Institute of Computer Technology

B. Tech. Computer Science and Engineering

Subject: Big Data Analytics

PRACTICAL 9

AIM- To understand the working of Pig Functions.

Exercise: You need to perform analysis to reach to conclusions about some datasets your manager has provided you, but your company hosts its data over Cloudera, and you do not understand Java concepts clearly, hence need a way to work with these files in Hadoop.

Tasks:

Apache Pig provides various built-in functions namely **eval, load, store, math, string, bag** and **tuple** functions.

Eval Functions

Given below is the list of **eval** functions provided by Apache Pig.

S.N.	Function & Description
1	AVG() To compute the average of the numerical values within a bag.
2	BagToString() To concatenate the elements of a bag into a string. While concatenating, we can place a delimiter between these values (optional).
3	CONCAT() To concatenate two or more expressions of same type.
4	COUNT() To get the number of elements in a bag, while counting the number of tuples in a bag.
5	COUNT_STAR() It is similar to the **COUNT()** function. It is used to get the number of elements in a bag.
6	DIFF() To compare two bags (fields) in a tuple.
7	IsEmpty()

	To check if a bag or map is empty.
8	MAX() To calculate the highest value for a column (numeric values or chararrays) in a single-column bag.
9	MIN() To get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag.
10	PluckTuple() Using the Pig Latin **PluckTuple()** function, we can define a string Prefix and filter the columns in a relation that begin with the given prefix.
11	SIZE() To compute the number of elements based on any Pig data type.
12	SUBTRACT() To subtract two bags. It takes two bags as inputs and returns a bag which contains the tuples of the first bag that are not in the second bag.
13	SUM() To get the total of the numeric values of a column in a single-column bag.
14	TOKENIZE() To split a string (which contains a group of words) in a single tuple and return a bag which contains the output of the split operation.

AVG()

The Pig-Latin **AVG()** function is used to compute the average of the numerical values within a bag. While calculating the average value, the **AVG()** function ignores the NULL values.

Note −

- To get the global average value, we need to perform a **Group All** operation, and calculate the average value using the **AVG()** function.
- To get the average value of a group, we need to group it using the **Group By** operator and proceed with the average function.

Syntax

Given below is the syntax of the **AVG()** function.

```
grunt> AVG(expression)
```

Example

Assume that we have a file named **student_details.txt** in the HDFS directory **/pig_data/** as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

And we have loaded this file into Pig with the relation name **student_details** as shown below.

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray, gpa:int);
```

Calculating the Average GPA

We can use the built-in function **AVG()** (case-sensitive) to calculate the average of a set of numerical values. Let's group the relation **student_details** using the Group All operator, and store the result in the relation named **student_group_all** as shown below.

```
grunt> student_group_all = Group student_details All;
```

This will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate the global average GPA of all the students using the **AVG()** function as shown below.

```
grunt> student_gpa_avg = foreach student_group_all  Generate
   (student_details.firstname, student_details.gpa),
AVG(student_details.gpa);
```

Verification

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

```
grunt> Dump student_gpa_avg;
```

Output

It will display the contents of the relation **student_gpa_avg** as follows.

```
(({(Bharathi),(Komal),(Archana),(Trupthi),(Preethi),(Rajesh),(sid
darth),(Rajiv) },
  {   (72)   ,   (83) ,   (87)   ,   (75)   ,   (93)   ,  (90)   ,
(78)   ,  (89)  }),83.375)
```

BagToString()

The Pig Latin **BagToString()** function is used to concatenate the elements of a bag into a string. While concatenating, we can place a delimiter between these values (optional).

Generally bags are disordered and can be arranged by using **ORDER BY** operator.

Syntax

Given below is the syntax of the **BagToString()** function.

```
grunt> BagToString(vals:bag [, delimiter:chararray])
```

Example

Assume that we have a file named **dateofbirth.txt** in the HDFS directory **/pig_data/** as shown below. This file contains the date-of-births.

dateofbirth.txt

```
22,3,1990
23,11,1989
1,3,1998
2,6,1980
26,9,1989
```

And we have loaded this file into Pig with the relation name **dob** as shown below.

```
grunt> dob = LOAD
'hdfs://localhost:9000/pig_data/dateofbirth.txt' USING
PigStorage(',')
   as (day:int, month:int, year:int);
```

Converting Bag to String

Using the **bagtostring()** function, we can convert the data in the bag to string. Let us group the **dob** relation. The group operation will produce a bag containing all the tuples of the relation.

Group the relation **dob** using the **Group All** operator, and store the result in the relation named **group_dob** as shown below.

