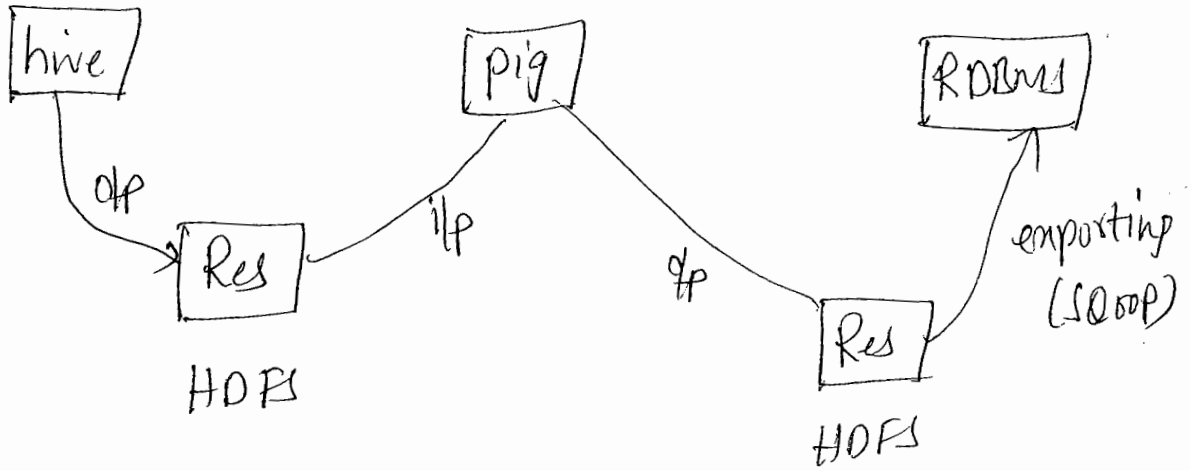
07/02/15

PIG: It is a dataflow language especially designed to process text data (unstructured)

→ The script used in pig is Piglatin pig latin

→ pig takes i/p from HDFS file and write o/p into HDFS files

→ There is no direct interaction b/w pig and other echo systems such as hive, MR, etc but all echo systems run on top of HDFS so one echo systems o/p can be taken as i/p by other echo systems.



→ When you submit pig script pig Compiler will produce java mapReduce code.

→ pig client is called grunt shell.

Ex:-

grunt> lines = load 'psam/pigtest' as (line : chararray);

lines → Alias of Relation
'psam/pigtest' → HDFS filepath
line → field
chararray → datatype

grunt> dump lines; (Map Reduce will be started)

↓

O/P: (Hadoop is big)

(big is not a Hadoop)

grunt> words = foreach lines generate TOKENIZE(lines) as w;

grunt> dump words;

({(Hadoop), (is), (big)})

({(big), (is), (not), (hadoop)})

grunt> NewWords = Foreach words generate FLATTEN(w) as word;

grunt> dump NewWords;

(Hadoop)
(is)
(big)
(big)
(is)
(not)
(Hadoop)

grunt> grp = group newwords by word;

grunt> dump grp;

(Hadoop, {(Hadoop)(Hadoop)}

(is, {(is)(is)}

(big, {(big)(big)})

(not, {(not)})

↓ group ↳ new words

Res = Foreach grp generate group as word, COUNT(newWords) as cnt;

grunt> dump Res;

(Hadoop, 2)

(is, 2)

(big, 2)

(not, 2)

grunt> store Res into 'pigfile';

Result will be stored in HDFS files

/user/training/pigfile/part-r-0000

LOADING INTO PIG RELATION:

Note: Input file should be available in HDFS.

