Module – 3

* **Operators:**
  + Combining & splitting – UNION, SPLIT
  + Sorting – ORDER BY, LIMIT
  + Grouping Operator – GROUP, CO-GROUP
  + Joining Operator - JOIN(INNER, SELF JOIN)
* **Pig Latin Built-in functions:**
  + Eval functions (Avg, Max, Min, Sum, Count, Size, Concat, Tokenize)
  + Bag & Tuple Functions
  + String Functions
  + Math Functions
* **Apache Pig - Running Scripts:** 
  + Creating pig script
  + Commenting pig script
  + Executing –running pig script – with/without parameters
  + Sample examples

[cloudera@localhost ~]$ gedit dr1

*Milan,1001,5,apollo,500*

*Jay,1002,10,apollo,500*

*lalit,1003,20,manipal,500*

*Mohit,1004,15,columbia,600*

*Chauhan,1005,30,narayana,550*

*Suraj,1006,25,manipal,650*

[cloudera@localhost ~]$ gedit dr2

*meena,2001,20,rxdx,650*

*leena,2001,15,st johns,450*

*sonam,2002,30,rxdx,600*

[cloudera@localhost ~]$ gedit empy1

*7001,ameena,10,bang*

*7002,amit,20,chennai*

*7003,anand,30,bang*

*7004,alen,15,hyd*

*7005,alester,10,hyd*

*7006,anshul,5,Chennai*

[cloudera@localhost ~]$ gedit pnt1

*101,harinath,5,domlur,1004*

*102,nagarjun,10,varthur,1005*

*103,chirajeevi,20,HAL,1006*

*104,tarun,25,HSR,1004*

*105,prabas,15,marthahalli,1006*

*106,chaitanya,30,belandur,1003*

*107,nani,27,krpuram,1004*

[cloudera@localhost ~]$ pig -x local

grunt> clear

grunt> doc1 = load '/home/cloudera/dr1' using PigStorage(',') as

(name:chararray, id:int, exp:int, hosp:chararray, fees:int);

grunt> dump doc1;

grunt> doc2 = load '/home/cloudera/dr2' using PigStorage(',') as

(name:chararray, id:int, exp:int, hosp:chararray, fees:int);

grunt> dump doc2;

**UNION OPERATOR**

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

grunt> Relation\_name3 = UNION Relation\_name1, Relation\_name2;

***grunt> result = union doc1,doc2;***

**grunt> dump result;**

*(Milan,1001,5,apollo,500)*

*(Jay,1002,10,apollo,500)*

*(lalit,1003,20,manipal,500)*

*(Mohit,1004,15,columbia,600)*

*(Chauhan,1005,30,narayana,550)*

*(Suraj,1006,25,manipal,650)*

*(,,,,)*

*(meena,2001,20,rxdx,650)*

*(leena,2001,15,st johns,450)*

*(sonam,2002,30,rxdx,600)*

**SPLIT OPERATOR**

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), Relation3\_name IF(condition2)

**grunt> split doc1 into senior if exp>15,junior if (exp>5 and exp<=15);**

**grunt> dump senior;**

*(lalit,1003,20,manipal,500)*

*(Chauhan,1005,30,narayana,550)*

*(Suraj,1006,25,manipal,650)*

**grunt> dump junior;**

*(Jay,1002,10,apollo,500)*

*(Mohit,1004,15,columbia,600)*

**ORDER BY OPERATOR**

The **ORDER BY** operator is used to display the contents of a relation in a sorted order based on one or more fields.