```
grunt> group_dob = Group dob All;
```

It will produce a relation as shown below.

```
grunt> Dump group_dob;

(all,{(26,9,1989),(2,6,1980),(1,3,1998),(23,11,1989),(22,3,1990)}
)
```

Here, we can observe a bag having all the date-of-births as tuples of it. Now, let's convert the bag to string using the function **BagToString()**.

```
grunt> dob_string = foreach group_dob Generate BagToString(dob);
```

Verification

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

```
grunt> Dump dob_string;
```

Output

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

```
(26_9_1989_2_6_1980_1_3_1998_23_11_1989_22_3_1990)
```

CONCAT()

The **CONCAT()** function of Pig Latin is used to concatenate two or more expressions of the same type.

Syntax

```
grunt> CONCAT (expression, expression, [...expression])
```

Example

Assume that we have a file named **student_details.txt** in the HDFS directory **/pig_data/** as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

And we have loaded this file into Pig with the relation name **student_details** as shown below.

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray, gpa:int);
```

Concatenating Two Strings

We can use the **CONCAT()** function to concatenate two or more expressions. First of all, verify the contents of the **student_details** relation using the Dump operator as shown below.

```
grunt> Dump student_details;

( 1,Rajiv,Reddy,21,9848022337,Hyderabad,89 )
( 2,siddarth,Battacharya,22,9848022338,Kolkata,78 )
( 3,Rajesh,Khanna,22,9848022339,Delhi,90 )
( 4,Preethi,Agarwal,21,9848022330,Pune,93 )
( 5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75 )
( 6,Archana,Mishra,23,9848022335,Chennai,87 )
( 7,Komal,Nayak,24,9848022334,trivendram,83 )
( 8,Bharathi,Nambiayar,24,9848022333,Chennai,72 )
```

And, verify the schema using **describe** operator as shown below.

```
grunt> Describe student_details;

student_details: {id: int, firstname: chararray, lastname:
chararray, age: int,
   phone: chararray, city: chararray, gpa: int}
```

In the above schema, you can observe that the name of the student is represented using two chararray values namely **firstname** and **lastname**. Let us concatinate these two values using the **CONCAT()** function.

```
grunt> student_name_concat = foreach student_details Generate
CONCAT (firstname, lastname);
```

Verification

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

```
grunt> Dump student_name_concat;
```

Output

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

```
(RajivReddy)
(siddarthBattacharya)
(RajeshKhanna)
(PreethiAgarwal)
(TrupthiMohanthy)
(ArchanaMishra)
(KomalNayak)
(BharathiNambiayar)
```

We can also use an optional delimiter between the two expressions as shown below.

```
grunt> CONCAT(firstname, '_',lastname);
```

Now, let us concatenate the first name and last name of the student records in the **student_details** relation by placing **'_'** between them as shown below.

```
grunt> student_name_concat = foreach student_details GENERATE
CONCAT(firstname, '_',lastname);
```

Verification

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

```
grunt> Dump student_name_concat;
```

Output

It will produce the following output, displaying the contents of the relation **student_name_concat** as follows.

```
(Rajiv_Reddy)
(siddarth_Battacharya)
```

```
(Rajesh_Khanna)
(Preethi_Agarwal)
(Trupthi_Mohanthy)
(Archana_Mishra)
(Komal_Nayak)
(Bharathi_Nambiayar)
```

COUNT()

The **COUNT()** function of Pig Latin is used to get the number of elements in a bag. While counting the number of tuples in a bag, the **COUNT()** function ignores (will not count) the tuples having a NULL value in the FIRST FIELD.

Note −

- To get the global count value (total number of tuples in a bag), we need to perform a **Group All** operation, and calculate the count value using the COUNT() function.
- To get the count value of a group (Number of tuples in a group), we need to group it using the Group By operator and proceed with the count function.

Syntax

Given below is the syntax of the **COUNT()** function.

```
grunt> COUNT(expression)
```

Example

Assume that we have a file named **student_details.txt** in the HDFS directory **/pig_data/** as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

And we have loaded this file into Pig with the relation named **student_details** as shown below.

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray, gpa:int);
```

Calculating the Number of Tuples

We can use the built-in function **COUNT()** (case sensitive) to calculate the number of tuples in a relation. Let us group the relation **student_details** using the **Group**

All operator, and store the result in the relation named **student_group_all** as shown below.

```
grunt> student_group_all = Group student_details All;
```

It will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate number of tuples/records in the relation.

