Université Mohammed V- Rabat
Ecole Mohammadia d'Ingénieurs

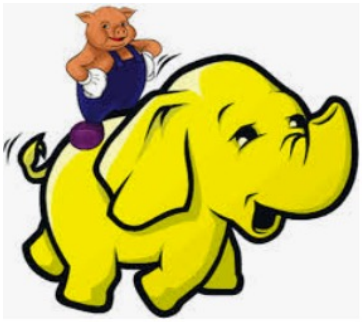
Département Génie Informatique
Filière Génie Informatique et Digitalisation

Fondements du Big Data

Pr. N. EL FADDOULI
nfaddouli@gmail.com

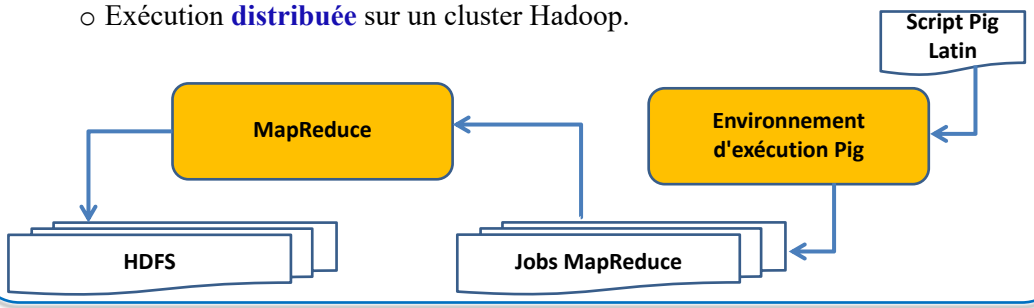
2023-2024

Apache Pig



Pig : Introduction

- Crée par Yahoo
- **Pig** est une **couche d'abstraction** au dessus de Hadoop
- **Pig** est constitué de:
  - Langage **Pig Latin** utilisé pour exprimer des flux de données à obtenir à partir de données en entrée (série d'opérations ou **transformations**).
  - **Environnement d'exécution** des programmes (scripts) Pig Latin:
    - Exécution **locale** dans une seule JVM pour le prototypage *(développer et tester en utilisant des fichiers du système local).*
    - Exécution **distribuée** sur un cluster Hadoop.



97

Pig : Introduction

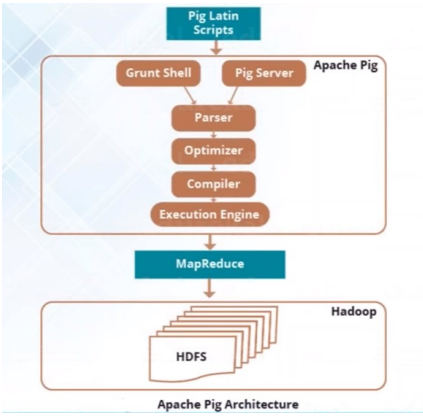
Pig Latin est un ***langage de script*** pour explorer des données de masse.

- Un programme **Pig Latin** est constitué d'une série d'**opérations**, ou **transformations**, appliquées aux données d'entrée pour produire des données de sortie.
- Langage extensible par l'ajout de nouvelles fonction (**UDF**) *(Java, Python (*via Jython une implémentation Python écrite en Java*), JavaScript, Ruby et Groovy)*
- Les opérations décrivent un **flux de données**
- L'environnement d'exécution **Pig** traduit ces opérations en une série de jobs **MapReduce** sur le cluster Hadoop.

98

Pig: Architecture

- **Script Pig Latin**: Contient des commandes **Pig** (transformations) dans un fichier (**.pig**)
- **Grunt Shell**: Shell Pig pour exécuter des commandes pour une exploration interactive des données.
- **Pig Server**: regrouper les commandes Pig dans un fichier et l'envoyer vers ce serveur afin de les exécuter.
- **Parser**: vérifie la syntaxe du script et produit un **DAG** (***Graphe Acyclique Dirigé***) représentant les commandes Pig Latin et les opérateurs logiques. Ces derniers sont représentés comme des nœuds et les flux de données sont représentés comme des arêtes.



