**UNIT-4**

**Structured Data Processing Tools**

**Topics:**

**Hive**: Installation, Running Hive, HiveQL, Tables, Querying Data, User Defined functions

**Sqoop**: Introduction, generate code, Database import, working with imported data, Importing large objects, performing an exports.

**What is HIVE**

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

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).

## Features of Hive

These are the following features of Hive:

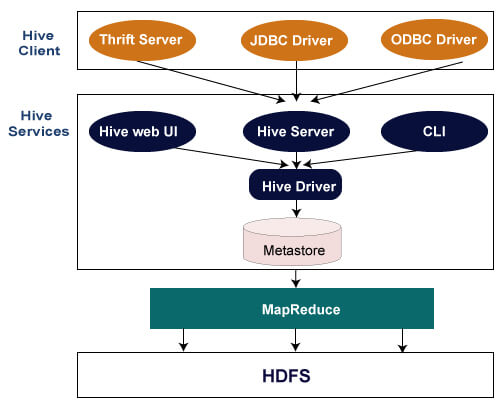
* Hive is fast and scalable.
* It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
* It is capable of analyzing large datasets stored in HDFS.
* It allows different storage types such as plain text, RCFile, and HBase.
* It uses indexing to accelerate queries.
* It can operate on compressed data stored in the Hadoop ecosystem.
* It supports user-defined functions (UDFs) where user can provide its functionality.

## Limitations of Hive

* Hive is not capable of handling real-time data.
* It is not designed for online transaction processing.
* Hive queries contain high latency.

# Hive Architecture

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



## Hive Client

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

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

## Hive Services

The following are the services provided by Hive:-

* Hive CLI - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
* Hive Web User Interface - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
* Hive MetaStore - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
* Hive 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 in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

**Apache Hive Installation**

In this section, we will perform the Hive installation.

Pre-requisite

* **Java Installation** - Check whether the Java is installed or not using the following command.

1. $ java -version

* **Hadoop Installation** - Check whether the Hadoop is installed or not using the following command.

1. $hadoop version

If any of them is not installed in your system, follow the below link to install

Steps to install Apache Hive

**Step1**

* Download the Apache Hive tar file.

[**http://mirrors.estointernet.in/apache/hive/hive-1.2.2/**](http://mirrors.estointernet.in/apache/hive/hive-1.2.2/)

* DUnzip the downloaded tar file.

**tar -xvf apache-hive-1.2.2-bin.tar.gz**

**Step2**

* DOpen the bashrc file.

**$ sudo nano ~/.bashrc**

**Step3**

* DNow, provide the following HIVE\_HOME path.

**export HIVE\_HOME=/home/codegyani/apache-hive-1.2.2-bin**

**export PATH=$PATH:/home/codegyani/apache-hive-1.2.2-bin/bin**

**Step4**

* DUpdate the environment variable.

**$ source ~/.bashrc**

**Step5**

* DLet's start the hive by providing the following command.

**$ hive**



HIVE Data Types

Hive data types are categorized in numeric types, string types, misc types, and complex types. A list of Hive data types is given below.

## 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

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

## Date/Time Types

**TIMESTAMP**

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

**DATES**

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

## String Types

**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, which specifies that the maximum number of characters allowed in the character string.

**CHAR**

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

## Complex Type

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

HiveQL - Operators

The HiveQL operators facilitate to perform various arithmetic and relational operations. Here, we are going to execute such type of operations on the records of the below table:

There are four types of operators in Hive:

* Relational Operators
* Arithmetic Operators
* Logical Operators
* Complex Operators

## Arithmetic Operators in Hive

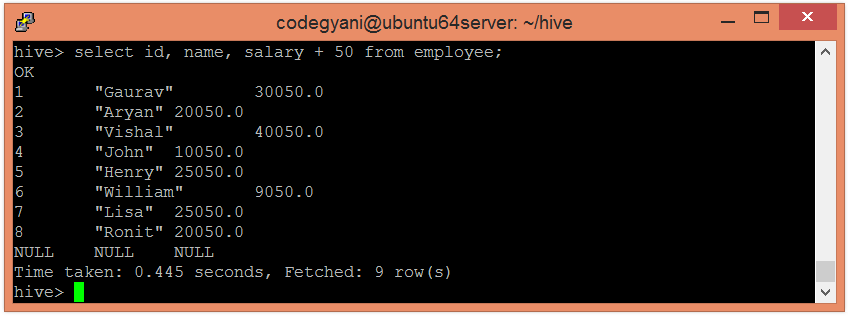
In Hive, the arithmetic operator accepts any numeric type. The commonly used arithmetic operators are:

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

### Examples of Arithmetic Operator in Hive

* Let's see an example to increase the salary of each employee by 50.

1. hive**>** select id, name, salary + 50 from employee;



## Relational Operators in Hive

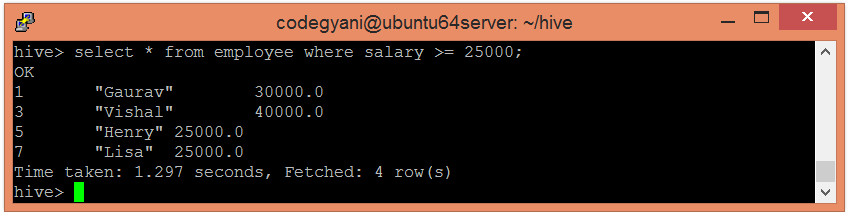
In Hive, the relational operators are generally used with clauses like Join and Having to compare the existing records. The commonly used relational operators are: -

|  |  |
| --- | --- |
| **Operator** | **Description** |
| A=B | It returns true if A equals B, otherwise false. |
| A <> B, A !=B | It returns null if A or B is null; true if A is not equal to B, otherwise false. |
| A<B | It returns null if A or B is null; true if A is less than B, otherwise false. |
| A>B | It returns null if A or B is null; true if A is greater than B, otherwise false. |
| A<=B | It returns null if A or B is null; true if A is less than or equal to B, otherwise false. |
| A>=B | It returns null if A or B is null; true if A is greater than or equal to B, otherwise false. |
| A IS NULL | It returns true if A evaluates to null, otherwise false. |
| A IS NOT NULL | It returns false if A evaluates to null, otherwise true. |

### Examples of Relational Operator in Hive

* Let's see an example to fetch the details of the employee having salary>=25000.

1. hive**>** select \* from employee where salary **>**= 25000;



## Logical Operators

The operators are logical expressions. All of them return either TRUE or FALSE.

|  |  |  |
| --- | --- | --- |
| **Operators** | **Operands** | **Description** |
| A AND B | boolean | TRUE if both A and B are TRUE, otherwise FALSE. |
| A && B | boolean | Same as A AND B. |
| A OR B | boolean | TRUE if either A or B or both are TRUE, otherwise FALSE. |
| A || B | boolean | Same as A OR B. |
| NOT A | boolean | TRUE if A is FALSE, otherwise FALSE. |
| !A | boolean | Same as NOT A. |

### Example

The following query is used to retrieve employee details whose Department is TP and Salary is more than Rs 40000.

hive> SELECT \* FROM employee WHERE Salary>40000 && Dept=TP;

On successful execution of the query, you get to see the following response:

+------+--------------+-------------+-------------------+--------+

| ID | Name | Salary | Designation | Dept |

+------+--------------+-------------+-------------------+--------+

|1201 | Gopal | 45000 | Technical manager | TP |

+------+--------------+-------------+-------------------+--------+

## Complex Operators

These operators provide an expression to access the elements of Complex Types.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Operand** | **Description** |
| A[n] | A is an Array and n is an int | It returns the nth element in the array A. The first element has index 0. |
| M[key] | M is a Map<K, V> and key has type K | It returns the value corresponding to the key in the map. |
| S.x | S is a struct | It returns the x field of S. |

HiveQL - Functions

The Hive provides various in-built functions to perform mathematical and aggregate type operations. Here, we are going to execute such type of functions on the records of the below table:

## Mathematical Functions in Hive

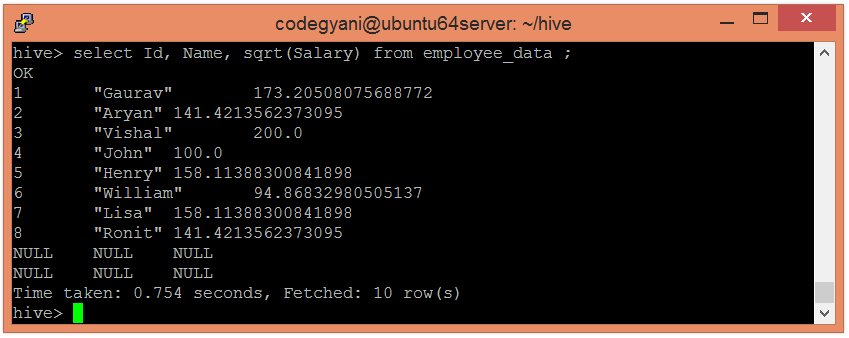
The commonly used mathematical functions in the hive are: -

|  |  |  |
| --- | --- | --- |
| **Return type** | **Functions** | **Description** |
| BIGINT | round(num) | It returns the BIGINT for the rounded value of DOUBLE num. |
| BIGINT | floor(num) | It returns the largest BIGINT that is less than or equal to num. |
| BIGINT | ceil(num), ceiling(DOUBLE num) | It returns the smallest BIGINT that is greater than or equal to num. |
| DOUBLE | exp(num) | It returns exponential of num. |
| DOUBLE | ln(num) | It returns the natural logarithm of num. |
| DOUBLE | log10(num) | It returns the base-10 logarithm of num. |
| DOUBLE | sqrt(num) | It returns the square root of num. |
| DOUBLE | abs(num) | It returns the absolute value of num. |
| DOUBLE | sin(d) | It returns the sin of num, in radians. |
| DOUBLE | asin(d) | It returns the arcsin of num, in radians. |
| DOUBLE | cos(d) | It returns the cosine of num, in radians. |
| DOUBLE | acos(d) | It returns the arccosine of num, in radians. |
| DOUBLE | tan(d) | It returns the tangent of num, in radians. |
| DOUBLE | atan(d) | It returns the arctangent of num, in radians. |

### Example of Mathematical Functions in Hive

* Let's see an example to fetch the square root of each employee's salary.

1. hive**>** select Id, Name, sqrt(Salary) from employee\_data ;



## Aggregate Functions in Hive

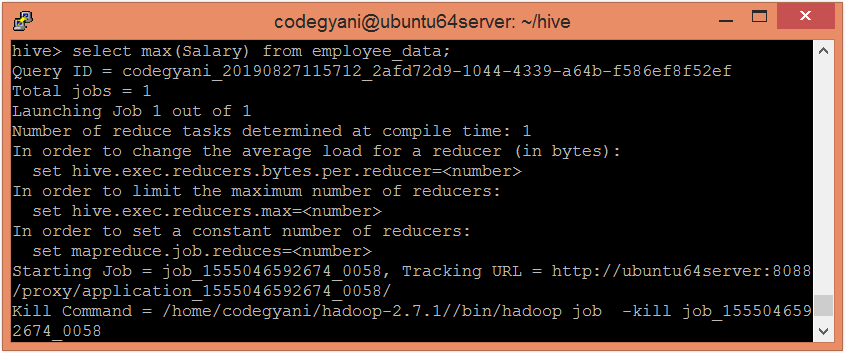
In Hive, the aggregate function returns a single value resulting from computation over many rows. Let''s see some commonly used aggregate functions: -

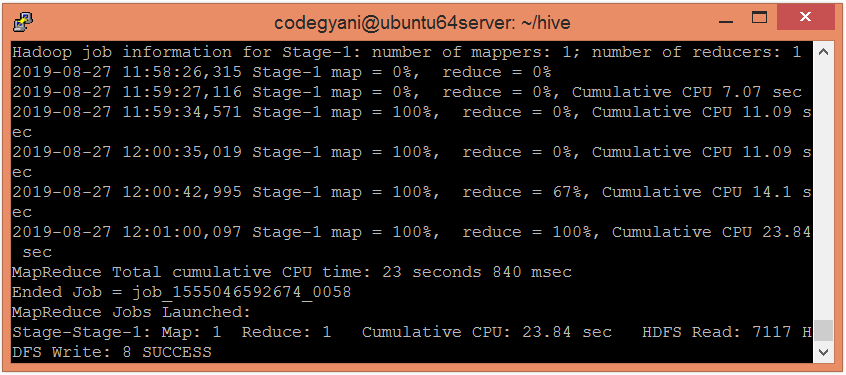
|  |  |  |
| --- | --- | --- |
| **Return Type** | **Operator** | **Description** |
| BIGINT | count(\*) | It returns the count of the number of rows present in the file. |
| DOUBLE | sum(col) | It returns the sum of values. |
| DOUBLE | sum(DISTINCT col) | It returns the sum of distinct values. |
| DOUBLE | avg(col) | It returns the average of values. |
| DOUBLE | avg(DISTINCT col) | It returns the average of distinct values. |
| DOUBLE | min(col) | It compares the values and returns the minimum one form it. |
| DOUBLE | max(col) | It compares the values and returns the maximum one form it. |

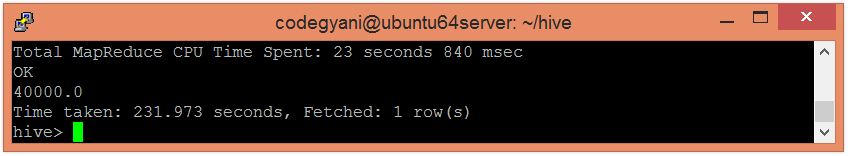
### Examples of Aggregate Functions in Hive

* Let's see an example to fetch the maximum salary of an employee.

1. hive**>** select max(Salary) from employee\_data;







## Other built-in Functions in Hive

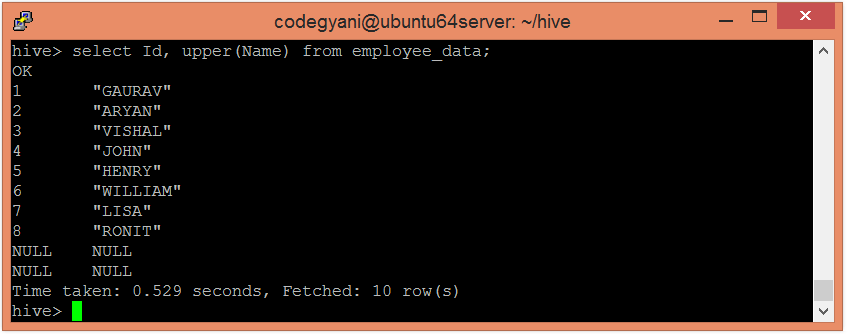
The following are some other commonly used in-built functions in the hive: -

|  |  |  |
| --- | --- | --- |
| **Return Type** | **Operator** | **Description** |
| INT | length(str) | It returns the length of the string. |
| STRING | reverse(str) | It returns the string in reverse order. |
| STRING | concat(str1, str2, ...) | It returns the concatenation of two or more strings. |
| STRING | substr(str, start\_index) | It returns the substring from the string based on the provided starting index. |
| STRING | substr(str, int start, int length) | It returns the substring from the string based on the provided starting index and length. |
| STRING | upper(str) | It returns the string in uppercase. |
| STRING | lower(str) | It returns the string in lowercase. |
| STRING | trim(str) | It returns the string by removing whitespaces from both the ends. |
| STRING | ltrim(str) | It returns the string by removing whitespaces from left-hand side. |
| TRING | rtrim(str) | It returns the string by removing whitespaces from right-hand side. |

### Examples of other in-built Functions in Hive

* Let's see an example to fetch the name of each employee in uppercase.

1. select Id, upper(Name) from employee\_data;

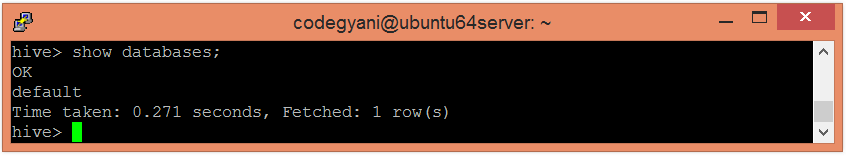


Hive - Create Database

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

* Initially, we check the default database provided by Hive. So, to check the list of existing databases, follow the below command: -

hive**>** show databases;

 Here, we can see the existence of a default database provided by Hive.

* Let's create a new database by using the following command: -

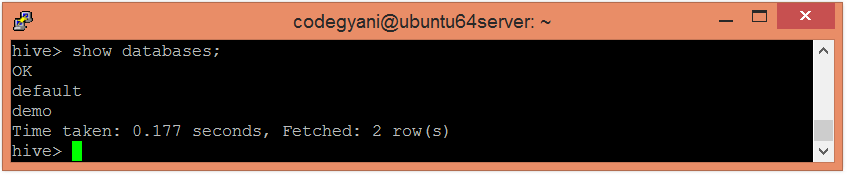
1. hive**>** create database demo;



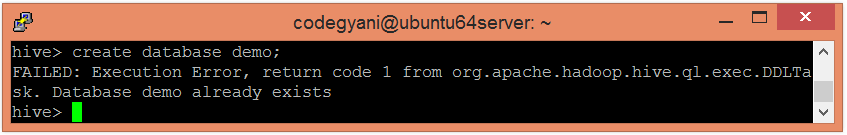
So, a new database is created.

