PIG

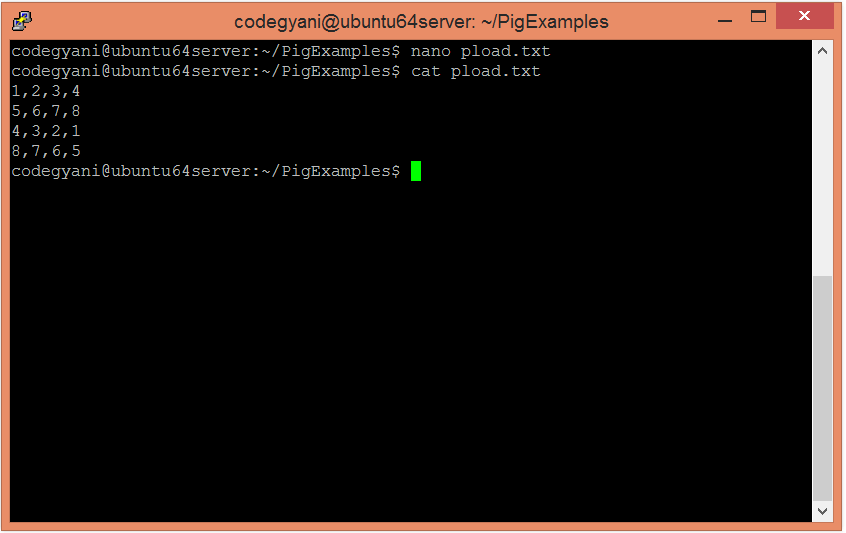
Example of LOAD Operator

In this example, we load the text file data from the file system.

Steps to execute LOAD Operator

* Create a text file in your local machine and provide some values to it.

1. $ cat pload.txt



* Upload the text files on HDFS in the specific directory.

$ hdfs dfs -put pload.txt /pigexample

* Open the pig MapReduce run mode.

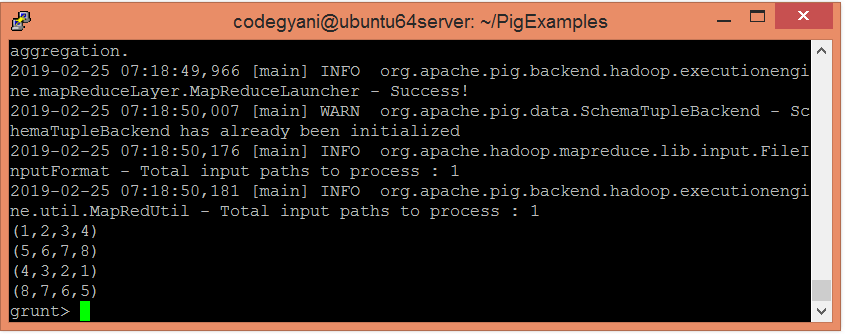
$ pig

* Load the file that contains the data.

grunt**>** A = LOAD '/pigexample/pload.txt' USING PigStorage(',') AS (a1:int,a2:int,a3:int,a4:int) ;

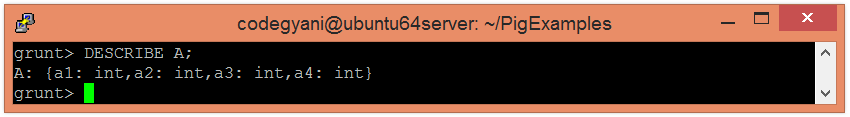
* Now, execute and verify the data.

grunt**>** DUMP A;



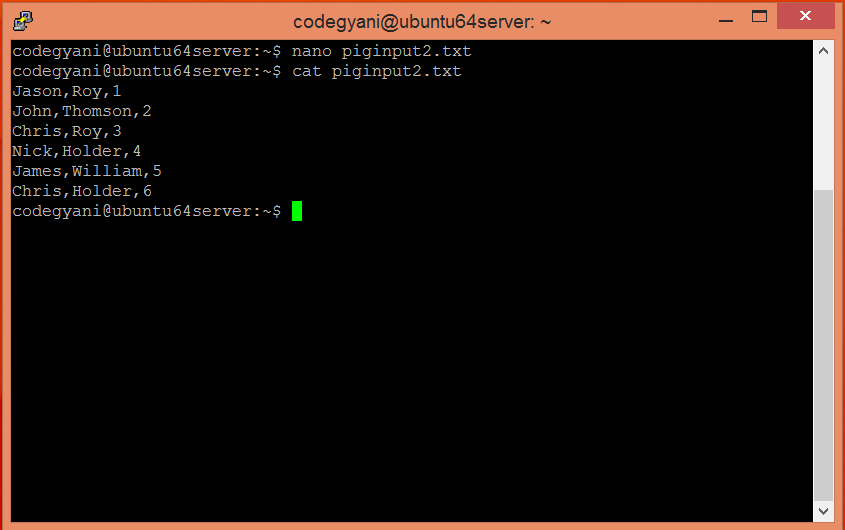
* Let's check the corresponding schema.