## Syntax

Given below is the syntax of the **ORDER BY** operator.

grunt> Relation\_name2 = ORDER Relatin\_name1 BY Field (ASC|DESC);

**grunt> a = order doc1 by name asc;**

**grunt> dump a;**

(Chauhan,1005,30,narayana,550)

(Jay,1002,10,apollo,500)

(Milan,1001,5,apollo,500)

(Mohit,1004,15,columbia,600)

(Suraj,1006,25,manipal,650)

(lalit,1003,20,manipal,500)

**LIMIT OPERATOR**

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

## Syntax

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

**grunt> a = limit doc1 2;**

**grunt> dump a;**

*(Jay,1002,10,apollo,500)*

*(Milan,1001,5,apollo,500)*

**GROUP OPERATOR**

The **GROUP** operator is used to group the data in a relation. It collects the data having the same key.

## Syntax

grunt> Group\_data = GROUP Relation\_name BY age;

**Q) Display the details of the doctors hospital wise.**

**grunt> gr = group doc1 by hosp;**

**grunt> dump gr;**

*(apollo,{(Milan,1001,5,apollo,500),(Jay,1002,10,apollo,500)})*

*(manipal,{(lalit,1003,20,manipal,500),(Suraj,1006,25,manipal,650)})*

*(columbia,{(Mohit,1004,15,columbia,600)})*

*(narayana,{(Chauhan,1005,30,narayana,550)})*

**Grouping By Multiple Columns**

**Q) Display the details of the doctors hospital wise with same fees**

**grunt> a = group doc1 by (hosp,fees);**

**grunt> dump a;**

*((apollo,500),{(Milan,1001,5,apollo,500),(Jay,1002,10,apollo,500)})*

*((manipal,500),{(lalit,1003,20,manipal,500)})*

*((manipal,650),{(Suraj,1006,25,manipal,650)})*

*((columbia,600),{(Mohit,1004,15,columbia,600)})*

*((narayana,550),{(Chauhan,1005,30,narayana,550)})*

**CO-GROUP OPERATOR**

The **COGROUP** operator works more or less in the same way as the [GROUP](https://www.tutorialspoint.com/apache_pig/apache_pig_group_operator.htm) 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.

**grunt> emp1 = load '/home/cloudera/empy1' using PigStorage(',') as**

**(id:int, name:chararray, exp:int, place:chararray);**

**grunt> a = cogroup doc1 by exp,emp1 by exp;**

**grunt> dump a;**

*(5,{(Milan,1001,5,apollo,500)},{(7006,anshul,5,chennai)})*

*(10,{(Jay,1002,10,apollo,500)},{(7001,ameena,10,bang),(7005,alester,10,hyd)})*

*(15,{(Mohit,1004,15,columbia,600)},{(7004,alen,15,hyd)})*

*(20,{(lalit,1003,20,manipal,500)},{(7002,amit,20,chennai)})*

*(25,{(Suraj,1006,25,manipal,650)},{})*

*(30,{(Chauhan,1005,30,narayana,550)},{(7003,anand,30,bang)})*

**JOIN OPERATOR**

The JOIN operator is used to combine records from two or more relations.

**Self-Join**

**Self-join** is used to join a table with itself as if the table were two relations.

grunt> doc = load '/home/cloudera/dr1' using PigStorage(',') as

(name:chararray, id:int, exp:int, hosp:chararray, fees:int);

**grunt> a = join doc by id,doc1 by id;**

**grunt> dump a;**

*(Milan,1001,5,apollo,500,Milan,1001,5,apollo,500)*

*(Jay,1002,10,apollo,500,Jay,1002,10,apollo,500)*

*(lalit,1003,20,manipal,500,lalit,1003,20,manipal,500)*

*(Mohit,1004,15,columbia,600,Mohit,1004,15,columbia,600)*

*(Chauhan,1005,30,narayana,550,Chauhan,1005,30,narayana,550)*

*(Suraj,1006,25,manipal,650,Suraj,1006,25,manipal,650)*

**Inner-Join**

**It is also referred to as equijoin. An inner join returns rows when there is a match in both tables.**

grunt> pat = load '/home/cloudera/pnt1' using PigStorage(',') as (pid:int,name:chararray,age:int,addr:chararray,docid:int);

