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

Syntax

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

```
grunt> Group_data = GROUP Relation_name BY age;
```

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 Apache 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 us group the records/tuples in the relation by age as shown below.

```
grunt> group_data = GROUP student_details by age;
```

Verification

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

```
grunt> Dump group_data;
```

Output

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

- One is **age**, by which we have grouped the relation.
- The other is a **bag**, which contains the group of tuples, student records with the respective age.

```
(21,{(4,Preethi,Agarwal,21,9848022330,Pune),(1,Rajiv,Reddy,21,9848022337,Hydera
bad)})
```

```
(22,{(3,Rajesh,Khanna,22,9848022339,Delhi),(2,siddarth,Battacharya,22,984802233
8,Kolkata)})
(23,{(6,Archana,Mishra,23,9848022335,Chennai),(5,Trupthi,Mohanthy,23,9848022336
,Bhuwaneshwar)})
(24,{(8,Bharathi,Nambiayar,24,9848022333,Chennai),(7,Komal,Nayak,24,9848022334,
trivendram)})
```

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

```
grunt> Describe group_data;

group_data: {group: int,student_details: {(id: int,firstname: chararray,
               lastname: chararray,age: int,phone: chararray,city: chararray)}}
```

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

```
$ Illustrate group_data;
```

It will produce the following output −

```
------------------------------------------------------------------------------
-----------------
|group_data|  group:int |
student_details:bag{:tuple(id:int,firstname:chararray,lastname:chararray,age:int
,phone:chararray,city:chararray)}|
------------------------------------------------------------------------------
-----------------
|          |     21     | { 4, Preethi, Agarwal, 21, 9848022330, Pune), (1,
Rajiv, Reddy, 21, 9848022337, Hyderabad)}|
|          |     2      |
{(2,siddarth,Battacharya,22,9848022338,Kolkata),(003,Rajesh,Khanna,22,9848022339
,Delhi)}|
------------------------------------------------------------------------------
-----------------
```

Grouping by Multiple Columns

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

```
grunt> group_multiple = GROUP student_details by (age, city);
```

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

```
grunt> Dump group_multiple;

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

Group All

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

```
grunt> group_all = GROUP student_details All;
```

Now, verify the content of the relation **group_all** as shown below.

```
grunt> Dump group_all;
```

```
(all,{(8,Bharathi,Nambiayar,24,9848022333,Chennai),(7,Komal,Nayak,24,9848022334
,trivendram),
(6,Archana,Mishra,23,9848022335,Chennai),(5,Trupthi,Mohanthy,23,9848022336,Bhuw
aneshwar),
(4,Preethi,Agarwal,21,9848022330,Pune),(3,Rajesh,Khanna,22,9848022339,Delhi),
(2,siddarth,Battacharya,22,9848022338,Kolkata),(1,Rajiv,Reddy,21,9848022337,Hyd
erabad)})
```

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

Grouping Two Relations using Cogroup

Assume that we have two files namely **student_details.txt** and **employee_details.txt** in the HDFS directory **/pig_data/** 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
```

employee_details.txt

```
001,Robin,22,newyork
002,BOB,23,Kolkata
003,Maya,23,Tokyo
004,Sara,25,London
005,David,23,Bhuwaneshwar
006,Maggy,22,Chennai
```

And we have loaded these files into Pig with the relation names **student_details** and **employee_details** respectively, 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);
```

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

Now, let us group the records/tuples of the relations **student_details** and **employee_details** with the key age, as shown below.

```
grunt> cogroup_data = COGROUP student_details by age, employee_details by age;
```

Verification

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

```
grunt> Dump cogroup_data;
```

Output

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