```
grunt> student_count = foreach student_group_all  Generate
COUNT(student_details.gpa);
```

Verification

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

```
grunt> Dump student_count;
```

Output

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

```
8
```

COUNT_STAR()

The **COUNT_STAR()** function of Pig Latin is similar to the **COUNT()** function. It is used to get the number of elements in a bag. While counting the elements, the **COUNT_STAR()** function includes the NULL values.

Note −

- To get the global count value (total number of tuples in a bag), we need to perform a **Group All** operation, and calculate the count_star value using the COUNT_STAR() function.
- To get the count value of a group (Number of tuples in a group), we need to group it using the **Group By** operator and proceed with the count_star function.

Syntax

Given below is the syntax of the **COUNT_STAR()** function.

```
grunt> COUNT_STAR(expression)
```

Example

Assume that we have a file named **student_details.txt** in the HDFS directory **/pig_data/** as shown below. This file contains an empty record.

student_details.txt

```
, , , , , , ,
001,Rajiv,Reddy,21,9848022337,Hyderabad,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

And we have loaded this file into Pig with the relation name **student_details** as shown below.

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray, gpa:int);
```

Calculating the Number of Tuples

We can use the built-in function **COUNT_STAR()** to calculate the number of tuples in a relation. Let us group the relation **student_details** using the **Group All** operator, and store the result in the relation named **student_group_all** as shown below.

```
grunt> student_group_all = Group student_details All;
```

It will produce a relation as shown below.

grunt> Dump student_group_all;

```
(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89),
( , , , , , , )})
```

Let us now calculate the number of tuples/records in the relation.

```
grunt> student_count = foreach student_group_all  Generate
COUNT_STAR(student_details.gpa);
```

Verification

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

```
grunt> Dump student_count;
```

Output

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

Since we have used the function **COUNT_STAR()**, it included the null tuple and returned 9.

DIFF()

The **DIFF()** function of Pig Latin is used to compare two bags (fields) in a tuple. It takes two fields of a tuple as input and matches them. If they match, it returns an empty bag. If they do not match, it finds the elements that exist in one field (bag) and not found in the other, and returns these elements by wrapping them within a bag.

Syntax

Given below is the syntax of the **DIFF()** function.

```
grunt> DIFF (expression, expression)
```

Example

Generally the **DIFF()** function compares two bags in a tuple. Given below is its example, here we create two relations, cogroup them, and calculate the difference between them.

Assume that we have two files namely **emp_sales.txt** and **emp_bonus.txt** in the HDFS directory **/pig_data/** as shown below. The **emp_sales.txt** contains the details of the employees of the sales department and the **emp_bonus.txt** contains the employee details who got bonus.

emp_sales.txt

```
1,Robin,22,25000,sales
2,BOB,23,30000,sales
3,Maya,23,25000,sales
4,Sara,25,40000,sales
5,David,23,45000,sales
6,Maggy,22,35000,sales
```

emp_bonus.txt

```
1,Robin,22,25000,sales
2,Jaya,23,20000,admin
3,Maya,23,25000,sales
4,Alia,25,50000,admin
5,David,23,45000,sales
6,Omar,30,30000,admin
```

And we have loaded these files into Pig, with the relation names **emp_sales** and **emp_bonus** respectively.

```
grunt> emp_sales = LOAD
'hdfs://localhost:9000/pig_data/emp_sales.txt' USING
PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);

grunt> emp_bonus = LOAD
'hdfs://localhost:9000/pig_data/emp_bonus.txt' USING
PigStorage(',')
```

```
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);
```

Group the records/tuples of the relations **emp_sales** and **emp_bonus** with the key **sno,** using the COGROUP operator as shown below.

```
grunt> cogroup_data = COGROUP emp_sales by sno, emp_bonus by sno;
```

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

```
grunt> Dump cogroup_data;

(1,{(1,Robin,22,25000,sales)},{(1,Robin,22,25000,sales)})
(2,{(2,BOB,23,30000,sales)},{(2,Jaya,23,20000,admin)})
(3,{(3,Maya,23,25000,sales)},{(3,Maya,23,25000,sales)})
(4,{(4,Sara,25,40000,sales)},{(4,Alia,25,50000,admin)})
(5,{(5,David,23,45000,sales)},{(5,David,23,45000,sales)})
(6,{(6,Maggy,22,35000,sales)},{(6,Omar,30,30000,admin)})
```

Calculating the Difference between Two Relations

Let us now calculate the difference between the two relations using **DIFF()** function and store it in the relation **diff_data** as shown below.

