

# Unit 3: Hive

# Hive

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

## **Hive is not**

- A relational database
- A design for OnLine Transaction Processing (OLTP)
- A language for real-time queries and row-level updates

## **Features of Hive**

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

# Hive

- The HIVE is developed by the Data Infrastructure team of Facebook. At Facebook, Hive's Hadoop cluster is capable to store more than 2 Petabytes of raw data, and daily it processes and loads around 15 Terabytes of data. Now it is being used by many companies also. Later, the Apache Foundation took over Hive and developed it further and made it an Open Source. It is also used and developed by other companies like Netflix, Financial Industry Regulatory Authority (FINRA), etc.

# Hive

- Hive is a declarative SQL based language, mainly used for data analysis and creating reports. Hive operates on the server-side of a cluster.
- Hive provides schema flexibility and evolution along with data summarization, querying of data, and analysis in a much easier manner.
- In Hive, we can make two types of tables – partitioned and bucketed which make it feasible to process data stored in HDFS and improves the performance as well.
- Hive tables are defined directly in the Hadoop File System(HDFS).
- In Hive, we have JDBC/ODBC drivers
- Hive is fast and scalable, and easy to learn.
- Hive has a rule-based optimizer for optimizing plans.
- Using Hive we can also execute Ad-hoc queries to analyze data.

# Hive Comparison with traditional Databases

Hive	Traditional database
Schema on READ – it's does not verify the schema while it's loaded the data	Schema on WRITE – table schema is enforced data load time i.e if the data being loaded does not conformed on schema in that case it will reject
It's very easily scalable at low cost	Not much Scalable, costly scale up.
It's based on hadoop notation that is Write once and read many times	In traditional database we can read and write many times
Record level updates is not possible in Hive	Record level updates, insertions and deletes, transactions and indexes are possible
<b>OLTP</b> (On-line Transaction Processing) is not yet supported in Hive but it's supported <b>OLAP</b> (On-line Analytical Processing)	Both <b>OLTP</b> (On-line Transaction Processing) and <b>OLAP</b> (On-line Analytical Processing) are supported in RDBMS

# HiveQL

- Hive provides a CLI for the use of Hive query language to write Hive queries. HQL syntax is usually similar to the SQL syntax most data analysts know about. The SQL language of Hive distinguishes the user from the scope of the programming Map Reduce.
- It uses well-known concepts such as rows, tables, columns and schemes from the relevant database environment to encourage learning. HiveQL's syntax is generally similar to SQL, which is familiar to most data analysts. TEXT FILE, SEQUENCE FILE, ORC and RCFILE (Column File Record) are four formats supported by Hive.

# HiveQL

- The Hive Query Language (HiveQL) is a query language for Hive to process and analyze structured data in a Metastore. This chapter explains how to use the SELECT statement with WHERE clause.
- SELECT statement is used to retrieve the data from a table. WHERE clause works similar to a condition. It filters the data using the condition and gives you a finite result. The built-in operators and functions generate an expression, which fulfils the condition.

## Syntax

Given below is the syntax of the SELECT query:

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

# Hive Tables

- Hive is a very important component or service in the Hadoop stack. It will be able to handle a huge amount of data i.e. in terms of the TB's, etc. The data will be stored on the distributed manager. In the hive, the actual data will be stored on the HDFS level. It is providing the MySQL solution on top of the HDFS data. The `hive show tables` will print the list of tables which is associated with the current database. But we can't directly trigger the command on the hive. There is a specific way or with a specific client, we can trigger the command. We can use the JDBC connection (JDBC client) or the ODBC connection (ODBC client).
- Managed tables are Hive owned tables where the entire lifecycle of the tables' data are managed and controlled by Hive. External tables are tables where Hive has loose coupling with the data. Replication Manager replicates external tables successfully to a target cluster. The managed tables are converted to external tables after replication.
- Hive supports replication of external tables with data to target cluster and it retains all the properties of external tables. The data files' permission and ownership are preserved so that the relevant external processes can continue to write in it even after failover.