* Let's check the existence of a newly created database.

1. hive**>** show databases;

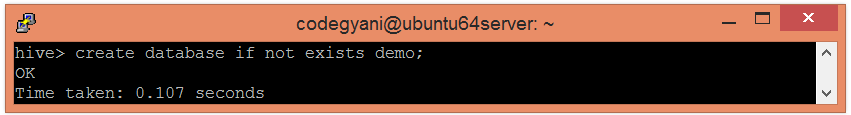


* Each database must contain a unique name. If we create two databases with the same name, the following error generates: -



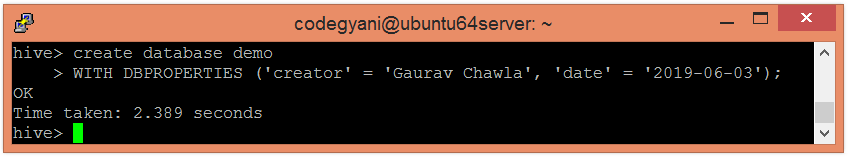
* If we want to suppress the warning generated by Hive on creating the database with the same name, follow the below command: -

1. hive**>** create a database if not exists demo;



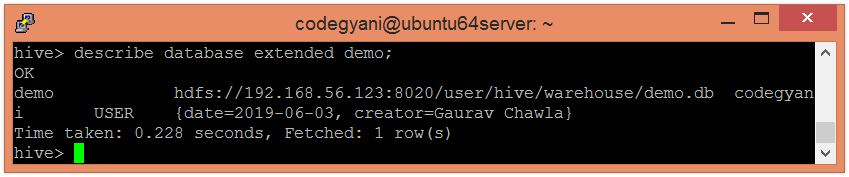
* Hive also allows assigning properties with the database in the form of key-value pair.

1. hive**>**create the database demo
2. **>**WITH DBPROPERTIES ('creator' = 'Gaurav Chawla', 'date' = '2019-06-03');



* Let's retrieve the information associated with the database.

1. hive**>** describe database extended demo;

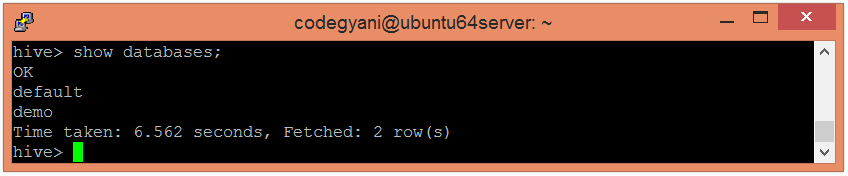


Hive - Drop Database

In this section, we will see various ways to drop the existing database.

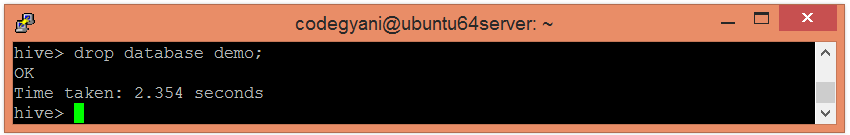
* Let's check the list of existing databases by using the following command: -

1. hive**>** show databases;



* Now, drop the database by using the following command.

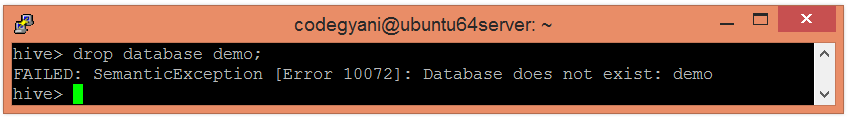
1. hive**>** drop database demo;



* Let's check whether the database is dropped or not.

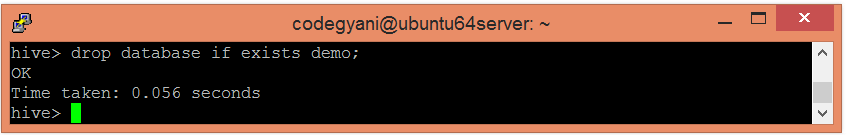
1. hive**>** show databases;

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



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

1. hive**>** drop database if exists demo;



* In Hive, it is not allowed to drop the database that contains the tables directly. In such a case, we can drop the database either by dropping tables first or use Cascade keyword with the command.
* Let's see the cascade command used to drop the database:-

hive**>** drop database if exists demo cascade;

# Hive - Create Table

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

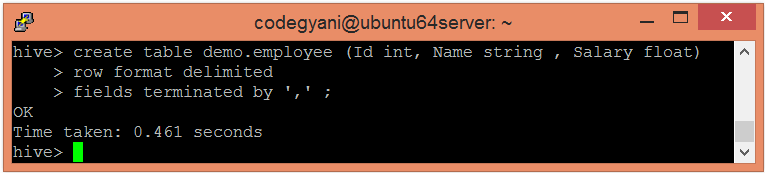
* Internal table
* External table

## Internal Table

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

* Let's create an internal table by using the following command:-

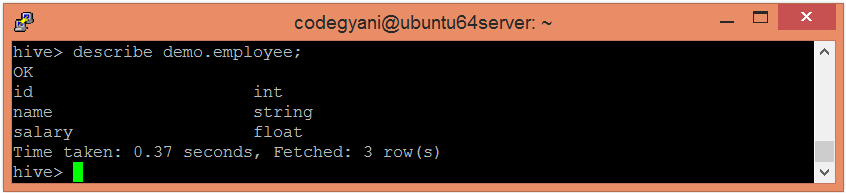
1. hive**>** create table demo.employee (Id int, Name string , Salary float)
2. row format delimited
3. fields terminated by ',' ;



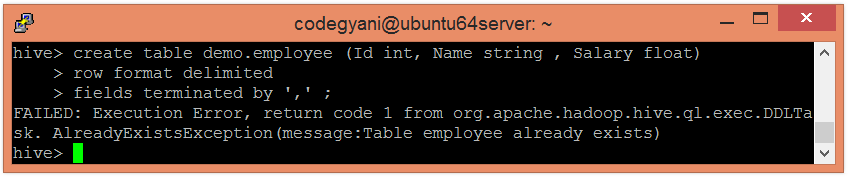
Here, the command also includes the information that the data is separated by ','.

* Let's see the metadata of the created table by using the following command:-

1. hive**>** describe demo.employee

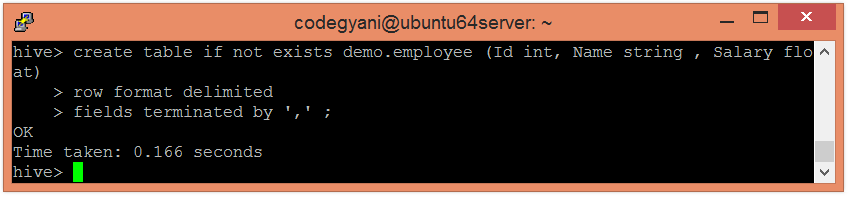


* Let's see the result when we try to create the existing table again.



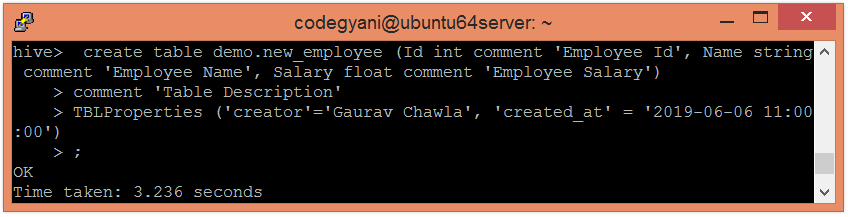
In such a case, the exception occurs. If we want to ignore this type of exception, we can use **if not exists** command while creating the table.

1. hive**>** create table if not exists demo.employee (Id int, Name string , Salary float)
2. row format delimited
3. fields terminated by ',' ;



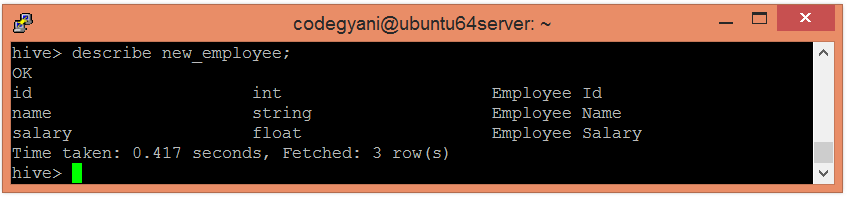
* While creating a table, we can add the comments to the columns and can also define the table properties.

1. hive**>** create table demo.new\_employee (Id int comment 'Employee Id', Name string comment 'Employee Name', Salary float comment 'Employee Salary')
2. comment 'Table Description'
3. TBLProperties ('creator'='Gaurav Chawla', 'created\_at' = '2019-06-06 11:00:00');



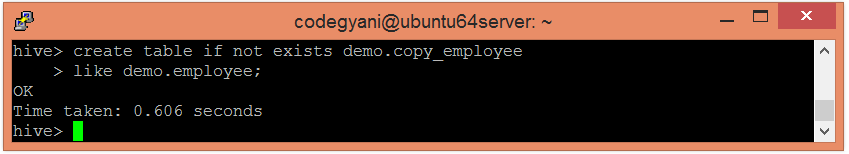
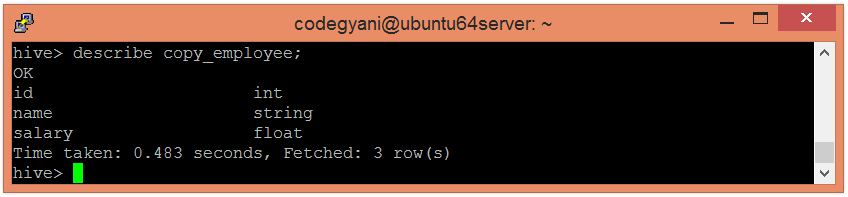
* Let's see the metadata of the created table by using the following command: -

1. hive**>** describe new\_employee;



* Hive allows creating a new table by using the schema of an existing table.

1. hive**>** create table if not exists demo.copy\_employee
2. like demo.employee;

Here, we can say that the new table is a copy of an existing table.

## External Table

The external table allows us to create and access a table and a data externally. The **external** keyword is used to specify the external table, whereas the **location** keyword is used to determine the location of loaded data.

As the table is external, the data is not present in the Hive directory. Therefore, if we try to drop the table, the metadata of the table will be deleted, but the data still exists.

To create an external table, follow the below steps: -

* Let's create a directory on HDFS by using the following command: -

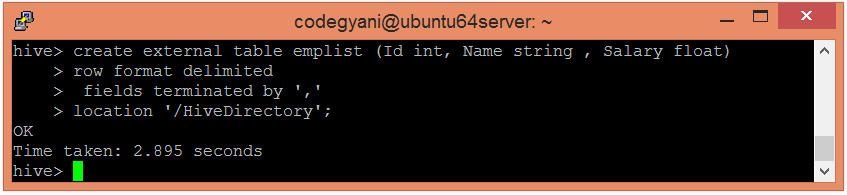
1. hdfs dfs -mkdir /HiveDirectory

* Now, store the file on the created directory.

1. hdfs dfs -put hive/emp\_details /HiveDirectory

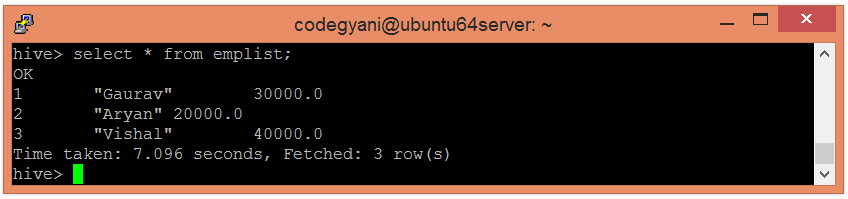
* Let's create an external table using the following command: -

1. hive**>** create external table emplist (Id int, Name string , Salary float)
2. row format delimited
3. fields terminated by ','
4. location '/HiveDirectory';



* Now, we can use the following command to retrieve the data: -

1. select \* from emplist;

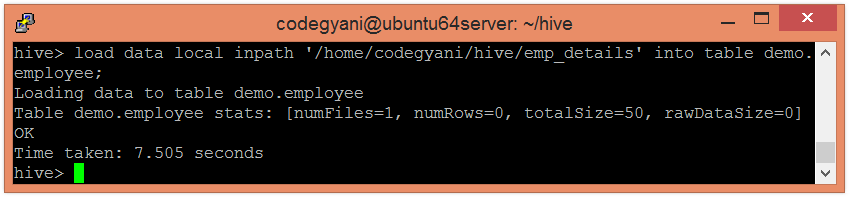


Hive - Load Data

Once the internal table has been created, the next step is to load the data into it. So, in Hive, we can easily load data from any file to the database.

* Let's load the data of the file into the database by using the following command: -

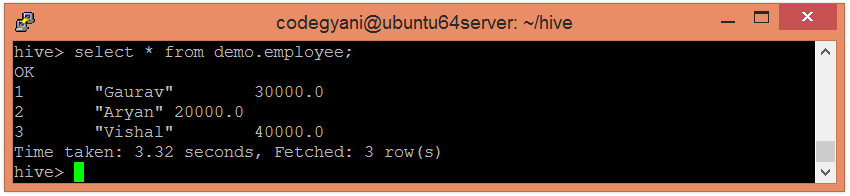
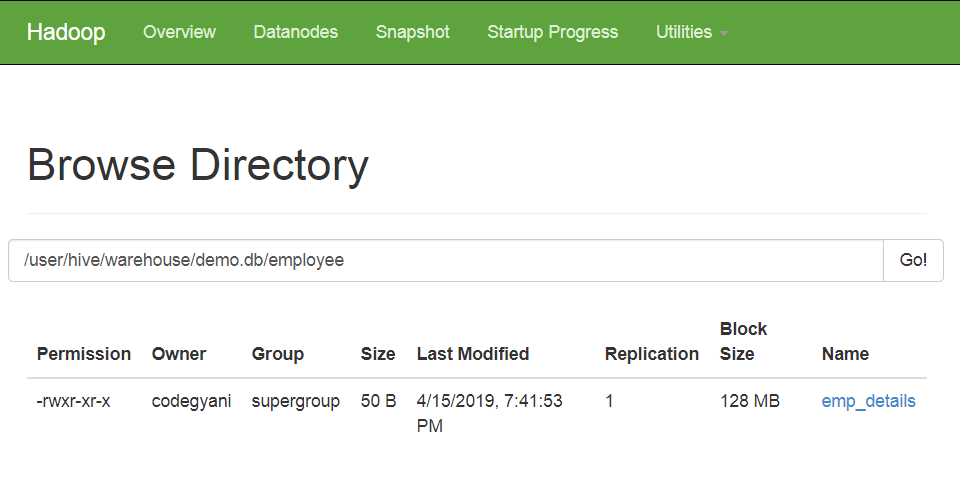
1. load data local inpath '/home/codegyani/hive/emp\_details' into table demo.employee;



Here, **emp\_details** is the file name that contains the data.

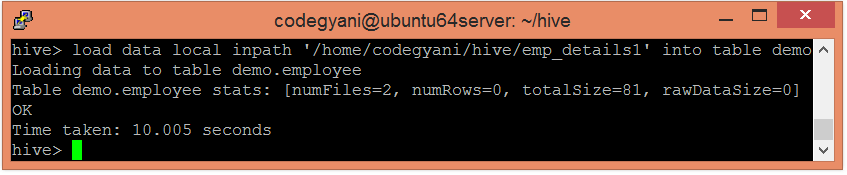
* Now, we can use the following command to retrieve the data from the database.

1. select \* from demo.employee;

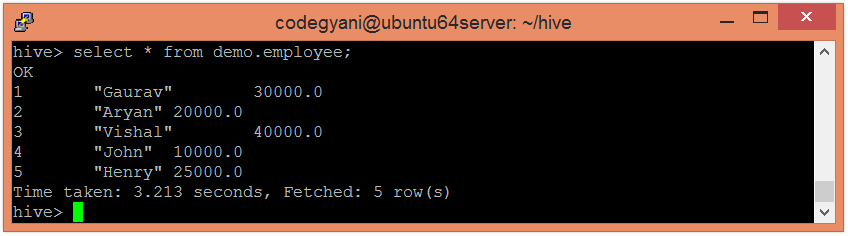
  


* If we want to add more data into the current database, execute the same query again by just updating the new file name.

1. load data local inpath '/home/codegyani/hive/emp\_details1' into table demo.employee;



* Let's check the data of an updated table: -



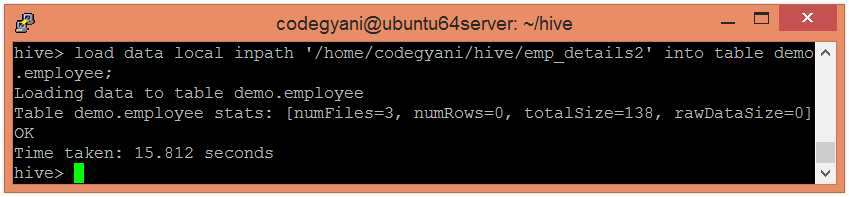
* In Hive, if we try to load unmatched data (i.e., one or more column data doesn't match the data type of specified table columns), it will not throw any exception. However, it stores the Null value at the position of unmatched tuple.
* Let's add one more file to the current table. This file contains the unmatched data.