→ While loading pig uses different storage methods such as PigStorage, BinStorage()... etc

→ default storage method is PigStorage()

→ default delimiter for PigStorage is '\t'
↳ octal code for tabspaces

Ex:1

HDFS file

Samp

```
100  200  300
200  300  400
400  500  600
  :
```

```
grunt> A = load "Samp" using PigStorage('\t') as (x:int, y:int, z:int);
grunt> B = load "Samp" using PigStorage() as (x:int, y:int, z:int);
grunt> C = load "Samp" as (x:int, y:int, z:int);
```

from above examples, o/p of A, B, C is same.

```
grunt> dump A/B/C; map Reduce is started
```

Ex:2

Samp2 (HDFS)

```
100, 200, 300
200, 300, 400
```

```
grunt> A = load 'Samp2' using PigStorage(',') as (x:int, y:int, z:int);
     > dump A;
```

Foreach:

→ to filter fields
→ to apply aggregations
→ to apply UDF's (Builtin/custom)
→ to apply expressions
→ to add new fields
→ to change positions of fields.
→ to change data types....

To filter fields:

>emp = load 'emp.txt' using pigstorage(',') as (ecode: chararray, ename :chararray, esal: int, dno: chararray, sex: chararray);

>emp2 = Foreach emp generate ecode, esal;
>emp3 = Foreach emp generate *;
>emp4 = Foreach emp generate $0, $1, $2, .. $n-1; → fieldn, fieldb, fieldi
>emp5 = Foreach emp generate ecode, esal, esal*0.1 as tax;
>emp6 = Foreach emp generate ecode as ec, ename as name, esal as sal;
>emp7 = Foreach emp generate ecode, esal, (int)tax;
>emp8 = Foreach emp generate ecode, ename, dno, sex, esal;

by using illustrate cmd to show schema of Relation

grunt> describe emp;

To show deriving hierarchy of a Relation

grunt> illustrate emp8;

Ex:- a is Rls b/w a, b

y is Rls b/w c, d where c=a, d=a-b

> Foreach a generate a to c, a-b as d;

Note: a mapReduce will started when we apply dump.

grunt> dump emp8;

Generating inner bags:-

When you group a relation inner bags are produced. The following operators are used to group Relation.

Group → for single → emp

Cogroup → for multiple → branch, b2, b3, ...

Applying Aggregations:

All are case sensitive. COUNT(), MAX(), MIN(), SUM(), AVG().

to apply aggregation, data should be grouped first.

Ex:- emp

101, 20000, m

102, 30000, f

103, 40000, m

104, 50000, f

grunt> grp = Group emp by sex ; (or) Group emp by $2 ;

grunt> describe grp;

group → chararray

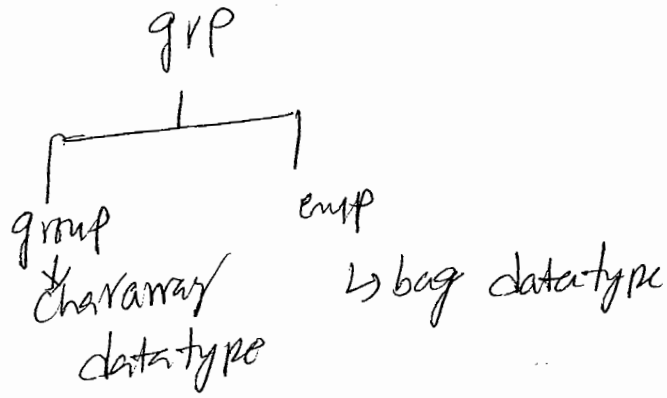
emp → bag

grunt> dump grp;

↓

(f, {(102, 30000, f), (104, 40000, f)})

(m, {(101, 26000, m), (103, 40000, m)})



> Res1 = Foreach grp generate group as sex, COUNT(emp) as cnt;

↓

sex → chararray

cnt → long

> dump Res1; {(f,2), (m,3)}

> STORE Res1 into myDir;

> Res2 = foreach grp generate group as sex, SUM(emp.sal) as totalSalary;

> Res = Foreach grp generate group as sex, COUNT(emp) as cnt, Sum(emp.esal) as totsal, Avg(emp.esal) as Avgsal, MAX(emp.esal) as maxsal, MIN(emp.esal) as minsal;

