# Diagnostic Operators

- The **load** statement will simply load the data into the specified relation in Apache Pig. To verify the execution of the **Load** statement, you have to use the Diagnostic Operators.
- Pig Latin provides four different types of diagnostic operators:
  - Dump operator
  - Describe operator
  - Explanation operator
  - Illustration operator

- Dump operator
- The Dump operator is used to run the Pig Latin statements and display the results on the screen. It is generally used for debugging Purpose.

```
grunt> customers= LOAD '/home/cloudera/customers.txt' USING
PigStorage(',') as
(id:int,name:chararray,age:int,address:chararray,salary:int);
grunt> dump customers;
```

customers.txt

- Describe operator
- The **describe** operator is used to view the schema of a relation/bag.

grunt> customers= LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int,name:chararray,age:int,address:chararray,salary:int);

grunt> describe customers;

```
customers: {id: int,name: chararray,age: int,address: chararray,salary: int}
```

- Explain operator
- The **explain** operator is used to display the logical, physical, and MapReduce execution plans of a relation/bag.

```
grunt> customers= LOAD '/home/cloudera/customers.txt' USING
PigStorage(',') as
(id:int,name:chararray,age:int,address:chararray,salary:int);
grunt> explain customers;
```

- Illustrate operator
- The **illustrate** operator is used to display the logical, physical, and MapReduce execution plans of a relation/bag.

```
grunt> customers= LOAD '/home/cloudera/customers.txt' USING
PigStorage(',') as
(id:int,name:chararray,age:int,address:chararray,salary:int);
grunt> illustrate customers;
```

# Grouping & Joining

# Group Operator

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

- grunt> **student_details** = LOAD '/home/cloudera/students.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
- grunt> **student_groupdata** = GROUP student_details by ***age***;

students.txt

- grunt> dump **student_groupdata;**

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

- grunt> describe **student_groupdata;**

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

- grunt> Illustrate **student_groupdata;**

# Grouping by Multiple Columns

- grunt> **student_details** = LOAD '/home/cloudera/students.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

- grunt> **student_multiplegroup** = GROUP student_details by (***age, city***);

- grunt> dump student_multiplegroup;

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

# Join Operator

- The **JOIN** operator is used to combine records from two or more relations.
- Types of Joins:
  - Self-join
  - Inner-join
  - Outer join : left join, right join, full join

customers.txt

orders.txt

# Self Join

- **customers** = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);
- **orders** = LOAD '/home/local/orders.txt' USING PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);

- grunt> **customers1** = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);
- grunt> **customers2** = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);

- grunt> **customers3** = JOIN customers1 BY id, customers2 BY id;

- grunt> Dump customers3;

# Inner Join (equijoin)

- grunt> customers = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);
- grunt> orders = LOAD '/home/cloudera/orders.txt' USING PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);
- grunt> customer_orders = JOIN customers BY id, orders BY customer_id;
- grunt> dump customer_orders;

# Left Outer Join

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

- grunt> customers = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);
- grunt> orders = LOAD '/home/cloudera/orders.txt' USING PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);
- grunt> outer_left = JOIN customers BY id LEFT OUTER, orders BY customer_id;
- grunt> Dump outer_left;

# Right Outer Join

- The **right outer join** operation returns all rows from the right table, even if there are no matches in the left table.
- grunt> customers = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);
- grunt> orders = LOAD '/home/cloudera/orders.txt' USING PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);

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

- grunt> Dump outer_right;

# Full Outer Join

- The **full outer join** operation returns rows when there is a match in one of the relations.
- grunt> customers = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);
- grunt> orders = LOAD '/home/cloudera/orders.txt' USING PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);

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

- grunt> Dump outer_full;

# Cross Operator

- grunt> customers = LOAD '/home/cloudera/customers.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);
- grunt> orders = LOAD '/home/cloudera/orders.txt' USING PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);

