APACHE PIG-0.16.0

Session1: Introduction to Pig

So far we have discussed Map-reduce, hive, sqoop. Lets discuss Pig.

1. Map-Reduce Concept was developed by google
2. Hive was developed at Facebook.
3. Pig was developed at Yahoo,2008.

Definition:-It is basically a component engine for executing data flow in parallel in hadoop.

1. Its uses its own language called PIG Latin & DAG(Directed Acyclic graph to represent the dataflow).
2. Like we have hive terminal in Hive, we have grunt terminal in Pig.
3. At Yahoo: 50-70 % of big data processing are done using PIG only.

Lets talk Hive:

Lets find the list of all the doctor who gets more salary than the highest salary of any engineer.

hive>select * from employees where desg='doctor' and salary>(select max(salary) from employees where desgn='engineer');

You cannot write the below command

```
HIVE> engineer_salary=select  max(salary) from employee where desg='Engineer';
HIVE> select * from employee where desg='doctor' and salary>engineer_salary;
```

In Pig You can write something similar.Please ignore syntax for now.

A=select max(salary) from employees where desgn='engineer';

B=select * from employees where desg='doctor' and salary>A;

pig->data1->data2->data3

SQL Vs PIG:-

SQL: mention what you want to achieve, How will be taken care by SQL engine.

PIG: User will define the flow

Example: Find out salary of all employees whose name starts with S and they belong to CSE dept

Solution 1: find out all employees from table whose name starts with S, and then filter them if they are from CSE dept.
Solution 2: find out all employees from CSE dept and check if there name starts with S

SQL: several related query should be grouped together, inform of nested query.

PIG: allows user to store the result of one query into a variable, which can then be used later.

Hive > select * from employees where desg='doctor' and salary>(select max(salary) from employees where desg='engineer');

PIG: (Ignore syntax again)

A=select max(salary) from employees where desg='engineer';

B=select * from employees where desg='doctor' and salary>A;

SQL: Is designed for RDBMS environment, where data is normalized with proper constraints.

PIG: is designed for big-data processing, structure as well as unstructured data. Can work on data which has no proper constraints and they are not normalized.

Why Name is PIG:-

1. **Eats anything**: structured as well as unstructured
2. **Lives anywhere**: should work with any file-system with any component
3. **Are Domestic**: easily controlled, UDF's, map-reduced command
4. **Can fly:** light weight component, with great processing power.

Installing Pig

1. **Download Pig from Central Repository.**
   Go to http://apache.mesi.com.ar/ and get the latest PIG and download it to /home/hduser/ecosystem directory or type the below command from hduser directory in Linux.

```
$ mkdir ecosystem --if needed
$ cd ecosystem
$ wget http://apache.mesi.com.ar/pig/pig-0.16.0/pig-0.16.0.tar.gz
```

2. unzip it & create a symbolic link

```
$ tar -xvf pig-0.16.0.tar.gz
$ ln -s pig-0.16.0   pig
```

3. Update PIG path in .bashrc

```
$vim /home/hduser/.bashrc
```

Scroll down till the end. Press I to get into Insert Mode.

Enter the below 2 statement in the new lines

```
export PIG_HOME=/home/hduser/ecosystem/pig
export PATH=$PATH:$PIG_HOME/bin
```

Press **Esc :wq** to write and quit the terminal

4. Start History server.

If you are using Hadoop 2, you need to start History server as well, as pig needs history server. If you are using hadoop 1.2.1, you don't need this step.

```
$mr-jobhistory-daemon.sh start historyserver
$mr-jobhistory-daemon.sh stop historyserver  --If you want to stop
```

5. Re launch your terminal, and type pig to get grunt terminal

```
$pig
grunt>
```

Working with basic pig

copy excite-small.log from /home/hduser/ecosystem/pig/tutorial/data/excite-small.log to hdfs

```
$ hadoop fs -mkdir /user/hduser/data/
$ hadoop fs -put /home/hduser/ecosystem/pig/tutorial/data/excite-small.log /user/hduser/data/excite-small.log
```

You can browse the file using the url http://localhost:50070

Launch Pig and execute the below commands

```
$pig
grunt>log = LOAD '/user/hduser/data/excite-small.log' AS (user, time, query);
grunt>lmt = LIMIT log 4;
grunt> DUMP lmt;
grunt>DUMP  --This will also work, It will just dump the last Alias (i.e; lmt)
```

The above command loads excite-small.log in a table kind of structure with 3 column user time query pointed to by log variable & assign 4 records to lmt variable and finally displays it.

Note: The dump command will fire a map-reduce job. By default the pig run on cluster-mode. You can get the job details using the below port.

http://localhost:8088

Can I just give the directory Name instead of file?

Yes You can. In that case all the files present in the directory will be loaded into the log. It will also take the zip files present in that directory. It will also load all the file present inside the subfolders and their subfolders and so on.

```
grunt> log = LOAD '/user/hduser/data/' AS (user, time, query);
```

Session 2: Word Count Using Pig

1.Place myword.txt in the below location (local location)

/home/hduser/pigExample/myword.txt

2.Open Terminal and navigate to pigExamples directory, copy myword.txt to hdfs path

```
$ cd /home/hduser/pigExample
$ hadoop fs -put myword.txt   /data/myword.txt
$ hadoop fs -cat /data/myword.txt                    --confirm
```

3.Load data in to Pig relation.

Load input from the file named myword.txt, in a table kind of structure with one column called line. The table kind of structure is referred with name inp.

```
$ pig
grunt>inp = LOAD  '/data/myword.txt' AS (line);
grunt>DUMP inp;
```

```
(apple ball cat)
(apple ball orange)
(orange orange ball)
(dog dog apple)
(cat )
(cat)
(apple)
(dog)
(dog)
(ball)
```

4. TOKENIZE splits the line into a field for each word. -- flatten will take the collection of records returned by -- TOKENIZE and produce a separate record for each one, calling the single -- field in the record word.

```
grunt>temp =  FOREACH  inp GENERATE TOKENIZE(line) AS word; --just to see how TOKENIZE works,TOKENIZE is case sensitive
grunt>dump temp;
```

```
 {{(apple),(ball),(cat)}}
({(apple),(ball),(orange)})
({(orange),(orange),(ball)})
({(dog),(dog),(apple)})
({(cat)})
({(cat)})
({(apple)})
({(dog)})
({(dog)})
({(ball)})
```

```
grunt>words = FOREACH inp GENERATE FLATTEN(TOKENIZE(line))  AS word;
grunt> words = FOREACH temp GENERATE FLATTEN(word)  AS word; --You can use this as well if working with temp relation
grunt>dump words;
```

```
(apple)
(ball)
(cat)
(apple)
(ball)
(orange)
(orange)
(orange)
(ball)
(dog)
(dog)
(apple)
(cat)
(cat)
(apple)
(dog)
(dog)
(ball)
```

5. Now group them together by each word.

```
grunt>grpd  = GROUP words BY word;   --here group is a keyword
grunt>DUMP grpd;
```

```
(cat,{(cat),(cat),(cat)})
(dog,{(dog),(dog),(dog),(dog)})
(ball,{(ball),(ball),(ball),(ball)})
(apple,{(apple),(apple),(apple),(apple)})
(orange,{(orange),(orange),(orange)})
```

6.Describe the relation

```
grunt>DESCRIBE grpd;
grunt>DESCRIBE ;    --This also works.No need to write the tuple name
grpd: {group: chararray,words: {(word: chararray)}}
```

grpd

group	words
cat dog ball apple	{(cat),(cat),(cat)} {(dog),(dog),(dog),(dog)} {(ball),(ball),(ball),(ball)} {(apple),(apple),(apple),(apple)} {(orange),(orange),(orange)}

6. For each group count the occurrences.

```
grunt>cntd  = FOREACH grpd GENERATE group, COUNT(words); --group is a column name here
```

7.Print out the results.

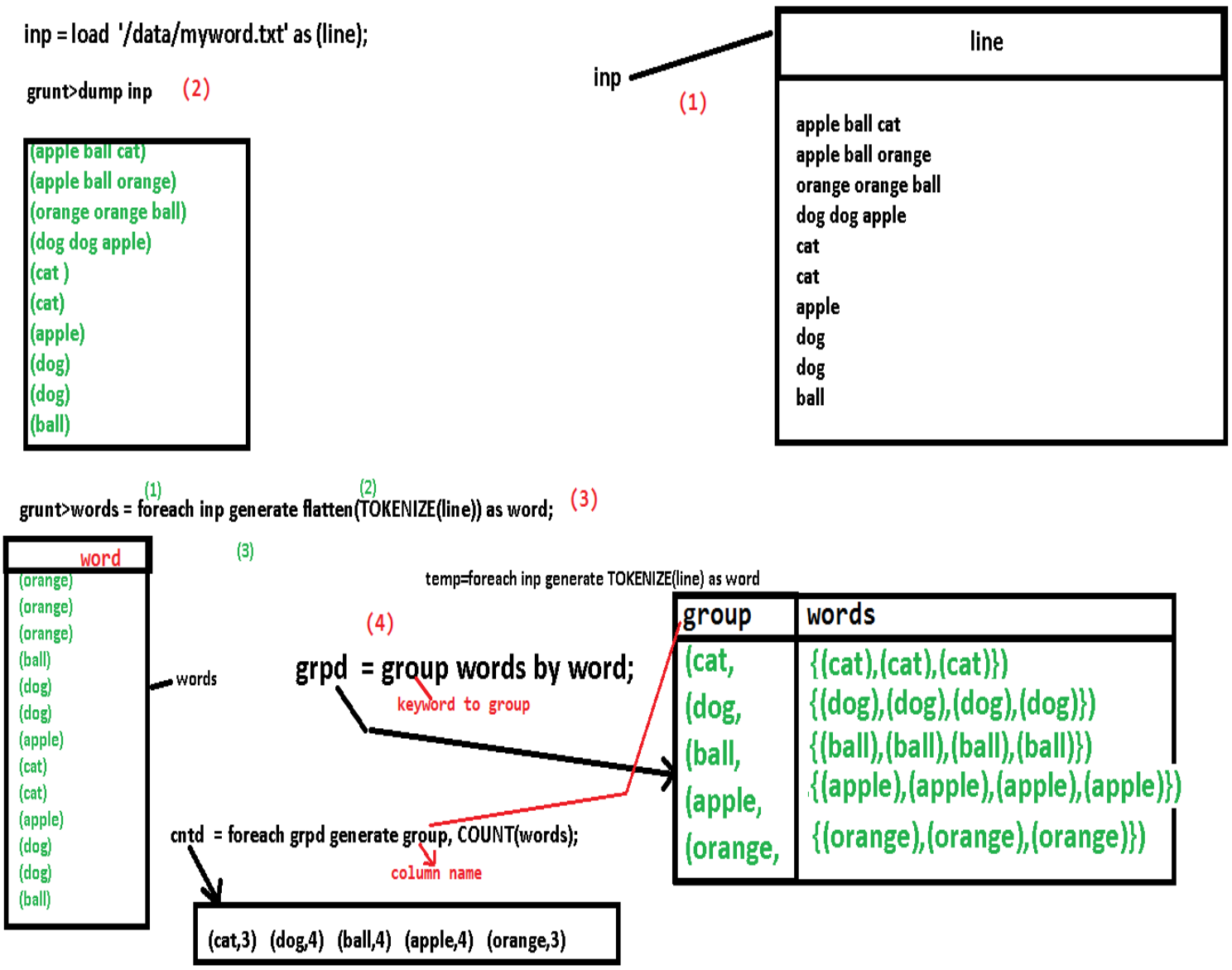
```
grunt> dump cntd;
(cat,3)
(dog,4)
(ball,4)
(apple,4)
(orange,3)
```

