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.

**PARTITION:**

1) Create a hive table

2)load the data in the hive table

3)create partition table

4)set hive.exec.dynamic.partition.mode=nonstrict;

5)load data into partition table from hive table.

**EXAMPLES:**

**Hive> create table all\_students(sno int, sname string, passout\_year string)**

**>row format delimited fields terminated by ‘,’;**

OK

Time taken: 0.245 seconds

**Hive> load data local inpath ‘/home/cloudera/training/hivedata/students.csv’ into table all\_students;**

Loading data to table default.all\_students

OK

Time taken: 0.423 seconds

**Hive> select \* from all\_students;**

OK

**Hive> create table students\_part(sno int, sname string) PARTITIONED BY (passout\_year string);**

Ok

Time taken:0.425 seconds

**Hive>describe students\_part;**

**Hive> set hive.exec.dynamic.partition.mode=nonstrict;**

**Hive> insert OVERWRITE TABLE students\_part PARTITION(passout\_year ) select sno,sname, passout\_year from all\_students;**

OK

Time taken: 44.25 seconds

**CREATE A NON-PARTITION TABLE:**

Create table user\_data\_no\_partition(sno int,user\_name string,city string) ROW FORMAT delimited fields terminated by ‘,’

LINES TERMINATED BY ‘\n’ STORED AS TEXTFILE;

Load data local inpath ‘/home/big/user\_data.txt’ into table user\_data\_no\_partition;

Select \* from user\_data\_no\_partition;

**PRACTICAL:**

**Hive> create table user\_data\_no\_partition**

**>(sno int,user\_name string,city string)**

**>ROW FORMAT delimited fields terminated by ‘,’ LINES TERMINATED BY ‘\n’ STORED AS TEXTFILE;**

OK

TIME taken: 0.214 seconds

**Hive> load data local inpath ‘/home/big/user\_data.txt’ into table user\_data\_no\_partition;**

Loading data to table default.user\_data\_no\_partition

OK

Time taken: 0.918 seconds

**Hive>select \* from user\_data\_no\_partition;**

1, Gowtham, Chennai

2,Saravana,chennai

3, ram, delhi

4,alex, mumbai

5,Rahul ,delhi

6, arun ,goa

7, nila, chennai

8, nandini, chennai

9, anitha ,delhi

10, jaya, delhi

\*\* For partitioned table,we cannot insert data by LOAD data command.

\*\* We can only insert a data into portioned table with only INSERT INTO (or) INSERT OVERWRITE.

\*\* By default, Hive will store all the tables in HDFS under

/user/hive/warehouse/ path only.

The partitioning in Hive can be executed in two ways -

1)Static partitioning

2)Dynamic Partitioning

## **Static Partitioning**

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

**STATIC PARTITION:**

Example:1

Create table user\_data

(sno int

User\_name string)

Partitioned by(city string);

Insert into table user\_data partition(city=’chennai’) select sno,user\_name from user\_data\_no\_partition where city=’chennai’;

Select \* from user\_data;

**PRACTICAL**

**Hive> Create table user\_data**

**>(sno intUser\_name string)**

**>Partitioned by(city string);**

OK

Time taken:0.259 seconds

**Hive> Insert into table user\_data partition(city=’chennai’) select sno,user\_name from user\_data\_no\_partition where city=’chennai’;**

OK

Time taken:37.473 seconds

**Hive> select \* from user\_data;**

OK

1 gowtham chennai

2 saravana chennai

7 nila chennai

8 nandini chennai

Time taken: 0.591 seconds,fetched: 4 row(s)

**Hive> show partitions user\_data;**

OK

City=Chennai

Time taken: 0.24 seconds, fetched: 1 row(s)

\*\* Insert into table user\_data partition(city=’chennai’) select sno,user\_name from user\_data\_no\_partition where city=’chennai’;

\*\* We have to mention value in static partition . i.e.. city=’chennai’ (or) city=’delhi’

\*\* We cannot include 3rd column i.e.. partitioned column(city) in the select statement.

WHERE and INSERT command must be same.

i.e.. (city=’chennai’),where city=’chennai;

.. (city=’Delhi’),where city=’Delhi;

If both are not same,we can get errors.