Here, the third column contains the data of string type, and the table allows the float type data. So, this condition arises in an unmatched data situation.

* Now, load the data into the table.

1. load data local inpath '/home/codegyani/hive/emp\_details2' into table demo.employee;



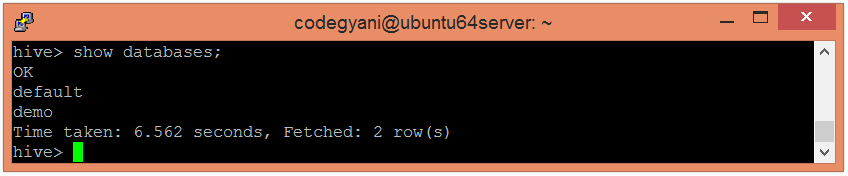
Here, data loaded successfully.

Hive - Drop Table

Hive facilitates us to drop a table by using the SQL **drop table** command. Let's follow the below steps to drop the table from the database.

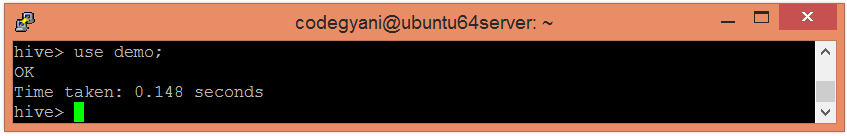
* Let's check the list of existing databases by using the following command: -

1. hive**>** show databases;



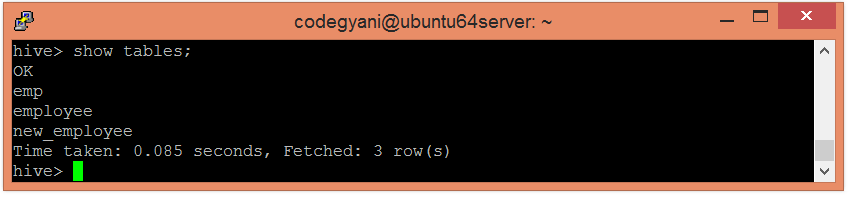
* Now select the database from which we want to delete the table by using the following command: -

1. hive**>** use demo;



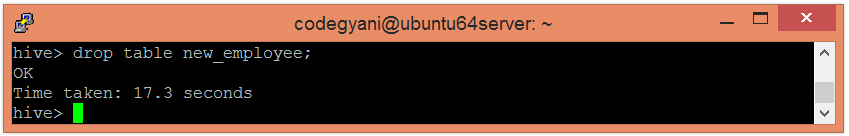
* Let's check the list of existing tables in the corresponding database.

1. hive**>** show tables;



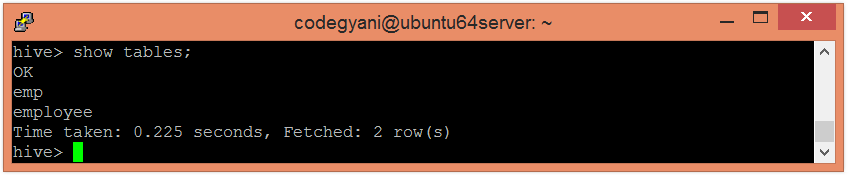
* Now, drop the table by using the following command: -

1. hive**>** drop table new\_employee;



* Let's check whether the table is dropped or not.

1. hive**>** show tables;



As we can see, the table **new\_employee** is not present in the list. Hence, the table is dropped successfully.

# Hive - Alter Table

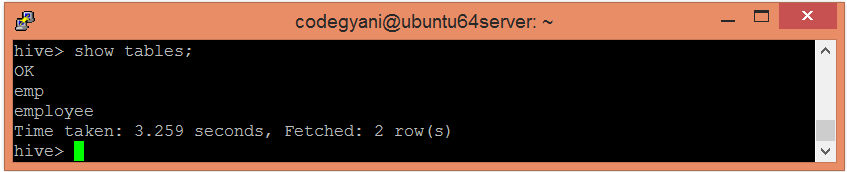
In Hive, we can perform modifications in the existing table like changing the table name, column name, comments, and table properties. It provides SQL like commands to alter the table.

### Rename a Table

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

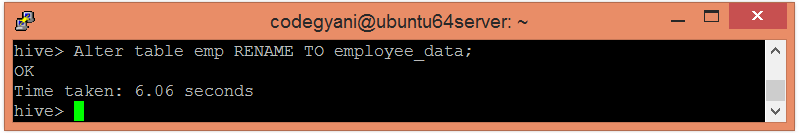
1. Alter table old\_table\_name rename to new\_table\_name;

* Let's see the existing tables present in the current database.

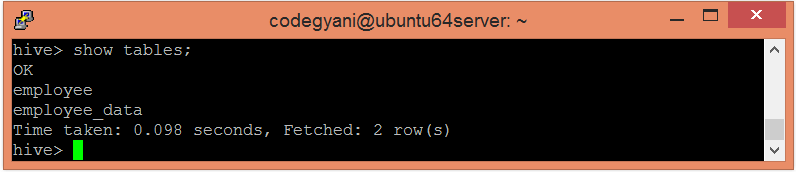


* Now, change the name of the table by using the following command: -

1. Alter table emp rename to employee\_data;



* Let's check whether the name has changed or not.



Here, we got the desired output.

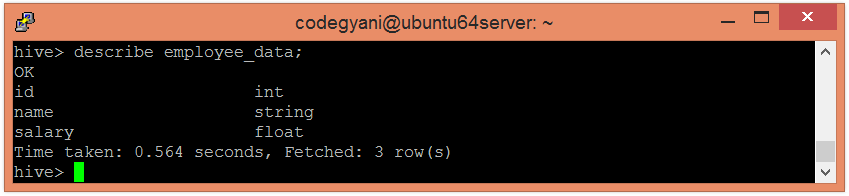
### Adding column

In Hive, we can add one or more columns in an existing table by using the following signature: -

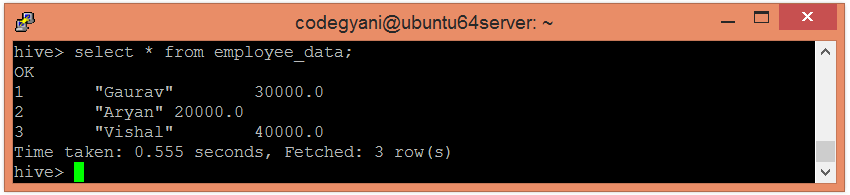
Play Video

1. Alter table table\_name add columns(column\_name datatype);

* Let's see the schema of the table.

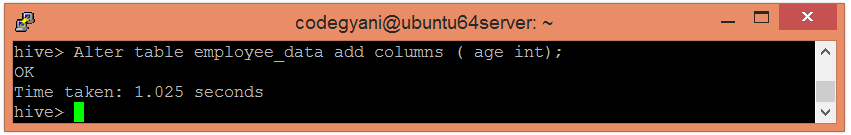


* Let's see the data of columns exists in the table.

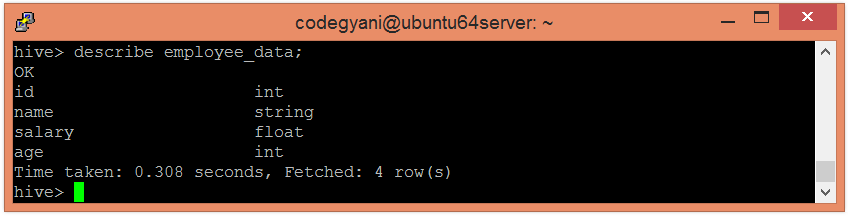


* Now, add a new column to the table by using the following command: -

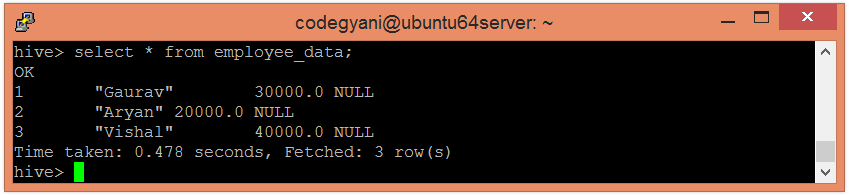
1. Alter table employee\_data add columns (age int);



* Let's see the updated schema of the table.



* Let's see the updated data of the table.



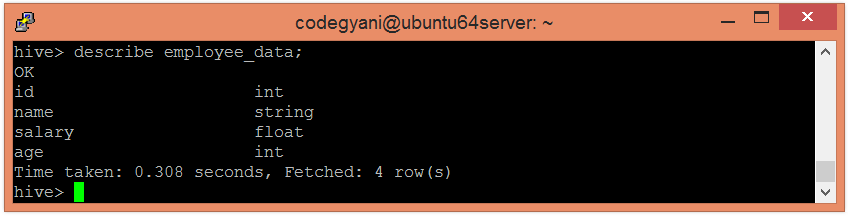
As we didn't add any data to the new column, hive consider NULL as the value.

### Change Column

In Hive, we can rename a column, change its type and position. Here, we are changing the name of the column by using the following signature: -

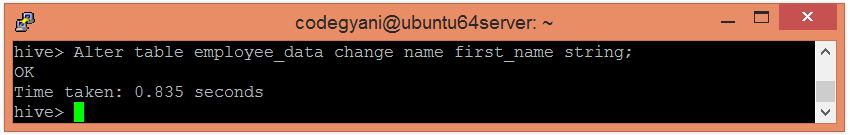
1. Alter table table\_name change old\_column\_name new\_column\_name  datatype;

* Let's see the existing schema of the table.

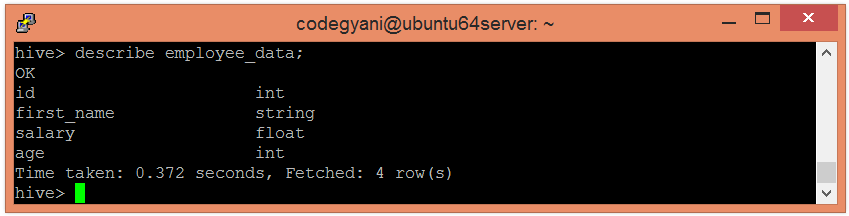


* Now, change the name of the column by using the following command: -

1. Alter table employee\_data change name first\_name string;



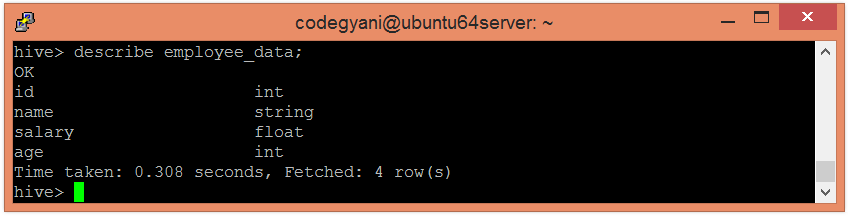
* Let's check whether the column name has changed or not.



### Delete or Replace Column

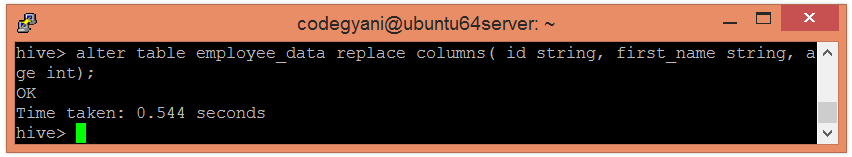
Hive allows us to delete one or more columns by replacing them with the new columns. Thus, we cannot drop the column directly.

* Let's see the existing schema of the table.

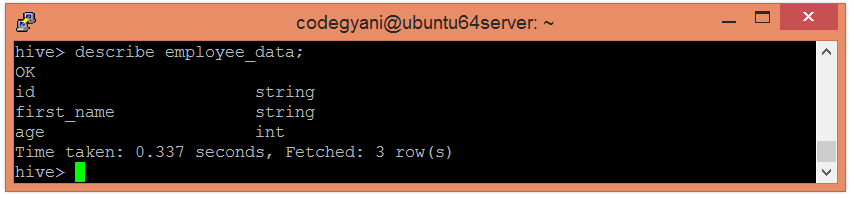


* Now, drop a column from the table.

1. alter table employee\_data replace columns( id string, first\_name string, age int);



* Let's check whether the column has dropped or not.



Here, we got the desired output.

# Partitioning in Hive

The partitioning in Hive means dividing the table into some parts based on the values of a particular column like date, course, city or country. The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

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

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

The partitioning in Hive can be executed in two ways -

* [Static partitioning](https://www.javatpoint.com/partitioning-in-hive#Static)
* [Dynamic partitioning](https://www.javatpoint.com/dynamic-partitioning-in-hive)

## Static Partitioning

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

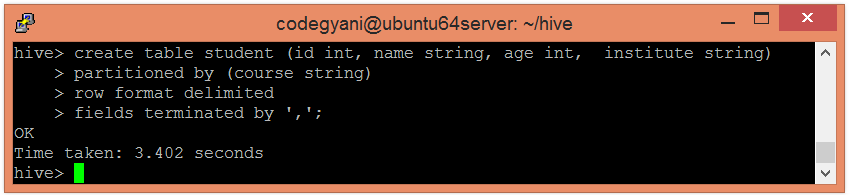
**Example of Static Partitioning**

* First, select the database in which we want to create a table.

1. hive**>** use test;

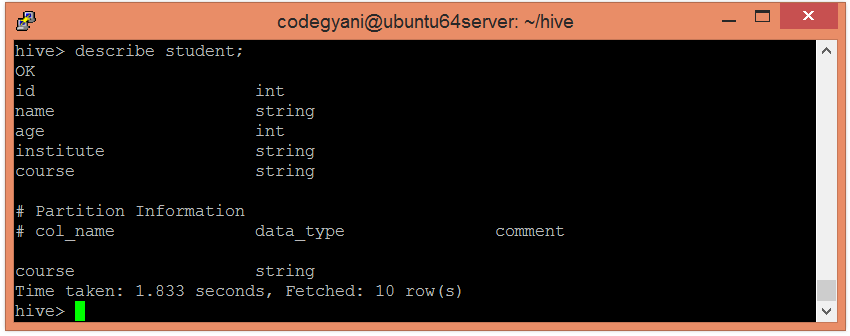
* Create the table and provide the partitioned columns by using the following command: -

1. hive**>** create table student (id int, name string, age int,  institute string)
2. partitioned by (course string)
3. row format delimited
4. fields terminated by ',';



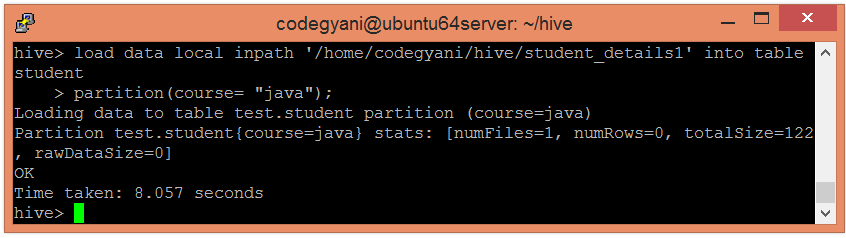
* Let's retrieve the information associated with the table.

1. hive**>** describe student;



* Load the data into the table and pass the values of partition columns with it by using the following command: -

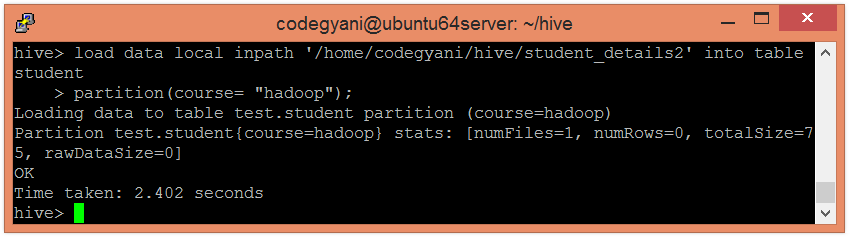
1. hive**>** load data local inpath '/home/codegyani/hive/student\_details1' into table student
2. partition(course= "java");



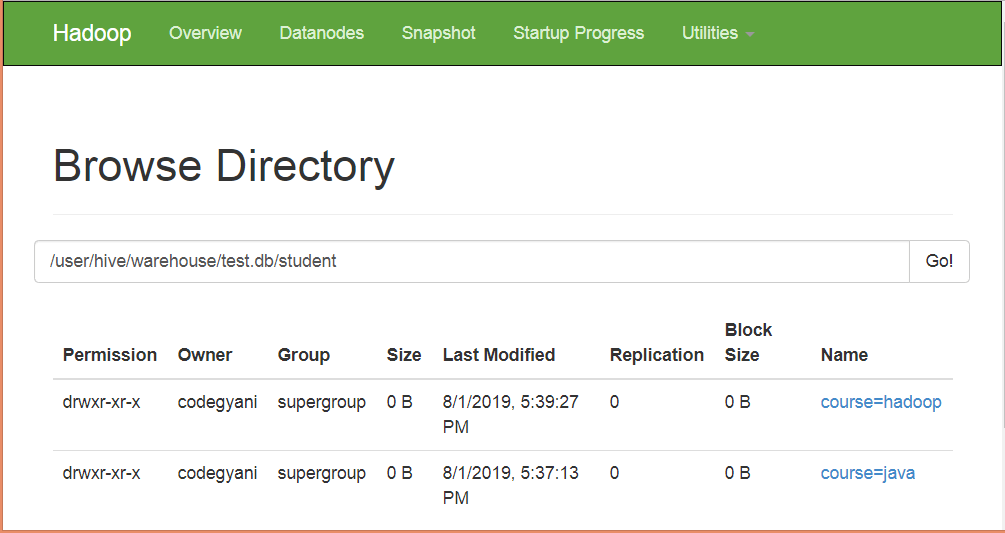
Here, we are partitioning the students of an institute based on courses.