8.Store the output in a file

```
grunt>STORE cntd into '/wordcount.txt';
```

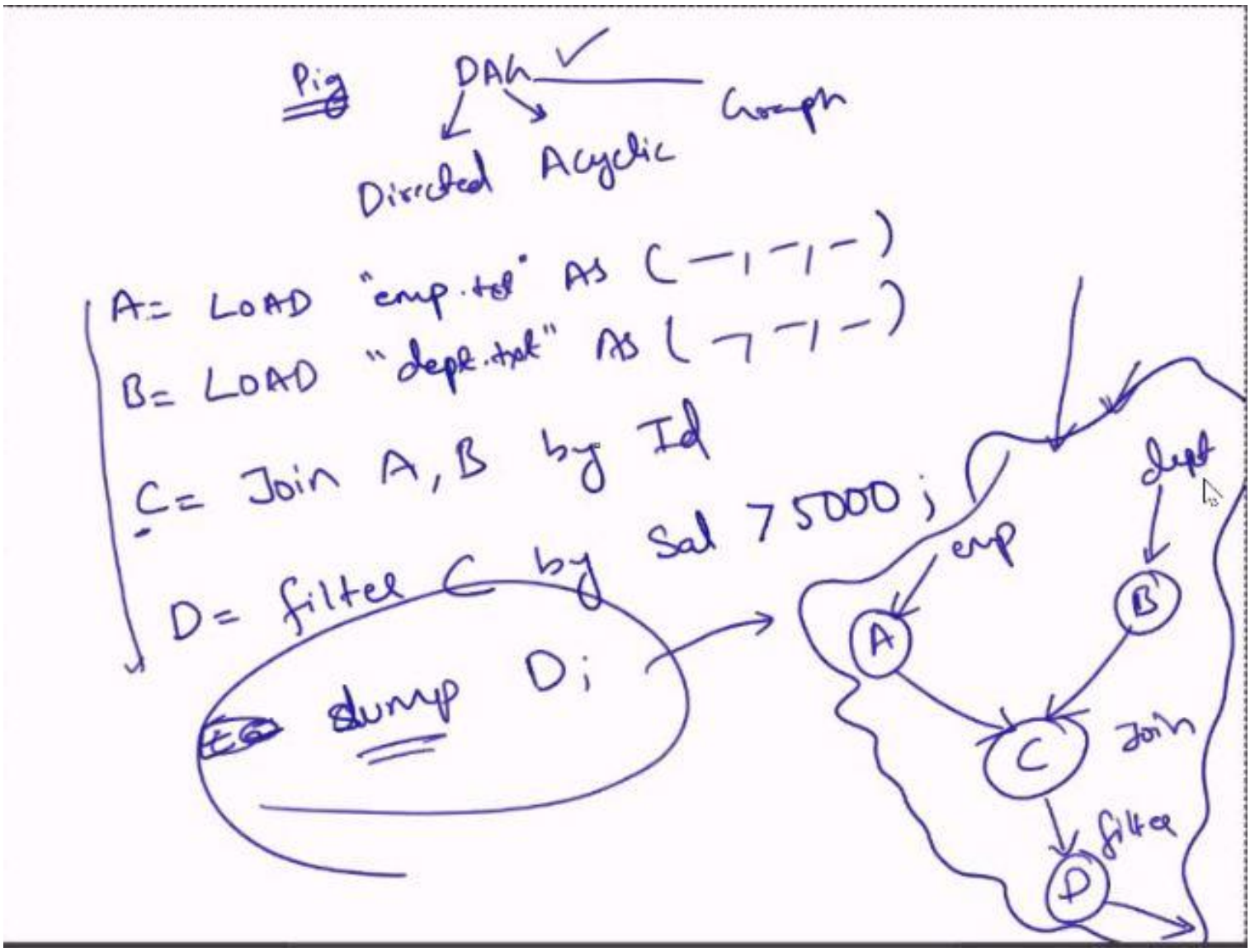
Note: The above command will execute the Map-Reduce job and it stores the output in a directory called /wordcount.txt. Inside that directory you will see part-r-000000 file storing your actual output.

wordcount.txt is a folder name


inp = load '/data/myword.txt' as (line);
grunt>dump inp (2)
(apple ball cat)
(apple ball orange)
(orange orange ball)
(dog dog apple)
(cat)
(cat)
(apple)
(dog)
(dog)
(ball)
inp
(1)
line
apple ball cat
apple ball orange
orange orange ball
dog dog apple
cat
cat
apple
dog
dog
ball
(1)
(2)
grunt>words = foreach inp generate flatten(TOKENIZE(line)) as word; (3)
(3)
word
(orange)
(orange)
(orange)
(ball)
(dog)
(dog)
(apple)
(cat)
(cat)
(apple)
(dog)
(dog)
(ball)
words
temp=foreach inp generate TOKENIZE(line) as word
(4)
grpd = group words by word;
keyword to group
group
words
(cat,
(dog,
(ball,
(apple,
(orange,
{(cat),(cat),(cat)})
{(dog),(dog),(dog),(dog)})
{(ball),(ball),(ball),(ball)})
{(apple),(apple),(apple),(apple)})
{(orange),(orange),(orange)})
cntd = foreach grpd generate group, COUNT(words);
column name
(cat,3) (dog,4) (ball,4) (apple,4) (orange,3)

Pig works on DAG concept (Directed Acyclic Graph)

Directed mean it have directions representing the flow
Acyclic mean it does not have any kind of loop in graph



When we do the DUMP statement it basically perform the MR operation

Session 3: Pig Datatypes

Data types are of 2 types:

1. scalar type: store a single value
2. complex type: multiple values

scalar type:-

1. int →4 bytes
2. long →8 bytes
3. float →4 bytes
4. double →8 bytes
5. chararray →String equivalent, character data
6. bytearray →binary data (default datatype)

complex types:-

1. map→key, value pair
2. tuples->Kind of Classes or Structure
3. bags->Collections of tuples

tuples

1. A fixed length, order collection of pig data elements
2. Tuples will be divided into fields, each field contains one data element, this element can be of any type Eg ('bat',200,6000)
3. You can have schema attached with it

bags:

1. A collection of unordered collection of tupels,
2. A bag can also have schema attached to it. Eg {('prasad',4000,...),('Raj',6000),('Sashi',7000)}

Defaults:

1. Default data separator is tab.
2. Default data type is bytearray.

Working with datatypes:

```
grunt> log = LOAD '/user/hduser/data/excite-small.log' AS (user, time, query);
grunt> describe log;
log: {user: bytearray,time: bytearray,query: bytearray}
```

Loading data in table with more fields than what exists in data file

```
grunt> log = LOAD '/user/hduser/data/excite-small.log' AS
       (
        user:chararray,
        time:chararray,
        query:chararray,
        city:chararray,
        zipcode
        );
grunt> describe log;
log: {user: chararray,time: chararray,query: chararray,city: chararray,zipcode: bytearray}
grunt> dump log;
(C5D01E05FF9CA265,970916155706,mary lou allgood,,)
(C5D01E05FF9CA265,970916155729,mary lou allgood,,)
(DB38E7AF26F3AD9A,970916114356,microsoft excel,,)
```

It gives all the 3 columns data followed by empty data for extra records

Loading data in table with lesser field than what exists in data file

It loads only few columns, and ignores other

```
grunt> log = LOAD '/user/hduser/data/excite-small.log' AS
       (
        user:chararray,
        time:chararray
       );
```

```
(98F5BBD3754D292F,970916082614)
(98F5BBD3754D292F,970916082616)
(98F5BBD3754D292F,970916082713)
```

Loading data without schema

```
grunt>log = LOAD '/user/hduser/data/excite-small.log';
grunt> describe log;
Schema for log unknown.
grunt> dump log;
```

Notes:

1. You can load the data without any schema.
2. Whenever you try to describe the schema less table ,it will give you Schema unknown error.
3. However you can dump the data.
4. To access the columns of schema less data use $0,$1....

Group by, Limit, Order, Local Mode in Pig & Storing Result

Objective:-

Find out top 2 highly paid employees Organization wise
Refer to employee.txt present along with this document. Copy the file into /home/hduser folder
Load the data into the pig relation by running the pig in local mode

```
$pig -x local
```

```
grunt>
empData = load '/home/hduser/employee.txt'
using PigStorage(',') as
(
emp_org:chararray,
emp_name:chararray,
emp_sal:int
);
```

```
grunt>dump empData;
```

```
(TCS,Ratan,10000)
(TCS,Kiran,8500)
(TCS,Ramesh,12000)
(TCS,Karan,5600)
(TCS,Mohan,7800)
(WIPRO,suraj,9000)
(WIPRO,Radha,11500)
(WIPRO,suman,9200)
(WIPRO,Raveena,6700)
(WIPRO,Kusum,9100)
(INFOSYS,Shyam,8900)
(INFOSYS,Karan,8700)
(INFOSYS,Mahesh,9200)
(INFOSYS,Ranjan,7510)
(INFOSYS,Binod,10200)
```

Find out how many employees are reviewed(total no of records)

```
grunt>grpd = GROUP empData ALL;
grunt>describe grpd;
grpd: {group: chararray,empData: {(emp_org: chararray,emp_name:
chararray,emp_sal: int)}}
```

```
grunt>dump grpd;
```

```
(all,{(TCS,Ratan,10000),(TCS,Kiran,8500),(TCS,Ramesh,12000),(TCS,Karan,5600),(TCS,Mohan,7800),(
WIPRO,suraj,9000),(WIPRO,Radha,11500),(WIPRO,suman,9200),(WIPRO,Raveena,6700),(WIPRO,Kus
um,9100),(INFOSYS,Shyam,8900),(INFOSYS,Karan,8700),(INFOSYS,Mahesh,9200),(INFOSYS,Ranjan,75
10),(INFOSYS,Binod,10200)})

grunt>emp_count = FOREACH grpd GENERATE COUNT(empData);
grunt>dump emp_count;

(15)
```

Group employees based on Organisation

```
grunt>grpd = GROUP empData BY emp_org;
grunt>dump grpd;

(TCS,{(TCS,Kiran,8500),(TCS,Ramesh,12000),(TCS,Karan,5600),(TCS,Ratan,10000
),(TCS,Mohan,7800)})

(WIPRO,{(WIPRO,Raveena,6700),(WIPRO,suraj,9000),(WIPRO,Radha,11500),(WIPRO,
suman,9200),(WIPRO,Kusum,9100)})

(INFOSYS,{(INFOSYS,Shyam,8900),(INFOSYS,Karan,8700),(INFOSYS,Mahesh,9200),(
INFOSYS,Ranjan,7510),(INFOSYS,Binod,10200)})
```

```
grunt>describe grpd;

grpd: {
        group: chararray,
        empData: {(emp_org: chararray,emp_name: chararray,emp_sal: int)}
       }
```

Find out all the companies that we are analysing

```
grunt>all_org = foreach grpd generate group;  --group is column name
grunt>dump all_org;
(TCS)
(WIPRO)
(INFOSYS)
```

Find out total employees surveyed from each company

```
grunt>totEmpOrgWise = foreach grpd generate group,COUNT(empData);
grunt>dump totEmpOrgWise;
(TCS,5)
(WIPRO,5)
(INFOSYS,5)
```

Find out top 2 highly paid employee from each organisation

```
grunt>top2HighPaid = foreach grpd{
sorted = order empData by emp_sal desc;
top2 = limit sorted 2;
generate top2;
};

grunt>dump top2HighPaid;
```

```
({(INFOSYS,Binod,10200),(INFOSYS,Mahesh,9200)})
({(TCS,Ramesh,12000),(TCS,Ratan,10000)})
({(WIPRO,Radha,11500),(WIPRO,suman,9200)})
```

```
grunt>top2HighPaid = foreach grpd{
sorted = order empData by emp_sal desc;
top2 = limit sorted 2;
generate flatten(top2);
};

grunt>dump top2HighPaid;
```

```
(INFOSYS,Binod,10200)
(INFOSYS,Mahesh,9200)
(TCS,Ramesh,12000)
(TCS,Ratan,10000)
(WIPRO,Radha,11500)
(WIPRO,suman,9200)
```

```
grunt> store  top2HighPaid into '/home/hduser/highPaidEmp.txt';
```

This will cause the result to be stored on the folder called highPaidEmp2.txt.It is the result of map-reduce execution

Assignment

1.Findout top 3 highly paid employees from the given dataset.
2.Findout all employees getting more than 10k salary
3.Findout all employees getting more than 10k but they are not from TCS

We are going to see how to perform basic operation in Pig. We are working on NYSE dataset

Running Pig in local mode

In case your input data is in the local file system(not HDFS),You can still process them using pig. Simply launch the pig in local mode as shown below.

```
grunt>pig -x local
```

When you are running the pig in local mode, you will be able to process local file system data
In this session, we will be using local file system to process the data.

