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

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

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)

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)})

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);

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),(siddarth),(Rajiv) } ,

{ (72) , (83) , (87) , (75) , (93) , (90) , (78)

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),(siddarth),(Rajiv) } ,

{ (72) , (83) , (87) , (75) , (93) , (90) , (78) ,

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;

## 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

002,siddarth,Battacharya,22,9848022338,Kolkata

003,Rajesh,Khanna,22,9848022339,Delhi

004,Preethi,Agarwal,21,9848022330,Pune

005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar

006,Archana,Mishra,23,9848022335,Chennai

007,Komal,Nayak,24,9848022334,trivendram

008,Bharathi,Nambiayar,24,9848022333,Chennai

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);

Now, let’s sort the relation in descending order based on the age of the student and store it into another relation named **limit\_data** using the **ORDER BY** operator as shown below.

grunt> limit\_data = LIMIT student\_details 4;

### Verification

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

grunt> Dump limit\_data;

### Output

It will produce the following output, displaying the contents of the relation **limit\_data** as follows.

(1,Rajiv,Reddy,21,9848022337,Hyderabad)

(2,siddarth,Battacharya,22,9848022338,Kolkata)

(3,Rajesh,Khanna,22,9848022339,Delhi)

(4,Preethi,Agarwal,21,9848022330,Pune)

In the previous chapter, we learnt how to load data into Apache Pig. You can store the loaded data in the file system using the **store** operator. This chapter explains how to store data in Apache Pig using the **Store** operator.

## Syntax

Given below is the syntax of the Store statement.

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

## Example

Assume we have a file **student\_data.txt** in HDFS with the following content.

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.

And we have read it into a relation **student** using the LOAD operator as shown below.

grunt> student = LOAD 'hdfs://localhost:9000/pig\_data/student\_data.txt'

USING PigStorage(',')

as ( id:int, firstname:chararray, lastname:chararray, phone:chararray,

city:chararray );

Now, let us store the relation in the HDFS directory **“/pig\_Output/”** as shown below.

grunt> STORE student INTO ' hdfs://localhost:9000/pig\_Output/ ' USING PigStorage (',');

### Output

After executing the **store** statement, you will get the following output. A directory is created with the specified name and the data will be stored in it.

2015-10-05 13:05:05,429 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.

MapReduceLau ncher - 100% complete

2015-10-05 13:05:05,429 [main] INFO org.apache.pig.tools.pigstats.mapreduce.SimplePigStats -

Script Statistics:

HadoopVersion PigVersion UserId StartedAt FinishedAt Features

2.6.0 0.15.0 Hadoop 2015-10-0 13:03:03 2015-10-05 13:05:05 UNKNOWN

Success!

Job Stats (time in seconds):

JobId Maps Reduces MaxMapTime MinMapTime AvgMapTime MedianMapTime

job\_14459\_06 1 0 n/a n/a n/a n/a

MaxReduceTime MinReduceTime AvgReduceTime MedianReducetime Alias Feature

0 0 0 0 student MAP\_ONLY

OutPut folder

hdfs://localhost:9000/pig\_Output/

Input(s): Successfully read 0 records from: "hdfs://localhost:9000/pig\_data/student\_data.txt"

Output(s): Successfully stored 0 records in: "hdfs://localhost:9000/pig\_Output"

Counters:

Total records written : 0

Total bytes written : 0

Spillable Memory Manager spill count : 0

Total bags proactively spilled: 0

Total records proactively spilled: 0

Job DAG: job\_1443519499159\_0006

2015-10-05 13:06:06,192 [main] INFO org.apache.pig.backend.hadoop.executionengine

.mapReduceLayer.MapReduceLau ncher - Success!

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;

## 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,9848022337,Hyderabad

002,siddarth,Battacharya,9848022338,Kolkata

002,siddarth,Battacharya,9848022338,Kolkata

003,Rajesh,Khanna,9848022339,Delhi

003,Rajesh,Khanna,9848022339,Delhi

004,Preethi,Agarwal,9848022330,Pune

005,Trupthi,Mohanthy,9848022336,Bhuwaneshwar

006,Archana,Mishra,9848022335,Chennai

006,Archana,Mishra,9848022335,Chennai

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, phone:chararray, city:chararray);

Let us now remove the redundant (duplicate) tuples from the relation named **student\_details** using the **DISTINCT** operator, and store it as another relation named **distinct\_data** as shown below.

grunt> distinct\_data = DISTINCT student\_details;

### Verification

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

grunt> Dump distinct\_data;

### Output

It will produce the following output, displaying the contents of the relation **distinct\_data** as follows.

(1,Rajiv,Reddy,9848022337,Hyderabad)

(2,siddarth,Battacharya,9848022338,Kolkata)

(3,Rajesh,Khanna,9848022339,Delhi)

(4,Preethi,Agarwal,9848022330,Pune)

(5,Trupthi,Mohanthy,9848022336,Bhuwaneshwar)

(6,Archana,Mishra,9848022335,Chennai)

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,23,30000,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)})