- grunt> cross_data = CROSS customers, orders;

- grunt> Dump cross_data;

# Combining & Splitting

# Union Operator

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

Student_data1.txt

Student_data2.txt

- grunt> student1 = LOAD '/home/cloudera/student_data1.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
- grunt> student2 = LOAD '/home/cloudera/student_data2.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
- grunt> student = UNION student1, student2;
- grunt> dump student;

# Split Operator

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

student_details.txt

- grunt> student_details = LOAD '/home/cloudera/student_details.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

- *Let us now split the relation into two, one listing the students age less than 23, and the other listing the students having the age between 23 and 25.*

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

- grunt> Dump student_details1;

- grunt> Dump student_details2;

# Filtering

# Filter Operator

- The **FILTER** operator is used to select the required tuples from a relation based on a condition.
- grunt> student_details = LOAD '/home/cloudera/student_details.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
- grunt> filter_data = FILTER student_details BY city == 'Chennai';
- grunt> dump filter_data;

student_details.txt

# Distinct Operator

- The **DISTINCT** operator is used to remove redundant (duplicate) tuples from a relation.
- grunt> student_details = LOAD '/home/cloudera/student_details.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
- grunt> distinct_data = DISTINCT student_details;
- grunt> dump distinct_data;

student_details.txt

# Foreach Operator

- The **FOREACH** operator is used to generate specified data transformations based on the column data.

  grunt> student_details = LOAD '/home/cloudera/student_details.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray,age:int, phone:chararray, city:chararray);

- get the id, age, and city values of each student from the relation **student_details** and store it into another relation named **foreach_data** using the **foreach** operator.
- grunt> foreach_data = FOREACH student_details GENERATE id,age,city;
- grunt> Dump foreach_data;

# Sorting

# Order By


- The **ORDER BY** operator is used to display the contents of a relation in a sorted order based on one or more fields.
- grunt> student_details = LOAD '/home/cloudera/student_details.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
- grunt> order_by_data = ORDER student_details BY age DESC;

# Limit Operator

- grunt> student_details = LOAD '/home/cloudera/student_details.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
- grunt> limit_data = LIMIT student_details 4;

# Pig Latin Built-In Functions

- Eval Functions
- String Functions
- Date-time Functions
- Math Functions

# Eval Functions

- Avg()
- CONCAT()
- COUNT()
- COUNT_STAR()
- DIFF()
- MAX()
- MIN()
- SIZE()
- SUBTRACT()
- SUM()

# AVG()

- Computes the average of the numeric values in a single-column bag.
- grunt> A = LOAD '/home/cloudera/student.txt' USING PigStorage(',') as (name:chararray, term:chararray, gpa:float);
- grunt> DUMP A;

  (John,fl,3.9F)
  (John,wt,3.7F)
  (John,sp,4.0F)
  (John,sm,3.8F)
  (Mary,fl,3.8F)
  (Mary,wt,3.9F)
  (Mary,sp,4.0F)
  (Mary,sm,4.0F)
- grunt> B = GROUP A BY name;
- grunt> DUMP B;

  (John,{(John,fl,3.9F),(John,wt,3.7F),(John,sp,4.0F),(John,sm,3.8F)})
  (Mary,{(Mary,fl,3.8F),(Mary,wt,3.9F),(Mary,sp,4.0F),(Mary,sm,4.0F)})
- grunt> C = FOREACH B GENERATE A.name, AVG(A.gpa);
- grunt> DUMP C;

  ({(John),(John),(John),(John)},3.850000023841858)
  ({(Mary),(Mary),(Mary),(Mary)},3.925000011920929)

# CONCAT()

- Concatenates two expressions of identical type.
- grunt>A = LOAD ‘/home/Cloudera/data.txt' as (f1:chararray, f2:chararray, f3:chararray);
- grunt>DUMP A;

  (apache,open,source)

  (hadoop,map,reduce)

  (pig,pig,latin)
- grunt>X = FOREACH A GENERATE CONCAT(f2,f3);
- grunt>DUMP X;

  (opensource)

  (mapreduce)

  (piglatin)

# COUNT

- Computes the number of elements in a bag.
- Note: You cannot use the tuple designator (*) with COUNT; that is, COUNT(*) will not work.

