SMIT R PATEL

19162121031

SEM 5

BDA

PRACTICAL 7

In general, Apache Pig works on top of Hadoop. It is an analytical tool that analyzes large datasets that exist in the **H**adoop **F**ile **S**ystem. To analyze data using Apache Pig, we have to initially load the data into Apache Pig. This chapter explains how to load data to Apache Pig from HDFS.

Preparing HDFS

In MapReduce mode, Pig reads (loads) data from HDFS and stores the results back in HDFS. Therefore, let us start HDFS and create the following sample data in HDFS.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student ID** | **First Name** | **Last Name** | **Phone** | **City** |
| 001 | Rajiv | Reddy | 9848022337 | Hyderabad |
| 002 | siddarth | Battacharya | 9848022338 | Kolkata |
| 003 | Rajesh | Khanna | 9848022339 | Delhi |
| 004 | Preethi | Agarwal | 9848022330 | Pune |
| 005 | Trupthi | Mohanthy | 9848022336 | Bhuwaneshwar |
| 006 | Archana | Mishra | 9848022335 | Chennai |

The above dataset contains personal details like id, first name, last name, phone number and city, of six students.

The input file of Pig contains each tuple/record in individual lines. And the entities of the record are separated by a delimiter (In our example we used **“,”**).

In the local file system, create an input file **student\_data.txt** containing data as shown below.

001,Rajiv,Reddy,9848022337,Hyderabad

002,siddarth,Battacharya,9848022338,Kolkata

003,Rajesh,Khanna,9848022339,Delhi

004,Preethi,Agarwal,9848022330,Pune

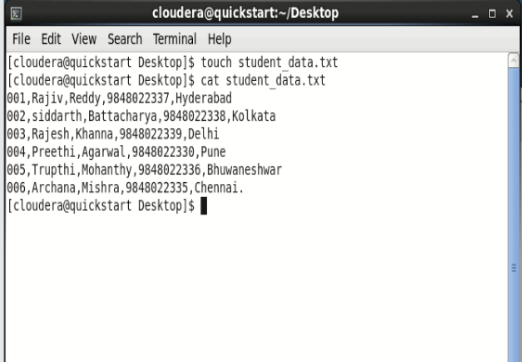
005,Trupthi,Mohanthy,9848022336,Bhuwaneshwar

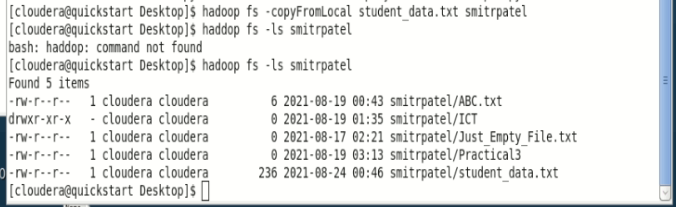
006,Archana,Mishra,9848022335,Chennai.

Now, move the file from the local file system to HDFS.

Verify whether the file has been moved into the HDFS.

You can load data into Apache Pig from the file system (HDFS/ Local) using **LOAD** operator of **Pig Latin**.





**Syntax**

The load statement consists of two parts divided by the “=” operator. On the left-hand side, we need to mention the name of the relation **where** we want to store the data, and on the right-hand side, we have to define **how** we store the data. Given below is the syntax of the **Load** operator.

Relation\_name = LOAD 'Input file path' USING function as schema; Where,

• **relation\_name** − We have to mention the relation in which we want to store the data.

• **Input file path** − We have to mention the HDFS directory where the file is stored. (In MapReduce mode)

• **function** − We have to choose a function from the set of load functions provided by Apache Pig (**BinStorage, JsonLoader, PigStorage, TextLoader**).

• **Schema** − We have to define the schema of the data. We can define the required schema as follows −

(column1 : data type, column2 : data type, column3 : data type);

**Note** − We load the data without specifying the schema. In that case, the columns will be addressed as $01, $02, etc… (check).

**Example**

As an example, let us load the data in **student\_data.txt** in Pig under the schema named **Student** using the **LOAD** command.

|  |
| --- |
| grunt> student = LOAD  'hdfs://localhost:9000/pig\_data/student\_data.txt'  USING PigStorage(',')  as ( id:int, firstname:chararray, lastname:chararray, phone:chararray,  city:chararray ); |

Following is the description of the above statement.

|  |  |
| --- | --- |
| Relation name | We have stored the data in the schema **student**. |
| Input  file path | We are reading data from the file **student\_data.txt,** which is in the /pig\_data/ directory of HDFS. |
| Storage function | We have used the **PigStorage()** function. It loads and stores data as structured text files. It takes a delimiter using which each entity of a tuple is separated, as a parameter. By default, it takes ‘\t’ as a parameter. |
| schema | We have stored the data using the following schema.  column id firstname lastname phone city datatype int char array char array char array char array |

**Note** − The **load** statement will simply load the data into the specified relation in Pig.

Dump Operator

The **Dump** operator is used to run the Pig Latin statements and display the results on the screen. It is generally used for debugging Purpose.

**Syntax**

Given below is the syntax of the **Dump** operator.

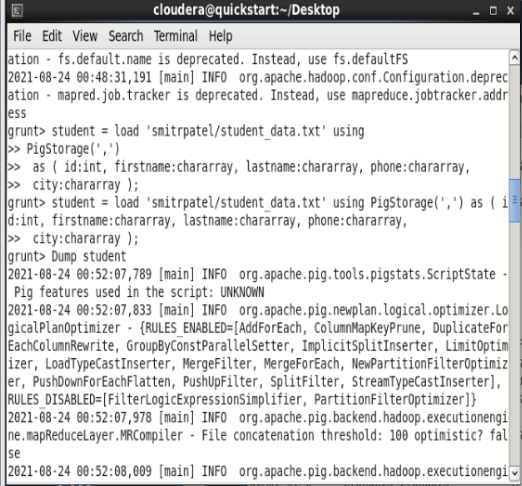
grunt> Dump Relation\_Name

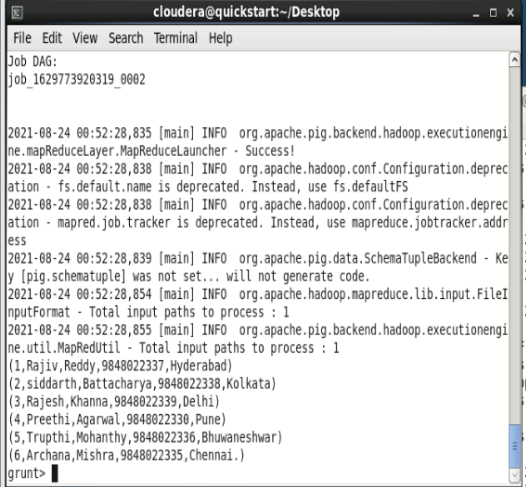
Now, let us print the contents of the relation using the **Dump operator** as shown below.

grunt> Dump student

Once you execute the above **Pig Latin** statement, it will start a MapReduce job to read data from HDFS.

grunt> student = load 'smitrpatel/student\_data.txt' using PigStorage(',') as ( id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray );





Describe Operator

The **describe** operator is used to view the schema of a relation.