Note: While defining variable = expression, there should be space between = and expression.

Example there is a space between = and load statement.

```
grunt>data= load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv';
```

Dealing with Structured data separated by TAB space

If You have structured data that can be represented in form of row and columns of tables, and each columns are separated by TAB then you load the data into pig using below syntax

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv'
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

Dealing with Structured data separated by any other delimiter

Say your data is separated by comma(,) and you would like to load it to pig for processing. Use **using PigStorage(',')** to achieve the same.

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv' using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

Dealing with Unstructured data separated by tab delimiter

We will not specify any column and data type details here

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv';
```

Dealing with Unstructured data separated by any delimiter

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv' using PigStorage(',');
```

Describing Your Relation structure

If you are working with structure data, and you have defined your table schema, You can always get access to the schema using below way

```
grunt>Describe data
```

Note: This will give you error, if you try to describe the table without any schema.

Accessing columns of table with schema

Let's say you want to display STOCK_SYMBOL and STOCK_VOLUME column of all the records, 2nd column and 8th column

If you have defined the table with schema, then use

```
grunt> data1  =  foreach data generate stock_symbol,stock_volume;
grunt> dump data1;
```

Processing Columns of table without schema

If you load the same table without schema, then you need to use $0,$1.... to refer to 1st column,2nd column and so on..

```
grunt> data1  =  foreach data generate $1,$7;
grunt> dump data1;
```

$1 = STOCK_SYMBOL
$7 = STOCK_VOLUME

Store the output to a file

Whenever you execute pig commands it does not gets executed immediately.
It will basically just check if the syntax is correct or not. If not it will throw you error.
When you dump the variable or store it in a file, then it generates & executes the Map-reduce program and you will have the output on the screen or file.
below is the syntax of **store** command

```
grunt>store data into '/home/hduser/mydata.txt';
```

Q. How data gets loaded into relation, if data is comma separated and you don't use using PigStorage(',').

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv'
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

In this case all the data of each line will be stored in the first field. The remaining field will remain empty.

(NASDAQ,ARRS,1993-09-21,22.75,23.50,21.00,21.75,1953700,21.75,,,,,,,,)
(NASDAQ,ARRS,1993-09-20,24.00,24.00,22.00,22.50,1475400,22.50,,,,,,,,)
(NASDAQ,ARRS,1993-09-17,24.00,25.00,23.25,24.00,7520300,24.00,,,,,,,,)

Running Pig in Local mode and storing the result in Local File System

Assuming your pig is running in local mode, and you issue the below command

```
grunt> store data into '/home/hduser/mydata.txt';
```

This will store the data into local file system.

Note mydata.txt is just a folder and not a file. When we executed this command, a map reduce program get created and it gets executed. The output gets written into the above directory called 'mydata.txt'

Running Pig in Local mode and storing the Result in HDFS

Assuming your pig is running in local mode, issue the below command to write data into hdfs.Please make sure hadoop is in running mode.

```
grunt>store data into 'hdfs://localhost:54310/user/hduser/mydata22.txt';
```

Running Pig in Cluster Mode

Copy the file from local to hdfs

```
hadoop fs -put '/home/hduser/nyse/NASDAQ_daily_prices_A.csv'
'/user/hduser/nyse/NASDAQ_daily_prices_A.csv'
```

Start Pig in cluster mode

```
$pig
```

```
grunt>data = load '/user/hduser/nyse/NASDAQ_daily_prices_A.csv' using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

Write the data in HDFS when running pig in cluster mode

Both these writes the data into hdfs

```
grunt> store data into '/user/hduser/temp/mydata1.txt';
grunt> store data into 'hdfs://localhost:54310/user/hduser/temp/mydata2.txt';
```

Write the data in Local when running pig in cluster mode

These writes the data into local file system even if pig is running in cluster mode

```
grunt>store data into 'file:///home/hduser/temp/mydata3.txt';
```

Note: We cannot load the data from local to hdfs directly as shown below.

```
grunt>data = load 'file:///home/hduser/nyse/NASDAQ_daily_prices_A.csv' using PigStorage(',');
```

We have pig in cluster mode and trying to load the data from local file system.

Checking log in case of program error

Whenever you execute dump or load command, pig script gets executed. i.e, it gets converted into map-reduce program and the map-reduce program gets executed. In the current working directory pig writes the log file, which you can refer if you get any error .In case your pig is running on cluster mode,then _log directory gets created in the place where the output is written.

Do I need to specify file name in quotes

```
grunt> store data into '/user/hduser/mydata1.txt';
```

Yes. You need to specify the file name within ' ' inorder to execute the command.

Auto Type casting in Pig

Restart pig in local mode. Let's say we are working with NASDAQ_daily_prices.csv.

```
$ pig -x local
```

Load the data as schema less data

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv' using PigStorage(',');
```

and we executed the below command, where data is pointing to csv file

```
grunt> A = foreach data generate $3+$4;
grunt> dump A;
```

Note:

$3 means 4th column : stock_price_high float

$4 means 5th column : stock_price_low float

In case any one of the column would have been int and other would have been float ,then int data-type would be auto-promoted to float and then it would be evaluated.

Wat if data is corrupted?

Say few $3 column do not have float values but some character, and few $4 columns have some character

NASDAQ,ABXA,2009-12-08,,2.74,2.52,2.55,131700,2.55

<Blank Line>

NASDAQ,ABXA,2009-12-04,ABC,2.66,2.53,2.65,230900,2.65

NASDAQ,ABXA,2009-12-03,2.55,BCD,2.51,2.60,360900,2.60

NASDAQ,ABXA,2009-12-02,ABC,BCD,2.40,2.53,287700,2.53

Just execute

```
grunt> A = foreach data generate $3+$4,$5,$6;
grunt> store A into '/home/hduser/temp/output4';
```

grunt>

You will get some sort of message on your terminal.When you check the output,the result with all the invalid data will be discarded and will have empty values.

```
2016-01-10 06:20:03,016 [pool-3-thread-1] WARN
org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigHadoopLogger -
org.apache.pig.builtin.Utf8StorageConverter(FIELD_DISCARDED_TYPE_CONVERSION_FAILED):
Unable to interpret value [65, 66, 67] in field being converted to double, caught
NumberFormatException <For input string: "ABC"> field discarded
```

```
         2.52    2.55

         2.53    2.65
         2.51    2.60
         2.40    2.53
5.0      2.40    2.53
```

```
grunt>describe A;
```

A: {double}

Working with some function

```
grunt> A = foreach data generate SUBSTRING($1,0,2);
grunt> dump A;
```

Will give you substring of data from 0 till 1(2 is not included)

Filtering the data

Let's find out all the record whose sales_volume >1,50,00,000

```
grunt> B = FILTER data by $7>15000000;
grunt> dump B;
```

Assignment: Find out the weather station id, whose temperature > 30

Filtering data based on text

Find All the records whose stock_symbol= ARRS

```
grunt> C = filter data by ($1 == 'ARRS');
grunt> dump C;
```

Find all the records whose stock_symbol contains AR

```
grunt> C = filter data by ($1 matches'.*AR.*');
grunt> dump C;
```

Find all the records whose stock_symbol= ARRS and stock_volume=1,00,00,000

```
grunt> C = filter data by ($1 == 'ARRS') AND $7>10000000;
grunt> dump C;
```

Find all the records whose stock_symbol= ARRS and stock_volume>24,00,00,000 or stock_symbol= ALOT and stock_volume>300000;

```
grunt> C = filter data by (($1 == 'ARRS') AND $7>24000000) OR
      (($1 == 'ALOT') AND $7 >300000 );
grunt> store C into '/home/hduser/temp/data5' using PigStorage(',');
```

Note: You can use all the logical operators (NOT, AND, OR) and relational operators (< , >, ==, !=, >=, <=) in the filter conditions.

If you say Describe C,you wont get any schema,because C is nothing but a filtered data out of the original data and original data donot have any schema.However if you write the below command

```
grunt>D=  FOREACH C GENERATE $0 .. $8;
```

you will get the schema for this data without any column name.

```
grunt>Describe D;
D: {bytearray,bytearray,bytearray,bytearray,bytearray,bytearray,bytearray,bytearray,bytearray}
```

Filter Data by Date

```
grunt>Dump D;
(NASDAQ,ALOT,2004-03-23,14.16,14.16,11.15,11.70,342600,7.42)
(NASDAQ,ALOT,2000-11-22,4.06,4.13,4.00,4.00,609100,2.30)
(NASDAQ,ALOT,1998-08-31,6.00,6.00,5.25,5.56,626400,3.01)
(NASDAQ,ALOT,1993-12-13,10.50,10.50,9.50,9.63,398400,4.88)
(NASDAQ,ALOT,1986-08-28,6.50,7.50,6.00,7.50,472400,1.66)
(NASDAQ,ALOT,1986-02-24,12.38,13.38,12.38,13.00,319600,2.87)
```

```
(NASDAQ,ARRS,2008-02-15,5.00,5.90,4.96,5.50,37682900,5.50)
(NASDAQ,ARRS,2004-04-22,6.94,7.50,6.39,6.55,24044800,6.55)
```

Convert date column to date type and regenerate all columns(We did not include last column here)

```
grunt>E = foreach D generate $0,$1,ToDate($2, 'yyyy-MM-dd'),$3 .. $7;
grunt>describe E;
E: {bytearray,bytearray,datetime,bytearray,bytearray,bytearray,bytearray,bytearray}
grunt>DUMP E;
```

Note: Always write MM in capital(Standard Practice) although it is not mandatory.

```
(NASDAQ,ALOT,2004-03-23T00:00:00.000-08:00,14.16,14.16,11.15,11.70,342600)
(NASDAQ,ALOT,2000-11-22T00:00:00.000-08:00,4.06,4.13,4.00,4.00,609100)
(NASDAQ,ALOT,1998-08-31T00:00:00.000-07:00,6.00,6.00,5.25,5.56,626400)
(NASDAQ,ALOT,1993-12-13T00:00:00.000-08:00,10.50,10.50,9.50,9.63,398400)
(NASDAQ,ALOT,1986-08-28T00:00:00.000-07:00,6.50,7.50,6.00,7.50,472400)
(NASDAQ,ALOT,1986-02-24T00:00:00.000-08:00,12.38,13.38,12.38,13.00,319600)
(NASDAQ,ARRS,2008-02-15T00:00:00.000-08:00,5.00,5.90,4.96,5.50,37682900)
(NASDAQ,ARRS,2004-04-22T00:00:00.000-07:00,6.94,7.50,6.39,6.55,24044800)
```

Find Only those records that are from Year 2001 onwards

```
grunt>F = filter E by DaysBetween($2,(datetime)ToDate('2001-01-01', 'yyyy-MM-dd')) >=(long)0;
grunt>dump F;
```

```
(NASDAQ,ALOT,2004-03-23T00:00:00.000-08:00,14.16,14.16,11.15,11.70,342600)
(NASDAQ,ARRS,2008-02-15T00:00:00.000-08:00,5.00,5.90,4.96,5.50,37682900)
(NASDAQ,ARRS,2004-04-22T00:00:00.000-07:00,6.94,7.50,6.39,6.55,24044800)
```

Find records of last 15 years

15 Years Means 365*15 days + 3 days for 3 leap year =5478 days

```
grunt>F = filter E by DaysBetween(CurrentTime(),$2) <=(long)5478;
grunt>dump F;
```

```
(NASDAQ,ALOT,2004-03-23T00:00:00.000-08:00,14.16,14.16,11.15,11.70,342600)
(NASDAQ,ARRS,2008-02-15T00:00:00.000-08:00,5.00,5.90,4.96,5.50,37682900)
(NASDAQ,ARRS,2004-04-22T00:00:00.000-07:00,6.94,7.50,6.39,6.55,24044800)
```