- grunt>A = LOAD '/home/cloudera/c.txt' USING PigStorage(',') as (f1:int, f2:int, f3:int);
- grunt>DUMP A;

  1,2,3
  4,2,null
  8,3,4
  4,3,null
  7,5,null
  8,4,3
- grunt>B = GROUP A BY f1;
- grunt>DUMP B;

  (1,{(1,2,3)})
  (4,{(4,2,1),(4,3,3)})
  (7,{(7,2,5)})
  (8,{(8,3,4),(8,4,3)})
- grunt>X = FOREACH B GENERATE COUNT(A);
- grunt>DUMP X;

  (1L)
  (2L)
  (1L)
  (2L)

# COUNT_STAR

- Computes the number of elements in a bag.
- COUNT_STAR includes NULL values in the count computation (unlike COUNT, which ignores NULL values).
- Example
- In this example COUNT_STAR is used the count the tuples in a bag.
- grunt>X = FOREACH B GENERATE COUNT_STAR(A);

# DIFF

- Compares two fields in a tuple.
- grunt> A = LOAD '/home/Cloudera/data.txt' AS (B1:bag{T1:tuple(t1:int,t2:int)},B2:bag{T2:tuple(f1:int,f2:int)});
- grunt> DUMP A;
  ({(8,9),(0,1)},{(8,9),(1,1)})
  ({(2,3),(4,5)},{(2,3),(4,5)})
  ({(6,7),(3,7)},{(2,2),(3,7)})
- grunt> DESCRIBE A;
  a: {B1: {T1: (t1: int,t2: int)},B2: {T2: (f1: int,f2: int)}}
- grunt> X = FOREACH A DIFF(B1,B2);
- grunt> dump X;
  ({(0,1),(1,1)})
  ({})
  ({(6,7),(2,2)})

# MAX

- Computes the maximum of the numeric values or chararrays in a single-column bag. MAX requires a preceding GROUP ALL statement for global maximums and a GROUP BY statement for group maximums.
- ***Example***
  - In this example the maximum GPA for all terms is computed for each student (see the GROUP operator for information about the field names in relation B).

- grunt> A = LOAD ‘home/Cloudera/student.txt' AS (name:chararray, session:chararray, gpa:float);
- grunt> DUMP A;

  (John,fl,3.9F)
  (John,wt,3.7F)
  (John,sp,4.0F)
  (John,sm,3.8F)
  (Mary,fl,3.8F)
  (Mary,wt,3.9F)
  (Mary,sp,4.0F)
  (Mary,sm,4.0F)
- grunt> B = GROUP A BY name;
- grunt> DUMP B;

  (John,{(John,fl,3.9F),(John,wt,3.7F),(John,sp,4.0F),(John,sm,3.8F)})
  (Mary,{(Mary,fl,3.8F),(Mary,wt,3.9F),(Mary,sp,4.0F),(Mary,sm,4.0F)})
- grunt> X = FOREACH B GENERATE group, MAX(A.gpa);
- grunt> DUMP X;

  (John,4.0F)
  (Mary,4.0F)

# MIN

- Computes the minimum of the numeric values or chararrays in a single-column bag. MIN requires a preceding GROUP… ALL statement for global minimums and a GROUP … BY statement for group minimums.
- ***Example***
  - In this example the minimum GPA for all terms is computed for each student (see the GROUP operator for information about the field names in relation B).