Syntax

The syntax of the **describe** operator is as follows −

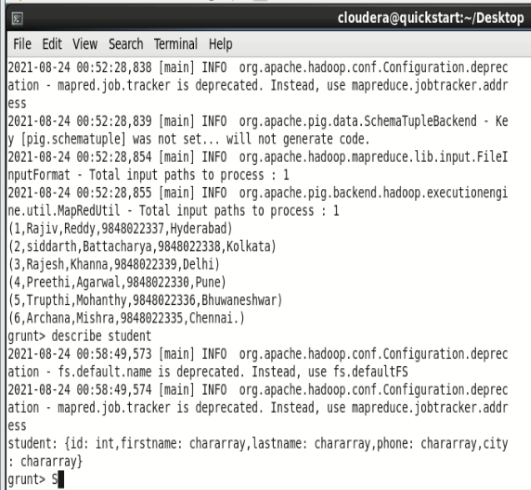
grunt> Describe Relation\_name

let us describe the relation named **student** and verify the schema as shown below. grunt> describe student;

Output

Once you execute the above **Pig Latin** statement, it will produce the following output.

grunt> student: { id: int,firstname: chararray,lastname: chararray,phone: chararray,city: chararray }



Explain Operator

The **explain** operator is used to display the logical, physical, and MapReduce execution plans of a relation.

Syntax

Given below is the syntax of the **explain** operator.

grunt> explain Relation\_name;

let us explain the relation named student using the **explain** operator as shown below.

grunt> explain student;

Output

It will produce the following output.

**grunt> explain student;**

2021-08-24 01:04:28,861 [main] INFO org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES\_ENABLED=[AddForEach, ColumnMapKeyPrune, DuplicateForEachColumnRewrite, GroupByConstParallelSetter, ImplicitSplitInserter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NewPartitionFilterOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter], RULES\_DISABLED=[FilterLogicExpressionSimplifier, PartitionFilterOptimizer]}

#-----------------------------------------------

# New Logical Plan:

#-----------------------------------------------

student: (Name: LOStore Schema: id#31:int,firstname#32:chararray,lastname#33:chararray,phone#34:chararray,city#35:chararray)

|

|---student: (Name: LOForEach Schema: id#31:int,firstname#32:chararray,lastname#33:chararray,phone#34:chararray,city#35:chararray)

| |

| (Name: LOGenerate[false,false,false,false,false] Schema: id#31:int,firstname#32:chararray,lastname#33:chararray,phone#34:chararray,city#35:chararray)ColumnPrune:InputUids=[34, 35, 32, 33, 31]ColumnPrune:OutputUids=[34, 35, 32, 33, 31]

| | |

| | (Name: Cast Type: int Uid: 31)

| | |

| | |---id:(Name: Project Type: bytearray Uid: 31 Input: 0 Column: (\*))

| | |

| | (Name: Cast Type: chararray Uid: 32)

| | |

| | |---firstname:(Name: Project Type: bytearray Uid: 32 Input: 1 Column: (\*))

| | |

| | (Name: Cast Type: chararray Uid: 33)

| | |

| | |---lastname:(Name: Project Type: bytearray Uid: 33 Input: 2 Column: (\*))

| | |

| | (Name: Cast Type: chararray Uid: 34)

| | |

| | |---phone:(Name: Project Type: bytearray Uid: 34 Input: 3 Column: (\*))

| | |

| | (Name: Cast Type: chararray Uid: 35)

| | |

| | |---city:(Name: Project Type: bytearray Uid: 35 Input: 4 Column: (\*))

| |

| |---(Name: LOInnerLoad[0] Schema: id#31:bytearray)

| |

| |---(Name: LOInnerLoad[1] Schema: firstname#32:bytearray)

| |

| |---(Name: LOInnerLoad[2] Schema: lastname#33:bytearray)

| |

| |---(Name: LOInnerLoad[3] Schema: phone#34:bytearray)

| |

| |---(Name: LOInnerLoad[4] Schema: city#35:bytearray)

|

|---student: (Name: LOLoad Schema: id#31:bytearray,firstname#32:bytearray,lastname#33:bytearray,phone#34:bytearray,city#35:bytearray)RequiredFields:null

#-----------------------------------------------

# Physical Plan:

#-----------------------------------------------

student: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-36

|

|---student: New For Each(false,false,false,false,false)[bag] - scope-35

| |

| Cast[int] - scope-21

| |

| |---Project[bytearray][0] - scope-20

| |

| Cast[chararray] - scope-24

| |

| |---Project[bytearray][1] - scope-23

| |

| Cast[chararray] - scope-27

| |

| |---Project[bytearray][2] - scope-26

| |

| Cast[chararray] - scope-30

| |

| |---Project[bytearray][3] - scope-29

| |

| Cast[chararray] - scope-33

| |

| |---Project[bytearray][4] - scope-32

|

|---student: Load(hdfs://quickstart.cloudera:8020/user/cloudera/smitrpatel/student\_data.txt:PigStorage(',')) - scope-19

2021-08-24 01:04:28,869 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompiler - File concatenation threshold: 100 optimistic? false

2021-08-24 01:04:28,870 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size before optimization: 1

2021-08-24 01:04:28,870 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQueryOptimizer - MR plan size after optimization: 1