13/2/13

Double Grouping / multi-field grouping in pig :-

grunt> dgrp = group emp by (dno, sex);

Syntax :-

[Alias = group <inputalias> by (*/fields, field, ...)]

grunt> describe dgrp;

↓

group : (dno : chararray, sex : chararray)

emp : { ecode : chararray, esal : int, dno : chararray, sex : chararray }

grunt> Res = Foreach dgrp generate group as ds, Sum(emp.esal) as tsal;

grunt> dump res;

((11, F), 30000)

((11, M), 25000)

((12, F), 40000)

((12, M), 50000)

↓ ds ↳ tsal

Separating fields from a tuple.

grunt> finalres = foreach res generate ds.dno as dno, ds.sex as sex, tsal;

grunt> dump finalRes;

(11, f, 30000)

(11, m, 25000)

(12, f, 40000)

(12, m, 5000)

grunt> store finalres into 'hdfsdir';

Finding total & Column Aggregation:

grunt> emp1 = foreach emp generate 'x' as f1, esal;

grunt> dump emp1;

x, 10000

x, 20000

x, 30000

grunt> grp = group emp1 by f1;

grunt> dump grp;

(x, {(x, 1000), (x, 2000), (x, 3000)})

grunt> res = foreach grp generate SUM(emp1.esal) as tot;

grunt> dump res;

O/p --> (60000)

In this way we can find aggregation

PIG OPERATORS:

load, store, foreach, group, filter, limit, cogroup, join (inner, outer, right, left), cross join, split

~~Cross~~

Filters: To filter tuples. i.e to separate tuples based on given criteria similar to where clause of SQL.

```
grunt> e1 = filter emp by esal >= 20000;
    > e2 = filter emp by esal <= 40000;
```

Hadoop is good
Hadoop is bad

```
grunt> lines = load 'comments' as (line:chararray);
    > words = foreach lines generate FLATEN (TOKENIZE(line)) as
      word
    > filterwords = group filterwords by word;
    > Res = foreach grpwords generate group as word, COUNT(
                filterwords) as cnt;
    > dump Res;
```

(Hadoop, 3)

(good, 4)

(bad, 3)

Comparison operators in filter clause:

==, !=, >, <, <=, >=

Logical operators in filter clause:

And ⇒ &&

OR ⇒ ||

not ⇒ !

limit: to fetch first 'n' no. of tuples where 'n' is tuples count.

Eg: grunt> x = limit emp by 3;

First 2 females

> f = filter emp by sex == 'f';

> f2 = limit f by 2;

> dump f2;

Select ecode, esal from emp where city = 'hyd'

> x = filter emp by city = 'hyd';

> y = foreach x generate ecode, esal;

ORDER BY: To sort the tuples in ascending order/descending order.

grunt> x = order emp by ename; (default ascending order)

> y = order emp by esal desc;

> z = order emp by esal, dno, sex desc;

Priority: 1) 1-9 3) A-Z

2) a-z

COGROUP. It is for multiple datasets.

branch1 branch2

(101, 20000, 11) (201, 40000, 11)
(102, 30000, 12) (102, 5000, 11)
(103, 40000, 13)
(104, 4000, 11)
(105, 60000, 12)

grunt> cg = cogroup branch1 by dno, branch2 by dno;

> describe cg;

group → chararray

branch1 → {-, -, -}

branch2 → {-, -, -}

> dump cg;

(11, {(101, 20000, 11), (104, 50000, 11)}, {(201, 10000, 11), (202, 2000, 4)})
4 etc

> res1 = foreach cg generate group as dno, COUNT(branch1) as cnt1, COUNT(branch2) as cnt2;

(11, 2, 2)
(12, 2, 1)
(13, 1, 0)
(20, 0, 1)

> Res2 = foreach cg generate group as dno Sum(branch1.esal) as tbl1,
Sum(branch2.esal) as tabl2;

> B1 = foreach branch1 generate 'branch1' as branch, esal;
(branch1, 2000)
(branch1, 30000)