# Hive Tables

## Syntax

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

```
hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,  
salary String, destination String)  
COMMENT 'Employee details'  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '\t'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE;
```

# Partitioning

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

## Adding a Partition

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

Syntax:

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec  
[LOCATION 'location1'] partition_spec [LOCATION 'location2'] ...;
```

partition\_spec:

```
: (p_column = p_col_value, p_column = p_col_value, ...)
```

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

```
hive> ALTER TABLE employee  
> ADD PARTITION (year='2012')  
> location '/2012/part2012';
```

## Renaming a Partition

The syntax of this command is as follows.

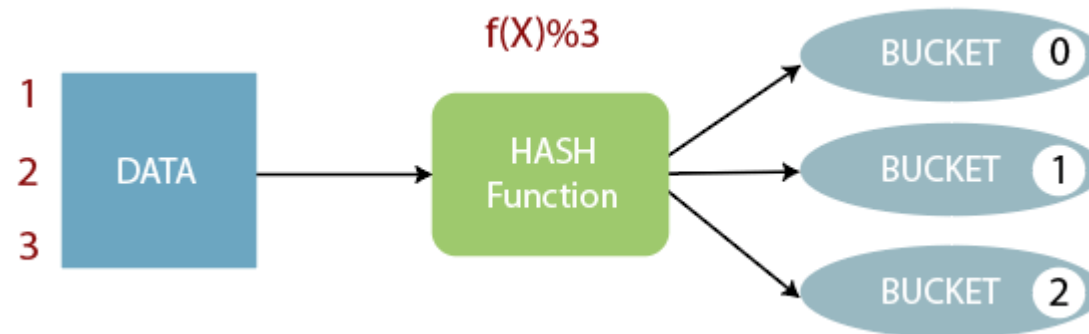
```
ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION partition_
```

The following query is used to rename a partition:

```
hive> ALTER TABLE employee PARTITION (year='1203')  
> RENAME TO PARTITION (Yoj='1203');
```

# Bucketing in Hive

- The bucketing in Hive is a data organizing technique. It is similar to partitioning in Hive with an added functionality that it divides large datasets into more manageable parts known as buckets. So, we can use bucketing in Hive when the implementation of partitioning becomes difficult. However, we can also divide partitions further in buckets.
- The concept of bucketing is based on the hashing technique.
- Here, modulus of current column value and the number of required buckets is calculated (let say,  $F(x) \% 3$ ).
- Now, based on the resulted value, the data is stored into the corresponding bucket.



# HIVE Script & UDF

- HIVE UDF (User Defined Functions) allow the user to extend **HIVE Query Language**. Once the UDF is added in the **HIVE script**, it works like a normal built-in function. To check which all UDFs are loaded in current hive session, we use SHOW command
- Basically, we can use two different interfaces for writing Apache Hive User Defined Functions.
- Simple API
- Complex API
- As long as our function reads and returns primitive types, we can use the simple API (org.apache.hadoop.hive ql.exec.UDF). In other words, it means basic **Hadoop** & Hive writable types. Such as Text, IntWritable, LongWritable, DoubleWritable, etc.

# HIVE Script & UDF

- a. Simple API
- Basically, with the simpler UDF API, building a Hive User Defined Function involves little more than writing a class with one function (evaluate). However, let's see an example to understand it well:

## Simple API – Hive UDF Example