**Example of Static Partitioning**

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

hive**>** use test;

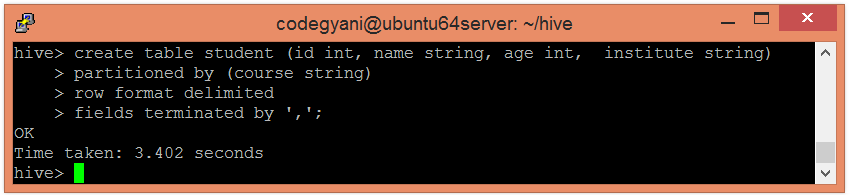
* Create the table and provide the partitioned columns by using the following command: -

hive**>** create table student (id int, name string, age int,  institute string)

partitioned by (course string)

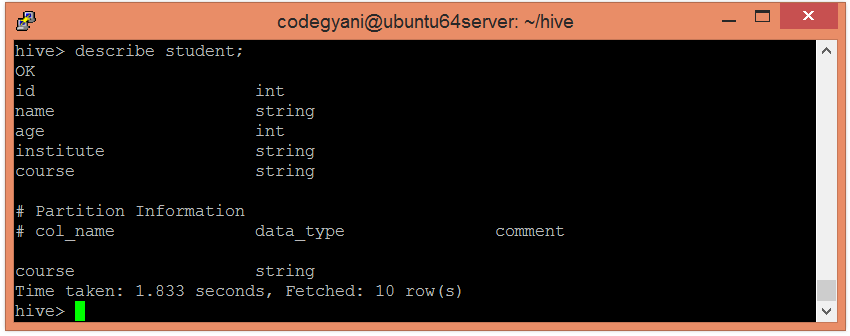
row format delimited

fields terminated by ',';



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

hive**>** describe student;

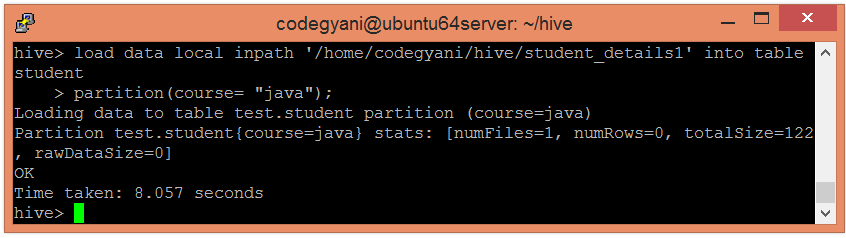


* Load the data into the table and pass the values of partition columns with it by using the following command: -

hive**>** load data local inpath '/home/codegyani/hive/student\_details1' into table

student

partition(course= "java");

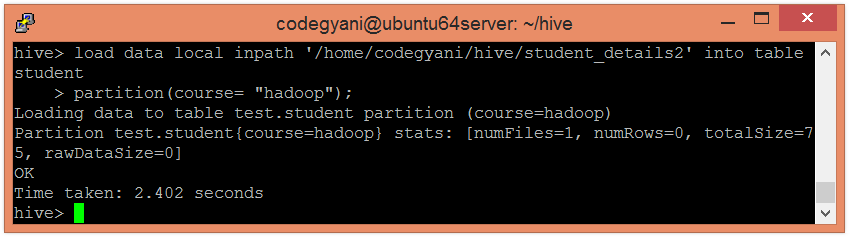


Here, we are partitioning the students of an institute based on courses.

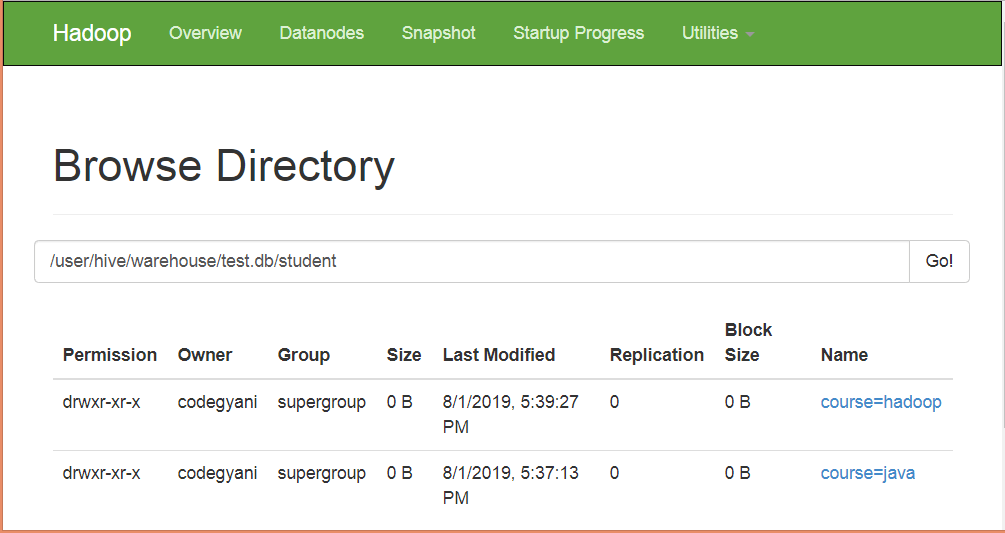
* Load the data of another file into the same table and pass the values of partition columns with it by using the following command: -