Apache Pig Architecture

https://www.tutorialspoint.com/apache_pig/index.htm

- **Optimizer**: Effectue des optimisations logiques en réarrangeant certaines opérations pour de meilleurs performances afin de réduire le flux de données. Il peut être **désactivé** par l'utilisateur.
- **Compiler**: Compile le plan logique optimisé en une série de jobs **MapReduce** (*des tâches **Map** et **Reduce***)
- **Execution engine**: soumet les jobs MapReduce au cluster Hadoop dans un ordre spécifique pour produire les résultats souhaités. Ces derniers sont affichés ou stockés dans un fichier HDFS.

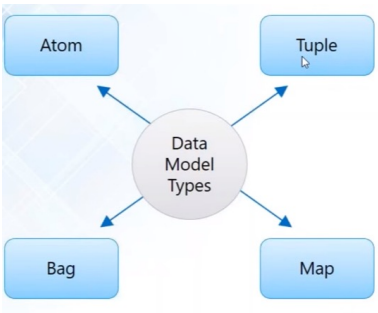
Pig: Modèle de données

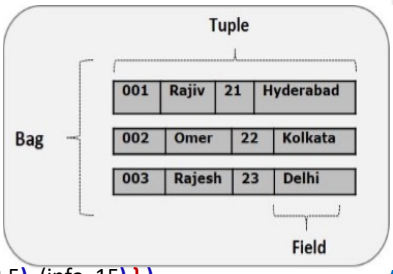
Atom: donnée **élémentaire** qui peut être de plusieurs types. Elle représente un **champ** (*field*) dans un enregistrement par exemple.

Tuple : similaire à une ligne dans une table dans le modèle relationnel et représentée par '**( )**'
Exemple: **(**Rabat, 2008, 3, 60**)**

Bag: Ensemble non ordonné de **tuples** qui peuvent avoir n'importe quel nombre de champs (*schéma flexible*). Un **bag** est mis entre '**{ }**'. Il est similaire à une table dans le modèle relationnel.
Exemple: **{(**Rabat, 2008, 3, 60**) , (**Rabat, 2008, 2, 80**)}**.

Map: une **série** de pairs **clé-valeur**. La **clé** doit être **unique** et de type **chararray**. La valeur peut être de n'importe quel type. Un **Map** est représenté par '**[ ]**' où la clé et la valeur sont séparées par '**#**'
Exemple: **[**nom**#**Ali, âge**#**10**]**





Exemple de tuple complexe: **(** 3 **,** [nom#Faridi, age#23] **, {** (math, 12.5**),** (info, 15**) })**

Pig: Structure d'un script Pig Latin

❑ Un programme Pig est constitué de **3** étapes:

- **LOAD**: pour charger des données à partir de HDFS en leur donnant un **schéma**.
- **TRANSFORM/PROCESS**
  - Opérateurs relationnels: **FILTER**, **ORDER**, **DISTINCT**, **JOIN**, **GROUP**, **FOREACH**, **UNION**, etc.
  - Ces transformations sont traduites en plusieurs tâches **Map/Reduce**
- **DUMP** ou **STORE**
  - Afficher le résultat sur l'écran ou le stocker dans un fichier HDFS

❑ Commentaires: */* commentaire sur plusieurs lignes */* ou *-- commentaire en une ligne*

❑ Types de données de Pig:

- Types simples: **int**, **long**, **float**, **double**, **chararray**, **bytearray**, **boolean**
- Types complexes:
  - **Tuple**: ensemble de champs ordonné (Alifi, 38, Prof)
  - **Collection** de tuples (**Bag**) {(Faridi, 18, Etudiant), (Nasser, 29, Médecin)}
  - **Map** de pairs clé#valeur

[nom # Alifi, Tel#1234567, Email#k.alifi@yahoo.com]

Pig: Structure d'un script Pig Latin

Scénario de traitement dans un script Pig

- Chaque transformation est une application d'un **opérateur relationnel** (*Sélection-Filtrer*, *Jointure*, *Ordre*, *Groupement*, …)
- Chaque transformation est traduite en un job **MapReduce**
- Une transformation peut agir sur plusieurs **Bag** (*Join*, …)

