Institute of Computer Technology

B. Tech. Computer Science and Engineering

Subject: Big Data Analytics

PRACTICAL 10

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:

The Load and Store functions in Apache Pig are used to determine how the data goes ad comes out of Pig. These functions are used with the load and store operators. Given below is the list of load and store functions available in Pig.

Function & Description

1. PigStorage()

To load and store structured files.

2. TextLoader()

To load unstructured data into Pig.

3. BinStorage()

To load and store data into Pig using machine readable format.

4. Handling Compression

In Pig Latin, we can load and store compressed data.

PigStorage()

The **PigStorage()** function 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.

Syntax

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

```
grunt> PigStorage(field_delimiter)
```

Example

Let us suppose we have a file named **student_data.txt** in the HDFS directory named **/data/** 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.
```

We can load the data using the PigStorage function 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 );
```

In the above example, we have seen that we have used comma **(',')** delimiter. Therefore, we have separated the values of a record using **(,)**.

In the same way, we can use the **PigStorage()** function to store the data in to HDFS directory as shown below.

```
grunt> STORE student INTO ' hdfs://localhost:9000/pig_Output/ '
USING PigStorage (',');
```

This will store the data into the given directory. You can verify the data as shown below.

Verification

You can verify the stored data as shown below. First of all, list out the files in the directory named **pig_output** using **ls** command as shown below.

```
$ hdfs dfs -ls 'hdfs://localhost:9000/pig_Output/'

Found 2 items
rw-r--r- 1 Hadoop supergroup 0 2015-10-05 13:03
hdfs://localhost:9000/pig_Output/_SUCCESS

rw-r--r- 1 Hadoop supergroup 224 2015-10-05 13:03
hdfs://localhost:9000/pig_Output/part-m-00000
```

You can observe that two files were created after executing the **Store** statement.

Then, using the **cat** command, list the contents of the file named **part-m-00000** as shown below.

```
$ hdfs dfs -cat 'hdfs://localhost:9000/pig_Output/part-m-00000'

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
```

TextLoader()

The Pig Latin function **TextLoader()** is a Load function which is used to load unstructured data in UTF-8 format.

Syntax

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

```
grunt> TextLoader()
```

Example

Let us assume there is a file with named **stu_data.txt** in the HDFS directory named **/data/** as shown below.

```
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
```

Now let us load the above file using the **TextLoader()** function.

```
grunt> details = LOAD
'hdfs://localhost:9000/pig_data/stu_data.txt' USING TextLoader();
```

You can verify the loaded data using the Dump operator.

```
grunt> dump details;

(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)
```

BinStorage()

The **BinStorage()** function is used to load and store the data into Pig using machine readable format. **BinStorge()** in Pig is generally used to store temporary data generated between the MapReduce jobs. It supports multiple locations as input.

Syntax

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

```
grunt> BinStorage();
```

Example

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

Stu_data.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
```

Let us load this data into Pig into a relation as shown below.

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

Now, we can **store** this relation into the HDFS directory named **/pig_data/** using the **BinStorage()** function.

```
grunt> STORE student_details INTO
'hdfs://localhost:9000/pig_Output/mydata' USING BinStorage();
```

After executing the above statement, the relation is stored in the given HDFS directory. You can see it using the HDFS **ls command** as shown below.

```
$ hdfs dfs -ls hdfs://localhost:9000/pig_Output/mydata/

Found 2 items
-rw-r--r--   1 Hadoop supergroup        0 2015-10-26 16:58
hdfs://localhost:9000/pig_Output/mydata/_SUCCESS

-rw-r--r--   1 Hadoop supergroup         372 2015-10-26 16:58
hdfs://localhost:9000/pig_Output/mydata/part-m-00000
```

Now, load the data from the file **part-m-00000**.

```
grunt> result = LOAD 'hdfs://localhost:9000/pig_Output/b/part-m-
00000' USING BinStorage();
```

Verify the contents of the relation as shown below

```
grunt> Dump result;

(1,Rajiv_Reddy,21,Hyderabad)
(2,siddarth_Battacharya,22,Kolkata)
(3,Rajesh_Khanna,22,Delhi)
(4,Preethi_Agarwal,21,Pune)
(5,Trupthi_Mohanthy,23,Bhuwaneshwar)
(6,Archana_Mishra,23,Chennai)
(7,Komal_Nayak,24,trivendram)
(8,Bharathi_Nambiayar,24,Chennai)
```

Handling Compression

We can load and store compressed data in Apache Pig using the functions **BinStorage()** and **TextLoader()**.

Example

Assume we have a file named **employee.txt.zip** in the HDFS directory **/pigdata/**. Then, we can load the compressed file into pig as shown below.