```
1. class SimpleUDFExample extends UDF
2. {
3.     public Text evaluate(Text input)
4.     {
5.         return new Text("Hello " + input.toString());
6.     }
7. }
```

# HIVE Script & UDF

- b. Complex API
- However, to write code for objects that are not writable types. Like struct, map and array types. Hence the `org.apache.hadoop.hive.ql.udf.generic.GenericUDF` API offers a way.
- In addition, for the function arguments, it needs us to manually manage object inspectors. Also, to verify the number and types of the arguments we receive. To be more specific, an object inspector offers a consistent interface for underlying object types.
- Hence, that different object implementation can all be accessed in a consistent way from within hive. For example, we could implement a struct as a Map so long as you provide a corresponding object inspector.

# Joins in Hive

- Joins are used to retrieve various outputs using multiple tables by combining them based on particular columns. Now, for the tables to be in Hive, we are required to create the tables and load the data in each table. We are going to use two tables (customer and product) here for understanding the purpose.
- This will give the cross product of both the table's data as output.
- **Full Join**
- Full Join without match condition will give the cross product of both tables.
- **Left Join**
- All the rows from the left table are joined with matched rows from the right table.
- **Inner Join**
- If the inner join is used without the “on” clause, it will give the cross product as the output.
- **Right Join**
- All the rows from the right table are matched with left table rows.

# Hive Query Optimizers

- To run queries on petabytes of data we all know that hive is a query language which is similar to SQL built on **Hadoop ecosystem**. So, there are several Hive optimization techniques to improve its performance which we can implement when we run our hive queries.
- There are several types of Hive Query Optimization techniques are available while running our hive queries to improve Hive performance with some Hive Performance tuning techniques.
- The different Hive Query Optimizers are
- **Tez Execution Engine** – Hive Optimization Techniques, to increase the Hive performance of our hive query by using our execution engine as Tez. On defining Tez, it is a new application framework built on **Hadoop Yarn**.
- That executes complex-directed acyclic graphs of general data processing tasks. However, we can consider it to be a much more flexible and powerful successor to the map-reduce framework.



- Hive Partition – Hive Optimization Techniques, Hive reads all the data in the directory Without partitioning. Further, it applies the query filters on it. Since all data has to be read this is a slow as well as expensive.

Also, users need to filter the data on specific column values frequently. Although, users need to understand the domain of the data on which they are doing analysis, to apply the partitioning in the Hive.

Basically, by Partitioning all the entries for the various columns of the dataset are segregated and stored in their respective partition.

- **Bucketing in Hive** – Hive Optimization Techniques, let's suppose a scenario. At times, there is a huge dataset available. However, after partitioning on a particular field or fields, the partitioned file size doesn't match with the actual expectation and remains huge.
- Still, we want to manage the partition results into different parts. Thus, to solve this issue of partitioning, Hive offers Bucketing concept. Basically, that allows the user to divide table data sets into more manageable parts.

- Vectorization In Hive – Hive Optimization Techniques, to improve the performance of operations we use Vectorized query execution. Here operations refer to scans, aggregations, filters, and joins. It happens by performing them in batches of 1024 rows at once instead of single row each time.

- Cost-Based Optimization in Hive – Hive Optimization Techniques, before submitting for final execution Hive optimizes each Query's logical and physical execution plan. Although, until now these optimizations are not based on the cost of the query.

However, CBO, performs, further optimizations based on query cost in a recent addition to Hive. That results in potentially different decisions: how to order joins, which type of join to perform, the degree of parallelism and others.

- Hive Index – Hive Optimization Techniques, one of the best ways is Indexing. To increase your query performance indexing will definitely help. Basically, for the original table use of indexing will create a separate called index table which acts as a reference.

# Data Type

All the data types in Hive are classified into four types, given as follows:

Column Types

Literals

Null Values

Complex Types

- Column Types

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

- Integral Types

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

The following table depicts various INT data types:

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

## String Types

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

The following table depicts various CHAR data types:

Data Type	Length
VARCHAR	1 to 65355
CHAR	255

- **Timestamp**

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

- **Dates**

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

- **Decimals**

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

```
DECIMAL(precision, scale)
decimal(10,0)
```



- Union Types

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