Pig: Lancement de Pig

• Pour lancer le **shell Pig**, on utilise la commande **pig** comme suit dans un terminal:

pig [-x {local | mapreduce}]

Où l'option:

- **local** permet d'utiliser pig en mode local lorsque les fichiers à traiter sont dans le système de fichier local.
- **mapreduce** est utilisée lorsque les fichiers à traiter sont dans un cluster hadoop (*c'est le **mode par défaut***)

• D'autres options sont possibles comme par exemple: **pig -optimizer_off**

Exemple: Lancer pig en mode local où le répertoire courant est ***/home/cloudera/tp_pig***

```
[cloudera@quickstart ~]$ cd /home/cloudera/tp_pig/
[cloudera@quickstart tp_pig]$ pig -x local
grunt>
```

• On peut lancer certaines commandes Linux ou HDFS dans le shell Grunt comme ls, cat, put, …etc. **Exemple**:

```
grunt>fs -ls /user/cloudera
grunt>sh ls
grunt>fs -put temp.csv /user/cloudera/labs
```


Pig: Chargement

bag_résultat = **LOAD** '*Source*' [**USING** *fonction*] [**AS** *schéma*]

• *Source*: nom de ***fichier*** ou ***répertoire*** (*pour charger tous ses fichiers*)

• *Fonction*: nom d'une fonction de chargement des données. Elle désigne une classe Java chargée d'**interpréter** les données depuis le système de fichiers (HDFS ou Local) et d'en sortir des types Pig (*tuples / bag / map*). Il existe dans Pig **plusieurs fonctions** dont celle **par défaut** est: **PigStorage("***Séparateur***")** avec le séparateur par défaut **\t**

• Quelques fonctions**: TextLoader , JsonLoader, …**

• *Schéma*: utilisé pour donner des noms et des types explicites aux différents membres du bag des données d'entrée chargées. On utilise la syntaxe suivante:

(*Champ*$_1$: *Type*$_1$, *Champ*$_2$: *Type*$_2$, ...)

où le type est optionnel (*par défaut c'est **bytearray***)

Il est recommandé de préciser **explicitement** le schéma de chargement avec tous les **types** et **noms** des membres définis

Pig: Chargement

Exemple 1: chargement du fichier **temp.csv**

```
2011;41;3
2011;20;3
2012;31;3
2012;34;3
2012;35;3
```

→ ***R***

```
(2011,41,3)
(2011,20,3)
(2012,31,3)
(2012,34,3)
(2012,35,3)
```

R = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING PigStorage**(';')
AS (year:chararray, temp:int, region:int);

• Accès aux membres du bag: **R.$0** et **R.$2** désignent respectivement **R.year** et **R.region**

Exemple 2: chargement du fichier **Notes.csv**

```
(Faridi,Salah)|(15,15,12)
(BERCHANE,Rachid)|(18,15,20)
(Amina,Saadi)|(16,11,18)
```

→ ***R***

```
((Faridi,Salah),(15,15,12))
((BERCHANE,Rachid),(18,15,20))
((Amina,Saadi),(16,11,18))
```

R = **LOAD** '/user/cloudera/pig_lab/input/Notes.csv' **USING PigStorage**('|')
AS (infos:**tuple**(nom, prenom), **notes**:**tuple**(n1:float,n2:float,n3:float));

• Accès aux membres du bag: **R**.infos.prenom, **R**.notes.n1, …

Pig: Affichage/ Sauvegarde

• Dans le shell Grunt, on peut avoir une description d'un bag avec la commande **DESCRIBE**

Exemple : **R** = **LOAD** '/user/cloudera/pig_lab/input/Notes.csv' **USING PigStorage**('|')
AS (infos:**tuple**(nom, prenom), **notes**:**tuple**(n1:float,n2:float,n3:float));
DESCRIBE R;

On obtient sur écran:

R: {infos: (nom: bytearray,prenom: bytearray), notes: (n1: float,n2: float,n3: float)}

• On affiche les résultats (*données d'un bag ou container*) avec **DUMP**, par exemple: **DUMP R;**

• La commande **STORE** sauvegarde les données sur le système de fichier (HDFS ou local selon le mode d'utilisation de Pig): **STORE** *alias* **INTO** '*repertoire*' [**USING** *fonction*];