```
Using PigStorage:

grunt> data = LOAD
'hdfs://localhost:9000/pig_data/employee.txt.zip' USING
PigStorage(',');

Using TextLoader:

grunt> data = LOAD
'hdfs://localhost:9000/pig_data/employee.txt.zip' USING
TextLoader;
```

In the same way, we can store the compressed files into pig as shown below.

```
Using PigStorage:

grunt> store data INTO 'hdfs://localhost:9000/pig_Output/data.bz'
USING PigStorage(',');
```

Bag & Tuple Functions

Given below is the list of Bag and Tuple functions.

Function & Description

1. TOBAG()

To convert two or more expressions into a bag.

2. TOP()

To get the top N tuples of a relation.

3. TOTUPLE()

To convert one or more expressions into a tuple.

4. TOMAP()

To convert the key-value pairs into a Map.

TOBAG()

The **TOBAG()** function of Pig Latin converts one or more expressions to individual tuples. And these tuples are placed in a bag.

Syntax

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

```
TOBAG(expression [, expression ...])
```

Example

Assume we have a file named **employee_details.txt** in the HDFS directory **/pig_data/**, with the following content.

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
```

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

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

Let us now convert the id, name, age and city, of each employee (record) into a tuple as shown below.

```
tobag = FOREACH emp_data GENERATE TOBAG (id,name,age,city);
```

Verification

You can verify the contents of the **tobag** relation using the **Dump** operator as shown below.

```
grunt> DUMP tobag;

({(1),(Robin),(22),(newyork)})
({(2),(BOB),(23),(Kolkata)})
({(3),(Maya),(23),(Tokyo)})
({(4),(Sara),(25),(London)})
({(5),(David),(23),(Bhuwaneshwar)})
({(6),(Maggy),(22),(Chennai)})
```

TOP()

The **TOP()** function of Pig Latin is used to get the top **N** tuples of a bag. To this function, as inputs, we have to pass a relation, the number of tuples we want, and the column name whose values are being compared. This function will return a bag containing the required columns.

Syntax

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

```
grunt> TOP(topN,column,relation)
```

Example

Assume we have a file named **employee_details.txt** in the HDFS directory **/pig_data/**, with the following content.

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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

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

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

Group the relation **emp_data** by age, and store it in the relation **emp_group**.

```
grunt> emp_group = Group emp_data BY age;
```

Verify the relation **emp_group** using the Dump **operator** as shown below.

```
grunt> Dump emp_group;
```

```
(22,{(12,Kelly,22,Chennai),(7,Robert,22,newyork),(6,Maggy,22,Chen
nai),(1,Robin, 22,newyork)})
(23,{(8,Syam,23,Kolkata),(5,David,23,Bhuwaneshwar),(3,Maya,23,Tok
yo),(2,BOB,23, Kolkata)})
(25,{(11,Stacy,25,Bhuwaneshwar),(10,Saran,25,London),(9,Mary,25,T
okyo),(4,Sara, 25,London)})
```

Now, you can get the top two records of each group arranged in ascending order (**based on id**) as shown below.

```
grunt> data_top = FOREACH emp_group {
   top = TOP(2, 0, emp_data);
   GENERATE top;
}
```

In this example we are retriving the top 2 tuples of a group having greater id. Since we are retriving top 2 tuples basing on the **id**, we are passing the index of the column name id as second parameter of TOP() function.

Verification

You can verify the contents of the **data_top** relation using the **Dump** operator as shown below.

```
grunt> Dump data_top;
```

```
({(7,Robert,22,newyork),(12,Kelly,22,Chennai)})
({(5,David,23,Bhuwaneshwar),(8,Syam,23,Kolkata)})
({(10,Saran,25,London),(11,Stacy,25,Bhuwaneshwar)})
```

TOTUPLE()

The **TOTUPLE()** function is used convert one or more expressions to the data type **tuple**.

Syntax

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

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

Example

Assume we have a file named **employee_details.txt** in the HDFS directory **/pig_data/**, with the following content.