#--------------------------------------------------

# Map Reduce Plan

#--------------------------------------------------

MapReduce node scope-37

Map Plan

student: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-36

|

|---student: New For Each(false,false,false,false,false)[bag] - scope-35

| |

| Cast[int] - scope-21

| |

| |---Project[bytearray][0] - scope-20

| |

| Cast[chararray] - scope-24

| |

| |---Project[bytearray][1] - scope-23

| |

| Cast[chararray] - scope-27

| |

| |---Project[bytearray][2] - scope-26

| |

| Cast[chararray] - scope-30

| |

| |---Project[bytearray][3] - scope-29

| |

| Cast[chararray] - scope-33

| |

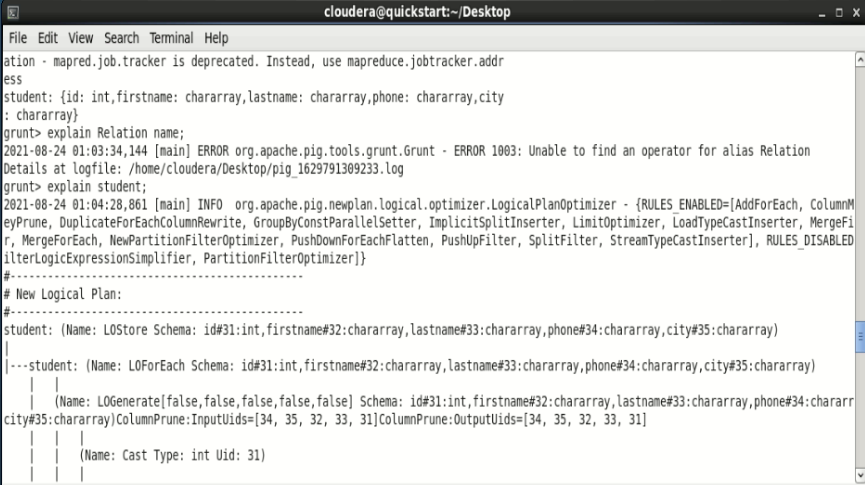
| |---Project[bytearray][4] - scope-32

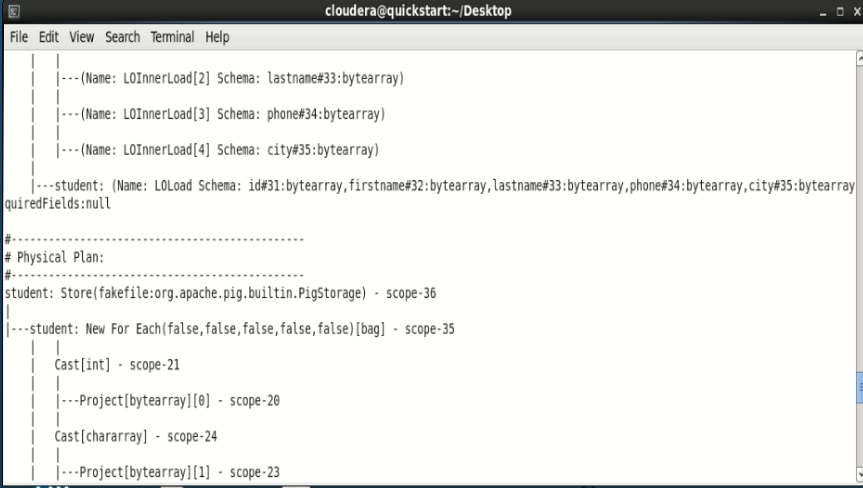
|

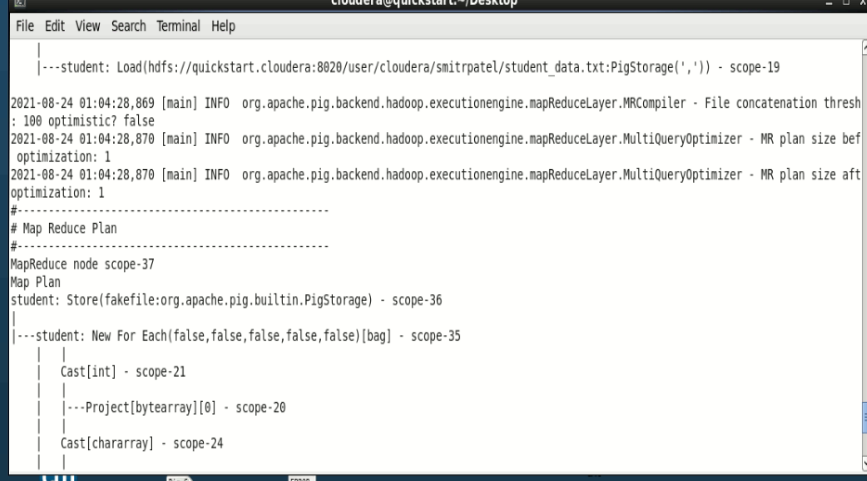
|---student: Load(hdfs://quickstart.cloudera:8020/user/cloudera/smitrpatel/student\_data.txt:PigStorage(',')) - scope-19--------

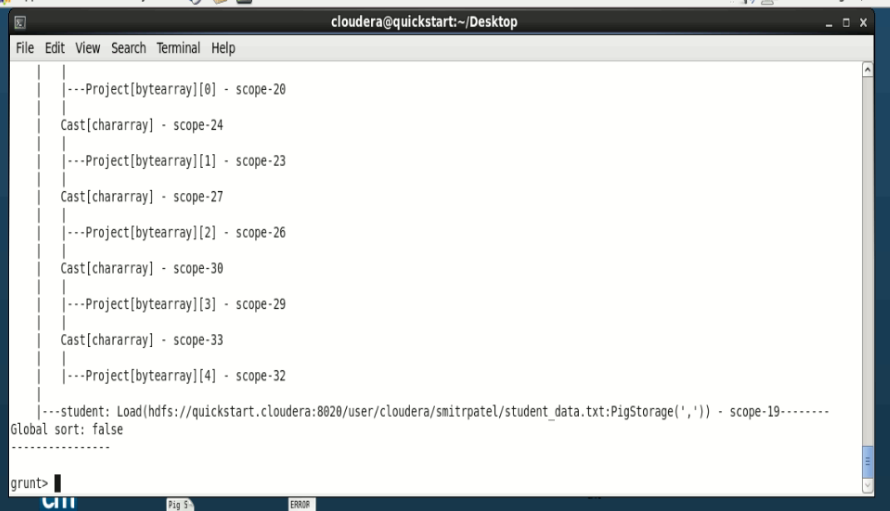
Global sort: false

----------------









Illustrate Operator

The **illustrate** operator gives you the step-by-step execution of a sequence of statements.

Syntax

Given below is the syntax of the **illustrate** operator.

grunt> illustrate Relation\_name;

let us illustrate the relation named student as shown below.

grunt> illustrate student;

Output

On executing the above statement, you will get the following output.