- grunt> A = LOAD ‘/home/Cloudera/student.txt' AS (name:chararray, session:chararray, gpa:float);
- grunt> DUMP A;

  (John,fl,3.9F)
  (John,wt,3.7F)
  (John,sp,4.0F)
  (John,sm,3.8F)
  (Mary,fl,3.8F)
  (Mary,wt,3.9F)
  (Mary,sp,4.0F)
  (Mary,sm,4.0F)
- grunt> B = GROUP A BY name;
- grunt> DUMP B;

  (John,{(John,fl,3.9F),(John,wt,3.7F),(John,sp,4.0F),(John,sm,3.8F)})
  (Mary,{(Mary,fl,3.8F),(Mary,wt,3.9F),(Mary,sp,4.0F),(Mary,sm,4.0F)})
- grunt> X = FOREACH B GENERATE group, MIN(A.gpa);
- grunt> DUMP X;

  (John,3.7F)
  (Mary,3.8F)

# SIZE

- Computes the number of elements based on any Pig data type.

- ***Example***
- In this example the number of characters in the first field is computed.

- grunt> A = LOAD 'data' as (f1:chararray, f2:chararray, f3:chararray);
  (apache,open,source)
  (hadoop,map,reduce)
  (pig,pig,latin)
- grunt> X = FOREACH A GENERATE SIZE(f1);
- grunt> DUMP X;
  (6L)
  (6L)
  (3L)

# SUM

- Computes the sum of the numeric values in a single-column bag. SUM requires a preceding GROUP ALL statement for global sums and a GROUP BY statement for group sums.
- ***Example***
- In this example the number of pets is computed.

- grunt> A = LOAD ‘/home/Cloudera/data' AS (owner:chararray, pet_type:chararray, pet_num:int);
- grunt> DUMP A;
  (Alice,turtle,1)
  (Alice,goldfish,5)
  (Alice,cat,2)
  (Bob,dog,2)
  (Bob,cat,2)
- grunt> B = GROUP A BY owner;
- grunt> DUMP B;
  (Alice,{(Alice,turtle,1),(Alice,goldfish,5),(Alice,cat,2)})
  (Bob,{(Bob,dog,2),(Bob,cat,2)})
- grunt> X = FOREACH B GENERATE group, SUM(A.pet_num);
- DUMP X;
  (Alice,8L)
  (Bob,4L)

# String Functions

- ENDSWITH
- STARTSWITH
- SUBSTRING
- EqualsIgnoreCase
- UPPER
- LOWER
- REPLACE
- TRIM, RTRIM, LTRIM

# ENDSWITH , STARTSWITH

- ENDSWITH - This function accepts two String parameters, it is used to verify whether the first string ends with the second. string.
- STARTSWITH - This function accepts two string parameters. It verifies whether the first string starts with the second.
- **emp.txt**

emp.txt

- grunt> emp_data = LOAD ‘/home/cloudera/emp.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray);
- grunt> emp_endswith = FOREACH emp_data GENERATE (id,name),ENDSWITH ( name, 'n' );
- grunt> Dump emp_endswith;
- grunt> startswith_data = FOREACH emp_data GENERATE (id,name), STARTSWITH (name,’Ro’);
- grunt> Dump startswith_data;

# SUBSTRING()

- This function returns a substring from the given string.

- EMP.TXT

  001,Robin,22,newyork

  002,Stacy,25,Bhuwaneshwar

  003,Kelly,22,Chennai

- grunt> emp_data = LOAD ‘/home/Cloudera/emp.txt' USING PigStorage(',')as (id:int, name:chararray, age:int, city:chararray);

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

- grunt> Dump substring_data;

  ((1,Robin),Rob)

  ((2,Stacy),Sta)

  ((3,Kelly),Kel)