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
```

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

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

Let us now convert the id, name and age of each student (record) into a tuple.

```
grunt> totuple = FOREACH emp_data GENERATE TOTUPLE (id,name,age);
```

Verification

You can verify the contents of the **totuple** schema using the **Dump** operator as shown below.

```
grunt> DUMP totuple;
```

```
((1,Robin,22))
((2,BOB,23))
((3,Maya,23))
((4,Sara,25))
((5,David,23))
((6,Maggy,22))
```

TOMAP()

The **TOMAP()** function of Pig Latin is used to convert the key-value pairs into a Map.

Syntax

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

```
grunt> TOMAP(key-expression, value-expression [, key-expression,
valueexpression ...])
```

Example

Assume we have a file named **employee_details.txt** in the HDFS directory **/pig_data/**, with the following content.

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
```

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

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

Let us now take the name and age of each record as key-value pairs and convert them into map as shown below.

```
grunt> tomap = FOREACH emp_data GENERATE TOMAP(name, age);
```

Verification

You can verify the contents of the **tomap** relation using the **Dump** operator as shown below.

```
grunt> DUMP tomap;

([Robin#22])
([BOB#23])
([Maya#23])
([Sara#25])
([David#23])
([Maggy#22])
```

String Functions

We have the following String functions in Apache Pig.

Functions & Description

1. ENDSWITH(string, testAgainst)

To verify whether a given string ends with a particular substring.

2. STARTSWITH(string, substring)

Accepts two string parameters and verifies whether the first string starts with the second.

3. SUBSTRING(string, startIndex, stopIndex)

Returns a substring from a given string.

4. EqualsIgnoreCase(string1, string2)

To compare two stings ignoring the case.

5. INDEXOF(string, 'character', startIndex)

Returns the first occurrence of a character in a string, searching forward from a start index.

6. LAST_INDEX_OF(expression)

Returns the index of the last occurrence of a character in a string, searching backward from a start index.

7. LCFIRST(expression)

Converts the first character in a string to lower case.

8. UCFIRST(expression)

Returns a string with the first character converted to upper case.

9. UPPER(expression)

UPPER(expression) Returns a string converted to upper case.

10. LOWER(expression)

Converts all characters in a string to lower case.

11. REPLACE(string, ‘oldChar’, ‘newChar’);

To replace existing characters in a string with new characters.

12. STRSPLIT(string, regex, limit)

To split a string around matches of a given regular expression.

13. STRSPLITTOBAG(string, regex, limit)

Similar to the STRSPLIT() function, it splits the string by given delimiter and returns the result in a bag.

14. TRIM(expression)

Returns a copy of a string with leading and trailing whitespaces removed.

15. LTRIM(expression)

Returns a copy of a string with leading whitespaces removed.

16. TRIM(expression)

Returns a copy of a string with trailing whitespaces removed.

ENDSWITH()

This function accepts two String parameters, it is used to verify whether the first string ends with the second string.

Syntax

```
grunt> ENDSWITH(string1, string2)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name age and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Following is an example of **ENDSWITH()** function, in this example we are verifying, weather the name of every employee ends with the character **n**.

```
grunt> emp_endswith = FOREACH emp_data GENERATE
(id,name),ENDSWITH ( name, 'n' );
```

The above statement verifies weather the name of the employee ends with the letter n. Since the names of the employees **Saran** and **Robin** ends with the letter n for these two tuples **ENDSWITH()** function returns the Boolean value **'true'** and for remaining tuples the value will be **'false'**.

The result of the statement will be stored in the relation named **emp_endswith**. Verify the content of the relation **emp_endswith**, using the Dump operator as shown below.

```
grunt> Dump emp_endswith;

((1,Robin),true)
((2,BOB),false)
((3,Maya),false)
((4,Sara),false)
((5,David),false)
((6,Maggy),false)
((7,Robert),false)
((8,Syam),false)
((9,Mary),false)
((10,Saran),true)
((11,Stacy),false)
((12,Kelly),false)
```

STARTSWITH()

This function accepts two string parameters. It verifies whether the first string starts with the second.

Syntax

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

```
grunt> STARTSWITH(string, substring)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Example

Following is an example of the **STARTSWITH()** function. In this example, we have verified whether the names of all the employees start with the substring **"Ro"**.

```
grunt> startswith_data = FOREACH emp_data GENERATE (id,name),
STARTSWITH (name,'Ro');
```

The above statement parses the names of all the employees if any of these names starts with the substring **'Ro'**. Since the names of the employees **'Robin'** and **'Robert'** starts with the substring **'Ro'** for these two tuples the **STARTSWITH()** function returns the Boolean value **'true'** and for remaining tuples the value will be **'false'**.