```
grunt> diff_data = FOREACH cogroup_data GENERATE
DIFF(emp_sales,emp_bonus);
```

Verification

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

```
grunt> Dump diff_data;

({})
({(2,BOB,23,30000,sales),(2,Jaya,23,20000,admin)})
({})
({(4,Sara,25,40000,sales),(4,Alia,25,50000,admin)})
({})
({(6,Maggy,22,35000,sales),(6,Omar,30,30000,admin)})
```

The **diff_data** relation will have an empty tuple if the records in **emp_bonus** and **emp_sales** match. In other cases, it will hold tuples from both the relations (tuples that differ).

For example, if you consider the records having **sno** as **1**, then you will find them same in both the relations (**(1,Robin,22,25000,sales), (1,Robin,22,25000,sales)**). Therefore, in the **diff_data** relation, which is the result of **DIFF()** function, you will get an empty tuple for **sno 1**.

IsEmpty()

The **IsEmpty()** function of Pig Latin is used to check if a bag or map is empty.

Syntax

Given below is the syntax of the **IsEmpty()** function.

```
grunt> IsEmpty(expression)
```

Example

Assume that we have two files namely **emp_sales.txt** and **emp_bonus.txt** in the HDFS directory **/pig_data/** as shown below. The **emp_sales.txt** contains the details of the employees of the sales department and the **emp_bonus.txt** contains the employee details who got bonus.

emp_sales.txt

```
1,Robin,22,25000,sales
2,BOB,23,30000,sales
3,Maya,23,25000,sales
4,Sara,25,40000,sales
5,David,23,45000,sales
6,Maggy,22,35000,sales
```

emp_bonus.txt

```
1,Robin,22,25000,sales
2,Jaya,23,20000,admin
3,Maya,23,25000,sales
4,Alia,25,50000,admin
5,David,23,45000,sales
6,Omar,30,30000,admin
```

And we have loaded these files into Pig, with the relation names **emp_sales** and **emp_bonus** respectively, as shown below.

```
grunt> emp_sales = LOAD
'hdfs://localhost:9000/pig_data/emp_sales.txt' USING
PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);

grunt> emp_bonus = LOAD
'hdfs://localhost:9000/pig_data/emp_bonus.txt' USING
PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);
```

Let us now group the records/tuples of the relations **emp_sales** and **emp_bonus** with the key **age**, using the **cogroup** operator as shown below.

```
grunt> cogroup_data = COGROUP emp_sales by age, emp_bonus by age;
```

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

```
grunt> Dump cogroup_data;

(22,{(6,Maggy,22,35000,sales),(1,Robin,22,25000,sales)},
{(1,Robin,22,25000,sales)})
(23,{(5,David,23,45000,sales),(3,Maya,23,25000,sales),(2,BOB,23,3
0000,sales)},

{(5,David,23,45000,sales),(3,Maya,23,25000,sales),(2,Jaya,23,2000
0,admin)})
(25,{(4,Sara,25,40000,sales)},{(4,Alia,25,50000,admin)})
(30,{},{(6,Omar,30,30000,admin)})
```

The COGROUP operator groups the tuples from each relation according to age. Each group depicts a particular age value.

For example, if we consider the 1st tuple of the result, it is grouped by age 22. And it contains two bags, the first bag holds all the tuples from the first relation (student_details in this case) having age 22, and the second bag contains all the tuples from the second relation (employee_details in this case) having age 22. In case a relation doesn't have tuples having the age value 22, it returns an empty bag.

Getting the Groups having Empty Bags

Let's list such empty bags from the **emp_sales** relation in the group using the **IsEmpty()** function.

```
grunt> isempty_data = filter cogroup_data by IsEmpty(emp_sales);
```

Verification

Verify the relation **isempty_data** using the DUMP operator as shown below. The **emp_sales** relation holds the tuples that are not there in the relation **emp_bonus**.

```
grunt> Dump isempty_data;

(30,{},{(6,Omar,30,30000,admin)})
```

MAX()

The Pig Latin **MAX()** function is used to calculate the highest value for a column (numeric values or chararrays) in a single-column bag. While calculating the maximum value, the **Max()** function ignores the NULL values.

Note −

- To get the global maximum value, we need to perform a **Group All** operation, and calculate the maximum value using the MAX() function.
- To get the maximum value of a group, we need to group it using the **Group By** operator and proceed with the maximum function.

Syntax

Given below is the syntax of the **Max()** function.