> B2 = foreach branch2 generate 'branch2' as branch, esal;
(branch2, 2000)
(branch2, 4000)

> All = union B1, B2;

> grp = group All by branch;

> Res = foreach grp generate group as branch, COUNT(All) as cnt,
Sum(All.esal) as tsal;

group : chararray
All : {-, -, -}

> dump grp;

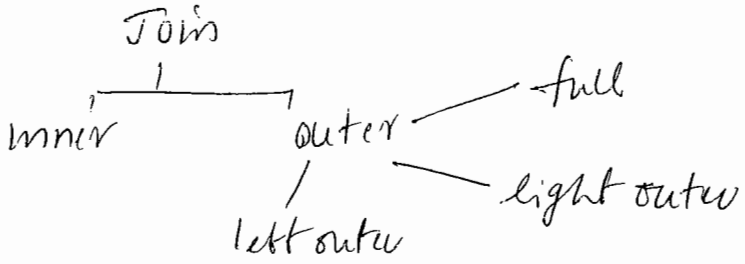
(branch, {(-), (-), (-), (-), (-)}

(branch2, {(-), (-), (-), (-), (-)}

> dump Res;

(branch1, 5, 20000)
(branch2, 4, 20000)

Join: To combine fields of multiple relations to fetch information from multiple tuples



userinfo	userclick
uid, username, location	(uid, url, IP, page)
(u101, Rani, hyd)	(u101, yes, IP, Px)

```
grunt> Jinfo = Join userinfo by uid, userclick by uid;
       P1 = foreach Jinfo generate location, page;
       P2 = group P2 by (location, page);
       res = foreach P2 generate group as LP, COUNT(P1) as cnt;
                                 [Common column is joined by :: operator]
dump res;
```

hyd	Px	3
hyd	Py	4
Del	Px	2
Del	Py	2

Location - no. of clicks

page - no. of clicks

User - no. of clicks

Location - page - no. of clicks

~~20/02/13~~

Top 3 pages from the site, which have been mostly page1 in mostly viewed by which location of page1 in mostly viewed by hyd. list of all the users of hyd.

→ of page1 in viewed by hyd, list all the users of hyd.

→ of page2 is mostly viewed by hyd, next preference page by hyd users.

emp

(101, A, 2000, 11)

(101, B, 13000, 11)

(103, C, 14000, 12)

(104, D, 15000, 13)

(205, D, 16000, 14)

dept

(11, mrkt, hyd)

(12, fin, del)

(17, hr, pune)

(18, prod, pune)

FJ = Join emp by dno FullOuter, Dept by dno.

Dump FJ;

(101, A, 2000, 11, 11, mrkt, hyd)
(102, B, 12000, 11, 11, mrk, hyd)
(103, C, 14000, 12, 12, fin, del)
} → ①

(104, D, 15000, 13, , ,)
(105, E, 16000, 14, , ,) } → ②

(, , , 17, hr, prime)
(, , , 18, prod, prime) } → ①

① ⇒ R1 = Filter FJ by (emp :: dept :: dno);

② ⇒ R2 = Filter FJ by isempty (dept :: dno);

③ ⇒ R3 = Filter FJ by isempty (emp :: dno);

isempty is used to identify NULL in PJ.

is Null
is NotNull } this

not (isempty ())

R4 = Filter FJ by isempty (dept :: dno) (or) isempty (emp :: dno)

free

FJ = Join emp by dno full outer;
" " " left outer; Modifications in Joins
" " " right outer;

FEB 17

CROSS: (cartisian product of pig relations)

ex:- grunt> dump A;

a:int (10,20) → b:int
(30,40)
(50,60)
] 3

grunt> dump B;

x:int
(100, 200)
(300, 400)
] 2
→ y:int

total 3 x 2 = 6-tuples.

grunt> C = CROSS A, B;

grunt> ~~dump C~~; describe C;

{ A: a :int
A : b : int
B : x : int
B = y : int

[:: → Ambiguous operator]

grunt> dump C;

[(10, 20, 100, 200) (50, 60, 100, 200)
(10, 20, 300, 400) (50, 60, 300, 400)
(30, 40, 100, 200)
(30, 40, 300, 400)]

Each element (tuple) of first Relation will be joined with each element (tuple) of 2nd Relation.