hive**>** load data local inpath '/home/codegyani/hive/student\_details2' into table student

partition(course= "hadoop");

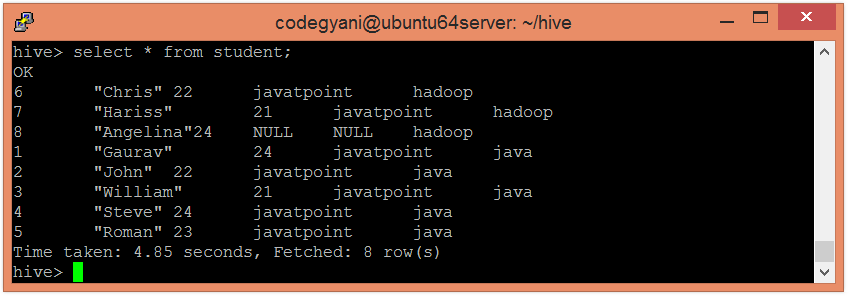


In the following screenshot, we can see that the table student is divided into two categories.

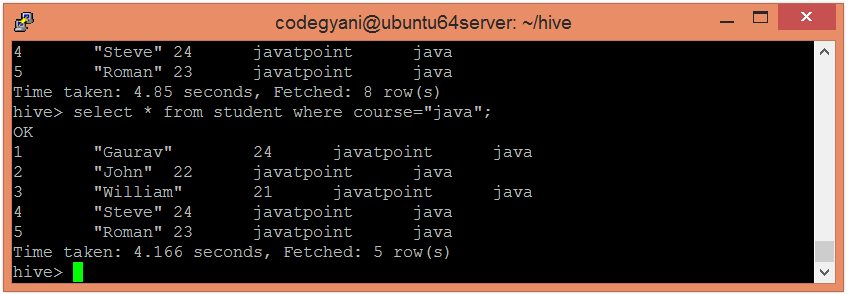


* Let's retrieve the entire data of the able by using the following command: -

hive**>** select \* from student;

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

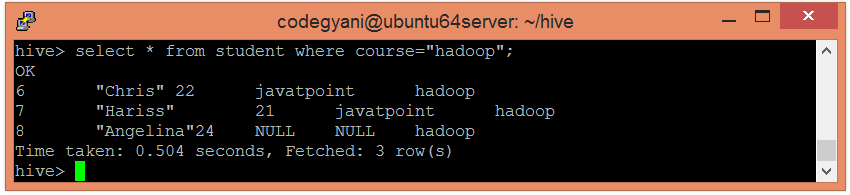
hive**>** select \* from student where course="java";



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: -

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.

**Example1:**

**DYNAMIC PARTITION:**

Create table user\_data\_dynamic

(sno int, user\_name string)

Partitioned by (city string);

Set hive.exec.dynamic.partition.mode=nonstrict;

Insert into table user\_data\_dynamic partition(city) select sno,user\_name,city from user\_data\_no\_partition where city=’chennai’;

Select \* from user\_data\_dynamic;

**PRACTICAL:**

**Hive> Create table user\_data\_dynamic**

**>(sno int, user\_name string)**

**>Partitioned by (city string);**

**Hive> Insert into table user\_data\_dynamic partition(city) select sno,user\_name,city from user\_data\_no\_partition where city=’chennai’**;

OK

Time taken: 43.56 seconds

**Hive> Set hive.exec.dynamic.partition.mode=nonstrict;**

**Hive>select \* from user\_data\_dynamic;**

OK

1 gowtham chennai

2 saravana chennai

7 nila chennai

8 nandini Chennai

Time taken: 0.4 seconds

\*\* Insert into table user\_data\_dynamic partition(city) select sno,user\_name,city from user\_data\_no\_partition where city=’chennai’;

\*\* We have mention only column name in insert statement,but column name is not necessary.