```
grunt> Max(expression)
```

Example

Assume that we have a file named **student_details.txt** in the HDFS directory **/pig_data/** as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

And we have loaded this file into Pig with the relation name **student_details** as shown below.

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray, gpa:int);
```

Calculating the Maximum GPA

We can use the built-in function **MAX()** (case-sensitive) to calculate the maximum value from a set of given numerical values. Let us group the relation **student_details** using the **Group All** operator, and store the result in the relation named **student_group_all** as shown below.

```
grunt> student_group_all = Group student_details All;
```

This will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate the global maximum of GPA, i.e., maximum among the GPA values of all the students using the **MAX()** function as shown below.

```
grunt> student_gpa_max = foreach student_group_all  Generate
   (student_details.firstname, student_details.gpa),
MAX(student_details.gpa);
```

Verification

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

```
grunt> Dump student_gpa_max;
```

Output

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

```
(({(Bharathi),(Komal),(Archana),(Trupthi),(Preethi),(Rajesh),(sid
darth),(Rajiv) } ,
    {    (72)    ,    (83)   ,     (87)   ,    (75)   ,    (93)   ,
(90)   ,     (78)   ,   (89)    }) ,93)
```

MIN()

The **MIN()** function of Pig Latin is used to get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag. While calculating the minimum value, the **MIN()** function ignores the NULL values.

Note −

- To get the global minimum value, we need to perform a **Group All** operation, and calculate the minimum value using the MIN() function.
- To get the minimum value of a group, we need to group it using the **Group By** operator and proceed with the minimum function.

Syntax

Given below is the syntax of the **MIN()** function.

```
grunt> MIN(expression)
```

Example

Assume that we have a file named **student_details.txt** in the HDFS directory **/pig_data/** as shown below.

student_details.txt

```
001,Rajiv,Reddy,21,9848022337,Hyderabad,89
002,siddarth,Battacharya,22,9848022338,Kolkata,78
003,Rajesh,Khanna,22,9848022339,Delhi,90
004,Preethi,Agarwal,21,9848022330,Pune,93
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar,75
006,Archana,Mishra,23,9848022335,Chennai,87
007,Komal,Nayak,24,9848022334,trivendram,83
008,Bharathi,Nambiayar,24,9848022333,Chennai,72
```

And we have loaded this file into Pig with the relation named **student_details** as shown below.

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray, gpa:int);
```

Calculating the Minimum GPA

We can use the built-in function **MIN()** (case sensitive) to calculate the minimum value from a set of given numerical values. Let us group the relation **student_details** using the **Group All** operator, and store the result in the relation named **student_group_all** as shown below

```
grunt> student_group_all = Group student_details All;
```

It will produce a relation as shown below.

```
grunt> Dump student_group_all;

(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai,72),
(7,Komal,Nayak,24,9848022 334,trivendram,83),
(6,Archana,Mishra,23,9848022335,Chennai,87),
(5,Trupthi,Mohan thy,23,9848022336,Bhuwaneshwar,75),
(4,Preethi,Agarwal,21,9848022330,Pune,93),
(3 ,Rajesh,Khanna,22,9848022339,Delhi,90),
(2,siddarth,Battacharya,22,9848022338,Ko lkata,78),
(1,Rajiv,Reddy,21,9848022337,Hyderabad,89)})
```

Let us now calculate the global minimum of GPA, i.e., minimum among the GPA values of all the students using the **MIN()** function as shown below.

```
grunt> student_gpa_min = foreach student_group_all  Generate
   (student_details.firstname, student_details.gpa),
MIN(student_details.gpa);
```

Verification

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

```
grunt> Dump student_gpa_min;
```

Output

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

```
(({(Bharathi),(Komal),(Archana),(Trupthi),(Preethi),(Rajesh),(sid
darth),(Rajiv) } ,
   {    (72)     ,    (83)   ,      (87)    ,     (75)    ,    (93)    ,
(90)    ,      (78)    ,   (89)     }) ,72)
```

PluckTuple()

After performing operations like join to differentiate the columns of the two schemas, we use the function **PluckTuple()**. To use this function, first of all, we have to define a string Prefix and we have to filter for the columns in a relation that begin with that prefix.

Syntax

Given below is the syntax of the **PluckTuple()** function.