* Load the data of another file into the same table and pass the values of partition columns with it by using the following command: -

1. hive**>** load data local inpath '/home/codegyani/hive/student\_details2' into table student
2. partition(course= "hadoop");

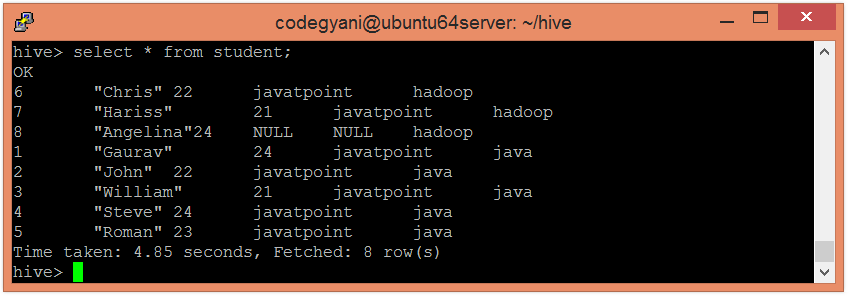


In the following screenshot, we can see that the table student is divided into two categories.



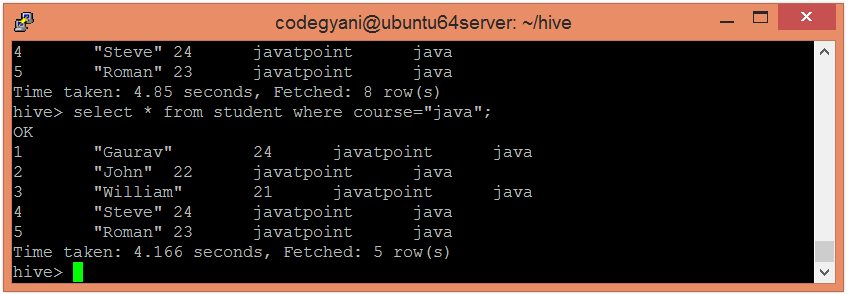
* Let's retrieve the entire data of the able by using the following command: -

1. hive**>** select \* from student;



* Now, try to retrieve the data based on partitioned columns by using the following command: -

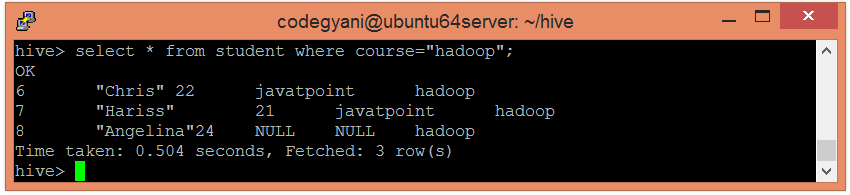
1. hive**>** select \* from student where course="java";



In this case, we are not examining the entire data. Hence, this approach improves query response time.

* Let's also retrieve the data of another partitioned dataset by using the following command: -

1. hive**>** select \* from student where course= "hadoop";

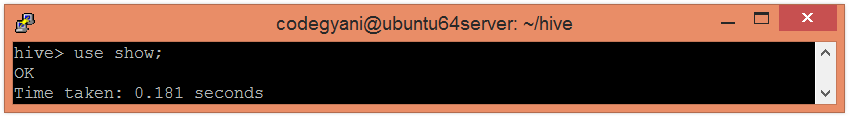


Dynamic Partitioning

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

* First, select the database in which we want to create a table.

1. hive**>** use show;

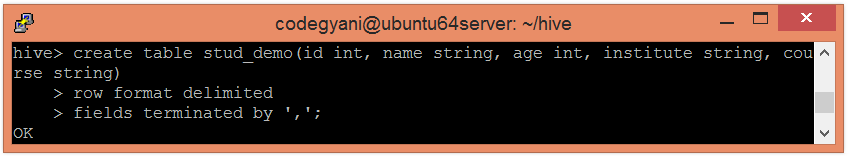


* Enable the dynamic partition by using the following commands: -

1. hive**>** set hive.exec.dynamic.partition=true;
2. hive**>** set hive.exec.dynamic.partition.mode=nonstrict;

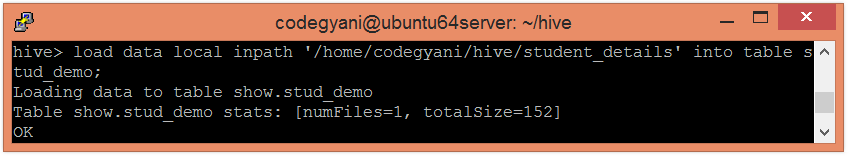
* Create a dummy table to store the data.

1. hive**>** create table stud\_demo(id int, name string, age int, institute string, course string)
2. row format delimited
3. fields terminated by ',';



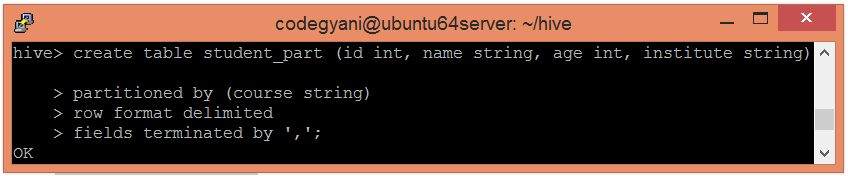
* Now, load the data into the table.

1. hive**>** load data local inpath '/home/codegyani/hive/student\_details' into table stud\_demo;



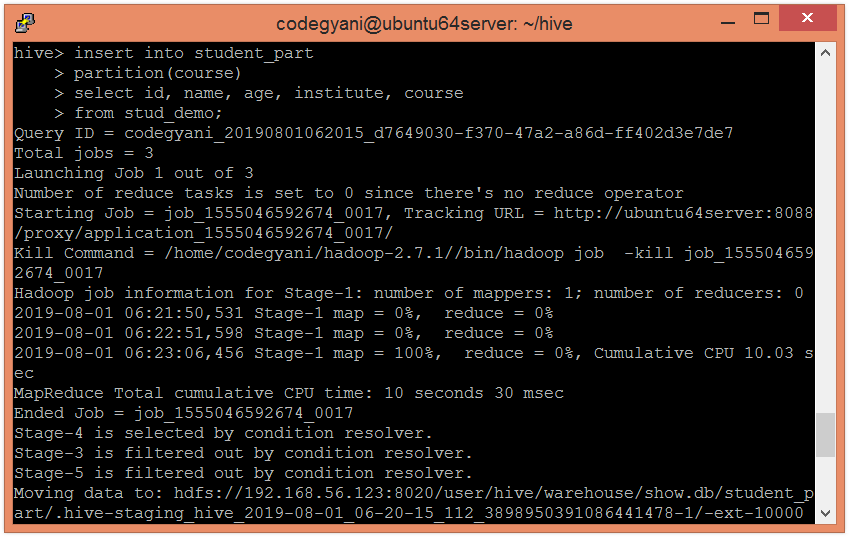
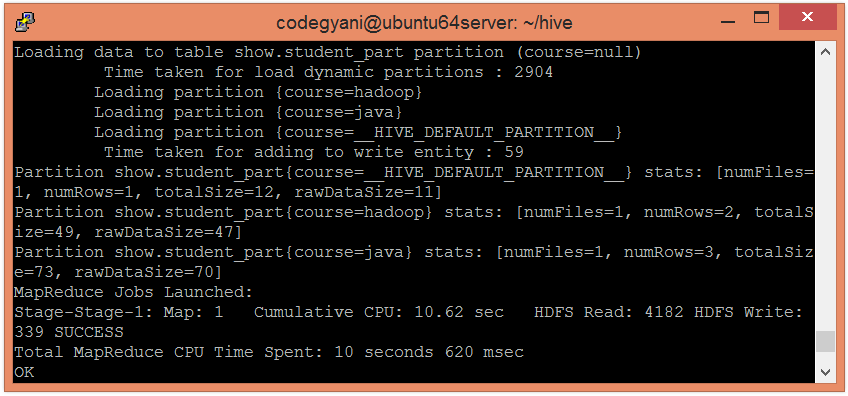
* Create a partition table by using the following command: -

1. hive**>** create table student\_part (id int, name string, age int, institute string)
2. partitioned by (course string)
3. row format delimited
4. fields terminated by ',';

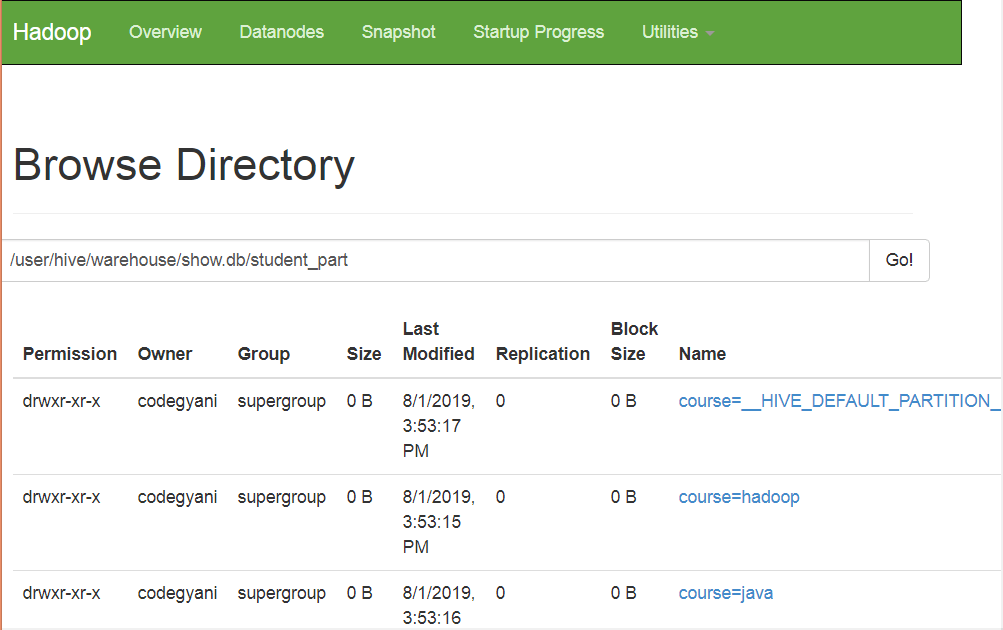


* Now, insert the data of dummy table into the partition table.

1. hive**>** insert into student\_part
2. partition(course)
3. select id, name, age, institute, course
4. from stud\_demo;

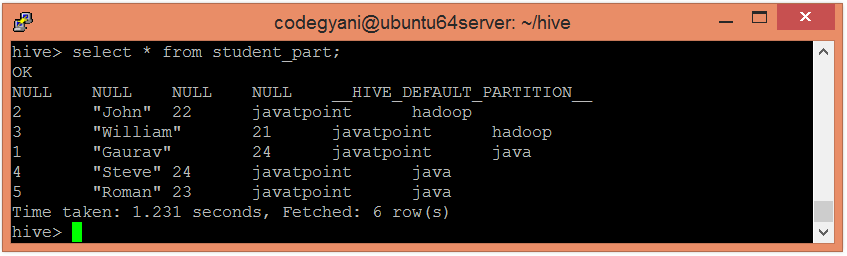
  


* In the following screenshot, we can see that the table student\_part is divided into two categories.



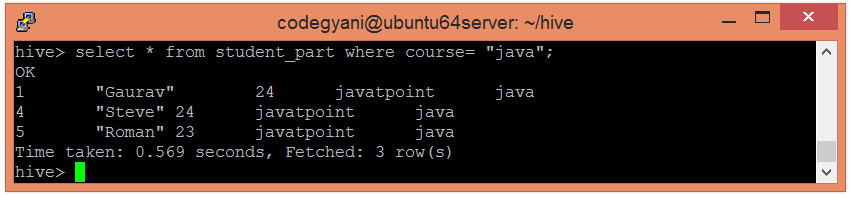
* Let's retrieve the entire data of the table by using the following command: -

1. hive**>** select \* from student\_part;



* Now, try to retrieve the data based on partitioned columns by using the following command: -

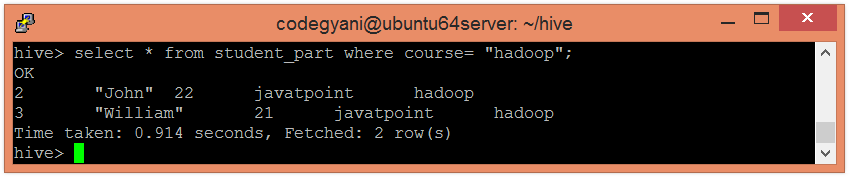
1. hive**>** select \* from student\_part where course= "java ";



In this case, we are not examining the entire data. Hence, this approach improves query response time.

* Let's also retrieve the data of another partitioned dataset by using the following command: -

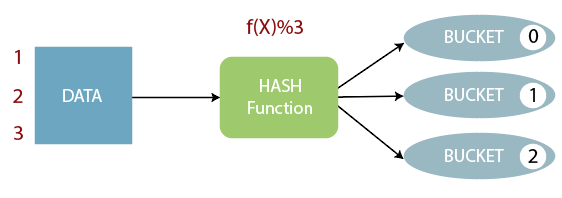
1. hive**>** select \* from student\_part where course= "hadoop";



# 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.

## Working of Bucketing in Hive

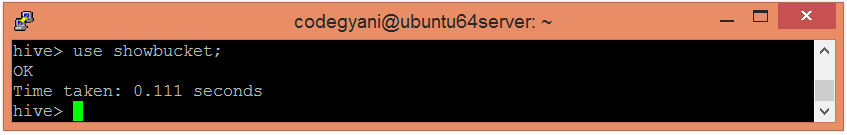


* The concept of bucketing is based on the hashing technique.
* Here, modules of current column value and the number of required buckets is calculated (let say, F(x) % 3).
* Now, based on the resulted value, the data is stored into the corresponding bucket.

### Example of Bucketing in Hive

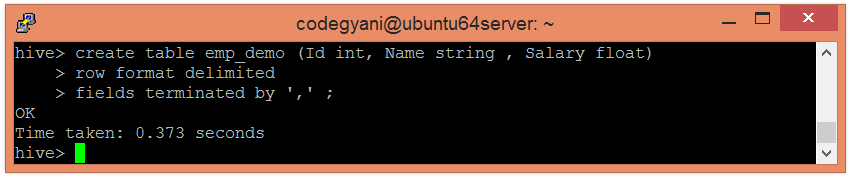
* First, select the database in which we want to create a table.

1. hive**>** use showbucket;



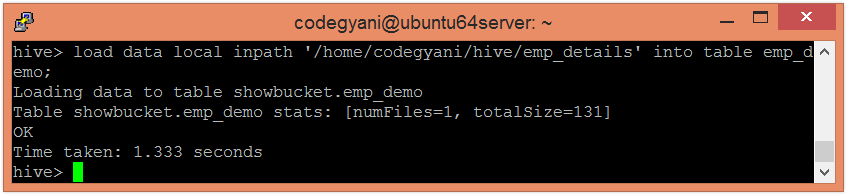
* Create a dummy table to store the data.

1. hive**>** create table emp\_demo (Id int, Name string , Salary float)
2. row format delimited
3. fields terminated by ',' ;



* Now, load the data into the table.