```
UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>
{0:1}
{1:2.0}
{2:["three","four"]}
{3:{"a":5,"b":"five"}}
{2:["six","seven"]}
{3:{"a":8,"b":"eight"}}
{0:9}
{1:10.0}
```

- **Literals**

The following literals are used in Hive:

- **Floating Point Types**

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

- **Decimal Type**

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

## **Null Value**

Missing values are represented by the special value NULL.

- **Complex Types**

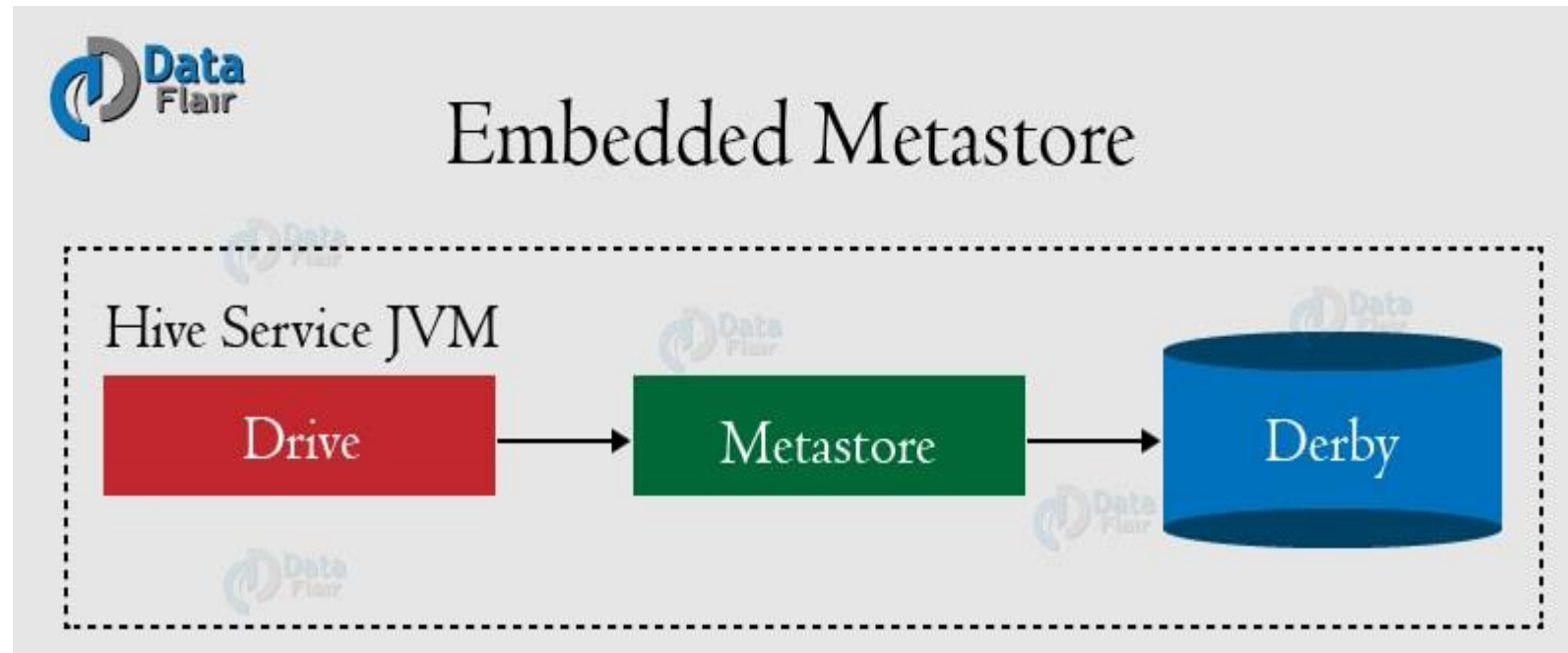
The Hive complex data types are as follows:

- **Arrays**

**Syntax:** ARRAY<data\_type>

# MetaStore

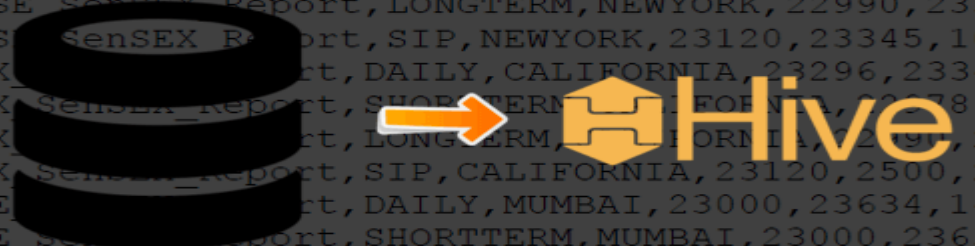
- Metastore is the **central repository of Apache Hive metadata**. It stores metadata for Hive tables (like their schema and location) and partitions in a relational database. It provides client access to this information by using metastore service API.



# Hive Tables

- **Managed tables are Hive owned tables where the entire lifecycle of the tables' data are managed and controlled by Hive.** External tables are tables where Hive has loose coupling with the data. Replication Manager replicates external tables successfully to a target cluster. The managed tables are converted to external tables after replication.
- **What's the difference between managed table and external table in hive?**
- the difference is , when you drop a table, if it is managed table hive deletes both data and meta data,if it is external table Hive only deletes metadata. by default It is Managed table . If you want to create a external table ,you will use external keyword.

- **How to import data in Hive using Sqoop**
- First you should import the RDBMS tables in HDFS- Check this link for details
- Convert the data into ORC file format
- Then create Hive table and import the HDFS data to Hive table using the below command



The diagram illustrates the process of importing data from a Relational Database Management System (RDBMS) into Apache Hive. On the left, a stack of three black cylinders represents the RDBMS. An orange arrow points from this stack to the right, where the Apache Hive logo (a stylized orange 'H') is displayed. Below the arrow, the text 'Import RDBMS Data to Hive Using Sqoop' is written in white and yellow. The background of the entire image is a dark gray terminal window showing a list of stock market data records, such as '12121,NYSE\_SenSEX\_Report,DAILY,NEWYORK,23296,23345,10'.