Glob(An expression where we can use regular expression

```
grunt>data1 = load '/home/hduser/nyse/NASDAQ_daily_prices_[A-C].csv' using PigStorage(',')
as(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

If you say **[!B-Z]**,then all the file not starting with B to Z will be loaded, i.e, A related data only

symbol	Description
?	matches single character
*	matches 0 or more
[abc]	matching any character within the list
[a-z]	matching any character within the range
[^abc]	not matching any character within the list

[^a-z]	not matching any character within the range

UDFs in foreach

Start Pig in local mode

```
grunt>divs = load '/home/hduser/nyse/NASDAQ_dividends_[A-C].csv' using PigStorage(',')
as(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);
```

Note: While using UDFs in the expression, the resultant table will not preserve the column name

```
grunt>upperdiv =  foreach divs generate UPPER(stock_symbol) ,dividends;
```

It will output stock_symbol in uppercase and dividends as it is.try describing this table(upperdiv)

```
grunt>describe upperdiv;
upperdiv: {org.apache.pig.builtin.upper_stock_symbol_80: chararray, dividends: float}
```

It is clear that dividends column is float, but the first column name is anonymous and given by pig.

If you want to access the first column of upperdiv, you can use $0,

To access 2nd column of upperdiv,you can use dividends(column name) as well as $1

```
grunt>sym = foreach upperdiv generate $0;
grunt>dump sym;
```

//This doesnot work

```
grunt>sym = foreach upperdiv generate org.apache.pig.builtin.upper_stock_symbol_80;
grunt>dump sym;
grunt> sym = foreach upperdiv generate dividends;
grunt>dump sym;
grunt>upperdiv =  foreach divs generate UPPER(stock_symbol) as symbol,dividends;
grunt>describe upperdiv;
upperdiv: {symbol: chararray,dividends: float} --1st column name is symbol or chararry type
grunt>dump upperdiv; --displays all stock_symbol in Upper case and dividends
grunt> sym = foreach upperdiv generate symbol; --Just get access to only stock_symbol
grunt> dump sym;
grunt> sym = foreach upperdiv generate $0; --This also works even though we have column name
grunt> dump sym;
```

What happens if you try to access $4 which is out of range?

You will get the below error. This error you will get while executing the script itself, without using load or store command.

Out of bound access. Trying to access non-existent column: 4

```
grunt>upperdiv =  foreach divs generate UPPER(stock_symbol) as symbol ,dividends;
```

upperdiv

Symbol	Dividends
NOKIA	.34
PANASONIC	.56
NOKIA	.67

```
grunt> grp_symbol = group upperdiv by symbol;
```

gives the output in the below format

```
(WPPGY,{(WPPGY,0.04),(WPPGY,0.036), (WPPGY,0.045)})
```

grp_symbol

Group	Upperdiv
NOKIA	{(nokia,.34),(nokia,.67)}
PANASONIC	(panasonic,.56)

```
grunt>describe grp_symbol;
grp_symbol: {group: chararray,upperdiv: {(symbol: chararray,dividends: float)}}
```

Note: The output is a tuple with 2 components chararray, bag.
Check the column name itself is group here for 1st column and upperdiv for 2nd column.

Find the sum of all the dividends for each symbol

```
grunt> sum = foreach grp_symbol generate group,SUM(upperdiv.dividends);--group is column name
grunt> describe sum;
sum: {group: chararray,double}
grunt> sum = foreach grp_symbol generate group as symbol,SUM(upperdiv.dividends);
grunt>describe sum;
sum: {symbol: chararray,double}  //No 2nd column name. Use as to get the column name as shown below
grunt> sum = foreach grp_symbol generate group as symbol,SUM(upperdiv.dividends) as dividends;
grunt>describe sum;
sum: {symbol: chararray,dividends: double}
```

sum

Symbol	Dividends
NOKIA	1.07
PANASONIC	.56

```
grunt> greaterThan30 = filter sum by dividends >30;
(ANAT,49.13999977707863)
(CNXT,30.679999351501465)
(COMS,50.75)
```

1. Assume that we have the following tables in sql database

```
SQL> select * from dept;
SQL> select * from emp;
```

EMPNO	ENAME	JOB	MGR	emp_deptNo
111	saketh	analyst	444	10
222	sudha	clerk	333	20
333	jagan	manager	111	10
444	madhu	engineer	222	40

DEPTNO	DNAME	LOC
10	INVENTORY	HYD
20	FINANCE	BGLR
30	HR	MUMBAI

2. Lets convert this into pig. Create below 2 files with the comma separated data file. The data file are present on the same place where this document is available.

```
$ more  /home/hduser/pigjoins/emp.csv
$ more /home/hduser/pigjoins/dept.csv
```

3. Start pig in local mode and Load data into pig

```
grunt>
emp = load '/home/hduser/pigjoins/emp.csv'
using PigStorage(',') as
(
emp_id:int,
emp_name:chararray,
emp_desg:chararray,
emp_manager:int,
emp_deptNo:int
);
```

```
grunt>
dept = load '/home/hduser/pigjoins/dept.csv'
using PigStorage(',') as
(
dept_no:int,
dept_name:chararray,
dept_loc:chararray
);
```

4. **INNER JOIN**

This will display all the records that have matched.

```
grunt>inner_join = join emp by emp_deptNo,dept by dept_no;
grunt>dump inner_join;
```

EMP_ID	EMP_NAME	EMP_DESG	MGR	emp_deptNo	dept_no	DNAME	LOC
111	saketh	analyst	444	10	10	INVENTORY	HYD
333	jagan	manager	111	10	10	INVENTORY	HYD
222	sudha	clerk	333	20	20	FINANCE	BGLR

```
grunt> describe inner_join;
inner_join:{emp::emp_id:int, emp::emp_name:chararray, emp::emp_desg: chararray, emp::emp_manager:int, emp::emp_deptNo:int, dept::dept_no:int, dept::dept_name: chararray, dept::dept_loc: chararray}
```

```
grunt> fewdetails1 = foreach inner_join generate emp::emp_id,emp::emp_name;
grunt>fewdetails2 = foreach inner_join generate $0,emp::emp_name;
grunt>dump fewdetails1;
grunt>dump fewdetails2;
```

5. **LEFT OUTER JOIN**

This will display the all matching records and the records which are in left hand side table those that are not in right hand side table.

```
grunt>left_outer_join = join emp by emp_deptNo left outer,dept by dept_no;
grunt>dump left_outer_join;
```

EMPNO	ENAME	JOB	MGR	Emp_deptNo	Dept_no	DNAME	LOC
111	saketh	analyst	444	10	10	INVENTORY	HYBD
333	jagan	manager	111	10	10	INVENTORY	HYBD
222	sudha	clerk	333	20	20	FINANCE	BGLR
444	madhu	engineer	222	40			

6. **RIGHT OUTER JOIN**

This will display all matching records and the records which are in right hand side table those that are not in left hand side table.

```
grunt>right_outer_join = join emp by emp_deptNo right outer,dept by dept_no;
grunt>dump right_outer_join;
```

```
(111,saketh,analyst,444,10,10,INVENTORY,HYD)
(333,jagan,manager,111,10,10,INVENTORY,HYD)
(222,sudha,clerk,333,20,20,FINANCE,BGLR)
(,,,,,30,HR,MUMBAI)
```

EMPNO	ENAME	JOB	DNAME	LOC
111	saketh	analyst	INVENTORY	HYBD
333	jagan	manager	INVENTORY	HYBD
222	sudha	clerk	FINANCE	BGLR
			HR	MUMBAI

7. **FULL OUTER JOIN**

This will display the all matching records and the non-matching records from both tables.

```
grunt>full_outer_join = join emp by emp_deptNo full outer,dept by dept_no;
grunt>dump full_outer_join;
```

EMPNO	ENAME	JOB	DNAME	LOC
333	Jagan	manager	INVENTORY	HYBD
111	Saketh	analyst	INVENTORY	HYBD
222	Sudha	clerk	FINANCE	BGLR
444	Madhu	engineer		
			HR	MUMBAI

```
(111,saketh,analyst,444,10,10,INVENTORY,HYD)
(333,jagan,manager,111,10,10,INVENTORY,HYD)
(222,sudha,clerk,333,20,20,FINANCE,BGLR)
(,,,,,30,HR,MUMBAI)
(444,madhu,engineer,222,40,,,)
```

CROSS JOIN

This will gives the cross product.

```
grunt>emp_cross_dept = CROSS emp,dept;
grunt>dump emp_cross_dept;
```

```
(111,saketh,analyst,444,10,10,INVENTORY,HYD)
(111,saketh,analyst,444,10,20,FINANCE,BGLR)
(111,saketh,analyst,444,10,30,HR,MUMBAI)
(222,sudha,clerk,333,20,10,INVENTORY,HYD)
(222,sudha,clerk,333,20,20,FINANCE,BGLR)
(222,sudha,clerk,333,20,30,HR,MUMBAI)
(333,jagan,manager,111,10,10,INVENTORY,HYD)
(333,jagan,manager,111,10,20,FINANCE,BGLR)
(333,jagan,manager,111,10,30,HR,MUMBAI)
(444,madhu,engineer,222,40,10,INVENTORY,HYD)
(444,madhu,engineer,222,40,20,FINANCE,BGLR)
(444,madhu,engineer,222,40,30,HR,MUMBAI)
```

EMPNO	ENAME	JOB	DNAME	LOC
111	saketh	analyst	INVENTORY	HYBD
222	sudha	clerk	INVENTORY	HYBD
333	jagan	manager	INVENTORY	HYBD
444	madhu	engineer	INVENTORY	HYBD
111	saketh	analyst	FINANCE	BGLR
222	sudha	clerk	FINANCE	BGLR
333	jagan	manager	FINANCE	BGLR
444	madhu	engineer	FINANCE	BGLR
111	saketh	analyst	HR	MUMBAI
222	sudha	clerk	HR	MUMBAI
333	jagan	manager	HR	MUMBAI
444	madhu	engineer	HR	MUMBAI

Limit:-

Limit will of course limit the total number of records in the output, but the output may not be same in each run.

first5 = limit emp_cross_dept 5; //no guarantee of order,differnt output

sample:-

somedata = sample emp_cross_dept 0.2; //gives 20% of the jnd record.

This also produces different number of records randomly.

Parallelism:-

group,order,distinct,join,limit →forces Reducers

If you are not working with above clauses,it will be map-only job.

If you working with the above clauses,it will be map-reduce job.

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv' using PigStorage(',')
```

```
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

grunt>bysymbol = group data by stock_symbol parallel 10;

will use 10 reducers parallely.

But if you are working in local mode, only 1 reducer will act. in hadoop 1 u can use 1 reducer in pig local mode but in hadoop 2 u can use multiple reducer in pig local

Before pig 0.8 the default reducer was 1,which many people don't change and if you dont work with parallel concept the work will be very slow.

from pig 0.8 onwards for each 1 GB they have arranged 1 reducer. Even though the developer does not set the parallism then also pig will use 1 reducer for each GB.

How to set parallel map?

No need to worry about it,as it is driven by Map-size.

Groups

- it collects together records with the same key
- In SQL, group by clause is used with aggregate function
- In Pig, no such association between them
- IN Pig, It collects all records with the same value for the provided key, together into a bag

Group in sql

Assume the below table

Order_details

Product	sale
Pepsi	10
Coca-cola	20
Pepsi	30
Coca-cola	10
Sprite	30

```
SQL>SELECT product, SUM(sale) AS "Total sales"
FROM order_details
GROUP BY product;
```

Result

Product	Sale
Pepsi	40
Coca-cola	30
Sprite	30

Lets use Cloudera VM and work with PIG.

In case you are not using cloudera, then make a copy of this document and replace all `/home/cloudera/Desktop/nyse` with `/home/hduser/nyse`

1. Download Cloudera-vm-5.4.2 & load it to your VMWare.
2. Copy & Extract NYSE folder into Desktop of cloudera OS
3. Launch Pig in local Mode
   $ pig -x local
4. Load the daily_price data from Desktop/nyse to pig

```
grunt>
daily= load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_A.csv'
using PigStorage(',') as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
```

```
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

5. Load dividends data into hive.

```
grunt>
divs = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_[A-Z].csv'
using PigStorage(',') as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);
```

6. **Group the daily_prices by stock symbol(one key)**

```
grunt>groupd = group daily by stock_symbol;
```

7. **Store the result into groupd**

```
grunt>store groupd into '/home/cloudera/Desktop/data1.txt';
```

This will store the groupd data into a folder called abc.txt(not file) in your Desktop. If you try to execute the above statement again then you will get error, saying folder already exists

The output will be something like below

ADP {(NASDAQ,ADP,1998-05-13,64.5,67.5,64.31,66.87,1382400,25.11),
(NASDAQ,ADP,1998-05-12,64.37,64.87,63.63,64.75,1127800,24.32), ... }

8. **Grouping the data based on more than one key**

Let's use dividends table to demonstrate this, because we have less data in that table as compared to daily_prices

```
grunt>groupd = group divs by (exchange,stock_symbol);
grunt>store groupd into '/home/cloudera/Desktop/div1.txt';
```

The sample output is something like this:
(NASDAQ,CY) {(NASDAQ,CY,30-09-2008,16.418),(NASDAQ,CY,29-09-2008,0.0)}

```
grunt>describe groupd;
```

```
groupd: {group: (exchange: chararray,stock_symbol: chararray),divs: {(exchange: chararray,stock_symbol: chararray,date: chararray,dividends: float)}}
```

9. **Sorting data based on single column**

Default sorting orderby is ascending(asc)

Use desc to sort the data in other way.