1. hive**>** load data local inpath '/home/codegyani/hive/emp\_details' into table emp\_demo;

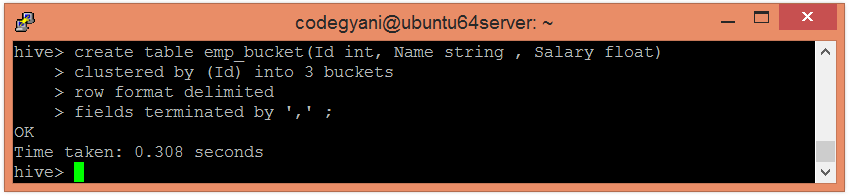


* Enable the bucketing by using the following command: -

1. hive**>** set hive.enforce.bucketing = true;

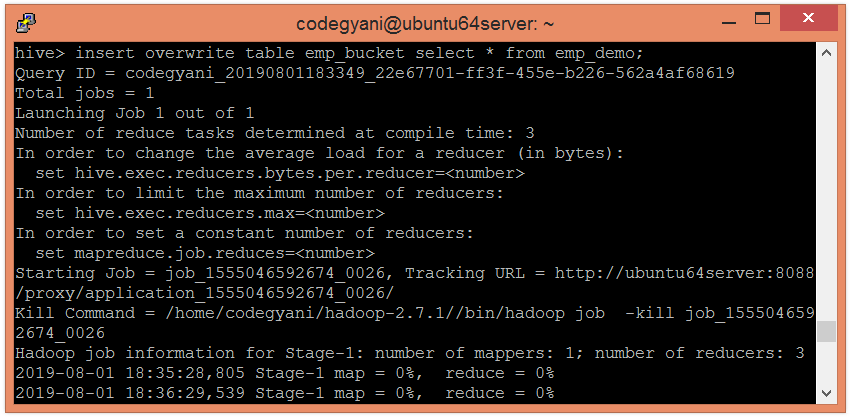
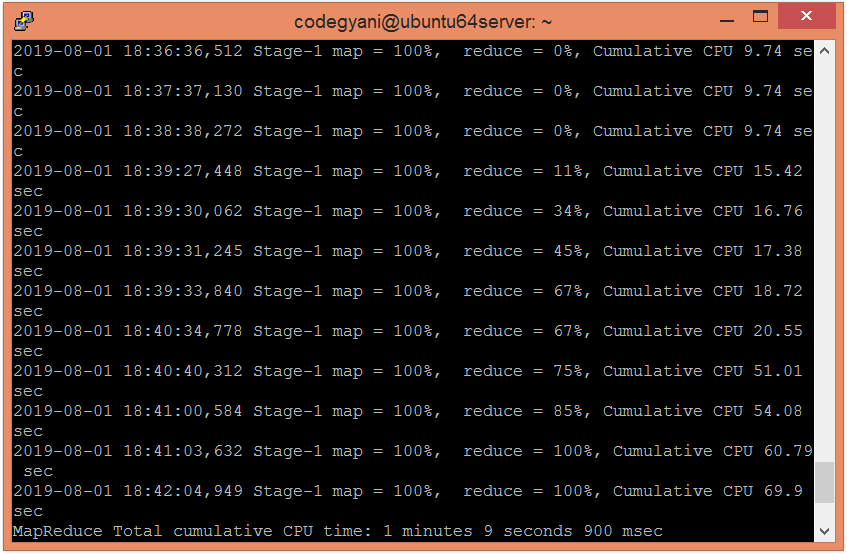
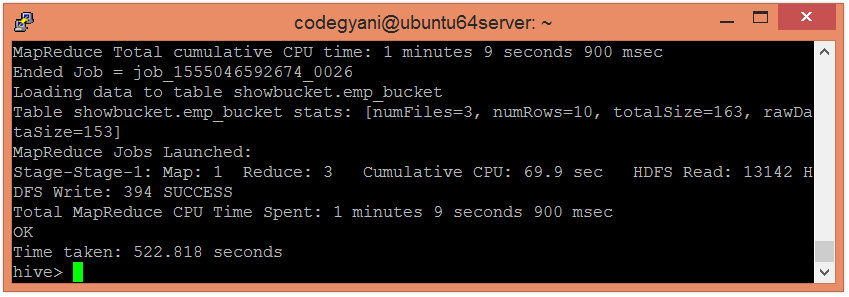
* Create a bucketing table by using the following command: -

1. hive**>** create table emp\_bucket(Id int, Name string , Salary float)
2. clustered by (Id) into 3 buckets
3. row format delimited
4. fields terminated by ',' ;

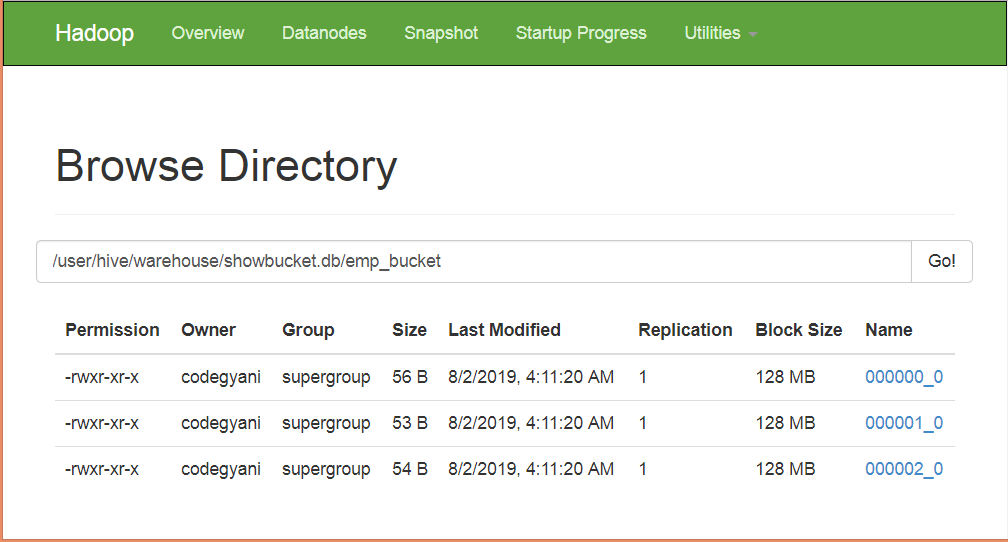


* Now, insert the data of dummy table into the bucketed table.

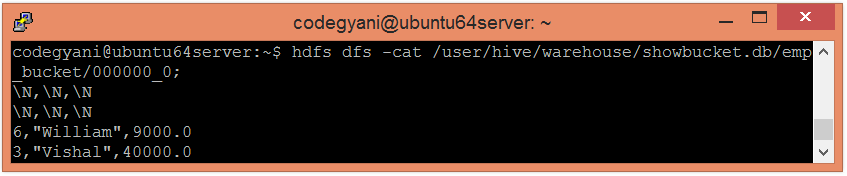
1. hive**>** insert overwrite table emp\_bucket select \* from emp\_demo;

* Here, we can see that the data is divided into three buckets.

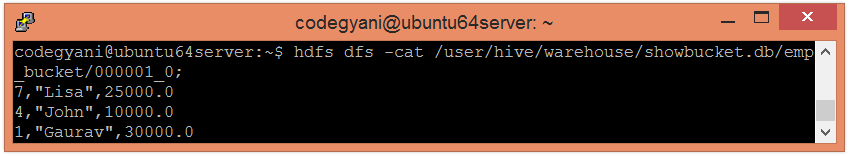


* Let's retrieve the data of bucket 0.



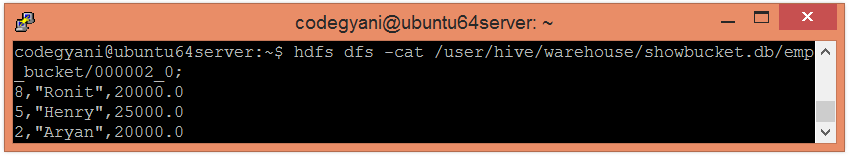
According to hash function :  
6%3=0  
3%3=0  
So, these columns stored in bucket 0.

* Let's retrieve the data of bucket 1.



According to hash function :  
7%3=1  
4%3=1  
1%3=1  
So, these columns stored in bucket 1.

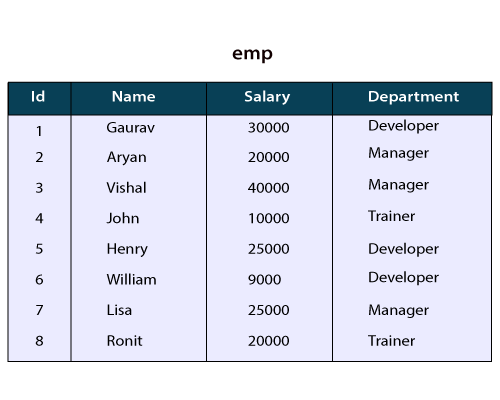
* Let's retrieve the data of bucket 2.



According   
8%3=2  
5%3=2  
2%3=2  
So, these columns stored in bucket 2.

HiveQL - GROUP BY and HAVING Clause

The Hive Query Language provides GROUP BY and HAVING clauses that facilitate similar functionalities as in SQL. Here, we are going to execute these clauses on the records of the below table:



## GROUP BY Clause

The **HQL Group By** clause is used to group the data from the multiple records based on one or more column. It is generally used in conjunction with the aggregate functions (like SUM, COUNT, MIN, MAX and AVG) to perform an aggregation over each group.

### Example of GROUP BY Clause in Hive

Let's see an example to sum the salary of employees based on department.

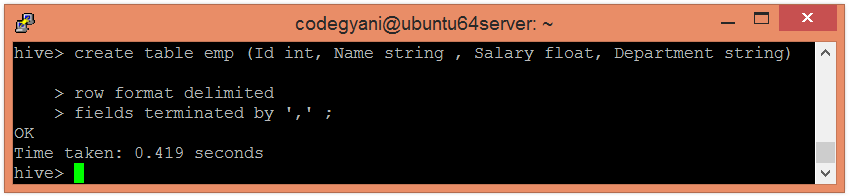
* Select the database in which we want to create a table.

1. hive**>** use hiveql;



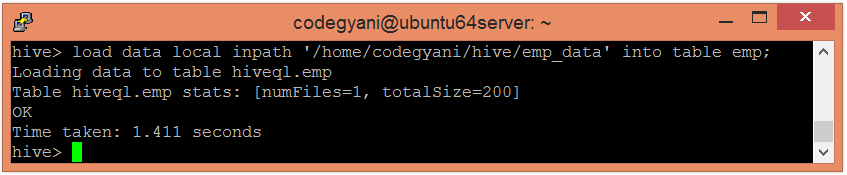
* Now, create a table by using the following command:

1. hive**>** create table emp (Id int, Name string , Salary float, Department string)
2. row format delimited
3. fields terminated by ',' ;



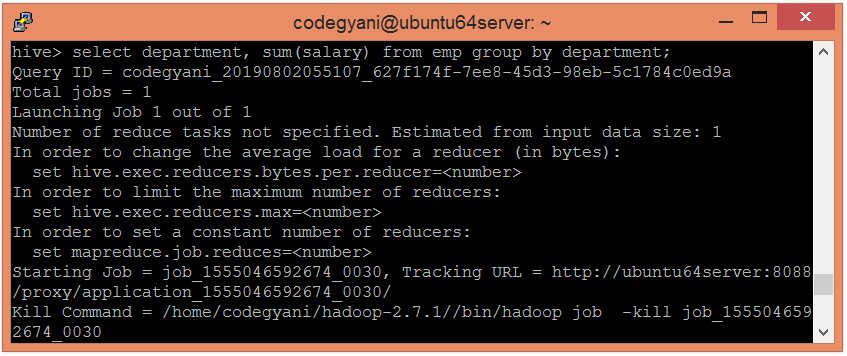
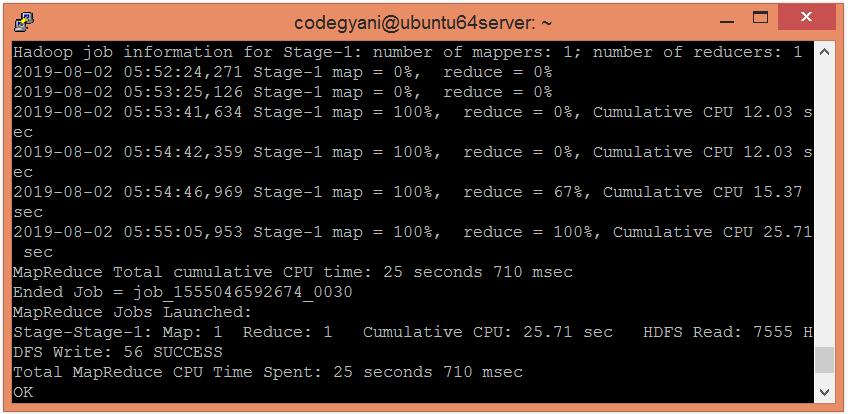
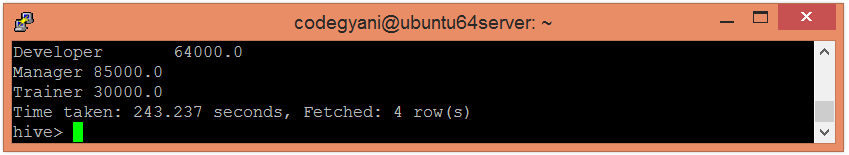
* Load the data into the table.

1. hive**>** load data local inpath '/home/codegyani/hive/emp\_data' into table emp;



* Now, fetch the sum of employee salaries department wise by using the following command:

1. hive**>** select department, sum(salary) from emp group by department;

Here, we got the desired output.

## HAVING CLAUSE

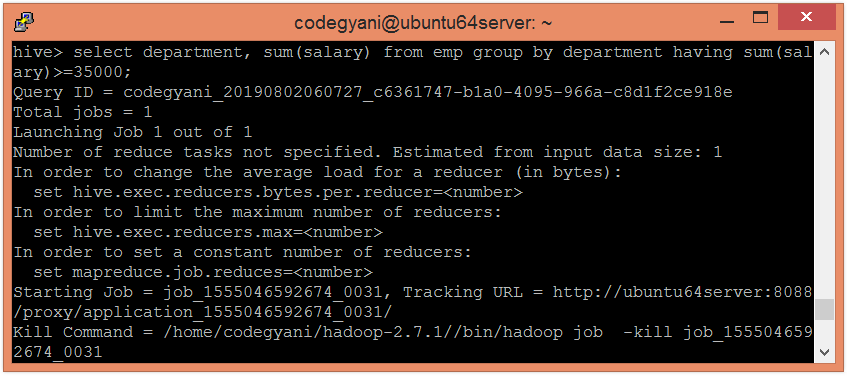
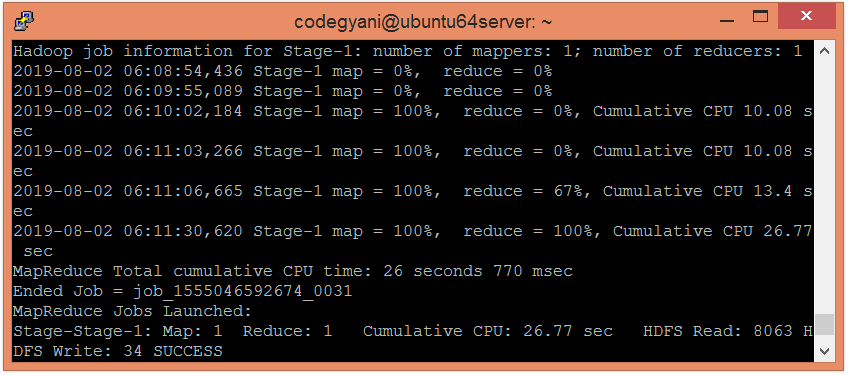
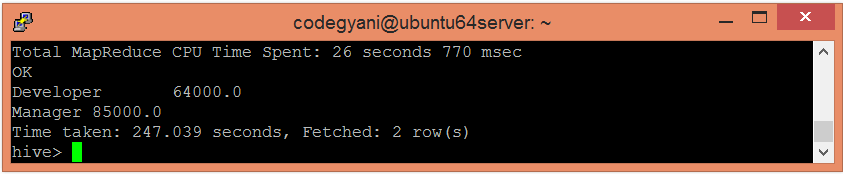
The HQL **HAVING clause** is used with **GROUP BY** clause. Its purpose is to apply constraints on the group of data produced by GROUP BY clause. Thus, it always returns the data where the condition is **TRUE**.

### Example of Having Clause in Hive

In this example, we fetch the sum of employee's salary based on department and apply the required constraints on that sum by using HAVING clause.

* Let's fetch the sum of employee's salary based on department having sum >= 35000 by using the following command:

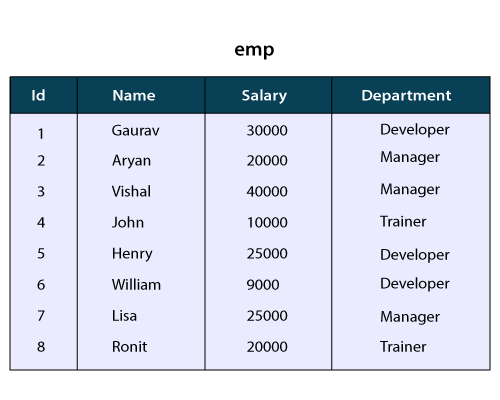
1. hive**>** select department, sum(salary) from emp group by department having sum(salary)**>**=35000;

Here, we got the desired output.

HiveQL - ORDER BY and SORT BY Clause

By using HiveQL ORDER BY and SORT BY clause, we can apply sort on the column. It returns the result set either in ascending or descending order. Here, we are going to execute these clauses on the records of the below table:



## HiveQL - ORDER BY Clause

In HiveQL, ORDER BY clause performs a complete ordering of the query result set. Hence, the complete data is passed through a single reducer. This may take much time in the execution of large datasets. However, we can use LIMIT to minimize the sorting time.

### Example of ORDER BY Clause in Hive

Let's see an example to arrange the data in the sorted order by using ORDER BY clause.

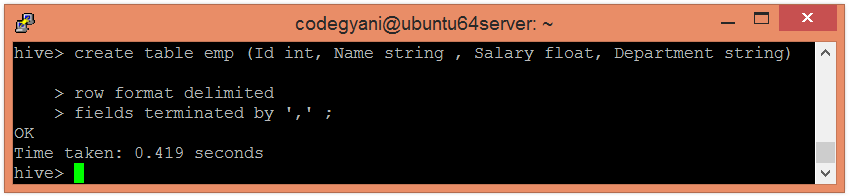
* Select the database in which we want to create a table.