Assume

grunt> describe Applicants;

[name: chararray, age: int, sex: chararray, city: chararray, income: int]

grunt> dump Applicants;

(Raj, 23, m, hyd, 20000)

(Rani, 23, f, hyd, 30000)

(Ramu, 24, m, del, 6000)

(venu, 30, m, pune, 8000)

(Radha, 21, f, pune, 75000)

(veni, 27, f, Del, 90000)

grunt> males = Filter Applicants by sex == 'm';

grunt> females = Filter Applicants by sex == 'f';

grunt> dump males;

(Raj, ___)

(Ramu, ___)

(venu, ___)

grunt> dump females;

(Rani, ___)

(Radha, ___)

(veni, ___)

grunt> describe mfcouns;

{ males::name:chararray, males::age:int, males::sex:chararray,
males::city:chararray, males::income:int, females::name:char
array, females::age:int, females::sex:chararray, females::city:chararray,
female::income:int}

grunt dump mfcouns; (Reality for Analytics)

↓

(Raj, ---, Rani, ---)

(Raj, ---, Radha, ---)

(Raj, -.-, -. Vani, ----)

(Ram, ---, Rani, ---)

(Ram, ----, Radha, -,--) total 3x3 = 9-tuples.

(Ram, -,-, --, Vani, -,--)

(Venu, -, -,-, Rani, -, -;-)

(Vani, -, -,-, Radha, -, -,-)

(Venu, -, -, -, Vani, -, -,-)

Ex: list the possible matches for each Applicant based on following criteria.

1) male age should be greater than female age
2) age gap should not cross 3

(iii) Income of male should be greater than to female income.

(iv) Co applicant should not be from same city.

```
grunt> MF = Foreach MFCROSS generate
        Males::name as mname,
        Females::name as fname,
        Males::age as mage,
        Females::age as fage,
        Males::city as mcity,
        Females::city as fcity,
        Males::income as mincome,
        Females::income as fincome,
        Males::age - Females::age as agegap;

grunt> Res = Filter MF by mage > fage and agegap <= 3
        and mincome > fincome and mcity != fcity;
```

SPLIT :-

```
grunt> SPLIT Emp if Sex == 'm' into males else into females;

grunt> Split emp if esal >= 50000 into emp1
        else if esal >= 30000 into emp2
        else if esal >= 40000 into emp3
        else into emp4;
```

Split is not working in current pig version 0.7, It works on 0.8 version of pig.

[We don't have Split cmd so use four filter cmds for above example then follow]

```
grunt> Grd1 = foreach emp1 generate 'A' as Grade, esal;
    > Grd2 = foreach emp2 generate 'B' as Grade, esal;
    > Grd3 = foreach emp3 generate 'C' as Grade, esal;
    > Grd4 = foreach emp4 generate 'D' as Grade, esal;

grunt> All = union Grd1, Grd2, Grd3, Grd4;

grunt> grp = Group All by Grade;

grunt> Res = foreach grp generate group as grade,
                COUNT(All) as cnt, SUM(All.esal) as Tsal;

grunt> dump Res;
        (A, 10, 82)
        (B, 7, 62)
        (C, 3, 75k)
        (D, 4, 8k)
```

Marks

(Name, m1, m2, m3) (pass / fail)

Requirement

DS

(x, y, z)

New)

(AA, –)
(BB, –)
(CC, –)
(DD, –)
(EE, –)
(FF, –)

(AA)
(BB)
(CC)
(DD)
(EE)
(FF)

DS = Foreach DS Generate *, 'x', as grp;

g = group DS1 by grp;

cnt = foreach y Generate COUNT(DS1) as n;

n: long

(last 2 Records getting as follows)

J2 = Filter J by IsEmpty (DS::x)