The result of the statement will be stored in the relation named **startswith_data**. Verify the content of the relation **startswith_data**, using the Dump operator as shown below.

```
grunt> Dump startswith_data;

((1,Robin),true)
((2,BOB),false)
((3,Maya),false)
((4,Sara),false)
((5,David),false)
((6,maggy),false)
((7,Robert),true)
((8,Syam),false)
((9,Mary),false)
((10,Saran),false)
```

```
((11,Stacy),false)
((12,Kelly),false)
```

SUBSTRING()

This function returns a substring from the given string.

Syntax

Given below is the syntax of the **SUBSTRING()** function. This function accepts three parameters one is the column name of the string we want. And the other two are the start and stop indexes of the required substring.

```
grunt> SUBSTRING(string, startIndex, stopIndex)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name age and city.

emp.txt

```
001,Robin,22,newyork
002,Stacy,25,Bhuwaneshwar
003,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Following is an example of the **SUBSTRING()** function. This example fetches the sub strings that starts with 0th letter and ends with 2nd letter from the employee names.

```
grunt> substring_data = FOREACH emp_data GENERATE (id,name),
SUBSTRING (name, 0, 2);
```

The above statement fetches the required substrings from the names of the employees. The result of the statement will be stored in the relation named **substring_data**.

Verify the content of the relation **substring_data**, using the Dump operator as shown below.

```
grunt> Dump substring_data;
```

```
((1,Robin),Rob)
((2,Stacy),Sta)
((3,Kelly),Kel)
```

EqualsIgnoreCase()

The **EqualsIgnoreCase()** function is used to compare two strings and verify whether they are equal. If both are equal this function returns the Boolean value **true** else it returns the value **false**.

Syntax

Given below is the syntax of the function **EqualsIgnoreCase()**

```
grunt> EqualsIgnoreCase(string1, string2)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name age and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Given below is an example of the **EqualsIgnoreCase()** function. In this example we are comparing the names of every employees with the string value **'Robin'**.

```
grunt> equals_data = FOREACH emp_data GENERATE (id,name),
EqualsIgnoreCase(name, 'Robin');
```

The above statement compares the string **"Robin"** (case insensitive) with the names of the employees, if the value matches it returns **true** else it returns **false**. In short, this statement searches the employee record whose name is **'Robin'**

The result of the statement will be stored in the relation named **equals_data**. Verify the content of the relation **equals_data**, using the Dump operator as shown below.

```
grunt>  Dump equals_data;

((1,Robin),true)
((2,BOB),false)
((3,Maya),false)
((4,Sara),false)
((5,David),false)
```

```
((6,Maggy),false)
((7,Robert),false)
((8,Syam),false)
((9,Mary),false)
((10,Saran),false)
((11,Stacy),false)
((12,Kelly),false)
```

INDEXOF()

The **INDEXOF()** function accepts a string value, a character and an index (integer). It returns the first occurrence of the given character in the string, searching forward from the given index.

Syntax

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

```
grunt> INDEXOF(string, 'character', startIndex)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Given below is an example of the **INDEXOF()** function. In this example, we are finding the occurrence of the letter **'r'** in the names of every employee using this function.

```
grunt> indexof_data = FOREACH emp_data GENERATE (id,name),
INDEXOF(name, 'r',0);
```

The above statement parses the name of each employee and returns the index value at which the letter **'r'** occurred for the first time. If the name doesn't contain the letter **'r'** it returns the value **-1**

The result of the statement will be stored in the relation named **indexof_data**. Verify the content of the relation **indexof_data**, using the Dump operator as shown below.

```
grunt> Dump indexof_data;
```

```
((1,Robin),-1)
((2,BOB),-1)
((3,Maya),-1)
((4,Sara),2)
((5,David),-1)
((6,Maggy),-1)
((7,Robert),4)
((8,Syam),-1)
((9,Mary),2)
((10,Saran),2)
((11,Stacy),-1)
((12,Kelly),-1)
```

LAST_INDEX_OF()

The **LAST_INDEX_OF()** function accepts a string value and a character. It returns the last occurrence of the given character in the string, searching backward from the end of the string.