**grunt> illustrate student;**

**2021-08-24 01:11:33,330 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigMapOnly$Map - Aliases being processed per job phase (AliasName[line,offset]): M: student[8,10] C: R:**

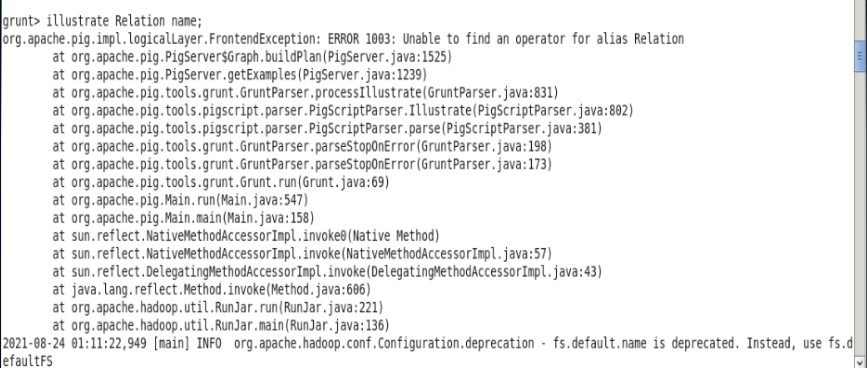
**---------------------------------------------------------------------------------------------------------------------**

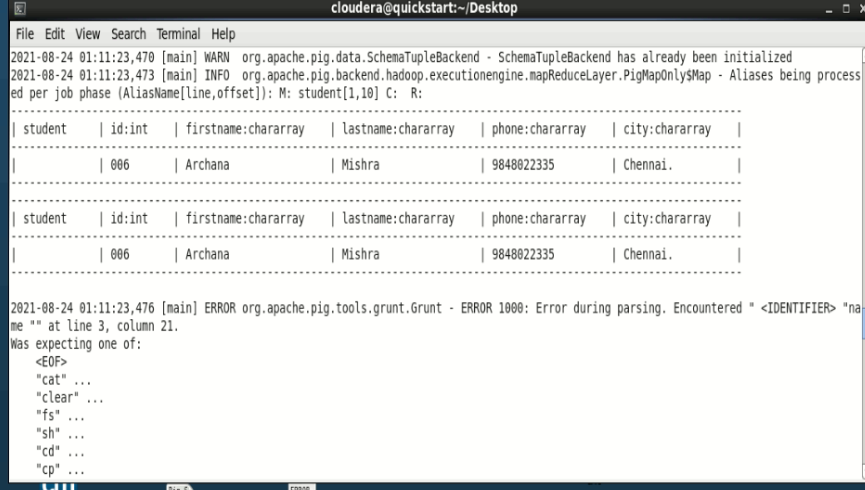
**| student | id:int | firstname:chararray | lastname:chararray | phone:chararray | city:chararray |**

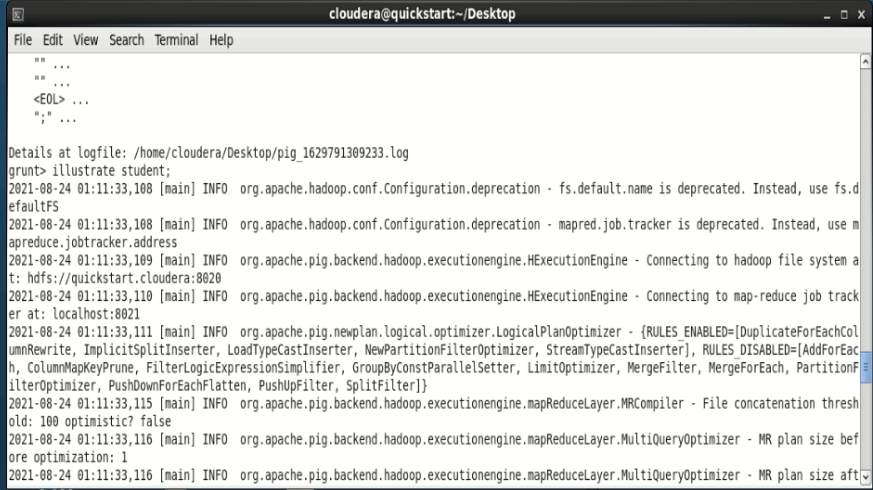
**---------------------------------------------------------------------------------------------------------------------**

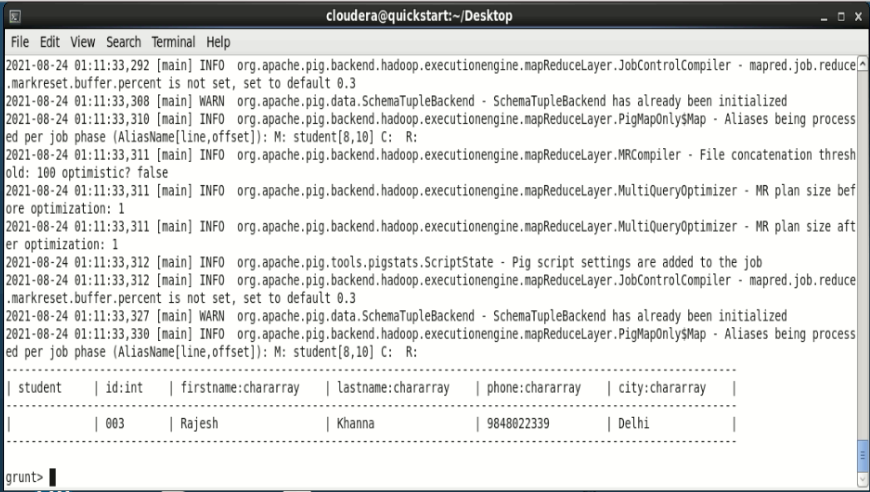
**| | 003 | Rajesh | Khanna | 9848022339 | Delhi |**

**---------------------------------------------------------------------------------------------------------------------**









Group Operator

The GROUP operator is used to group the data in one or more relations. It collects the data having the same key.

Syntax

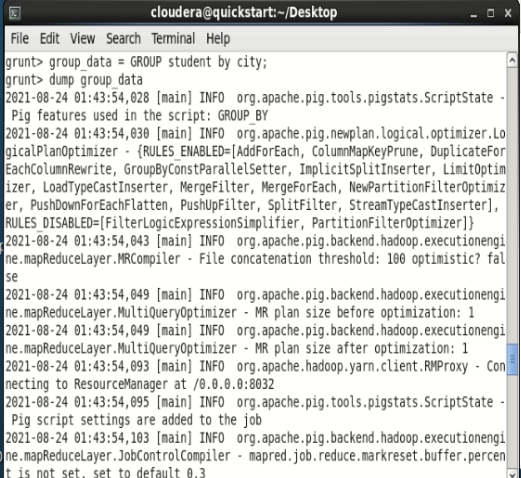
Given below is the syntax of the group operator.

grunt> Group\_data = GROUP Relation\_name BY age;

let us group the records/tuples in the relation by age as shown below. grunt> group\_data = GROUP student\_details by age;

grunt> group\_data = GROUP student by city;

grunt> dump group\_data



Verification

Verify the relation **group\_data** using the **DUMP** operator as shown below. grunt> Dump group\_data;

Output

Then you will get output displaying the contents of the relation named **group\_data** as shown below. Here you can observe that the resulting schema has two columns −

• One is **age**, by which we have grouped the relation.

