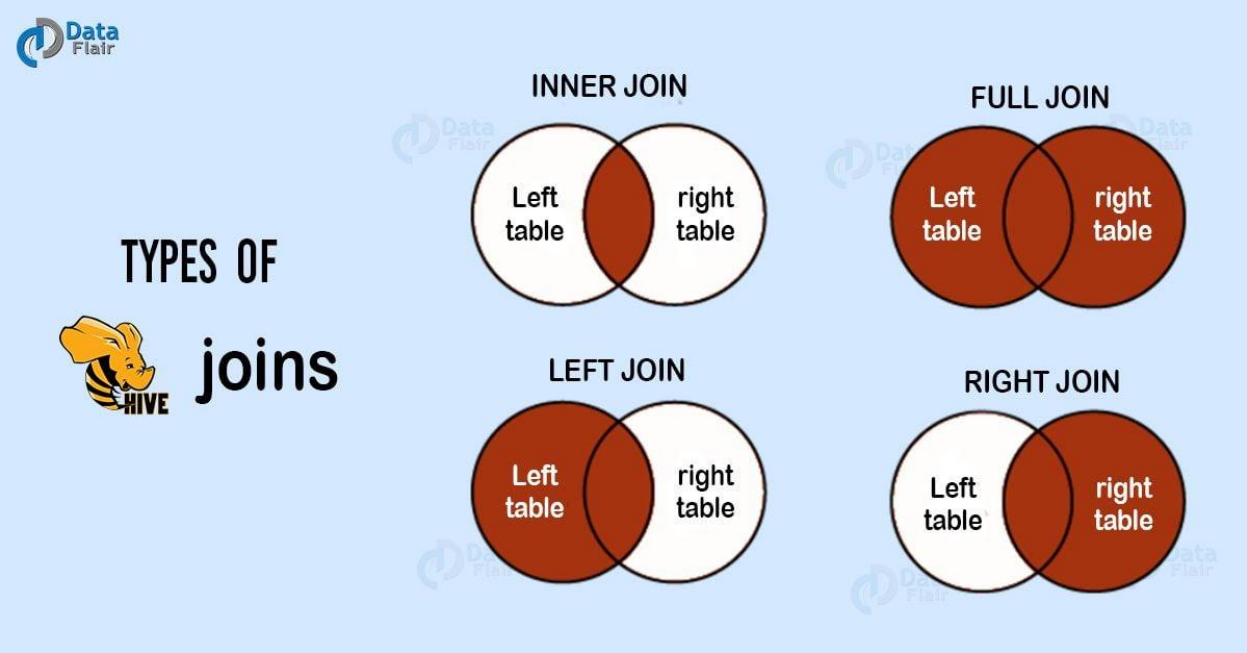
UPDATE tablename SET column = value [, column = value ...] [WHER E expression];

Delete Command Syntax:

DELETE FROM tablename [WHERE expression];

Joins:

- Apache Hive JOINs are used to combine columns from one (self-join) or more tables by using values common to each. Using join we can fetch corresponding records from two or more tables. It is almost similar to SQL joins.
- There are some points we need to observe about Hive Join:
  - In Hive Joins, only Equality joins are allowed.
  - However, in the same query more than two tables can be joined.
  - Also, note that Hive Joins are not Commutative



Example of Join in Hive

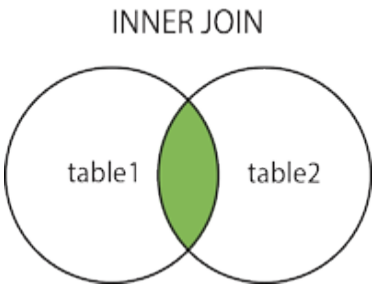
Example of Hive Join – HiveQL Select Clause.
Consider the following table named CUSTOMERS & ORDERS table
Table.1 Hive Join Example(CUSTOMERS TABLE):

ID	Name	Age	Address	Salary
1	Ross	32	Ahmedabad	2000
2	Rachel	25	Delhi	1500
3	Chandler	23	Kota	2000
4	Monika	25	Mumbai	6500
5	Mike	27	Bhopal	8500
6	Phoebe	22	MP	4500
7	Joey	24	Indore	10000

Table.2 – Hive Join Example(ORDERS Table):

OID	Date	Customer_ID	Amount
102	2016-10-08 00:00:00	3	3000
100	2016-10-08 00:00:00	3	1500
101	2016-11-20 00:00:00	2	1560
103	2015-05-20 00:00:00	4	2060

Inner Join: Returns records that have matching values in both tables.



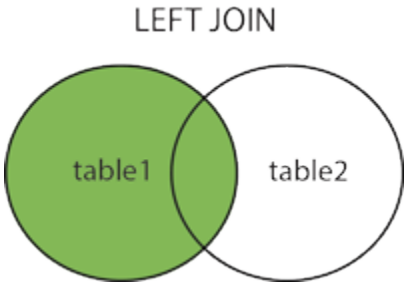
Syntax:

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

Example: Inner Join in Hive- output

ID	Name	Age	Amount
3	Chandler	23	3000
3	Chandler	23	1500
2	Rachel	25	1560
4	Monika	25	2060

Left Outer Join: Returns all records from the left table, and the matched records from the right table. Although, it returns with NULL in each column from the right table in case of no matching JOIN predicate.