Lets sort the dividends table based on date

```
grunt>bydate =  ORDER divs BY date;
grunt> store bydate  into '/home/cloudera/Desktop/abc3.txt';
```

The entire divs are order by date column in ascending order. Check the file the data is arranged as per date.

10. **Sorting data based on multiple column.**

```
grunt>bydateValue =  order divs by date,$3;
grunt> store bydateValue  into '/home/cloudera/Desktop/abc4.txt';
```

This will sort the data based on date as well as 4th column.

```
grunt>bydateValue =  order divs by date desc,$3
asc;
grunt> store bydateValue  into '/home/cloudera/Desktop/abc5.txt';
```

Order by will not work with complex datatype(tuples,bag),as there will not be any clue to orderby with which field.

If you are sorting based on a column and that column contains few null values, then null value will always considered as the smallest value, so if you are sorting the column in ascending order, you will always see the null column at the top, followed by not null columns.

11. **Distinct**

It will give you unique rows.

Lets load a file with small data. Start your pig in local mode

```
divs = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_Z.csv'
using PigStorage(',') as
(
exchange:chararray,
```

```
stock_symbol:chararray,
date:chararray,
dividends:float
);
```

Get only the unique stock symbol

```
grunt>all_stock_symbol =   foreach divs generate stock_symbol;
grunt>unique_stock_symbol = distinct all_stock_symbol;
grunt>dump unique_stock_symbol;
```

```
(ZEUS)
(ZION)
(ZIXI)
(ZRBA)
```

1. Assume that we have the following tables in sql database

```
SQL> select * from dept;
SQL> select * from emp;
```

EMPNO	ENAME	JOB	MGR	emp_deptNo
111	saketh	analyst	444	10
222	sudha	clerk	333	20
333	jagan	manager	111	10
444	madhu	engineer	222	40

DEPTNO	DNAME	LOC
10	INVENTORY	HYD
20	FINANCE	BGLR
30	HR	MUMBAI

2. Lets convert this into pig. Create below 2 files with the comma separated data file. The data file are present on the same place where this document is available.

```
$ more  /home/hduser/pigjoins/emp.csv
$ more /home/hduser/pigjoins/dept.csv
```

3. Start pig in local mode and Load data into pig

```
grunt>
emp = load '/home/hduser/pigjoins/emp.csv'
using PigStorage(',') as
(
emp_id:int,
emp_name:chararray,
emp_desg:chararray,
emp_manager:int,
emp_deptNo:int
);

grunt>
dept = load '/home/hduser/pigjoins/dept.csv'
using PigStorage(',') as
(
dept_no:int,
dept_name:chararray,
dept_loc:chararray
);
```

4. **INNER JOIN**

This will display all the records that have matched.

```
grunt>inner_join = join emp by emp_deptNo,dept by dept_no;
grunt>dump inner_join;
```

EMP_ID	EMP_NAME	EMP_DESG	MGR	emp_deptNo	dept_no	DNAME	LOC
111	saketh	analyst	444	10	10	INVENTORY	HYD
333	jagan	manager	111	10	10	INVENTORY	HYD
222	sudha	clerk	333	20	20	FINANCE	BGLR

```
grunt> describe inner_join;
inner_join:{emp::emp_id:int, emp::emp_name:chararray, emp::emp_desg: chararray, emp::emp_manager:int, emp::emp_deptNo:int, dept::dept_no:int, dept::dept_name: chararray, dept::dept_loc: chararray}
```

```
grunt> fewdetails1 = foreach inner_join generate emp::emp_id,emp::emp_name;
grunt>fewdetails2 = foreach inner_join generate $0,emp::emp_name;
grunt>dump fewdetails1;
grunt>dump fewdetails2;
```

5. **LEFT OUTER JOIN**

This will display the all matching records and the records which are in left hand side table those that are not in right hand side table.

```
grunt>left_outer_join = join emp by emp_deptNo left outer,dept by dept_no;
grunt>dump left_outer_join;
```

EMPNO	ENAME	JOB	MGR	Emp_deptNo	Dept_no	DNAME	LOC
111	saketh	analyst	444	10	10	INVENTORY	HYBD
333	jagan	manager	111	10	10	INVENTORY	HYBD
222	sudha	clerk	333	20	20	FINANCE	BGLR
444	madhu	engineer	222	40			

6. **RIGHT OUTER JOIN**

This will display all matching records and the records which are in right hand side table those that are not in left hand side table.

```
grunt>right_outer_join = join emp by emp_deptNo right outer,dept by dept_no;
grunt>dump right_outer_join;
```

```
(111,saketh,analyst,444,10,10,INVENTORY,HYD)
(333,jagan,manager,111,10,10,INVENTORY,HYD)
(222,sudha,clerk,333,20,20,FINANCE,BGLR)
(,,,,,30,HR,MUMBAI)
```

EMPNO	ENAME	JOB	DNAME	LOC
111	saketh	analyst	INVENTORY	HYBD
333	jagan	manager	INVENTORY	HYBD
222	sudha	clerk	FINANCE	BGLR
			HR	MUMBAI

7. **FULL OUTER JOIN**

This will display the all matching records and the non-matching records from both tables.

```
grunt>full_outer_join = join emp by emp_deptNo full outer,dept by dept_no;
grunt>dump full_outer_join;
```

EMPNO	ENAME	JOB	DNAME	LOC
333	Jagan	manager	INVENTORY	HYBD
111	Saketh	analyst	INVENTORY	HYBD
222	Sudha	clerk	FINANCE	BGLR
444	Madhu	engineer		
			HR	MUMBAI

```
(111,saketh,analyst,444,10,10,INVENTORY,HYD)
(333,jagan,manager,111,10,10,INVENTORY,HYD)
(222,sudha,clerk,333,20,20,FINANCE,BGLR)
(,,,,,30,HR,MUMBAI)
(444,madhu,engineer,222,40,,,)
```

CROSS JOIN

This will gives the cross product.

```
grunt>emp_cross_dept = CROSS emp,dept;
grunt>dump emp_cross_dept;
```

```
(111,saketh,analyst,444,10,10,INVENTORY,HYD)
(111,saketh,analyst,444,10,20,FINANCE,BGLR)
(111,saketh,analyst,444,10,30,HR,MUMBAI)
(222,sudha,clerk,333,20,10,INVENTORY,HYD)
(222,sudha,clerk,333,20,20,FINANCE,BGLR)
(222,sudha,clerk,333,20,30,HR,MUMBAI)
(333,jagan,manager,111,10,10,INVENTORY,HYD)
(333,jagan,manager,111,10,20,FINANCE,BGLR)
(333,jagan,manager,111,10,30,HR,MUMBAI)
(444,madhu,engineer,222,40,10,INVENTORY,HYD)
(444,madhu,engineer,222,40,20,FINANCE,BGLR)
(444,madhu,engineer,222,40,30,HR,MUMBAI)
```

EMPNO	ENAME	JOB	DNAME	LOC
111	saketh	analyst	INVENTORY	HYBD
222	sudha	clerk	INVENTORY	HYBD
333	jagan	manager	INVENTORY	HYBD
444	madhu	engineer	INVENTORY	HYBD
111	saketh	analyst	FINANCE	BGLR
222	sudha	clerk	FINANCE	BGLR
333	jagan	manager	FINANCE	BGLR
444	madhu	engineer	FINANCE	BGLR
111	saketh	analyst	HR	MUMBAI
222	sudha	clerk	HR	MUMBAI
333	jagan	manager	HR	MUMBAI
444	madhu	engineer	HR	MUMBAI

Limit:-

Limit will of course limit the total number of records in the output, but the output may not be same in each run.

first5 = limit emp_cross_dept 5; //no guarantee of order,differnt output

sample:-

somedata = sample emp_cross_dept 0.2; //gives 20% of the jnd record.

This also produces different number of records randomly.

Parallelism:-

group,order,distinct,join,limit →forces Reducers

If you are not working with above clauses,it will be map-only job.

If you working with the above clauses,it will be map-reduce job.

```
grunt>data = load '/home/hduser/nyse/NASDAQ_daily_prices_A.csv' using PigStorage(',')
```

```
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

grunt>bysymbol = group data by stock_symbol parallel 10;

will use 10 reducers parallely.

But if you are working in local mode, only 1 reducer will act. in hadoop 1 u can use 1 reducer in pig local mode but in hadoop 2 u can use multiple reducer in pig local

Before pig 0.8 the default reducer was 1,which many people don't change and if you dont work with parallel concept the work will be very slow.

from pig 0.8 onwards for each 1 GB they have arranged 1 reducer. Even though the developer does not set the parallism then also pig will use 1 reducer for each GB.

How to set parallel map?

No need to worry about it,as it is driven by Map-size.

Groups

- it collects together records with the same key
- In SQL, group by clause is used with aggregate function
- In Pig, no such association between them
- IN Pig, It collects all records with the same value for the provided key, together into a bag

Group in sql

Assume the below table

Order_details

Product	sale
Pepsi	10
Coca-cola	20
Pepsi	30
Coca-cola	10
Sprite	30

```
SQL>SELECT product, SUM(sale) AS "Total sales"
FROM order_details
GROUP BY product;
```

Result

Product	Sale
Pepsi	40
Coca-cola	30
Sprite	30

Lets use Cloudera VM and work with PIG.

In case you are not using cloudera, then make a copy of this document and replace all `/home/cloudera/Desktop/nyse` with `/home/hduser/nyse`

1. Download Cloudera-vm-5.4.2 & load it to your VMWare.
2. Copy & Extract NYSE folder into Desktop of cloudera OS
3. Launch Pig in local Mode
   $ pig -x local
4. Load the daily_price data from Desktop/nyse to pig

```
grunt>
daily= load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_A.csv'
using PigStorage(',') as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
```

```
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

5. Load dividends data into hive.

```
grunt>
divs = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_[A-Z].csv'
using PigStorage(',') as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);
```

6. **Group the daily_prices by stock symbol(one key)**

```
grunt>groupd = group daily by stock_symbol;
```

7. **Store the result into groupd**

```
grunt>store groupd into '/home/cloudera/Desktop/data1.txt';
```

This will store the groupd data into a folder called abc.txt(not file) in your Desktop. If you try to execute the above statement again then you will get error, saying folder already exists

The output will be something like below

ADP {(NASDAQ,ADP,1998-05-13,64.5,67.5,64.31,66.87,1382400,25.11),
(NASDAQ,ADP,1998-05-12,64.37,64.87,63.63,64.75,1127800,24.32), ... }

8. **Grouping the data based on more than one key**

Let's use dividends table to demonstrate this, because we have less data in that table as compared to daily_prices

```
grunt>groupd = group divs by (exchange,stock_symbol);
grunt>store groupd into '/home/cloudera/Desktop/div1.txt';
```

The sample output is something like this:
(NASDAQ,CY) {(NASDAQ,CY,30-09-2008,16.418),(NASDAQ,CY,29-09-2008,0.0)}

```
grunt>describe groupd;
```

```
groupd: {group: (exchange: chararray,stock_symbol: chararray),divs: {(exchange: chararray,stock_symbol: chararray,date: chararray,dividends: float)}}
```

9. **Sorting data based on single column**

Default sorting orderby is ascending(asc)

Use desc to sort the data in other way.