Syntax

Given below is the syntax of the **LAST_INDEX_OF()** function

```
grunt> LAST_INDEX_OF(string, 'character')
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Given below is an example of the **LAST_INDEX_OF()** function. In this example, we are going to find the occurrence of the letter **'g'** from the end, in the names of every employee.

```
grunt> last_index_data = FOREACH emp_data GENERATE (id,name),
LAST_INDEX_OF(name, 'g');
```

The above statement parses the name of each employee from the end and returns the index value at which the letter **'g'** occurred for the first time. If the name doesn't contain the letter **'g'** it returns the value **−1**

The result of the statement will be stored in the relation named **last_index_data**. Verify the content of the relation **last_index_data** using the Dump operator as shown below.

```
grunt> Dump last_index_data;

((1,Robin),-1)
((2,BOB),-1)
((3,Maya),-1)
((4,Sara),-1)
((5,David),-1)
((6,Maggy),3)
((7,Robert),-1)
((8,Syam),-1)
((9,Mary),-1)
((10,Saran),-1)
((11,Stacy),-1)
((12,Kelly),-1)
```

LCFIRST()

This function is used to covert the first character of the given string into lowercase.

Syntax

Following is the syntax of the **LCFIRST()** function.

```
grunt> LCFIRST(expression)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Given below is an example of the **LCFIRST()** function. In this example, we have converted all the first letters of the names of the employees to lowercase.

```
grunt> Lcfirst_data = FOREACH emp_data GENERATE (id,name),
LCFIRST(name);
```

The result of the statement will be stored in the relation named **Lcfirst_data**. Verify the content of the relation **Lcfirst_data**, using the Dump operator as shown below.

```
grunt> Dump Lcfirst_data;
```

```
((1,Robin),robin)
((2,BOB),bob)
((3,Maya),maya)
((4,Sara),sara)
((5,David),david)
((6,Maggy),maggy)
((7,Robert),robert)
((8,Syam),syam)
((9,Mary),mary)
((10,Saran),saran)
((11,Stacy),stacy)
((12,Kelly),kelly)
```

UCFIRST()

This function accepts a string, converts the first letter of it into uppercase, and returns the result.

Syntax

Here is the syntax of the function **UCFIRST()** function.

```
grunt> UCFIRST(expression)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
```

```
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Following is an example of the **UCFIRST()** function. In this example, we are trying to convert the first letters of the names of the cities, to which the employees belong to, to uppercase.

```
grunt> ucfirst_data = FOREACH emp_data GENERATE (id,city),
UCFIRST(city);
```

The result of the statement will be stored in the relation named **ucfirst_data**. Verify the content of the relation **ucfirst_data**, using the Dump operator as shown below.

In our example, the first letter of the name of the city "**newyork**" is in lowercase. After applying UCFIRST() function, it turns into "**NEWYORK**"

```
grunt>Dump ucfirst_data;

((1,newyork),Newyork)
((2,Kolkata),Kolkata)
((3,Tokyo),Tokyo)
((4,London),London)
((5,Bhuwaneshwar),Bhuwaneshwar)
((6,Chennai),Chennai)
((7,newyork),Newyork)
((8,Kolkata),Kolkata)
((9,Tokyo),Tokyo)
((10,London),London)
((11,Bhuwaneshwar),Bhuwaneshwar)
((12,Chennai),Chennai)
```

UPPER()

This function is used to convert all the characters in a string to uppercase.

Syntax

The syntax of the **UPPER()** function is as follows −

```
grunt> UPPER(expression)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/**. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Given below is an example of the **UPPER()** function. In this example, we have converted the names of all the employees to upper case.

```
grunt> upper_data = FOREACH emp_data GENERATE (id,name),
UPPER(name);
```

The above statement converts the names of all the employees to uppercase and returns the result.

The result of the statement will be stored in a relation named **upper_data**. Verify the content of the relation **upper_data**, using the Dump operator as shown below.

```
grunt> Dump upper_data;

((1,Robin),ROBIN)
((2,BOB),BOB)
((3,Maya),MAYA)
((4,Sara),SARA)
((5,David),DAVID)
((6,Maggy),MAGGY)
((7,Robert),ROBERT)
((8,Syam),SYAM)
((9,Mary),MARY)
((10,Saran),SARAN)
((11,Stacy),STACY)
((12,Kelly),KELLY)
```

LOWER()

This function is used to convert all the characters in a string to lowercase.