1. grunt**>** DESCRIBE A;



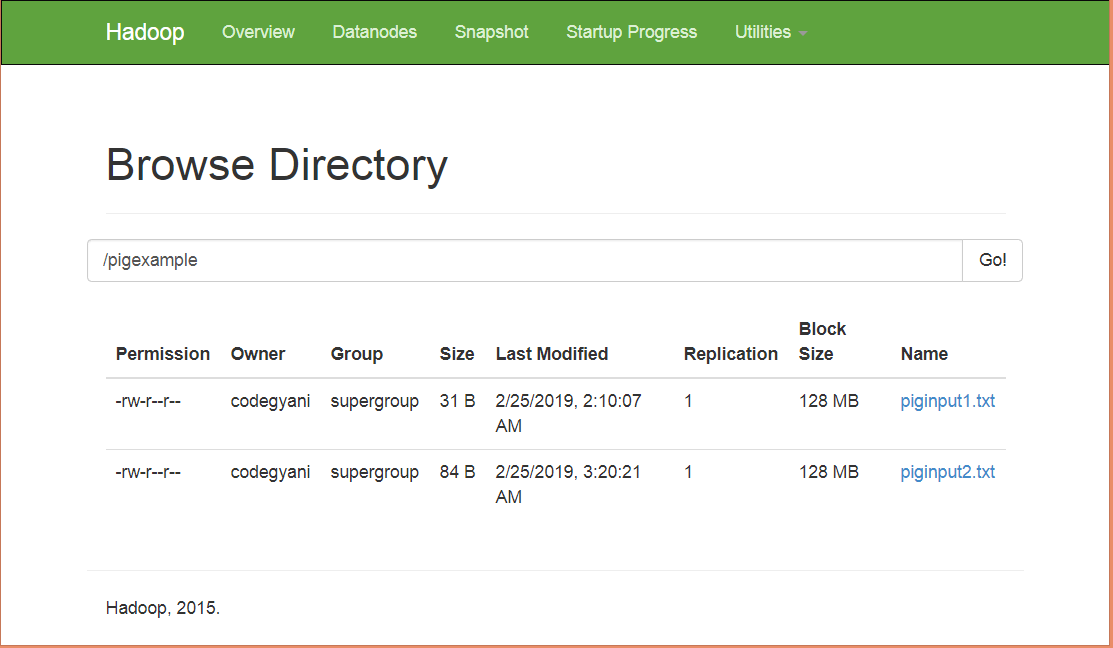
**GROUP**

1. cat piginput2.txt



* Upload the piginput2.txt file on HDFS in the specific directory.

1. $ hdfs dfs -put /home/codegyani/piginput2.txt /pigexample



* Open the pig MapReduce run mode.

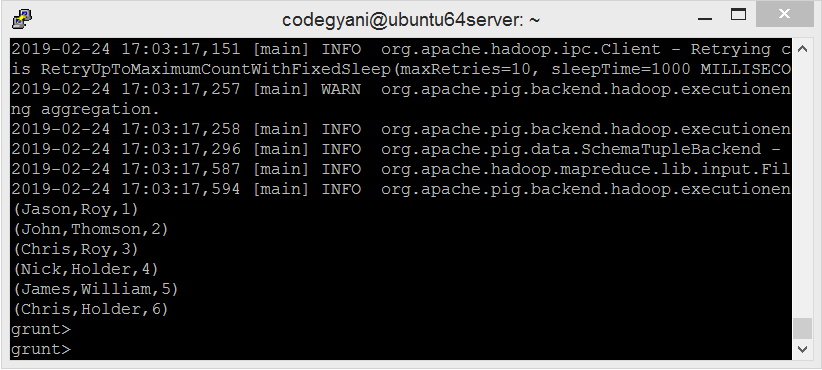
1. $ pig

* Load the data into the bag.