J3 = Foreach J2 Generate DS::x as x;

(EE)
(FF)

J = Join DS by x left outer, New1 by x;

ds	y
AA	AA
BB	BB
:	:

Built in functions in Pig

SUBSTRING() , SIZE()

INDEXOF()
LASTINDEX()
UPPER()
LOWER()
MONTH()
YEAR()
DAY()
LOG()
LOG10()
:
etc

SUBSTRING() : This is a part of String, return type is Chararray, Used to pick
arg: Start index, end index

INDEXOF : to find index number of char in a given String, Start index given char of its first occurrence

return type is int, arguments: chararray, character

LAST_INDEXOF() : to find the index number of a given char of its last occurrence, arg: chararray, character

return type: int

UPPER() : it converts all lowercase characters into uppercase characters

arguments: Chararray, return type: Chararray

LOWER() : it converts all UPPER case characters into lowercase characters

arguments: Chararray, return type: Chararray.

SIZE() : it count no of characters in string but it won't count left most spaces and right most spaces of String.

Samp

I like it
I Love it
2 Like old one also.

lines = load 'samp' as (line : chararray);

words = foreach lines generate FLATTEN (TOKENIZE(line)) as word;

Newwords = foreach words generate LOWER(word) as word;

grpwords = group newwords by word;

Res = foreach grpwords generate group as word, COUNT(newwords) as cnt;

TOKENIZE → Similar to Java String tokenizer class.

it splits the line into words by treating space as delimiter.

each word is a tuple of a bag (inner)

TOKENIZE ("I love india")

{(I), (love), (india)}

FLATTEN : It converts inner bag tuples into outer bag tuples.

FLATTEN({(I), (love), (INDIA)})

↓

(I)

(love)

(INDIA)

Default Delimiter for TOKENIZER is space, to specify the delimiter given the following

TOKENIZER ("a, ab, abc, abcd", ",");

o/p {(a), (ab), (abc), (abcd)};

COUNT(): It will aggregate the number which count number of tuples in an inner bag.

emp → {(101, 2000), (102, 30000)}

COUNT(emp); → 2

SUBSTRING():

str = computer
0 1 2 3 4 5 6 7

SUBSTRING(str, 3, 6);

Ex:- 10111, 20000, m, hyd
10211, 30000, m, hyd
10312, 40000, F, pune
10412, 50000, F, hyd
ecode dptno.

1) s1 = load 'emp' using pigstorage(',') as (str: chararray, esal: int, sex: chararray, city: chararray);

2) s2 = foreach s1 generate SUBSTRING(str, 0, 3) as ecode, SUBSTRING(str, 3, 5) as dno, esal, sex, city;

101mrkt
102fin
103hr
104mrkt

s1 = load 'emp' using PigStorage(',') as (str: chararray, esal: int, sex = chararray, city = chararray);

s01 = foreach s1 generate *, SIZE(str) as len;

s2 = foreach s01 generate SUBSTRING(str, 0, 3) as ecode, SUBSTRING(str, 3, len) as dname, esal, sex, city;

1011 - 11, -, -, -

102 - 101, -, -, -

10345 - 1001, -, -, -

10 - 11, -, -, -

s1 = Same as above

s01 = foreach s1 generate *, SIZE(str) as len, INDEXOF(str, '-', 0) as h;

final = foreach s01 generate SUBSTRING(str, 0, h) as ecode, SUBSTRING(str, h+1, len) as dno, esal, sex, city;

1011 - 11 - SE, -, -, -

102 - 101 - SSE, -, -, -

10345 - 1001 - PM, -, -, -

10 - 11 - M, -, -, -

s1 = Same as above

s01 = foreach s1 generate *, SIZE(str) as len, INDEXOF(str, '-', 0) as h1, LAST_INDEXOF(str, '-') as h2;

final = foreach s01 generate SUBSTRING (str, 0, h1) as ecode, SUBSTRING (str, h1+1, h2) as dno, SUBSTRING (str, h2+1, len) as desig, esal, sim, city;

CONCAT() : It Concatinate to given String.