Syntax

Following is the syntax of the **LOWER()** function.

```
grunt> LOWER(expression)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Given below is an example of the **LOWER()** function. In this example, we have converted the names of all the employees to lowercase.

```
grunt> lower_data = FOREACH emp_data GENERATE (id,name),
LOWER(name);
```

The above statement converts the names of all the employees to lowercase and returns the result.

The result of the statement will be stored in the relation named **lower_data**. Verify the content of the relation **lower_data**, using the Dump operator.

```
grunt> Dump lower_data;

((1,Robin),robin)
((2,BOB),bob)
((3,Maya),maya)
((4,Sara),sara)
((5,David),david)
((6,Maggy),maggy)
((7,Robert),robert)
((8,Syam),syam)
((9,Mary),mary)
((10,Saran),saran)
((11,Stacy),stacy)
((12,Kelly),kelly)
```

REPLACE()

This function is used to replace all the characters in a given string with the new characters.

Syntax

Given below is the syntax of the **REPLACE()** function. This function accepts three parameters, namely,

- **string** − The string that is to be replaced. If we want to replace the string within a relation, we have to pass the column name the string belongs to.
- **regEXP** − Here we have to pass the string/regular expression we want to replace.
- **newChar** − Here we have to pass the new value of the string.

```
grunt> REPLACE(string, 'regExp', 'newChar');
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.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
007,Robert,22,newyork
008,Syam,23,Kolkata
009,Mary,25,Tokyo
010,Saran,25,London
011,Stacy,25,Bhuwaneshwar
012,Kelly,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

```
grunt> emp_data = LOAD 'hdfs://localhost:9000/pig_data/emp1.txt'
USING PigStorage(',')
   as (id:int, name:chararray, age:int, city:chararray);
```

Following is an example of the **REPLACE()** function. In this example, we have replaced the name of the city **Bhuwaneshwar** with a shorter form **Bhuw**.

```
grunt> replace_data = FOREACH emp_data GENERATE
(id,city),REPLACE(city,'Bhuwaneshwar','Bhuw');
```

The above statement replaces the string **'Bhuwaneshwar'** with **'Bhuw'** in the column named **city** in the **emp_data** relation and returns the result. This result is stored in the relation named **replace_data**. Verify the content of the relation **replace_data** using the Dump operator as shown below.

```
grunt> Dump replace_data;

((1,newyork),newyork)
((2,Kolkata),Kolkata)
((3,Tokyo),Tokyo)
((4,London),London)
((5,Bhuwaneshwar),Bhuw)
((6,Chennai),Chennai)
((7,newyork),newyork)
((8,Kolkata),Kolkata)
((9,Tokyo),Tokyo)
((10,London),London)
```

```
((11,Bhuwaneshwar),Bhuw)
((12,Chennai),Chennai)
```

STRSPLIT()

This function is used to split a given string by a given delimiter.

Syntax

The syntax of **STRSPLIT()** is given below. This function accepts a string that is needed to be split, a regular expression, and an integer value specifying the limit (the number of substrings the string should be split). This function parses the string and when it encounters the given regular expression, it splits the string into **n** number of substrings where **n** will be the value passed to **limit**.

```
grunt> STRSPLIT(string, regex, limit)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.txt

```
001,Robin_Smith,22,newyork
002,BOB_Wilson,23,Kolkata
003,Maya_Reddy,23,Tokyo
004,Sara_Jain,25,London
005,David_Miller,23,Bhuwaneshwar
006,Maggy_Moore,22,Chennai
007,Robert_Scott,22,newyork
008,Syam_Ketavarapu,23,Kolkata
009,Mary_Carter,25,Tokyo
010,Saran_Naidu,25,London
011,Stacy_Green,25,Bhuwaneshwar
012,Kelly_Moore,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Following is an example of the **STRSPLIT()** function. If you observe the emp.txt file, you can find that, in the **name** column, we have the names and surnames of the employees separated by the delemeter **'_'**.