```
12121,NYSE_SenSEX_Report,DAILY,NEWYORK,23296,23345,10
12122,NYSE_SenSEX_Report,SHORTTERM,NEWYORK,22978,23345,11
12123,NYSE_SenSEX_Report,LONGTERM,NEWYORK,22990,23345,12
12124,NYSE_SenSEX_Report,SIP,NEWYORK,23120,23345,10
12125,CSX_SenSEX_Report,DAILY,CALIFORNIA,23296,23345,22
12126,CSX_SenSEX_Report,SHORTTERM,CALIFORNIA,22978,23345,23
12127,CSX_SenSEX_Report,LONGTERM,CALIFORNIA,22990,23345,24
12128,CSX_SenSEX_Report,SIP,CALIFORNIA,23120,2500,25
12129,NSE_SenSEX_Report,DAILY,MUMBAI,23000,23634,14
12130,NSE_SenSEX_Report,SHORTTERM,MUMBAI,23000,23634,15
12131,NSE_SenSEX_Report,LONGTERM,MUMBAI,23000,23634,16
12132,NSE_SenSEX_Report,SIP,MUMBAI,23120,2500,25
16161,BSE_SENSEX_REPORT,DAILY,MUMBAI,45632,45644,19
16162,BSE_SENSEX_REPORT,SHORTTERM,MUMBAI,45632,45644,18
16163,BSE_SENSEX_REPORT,LONGTERM,MUMBAI,45632,45644,20
16164,BSE_SENSEX_REPORT,SIP,MUMBAI,45632,45644,21
```

**Import RDBMS Data to Hive Using Sqoop**

[HdfsTutorial.com/Blog](http://HdfsTutorial.com/Blog)

# Importing Data from Files into Hive Tables

- Apache Hive is an SQL-like tool for analyzing data in HDFS. Data scientists often want to import data into Hive from existing text-based files exported from spreadsheets or databases. These file formats often include tab-separated values (TSV), comma-separated values (CSV), raw text, JSON, and others. Having the data in Hive tables enables easy access to it for subsequent modeling steps, the most common of which is feature generation, which we discuss in Chapter 5, “Data Munging with Hadoop.”
- Once data are imported and present as a Hive table, it is available for processing using a variety of tools including Hive’s SQL query processing, Pig, or Spark.

# Hive supports two types of tables.

- The first type of table is an *internal table* and is fully managed by Hive. If you delete an internal table, both the definition in Hive *and* the data will be deleted. Internal tables are stored in an optimized format such as ORC and thus provide a performance benefit.
- second type of table is an *external table* that is not managed by Hive. External tables use only a metadata description to access the data in its raw form. If you delete an external table, only the definition (metadata about the table) in Hive is deleted and the actual data remain intact. External tables are often used when the data resides outside of Hive (i.e., some other application is also using/creating/managing the files), or the original data need to remain in the underlying location even after the table is deleted

# Hive Queries

- [Order by query](#)
- [Group by query](#)
- [Sort by](#)
- [Cluster By](#)
- [Distribute By](#)

1) The **ORDER BY** syntax in HiveQL is similar to the syntax of ORDER BY in SQL language.

- Order by is the clause we use with “SELECT” statement in Hive queries, which helps sort data. Order by clause use columns on Hive tables for sorting particular column values mentioned with Order by. For whatever the column name we are defining the order by clause the query will select and display results by ascending or descending order the particular column values.



1) The **ORDER BY** syntax in HiveQL is similar to the syntax of ORDER BY in SQL language. Order by is the clause we use with “SELECT” statement in Hive queries, which helps sort data. Order by clause use columns on Hive tables for sorting particular column values mentioned with Order by. For whatever the column name we are defining the order by clause the query will select and display results by ascending or descending order the particular column values.