```
(21,{(4,Preethi,Agarwal,21,9848022330,Pune),
(1,Rajiv,Reddy,21,9848022337,Hyderabad)},
   {    })
(22,{ (3,Rajesh,Khanna,22,9848022339,Delhi),
(2,siddarth,Battacharya,22,9848022338,Kolkata) },
   { (6,Maggy,22,Chennai),(1,Robin,22,newyork) })
(23,{(6,Archana,Mishra,23,9848022335,Chennai),(5,Trupthi,Mohanthy,23,9848022336
,Bhuwaneshwar)},
   {(5,David,23,Bhuwaneshwar),(3,Maya,23,Tokyo),(2,BOB,23,Kolkata)})
(24,{(8,Bharathi,Nambiayar,24,9848022333,Chennai),(7,Komal,Nayak,24,9848022334,
trivendram)},
   { })
(25,{   },
   {(4,Sara,25,London)})
```

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

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

- the first bag holds all the tuples from the first relation (**student_details** in this case) having age 21, and
- the second bag contains all the tuples from the second relation (**employee_details** in this case) having age 21.

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

The **JOIN** operator is used to combine records from two or more relations. While performing a join operation, we declare one (or a group of) tuple(s) from each relation, as keys. When these keys match, the two particular tuples are matched, else the records are dropped. Joins can be of the following types −

- Self-join
- Inner-join
- Outer-join − left join, right join, and full join

This chapter explains with examples how to use the join operator in Pig Latin. Assume that we have two files namely **customers.txt** and **orders.txt** in the **/pig_data/** directory of HDFS as shown below.

customers.txt

```
1,Ramesh,32,Ahmedabad,2000.00
2,Khilan,25,Delhi,1500.00
3,kaushik,23,Kota,2000.00
4,Chaitali,25,Mumbai,6500.00
5,Hardik,27,Bhopal,8500.00
6,Komal,22,MP,4500.00
7,Muffy,24,Indore,10000.00
```

orders.txt

```
102,2009-10-08 00:00:00,3,3000
100,2009-10-08 00:00:00,3,1500
101,2009-11-20 00:00:00,2,1560
103,2008-05-20 00:00:00,4,2060
```

And we have loaded these two files into Pig with the relations **customers** and **orders** as shown below.

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

grunt> orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING
PigStorage(',')
   as (oid:int, date:chararray, customer_id:int, amount:int);
```

Let us now perform various Join operations on these two relations.

Self - join

Self-join is used to join a table with itself as if the table were two relations, temporarily renaming at least one relation.

Generally, in Apache Pig, to perform self-join, we will load the same data multiple times, under different aliases (names). Therefore let us load the contents of the file **customers.txt** as two tables as shown below.

```
grunt> customers1 = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING
PigStorage(',')
   as (id:int, name:chararray, age:int, address:chararray, salary:int);
```

```
grunt> customers2 = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING
PigStorage(',')
   as (id:int, name:chararray, age:int, address:chararray, salary:int);
```

Syntax

Given below is the syntax of performing **self-join** operation using the **JOIN** operator.

```
grunt> Relation3_name = JOIN Relation1_name BY key, Relation2_name BY key ;
```

Example

Let us perform **self-join** operation on the relation **customers**, by joining the two relations **customers1** and **customers2** as shown below.

```
grunt> customers3 = JOIN customers1 BY id, customers2 BY id;
```

Verification

Verify the relation **customers3** using the **DUMP** operator as shown below.

```
grunt> Dump customers3;
```

Output

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

```
(1,Ramesh,32,Ahmedabad,2000,1,Ramesh,32,Ahmedabad,2000)
(2,Khilan,25,Delhi,1500,2,Khilan,25,Delhi,1500)
(3,kaushik,23,Kota,2000,3,kaushik,23,Kota,2000)
(4,Chaitali,25,Mumbai,6500,4,Chaitali,25,Mumbai,6500)
(5,Hardik,27,Bhopal,8500,5,Hardik,27,Bhopal,8500)
(6,Komal,22,MP,4500,6,Komal,22,MP,4500)
(7,Muffy,24,Indore,10000,7,Muffy,24,Indore,10000)
```

Inner Join

Inner Join is used quite frequently; it is also referred to as **equijoin**. An inner join returns rows when there is a match in both tables.

It creates a new relation by combining column values of two relations (say A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, the column values for each matched pair of rows of A and B are combined into a result row.

Syntax

Here is the syntax of performing **inner join** operation using the **JOIN** operator.

```
grunt> result = JOIN relation1 BY columnname, relation2 BY columnname;
```

Example

Let us perform **inner join** operation on the two relations **customers** and **orders** as shown below.

```
grunt> coustomer_orders = JOIN customers BY id, orders BY customer_id;
```

Verification

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

```
grunt> Dump coustomer_orders;
```

Output

You will get the following output that will the contents of the relation named **coustomer_orders**.

```
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
```

Note −

Outer Join: Unlike inner join, **outer join** returns all the rows from at least one of the relations. An outer join operation is carried out in three ways −

- Left outer join
- Right outer join
- Full outer join

Left Outer Join

The **left outer Join** operation returns all rows from the left table, even if there are no matches in the right relation.