Syntax:

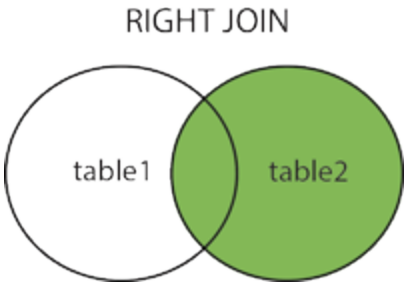
```
hive> SELECT c.ID, c.NAME, o.AMOUNT, o.DATE
FROM CUSTOMERS c LEFT OUTER JOIN ORDERS o ON (c.ID
= o.CUSTOMER_ID);
```

Example: **Left Outer Join in Hive Output**

ID	Name	Amount	Date
1	Ross	NULL	NULL
2	Rachel	1560	2016-11-20 00:00:00
3	Chandler	3000	2016-10-08 00:00:00
3	Chandler	1500	2016-10-08 00:00:00
4	Monika	2060	2015-05-20 00:00:00
5	Mike	NULL	NULL
6	Phoebe	NULL	NULL
7	Joey	NULL	NULL

Right Outer Join:

Returns all records from the right table, and the matched records from the left table. Although, it returns with NULL in each column from the left table in case of no matching join predicate.



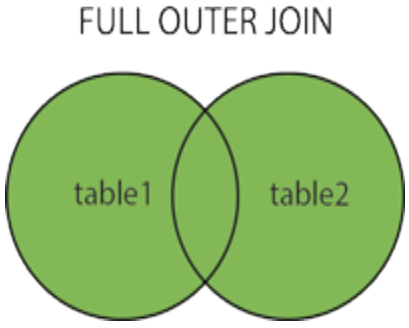
Syntax:

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

Example Right Outer Join in Hive Output:

ID	Name	Amount	Date
3	Chandler	1300	2016-10-08 00:00:00
3	Chandler	1500	2016-10-08 00:00:00
2	Rachel	1560	2016-11-20 00:00:00
4	Monika	2060	2015-05-20 00:00:00

Full Outer Join : The major purpose of this HiveQL Full outer Join is it combines the records of both the left and the right outer tables which fulfills the Hive JOIN condition. Moreover, this joined table contains either all the records from both the tables or fills in NULL values for missing matches on either side.



Syntax:

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

Example: Full Outer Join in Hive Output:

ID	Name	Amount	Date
1	Ross	NULL	NULL
2	Rachel	1560	2016-11-20 00:00:00
3	Chandler	3000	2016-10-08 00:00:00
3	Chandler	1500	2016-10-08 00:00:00
4	Monika	2060	2015-05-20 00:00:00
5	Mike	NULL	NULL
6	Phoebe	NULL	NULL
7	Joey	NULL	NULL
3	Chandler	3000	2016-10-08 00:00:00
3	Chandler	1500	2016-10-08 00:00:00
2	Rachel	1560	2016-11-20 00:00:00
4	Monika	2060	2015-05-20 00:00:00

Window functions in HIVE:

- A windowfunction performs a calculation across a set of table rows .
- The use of the windowing feature is to create a window on the set of data , in order to operate aggregation like Standard aggregations:
- Window functions in **Hive are two types**