• The other is a **bag**, which contains the group of tuples, student records with the respective age.

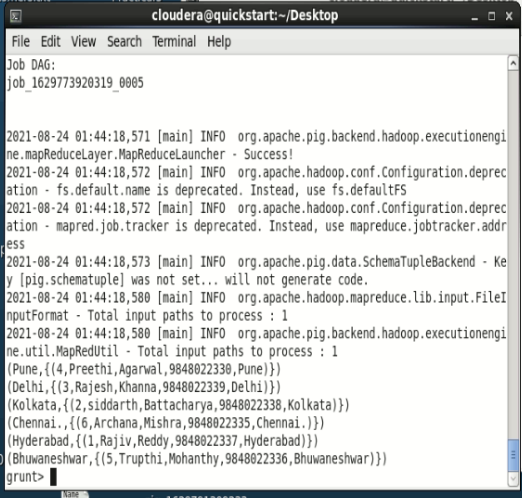
(21,{(4,Preethi,Agarwal,21,9848022330,Pune),(1,Rajiv,Reddy,21,984 8022337,Hydera bad)})

(22,{(3,Rajesh,Khanna,22,9848022339,Delhi),(2,siddarth,Battachary a,22,984802233 8,Kolkata)})

(23,{(6,Archana,Mishra,23,9848022335,Chennai),(5,Trupthi,Mohanthy ,23,9848022336 ,Bhuwaneshwar)})

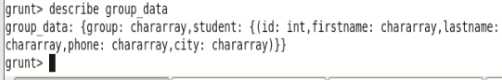
(24,{(8,Bharathi,Nambiayar,24,9848022333,Chennai),(7,Komal,Nayak, 24,9848022334, trivendram)})

You can see the schema of the table after grouping the data using the **describe** command as shown below.



**grunt> Describe group\_data;**

group\_data: {group: int,student\_details: {(id: int,firstname: chararray, lastname: chararray,age: int,phone: chararray,city: chararray)}}



In the same way, you can get the sample illustration of the schema using the illustrate command as shown below.

grunt>illustrate group\_data;

It will produce the following output –

WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized

2021-08-24 01:49:45,048 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigMapReduce$Reduce - Aliases being processed per job phase (AliasName[line,offset]): M: student[2,10],student[-1,-1],group\_data[3,13] C: R:

------------------------------------------------------------------------------------------------------------------------

| student | id:int | firstname:chararray | lastname:chararray | phone:chararray | city:chararray |

------------------------------------------------------------------------------------------------------------------------

| | 1 | Rajiv | Reddy | 9848022337 | Hyderabad |

| | 1 | Rajiv | Reddy | 9848022337 | Hyderabad |

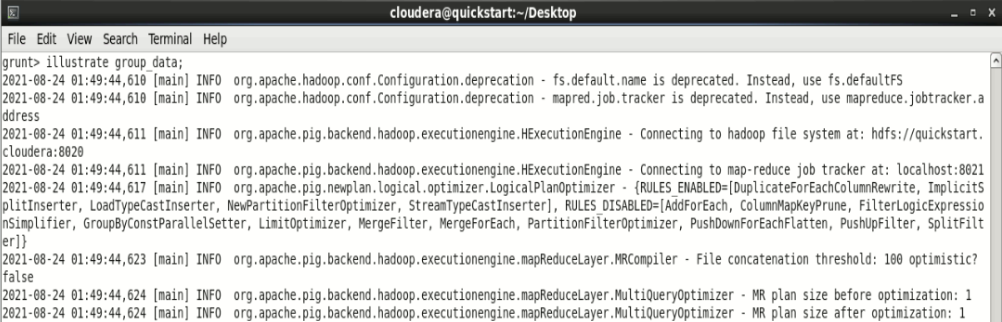
------------------------------------------------------------------------------------------------------------------------

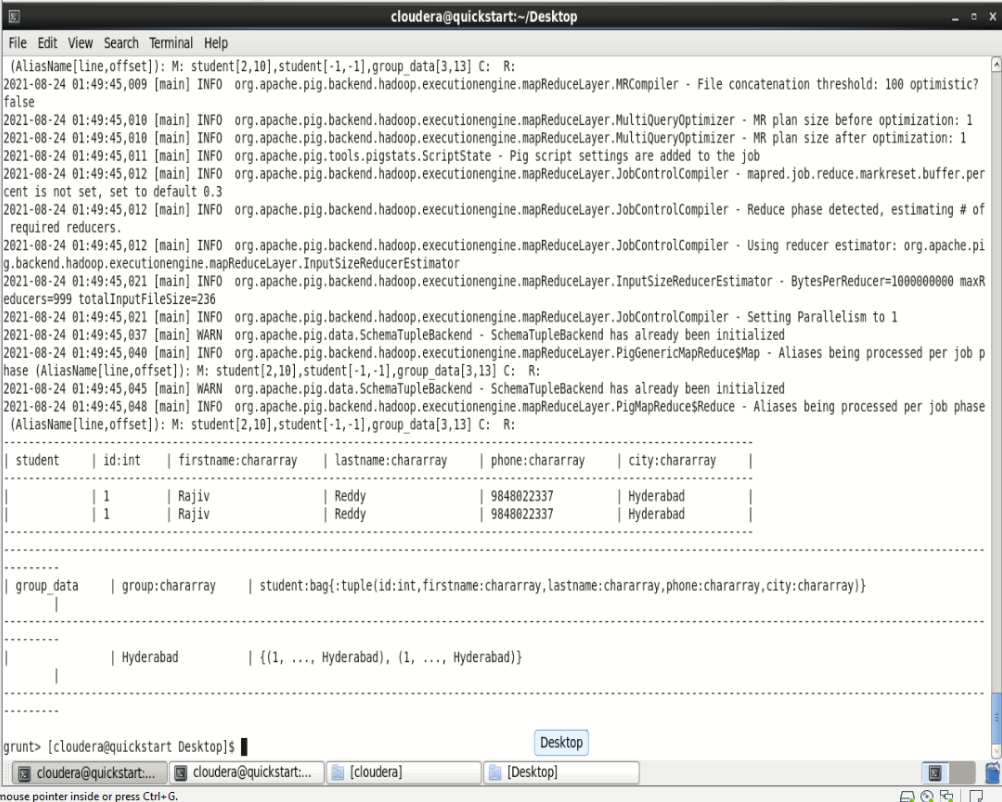
----------------------------------------------------------------------------------------------------------------------------------------------------------------------

| group\_data | group:chararray | student:bag{:tuple(id:int,firstname:chararray,lastname:chararray,phone:chararray,city:chararray)} |