```
SELECT * FROM employees_guru ORDER BY Department;
```

```
hive> SELECT * FROM employees_guru ORDER BY Department;
Query ID = hduser_20151117170229_6e8af657-126b-470f-8b88
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined by job: 1
In order to change the amount of memory per reducer, set hive.exec.reducers.bytes-allocated to a new value.
In order to limit the maximum number of reducers, set hive.exec.reducers.max to a new value.
In order to set a constant number of reducers:
set mapred.reduce.tasks=<number>
Starting Job = job_201511171701_0001, Tracking URL = http://localhost:8020/jobdetails.jsp?jobid=job_201511171701_0001
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/killall.py -f -s 1171701_0001
Hadoop job information for Stage-1: number of mappers: 1
2015-11-17 17:02:39,820 Stage-1 map = 0%, reduce = 0%
2015-11-17 17:02:44,008 Stage-1 map = 100%, reduce = 0%
2015-11-17 17:02:52,078 Stage-1 map = 100%, reduce = 33%
2015-11-17 17:02:53,085 Stage-1 map = 100%, reduce = 100%
MapReduce Total elapsed time: 2 seconds 40 msec
Ended Job = job_201511171701_0001
MapReduce Jobs: 1
Stage-Stage-1: Map: 1 Red: 1 Cumulative CPU: 2.04
FS Write: 380 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 40 msec
OK
103      Animesh  26      Bangalore  25000.0  ADMIN
104      Anirudh  27      bangalore  27000.0  ADMIN
106      Ramesh   30      Goa       24000.0  FINANCE
101      Rajesh   27      Bangalore  20000.0  HR
102      Rajiv    28      Delhi     30000.0  HR
107      Sravanthi 32      Chennai   20000.0  IT
108      Sravan   31      Mumbai    60000.0  IT
109      Suresh   32      Kolkata   20000.0  IT
110      Ravi     28      bangalore  20000.0  IT
105      Santosh  33      bangalore  25000.0  PR
111      Syam     33      bangalore  25000.0  PR
Time taken: 25.224 seconds, Fetched: 11 row(s)
```

order by query on "employees\_guru" table

order by query output

- **Group by query:**
- Group by clause use columns on Hive tables for grouping particular column values mentioned with the group by. For whatever the column name we are defining a “groupby” clause the query will selects and display results by grouping the particular column values.
- For example, in the below screen shot it’s going to display the total count of employees present in each department. Here we have “Department” as Group by value.