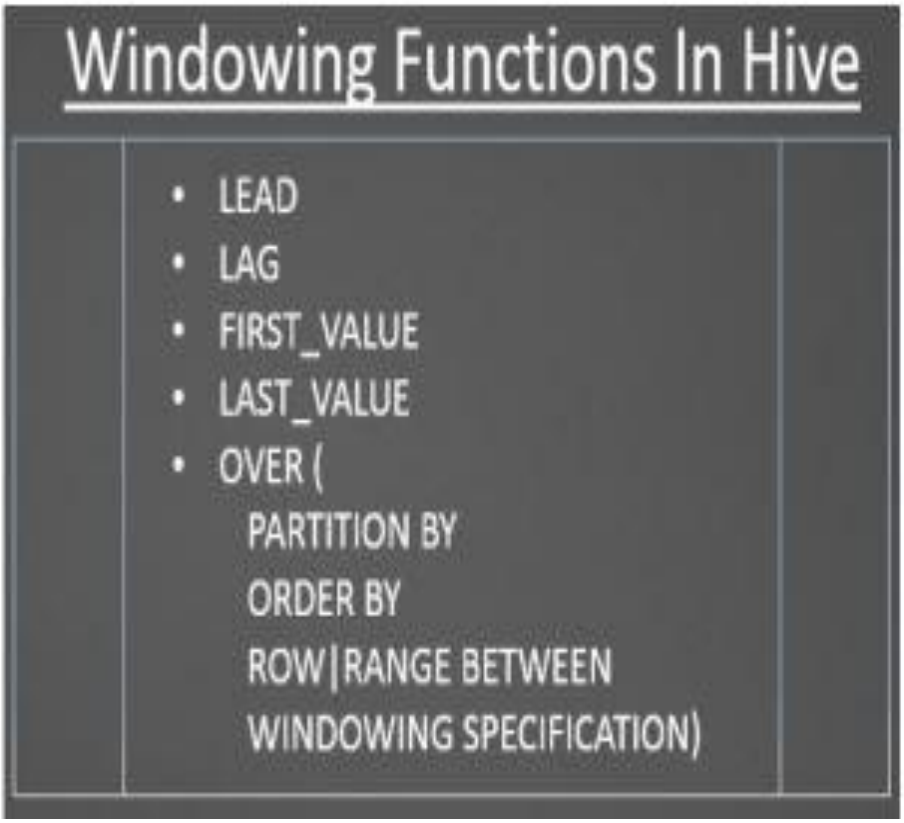
Aggregate functions

- COUNT()
- SUM()
- MIN()
- MAX()
- AVG()

Analyticalfunctions

- LEAD
- LAG
- FIRST_VALUE
- LAST_VALUE

- **OVER Clause(PARTITION BY,ORDER BY)**



Consider an Example Table:

ID	FIRST_NAME	LAST_NAME	DESIGNATION	DEPARTMENT	SALARY
1001	Jervis	Roll	Director of Sales	Sales	30000
1002	Gordon	Mattster	Marketing Manager	Sales	25000
1003	Gracie	Fronllen	Assistant Manager	Sales	25000
1004	Joelly	Wellback	Account Coordinator	Account	15000
1005	Bob	Havock	Accountant II	Account	20000
1006	Carmiae	Courage	Account Coordinator	Account	15000
1007	Cellie	Trevaskiss	Assistant Manager	Sales	25000
1008	Gally	Johnson	Manager	Account	28000
1009	Richard	Grill	Account Coordinator	Account	12000
1010	Sofia	Ketty	Sales Coordinator	Sales	20000

SELECT <columns_name>, <aggregate>(column_name) OVER (<windowing specification>) FROM <table_name>;

where,

- **column_name** – column name of the table
- **Aggregate** – Any aggregate function(s) like COUNT, AVG, MIN, MAX
- **Windowing specification** – It includes following:
  PARTITION BY – Takes a column(s) of the table as a reference.
  ORDER BY – Specified the Order of column(s) either Ascending or Descending.

PARTITION BY

Count Employees in each department

Syntax:

SELECT department, COUNT(id) OVER (PARTITION BY department) FROM emp_dept_tbl;

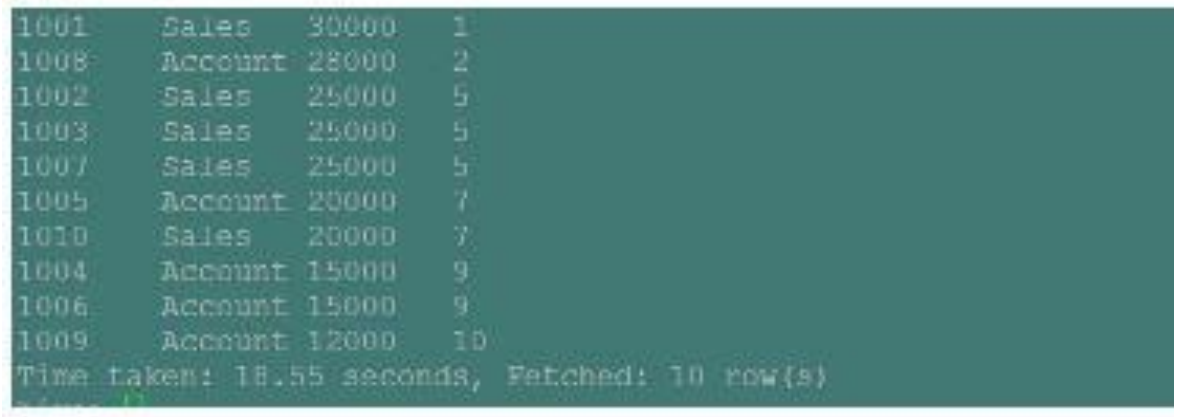
ORDER BY

Case I: Without PARTITION

Count Employee with salary descending order

Syntax

SELECT id, department, salary, COUNT(id) OVER (ORDER BY salary DESC) FROM emp_dept_tbl;