----------------------------------------------------------------------------------------------------------------------------------------------------------------------

| | Hyderabad | {(1, ..., Hyderabad), (1, ..., Hyderabad)} |

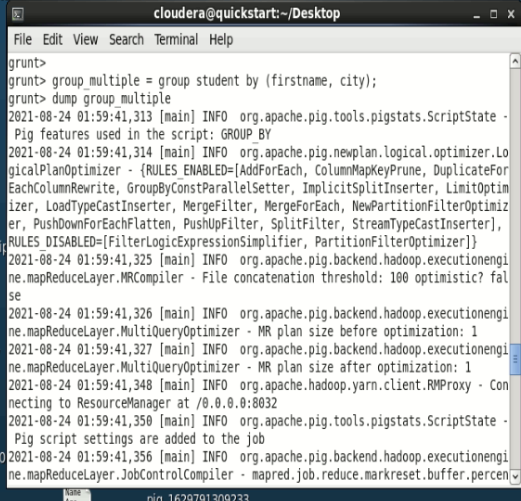




Grouping by Multiple Columns

Let us group the relation by age and city as shown below.

grunt> group\_multiple = GROUP student\_details by (age, city);



You can verify the content of the relation named **group\_multiple** using the Dump operator as shown below.

**grunt> dump group\_multiple;**

2021-08-24 02:00:05,184 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

((Rajiv,Hyderabad),{(1,Rajiv,Reddy,9848022337,Hyderabad)})

((Rajesh,Delhi),{(3,Rajesh,Khanna,9848022339,Delhi)})

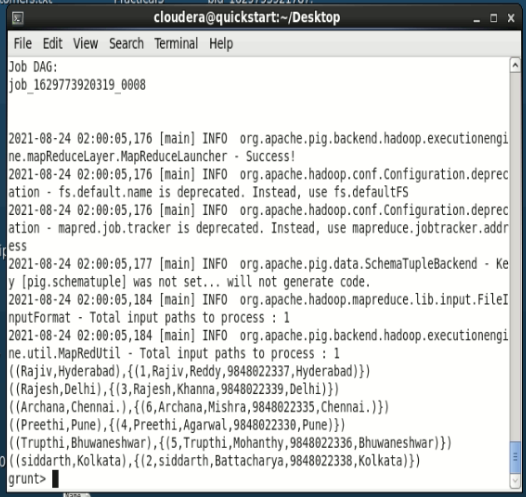
((Archana,Chennai.),{(6,Archana,Mishra,9848022335,Chennai.)})

((Preethi,Pune),{(4,Preethi,Agarwal,9848022330,Pune)})

((Trupthi,Bhuwaneshwar),{(5,Trupthi,Mohanthy,9848022336,Bhuwaneshwar)})

((siddarth,Kolkata),{(2,siddarth,Battacharya,9848022338,Kolkata)})

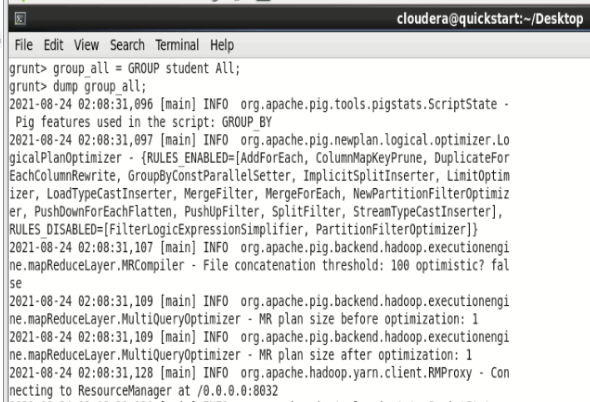
grunt>



Group All

You can group a relation by all the columns as shown below.

grunt> **group\_all** = GROUP **student\_details** All;



Now, verify the content of the relation **group\_all** as shown below.

**grunt> Dump group\_all;**

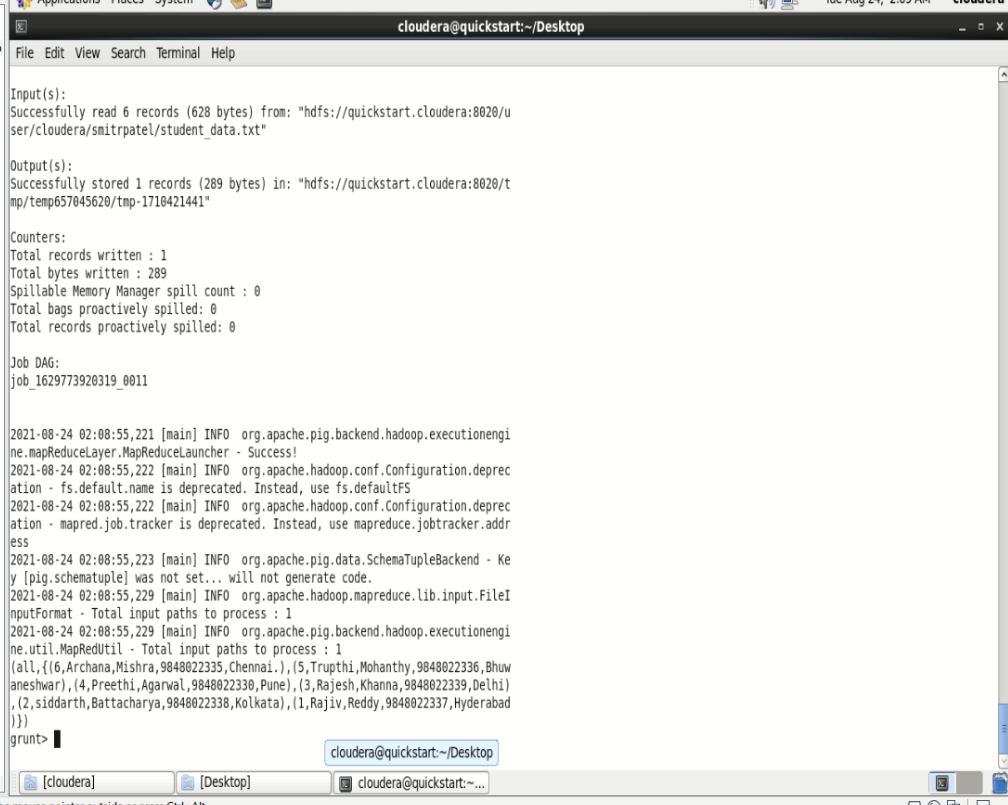
2021-08-24 02:08:55,223 [main] INFO org.apache.pig.data.SchemaTupleBackend - Key [pig.schematuple] was not set... will not generate code.