Syntax

Given below is the syntax of performing **left outer join** operation using the **JOIN** operator.

```
grunt> Relation3_name = JOIN Relation1_name BY id LEFT OUTER, Relation2_name BY
customer_id;
```

Example

Let us perform left outer join operation on the two relations customers and orders as shown below.

```
grunt> outer_left = JOIN customers BY id LEFT OUTER, orders BY customer_id;
```

Verification

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

```
grunt> Dump outer_left;
```

Output

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

```
(1,Ramesh,32,Ahmedabad,2000,,,,)
```

```
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
(5,Hardik,27,Bhopal,8500,,,,)
(6,Komal,22,MP,4500,,,,)
(7,Muffy,24,Indore,10000,,,,)
```

Right Outer Join

The **right outer join** operation returns all rows from the right table, even if there are no matches in the left table.

Syntax

Given below is the syntax of performing **right outer join** operation using the **JOIN** operator.

```
grunt> outer_right = JOIN customers BY id RIGHT, orders BY customer_id;
```

Example

Let us perform **right outer join** operation on the two relations **customers** and **orders** as shown below.

```
grunt> outer_right = JOIN customers BY id RIGHT, orders BY customer_id;
```

Verification

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

```
grunt> Dump outer_right
```

Output

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

```
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
```

Full Outer Join

The **full outer join** operation returns rows when there is a match in one of the relations.

Syntax

Given below is the syntax of performing **full outer join** using the **JOIN** operator.

```
grunt> outer_full = JOIN customers BY id FULL OUTER, orders BY customer_id;
```

Example

Let us perform **full outer join** operation on the two relations **customers** and **orders** as shown below.

```
grunt> outer_full = JOIN customers BY id FULL OUTER, orders BY customer_id;
```

Verification

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

```
grun> Dump outer_full;
```

Output

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

```
(1,Ramesh,32,Ahmedabad,2000,,,,)
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
(5,Hardik,27,Bhopal,8500,,,,)
(6,Komal,22,MP,4500,,,,)
(7,Muffy,24,Indore,10000,,,,)
```

Using Multiple Keys

We can perform JOIN operation using multiple keys.

Syntax

Here is how you can perform a JOIN operation on two tables using multiple keys.

```
grunt> Relation3_name = JOIN Relation2_name BY (key1, key2), Relation3_name BY
(key1, key2);
```

Assume that we have two files namely **employee.txt** and **employee_contact.txt** in the **/pig_data/** directory of HDFS as shown below.

employee.txt

```
001,Rajiv,Reddy,21,programmer,003
002,siddarth,Battacharya,22,programmer,003
003,Rajesh,Khanna,22,programmer,003
004,Preethi,Agarwal,21,programmer,003
005,Trupthi,Mohanthy,23,programmer,003
006,Archana,Mishra,23,programmer,003
007,Komal,Nayak,24,teamlead,002
008,Bharathi,Nambiayar,24,manager,001
```

employee_contact.txt

```
001,9848022337,Rajiv@gmail.com,Hyderabad,003
002,9848022338,siddarth@gmail.com,Kolkata,003
003,9848022339,Rajesh@gmail.com,Delhi,003
004,9848022330,Preethi@gmail.com,Pune,003
005,9848022336,Trupthi@gmail.com,Bhuwaneshwar,003
006,9848022335,Archana@gmail.com,Chennai,003
007,9848022334,Komal@gmail.com,trivendram,002
008,9848022333,Bharathi@gmail.com,Chennai,001
```

And we have loaded these two files into Pig with relations **employee** and **employee_contact** as shown below.