\*\* We can 3rd column i.e.. partitioned column , because we are not giving the value for the column city.

\*\* We can change WHERE condition.

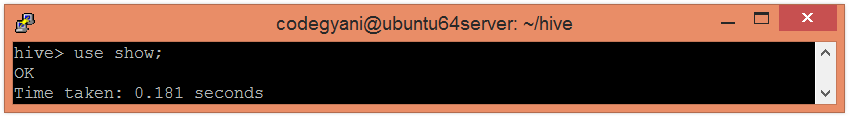
* + By chamging WHERE condition, we cannot get errors like static partition.

\*\* In real time mostly we can use dynamic partition.

**EXAMPLE2:**

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

hive**>** use show;



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

hive**>** set hive.exec.dynamic.partition=true;

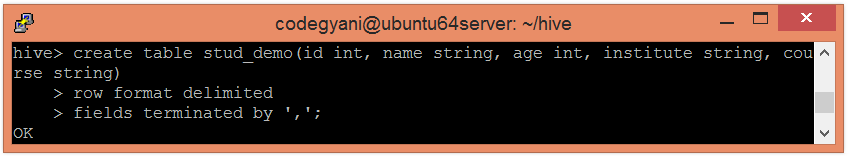
hive**>** set hive.exec.dynamic.partition.mode=nonstrict;

* Create a dummy table to store the data.

hive**>** create table stud\_demo(id int, name string, age int, institute string, course string)

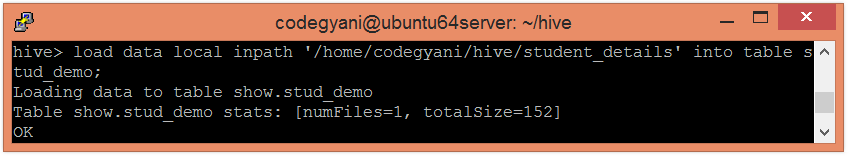
row format delimited

fields terminated by ',';



* Now, load the data into the table.

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



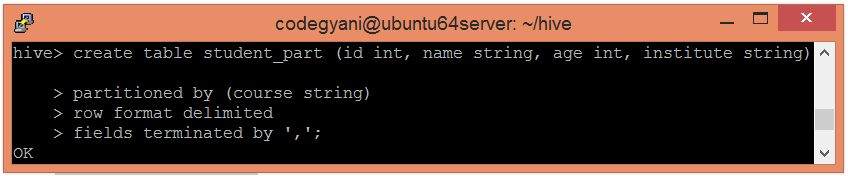
* Create a partition table by using the following command: -

hive**>** create table student\_part (id int, name string, age int, institute string)

partitioned by (course string)

row format delimited

fields terminated by ',';



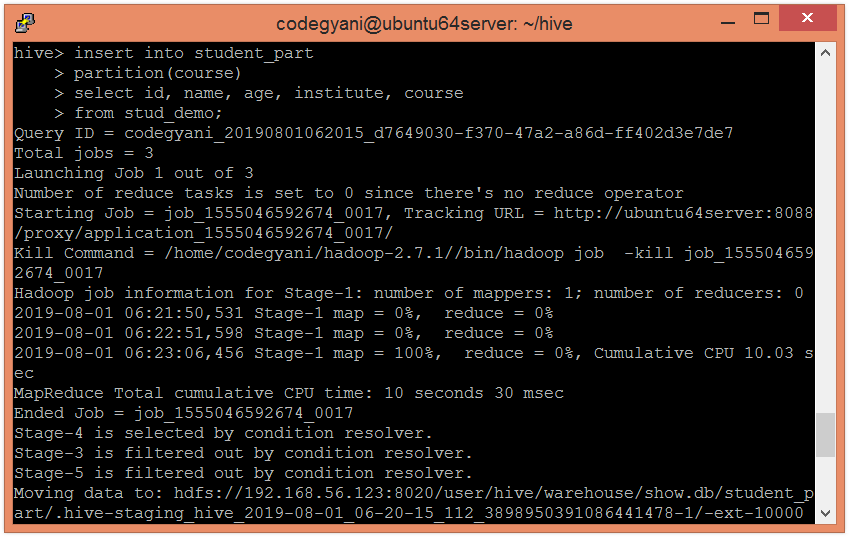
* Now, insert the data of dummy table into the partition table.