2021-08-24 02:08:55,229 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input paths to process : 1

2021-08-24 02:08:55,229 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(all,{(6,Archana,Mishra,9848022335,Chennai.),(5,Trupthi,Mohanthy,9848022336,Bhuwaneshwar),(4,Preethi,Agarwal,9848022330,Pune),(3,Rajesh,Khanna,9848022339,Delhi),(2,siddarth,Battacharya,9848022338,Kolkata),(1,Rajiv,Reddy,9848022337,Hyderabad)})

grunt>



Cogroup Operator

The **COGROUP** operator works more or less in the same way as the GROUP operator. The only difference between the two operators is that the **group** operator is normally used with one relation, while the **cogroup** operator is used in statements involving two or more relations.

Grouping Two Relations using Cogroup

Assume that we have two files namely **student\_details.txt** and **employee\_details.txt** in the HDFS directory **/pig\_data/**

**employee\_details.txt**

001,Robin,22,newyork

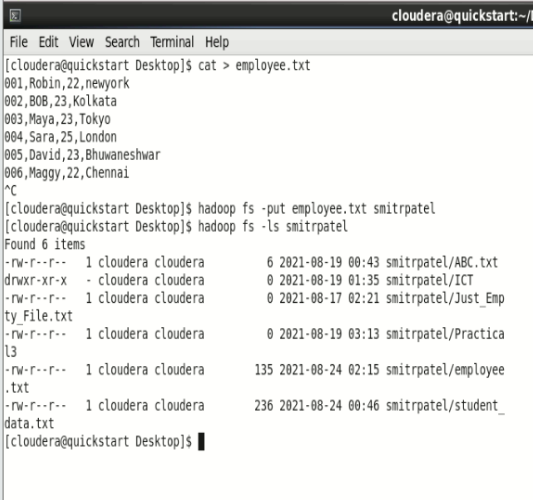
002,BOB,23,Kolkata

003,Maya,23,Tokyo

004,Sara,25,London

005,David,23,Bhuwaneshwar

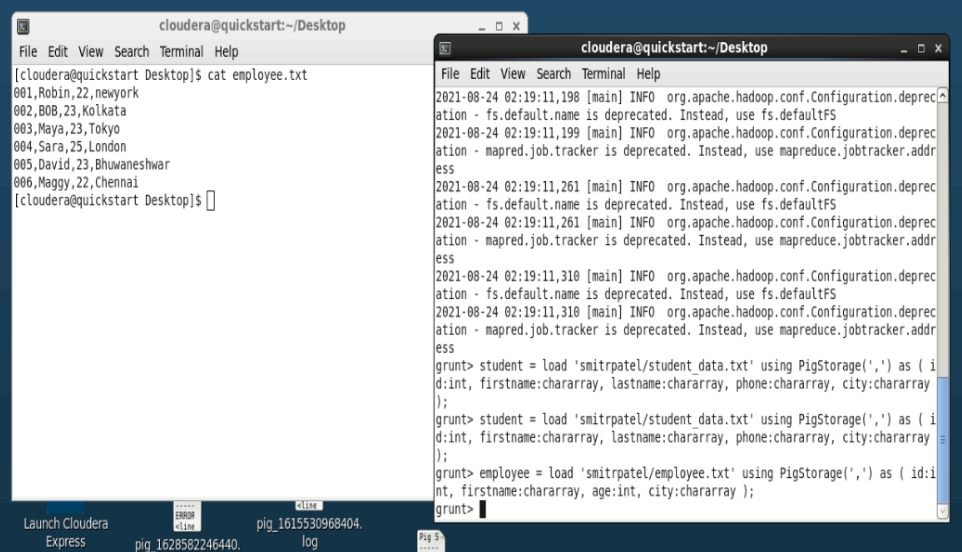
006,Maggy,22,Chennai



And we have loaded these files into Pig with the relation names **student\_details** and **employee\_details** respectively

Now, let us group the records/tuples of the relations **student\_details** and **employee\_details** with the key age, as shown below.

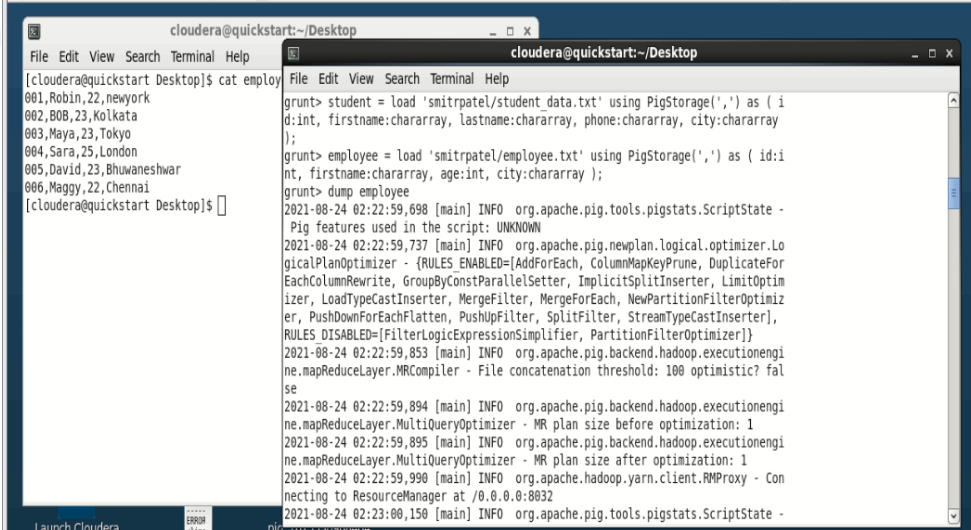
|  |
| --- |
| grunt> cogroup\_data = COGROUP student\_details by age, employee\_details by age; |