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

```
   as (id:int, firstname:chararray, lastname:chararray, age:int,
designation:chararray, jobid:int);

grunt> employee_contact = LOAD
'hdfs://localhost:9000/pig_data/employee_contact.txt' USING PigStorage(',')
   as (id:int, phone:chararray, email:chararray, city:chararray, jobid:int);
```

Now, let us join the contents of these two relations using the **JOIN** operator as shown below.

```
grunt> emp = JOIN employee BY (id,jobid), employee_contact BY (id,jobid);
```

Verification

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

```
grunt> Dump emp;
```

Output

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

```
(1,Rajiv,Reddy,21,programmer,113,1,9848022337,Rajiv@gmail.com,Hyderabad,113)
(2,siddarth,Battacharya,22,programmer,113,2,9848022338,siddarth@gmail.com,Kolka
ta,113)
(3,Rajesh,Khanna,22,programmer,113,3,9848022339,Rajesh@gmail.com,Delhi,113)
(4,Preethi,Agarwal,21,programmer,113,4,9848022330,Preethi@gmail.com,Pune,113)
(5,Trupthi,Mohanthy,23,programmer,113,5,9848022336,Trupthi@gmail.com,Bhuwaneshw
ar,113)
(6,Archana,Mishra,23,programmer,113,6,9848022335,Archana@gmail.com,Chennai,113)
(7,Komal,Nayak,24,teamlead,112,7,9848022334,Komal@gmail.com,trivendram,112)
(8,Bharathi,Nambiayar,24,manager,111,8,9848022333,Bharathi@gmail.com,Chennai,111
)
```

The **CROSS** operator computes the cross-product of two or more relations. This chapter explains with example how to use the cross operator in Pig Latin.

Syntax

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

```
grunt> Relation3_name = CROSS Relation1_name, Relation2_name;
```

Example

Assume that we have two files namely **customers.txt** and **orders.txt** in the **/pig_data/** directory of HDFS as shown below.

customers.txt

```
1,Ramesh,32,Ahmedabad,2000.00
2,Khilan,25,Delhi,1500.00
3,kaushik,23,Kota,2000.00
4,Chaitali,25,Mumbai,6500.00
5,Hardik,27,Bhopal,8500.00
6,Komal,22,MP,4500.00
7,Muffy,24,Indore,10000.00
```

orders.txt

```
102,2009-10-08 00:00:00,3,3000
100,2009-10-08 00:00:00,3,1500
101,2009-11-20 00:00:00,2,1560
103,2008-05-20 00:00:00,4,2060
```

And we have loaded these two files into Pig with the relations **customers** and **orders** as shown below.

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

grunt> orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING
PigStorage(',')
   as (oid:int, date:chararray, customer_id:int, amount:int);
```

Let us now get the cross-product of these two relations using the **cross** operator on these two relations as shown below.

```
grunt> cross_data = CROSS customers, orders;
```

Verification

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

```
grunt> Dump cross_data;
```

Output

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

```
(7,Muffy,24,Indore,10000,103,2008-05-20 00:00:00,4,2060)
(7,Muffy,24,Indore,10000,101,2009-11-20 00:00:00,2,1560)
(7,Muffy,24,Indore,10000,100,2009-10-08 00:00:00,3,1500)
(7,Muffy,24,Indore,10000,102,2009-10-08 00:00:00,3,3000)
(6,Komal,22,MP,4500,103,2008-05-20 00:00:00,4,2060)
(6,Komal,22,MP,4500,101,2009-11-20 00:00:00,2,1560)
(6,Komal,22,MP,4500,100,2009-10-08 00:00:00,3,1500)
(6,Komal,22,MP,4500,102,2009-10-08 00:00:00,3,3000)
(5,Hardik,27,Bhopal,8500,103,2008-05-20 00:00:00,4,2060)
(5,Hardik,27,Bhopal,8500,101,2009-11-20 00:00:00,2,1560)
(5,Hardik,27,Bhopal,8500,100,2009-10-08 00:00:00,3,1500)
(5,Hardik,27,Bhopal,8500,102,2009-10-08 00:00:00,3,3000)
(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)
(4,Chaitali,25,Mumbai,6500,101,2009-20 00:00:00,4,2060)
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(2,Khilan,25,Delhi,1500,100,2009-10-08 00:00:00,3,1500)
(2,Khilan,25,Delhi,1500,102,2009-10-08 00:00:00,3,3000)
(1,Ramesh,32,Ahmedabad,2000,103,2008-05-20 00:00:00,4,2060)
(1,Ramesh,32,Ahmedabad,2000,101,2009-11-20 00:00:00,2,1560)
(1,Ramesh,32,Ahmedabad,2000,100,2009-10-08 00:00:00,3,1500)
(1,Ramesh,32,Ahmedabad,2000,102,2009-10-08 00:00:00,3,3000)-11-20
00:00:00,2,1560)
(4,Chaitali,25,Mumbai,6500,100,2009-10-08 00:00:00,3,1500)
(4,Chaitali,25,Mumbai,6500,102,2009-10-08 00:00:00,3,3000)
(3,kaushik,23,Kota,2000,103,2008-05-20 00:00:00,4,2060)
(3,kaushik,23,Kota,2000,101,2009-11-20 00:00:00,2,1560)
(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)
(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)
(2,Khilan,25,Delhi,1500,103,2008-05-20 00:00:00,4,2060)
```