**Q)display entire details of patient and their corresponding doctor**

**grunt> a = join doc1 by id, pat by docid;**

**grunt> dump a;**

*(lalit,1003,20,manipal,500,106,chaitanya,30,belandur,1003)*

*(Mohit,1004,15,columbia,600,101,harinath,5,domlur,1004)*

*(Mohit,1004,15,columbia,600,104,tarun,25,HSR,1004)*

*(Mohit,1004,15,columbia,600,107,nani,27,krpuram,1004)*

*(Chauhan,1005,30,narayana,550,102,nagarjun,10,varthur,1005)*

*(Suraj,1006,25,manipal,650,103,chirajeevi,20,HAL,1006)*

*(Suraj,1006,25,manipal,650,105,prabas,15,marthahalli,1006)*

**Pig Latin Built-in functions:**

**Eval functions** (Avg, Max, Min, Sum, Count, Size, Concat, Tokenize)

[cloudera@localhost ~]$ gedit dr1

*Milan,1001,5,apollo,500*

*Jay,1002,10,apollo,500*

*lalit,1003,20,manipal,500*

*Mohit,1004,15,columbia,600*

*Chauhan,1005,30,narayana,550*

*Suraj,1006,25,manipal,650*

[cloudera@localhost ~]$ pig -x local

grunt> clear

grunt> doc = load '/home/cloudera/dr1' using PigStorage(',') as

(name:chararray, id:int, exp:int, hosp:chararray, fees:int);

grunt> dump doc;

*(Milan,1001,5,apollo,500)*

*(Jay,1002,10,apollo,500)*

*(lalit,1003,20,manipal,500)*

*(Mohit,1004,15,columbia,600)*

*(Chauhan,1005,30,narayana,550)*

*(Suraj,1006,25,manipal,650)*

## Group All

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

grunt> **group\_all** = GROUP **relation\_name** All;

**grunt> gr = group doc all;**

**grunt> dump gr;**

*(all,{(Milan,1001,5,apollo,500),(Jay,1002,10,Apollo,500),(lalit,1003,20,manipal,500),(Mohit,15,1004,15,),(Chauhan,1005,30,narayana,550),(Suraj,1006,25,manipal,650),(Jay,102,10,apollo,50)})*

**AVG():**To compute the average of the numerical values within a bag.

**Q)Display hospital name, fees and average fees among all the hospital.**

**grunt> result = foreach gr generate doc.hosp,doc.fees,AVG(doc.fees);**

*({(apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},550.0)*

**MAX():**To calculate the highest value for a column in a single-column bag.

**Q)Display hospital name, fees and maximum fees among all the hospital.**

**grunt> result = foreach gr generate doc.hosp,doc.fees,MAX(doc.fees);**

**grunt> dump result;**

*({(apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},650)*

**MIN():**To get the minimum (lowest) value (numeric or chararray) for a certain column in a single-column bag.

**Q)Display hospital name, fees and minimum fees among all the hospital.**

**grunt> result = foreach gr generate doc.hosp,doc.fees,MIN(doc.fees);**

**grunt> dump result;**

*({(apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},500)*

**SUM():**To get the total of the numeric values of a column in a single-column bag.

**Q)Display hospital name, fees and total fees among all the hospital.**

**grunt> result = foreach gr generate doc.hosp,doc.fees,SUM(doc.fees);**

**grunt> dump result;**

*({(apollo),(apollo),(manipal),(columbia),(narayana),(manipal),()},{(500),(500),(500),(600),(550),(650),()},3300)*