1. hive**>** use hiveql;



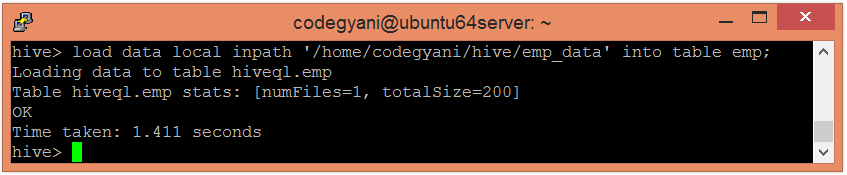
* Now, create a table by using the following command:

1. hive**>** create table emp (Id int, Name string , Salary float, Department string)
2. row format delimited
3. fields terminated by ',' ;



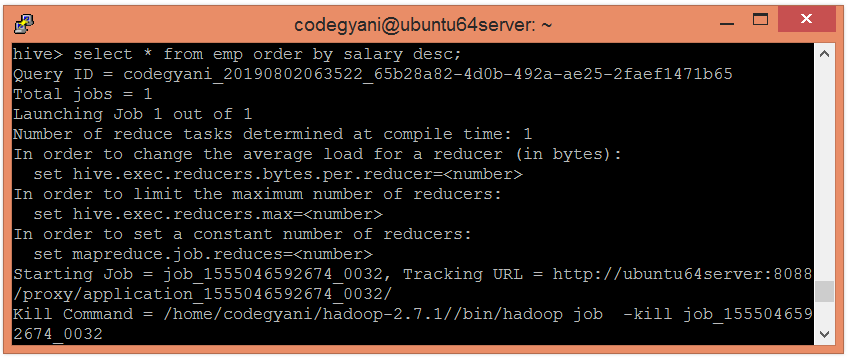
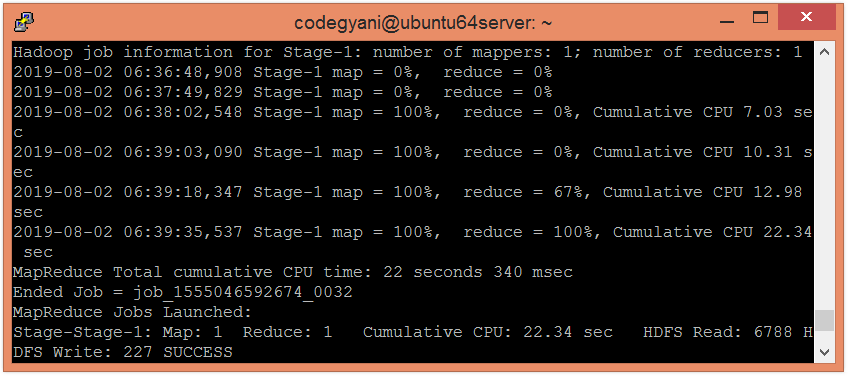
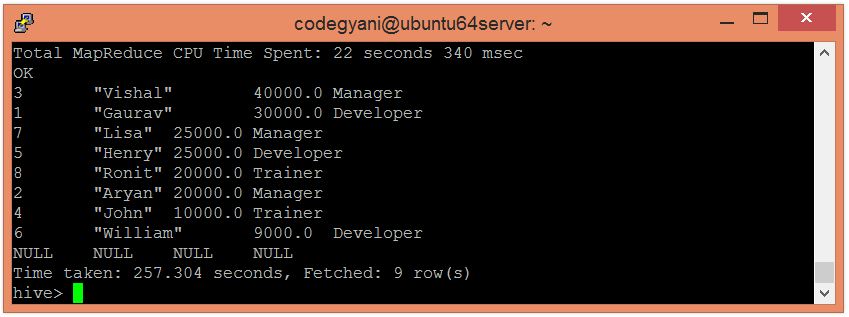
* Load the data into the table.

1. hive**>** load data local inpath '/home/codegyani/hive/emp\_data' into table emp;



* Now, fetch the data in the descending order by using the following command:

1. hive**>** select \* from emp order by salary desc;

Here, we got the desired result.

## HiveQL - SORT BY Clause

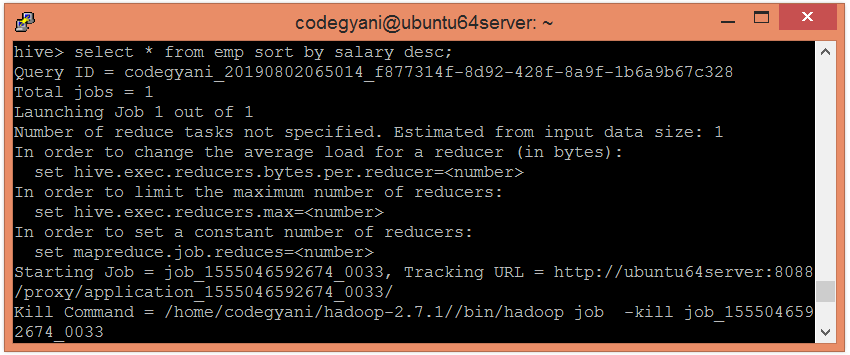
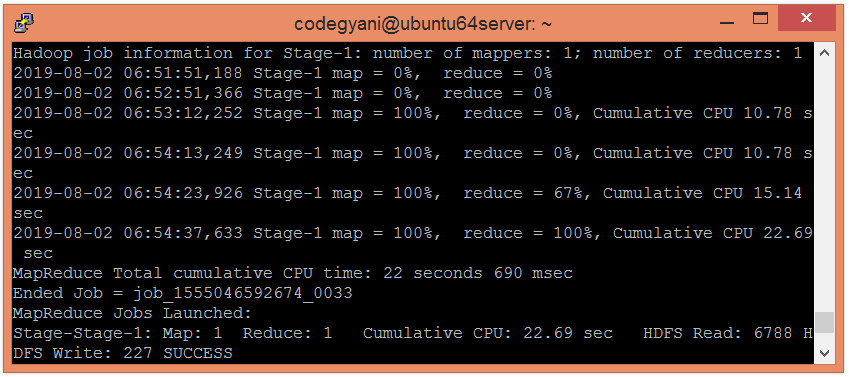
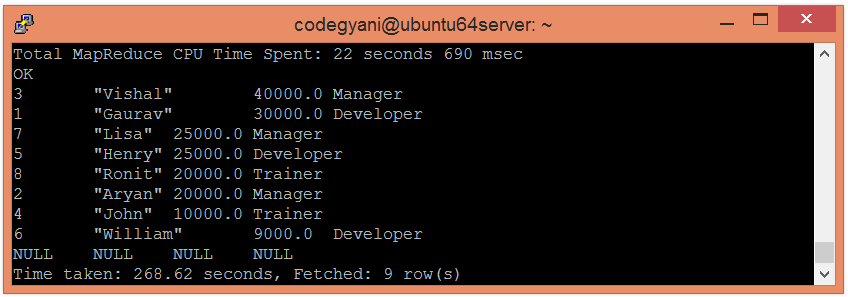
The HiveQL SORT BY clause is an alternative of ORDER BY clause. It orders the data within each reducer. Hence, it performs the local ordering, where each reducer's output is sorted separately. It may also give a partially ordered result.

### Example of SORT BY Clause in Hive

In this example, we arrange the data in the sorted order by using SORT BY clause.

* Let's fetch the data in the descending order by using the following command:

1. hive**>** select \* from emp sort by salary desc;

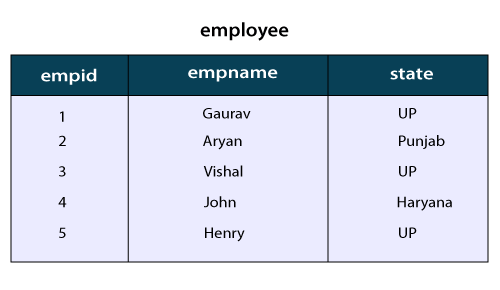
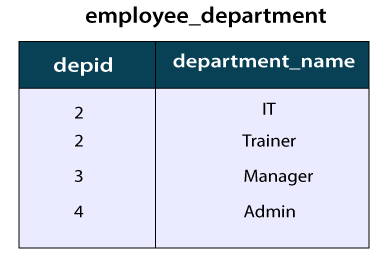
Here, we got the desired result.

# HiveQL - JOIN

The HiveQL Join clause is used to combine the data of two or more tables based on a related column between them. The various type of HiveQL joins are: -

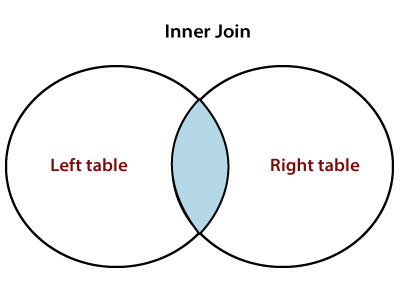
* Inner Join
* Left Outer Join
* Right Outer Join
* Full Outer Join

Here, we are going to execute the join clauses on the records of the following table:

## Inner Join in HiveQL

The HiveQL inner join is used to return the rows of multiple tables where the join condition satisfies. In other words, the join criteria find the match records in every table being joined.

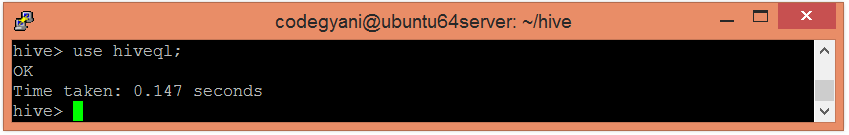


## Example of Inner Join in Hive

In this example, we take two table employee and employee\_department. The primary key (empid) of employee table represents the foreign key (depid) of employee\_department table. Let's perform the inner join operation by using the following steps: -

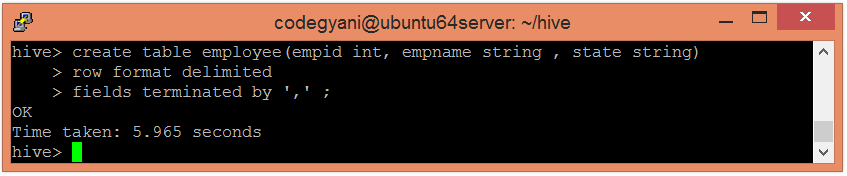
* Select the database in which we want to create a table.

1. hive**>** use hiveql;



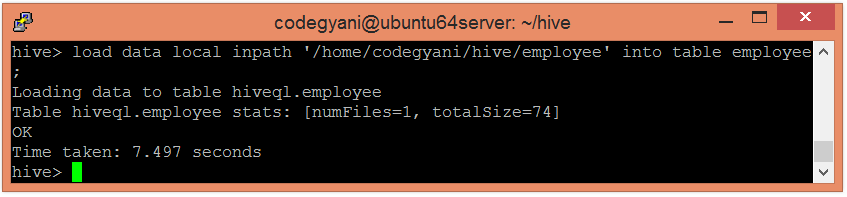
* Now, create a table by using the following command:

1. hive**>** create table employee(empid int, empname string , state string)
2. row format delimited
3. fields terminated by ',' ;



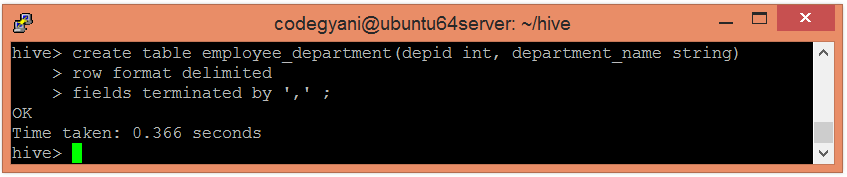
* Load the corresponding data into the table.

1. hive**>** load data local inpath '/home/codegyani/hive/employee' into table employee;



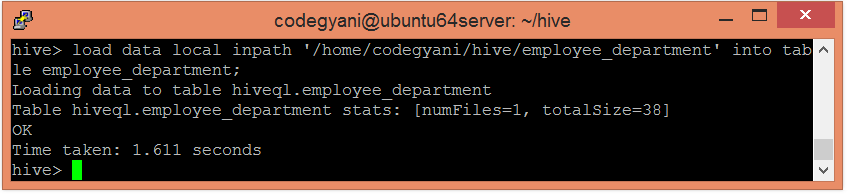
* Now, create another table by using the following command:

1. hive**>** create table employee\_department(depid int, department\_name string)
2. row format delimited
3. fields terminated by ',' ;



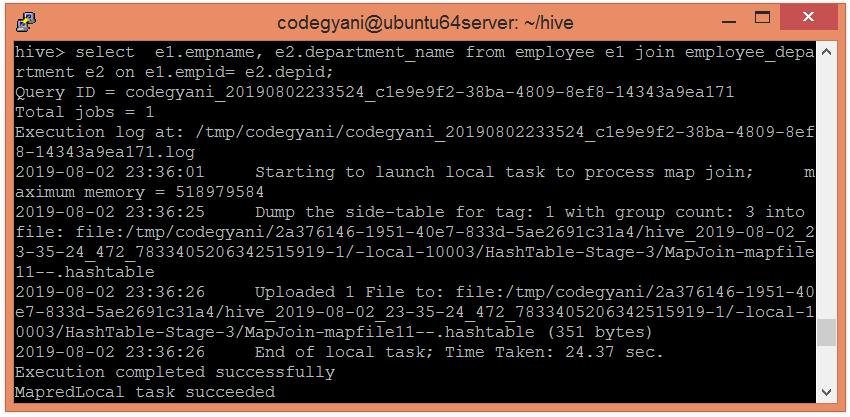
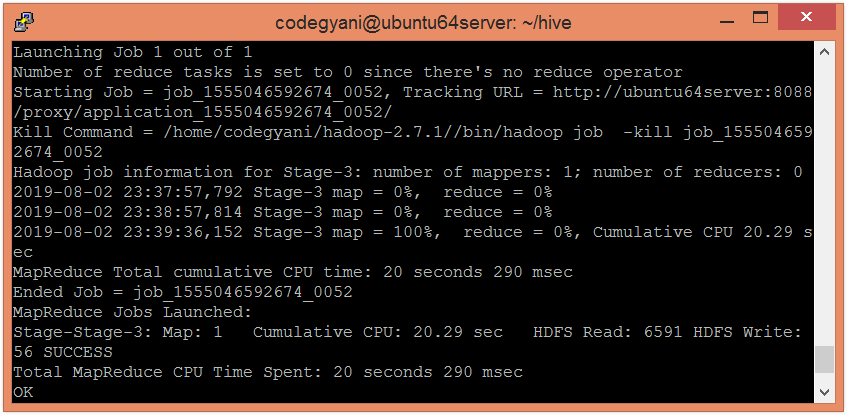
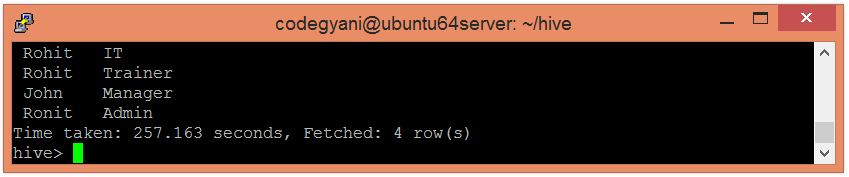
* Load the corresponding data into the table.

1. hive**>** load data local inpath '/home/codegyani/hive/employee\_department' into table employee\_department;



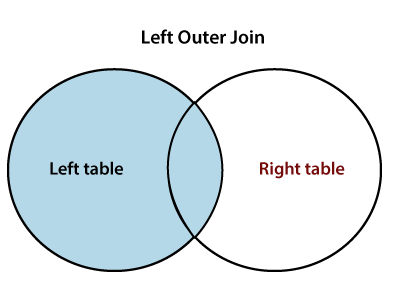
* Now, perform the inner join operation by using the following command: -

1. hive**>**select  e1.empname, e2.department\_name from employee e1 join employee\_department e2 on e1.empid= e2.depid;

## Left Outer Join in HiveQL

The HiveQL left outer join returns all the records from the left (first) table and only that records from the right (second) table where join criteria find the match.

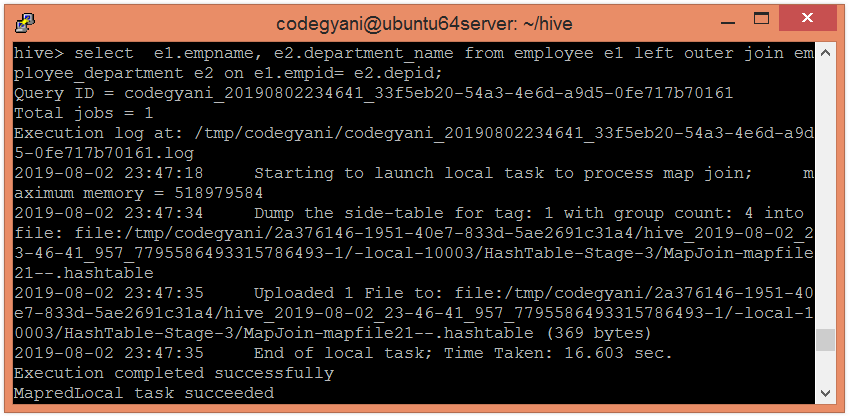
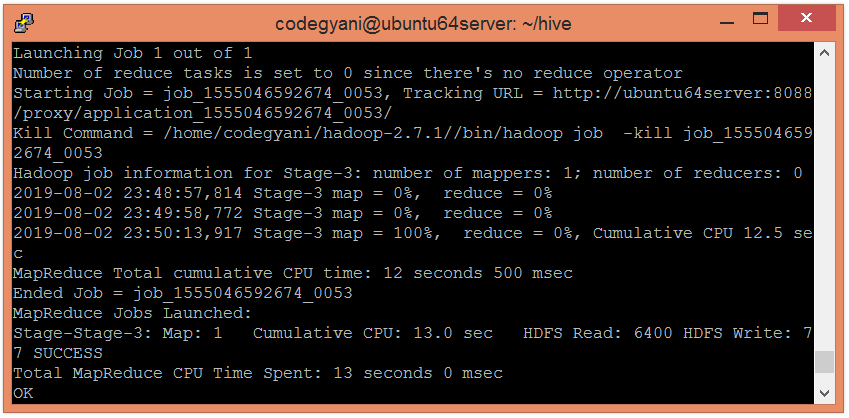
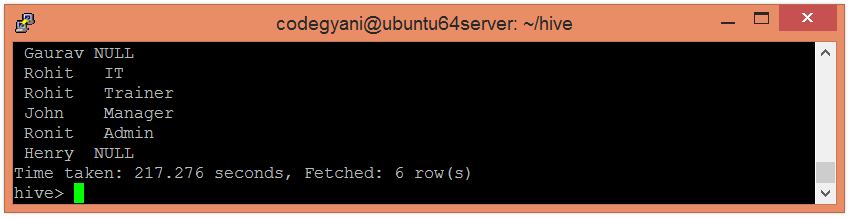


### Example of Left Outer Join in Hive

In this example, we perform the left outer join operation.