hive**>** insert into student\_part

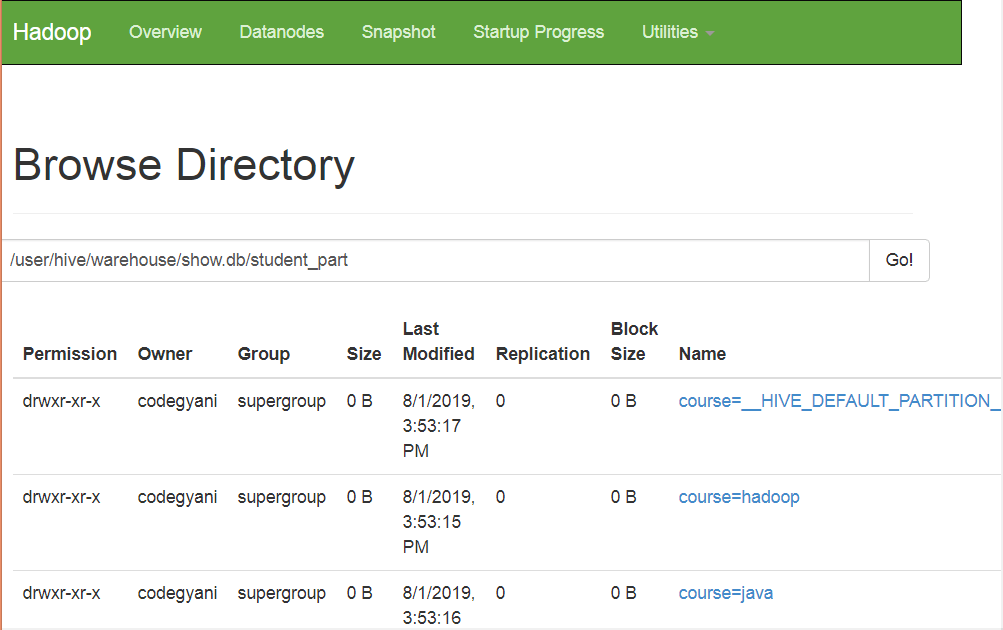
partition(course)

select id, name, age, institute, course

from stud\_demo;

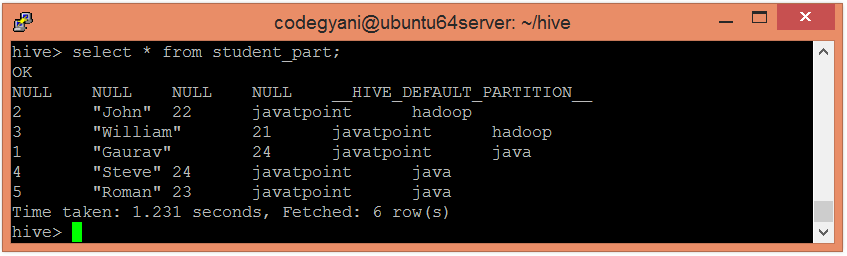


* In the following screenshot, we can see that the table student\_part is divided into two categories.



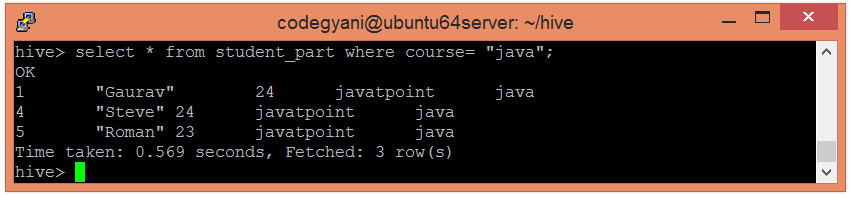
* Let's retrieve the entire data of the table by using the following command: -