1. grunt**>** A = LOAD '/pigexample/piginput2.txt' USING PigStorage(',') AS (fname:chararray,l\_name:chararray,id:int);

* Now execute and verify the data.

1. grunt**>** DUMP A;

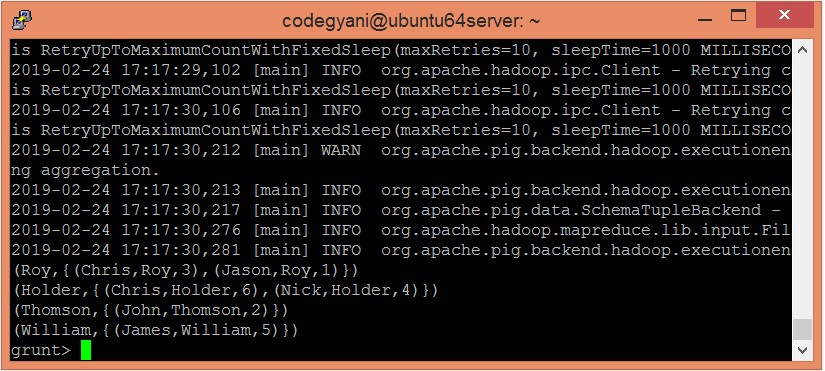


* Let us group the data on the basis of l\_name.

grunt**>** group groupbylname = group A by l\_name ;

* Now, execute and verify the data.

grunt**>** DUMP groupbylname;

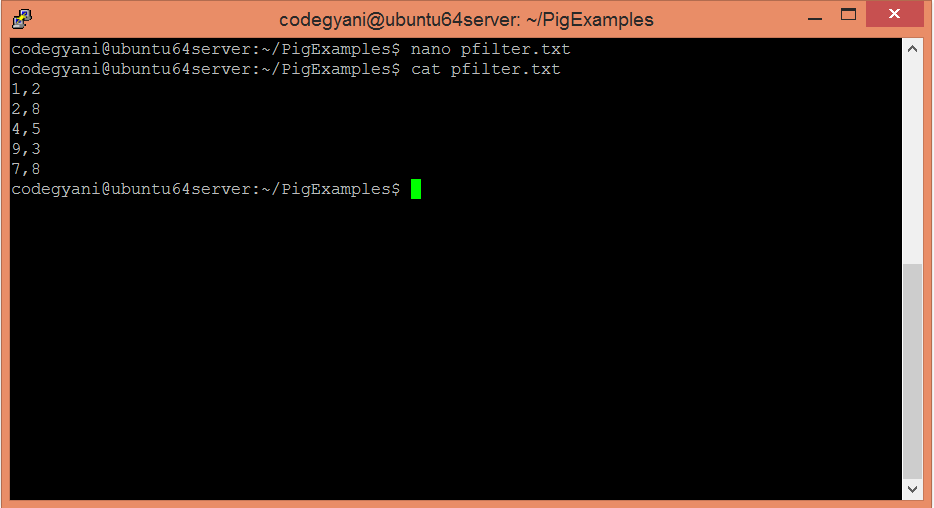


Here, we got the desired output.

## Example of FILTER Operator

In this example, we eliminate duplicate tuples.

1. cat pfilter.txt



* Upload the text files on HDFS in the specific directory.

1. $ hdfs dfs -put pfilter.txt /pigexample

* Open the pig MapReduce run mode.

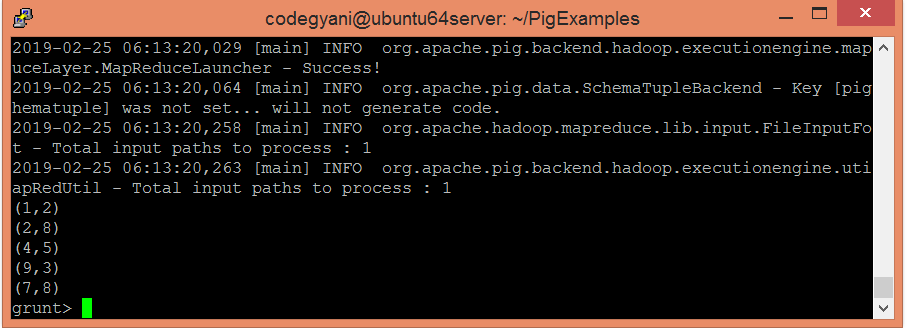
1. $ pig

* Load the file that contains the data.