In this example, we are trying to split the name and surname of the employees using **STRSPLIT()** function.

```
grunt> strsplit_data = FOREACH emp_data GENERATE (id,name),
STRSPLIT (name,'_',2);
```

The result of the statement will be stored in the relation named strsplit_data. Verify the content of the relation **strsplit_data**, using the Dump operator as shown below.

```
grunt> Dump strsplit_data;
```

```
((1,Robin_Smith),(Robin,Smith))
((2,BOB_Wilson),(BOB,Wilson))
((3,Maya_Reddy),(Maya,Reddy))
((4,Sara_Jain),(Sara,Jain))
((5,David_Miller),(David,Miller))
((6,Maggy_Moore),(Maggy,Moore))
((7,Robert_Scott),(Robert,Scott))
((8,Syam_Ketavarapu),(Syam,Ketavarapu))
((9,Mary_Carter),(Mary,Carter))
((10,Saran_Naidu),(Saran,Naidu))
((11,Stacy_Green),(Stacy,Green))
((12,Kelly_Moore),(Kelly,Moore))
```

STRSPLITTOBAG()

This function is similar to the **STRSPLIT()** function. It splits the string by a given delimiter and returns the result in a bag.

Syntax

The syntax of **STRSPLITTOBAG()** is given below. This function accepts a string that is needed to be split, a regular expression, and an integer value specifying the limit (the number of substrings the string should be split). This function parses the string and when it encounters the given regular expression, it splits the sting into **n** number of substrings where **n** will be the value passed to **limit**.

```
grunt> STRSPLITTOBAG(string, regex, limit)
```

Example

Assume that there is a file named **emp.txt** in the **HDFS** directory **/pig_data/** as shown below. This file contains the employee details such as id, name, age, and city.

emp.txt

```
001,Robin_Smith,22,newyork
002,BOB_Wilson,23,Kolkata
003,Maya_Reddy,23,Tokyo
004,Sara_Jain,25,London
005,David_Miller,23,Bhuwaneshwar
006,Maggy_Moore,22,Chennai
007,Robert_Scott,22,newyork
008,Syam_Ketavarapu,23,Kolkata
009,Mary_Carter,25,Tokyo
010,Saran_Naidu,25,London
011,Stacy_Green,25,Bhuwaneshwar
012,Kelly_Moore,22,Chennai
```

And, we have loaded this file into Pig with a relation named **emp_data** as shown below.

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

Following is an example of the **STRSPLITTOBAG()** function. If you observe the emp.txt file, you can find that, in the **name** column, we have name and surname of the employees separated by the delemeter **"_"**.

In this example we are trying to split the name and surname of the employee, and get the result in a bag using **STRSPLITTOBAG()** function.

```
grunt> strsplittobag_data = FOREACH emp_data GENERATE (id,name),
STRSPLITTOBAG (name,'_',2);
```

The result of the statement will be stored in the relation named **strsplittobag_data**. Verify the content of the relation **strsplittobag_data,** using the Dump operator as shown below.

```
grunt> Dump strsplittobag_data;

((1,Robin_Smith),{(Robin),(Smith)})
((2,BOB_Wilson),{(BOB),(Wilson)})
((3,Maya_Reddy),{(Maya),(Reddy)})
((4,Sara_Jain),{(Sara),(Jain)})
((5,David_Miller),{(David),(Miller)})
((6,Maggy_Moore),{(Maggy),(Moore)})
((7,Robert_Scott),{(Robert),(Scott)})
((8,Syam_Ketavarapu),{(Syam),(Ketavarapu)})
((9,Mary_Carter),{(Mary),(Carter)})
((10,Saran_Naidu),{(Saran),(Naidu)})
((11,Stacy_Green),{(Stacy),(Green)})
((12,Kelly_Moore),{(Kelly),(Moore)})
```

TRIM()

The **TRIM()** function accepts a string and returns its copy after removing the unwanted spaces before and after it.