```
hive> SELECT Department, count(*) FROM employees_guru GROUP BY Department;
Query ID = huser_20151105155307_1574cd2b-866e-437a-8d14-637e02b7e515
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified, assuming 1 task per map.
In order to change the average size of the reducers:
    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:
    set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0005, Tracking URL = http://localhost:5003
Kill Command = /usr/local/hadoop-1.2.1/libexec/./bin/hadoop job -kill job_201511051442_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-11-05 15:53:19,229 Stage-1 map = 0%, reduce = 0%
2015-11-05 15:53:21,235 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.13 sec
2015-11-05 15:53:28,277 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.13 sec
2015-11-05 15:53:29,281 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 2.13 sec
MapReduce Total cumulative CPU time: 2 seconds 130 msec
Ended Job = job_201511051442_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.13 sec HDFS Read: 7 KB Write: 1 KB
Total MapReduce CPU Time Spent: 2 seconds 130 msec
OK
ADMIN      2
FINANCE    1
HR          2
IT          4
PR          2
Time taken: 23.057 seconds, Fetched: 5 row(s)
```

Groupby query on "employees\_guru" size: 1

Group by query output

- **Sort by:**
- Sort by clause performs on column names of Hive tables to sort the output. We can mention DESC for sorting the order in descending order and mention ASC for Ascending order of the sort.
- In this sort by it will sort the rows before feeding to the reducer. Always sort by depends on column types.
- For instance, if column types are numeric it will sort in numeric order if the columns types are string it will sort in lexicographical order.

```
hive> Select * from employees guru SORT BY id DESC;
Query ID = hdus-1-20151105164027_55bf2f64-6f5b-4764-b94e-e03f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified
In order to change the average size of the reduce tasks:
  set hive.exec.reducers.bytesinreduce=<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:
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0007, Tracking URL = http://1
Kill Command = /usr/local/hadoop-1.2.1/libexec/./bin/hadoop
Hadoop job information for Stage-1: Map: 1, Reducers: 1; num
2015-11-05 16:40:34,093 Stage-1: Map: 1, Reducers: 1; num
2015-11-05 16:40:36,098 Stage-1: Map: 1, Reducers: 1; num
2015-11-05 16:40:44,145 Stage-1: Map: 1, Reducers: 1; num
MapReduce Total cumulative CPU time: 1.62 seconds 620 msec
Ended Job = job_201511051442_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1, Reducers: 1, Cumulative CPU: 1.62 sec
Total MapReduce CPU Time Spent: 1 seconds 620 msec

OK
111      Syam      33      bangalore      25000.0  PR
110      Ravi       28      bangalore      20000.0  IT
109      Suresh     32      Kolkata 20000.0  IT
108      Sravan     31      Mumbai  60000.0  IT
107      Sravanthi  32      Chennai 20000.0  IT
106      Ramesh     30      Goa      24000.0  FINANCE
105      Santosh    33      bangalore      25000.0  PR
104      Anirudh    27      bangalore      27000.0  ADMIN
103      Animesh    26      Bangalore      25000.0  ADMIN
102      Rajiv     28      Delhi   30000.0  HR
101      Rajesh     27      Bangalore      20000.0  HR
Time taken: 18.088 seconds, Fetched: 11 row(s)
```

sort by query

sort by output on  
"employees\_guru"  
table

- **Cluster By:**
- Cluster By used as an alternative for both Distribute BY and Sort BY clauses in Hive-QL.
- Cluster BY clause used on tables present in Hive. Hive uses the columns in Cluster by to distribute the rows among reducers. Cluster BY columns will go to the multiple reducers.
- It ensures sorting orders of values present in multiple reducers
- Syntax-

```
SELECT Id, Name from employees_guru CLUSTER BY Id;
```

- **Distribute By:**
- Distribute BY clause used on tables present in Hive. Hive uses the columns in Distribute by to distribute the rows among reducers. All Distribute BY columns will go to the same reducer.
- Query-

```
SELECT Id, Name from employees_guru DISTRIBUTE BY Id;
```

# Hive Thrift Server

- HiveServer is an optional service that allows a remote [client](#) to submit requests to Hive, using a variety of programming languages, and retrieve results. HiveServer is built on Apache Thrift™
- Hive Service is nothing but daemon which runs on your client node which sends requests to Hive Server.
- Thrift is an RPC framework for building cross-platform services. Its stack consists of 4 layers: Server, Transport, Protocol, and Processor
- HiveServer2 (HS2) is a service that enables clients to execute queries against Hive. HiveServer2 is the successor to HiveServer1 which has been deprecated. HS2 supports multi-client concurrency and authentication. It is designed to provide better support for open API clients like JDBC and ODBC.

Thank You