```
DEFINE pluck PluckTuple(expression1)
DEFINE pluck PluckTuple(expression1,expression3)
pluck(expression2)
```

Example

Assume that we have two files namely **emp_sales.txt** and **emp_bonus.txt** in the HDFS directory **/pig_data/**. The **emp_sales.txt** contains the details of the employees of the sales department and the **emp_bonus.txt** contains the employee details who got bonus.

emp_sales.txt

```
1,Robin,22,25000,sales
2,BOB,23,30000,sales
3,Maya,23,25000,sales
4,Sara,25,40000,sales
5,David,23,45000,sales
6,Maggy,22,35000,sales
```

emp_bonus.txt

```
1,Robin,22,25000,sales
2,Jaya,23,20000,admin
3,Maya,23,25000,sales
```

```
4,Alia,25,50000,admin
5,David,23,45000,sales
6,Omar,30,30000,admin
```

And we have loaded these files into Pig, with the relation names **emp_sales** and **emp_bonus** respectively.

```
grunt> emp_sales = LOAD
'hdfs://localhost:9000/pig_data/emp_sales.txt' USING
PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);

grunt> emp_bonus = LOAD
'hdfs://localhost:9000/pig_data/emp_bonus.txt' USING
PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);
```

Join these two relations using the **join** operator as shown below.

```
grunt> join_data = join emp_sales by sno, emp_bonus by sno;
```

Verify the relation **join_data** using the **Dump** operator.

```
grunt> Dump join_data;

(1,Robin,22,25000,sales,1,Robin,22,25000,sales)
(2,BOB,23,30000,sales,2,Jaya,23,20000,admin)
(3,Maya,23,25000,sales,3,Maya,23,25000,sales)
(4,Sara,25,40000,sales,4,Alia,25,50000,admin)
(5,David,23,45000,sales,5,David,23,45000,sales)
(6,Maggy,22,35000,sales,6,Omar,30,30000,admin)
```

Using PluckTuple() Function

Now, define the required expression by which you want to differentiate the columns using **PluckTupe()** function.

```
grunt> DEFINE pluck PluckTuple('a::');
```

Filter the columns in the **join_data** relation as shown below.

```
grunt> data = foreach join_data generate FLATTEN(pluck(*));
```

Describe the relation named **data** as shown below.

```
grunt> Describe data;

data: {emp_sales::sno: int, emp_sales::name: chararray,
emp_sales::age: int,
   emp_sales::salary: int, emp_sales::dept: chararray,
emp_bonus::sno: int,
   emp_bonus::name: chararray, emp_bonus::age: int,
emp_bonus::salary: int,
   emp_bonus::dept: chararray}
```

Since we have defined the expression as **"a::"**, the columns of the **emp_sales** schema are plucked as **emp_sales::column name** and the columns of the **emp_bonus** schema are plucked as **emp_bonus::column name**

SIZE()

The **SIZE()** function of Pig Latin is used to compute the number of elements based on any Pig data type.

Syntax

Given below is the syntax of the **SIZE()** function.

```
grunt> SIZE(expression)
```

The return values vary according to the data types in Apache Pig.

Data type	Value
ng, float, double	these types, the size function returns 1.
array	char array, the size() function returns the number of characters in the array.
rray	bytearray, the size() function returns the number of bytes in the array.
	tuple, the size() function returns number of fields in the tuple.
	bag, the size() function returns number of tuples in the bag.
	map, the size() function returns the number of key/value pairs in the map.

Example

Assume that we have a file named **employee.txt** in the HDFS directory **/pig_data/** as shown below.

employee.txt

```
1,John,2007-01-24,250
2,Ram,2007-05-27,220
3,Jack,2007-05-06,170
3,Jack,2007-04-06,100
4,Jill,2007-04-06,220
5,Zara,2007-06-06,300
5,Zara,2007-02-06,350
```

And we have loaded this file into Pig with the relation name **employee_data** as shown below.

```
grunt> employee_data = LOAD 'hdfs://localhost:9000/pig_data/
employee.txt' USING PigStorage(',')
   as (id:int, name:chararray, workdate:chararray,
daily_typing_pages:int);
```

Calculating the Size of the Type

To calculate the size of the type of a particular column, we can use the **SIZE()** function. Let's calculate the size of the name type as shown below.

```
grunt> size = FOREACH employee_data GENERATE SIZE(name);
```

Verification

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

```
grunt> Dump size;
```

Output

It will produce the following output, displaying the contents of the relation **size** as follows. In the example, we have calculated the size of the **name** column. Since it is of varchar type, the **SIZE()** function gives you the number of characters in the name of each employee.

```
(4)
(3)
(4)
(4)
(4)
(4)
(4)
```

SUBTRACT()

The **SUBTRACT()** function of Pig Latin is used to subtract two bags. It takes two bags as inputs and returns a bag which contains the tuples of the first bag that are not in the second bag.

Syntax

Given below is the syntax of the **SUBTRACT()** function.