Lets sort the dividends table based on date

```
grunt>bydate =  ORDER divs BY date;
grunt> store bydate  into '/home/cloudera/Desktop/abc3.txt';
```

The entire divs are order by date column in ascending order. Check the file the data is arranged as per date.

10. **Sorting data based on multiple column.**

```
grunt>bydateValue =  order divs by date,$3;
grunt> store bydateValue  into '/home/cloudera/Desktop/abc4.txt';
```

This will sort the data based on date as well as 4th column.

```
grunt>bydateValue =  order divs by date desc,$3
asc;
grunt> store bydateValue  into '/home/cloudera/Desktop/abc5.txt';
```

Order by will not work with complex datatype(tuples,bag),as there will not be any clue to orderby with which field.

If you are sorting based on a column and that column contains few null values, then null value will always considered as the smallest value, so if you are sorting the column in ascending order, you will always see the null column at the top, followed by not null columns.

11. **Distinct**

It will give you unique rows.

Lets load a file with small data. Start your pig in local mode

```
divs = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_Z.csv'
using PigStorage(',') as
(
exchange:chararray,
```

```
stock_symbol:chararray,
date:chararray,
dividends:float
);
```

Get only the unique stock symbol

```
grunt>all_stock_symbol =   foreach divs generate stock_symbol;
grunt>unique_stock_symbol = distinct all_stock_symbol;
grunt>dump unique_stock_symbol;
```

```
(ZEUS)
(ZION)
(ZIXI)
(ZRBA)
```

This Session is continuation of session 4(Working with complex data type).

We are going to deal with same dataset student details. But for easiness we have removed all null from the dataset. For example earlier we used to have null ,for email provider. But here we have a meaningful value

Earlier: Email,prasad@gmail.com,null
Now: Email,prasad@gmail.com,google

STEPS

1.start pig in local mode

```
$ pig -x local
```

Note: Here pwd is /home/hduser from where the pig is launched

2.copy student1.txt present with this document in /home/hduser folder

3.Load the data into pig structure

```
grunt> student =  load 'student1.txt'
as
(
  studentId:chararray,
  studentName:chararray,
  contact:bag{t:(type:chararray,detail:chararray,provider:chararray)},
  marks:map[]
);
```

Note: Since student1.txt is present in pwd as well as we lauched pig in local mode from the same directory ,no need to give full path while loading student1.txt.

```
grunt>dump student;
```

To get all student ID

```
grunt>studentIds= foreach student generate studentId;
grunt> AllContact = foreach student generate contact;
grunt> dump AllContact;
({(Mob1,9032412236,Docomo),(Mob2,9032412234,Airtel),(Email,prasad@gmail.com,google)})
({(Mob1,8023122562,Docomo),(Mob2,8023165786,Airtel),(Email,kumar@gmail.com,google)})
({(Mob1,7014305432,Docomo),(Mob2,2342342312,BSNL),(Email,manoj@gmail.com,google)})
({(Mob1,9023123455,Airtel),(Mob2,9032123345,Vodafone),(Email,suman@gmail.com,google)})
({(Mob1,5678956789,Docomo),(Mob2,1234123122,Airtel),(Email,raveena@gmail.com,google)})
```

1.Flatten

In Hive we learned explode command which is used to break the column(with multiple value) into multiple row. Flatten works similarly in Pig.

Flatten Operator

The FLATTEN operator looks like a UDF syntactically, but it is actually an operator that changes the structure of tuples and bags in a way that a UDF cannot. Flatten un-nests tuples as well as bags. The idea is the same, but the operation and result is different for each type of structure.

For tuples, flatten substitutes the fields of a tuple in place of the tuple. For example, consider a relation that has a tuple of the form (a, (b, c)). The expression GENERATE $0, flatten($1), will cause that tuple to become (a, b, c).

For bags, the situation becomes more complicated. When we un-nest a bag, we create new tuples. If we have a relation that is made up of tuples of the form ({(b,c),(d,e)}) and we apply GENERATE flatten($0), we end up with two tuples (b,c) and (d,e). When we remove a level of nesting in a bag, sometimes we cause a cross product to happen. For example, consider a relation that has a tuple of the form (a, {(b,c), (d,e)}), commonly produced by the GROUP operator. If we apply the expression GENERATE $0, flatten($1) to this tuple, we will create new tuples: (a, b, c) and (a, d, e).

1.Get All StudentName,with its contact number linearly

```
grunt>cont = foreach student generate studentName,flatten(contact) ;
grunt>dump cont;
```

```
(Prasad,Mob1,9032412236,Docomo)
(Prasad,Mob2,9032412234,Airtel)
(Prasad,Email,prasad@gmail.com,google)
(Kumar,Mob1,8023122562,Docomo)
(Kumar,Mob2,8023165786,Airtel)
(Kumar,Email,kumar@gmail.com,google)
(Manoj,Mob1,7014305432,Docomo)
(Manoj,Mob2,2342342312,BSNL)
(Manoj,Email,manoj@gmail.com,google)
(Suman,Mob1,9023123455,Airtel)
(Suman,Mob2,9032123345,Vodafone)
(Suman,Email,suman@gmail.com,google)
(Raveena,Mob1,5678956789,Docomo)
(Raveena,Mob2,1234123122,Airtel)
(Raveena,Email,raveena@gmail.com,google)
```

Note: Here it Explodes all tuples inside the bag and mix them all with other column. The result contains each tuple which contains studentName,type,detail,proider as a single tuple

2.Get Email Id of all the users

```
grunt> email = filter cont by $1 == 'Email';
grunt> dump email;
(Prasad,Email,prasad@gmail.com,google)
(Kumar,Email,kumar@gmail.com,google)
(Manoj,Email,manoj@gmail.com,google)
(Suman,Email,suman@gmail.com,google)
(Raveena,Email,raveena@gmail.com,google)
```

```
grunt>emailId = foreach email generate $2;
grunt>dump emailId;
(prasad@gmail.com)
```

```
(kumar@gmail.com)
(manoj@gmail.com)
(suman@gmail.com)
(raveena@gmail.com)
```

```
grunt> store emailId into '/home/hduser/emailId';
```

3.Only Using Flatten

```
grunt> cont = foreach student generate flatten(contact);
grunt>dump cont;
(Mob1,9032412236,Docomo)
(Mob2,9032412234,Airtel)
(Email,prasad@gmail.com,google)
(Mob1,8023122562,Docomo)
(Mob2,8023165786,Airtel)
(Email,kumar@gmail.com,google)
(Mob1,7014305432,Docomo)
(Mob2,2342342312,BSNL)
(Email,manoj@gmail.com,google)
(Mob1,9023123455,Airtel)
(Mob2,9032123345,Vodafone)
(Email,suman@gmail.com,google)
(Mob1,5678956789,Docomo)
(Mob2,1234123122,Airtel)
(Email,raveena@gmail.com,google)
```

Assignment:

Find all Airtel customer

Find All Customer Primary mobile number

Cogroup:- Please use the dataset which was used in joins(emp,dept)

- Looks very similar to group but with some differences.
- Group is applied on one table where as cogroup is used with more than one table.
- Instead of collecting records of one input based on a key, it collects record from N input based on the key.

We will use same emp,dept table

```
grunt>
emp = load '/home/hduser/pigjoins/emp.csv'
using PigStorage(',') as
(
emp_id:int,
emp_name:chararray,
emp_desg:chararray,
emp_manager:int,
emp_deptNo:int
);
```

```
grunt>
dept = load '/home/hduser/pigjoins/dept.csv'
```

```
using PigStorage(',') as
(
dept_no:int,
dept_name:chararray,
dept_loc:chararray
);
```

```
grunt>data  = cogroup emp by emp_deptNo,dept by dept_no;
grunt>describe data;
data: {group: int,emp: {(emp_id: int,emp_name: chararray,emp_desg: chararray,emp_manager:
int,emp_deptNo: int)},dept: {(dept_no: int,dept_name: chararray,dept_loc: chararray)}}
grunt>dump data;
```

```
(10,{(333,jagan,manager,111,10),(111,saketh,analyst,444,10)},{(10,INVENTORY,HYD)})
(20,{(222,sudha,clerk,333,20)},{(20,FINANCE,BGLR)})
(30,{},{(30,HR,MUMBAI)})
(40,{(444,madhu,engineer,222,40)},{})
```

User Defined Functions

- Till pig 0.7,Pig UDFS were written only on Java.
- As of pig 0.15+,we can use Java, Jython, Python, JavaScript, Ruby and Groovy for writing our UDF.
- Using Java is the preferred language. Limited support is provided for Jython, Python, JavaScript, Ruby and Groovy functions.
- In other language currently only the basic interface is supported; load/store functions are not supported. Furthermore, JavaScript, Ruby and Groovy are provided as experimental features because they did not go through the same amount of testing as Java or Jython.
- Pig also provides support for Piggy Bank, a repository for JAVA UDFs. Through Piggy Bank you can access Java UDFs written by other users and contribute Java UDFs that you have written.

Built in UDFS

PigStorage:- we use this UDF when our dataset are not tab space-separated

Launch pig in local mode cloudera system, and make sure you have nyse folder extracted to Desktop

```
$ pig -x local
grunt>
data1 = load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_Z.csv'
using PigStorage(',') as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

Math UDFS: All this UDFs excepts double value, and returns double value

double ABS(double)
double CEIL(double)
double EXP(double)
double FLOOR(double)
double ROUND(double)
double SQRT(double)

Examples:-

grunt> data2 = foreach data1 generate CEIL(stock_price_open);
grunt>dump data2;

Aggregate UDFS

long COUNT(input);
long AVG(input);
long COUNT_STAR(input);
int MAX(input);
int MIN(input);
long SUM(input);

Using COUNT

```
grunt>data_Group= GROUP data1 ALL;
grunt>data_Count = FOREACH data_Group GENERATE COUNT(data1);
```

grunt> dump data_Count;

(58523)

grunt> describe data_Group;
data_Group: {group: chararray,data1: {(exchange: chararray,stock_symbol: chararray,date: chararray,stock_price_open: float,stock_price_high: float,stock_price_low: float,stock_price_close: float,stock_volume: int,stock_price_adj_close: float)}}

The First command groups all the elements into a single entity. That single entity is named as ALL

all {(NASDAQ,ZION,26-03-1990,22.06,22.46,22.06,22.46,21600,1.82),(NASDAQ,ZION,28-03-1990,22.46,22.66,22.06,22.46,46400,1.82),(NASDAQ,ZION,29-03-1990,22.66,22.66,22.46,22.46,26400,1.81).....}

Create your own UDF

1.Create a Java Project called MyUDF. Copy the below jars from the respective folder into lib,and add all of them into the class path.

In cloudera 5.4,for 2nd jar check the below location/usr/jars

2.Create Your class Extends EvalFunc class and override exec(Tuple)

```
package p1;
```

```
import java.io.IOException;

import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;

public class Reverse extends EvalFunc<String>{

    @Override
    public String exec(Tuple input) throws IOException {
        String str=(String)input.get(0);
        StringBuffer sb=new StringBuffer(str);
        return sb.reverse().toString();
    }
 }
```

3.Create a Jar file(StringUDF.jar) and place it under /home/cloudera.

While creating the jar,no need to include those 3 jar files which we had placed under lib folder.

4.Register your jar in Pig environment. /home/cloudera/StringUDF.jar

```
grunt>register '/home/cloudera/StringUDF.jar';
```

5.Load your data,

```
grunt>data = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_Z.csv'
using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);
```

6.Shorten Your data,by finding the unique stock_symbol (optional)

```
grunt>sym = foreach data generate stock_symbol;
grunt>unique_sym = distinct sym;
grunt>dump unique_sym;
```

```
(ZEUS)
(ZION)
(ZIXI)
(ZRBA)
```

6.Use Your UDF(Working in cloudera 4.4 But having some problem in 5.4)

```
grunt>rev = foreach unique_sym generate p1.Reverse($0);
grunt>dump rev;
```

```
(SUEZ)
(NOIZ)
(IXIZ)
(ABRZ)
```

Externalizing your script

Create a script file "myUDF.pig" & put the highlighted script in it. Note the extension can be anything. Quit your grunt terminal

```
$ vim myScript.pig
```

```
register '/home/cloudera/StringUDF.jar';
data2 = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_Z.csv'
using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);

sym =  foreach data2 generate stock_symbol;
unique_sym = distinct sym;
dump unique_sym;
rev= foreach unique_sym generate p1.Reverse($0);
dump rev;
```

Execute your Script

```
$ pig -x local  myScript.pig
```

All the script will be executed one-by one and you will get the same output as you got in the above case.

Passing default import to pig script

Say you have too many UDF present in p1 package (In real time these package name would be meaningful and lengthy),and you wud like to import it by default so that you need not write it in the script while calling.