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



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

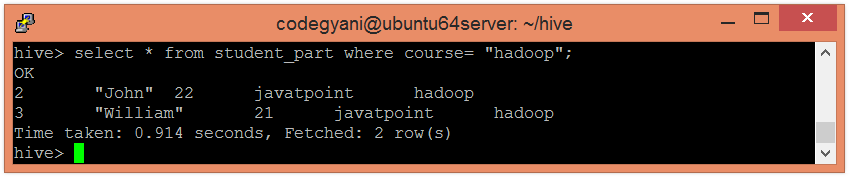
hive**>** select \* from student\_part where course= "java ";



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: -

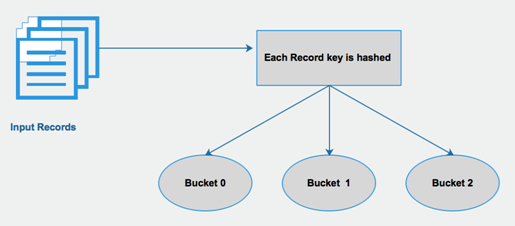
hive**>** select \* from student\_part where course= "hadoop";



# 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.

In bucketing, the partitions can be subdivided into buckets based on the hash function of a column.



* 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.

**EXAMPLES:**

1) Create a hive table

2)load the data in the hive table

3)create bucket table

4) set hive.enforce.bucketing=true;

5)load eh data into bucket table.

**PRACTICAL**

**Hive> select \* from all students;**

**Hive> create table students\_buck(sno int,sname string ,passout\_year string) clustered by(passout\_year) sorted by (sno) into 3 buckets;**

OK

Time taken:0.23 seconds

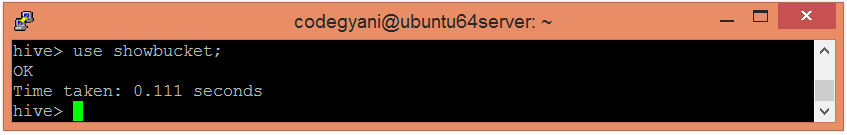
**Hive> set hive.enforce.bucketing=true;**

**Hive> insert overwrite table students\_buck select \* from all\_students;**

### **Example of Bucketing in Hive**

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

hive**>** use showbucket;

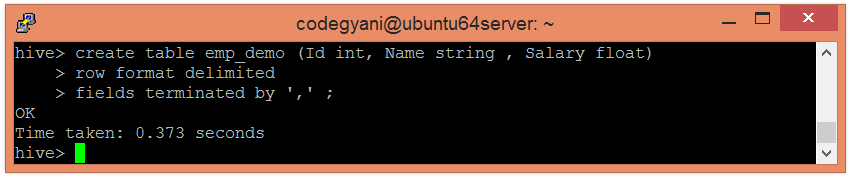


* Create a dummy table to store the data.

hive**>** create table emp\_demo (Id int, Name string , Salary float)

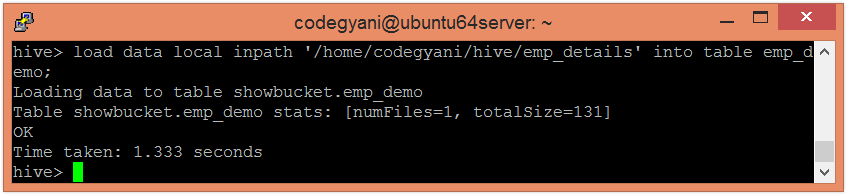
row format delimited

fields terminated by ',' ;



* Now, load the data into the table.

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



* Enable the bucketing by using the following command: -

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

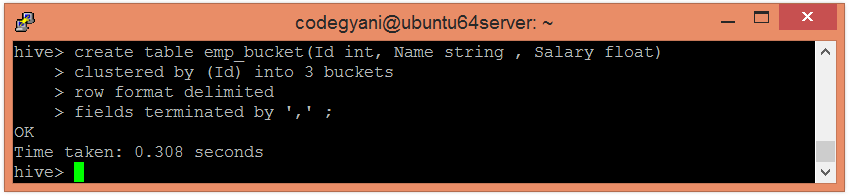
* Create a bucketing table by using the following command: -

hive**>** create table emp\_bucket(Id int, Name string , Salary float)