```
grunt> SUBTRACT(expression, expression)
```

Example

Assume that we have two files namely **emp_sales.txt** and **emp_bonus.txt** in the HDFS directory **/pig_data/** as shown below. The **emp_sales.txt** contains the details of the employees of the sales department and the **emp_bonus.txt** contains the employee details who got bonus.

emp_sales.txt

```
1,Robin,22,25000,sales
2,BOB,23,30000,sales
3,Maya,23,25000,sales
4,Sara,25,40000,sales
5,David,23,45000,sales
6,Maggy,22,35000,sales
```

emp_bonus.txt

```
1,Robin,22,25000,sales
2,Jaya,23,20000,admin
3,Maya,23,25000,sales
4,Alia,25,50000,admin
5,David,23,45000,sales
```

```
6,Omar,30,30000,admin
```

And we have loaded these files into Pig, with the relation names **emp_sales** and **emp_bonus** respectively.

```
grunt> emp_sales = LOAD
'hdfs://localhost:9000/pig_data/emp_sales.txt' USING
PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);

grunt> emp_bonus = LOAD
'hdfs://localhost:9000/pig_data/emp_bonus.txt' USING
PigStorage(',')
   as (sno:int, name:chararray, age:int, salary:int,
dept:chararray);
```

Let us now group the records/tuples of the relations **emp_sales** and **emp_bonus** with the key **sno**, using the COGROUP operator as shown below.

```
grunt> cogroup_data = COGROUP emp_sales by sno, emp_bonus by sno;
```

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

```
grunt> Dump cogroup_data;

(1,{(1,Robin,22,25000,sales)},{(1,Robin,22,25000,sales)})
(2,{(2,BOB,23,30000,sales)},{(2,Jaya,23,30000,admin)})
(3,{(3,Maya,23,25000,sales)},{(3,Maya,23,25000,sales)})
(4,{(4,Sara,25,40000,sales)},{(4,Alia,25,50000,admin)})
(5,{(5,David,23,45000,sales)},{(5,David,23,45000,sales)})
(6,{(6,Maggy,22,35000,sales)},{(6,Omar,30,30000,admin)})
```

Subtracting One Relation from the Other

Let us now subtract the tuples of **emp_bonus** relation from **emp_sales** relation. The resulting relation holds the tuples of **emp_sales** that are not there in **emp_bonus**.

```
grunt> sub_data = FOREACH cogroup_data GENERATE
SUBTRACT(emp_sales, emp_bonus);
```

Verification

Verify the relation **sub_data** using the DUMP operator as shown below. The **emp_sales** relation holds the tuples that are not there in the relation **emp_bonus**.

```
grunt> Dump sub_data;

({})
({(2,BOB,23,30000,sales)})
({})
({(4,Sara,25,40000,sales)})
({})
({(6,Maggy,22,35000,sales)})
```

In the same way, let us subtract the **emp_sales** relation from **emp_bonus** relation as shown below.

```
grunt> sub_data = FOREACH cogroup_data GENERATE
SUBTRACT(emp_bonus, emp_sales);
```

Verify the contents of the **sub_data** relation using the Dump operator as shown below.

```
grunt> Dump sub_data;

({})
({(2,Jaya,23,20000,admin)})
({})
({(4,Alia,25,50000,admin)})
({})
({(6,Omar,30,30000,admin)})
```

SUM()

You can use the **SUM()** function of Pig Latin to get the total of the numeric values of a column in a single-column bag. While computing the total, the **SUM()** function ignores the NULL values.

Note −

- To get the global sum value, we need to perform a **Group All** operation, and calculate the sum value using the SUM() function.
- To get the sum value of a group, we need to group it using the **Group By** operator and proceed with the sum function.

Syntax

Given below is the syntax of the **SUM()** function.

```
grunt> SUM(expression)
```

Example

Assume that we have a file named **employee.txt** in the HDFS directory **/pig_data/** as shown below.

employee.txt

```
1,John,2007-01-24,250
2,Ram,2007-05-27,220
3,Jack,2007-05-06,170
3,Jack,2007-04-06,100
4,Jill,2007-04-06,220
5,Zara,2007-06-06,300
5,Zara,2007-02-06,350
```

And we have loaded this file into Pig with the relation name **employee_data** as shown below.

```
grunt> employee_data = LOAD 'hdfs://localhost:9000/pig_data/
employee.txt' USING PigStorage(',')
   as (id:int, name:chararray, workdate:chararray,
daily_typing_pages:int);
```

Calculating the Sum of All GPA

To demonstrate the **SUM()** function, let's try to calculate the total number of pages typed daily of all the employees. We can use the Apache Pig's built-in function **SUM()** (case sensitive) to calculate the sum of the numerical values. Let us group the relation employee_data using the **Group All** operator, and store the result in the relation named employee_group as shown below.