```
$ vim myScript.txt
```

```
register '/home/cloudera/StringUDF.jar';
data2 = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_Z.csv'
using PigStorage(',')
as
```

```
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);

sym =  foreach data2 generate stock_symbol;
unique_sym = distinct sym;
dump unique_sym;
rev= foreach unique_sym generate Reverse($0);
dump rev;
```

Note: This time we are not using p1 package while calling Reverse() in the script txt file.You need to pass the package while calling the script file. Also script extension is .txt, not .pig

```
$ pig -Dudf.import.list=p1   -x local  myScript.txt
```

This will also produce the same output

Register Pig UDF library at runtime

In the above example,we configured StringUDF.jar within the script itself as the very first line.

```
register '/home/cloudera/StringUDF.jar';
```

However, this is optional. You can pass this jar file location while invoking the script and you can remove the above script from the script file as shown below

```
$ vim myScript.txt
```

```
data2 = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_Z.csv'
using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);

sym =  foreach data2 generate stock_symbol;
unique_sym = distinct sym;
dump unique_sym;
rev= foreach unique_sym generate Reverse($0);
dump rev;
```

```
$ pig -Dpig.additional.jars=/home/cloudera/StringUDF.jar -Dudf.import.list=p1   -x  local  myScript.txt
```

Providing Alias to UDFs

If You feel your pig UDF has a very long package name as well as class name,and you would like to give a fancy name to it. Here is the process.

```
$ vim myScript.txt
```

```
register '/home/cloudera/StringUDF.jar';
define rev p1.Reverse();
data2 = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_Z.csv'
using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);

sym =  foreach data2 generate stock_symbol;
unique_sym = distinct sym;
dump unique_sym;
rev = foreach unique_sym generate rev($0);
dump rev;
```

Note: We have given a nickname called rev to p1.Reverse().In the 2nd last row, we have used rev instead of p1.Reverse().We even don't need to pass **-Dudf.import.list=p1** while calling the script externally.

```
$ pig -x local myScript.txt
```

Calling Java Functions directly from Pig

Assume you have a java class which contains some beneficial method and you want to call that method from Pig environment directly. You can achieve this based on below condition.

- Your Java class method should be public static method, to be called from Pig env.
- Pig provides InvokeForXXX() to call the java method XXX-> Int,Long,Float,Double,String

Integer.toHexString(12) :takes int and returns String in hexadecimal format.so You can use InvokeForString() as shown below

```
$ vim myScript.txt
```

```
define hex  InvokeForString('java.lang.Integer.toHexString','int');

data = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_A.csv'
using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);
data = foreach data generate hex((int)dividends);
nonzero = filter data by (int)$0>0;
first5 = limit nonzero 5;
dump first5;
```

```
$ pig -x local myScript.txt
```

```
(1)
(4)
(5)
(6)
(10)
```

Extra Reads

Apache Pig provides extensive support for user defined functions (UDFs) as a way to specify custom processing. Pig UDFs can currently be executed in three languages: *Java, Python, JavaScript and Ruby.* The most extensive support is provided for Java functions.

Java UDFs can be invoked through multiple ways. The simplest UDF can just extend EvalFunc, which requires only the exec function to be implemented. Every Eval UDF must implement this. Additionally, if a function is algebraic, it can implement Algebraic interface to significantly improve query performance.

Importance of UDFs in Pig:

Pig allows users to combine existing operators with their own or others' code via UDFs. The advantage of Pig is its ability to let users combine its operators with their own or others' code via UDFs. Up through version 0.7, all UDFs must be written in Java and are implemented as Java classes. This makes it easier to add new UDFs to Pig by writing a Java class and informing Pig about the JAR file.

Pig itself comes with some UDFs. Prior to version 0.8, it was a very limited set with only the standard SQL aggregate functions and a few others. In 0.8, a large number of standard string-processing, math, and complex-type UDFs were added.

What is a Piggybank?

Piggybank is a collection of user-contributed UDFs that is released along with Pig. Piggybank UDFs are not included in the Pig JAR, so you have to register them manually in your script. You can also write your own UDFs or use those written by other users.

Eval Functions

The UDF class extends the EvalFunc class which is the base for all Eval functions. All Evaluation functions extend the Java class 'org.apache.pig.EvalFunc. 'It is parameterized with the return type of the UDF which is a Java String in this case. The core method in this class is 'exec.' The 1st line of the code indicates that the function is a part of myudfs package.

It takes one record and returns one result, which will be invoked for every record that passes through the execution pipeline. It takes a tuple , which contains all of the fields the script passes to your UDF as a input. It then returns the type by which you have parameterized EvalFunc.

This function is invoked on every input tuple. The input into the function is a tuple with input parameters in the order they are passed to the function in the Pig script. In the example shown below, the function takes string as input. The following function converts the string from lowercase to uppercase. Now that the function is implemented, it needs to be compiled and included in a JAR.

Aggregate Functions:

Aggregate functions are another common type of Eval function. Aggregate functions are usually applied to grouped data. The Aggregate function takes a bag and returns a scalar value. An interesting and valuable feature of many Aggregate functions is that they can be computed incrementally in a distributed manner. In Hadoop world, this means that the partial computations can be done by the Map and Combiner and the final result can be computed by the Reducer.

It is very important to make sure that Aggregate functions that are algebraic are implemented as such. Examples of this type include the built-in COUNT, MIN, MAX and AVERAGE.

COUNT is an example of an algebraic function where we can count the number of elements in a subset of the data and then sum the counts to produce a final output. Let's look at the implementation of the COUNT function:

Filter Functions:

Filter functions are Eval functions that returns a Boolean value. It can be used anywhere a Boolean expression is appropriate, including the FILTER operator or Bincond expression. Apache Pig does not support Boolean totally, so Filter functions cannot appear in statements such as 'Foreach', where the results are output to another operator. However, Filter functions can be used in filter statements.

Union:-

Similar to Union All in sql.
can be used to put two data sets together by concatenation.

Syntax:-
A = load input1;
B= load input2;
C= union A,B;

Lets Load 2 Table

```
grunt>pricesAB = load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_[A-B].csv'
using PigStorage(',') as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

```
pricesCD = load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_[C-D].csv'
using PigStorage(',') as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
```

Filter the data

```
ABPriceCloseGT26000 = filter pricesAB by stock_price_adj_close >26000;
dump ABPriceCloseGT26000;

CDPriceCloseGT8000 = filter pricesCD by $8 >8000;
dump CDPriceCloseGT8000;
```

```
result =  union ABPriceCloseGT26000,CDPriceCloseGT8000 ;
store result into '/home/cloudera/Desktop/data3';
```

```
(NASDAQ,DARA,20-09-1995,19.25,20.75,19,20.12,3300,8050)
(NASDAQ,DARA,12-09-1995,19.12,21,19.12,21,2900,8400)
(NASDAQ,BHIP,20-12-1995,6.0,7.0,6.0,6.75,4600,27000.0)
(NASDAQ,BHIP,24-11-1995,6.75,6.75,6.25,6.75,400,27000.0)
```

You can take more than 2 relation while performing Union, provided the relation name should be different. Make sure you load all 3 relation properly

```
result =  union ABPriceCloseGT26000,CDPriceCloseGT8000, ABPriceCloseGT25000;
store result into '/home/cloudera/Desktop/data3';
```

It wont work when you have same realation name in the command

```
result =  union ABPriceCloseGT26000,CDPriceCloseGT8000, ABPriceCloseGT26000;
store result into '/home/cloudera/Desktop/data3';
```

Lets prepare the datasets

```
$ vim input1
101,5000
102,6000
103,7000
```

```
$ vim input2
201,8000
202,9000
203,10000
```

Same structure with same column name,same datatype

```
A = load 'input1' as (x:int,y:float);
B = load 'input2' as (x:int,y:float);
C= union A,B;
describe C;
C: {x: int,y: float}
```

Same structure with different column name,same datatype

```
A = load 'input1' as (x1:int,y1:float);
B = load 'input2' as (x:int,y:float);
C= union A,B;
describe C;
C: {x1: int,y1: float}
```

Same structure with same column name,different datatype

```
A = load 'input1' as (x:int,y:float,z:int);
B = load 'input2' as (x:float,y:int,z:int);
C= union A,B;
describe C;
C: {x: float,y: float,z: int}
```

Same structure with same column name,different incompatible datatype

```
A = load 'input1' as (x:int,y:float,z:int);
B = load 'input2' as (x:float,y:int,z:chararray);
C= union A,B;   --Error:cannot cast to byte array.
```

Note: After union, if there are any duplicate data, then it will not be eliminated. You need to use distinct.

Two datasets of different schema

```
$ vim input1
101,5000.0,SE
102,6000.0,SSE
103,7000.0,Admin
```

```
$ vim input2
201,suraz,5000.0,suraz.hadoop@gmail.com
202,vinod,6000.0,vinod@yahoo.com
203,rajesh,7000.0,rajesh@gmail.com
```

```
A = load 'input1' as (empId:int,empSal:float,empDesg:chararray);
B = load 'input2' as (empId:int,empName:chararray,empSal:float,empEmail:chararray);
C= union A,B;
describe C;
schema unknown
```

we can do projection like foreach generate but we cant do operation on data like max,min inorder to work with this we can use UDF

Preserving Schema

the error of above can be resolved by this by allowing pig to forcefully create schema on its own

```
C= union onschema A,B;
describe C;
C: {empId: int,empSal: float,empDesg: chararray,empName: chararray,empEmail: chararray}
DUMP C;
(201,5000.0,,suraz,suraz.hadoop@gmail.com)
(202,6000.0,,vinod,vinod@yahoo.com)
(203,7000.0,,rajesh,rajesh@gmail.com)
(101,5000.0,SE,,)
(102,6000.0,SSE,,)
(103,7000.0,Admin,,)
```

Explain:

```
grunt>explain result;
$pig -x local -e 'explain -script myscript.pig'
```

In case explain did not work. Try this.

Write the below piece of configuration in /etc/pig/conf/pig.properties

```
pig.enable.plan.serialization=false
```

illustrate:

- Illustrate need schema data so before you use illustrate make sure the relation is loaded with schema.
- Illustrate cannot work with Union.

```
divs = load '/home/cloudera/Desktop/nyse/NASDAQ_dividends_[A-Z].csv' using PigStorage(',')
as
(
exchange:chararray,
stock_symbol:chararray,
date:chararray,
dividends:float
);
divGT50 = filter divs by dividends>50;
fewDetails = foreach divGT50 generate stock_symbol,dividends;
```

```
grunt>illustrate fewDetails;
```

Running sh Commands being from grunt terminal

If you are in pig terminal and you want to execute any shell script command(Linux command),then you can use sh command as shown below

```
grunt>sh ls
grunt>sh ls -ltr
grunt> sh rm -r '/home/cloudera/Desktop/myResult.txt'
grunt> sh mkdir Desktop/data1
```

Running fs Commands being from grunt terminal

If you are in pig terminal and you want to execute any hdfs command, then you can use fs command as shown below.

Please make sure you login to pig in cluster mode and not in Local Mode.Otherwise it will perform all operations in local mode.