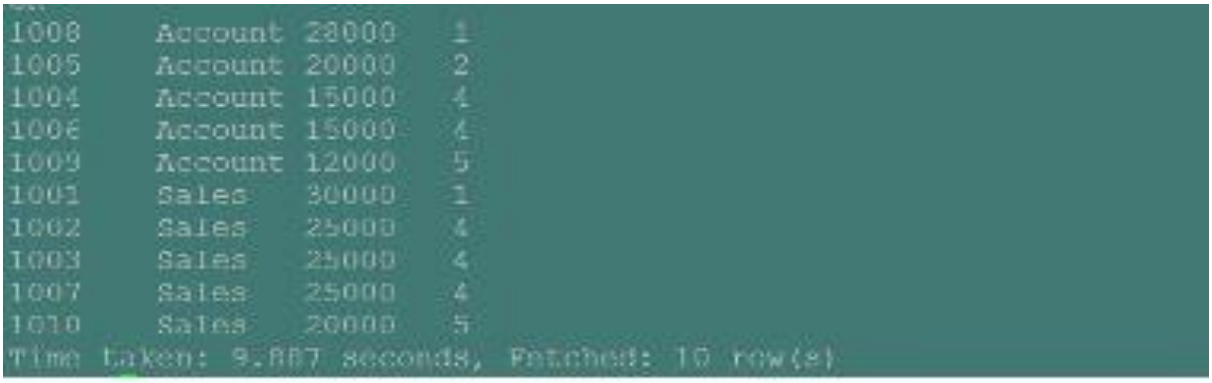


Case II: With PARTITION

Count Employees of each department order by salary

Syntax:

SELECT id, department, salary, COUNT(id) OVER (PARTITION BY department ORDER BY salary DESC) FROM emp_dept_tbl;



LEAD:

It is an analytics function used to return the data from the next set of rows. By default, the lead is of 1 row and it will return NULL in case it exceeds the current window.

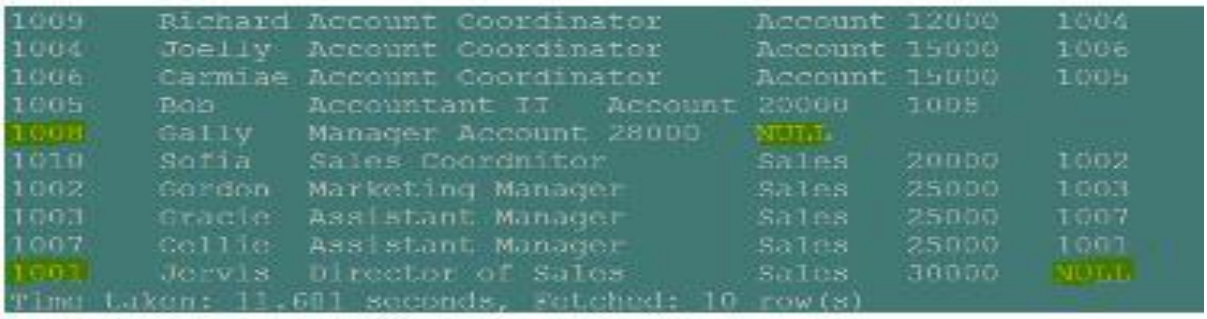
Syntax:

```
SELECT id, first_name, designation, department,salary, LEAD(id) OVER
(PARTITION BY department ORDER BY salary) FROM emp_dept_tbl;
```

OUTPUT:

ID	FIRST_NAME	DESIGNATION	DEPARTMENT	SALARY	LEAD

1009	Richard	Account Coordinator	Account	12000	1004
1004	Joelly	Account Coordinator	Account	15000	1006
1006	Carmiae	Account Coordinator	Account	15000	1005
1005	Bob	Accountant II	Account	20000	1008
1008	Gally	Manager	Account	28000	NULL
1010	Sofia	Sales Coordnitor	Sales	20000	1002
1002	Gordon	Marketing Manager	Sales	25000	1003
1003	Gracie	Assistant Manager	Sales	25000	1007
1007	Cellie	Assistant Manager	Sales	25000	1001
1001	Jervis	Director of Sales	Sales	30000	NULL



1009	Richard	Account Coordinator	Account	12000	1004
1004	Joelly	Account Coordinator	Account	15000	1006
1006	Carmiae	Account Coordinator	Account	15000	1005
1005	Bob	Accountant II	Account	20000	1008
1008	Gally	Manager Account	28000	NULL	
1010	Sofia	Sales Coordnitor	Sales	20000	1002
1002	Gordon	Marketing Manager	Sales	25000	1003
1003	Gracie	Assistant Manager	Sales	25000	1007
1007	Cellie	Assistant Manager	Sales	25000	1001
1001	Jervis	Director of Sales	Sales	30000	NULL

Time taken: 11.681 seconds, fetched: 10 row(s)

LAG:

It is the opposite of LEAD function, it returns the data from the previous set of data. By default lag is of 1 row and return NULL in case the lag for the current row is exceeded before the beginning of the window:

Syntax:

SELECT id, first_name, designation, department,salary, LAG(id) OVER (PARTITION BY department ORDER BY salary) FROM emp_dept_tbl;

OUTPUT:

ID	FIRST_NAME	DESIGNATION	DEPARTMENT	SALARY	LAG
1009	Richard	Account Coordinator	Account	12000	NULL
1004	Joelly	Account Coordinator	Account	15000	1009
1006	Carmiae	Account Coordinator	Account	15000	1004
1005	Bob	Accountant II	Account	20000	1006
1008	Gally	Manager	Account	28000	1005
1010	Sofia	Sales Coordnitor	Sales	20000	NULL
1002	Gordon	Marketing Manager	Sales	25000	1010
1003	Gracie	Assistant Manager	Sales	25000	1002
1007	Cellie	Assistant Manager	Sales	25000	1003
1001	Jervis	Director of Sales	Sales	30000	1007