20/02/13

UDF

```
public class Mystring extends EvalFunc<String>
{   public String exec( Tuple input)
    {
        String str = input.get(0);
        return str.charAt(0).upper(0) + str.substring(2, str.length ).lower();
    }
}
```

> Register /home/training/Desktop/udf.jar;

> define fconvert ' myorg.String.Analytics.myString(); (pkg name, class name)

> Res = foreach input ds generate name, convert(name) as processedname; (or)

> Dump Res;

Res = foreach input ds generate myorg.String.Analytics.myString(name);

```
public class mystring extends evalfunc<String>
{
    public String exec(Tuple input)
    {
        String str=(String).input.get(0);
    }
    return str.replaceAll("ll","ll");
}
```

x = foreach commands generate user, comment, check(comment)

```
public class mystring extends evalfunc<String>
{
    public Integer exec(Tuple input){
        String str=(String)input.get(0);
        if str.contains("hadoop/hadoop")
            return 1
        else
            return 0;
    }
}
```

PROCESS OF WRITING UDF IN PIG

Program Structure:

=} Importing Java and pig-API classes.

```
public class <className> extends EvalFunc<Arguments>
{
    public <Returntype> evaluate(Tuple <variable name>)
    {
        fln of UDF } }
```

The pig classes are available with pig-core.jar

Location :- /usr/lib/pig/pig-core.jar

The classes used : EvalFunc
Tuple
DataBy

Pkgs : org.apache.hadoop.pig.EvalFunc
org.apache.hadoop.pig.Tuple
org.apache.hadoop.pig.DataBy

EvalFunc<String>
<Integer>

This is based on what type of data-field you are passing from Foreach statement.

Ex:- Foreach genarate myFunc(ename);
↳ chararray
EvalFunc<String>

Foreach emp genarate myfunc(esal);
EvalFunc<Integer>

Public void exec evaluate()
↳ is a null method in EvalFunc class

we need to supply core body with in class argument type is Tuple.

Which takes from Foreach Stat and make it as a Tuple to take a value from tuple variable.

Variable.get(0);

The return type of this method is Object.

you need to parse into String

String str = (String) variable.get(0);

After doing manipulating over the value, now the parsed value is available in str, return the final value.

ex:- return str.upper();

To handle IOExceptions while passing values from Pig to UDF (or) while returning UDF to Pig.

WrapperIOExcepting class is used, it is also available in Pig-core.jar

import org.apache.pig.WrappedIOExceptions.

CREATE A JAR FILE FOR UDF CLASS

Register the jar in pig

Ex: Syntax: grunt> register <jarfilepath>;↵

Ex: grunt> register /home/training/Desktop/mypigUDF.jar↵

define a function name for your UDF class.

Syntax: grunt> define <functionname> <classname>();↵

Ex:- grunt> define test my.pig.udfs.myudf();↵

Calling UDF :-

grunt> A = foreach B generate x,y, test(x,y) as Z; (OR)

grunt> foreach a generate ... mypig.udf.myudf(...) > 2;

Note: this functions are temporary, once you come out of the grunt shell, the pig UDF (custom) are not available.
you need to re-register and define them.

Solutions: keep all the Registration and function definition in one script file (pig) and run the pigscript.

Note: pig script file should have extension .pig

Ex/ myscript.pig

```
register path1/jar1;
register path2/jar2;
register path3/jar3;
define f1 pack1.class1;
define f2 pack2.class2;
define f3 pack3.class3;
```

grunt> run /home/sales/mypigscript.pig ↲

grunt> ≡ } you can use all the UDFs

Similar way hive script can be define.

hive script should saved with .hql extension

To execute the script

$ hive -f 'myhivescript.hql' ↲

```
$ cat > myhivescript.hql >
  add jar '/home/training/jar1;
  add jar '/home/training/jar2;
  create temporary function func1 as 'my.hive.class';
  create temporary function func2 as '   "   ';
```