Syntax

Here is the syntax of the **TRIM()** function.

```
grunt> TRIM(expression)
```

Example

Assume we have some unwanted spaces before and after the names of the employees in the records of the **emp_data** relation.

```
grunt> Dump emp_data;

(1, Robin ,22,newyork)
(2,BOB,23,Kolkata)
(3, Maya ,23,Tokyo)
(4,Sara,25,London)
(5, David ,23,Bhuwaneshwar)
(6,maggy,22,Chennai)
(7,Robert,22,newyork)
(8, Syam ,23,Kolkata)
(9,Mary,25,Tokyo)
```

```
(10, Saran ,25,London)
(11, Stacy,25,Bhuwaneshwar)
(12, Kelly ,22,Chennai)
```

Using the **TRIM()** function, we can remove these heading and tailing spaces from the names, as shown below.

```
grunt> trim_data = FOREACH emp_data GENERATE (id,name),
TRIM(name);
```

The above statement returns the copy of the names by removing the heading and tailing spaces from the names of the employees. The result is stored in the relation named **trim_data**. Verify the result of the relation **trim_data** using the Dump operator as shown below.

```
grunt> Dump trim_data;

((1, Robin ),Robin)
((2,BOB),BOB)
((3, Maya ),Maya)
((4,Sara),Sara)
((5, David ),David)
((6,maggy),maggy)
((7,Robert),Robert)
((8, Syam ),Syam)
((9,Mary),Mary)
((10, Saran ),Saran)
((11, Stacy),Stacy)
((12, Kelly ),Kelly)
```

LTRIM()

The function **LTRIM()** is same as the function **TRIM()**. It removes the unwanted spaces from the left side of the given string (heading spaces).

Syntax

Here is the syntax of the LTRIM() function.

```
grunt> LTRIM(expression)
```

Example

Assume we have some unwanted spaces before and after the names of the employees in the records of the **emp_data** relation.

```
grunt> Dump emp_data;

(1, Robin ,22,newyork)
(2, BOB,23,Kolkata)
(3, Maya ,23,Tokyo)
(4, Sara,25,London)
(5, David ,23,Bhuwaneshwar)
(6, maggy,22,Chennai)
(7, Robert,22,newyork)
(8, Syam ,23,Kolkata)
(9, Mary,25,Tokyo)
(10, Saran ,25,London)
```

```
(11, Stacy,25,Bhuwaneshwar)
(12, Kelly ,22,Chennai)
```

Using the **LTRIM()** function, we can remove the heading spaces from the names as shown below.

```
grunt> ltrim_data = FOREACH emp_data GENERATE (id,name),
LTRIM(name);
```

The above statement returns the copy of the names by removing the heading spaces from the names of the employees. The result is stored in the relation named **ltrim_data**. Verify the result of the relation **ltrim_data** using the Dump operator as shown below.

```
grunt> Dump ltrim_data;

((1, Robin ),Robin )
((2,BOB),BOB)
((3, Maya ),Maya )
((4,Sara),Sara)
((5, David ),David )
((6,maggy),maggy)
((7,Robert),Robert)
((8, Syam ),Syam )
((9,Mary),Mary)
((10, Saran),Saran)
((11, Stacy),Stacy)
((12, Kelly ),Kelly )
```

RTRIM()

The function **RTRIM()** is same as the function **TRIM()**. It removes the unwanted spaces from the right side of a given string (tailing spaces).

Syntax

The syntax of the **RTRIM()** function is as follows −

```
grunt> RTRIM(expression)
```

Example

Assume we have some unwanted spaces before and after the names of the employees in the records of the **emp_data** relation as shown below.

```
grunt> Dump emp_data;

(1, Robin ,22,newyork)
(2, BOB,23,Kolkata)
(3, Maya ,23,Tokyo)
(4, Sara,25,London)
(5, David ,23,Bhuwaneshwar)
(6, maggy,22,Chennai)
(7, Robert,22,newyork)
(8, Syam ,23,Kolkata)
(9, Mary,25,Tokyo)
```

```
(10, Saran ,25,London)
(11, Stacy,25,Bhuwaneshwar)
(12, Kelly ,22,Chennai)
```

Using the **RTRIM()** function, we can remove the heading spaces from the names as shown below

```
grunt> rtrim_data = FOREACH emp_data GENERATE (id,name),
RTRIM(name);
```

The above statement returns the copy of the names by removing the **tailing** spaces from the names of the employees. The result is stored in the relation named **rtrim_data**. Verify the result of the relation **rtrim_data** using the Dump operator as shown below.

```
grunt> Dump rtrim_data;

((1, Robin ), Robin)
((2,BOB),BOB)
((3, Maya ), Maya)
((4,Sara),Sara)
((5, David ), David)
((6,maggy),maggy)
((7,Robert),Robert)
((8, Syam ), Syam)
((9,Mary),Mary)
((10, Saran ), Saran)
((11, Stacy), Stacy)
((12, Kelly ), Kelly)
```