**Problem Statement**

**Create a sample dataset and implement the below Pig commands on the same dataset.**

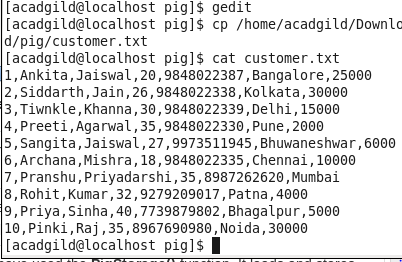
1. **concat**

The concat () function of pig latin is used to concatenate two or more expressions of same type

Syntax

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

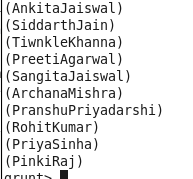
Assume that we have a file named customer.txt with fields like

****

**Performing Concat () on customer.txt dataset**

customer\_concat = foreach customer.txt Generate CONCAT (fname, lname);

**dump customer\_concat;**

****

1. **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 [ \* ].

For example, let us split the above customer.txt using the following command.

**Performing Tokenize () on customer.txt dataset**

**customer\_name\_tokenize = foreach customer.txt Generate TOKENIZE(fname);**

**dump customer\_name\_tokenize;**



1. **Sum**

**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)





1. **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.

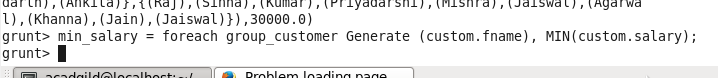
**Note** −

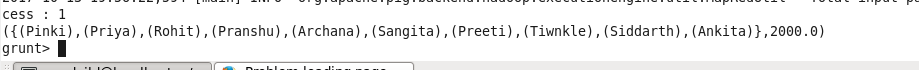
* To get the global minimum value, we need to perform a **Group All**operation, and calculate the minimum value using the MIN() function.
* To get the minimum value of a group, we need to group it using the **Group By** operator and proceed with the minimum function.

## Syntax

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

grunt> MIN(expression)





**5. 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** −

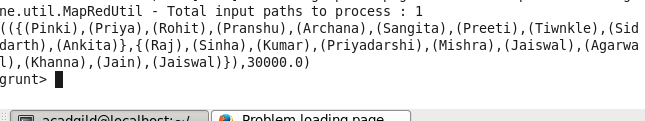
* To get the global maximum value, we need to perform a **Group All**operation, and calculate the maximum value using the MAX() function.
* To get the maximum value of a group, we need to group it using the **Group By** operator and proceed with the maximum function.

## Syntax

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

grunt> Max(expression)

****

****

**6. Limit**

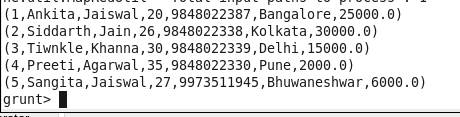
The **LIMIT** operator is used to get a limited number of tuples from a relation.

## Syntax

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

grunt> Result = LIMIT Relation\_name required number of tuples;



****

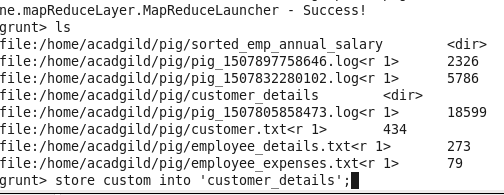
**7. Store**

We can store the loaded data in the file system using the store operator.

## Syntax

Given below is the syntax of the Store statement.

STORE Relation\_name INTO ' required\_directory\_path ' [USING function];



**8. Distinct**

The **DISTINCT** operator is used to remove redundant (duplicate) tuples from a relation.

## Syntax

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

grunt> Relation\_name2 = DISTINCT Relatin\_name1;

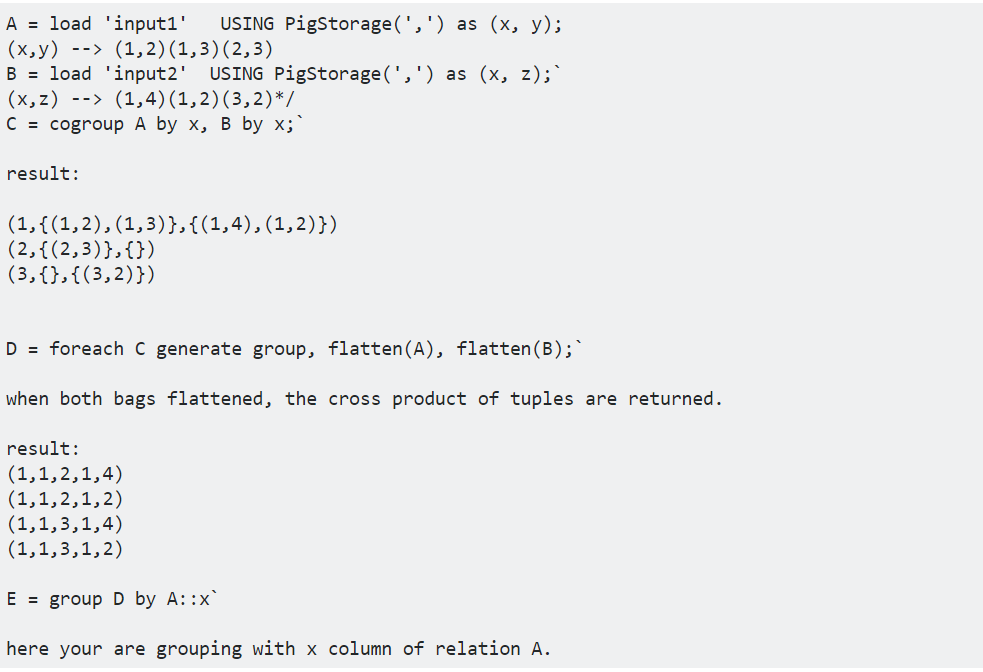


**Since there are no duplicate entries in the relation all the rows/entries are displayed.**

****

**9. Flatten**

The FLATTEN operator looks like a UDF syntactically, but it is an operator that changes the structure of tuples and bags in a way that a UDF cannot. Flatten un-nests tuples as well as bags. The idea is the same, but the operation and result is different for each type of structure



**10. IsEmpty**

The IsEmpty() function of pig latin is used to check if a bag or map is empty.

**Syntax:**

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,23,30000,sales)},

{(5,David,23,45000,sales),(3,Maya,23,25000,sales),(2,Jaya,23,20000,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)})