FIRST_VALUE:

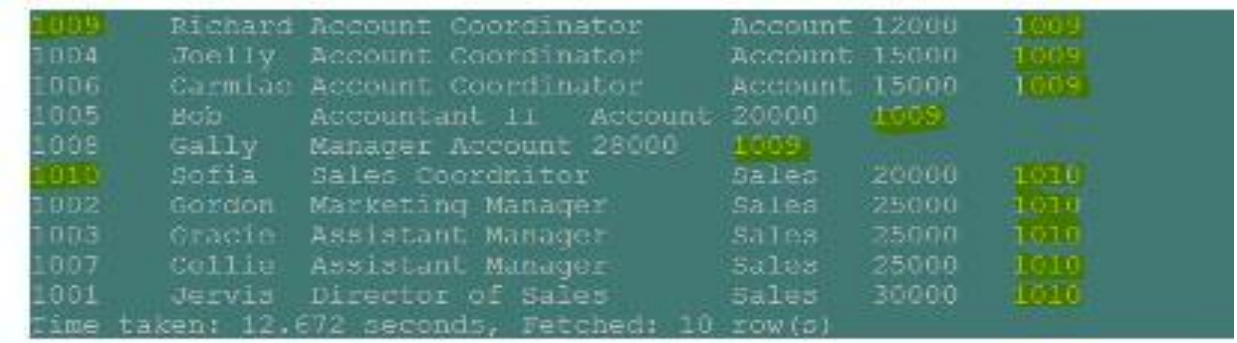
This function returns the value from the first row in the window based on the clause and assigned to all the rows of the same group:

Syntax:

SELECT id, first_name, designation, department,salary, FIRST_VALUE(id) OVER (PARTITION BY department ORDER BY salary) FROM emp_dept_tbl;

OUTPUT:

ID	FIRST_NAME	DESIGNATION	DEPARTMENT	SALARY	FIRST_VALUE
1009	Richard	Account Coordinator	Account	12000	1009
1004	Joelly	Account Coordinator	Account	15000	1009
1006	Carmiae	Account Coordinator	Account	15000	1009
1005	Bob	Accountant II	Account	20000	1009
1008	Gally	Manager	Account	28000	1009
1010	Sofia	Sales Coordnitor	Sales	20000	1010
1002	Gordon	Marketing Manager	Sales	25000	1010
1003	Gracie	Assistant Manager	Sales	25000	1010
1007	Cellie	Assistant Manager	Sales	25000	1010
1001	Jervis	Director of Sales	Sales	30000	1010



LAST_VALUE

In reverse of FIRST_VALUE, it return the value from the last row in a window based on the clause and assigned to all the rows of the same group:

OUTPUT:

ID	FIRST_NAME	DESIGNATION	DEPARTMENT	SALARY	LAST_VALUE
1009	Richard	Account Coordinator	Account	12000	1009
1004	Joelly	Account Coordinator	Account	15000	1006

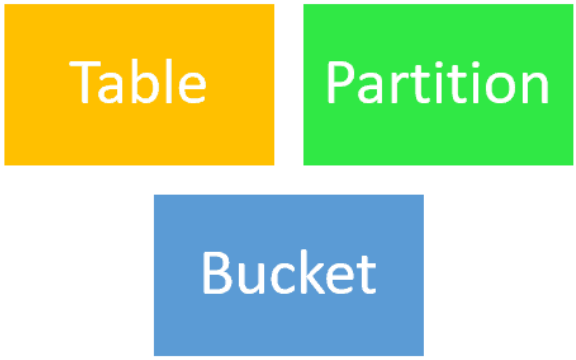
1006	Carmiae	Account Coordinator	Account	15000	1006
1005	Bob	Accountant II	Account	20000	1005
1008	Gally	Manager	Account	28000	1008
1010	Sofia	Sales Coordnitor	Sales	20000	1010
1002	Gordon	Marketing Manager	Sales	25000	1007
1003	Gracie	Assistant Manager	Sales	25000	1007
1007	Cellie	Assistant Manager	Sales	25000	1007
1001	Jervis	Director of Sales	Sales	30000	1001

Here, the first row has been assigned value 1009 but for the next two rows have assigned 1006 as both ids 1004 and 1006 have same salary 15000.

Hive Data Models:

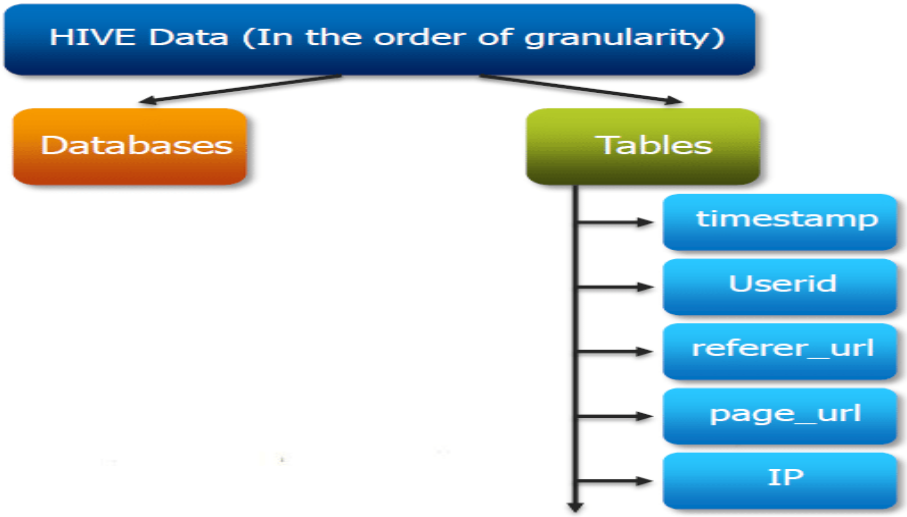
Hive organizes the data into the different data models