```
grunt> employee_group = Group employee_data all;
```

It will produce a relation as shown below.

```
grunt> Dump employee_group;

(all,{(5,Zara,2007-02-06,350),
(5,Zara,2007-06-06,300),
(4,Jill,2007-0406,220),
(3,Jack,2007-04-06,100),
(3,Jack,2007-05-06,170),
(2,Ram,2007-0527,220),
(1,John,2007-01-24,250)})
```

Let us now calculate the global sum of the pages typed daily.

```
grunt> student_workpages_sum = foreach employee_group Generate

(employee_data.name,employee_data.daily_typing_pages),SUM(employe
e_data.daily_typing_pages);
```

Verification

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

```
grunt> Dump student_workpages_sum;
```

Output

It will produce the following output, displaying the contents of the relation **student_workpages_sum** as follows.

```
(({ (Zara), (Zara), (Jill) ,(Jack) , (Jack) , (Ram) , (John) },
{ (350) , (300) , (220) ,(100) , (170)  ,  (220)  , (250)
}),1610)
```

TOKENIZE()

The **TOKENIZE()** function of Pig Latin is used to split a string (which contains a group of words) in a single tuple and returns a bag which contains the output of the split operation.

Syntax

Given below is the syntax of the **TOKENIZE()** function.

```
grunt> TOKENIZE(expression [, 'field_delimiter'])
```

As a delimeter to the **TOKENIZE()** function, we can pass space [ ], double quote [" "], coma [,], parenthesis [()], star [*].

Example

Assume that we have a file named **student_details.txt** in the HDFS directory **/pig_data/** as shown below. This file contains the details of a student like id, name, age and city. If we closely observe, the name of the student includes first and last names separated by space [ ].

student_details.txt

```
001,Rajiv Reddy,21,Hyderabad
002,siddarth Battacharya,22,Kolkata
003,Rajesh Khanna,22,Delhi
004,Preethi Agarwal,21,Pune
005,Trupthi Mohanthy,23,Bhuwaneshwar
006,Archana Mishra,23 ,Chennai
007,Komal Nayak,24,trivendram
008,Bharathi Nambiayar,24,Chennai
```

We have loaded this file into Pig with the relation name **student_details** as shown below.

```
grunt> student_details = LOAD
'hdfs://localhost:9000/pig_data/student_details.txt' USING
PigStorage(',')
   as (id:int, name:chararray, age:int,  city:chararray);
```

Tokenizing a String

We can use the **TOKENIZE()** function to split a string. As an example let us split the name using this function as shown below.

```
grunt> student_name_tokenize = foreach student_details  Generate
TOKENIZE(name);
```

Verification

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

```
grunt> Dump student_name_tokenize;
```

Output

It will produce the following output, displaying the contents of the relation **student_name_tokenize** as follows.

```
({(Rajaiv),(Reddy)})
({(siddarth),(Battacharya)})
({(Rajesh),(Khanna)})
({(Preethi),(Agarwal)})
({(Trupthi),(Mohanthy)})
({(Archana),(Mishra)})
({(Komal),(Nayak)})
({(Bharathi),(Nambiayar)})
```

Other Delimeters

In the same way, including space [], the TOKENIZE() function accepts double quote [" "], coma [,], parenthesis [()], star [*] as delimeters.

Example

Suppose there is a file named **details.txt** with students details like id, name, age, and city. Under the name column this file contains the first name and the last name of the students separated by various delimeters as shown below.

details.txt

```
001,"siddarth""Battacharya",22,Kolkata
002,Rajesh*Khanna,22,Delhi
003,(Preethi)(Agarwal),21,Pune
```

We have loaded this file into Pig with the relation name **details** as shown below.

```
grunt> details = LOAD
'hdfs://localhost:9000/pig_data/details.txt' USING
PigStorage(',')
   as (id:int, name:chararray, age:int,  city:chararray);
```

Now, try to separate the first name and the last name of the students using TOKENIZE() as follows.

```
grunt> tokenize_data = foreach details Generate TOKENIZE(name);
```

On verifying the **tokenize_data** relation using dump operator you will get the following result.

```
grunt> Dump tokenize_data;

({(siddarth),(Battacharya)})
({(Rajesh),(Khanna)})
({(Preethi),(Agarwal)})
```