**COUNT():** To get the the number of tuples in a bag.

**Q)Display total no:of tuples/rows in relation.**

**grunt> result = foreach gr generate COUNT(doc.id);**

**grunt> dump result;**

*(6)*

**SIZE():** To compute the number of elements based on any Pig data type.

**Q)Display doctor name along with the length of doctor name in each row.**

**grunt> ans = foreach doc generate name,SIZE(name);**

**grunt> dump ans;**

*(Milan,5)*

*(Jay,3)*

*(lalit,5)*

*(Mohit,5)*

*(Chauhan,7)*

*(Suraj,5)*

**CONCAT():** To concatenate two or more expressions of same type.

**grunt> ans = foreach doc generate CONCAT(name,hosp);**

**grunt> dump ans;**

***(Milanapollo)***

***(Jayapollo)***

***(lalitmanipal)***

***(Mohitcolumbia)***

***(Chauhannarayana)***

***(Surajmanipal)***

**Bag & Tuple Functions**

**TUPLE CONSTRUCTION:**

**grunt> a = foreach doc generate name,id,exp;**

**grunt> dump a;**

*(Milan,1001,5)*

*(Jay,1002,10)*

*(lalit,1003,20)*

*(Mohit,1004,15)*

*(Chauhan,1005,30)*

*(Suraj,1006,25)*

**BAG CONSTRUCTION:**

**grunt> a = foreach doc generate {(name,id,exp)},{name,id,exp};**

**grunt> dump a;**

*({(Milan,1001,5)},{(Milan),(1001),(5)})*

*({(Jay,1002,10)},{(Jay),(1002),(10)})*

*({(lalit,1003,20)},{(lalit),(1003),(20)})*

*({(Mohit,1004,15)},{(Mohit),(1004),(15)})*

*({(Chauhan,1005,30)},{(Chauhan),(1005),(30)})*

*({(Suraj,1006,25)},{(Suraj),(1006),(25)})*

**MAP CONSTRUCTION:**

**grunt> a = foreach doc generate [name,exp];**

**grunt> dump a;**

*([Milan#5])*

*([Jay#10])*

*([lalit#20])*

*([Mohit#15])*

*([Chauhan#30])*

*([Suraj#25])*

**STRING BUILT\_IN FUNCTIONS**

**SUBSTRING()**

Returns a substring from a given string.

**Syntax:**

[**SUBSTRING(string, startIndex, ending index+1)**](https://www.tutorialspoint.com/apache_pig/apache_pig_substring.htm)

**grunt> ans = foreach doc generate (id,name),SUBSTRING (name, 0 , 2);**

**grunt> dump ans;**

***((1001,Milan),Mi)***

***((1002,Jay),Ja)***

***((1003,lalit),la)***

***((1004,Mohit),Mo)***

***((1005,Chauhan),Ch)***

***((1006,Suraj),Su)***

**INDEXOF():**Returns the first occurrence of a character in a string, searching forward from a start index.

Syntax:

[**INDEXOF(string, ‘character’, startIndex)**](https://www.tutorialspoint.com/apache_pig/apache_pig_indexof.htm)

**grunt> ans = foreach doc generate (id,name),INDEXOF(name,'a',0);**

**grunt> dump ans;**

*((1001,Malan),3)*

*((1002,Jay),1)*

*((1003,lalit),1)*

*((1004,Mohit),-1)*

*((1005,Chauhan),2)*

*((1006,Suraj),3)*

**LCFIRST():** Converts the first character in a string to lower case.

**Syntax:**

[**LCFIRST(expression)**](https://www.tutorialspoint.com/apache_pig/apache_pig_lcfirst.htm)

**grunt> ans = foreach doc generate (id,name),LCFIRST(name);**

**grunt> dump ans;**

*((1001,Milan),milan)*

*((1002,Jay),jay)*

*((1003,lalit),lalit)*

*((1004,Mohit),mohit)*

*((1005,Chauhan),chauhan)*

*((1006,Suraj),suraj)*

**UCFIRST():** Returns a string with the first character converted to upper case.

**Syntax:**

[**UCFIRST(expression)**](https://www.tutorialspoint.com/apache_pig/apache_pig_ucfirst.htm)