Exemple:

R = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING PigStorage**(';')
AS (year:chararray, temp:int, region:int);
STORE R **INTO** '/user/cloudera/pig_lab/**input2**' **USING PigStorage**(',');
STORE R **INTO** '/user/cloudera/pig_lab/**input2**' **USING JsonStorage**();

```
2011,41,3
2011,20,3
2012,31,3
2012,34,3
```

```
{"year":"2011","temp":41,"region":3}
{"year":"2011","temp":20,"region":3}
{"year":"2012","temp":31,"region":3}
{"year":"2012","temp":34,"region":3}
```

Pig: Traitement de données [Filter , Order By]

- L'opérateur **FILTER** permet de filtrer les éléments d'un bag selon une condition:

  DEST = **FILTER** *source* **BY** *expression*;

  Exemple: F = **FILTER** R **BY** temp > 15;

  F = **FILTER** R **BY** temp > 15 **AND** region **IN** (1, 3, 5);

  F = **FILTER** R **BY** year **MATCHES** '2015';

  F = **FILTER** R **BY** year **MATCHES** '.*5';

  R = FILTER M BY nom **MATCHES** '.*Amr.*'

  R = FILTER M BY nom MATCHES '.*(Amr|Ait).*'

- L'opérateur **ORDER** trie les éléments selon une colonne ou plusieurs:

  DEST = **ORDER** *SOURCE* **BY** *champs* [**ASC**|**DESC**];

  Exemple: O = **ORDER** R **BY** temp ASC;

  O = **ORDER** R **BY** year DEC, temp ASC;

Pig: Traitement de données [Group By]

- L'opérateur **GROUP** permet de **grouper** les tuples d'un bag selon un champ:

  destination = **GROUP** *source* **BY** *champ*;

- Il génère un bag (*destination*) dont chaque **tuple** est constitué de **deux champs**:
  - Le premier (**group**), est un champ **simple** dont la **valeur** est celle du **champ de groupement**.
  - Le second est un **bag** portant le **nom de bag** ***source*** sur lequel on a appliqué **GROUP.** Il contient tous les tuples de ***source*** ayant la **même valeur** du champ de groupement.

Exemple 1: **R** = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING PigStorage**(';')
AS (year:chararray **, temp:**int **, region:**int **)**;

(2011, 41, 3)
(2011, 20, 3)
(2012, 20, 4)
(2012, 31, 3)
(2013, 24, 4)
(2011, 22, 5)
(2010, 16, 5)
.....

G

(3, {(2011,41,3), (2011,20,3), (2012,31,3), ...})
(4, {(2012, 20, 4), (2013, 24, 4), ...})
(5, {(2011, 22, 5), (2010, 16, 5), ...})

group R

G = **GROUP** R **BY region**;

DESCRIBE G; ⇒ On obtient: **G:** {**group**: **int** , **R**:{(**year**:chararray,**temp**:int,**region**:int)}}

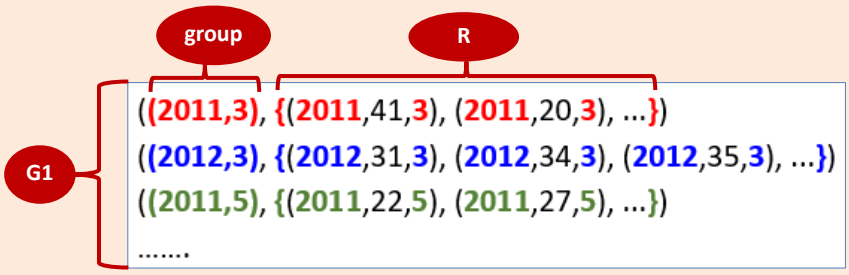
Pig: Traitement de données [Group By]

Exemple 2: **R** = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING PigStorage**(';')
AS (year:chararray **, temp:**int **, region:**int **)**;

```
(2011,41,3)
(2011,20,3)
(2012,31,3)
(2012,34,3)
(2012,35,3)
```

G1 = **GROUP** R **BY (year , region)**; -- *groupement par deux champs*
DESCRIBE G1;