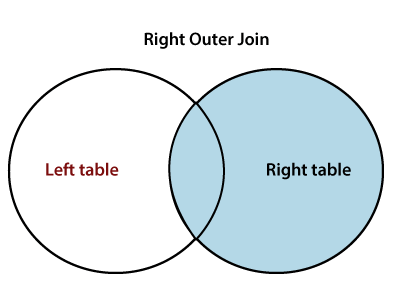
* Let's us execute the left outer join operation by using the following command: -

1. hive**>** select  e1.empname, e2.department\_name from employee e1 left outer join employee\_department e2 on e1.empid= e2.depid;

## Right Outer Join in HiveQL

The HiveQL right outer join returns all the records from the right (second) table and only that records from the left (first) table where join criteria find the match.

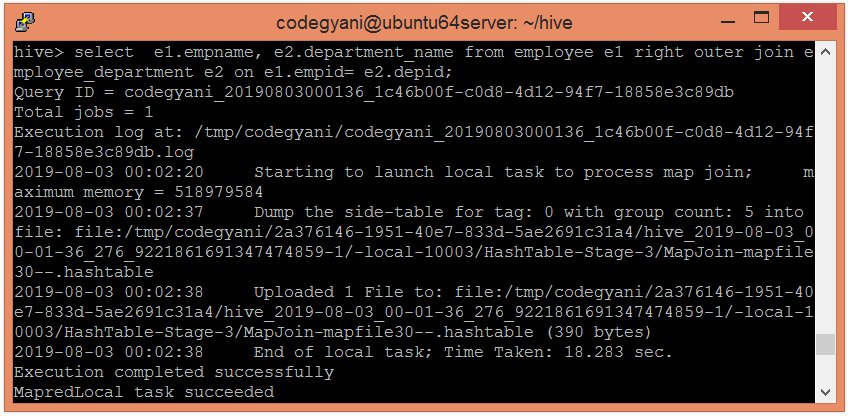
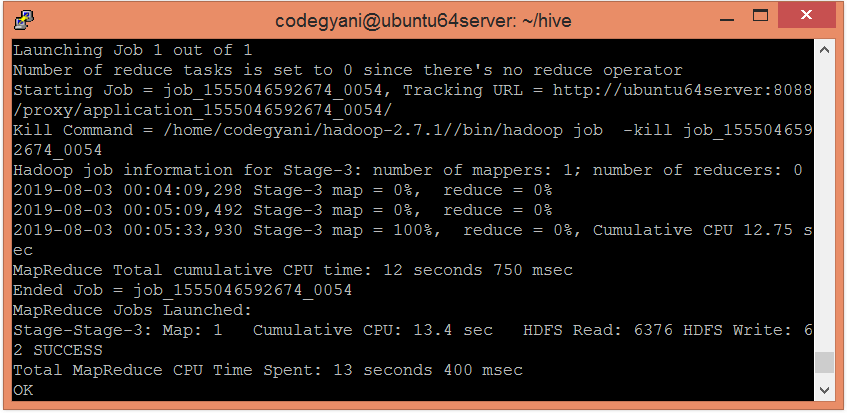
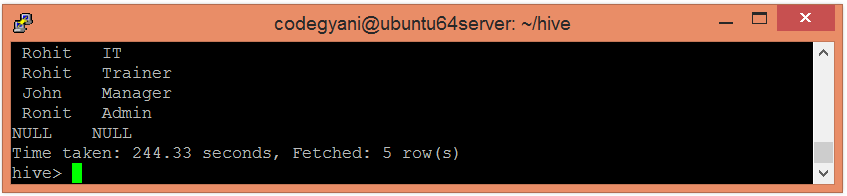


### Example of Left Outer Join in Hive

In this example, we perform the left outer join operation.

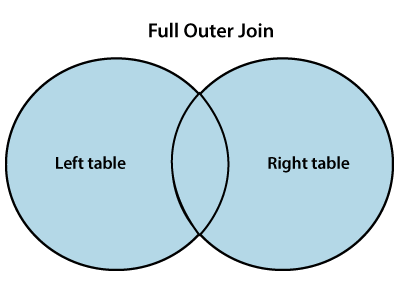
* Let's us execute the left outer join operation by using the following command: -

1. hive**>** select  e1.empname, e2.department\_name from employee e1 right outer join employee\_department e2 on e1.empid= e2.depid;

## Full Outer Join

The HiveQL full outer join returns all the records from both the tables. It assigns Null for missing records in either table.

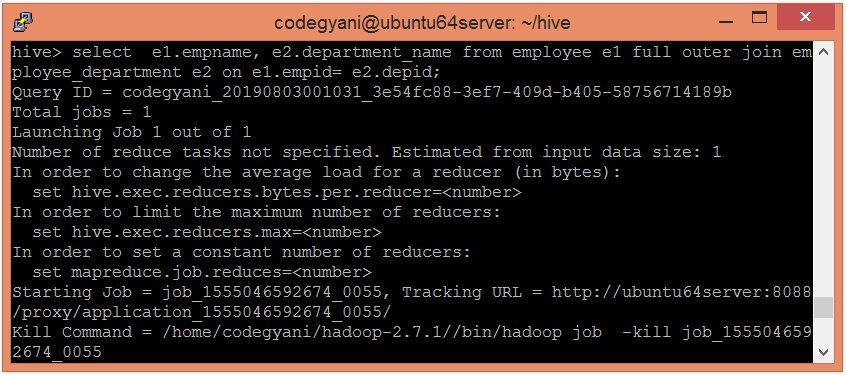
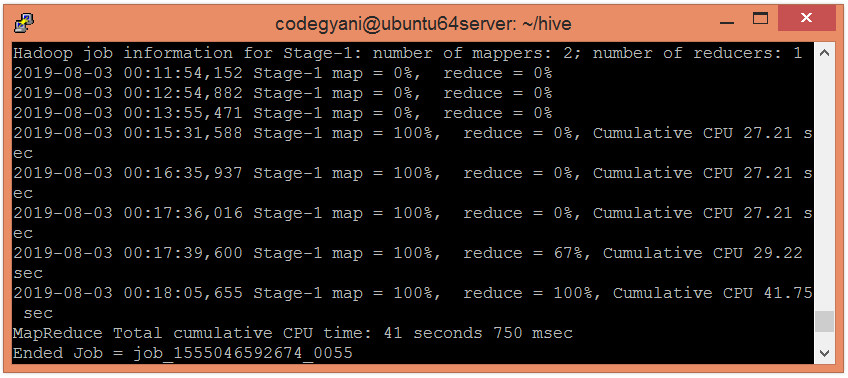
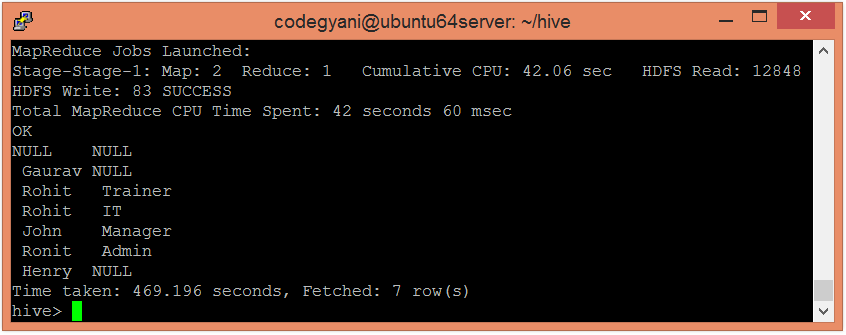


### Example of Full Outer Join in Hive

In this example, we perform the full outer join operation.

* Let's us execute the full outer join operation by using the following command: -

1. select  e1.empname, e2.department\_name from employee e1 full outer join employee\_department e2 on e1.empid= e2.depid;

**VIEWS**

describes how to create and manage views. Views are generated based on user requirements. You can save any result set data as a view. The usage of view in Hive is same as that of the view in SQL. It is a standard RDBMS concept. We can execute all DML operations on a view.

## Creating a View

You can create a view at the time of executing a SELECT statement. The syntax is as follows:

CREATE VIEW [IF NOT EXISTS] view\_name [(column\_name [COMMENT column\_comment], ...) ]

[COMMENT table\_comment]

AS SELECT ...

## Example

Let us take an example for view. Assume employee table as given below, with the fields Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000. We store the result in a view named **emp\_30000.**

+------+--------------+-------------+-------------------+--------+

| ID | Name | Salary | Designation | Dept |

+------+--------------+-------------+-------------------+--------+

|1201 | Gopal | 45000 | Technical manager | TP |

|1202 | Manisha | 45000 | Proofreader | PR |

|1203 | Masthanvali | 40000 | Technical writer | TP |

|1204 | Krian | 40000 | Hr Admin | HR |

|1205 | Kranthi | 30000 | Op Admin | Admin |

+------+--------------+-------------+-------------------+--------+

The following query retrieves the employee details using the above scenario:

hive> CREATE VIEW emp\_30000 AS

SELECT \* FROM employee

WHERE salary>30000;

## Dropping a View

Use the following syntax to drop a view:

DROP VIEW view\_name

The following query drops a view named as emp\_30000:

hive> DROP VIEW emp\_30000;

## Creating an Index

An Index is nothing but a pointer on a particular column of a table. Creating an index means creating a pointer on a particular column of a table. Its syntax is as follows:

CREATE INDEX index\_name

ON TABLE base\_table\_name (col\_name, ...)

AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

[IDXPROPERTIES (property\_name=property\_value, ...)]

[IN TABLE index\_table\_name]

[PARTITIONED BY (col\_name, ...)]

[

[ ROW FORMAT ...] STORED AS ...

| STORED BY ...

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

## Example

Let us take an example for index. Use the same employee table that we have used earlier with the fields Id, Name, Salary, Designation, and Dept. Create an index named index\_salary on the salary column of the employee table.

The following query creates an index:

hive> CREATE INDEX inedx\_salary ON TABLE employee(salary)

AS 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler';

It is a pointer to the salary column. If the column is modified, the changes are stored using an index value.

## Dropping an Index

The following syntax is used to drop an index:

DROP INDEX <index\_name> ON <table\_name>

The following query drops an index named index\_salary:

hive> DROP INDEX index\_salary ON employee;

**SQOOP**

**Introduction:**

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. This is a brief tutorial that explains how to make use of Sqoop in Hadoop ecosystem.

When Big Data storages and analyzers such as MapReduce, Hive, HBase, Cassandra, Pig, etc. of the Hadoop ecosystem came into picture, they required a tool to interact with the relational database servers for importing and exporting the Big Data residing in them. Here, Sqoop occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop’s HDFS.

**Sqoop** − “SQL to Hadoop and Hadoop to SQL”

Sqoop is a tool designed to transfer data between Hadoop and relational database servers. It is used to import data from relational databases such as MySQL, Oracle to Hadoop HDFS, and export from Hadoop file system to relational databases. It is provided by the Apache Software Foundation.

**Installation**

As Sqoop is a sub-project of Hadoop, it can only work on Linux operating system. Follow the steps given below to install Sqoop on your system.

## Step 1: Verifying JAVA Installation

You need to have Java installed on your system before installing Sqoop. Let us verify Java installation using the following command −

$ java –version

If Java is already installed on your system, you get to see the following response −

java version "1.7.0\_71"

Java(TM) SE Runtime Environment (build 1.7.0\_71-b13)

Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)

If Java is not installed on your system, then follow the steps given below.

## Installing Java

Follow the simple steps given below to install Java on your system.

### Step 1

Download Java (JDK <latest version> - X64.tar.gz) by visiting the following [link](http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-).

Then jdk-7u71-linux-x64.tar.gz will be downloaded onto your system.

### Step 2

Generally, you can find the downloaded Java file in the Downloads folder. Verify it and extract the jdk-7u71-linux-x64.gz file using the following commands.

$ cd Downloads/

$ ls

jdk-7u71-linux-x64.gz

$ tar zxf jdk-7u71-linux-x64.gz

$ ls

jdk1.7.0\_71 jdk-7u71-linux-x64.gz

### Step 3

To make Java available to all the users, you have to move it to the location “/usr/local/”. Open root, and type the following commands.

$ su

password:

# mv jdk1.7.0\_71 /usr/local/java

# exitStep IV:

### Step 4

For setting up PATH and JAVA\_HOME variables, add the following commands to ~/.bashrc file.

export JAVA\_HOME=/usr/local/java

export PATH=$PATH:$JAVA\_HOME/bin

Now apply all the changes into the current running system.

$ source ~/.bashrc

### Step 5

Use the following commands to configure Java alternatives −

# alternatives --install /usr/bin/java java usr/local/java/bin/java 2

# alternatives --install /usr/bin/javac javac usr/local/java/bin/javac 2

# alternatives --install /usr/bin/jar jar usr/local/java/bin/jar 2

# alternatives --set java usr/local/java/bin/java

# alternatives --set javac usr/local/java/bin/javac

# alternatives --set jar usr/local/java/bin/jar

Now verify the installation using the command **java -version** from the terminal as explained above.

## Step 2: Verifying Hadoop Installation

Hadoop must be installed on your system before installing Sqoop. Let us verify the Hadoop installation using the following command −

$ hadoop version

If Hadoop is already installed on your system, then you will get the following response −

Hadoop 2.4.1

--

Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768

Compiled by hortonmu on 2013-10-07T06:28Z

Compiled with protoc 2.5.0

From source with checksum 79e53ce7994d1628b240f09af91e1af4

If Hadoop is not installed on your system, then proceed with the following steps −

## Downloading Hadoop

Download and extract Hadoop 2.4.1 from Apache Software Foundation using the following commands.

$ su

password:

# cd /usr/local

# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/

hadoop-2.4.1.tar.gz

# tar xzf hadoop-2.4.1.tar.gz

# mv hadoop-2.4.1/\* to hadoop/

# exit

## Installing Hadoop in Pseudo Distributed Mode

Follow the steps given below to install Hadoop 2.4.1 in pseudo-distributed mode.

### Step 1: Setting up Hadoop

You can set Hadoop environment variables by appending the following commands to ~/.bashrc file.

export HADOOP\_HOME=/usr/local/hadoop

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export YARN\_HOME=$HADOOP\_HOME

export HADOOP\_COMMON\_LIB\_NATIVE\_DIR=$HADOOP\_HOME/lib/native

export PATH=$PATH:$HADOOP\_HOME/sbin:$HADOOP\_HOME/bin

Now, apply all the changes into the current running system.

$ source ~/.bashrc

### Step 2: Hadoop Configuration

You can find all the Hadoop configuration files in the location “$HADOOP\_HOME/etc/hadoop”. You need to make suitable changes in those configuration files according to your Hadoop infrastructure.

$ cd $HADOOP\_HOME/etc/hadoop

In order to develop Hadoop programs using java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing JAVA\_HOME value with the location of java in your system.

export JAVA\_HOME=/usr/local/java

Given below is the list of files that you need to edit to configure Hadoop.

**core-site.xml**

The core-site.xml file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and the size of Read/Write buffers.

Open the core-site.xml and add the following properties in between the <configuration> and </configuration> tags.

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:9000 </value>

</property>

</configuration>

**hdfs-site.xml**

The hdfs-site.xml file contains information such as the value of replication data, namenode path, and datanode path of your local file systems. It means the place where you want to store the Hadoop infrastructure.

Let us assume the following data.

dfs.replication (data replication value) = 1

(In the following path /hadoop/ is the user name.

hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)

namenode path = //home/hadoop/hadoopinfra/hdfs/namenode

(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)

datanode path = //home/hadoop/hadoopinfra/hdfs/datanode

Open this file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>

</property>

<property>

<name>dfs.data.dir</name>

<value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>

</property>

</configuration>

**Note** − In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

**yarn-site.xml**

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>

**mapred-site.xml**

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, you need to copy the file from mapred-site.xml.template to mapred-site.xml file using the following command.

$ cp mapred-site.xml.template mapred-site.xml

Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

## Verifying Hadoop Installation

The following steps are used to verify the Hadoop installation.