clustered by (Id) into 3 buckets

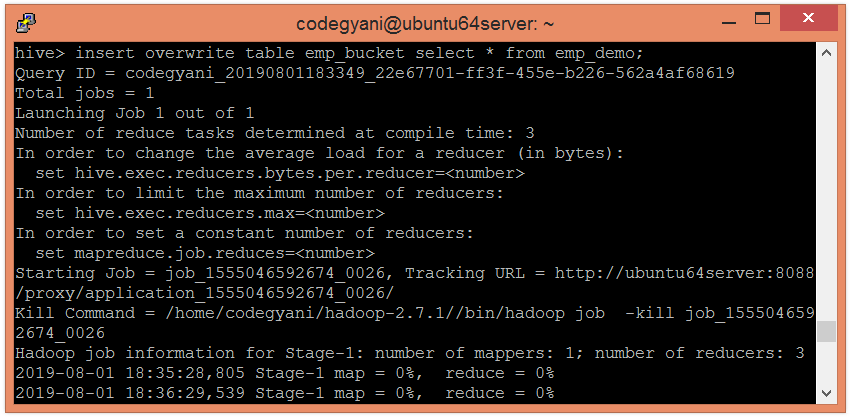
row format delimited

fields terminated by ',' ;

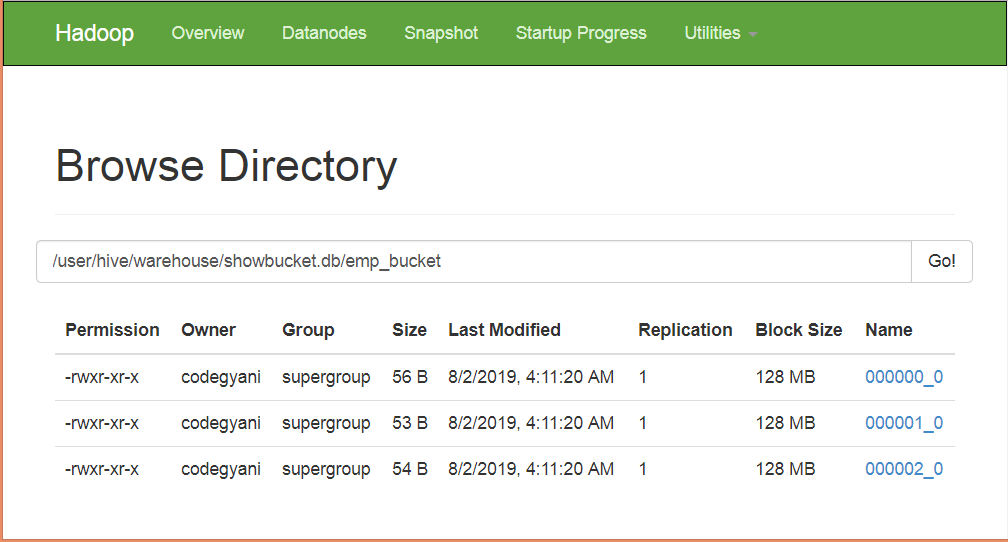


* Now, insert the data of dummy table into the bucketed table.

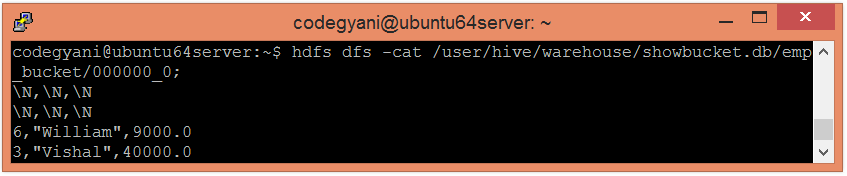
hive**>** insert overwrite table emp\_bucket select \* from emp\_demo;



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

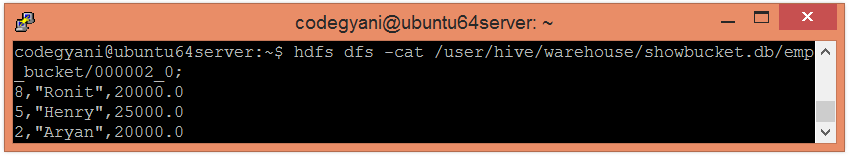


* Let's retrieve the data of bucket 0.



According to hash function :  
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 to hash function :  
8%3=2  
5%3=2  
2%3=2  
So, these columns stored in bucket 2.

In bucketing, splitting the data will be based upon rows.

In partitioning, spilting the data will be based upon on columns.