# EqualsIgnoreCase()

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

emp.txt

- grunt> emp_data = LOAD ‘/home/Cloudera/emp.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray);
- grunt> equals_data = FOREACH emp_data GENERATE (id,name), EqualsIgnoreCase(name, 'Robin');
- grunt>  Dump equals_data;
- ((1,Robin),true)
  ((2,BOB),false)
  ((3,Maya),false)
  ((4,Sara),false)
  ((5,David),false)
  ((6,Maggy),false)
  ((7,Robert),false)
  ((8,Syam),false)
  ((9,Mary),false)
  ((10,Saran),false)
  ((11,Stacy),false)
  ((12,Kelly),false)

# UPPER(), LOWER()

- UPPER- This function is used to convert all the characters in a string to uppercase.
- LOWER- This function is used to convert all the characters in a string to lowercase.

emp.txt

- grunt> emp_data = LOAD '/home/cloudera/emp.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray);

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

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

# REPLACE()

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

emp.txt

- grunt> emp_data = LOAD ‘/home/cloudera/emp.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray);

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

- grunt> Dump replace_data;

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

# TRIM(), RTRIM(), LTRIM()

- The **TRIM()** function accepts a string and returns its copy after removing the unwanted spaces before and after it.
- The function **LTRIM()** is same as the function **TRIM()**. It removes the unwanted spaces from the left side of the given string (heading spaces).
- The function **RTRIM()** is same as the function **TRIM()**. It removes the unwanted spaces from the right side of a given string (tailing spaces).

emp.txt

- grunt> emp_data = LOAD ‘/home/cloudera/emp.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray);
- grunt> trim_data = FOREACH emp_data GENERATE (id,name), TRIM(name);
- grunt> ltrim_data = FOREACH emp_data GENERATE (id,name), LTRIM(name);
- grunt> rtrim_data = FOREACH emp_data GENERATE (id,name), RTRIM(name);
- grunt> Dump trim_data;
- grunt> Dump ltrim_data;
- grunt> Dump rtrim_data;

# Date-time Functions

- ToDate()
- GetDay()
- GetMonth()
- GetYear()

# ToDate()

- This function is used to generate a **DateTime** object according to the given parameters.

- **date.txt**

  001,1989/09/26 09:00:00

  002,1980/06/20 10:22:00

  003,1990/12/19 03:11:44

- grunt> date_data = LOAD ‘/home/cloudera/date.txt' USING PigStorage(',') as (id:int,date:chararray);

- grunt> todate_data = foreach date_data generate ToDate(date,'yyyy/MM/dd HH:mm:ss') as (date_time:DateTime);

- grunt> Dump todate_data;

  (1989-09-26T09:00:00.000+05:30)

  (1980-06-20T10:22:00.000+05:30)

  (1990-12-19T03:11:44.000+05:30)

# GetDay()

- This function accepts a date-time object as a parameter and returns the current day of the given date-time object.

- **date.txt**

  001,1989/09/26 09:00:00

  002,1980/06/20 10:22:00

  003,1990/12/19 03:11:44

# UDF’S

# User Defined Functions

- Apache Pig provides extensive support for **U**ser **D**efined **F**unctions (UDF's).
- Using these UDF's, we can define our own functions and use them.
- The UDF support is provided in six programming languages. Java, Jython, Python, JavaScript, Ruby and Groovy.

# Creating UDF'S

- Open Eclipse and create a new project.
- Convert the newly created project into a Maven project.
- Copy the pom.xml. This file contains the Maven dependencies for Apache Pig and Hadoop-core jar files.

pom.xml

# Java code

```
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
 import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;

public class Sample_Eval extends EvalFunc<String>{
  public String exec(Tuple input) throws IOException {
    if (input == null || input.size() == 0)
    return null;
    String str = (String)input.get(0);
    return str.toUpperCase();
  }
}
```

# Registering the Jar file

- grunt> REGISTER '/home/cloudera/sample_udf.jar';
- grunt> DEFINE Sample_Eval sample_eval();
- grunt> emp_data = LOAD '/home/cloudera/pigdata.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, city:chararray);
- grunt> Upper_case = FOREACH emp_data GENERATE sample_eval(name);