```
(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)
(2,Khilan,25,Delhi,1500,100,2009-10-08 00:00:00,3,1500)
(2,Khilan,25,Delhi,1500,102,2009-10-08 00:00:00,3,3000)
(1,Ramesh,32,Ahmedabad,2000,103,2008-05-20 00:00:00,4,2060)
(1,Ramesh,32,Ahmedabad,2000,101,2009-11-20 00:00:00,2,1560)
(1,Ramesh,32,Ahmedabad,2000,100,2009-10-08 00:00:00,3,1500)
(1,Ramesh,32,Ahmedabad,2000,102,2009-10-08 00:00:00,3,3000)
```

The **UNION** operator of Pig Latin is used to merge the content of two relations. To perform UNION operation on two relations, their columns and domains must be identical.

Syntax

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

```
grunt> Relation_name3 = UNION Relation_name1, Relation_name2;
```

Example

Assume that we have two files namely **student_data1.txt** and **student_data2.txt** in the **/pig_data/** directory of HDFS as shown below.

Student_data1.txt

```
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.
```

Student_data2.txt

```
7,Komal,Nayak,9848022334,trivendram.
8,Bharathi,Nambiayar,9848022333,Chennai.
```

And we have loaded these two files into Pig with the relations **student1** and **student2** as shown below.

```
grunt> student1 = LOAD 'hdfs://localhost:9000/pig_data/student_data1.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, phone:chararray,
city:chararray);

grunt> student2 = LOAD 'hdfs://localhost:9000/pig_data/student_data2.txt' USING
PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, phone:chararray,
city:chararray);
```

Let us now merge the contents of these two relations using the **UNION** operator as shown below.

```
grunt> student = UNION student1, student2;
```

Verification

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

```
grunt> Dump student;
```

Output

It will display the following output, displaying the contents of the relation **student**.

```
(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)
(7,Komal,Nayak,9848022334,trivendram)
(8,Bharathi,Nambiayar,9848022333,Chennai)
```

The **SPLIT** operator is used to split a relation into two or more relations.

Syntax

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

```
grunt> SPLIT Relation1_name INTO Relation2_name IF (condition1), Relation2_name
(condition2),
```

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.

```
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);
```

Let us now split the relation into two, one listing the employees of age less than 23, and the other listing the employees having the age between 22 and 25.

```
SPLIT student_details into student_details1 if age<23, student_details2 if
(22<age and age>25);
```

Verification

Verify the relations **student_details1** and **student_details2** using the **DUMP** operator as shown below.

```
grunt> Dump student_details1;

grunt> Dump student_details2;
```

Output

It will produce the following output, displaying the contents of the relations **student_details1** and **student_details2** respectively.

```
grunt> Dump student_details1;
(1,Rajiv,Reddy,21,9848022337,Hyderabad)
(2,siddarth,Battacharya,22,9848022338,Kolkata)
(3,Rajesh,Khanna,22,9848022339,Delhi)
(4,Preethi,Agarwal,21,9848022330,Pune)

grunt> Dump student_details2;
(5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar)
(6,Archana,Mishra,23,9848022335,Chennai)
(7,Komal,Nayak,24,9848022334,trivendram)
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
```

The **FILTER** operator is used to select the required tuples from a relation based on a condition.

Syntax

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

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

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);
```

Let us now use the Filter operator to get the details of the students who belong to the city Chennai.

```
filter_data = FILTER student_details BY city == 'Chennai';
```

Verification

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

```
grunt> Dump filter_data;
```

Output

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

```
(6,Archana,Mishra,23,9848022335,Chennai)
(8,Bharathi,Nambiayar,24,9848022333,Chennai)
```