**grunt> ans = foreach doc generate (id,hosp),UCFIRST(hosp);**

**grunt> dump ans;**

*((1001,apollo),Apollo)*

*((1002,apollo),Apollo)*

*((1003,manipal),Manipal)*

*((1004,columbia),Columbia)*

*((1005,narayana),Narayana)*

*((1006,manipal),Manipal)*

**UPPER():**Returns a string converted to upper case

Syntax:

UPPER(expression)

grunt> ans = foreach doc generate (id,name),UPPER(name);

grunt> dump ans;

*((1001,Milan),MILAN)*

*((1002,Jay),JAY)*

*((1003,lalit),LALIT)*

*((1004,Mohit),MOHIT)*

*((1005,Chauhan),CHAUHAN)*

*((1006,Suraj),SURAJ)*

**LOWER():** Converts all characters in a string to lower case.

Synatx:

[**LOWER(expression)**](https://www.tutorialspoint.com/apache_pig/apache_pig_lower.htm)

**grunt> ans = foreach doc generate (id,name),LOWER(name);**

**grunt> dump ans;**

*((1001,Milan),milan)*

*((1002,Jay),jay)*

*((1003,lalit),lalit)*

*((1004,Mohit),mohit)*

*((1005,Chauhan),chauhan)*

*((1006,Suraj),suraj)*

**REPLACE():** To replace existing characters in a string with new characters.

Syntax:

[**REPLACE(string, ‘oldChar’, ‘newChar’);**](https://www.tutorialspoint.com/apache_pig/apache_pig_replace.htm)

**grunt> ans = foreach doc generate (id,hosp),REPLACE(hosp,'apollo','appo');**

**grunt> dump ans;**

*((1001,apollo),appo)*

*((1002,apollo),appo)*

*((1003,manipal),manipal)*

*((1004,columbia),columbia)*

*((1005,narayana),narayana)*

*((1006,manipal),manipal)*

**BUILT\_IN MATH FUNCTIONS**

**$gedit math1.txt**

***5***

***16***

***9***

***2.5***

***2***

***3.5***

***3.14***

***-2.2***

**grunt> mat = load '/home/cloudera/ math1.txt' using PigStorage(',') as**

**(data:float);**

**ABS(): ABSOLUTE VALUE**

To get the absolute value of an expression

**grunt> ans = foreach mat generate data,ABS(data);**

**grunt> dump ans;**

*(5.0,5.0)*

*(16.0,16.0)*

*(9.0,9.0)*

*(2.5,2.5)*

*(2.0,2.0)*

*(3.5,3.5)*

*(3.14,3.14)*

*(-2.2,2.2)*

**CBRT() : cube root**

This function is used to get the cube root of an expression.

**grunt> ans = foreach mat generate data,CBRT(data);**

**grunt> dump ans;**

*(5.0,1.709975946676697)*

*(16.0,2.5198420997897464)*

*(9.0,2.080083823051904)*

*(2.5,1.3572088082974532)*

*(2.0,1.2599210498948732)*

*(3.5,1.5182944859378313)*

*(3.14,1.464344366810533)*

*(-2.2,-1.300591456247907)*

**SQRT() : square root**

To get the positive square root of an expression.

**grunt> ans = foreach mat generate data,SQRT(data);**

**grunt> dump ans;**

*(5.0,2.23606797749979)*

*(16.0,4.0)*

*(9.0,3.0)*

*(2.5,1.5811388300841898)*

*(2.0,1.4142135623730951)*

*(3.5,1.8708286933869707)*

*(3.14,1.7720045442673602)*

*(-2.2,NaN)*

**COS():**

This function is used to get the trigonometric cosine of an expression.

**grunt> ans = foreach mat generate data,COS(data);**

**grunt> dump ans;**

*(5.0,0.28366218546322625)*

*(16.0,-0.9576594803233847)*

*(9.0,-0.9111302618846769)*

*(2.5,-0.8011436155469337)*

*(2.0,-0.4161468365471424)*

*(3.5,-0.9364566872907963)*

*(3.14,-0.99999873189461)*

*(-2.2,-0.5885011558074578)*

**SIN():**

To get the sine of an expression.