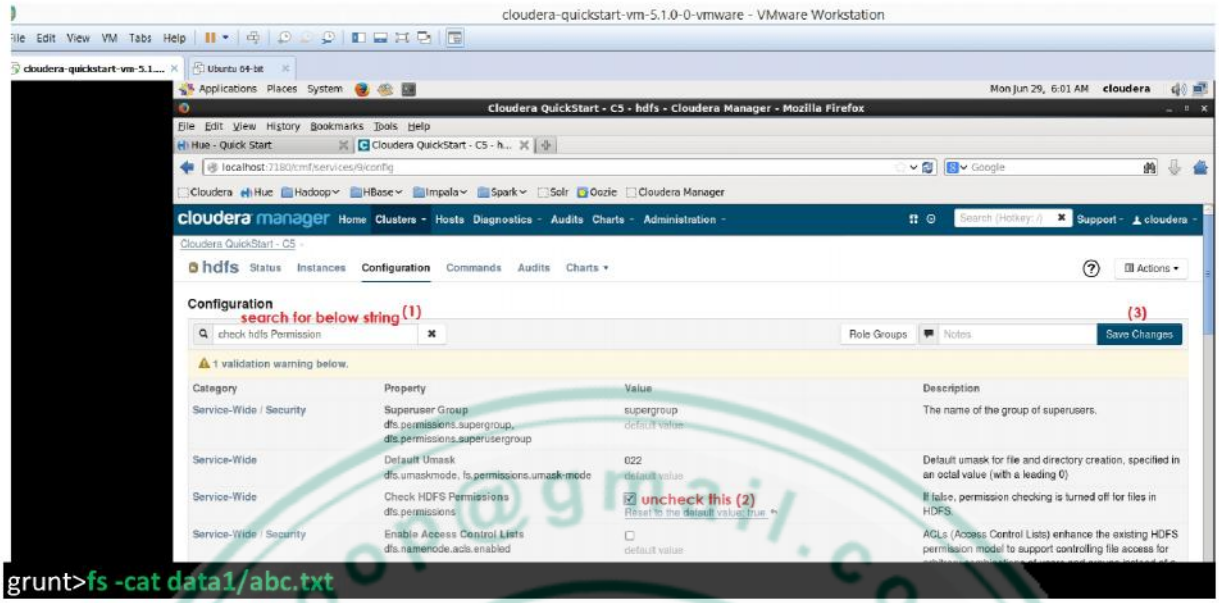
```
grunt>fs -mkdir data1      --creates folder in linux if pig is in local mode
grunt>fs -ls               --displays all files and folder of local file system if pig is in local mode
grunt> sh ls -ltr | grep 'data1'
--did not work in cloudera 5.5.0
grunt>fs -put - /data1/abc.txt  --No sense of running this in local mode
```

For the first time you will get the error as shown below. check if you have started pig in local mode and trying to put the data in cluster.

```
grunt> fs -put - /data1/abc.txt   --run this in cluster mode
put: Permission denied: user=cloudera, access=WRITE, inode="/":hdfs:supergroup:drwxr-xr-x
--Write something and press ctrl+x when you are done
```

Solution:

1. Log into cloudera Manager at http://localhost:7180
2. Enter credentials as cloudera/cloudera
3. Click on HDFS.
4. Navigate to configuration
5. Search for "check hdfs Permission" in the search box and uncheck the select box as shown below.
6. Click on save changes.
7. Re-try the above step. You may have to restart the HDFS service if it is not reflected.



Parameter substitution:-

Assume there is a script which has to be run daily after 8:30 AM once you reach the office. The script should run based on yesterday date. We would like to get all previous date data, and do some processing on it. This is a daily monitoring task. Assume daily_prices_z.csv containing data of various date and it keep on getting added every day.

myDailyRun.pig

```
daily = load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_Z.csv'
using PigStorage(',') as (
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
yesterday = filter daily by date=='11-01-2010';
store  yesterday into '/home/cloudera/Desktop/myResult.txt';
```

```
$ pig -x local myDailyRun.pig
```

You can always replace the date with your choice date everytime before running it. But why not externalize it. Lets not hardcode the date. And lets say sometimes you may have to run this script with some other date as requested by client by creating some ticket.

After Substituion

myDailyRun.pig

```
--sh rm -r '/home/cloudera/Desktop/myResult.txt'; --useful for 2nd time
daily = load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_Z.csv'
```

```
using PigStorage(',') as (
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
yesterday = filter daily by date=='$MYDATE';   this $ means mydate is passed from external source
store  yesterday into '/home/cloudera/Desktop/myResult.txt';
```

```
$ pig -x local -p MYDATE=11-01-2010 myDailyRun.pig
```

Use Parameter file:-

Create a Param file and set all the properties as key value pair.

```
$ vim /home/cloudera/daily.param
MYDATE=11-01-2010
```

Now execute the script in the following way.

```
$ pig -x local -param_file daily.param myDailyRun.pig
```

Prepare a script file

```
$ vim myDailyRun.sh --write the below 2 lines and save them
  rm -r /home/cloudera/Desktop/myResult.txt
  pig -x local -param_file daily.param myDailyRun.pig
```

Execute the script File

```
$ sh myDailyRun.sh
```

You can later use crone,Autosys,Quartz schedular to schedule the job regularly.

Dry run

Assume you have a script file with lots of script in it, and you have done a lot of substitution. You would like to see how your script will look like after substitution of all the parameters without executing them so that you can correct your script if needed.

```
$ pig -x local -dryrun -param_file daily.param myDailyRun.pig
```

On Executing the above query, it will create a file called **myDailyRun.pig.substituted** where you will find the script after substituting the value.

```
$ more myDailyRun.pig.substituted
```

```
daily = load '/home/cloudera/Desktop/nyse/NASDAQ_daily_prices_Z.csv'
using PigStorage(',') as (
exchange:chararray,
stock_symbol:chararray,
date:chararray,
stock_price_open:float,
stock_price_high:float,
stock_price_low:float,
stock_price_close:float,
stock_volume:int,
stock_price_adj_close:float
);
yesterday = filter daily by date=='11-01-2010';
store  yesterday into '/home/cloudera/Desktop/myResult.txt';
```

Now You can directly run the substituted script as below

```
$ pig -x local myDailyRun.pig.substituted
```

XML Processing with Pig

Note:
Tested on Hadoop 2.7.2 with Pig 0.16.
Hadoop 1.2.1 with Pig 0.16 may not work (Compatibility issues).Please try with some old version

Basically we can use 2 techniques to process XML using pig.

1. Using Regular Expression
2. Using XPath

Pig provides the below 2 Loaders process XML file.

1. XMLLoader
2. StreamingXMLLoader

Objective:
We have employee.xml under xmlData folder present along with this document.
Parse the XML data and load them in row format.

Using XPath:-

1. Login to pig in local mode

```
$pig -x local
```

2. Import Piggybank.jar to use XPath udf's

```
grunt>REGISTER /home/hduser/ecosystem/pig/lib/piggybank.jar;
```

3. Define an alias for XPath udf

```
grunt>DEFINE XPath org.apache.pig.piggybank.evaluation.xml.XPath();
```

4. Load data into relation 'A'

```
grunt>A =  LOAD '/home/hduser/xmlData/employee.xml' using
org.apache.pig.piggybank.storage.XMLLoader('employee') as (data:chararray);
```

Note: We are trying to parse the employee.xml by <employee> tag. The output will be a bag(A) containing 5 tuples. Each tuple having <employee>…<employee> with a single column named data.

5. Split each employee tag into multiple columns

```
grunt>B = FOREACH A GENERATE XPath(data, 'employee/empId') AS EMP_ID,
XPath(data, 'employee/empName') AS EMP_NAME,
XPath(data, 'employee/empSalary') AS EMP_SAL,
XPath(data, 'employee/empCompany') AS EMP_ORG,
XPath(data, 'employee/empEmail'); --We did not provide any column name for last column
```

Note: It is because of DEFINE in step-3 that we are directly using XPath instead of fully qualified classname.

```
grunt> describe; --semicolon not necessary
B: {EMP_ID: chararray,EMP_NAME: chararray,EMP_SAL: chararray,EMP_ORG: chararray,chararray}
```

```
grunt> dump;
```

```
(101,suraj,5000,TCS,suraj@tcs.com)
(102,shyam,7000,capgemini,shyam@capgemini.com)
(103,kumar,5000,cognizant,kumar@congizant.com)
(104,mahesh,7000,amazon,mahesh@amazon.com)
(105,kaith,9000,nowonders,kaith@nowonders.com)
```

We have successfully converted XML to tabular Format. Now you can use relation 'B' to get your Desired output.

Using Regular Expression:-

1. Load the XML file into relation 'A'

```
$pig -x local
grunt>REGISTER /home/hduser/ecosystem/pig/lib/piggybank.jar;
grunt>A =  LOAD '/home/hduser/xmlData/employee.xml' using
org.apache.pig.piggybank.storage.XMLLoader('employee') as (data:chararray);
```

2. Extract the field from the above alias.

```
grunt>C = foreach A generate
REGEX_EXTRACT_ALL(data,'.*(?:<empId>)([^<]*).*(?:<empName>)([^<]*).*(?:<empSalary>)([^<]
*).*(?:<empCompany>)([^<]*).*(?:<empEmail>)([^<]*).*');
```

OR

```
grunt>C = foreach A GENERATE
REGEX_EXTRACT_ALL(data,'<employee>\\s*<empId>(.*)</empId>\\s*<empName>(.*)</empNa
me>\\s*<empSalary>(.*)</empSalary>\\s*<empCompany>(.*)</empCompany>\\s*<empEmail>
(.*)</empEmail>\\s*</employee>');
```

Note:

Statement 1: start from <empId>,scan for all the characters till < is not encountered. It means the body of <empId></empId> and so on.

Statement 2: Get the content of <empId></empId> and so on.

```
grunt>dump
((101,suraj,5000,TCS,suraj@tcs.com))
((102,shyam,7000,capgemini,shyam@capgemini.com))
((103,kumar,5000,cognizant,kumar@congizant.com))
((104,mahesh,7000,amazon,mahesh@amazon.com))
((105,kaith,9000,nowonders,kaith@nowonders.com))
```

Note: The output is tuples at each row, which contains another tuples i.e. just one column.

3. Unpack the tuple and get access to all the column

```
grunt>D= foreach C GENERATE FLATTEN($0) AS
(emp_id:chararray,emp_name:chararray,emp_sal:float, emp_org:chararray,
emp_email:chararray);
```

```
grunt>dump
(101,suraj,5000,TCS,suraj@tcs.com)
(102,shyam,7000,capgemini,shyam@capgemini.com)
(103,kumar,5000,cognizant,kumar@congizant.com)
(104,mahesh,7000,amazon,mahesh@amazon.com)
(105,kaith,9000,nowonders,kaith@nowonders.com)
```

Observation:

When I execute the below command

```
$echo $PIG_HOME
```

The output is /home/hduser/ecosystem/pig

So, Can I simply replace /home/hduser/ecosystem/pig with $PIG_HOME as shown below?

```
grunt>REGISTER /home/hduser/ecosystem/pig/lib/piggybank.jar;
grunt>REGISTER $PIG_HOME/lib/piggybank.jar;
```

Answer: No.

$PIG_HOME is an environmental variable, and you cannot access its value in the pig command using the interactive terminal.However you can get access to the variable when using with script file.

myscript.pig

```
REGISTER $PIG_HM/lib/piggybank.jar;
--More pig scripts here----
```

```
$ pig -x local -param PIG_HM=$PIG_HOME myscript.pig --The env variable value is passed to Pig variable
```

Assignment:

1. Study more about
   XPath: http://www.w3schools.com/xsl/xpath_intro.asp
   Regular Expression: http://www.hongkiat.com/blog/getting-started-with-regex/
2. Explore piggybank.jar and check the other functionalities available in it.
3. Explore more on XMLLoader & StreamingXMLLoader.
4. Write the Regular Expression to validate
   a) Email,
   b) Date-Format(dd-mm-YYYY),
   c) Credit-Card,
   d) SSN(For US),
   e) PAN Card(for Indian)

Note: Please do not neglect the assignment. You will thank me either during interview or at your work place☺