1. grunt**>** A = LOAD '/pigexample/pfilter.txt' USING PigStorage(',') AS (a1:int,a2:int);

* Now, execute and verify the data

1. grunt**>** DUMP A;

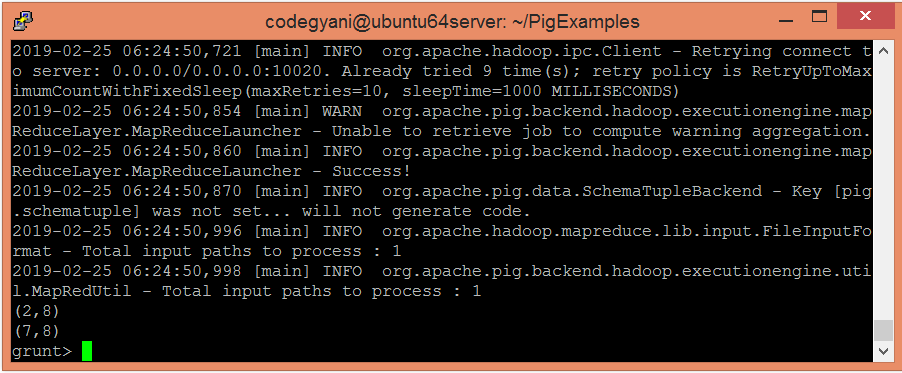


* Let's execute FILTER operator to eliminate duplicate tuples.

1. grunt**>** Result = FILTER A BY a2==8;

* Now, execute and verify the data.

1. grunt**>** DUMP Result;



**The JOIN operator**

The **JOIN** operator is used to combine records from two or more relations. While performing a join operation, we declare one (or a group of) tuple(s) from each relation, as keys. When these keys match, the two particular tuples are matched, else the records are dropped. Joins can be of the following types −

* Self-join
* Inner-join
* Outer-join − left join, right join, and full join

This chapter explains with examples how to use the join operator in Pig Latin. Assume that we have two files namely **customers.txt** and **orders.txt** in the **/pig\_data/** directory of HDFS as shown below.

**customers.txt**

1,Ramesh,32,Ahmedabad,2000.00

2,Khilan,25,Delhi,1500.00

3,kaushik,23,Kota,2000.00

4,Chaitali,25,Mumbai,6500.00

5,Hardik,27,Bhopal,8500.00

6,Komal,22,MP,4500.00

7,Muffy,24,Indore,10000.00

**orders.txt**

102,2009-10-08 00:00:00,3,3000

100,2009-10-08 00:00:00,3,1500

101,2009-11-20 00:00:00,2,1560

103,2008-05-20 00:00:00,4,2060

And we have loaded these two files into Pig with the relations **customers** and **orders** as shown below.

grunt> customers = LOAD 'hdfs://localhost:9000/pig\_data/customers.txt' USING PigStorage(',')

as(id:int,name:chararray,age:int,address:chararray,salary:int);

grunt> orders = LOAD 'hdfs://localhost:9000/pig\_data/orders.txt' USING PigStorage(',')

as(oid:int,date:chararray,customer\_id:int,amount:int);

Let us now perform various Join operations on these two relations.

Self - join

**Self-join** is used to join a table with itself as if the table were two relations, temporarily renaming at least one relation.

Generally, in Apache Pig, to perform self-join, we will load the same data multiple times, under different aliases (names). Therefore let us load the contents of the file **customers.txt** as two tables as shown below.

grunt> customers1 = LOAD 'hdfs://localhost:9000/pig\_data/customers.txt' USING PigStorage(',')

as(id:int,name:chararray,age:int,address:chararray,salary:int);

grunt> customers2 = LOAD 'hdfs://localhost:9000/pig\_data/customers.txt' USING PigStorage(',')

as(id:int,name:chararray,age:int,address:chararray,salary:int);

Syntax

Given below is the syntax of performing **self-join** operation using the **JOIN** operator.

grunt> Relation3\_name = JOIN Relation1\_name BY key, Relation2\_name BY key ;

Example

Let us perform **self-join** operation on the relation **customers**, by joining the two relations **customers1** and **customers2** as shown below.

grunt> customers3 = JOIN customers1 BY id, customers2 BY id;

Verification

Verify the relation **customers3** using the **DUMP** operator as shown below.

grunt>Dump customers3;

Output

It will produce the following output, displaying the contents of the relation **customers**.