**grunt> ans = foreach mat generate data,SIN(data);**

**grunt> dump ans;**

*(5.0,-0.9589242746631385)*

*(16.0,-0.2879033166650653)*

*(9.0,0.4121184852417566)*

*(2.5,0.5984721441039564)*

*(2.0,0.9092974268256817)*

*(3.5,-0.35078322768961984)*

*(3.14,0.0015925480124451862)*

*(-2.2,-0.8084963757576692)*

**TAN():**

To get the trigonometric tangent of an angle.

**grunt> ans = foreach mat generate data,TAN(data);**

**grunt> dump ans;**

*(5.0,-3.380515006246586)*

*(16.0,0.3006322420239034)*

*(9.0,-0.45231565944180985)*

*(2.5,-0.7470222972386603)*

*(2.0,-2.185039863261519)*

*(3.5,0.3745856401585947)*

*(3.14,-0.0015925500319664656)*

*(-2.2,1.37382291908733)*

**CEIL():**

This function is used to get the value of an expression rounded up to the nearest integer.

**grunt> ans = foreach mat generate data,CEIL(data);**

**grunt> dump ans;**

*(5.0,5.0)*

*(16.0,16.0)*

*(9.0,9.0)*

*(2.5,3.0)*

*(2.0,2.0)*

*(3.5,4.0)*

*(3.14,4.0)*

*(-2.2,-2.0)*

**FLOOR():**

To get the value of an expression rounded down to the nearest integer.

**grunt> ans = foreach mat generate data,FLOOR(data);**

**grunt> dump ans;**

*(5.0,5.0)*

*(16.0,16.0)*

*(9.0,9.0)*

*(2.5,2.0)*

*(2.0,2.0)*

*(3.5,3.0)*

*(3.14,3.0)*

*(-2.2,-3.0)*

**ROUND():**

To get the value of an expression rounded to an integer (if the result type is float) or rounded to a long (if the result type is double).

**grunt> ans = foreach mat generate data,ROUND(data);**

**grunt> dump ans;**

*(5.0,5)*

*(16.0,16)*

*(9.0,9)*

*(2.5,3)*

*(2.0,2)*

*(3.5,4)*

*(3.14,3)*

*(-2.2,-2)*

**EXP():**

This function is used to get the Euler’s number e raised to the power of x.

**grunt> ans = foreach mat generate data, EXP(data);**

**grunt> dump ans;**

*(5.0,148.4131591025766)*

*(16.0,8886110.520507872)*

*(9.0,8103.083927575384)*

*(2.5,12.182493960703473)*

*(2.0,7.38905609893065)*

*(3.5,33.11545195869231)*

*(3.14,23.103869282414397)*

*(-2.2,0.1108031530788277)*

**LOG10():**

To get the base 10 logarithm of an expression.

**grunt> ans = foreach mat generate data,LOG10(data);**

**grunt> dump ans;**

*(5.0,0.6989700043360189)*

*(16.0,1.2041199826559248)*

*(9.0,0.9542425094393249)*

*(2.5,0.3979400086720376)*

*(2.0,0.3010299956639812)*

*(3.5,0.5440680443502757)*

*(3.14,0.4969296625825472)*

*(-2.2,NaN)*

**LOG():**

To get the natural logarithm (base e) of an expression.

**grunt> ans = foreach mat generate data,LOG(data);**

**grunt> dump ans;**

*(5.0,1.6094379124341003)*

*(16.0,2.772588722239781)*

*(9.0,2.1972245773362196)*

*(2.5,0.9162907318741551)*

*(2.0,0.6931471805599453)*

*(3.5,1.252762968495368)*

*(3.14,1.1442228333291342)*

*(-2.2,NaN)*