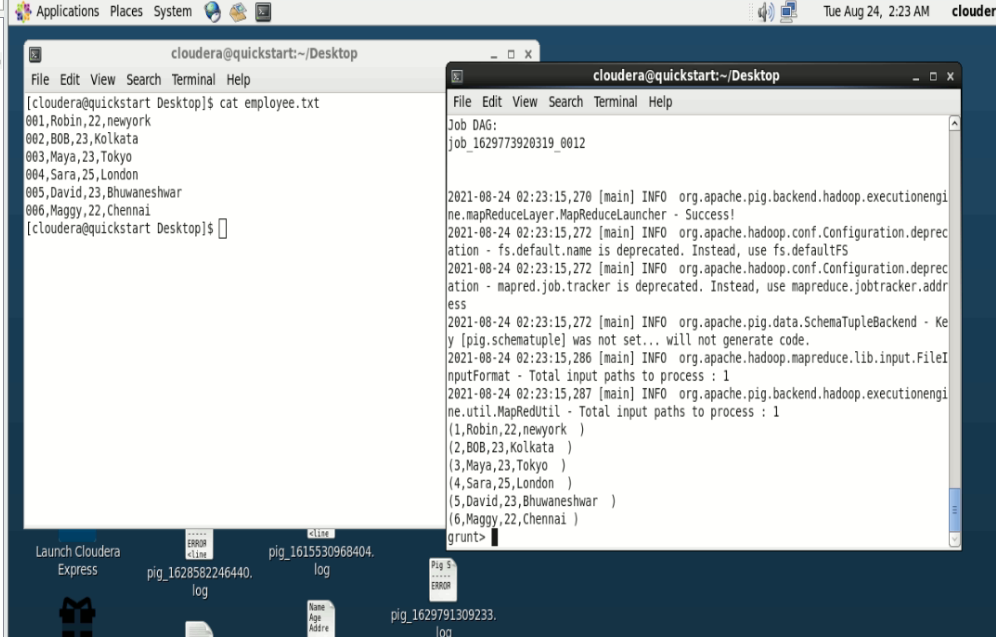


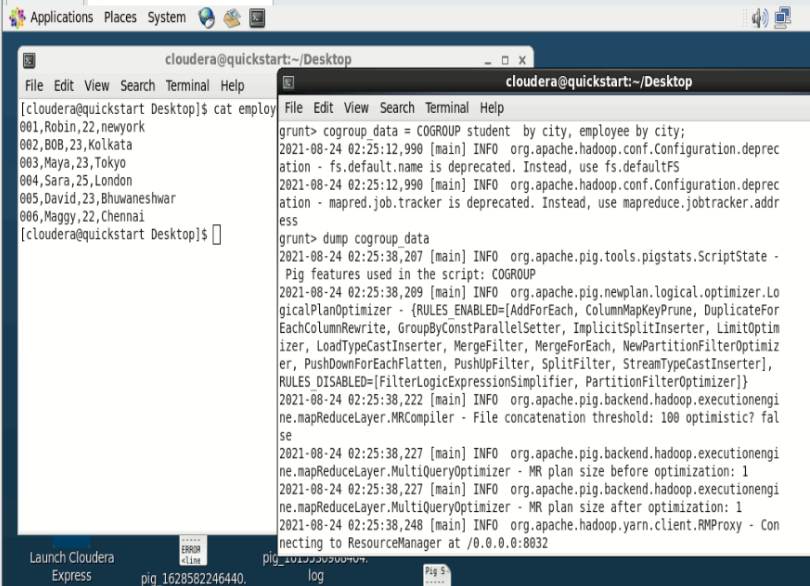
grunt>student = load 'smitrpatel/student\_data.txt' using PigStorage(',') as ( id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray );

grunt>employee = load 'smitrpatel/employee.txt' using PigStorage(',') as ( id:int, firstname:chararray, age:int, city:chararray );

grunt>dump employee





****

**Verification**

Verify the relation **cogroup\_data** using the **DUMP** operator as shown below.

|  |
| --- |
| grunt> Dump cogroup\_data; |

**Output**

It will produce the following output, displaying the contents of the relation named **cogroup\_data** as shown below.

2021-08-24 02:26:02,647 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(Pune,{(4,Preethi,Agarwal,9848022330,Pune)},{})

(Delhi,{(3,Rajesh,Khanna,9848022339,Delhi)},{})

(Kolkata,{(2,siddarth,Battacharya,9848022338,Kolkata)},{})

(Tokyo ,{},{(3,Maya,23,Tokyo )})

(Chennai ,{},{(6,Maggy,22,Chennai )})

(Chennai.,{(6,Archana,Mishra,9848022335,Chennai.)},{})

(London ,{},{(4,Sara,25,London )})

(Hyderabad,{(1,Rajiv,Reddy,9848022337,Hyderabad)},{})

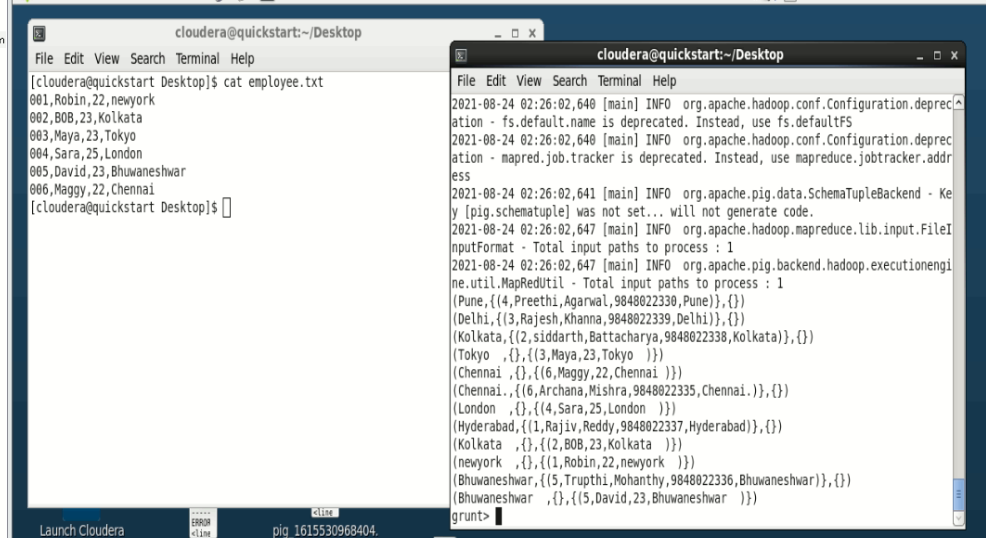
(Kolkata ,{},{(2,BOB,23,Kolkata )})

(newyork ,{},{(1,Robin,22,newyork )})

(Bhuwaneshwar,{(5,Trupthi,Mohanthy,9848022336,Bhuwaneshwar)},{})

(Bhuwaneshwar ,{},{(5,David,23,Bhuwaneshwar )})

grunt>



The **cogroup** operator groups the tuples from each relation according to age where each group depicts a particular age value.

For example, if we consider the 1st tuple of the result, it is grouped by age 21. And it contains two bags −

• the first bag holds all the tuples from the first relation (**student\_details** in this case) having age 21, and

• the second bag contains all the tuples from the second relation (**employee\_details** in this case) having age 21.

In case a relation doesn’t have tuples having the age value 21, it returns an empty bag.