(1,Ramesh,32,Ahmedabad,2000,1,Ramesh,32,Ahmedabad,2000)

(2,Khilan,25,Delhi,1500,2,Khilan,25,Delhi,1500)

(3,kaushik,23,Kota,2000,3,kaushik,23,Kota,2000)

(4,Chaitali,25,Mumbai,6500,4,Chaitali,25,Mumbai,6500)

(5,Hardik,27,Bhopal,8500,5,Hardik,27,Bhopal,8500)

(6,Komal,22,MP,4500,6,Komal,22,MP,4500)

(7,Muffy,24,Indore,10000,7,Muffy,24,Indore,10000)

Inner Join

**Inner Join** is used quite frequently; it is also referred to as **equijoin**. An inner join returns rows when there is a match in both tables.

It creates a new relation by combining column values of two relations (say A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, the column values for each matched pair of rows of A and B are combined into a result row.

Syntax

Here is the syntax of performing **inner join** operation using the **JOIN** operator.

grunt> result = JOIN relation1 BY columnname, relation2 BY columnname;

Example

Let us perform **inner join** operation on the two relations **customers** and **orders** as shown below.

grunt>coustomer\_orders= JOIN customers BY id, orders BY customer\_id;

Verification

Verify the relation **coustomer\_orders** using the **DUMP** operator as shown below.

grunt>Dumpcoustomer\_orders;

Output

You will get the following output that will the contents of the relation named **coustomer\_orders**.

(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)

(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)

(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)

(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)

**8. a. Run the Pig Latin Scripts to find Word Count**

**b. Run the Pig Latin Scripts to find a max temp for each and every year.**

Assume we have data in the file like below.  
filename:file

This is a hadoop post  
hadoop is a bigdata technology

1.Load the data from HDFS  
  
Use Load statement to load the data into a relation .  
As keyword used to declare column names, as we dont have any columns, we declared only one column named line.  
   
input = LOAD '/path/to/file/' AS(line:Chararray);

2. Convert the Sentence into words.  
  
The data we have is in sentences. So we have to convert that data into words using  
TOKENIZE Function.  
  
(TOKENIZE(line));  
(or)  
If we have any delimeter like space we can specify as  
(TOKENIZE(line,' '));  
  
Output will be like this:  
  
({(This),(is),(a),(hadoop),(class)})  
({(hadoop),(is),(a),(bigdata),(technology)})  
  
but we have to convert it into multiple rows like below  
  
(This)  
(is)  
(a)  
(hadoop)  
(class)  
(hadoop)  
(is)  
(a)  
(bigdata)  
(technology)

3.Convert Column into Rows  
   
I mean we have to convert every line of data into multiple rows ,for this we have function called  
FLATTEN in pig.  
  
Using FLATTEN function the bag is converted into tuple, means the array of strings  
converted into multiple rows.  
  
Words = FOREACH input GENERATE FLATTEN(TOKENIZE(line,' ')) AS word;  
  
Then the ouput is like below  
  
(This)  
(is)  
(a)  
(hadoop)  
(class)  
(hadoop)  
(is)  
(a)  
(bigdata)  
(technology)  
  
3. Apply GROUP BY  
  
We have to count each word occurance, for that we have to group all the words.  
  
Grouped = GROUP words BY word;  
  
  
4. Generate word count   
   
wordcount = FOREACH Grouped GENERATE group, COUNT(words);  
  
We can print the word count on console using Dump.  
  
DUMP wordcount;  
  
  
Output will be like below.  
   
(a,2)  
(is,2)  
(This,1)  
(class,1)  
(hadoop,2)  
(bigdata,1)  
(technology,1)

OR

Below is the complete program for the same.  
  
input = LOAD '/path/to/file/' AS(line:Chararray);  
Words = FOREACH input GENERATE FLATTEN(TOKENIZE(line,' ')) AS word;  
Grouped = GROUP words BY word;  
wordcount = FOREACH Grouped GENERATE group, COUNT(words);

**b. Run the Pig Latin Scripts to find a max temp for each and every year.**

Suppose I have the following dataset :-

Year Temp

1974 48

1974 48

1991 56

1983 89

1993 91

1938 41

1938 56

1941 93

1983 87

I want my final answer to be 93 (Pertaining to the year 1941). I am able to find the Maximum temperature for each year (Say 1941-93) but unable to find only the maximum. Any suggestions are appreciated.