- Database
- Tables
- Partitions
- Buckets or clusters



1. **Tables:**

- These tables are similar to the RDBMS database tables. We can perform filters, joins, projects, and union of the hive tables. All the data of a table is stored as a directory in HDFS.



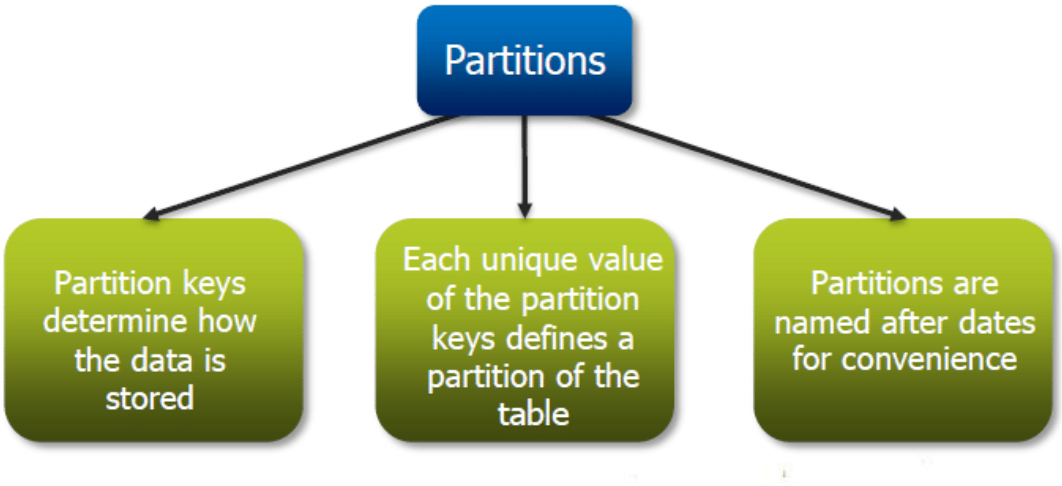
2. **Partitions:**

- Here, Hive organizes tables tables into partitions for grouping similar types of data togetherbased on the partition key.This makes it faster to do queries on slices of data

- Hive partitions are used to split the larger table into several smaller parts based on one or multiple columns (partition key, for example, date, state e.t.c). This makes it faster to do queries on slices of data.

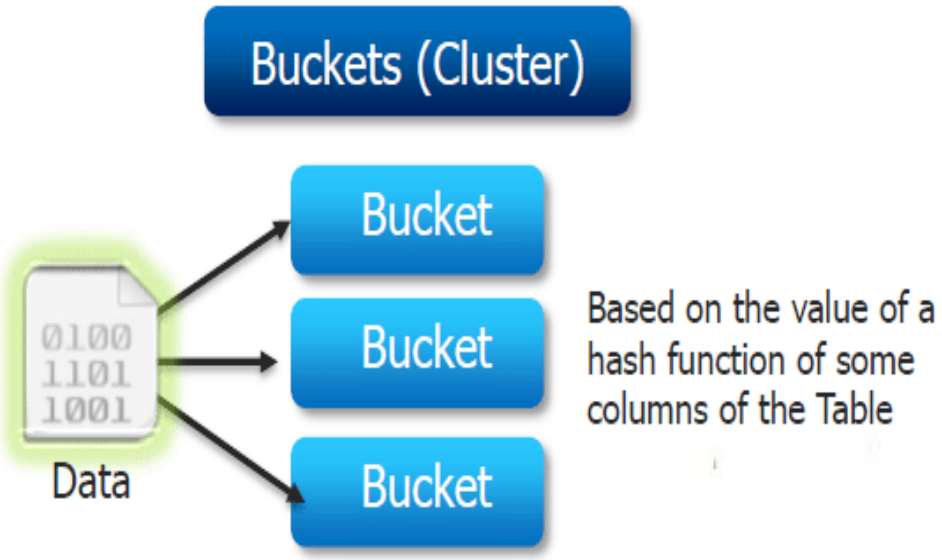
- The hive partition is similar to table partitioning available in SQL server or any other RDBMS database tables.