### Step 1: Name Node Setup

Set up the namenode using the command “hdfs namenode -format” as follows.

$ cd ~

$ hdfs namenode -format

The expected result is as follows.

10/24/14 21:30:55 INFO namenode.NameNode: STARTUP\_MSG:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

STARTUP\_MSG: Starting NameNode

STARTUP\_MSG: host = localhost/192.168.1.11

STARTUP\_MSG: args = [-format]

STARTUP\_MSG: version = 2.4.1

...

...

10/24/14 21:30:56 INFO common.Storage: Storage directory

/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.

10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to

retain 1 images with txid >= 0

10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0

10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN\_MSG:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SHUTDOWN\_MSG: Shutting down NameNode at localhost/192.168.1.11

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### Step 2: Verifying Hadoop dfs

The following command is used to start dfs. Executing this command will start your Hadoop file system.

$ start-dfs.sh

The expected output is as follows −

10/24/14 21:37:56

Starting namenodes on [localhost]

localhost: starting namenode, logging to /home/hadoop/hadoop-

2.4.1/logs/hadoop-hadoop-namenode-localhost.out

localhost: starting datanode, logging to /home/hadoop/hadoop-

2.4.1/logs/hadoop-hadoop-datanode-localhost.out

Starting secondary namenodes [0.0.0.0]

### Step 3: Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

$ start-yarn.sh

The expected output is as follows −

starting yarn daemons

starting resourcemanager, logging to /home/hadoop/hadoop-

2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out

localhost: starting node manager, logging to /home/hadoop/hadoop-

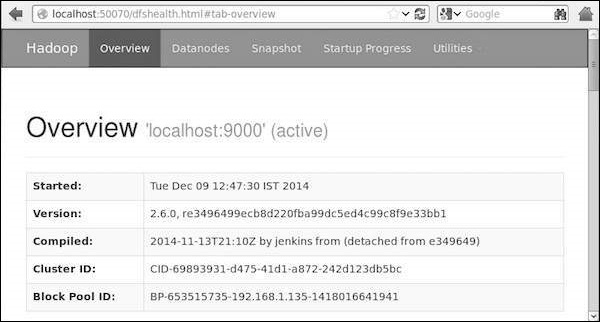
2.4.1/logs/yarn-hadoop-nodemanager-localhost.out

### Step 4: Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following URL to get Hadoop services on your browser.

http://localhost:50070/

The following image depicts a Hadoop browser.

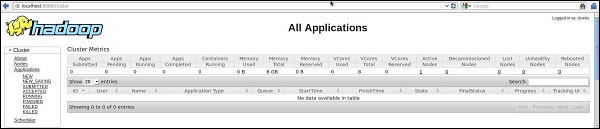


### Step 5: Verify All Applications for Cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

http://localhost:8088/

The following image depicts the Hadoop cluster browser.



## Step 3: Downloading Sqoop

We can download the latest version of Sqoop from the following [link](https://sqoop.apache.org/) For this tutorial, we are using version 1.4.5, that is, **sqoop-1.4.5.bin\_\_hadoop-2.0.4-alpha.tar.gz**.

## Step 4: Installing Sqoop

The following commands are used to extract the Sqoop tar ball and move it to “/usr/lib/sqoop” directory.

$tar -xvf sqoop-1.4.4.bin\_\_hadoop-2.0.4-alpha.tar.gz

$ su

password:

# mv sqoop-1.4.4.bin\_\_hadoop-2.0.4-alpha /usr/lib/sqoop

#exit

## Step 5: Configuring bashrc

You have to set up the Sqoop environment by appending the following lines to ~/**.bashrc** file −

#Sqoop

export SQOOP\_HOME=/usr/lib/sqoop export PATH=$PATH:$SQOOP\_HOME/bin

The following command is used to execute ~/**.bashrc** file.

$ source ~/.bashrc

## Step 6: Configuring Sqoop

To configure Sqoop with Hadoop, you need to edit the **sqoop-env.sh** file, which is placed in the **$SQOOP\_HOME/conf** directory. First of all, Redirect to Sqoop config directory and copy the template file using the following command −

$ cd $SQOOP\_HOME/conf

$ mv sqoop-env-template.sh sqoop-env.sh

Open **sqoop-env.sh** and edit the following lines −

export HADOOP\_COMMON\_HOME=/usr/local/hadoop

export HADOOP\_MAPRED\_HOME=/usr/local/hadoop

## Step 7: Download and Configure mysql-connector-java

We can download **mysql-connector-java-5.1.30.tar.gz** file from the following [link](http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/).

The following commands are used to extract mysql-connector-java tarball and move **mysql-connector-java-5.1.30-bin.jar** to /usr/lib/sqoop/lib directory.

$ tar -zxf mysql-connector-java-5.1.30.tar.gz

$ su

password:

# cd mysql-connector-java-5.1.30

# mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib

## Step 8: Verifying Sqoop

The following command is used to verify the Sqoop version.

$ cd $SQOOP\_HOME/bin

$ sqoop-version

Expected output −

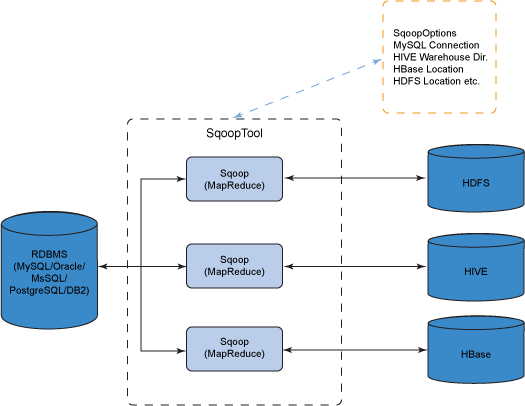
14/12/17 14:52:32 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5

Sqoop 1.4.5 git commit id 5b34accaca7de251fc91161733f906af2eddbe83

Compiled by abe on Fri Aug 1 11:19:26 PDT 2014

Sqoop installation is complete.

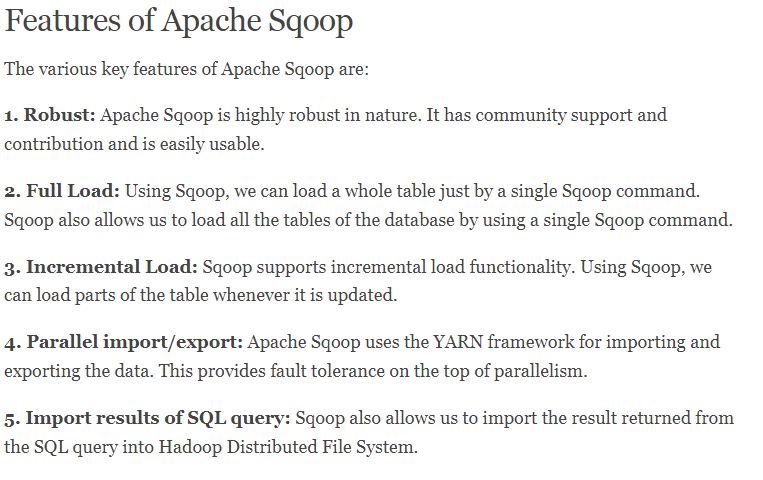
## Sqoop Working

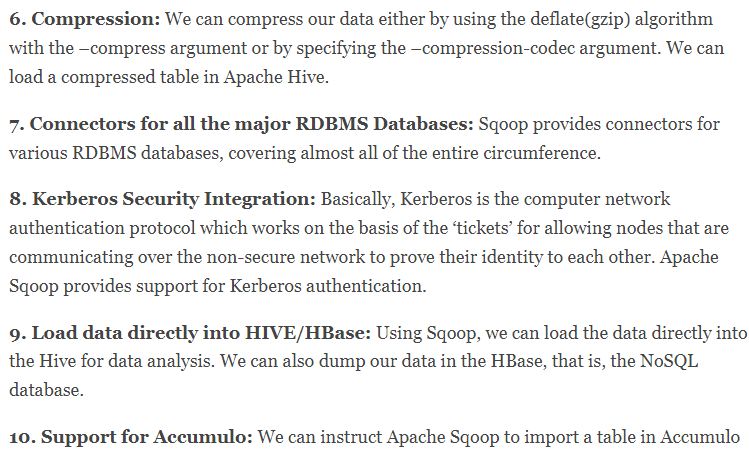


**Step 1:** Sqoop send the request to Relational DB to send the return the metadata informationabout the table(Metadata here is the data about the table in relational DB).

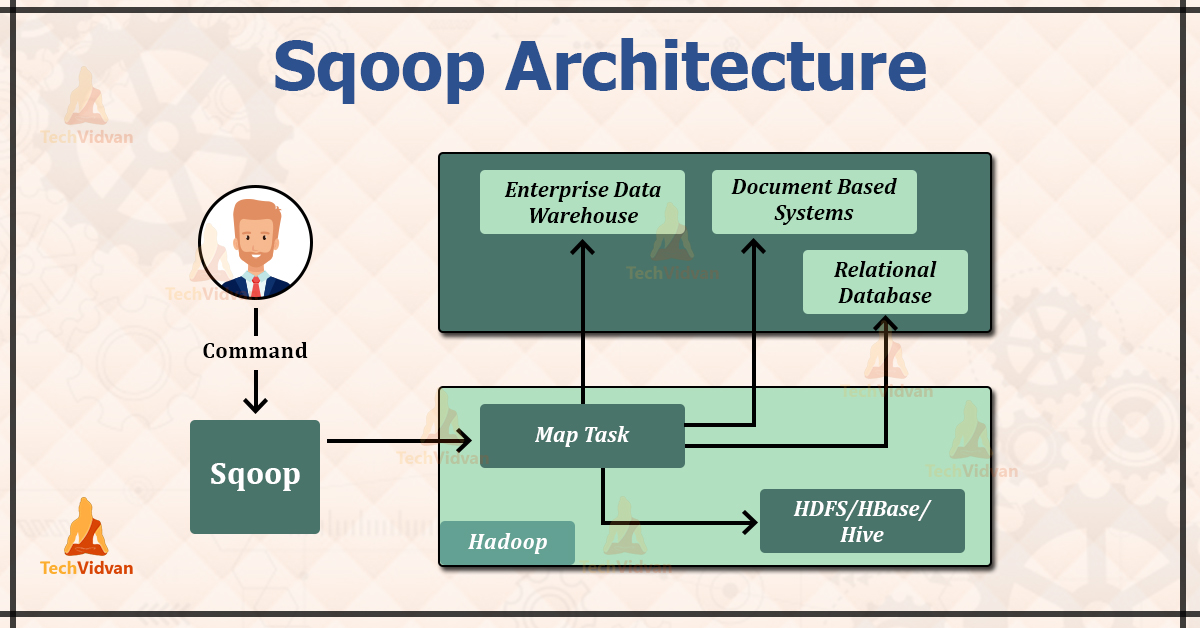
**Step 2:** From the received information it will generate the java classes (Reason why you shouldhave Java configured before get it working-Sqoop internally uses JDBC API to generate data).

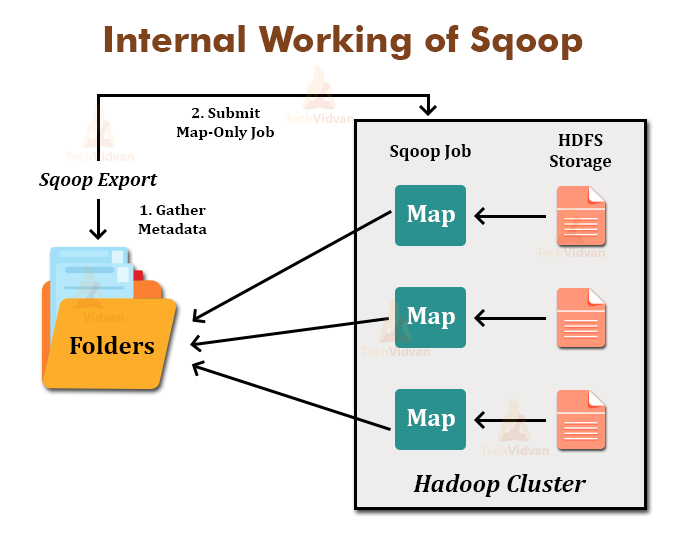
**Step 3:** Now Sqoop (As its written in java ?tries to package the compiled classes to beable togenerate table structure) , post compiling creates jar file(Java packaging standard).

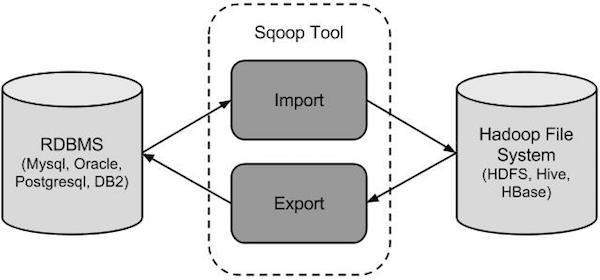
****

****

**SQOOP ARCHITECTURE:**

****

****

****

**Sqoop Working**

* **Sqoop Import**

The import tool imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in text files or as binary data in Avro and Sequence files.

* **Sqoop Export**

The export tool exports a set of files from HDFS back to an RDBMS. The files given as input to Sqoop contain records, which are called as rows in table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

**Generate Code**

* In order to run the object-oriented application, every table in the database must have one DAO class that contains the ‘getter’ and the ‘setter’ methods for objects initialization. In Sqoop, the Codegen tool automatically generates the DAO class.
* This tool generates the DAO class in Java, based on the Table Schema structure.
* The Java definition of the record is instantiated as a part of the import process.
* This tool is mainly used for checking whether the Java lost the Java code. If java lost the java code, then this tool will create the new version of Java with default delimiter between the fields.

**Syntax for Sqoop Codegen**

The Syntax for Sqoop Codegen is:

* $ sqoop codegen (generic-args) (codegen-args)
* $ sqoop-codegen (generic-args) (codegen-args)

**Sqoop Import**

We all know that for transferring data from RDBMS to HDFS or vice-versa, we use Apache Sqoop. In this Sqoop Import article, we will discuss the Sqoop Import tool used for importing tables from the RDBMS to the HDFS.

In this article, you will explore how to import tables to HDFS, Hive, HBase, and Accumulo. You will also learn the syntax as well as the different arguments.

Moreover, you will study the purpose of Sqoop Import as well as examples of the Sqoop import query to understand it well.

Let us first explore what Sqoop Import is.

#### Sqoop Import Syntax

he syntax for Sqoop Import command is:

$ sqoop import (generic-args) (import-args)

$ sqoop-import (generic-args) (import-args)

# Sqoop Export Files from HDFS to RDBMS

Similar to Sqoop Import, there is another tool named Sqoop Export in Sqoop, which is used for exporting a set of files from the HDFS back to the RDBMS. In this Sqoop Export article, you will explore the entire concept of Sqoop Export.

The article will explain what Sqoop Export is, modes in which Sqoop Export works, its syntax, arguments, and many more. The article also covers the difference between the insert mode and update mode.

Moreover, we will learn the Sqoop Export syntax with an example invocation to understand it better.

Let us first start with an introduction to Sqoop Export.

### What is Sqoop Export?

The Sqoop export tool is used for exporting a set of files from the Hadoop Distributed File System back to the RDBMS. For performing export, the target table must exist on the target database.

The files given as an input to Apache Sqoop contain the records, which are called as rows in the table. These files are read and parsed into the set of records and delimited with the user-specified delimiter.  
The export command works in two modes- insert mode and update mode.

**1. Insert mode:** It is the default mode. In this mode, the records from the input files are inserted into the database table by using the INSERT statement.  
**2. Update mode:** In the update mode, Sqoop generates an UPDATE statement that replaces existing records into the database.

#### Syntax for Sqoop Export

The Syntax for Sqoop Export are:

$ sqoop export (generic-args) (export-args)

$ sqoop-export (generic-args) (export-args)

The Hadoop generic arguments should be passed before any export arguments, and we can enter export arguments in any order with respect to each other.

**The common arguments are:**

|  |  |
| --- | --- |
| **Argument** | **Description** |
| –connect <jdbc-uri> | It specifies the JDBC connect string |
| –connection-manager <class-name> | It specifies the connection manager class to be used |
| –driver <class-name> | Manually specify JDBC driver class to use |
| –hadoop-home <dir> | Override $HADOOP\_HOME |
| –help | Print usage instructions |
| -P | Read the password from console |
| –password <password> | Set authentication password |
| –username <username> | Set authentication username |
| –verbose | Print more information while working |
| –connection-param-file <filename> | Optional properties file that provides connection parameters |

**The Export control arguments are:**

|  |  |
| --- | --- |
| **Argument** | **Description** |
| –direct | Use direct export fast path |
| –export-dir <dir> | It specifies the HDFS source path for export |
| -m,–num-mappers <n> | Use *n* map tasks to export in parallel |
| –table <table-name> | Table to populate |
| –update-key <col-name> | It specifies the anchor column to be used for updates. If there are more than one column, then we use the comma-separated list of columns. |
| –update-mode <mode> | It will specify how the updates were performed when the new rows were found with the non-matching keys in a database. The updateonly and the allowinsert are the legal values for mode. |
| –input-null-string <null-string> | It specify the string which is to be interpreted as null for string columns |
| –input-null-non-string <null-string> | It specify the string which is to be interpreted as null for non-string columns |