Can solve this problem in two ways.

**Option1: Using (Group ALL + MAX)**

A = LOAD 'input' USING PigStorage() AS (Year:int,Temp:int);

B = GROUP A ALL;

C = FOREACH B GENERATE MAX(A.Temp);

DUMP C;

**Output:**

(93)

**Option2: Using (ORDER and LIMIT)**

A = LOAD 'input' USING PigStorage() AS (Year:int,Temp:int);

B = ORDER A BY Temp DESC;

C = LIMIT B 1;

D = FOREACH C GENERATE Temp;

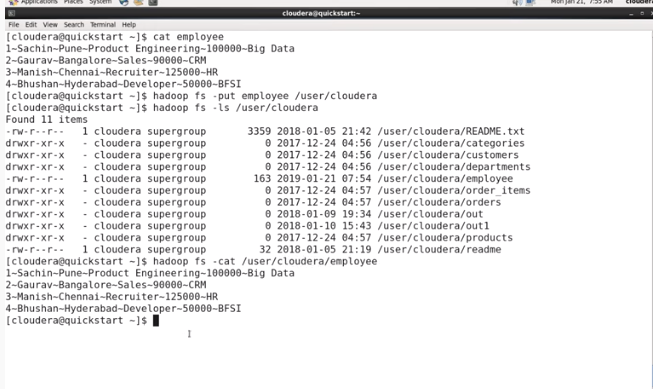
DUMP D;

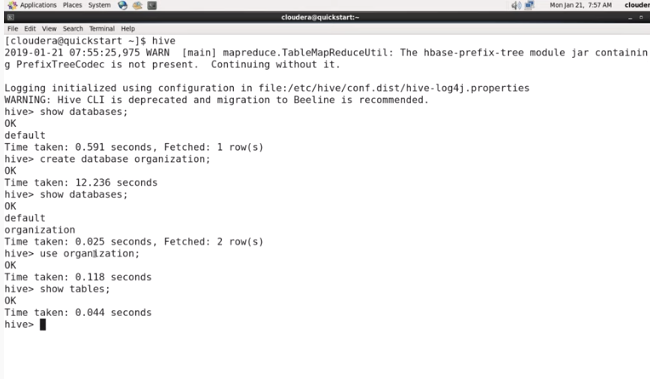
**Output:**

(93)

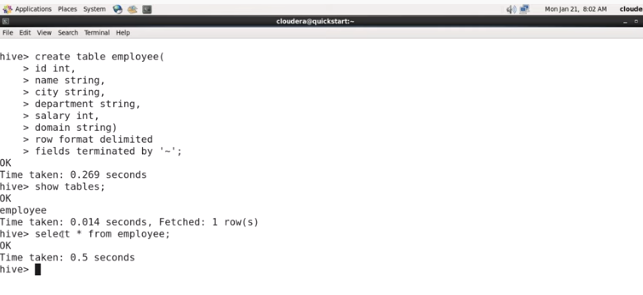
**Installation of HIVE.**

**Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.**

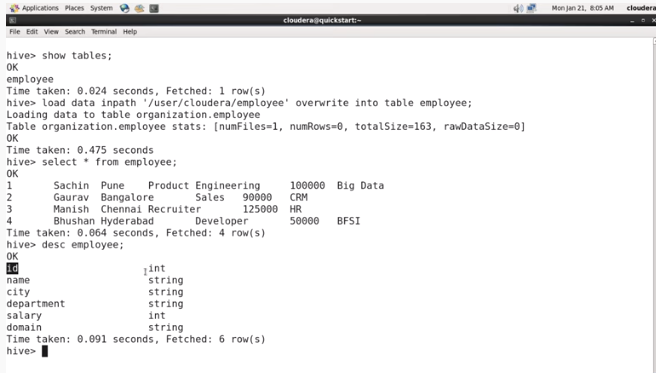




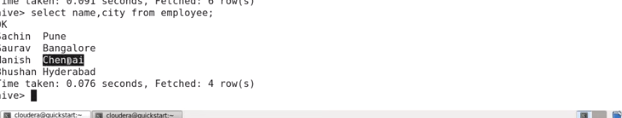
**Create table**



**Transfer data to from hadoop to hive**



**Retrieve the selected data**

****