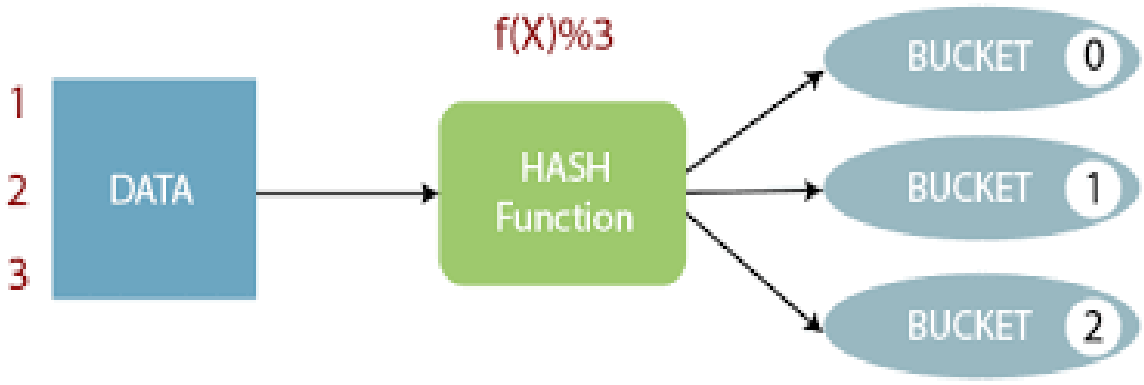
- When you load the data into the partition table, Hive internally splits the records based on the partition key and stores each partition data into a sub-directory of tables directory on HDFS.
- The name of the directory would be partition key and it's value.



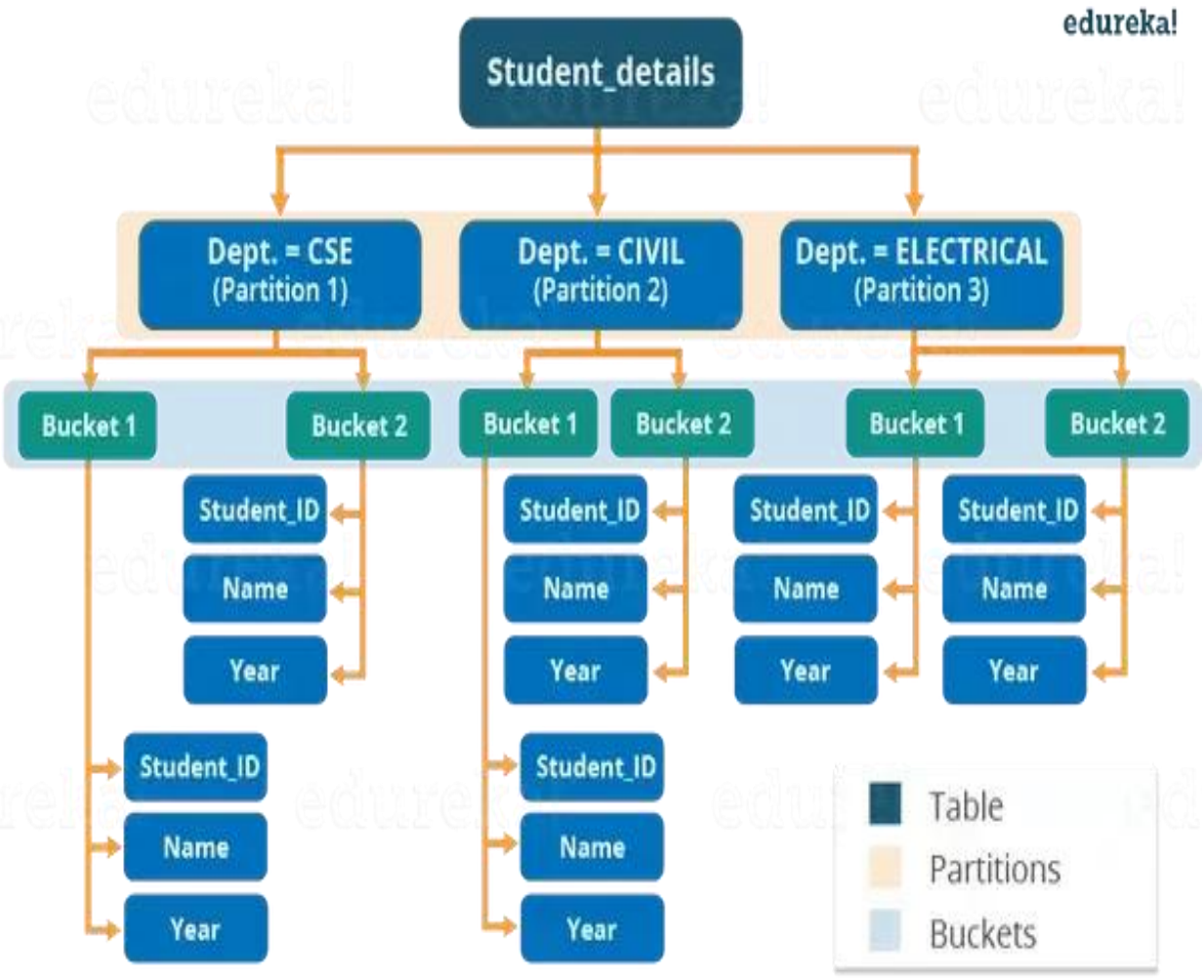
3. **Buckets:** Bucketing is a storage technique that involves grouping or portioning data based on specific criteria, such as a column value or range of values. It is commonly used in Distributed storage systems to optimize data processing and analytics tasks.

- Buckets give extra structure to the data that may be used for efficient queries. The data present in partitions are further divided into buckets buckets for efficient querying based on the hash function of a column in the table. These buckets are stored as a file in the partition directory.
- A join of two tables that are bucketed on the same columns, including the join column can be implemented as a Map-Side Join. Bucketing by used ID means we can quickly evaluate a user-based query by running it on a randomized sample of the total set of users.



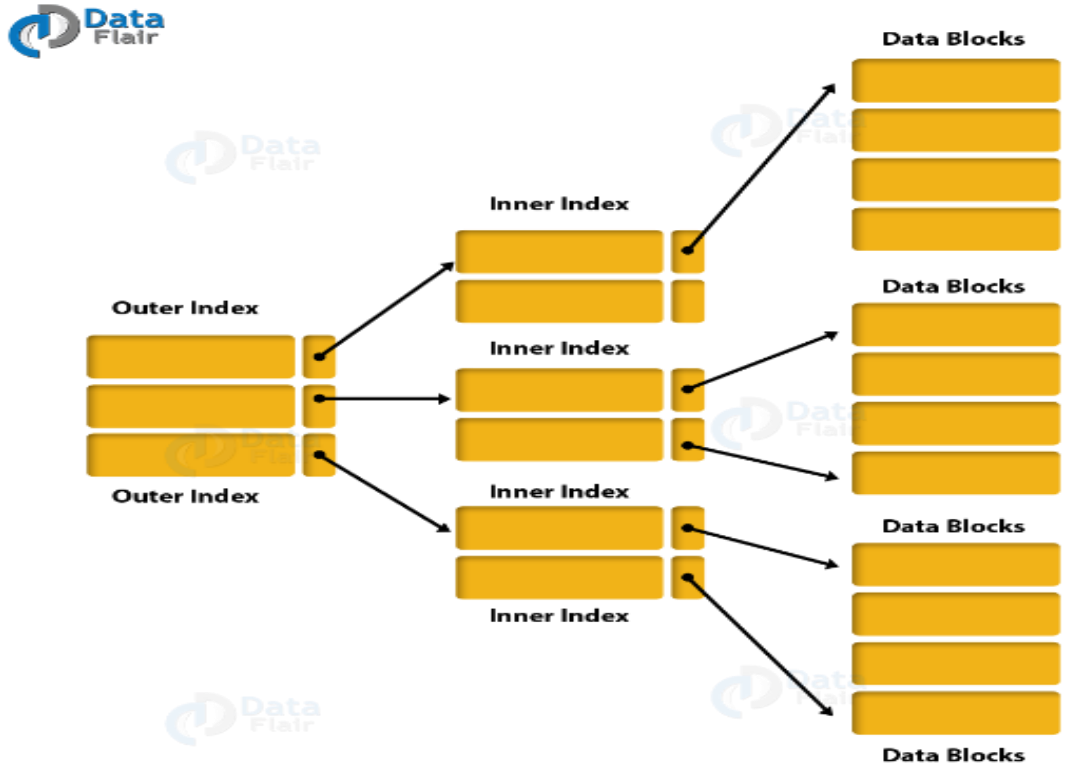


EXAMPLE (table,partition,bucket)



Indexing in HIVE:

Indexing in Hive :These are pointers to particular column name of a table.



- However, the user has to manually define the Hive index
- Basically, we are creating the pointer to particular column name of the table, wherever we are creating Hive index.
- By using the Hive index value created on the column name, any Changes made to the column present in tables are stored.

Apache Hive index Syntax

Create INDEX < INDEX_NAME> ON TABLE <TABLE_NAME(column names)>

ii. Create an Indexing in Hive

However, creating a Apache Hive index means creating a pointer on a particular column of a table. So, to create an indexing in hive.
Apache Hive Index Syntax is:

CREATE INDEX index_nameON TABLE base_table_name (col_name, ...)AS 'index.handler.class.name'[WITH DEFERRED REBUILD][IDXPROPERTIES (property_name=property_value, ...)][IN TABLEindex_table_name][PARTITIONED BY (col_name, ...)]
[[ROW FORMAT ...] STORED AS ... | STORED BY ...
][LOCATION hdfs_path][TBLPROPERTIES (...)]

Example:
create an index named **index_salary on the salary column** of the employee table.
Hence, we use the following query to create a Hive index:

hive> CREATE INDEX inedx_salary ON TABLE employee(salary) AS'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';

However, it is a pointer to the salary column. Basically, the changes are stored using an index value, if the column is modified.

When to use Hive Indexing

Under the following circumstances, we can use Indexing in Hive:

- While the dataset is very large.
- Whenever the query execution takes more amount of time than you expected.
- While we need a speedy query execution.
- While we build a data model.

Hive Index is maintained in a **separate table.** Hence, it won't affect the data inside the table, which contains the data.