On obtient: **G1:** {**group**: **(chararray, int)**, **R**:{(**year**:chararray,**temp**:int,**region**:int)}}



Pig: Traitement de données [Group By]

- L'opérateur **GROUP** peut être utilisé pour que tous les tuples d'un bag se retrouvent dans un seul groupe; par exemple, lors de l'agrégation de relations (bag) entières:

  destination = **GROUP** *source* **ALL**;

- Il génère un bag contenant **un seul tuple** composé de **deux champs**:
  - Le premier (**group**) est de type **chararray** dont la valeur est "**all**".
  - Le second est un **bag** portant le nom du bag ***source***. Il contient **tous les tuples** de ***source***.

Exemple 1: **R** = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING PigStorage**(';')
AS (year:chararray **, temp:**int **, region:**int **)**;

G = **GROUP** R **ALL**;
Dump G;

```
(2011, 41, 3)
(2011, 20, 3)
(2012, 20, 4)
(2012, 31, 3)
(2013, 24, 4)
(2011, 22, 5)
(2010, 16, 5)
.....
```

(all, {(2011,41,3), (2011,20,3), (2012,20,4), (2012,11,3), ...})

DESCRIBE G;

⇒ On obtient: **G:** {**group**:**chararray** , **R**:{(**year**:chararray,**temp**:int, **region**:int)}}

Pig: Traitement de données [Foreach…Generate]

- L'opérateur **FOREACH ... GENERATE** permet de parcourir les tuples d'un bag et de générer d'autres tuples: *destination* = **FOREACH** *source* **GENERATE** *expression*;
- Dans ***expression*** on peut générer des champs de types **atom**, **tuples**, **map** et/ou des **bag** et appeler des **fonctions** à la volée.
- **Exemple: Déterminer la température maximale par année pour les régions 1, 3 et 4**

R = **LOAD** '/user/cloudera/pig_lab/input/temp.csv'
USING PigStorage(';')
AS (year:chararray, temp:int, region:int);
FR = **FILTER** *R* **BY** temp **!=** 999 **AND** region **IN** (1, 3, 4);
GR = **GROUP** *FR* **BY** year;

FR
(2011, 41, 3)
(2011, 20, 3)
(2012, 20, 4)
(2012, 31, 3)
(2013, 24, 4)
.....

R
(2011, 41, 3)
(2011, 20, 3)
(2012, 20, 4)
(2012, 31, 3)
(2013, 24, 4)
(2011, 22, 5)
(2010, 16, 5)
.....

GR
group | FR
(2011 , {(2011,41,3),(2011,20,3),..})
(2012 , {(2012,20,4),(2012,31,3),...})
(2013 , {(2013,24,4),...})
$0

MaxT= **FOREACH** *GR* **GENERATE** *group* **AS** *year*, **MAX**(**FR**.temp) **AS** *temp_max*;
DESCRIBE *MaxT*; ⟹ On obtient: **MaxT:** {**year**: **chararray** , **temp_max**: **int** }

Pig: Traitement de données [Foreach…Generate]

L'opérateur **FOREACH** peut être utilisé de plusieurs façons:

1. **Extraire** certains champs de chaque tuple d'un bag.

Exemple: ***Extraction des champs year et region***

```
grunt> R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(';')
       AS (year:chararray, temp:int, region:int);
grunt> R1 = FOREACH R GENERATE year , region;   -- sélection de year et region
grunt> R2 = DISTINCT R1; -- R2 contiendra les tuples de R1 sans doublons
grunt> R3 = ORDER R2 BY year , region ;
grunt> DUMP R3;
```

2. **Réorganiser** les champs d'un bag.

Exemple: ***Réorganisation des champs***

```
grunt> R = LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(';')
       AS (year:chararray , temp:int , region:int);
grunt> R1 = FOREACH R GENERATE year , (temp, region) AS details;
grunt> DESCRIBE R1;
```

R1: {year: chararray , **details**: (temp: int, region: int)}

Pig: Traitement de données [Foreach…Generate]

grunt> R = **LOAD** '/user/cloudera/pig_lab/input/**info.csv**' **USING** PigStorage(';') **AS** (**code**:int , **details**: **map[ ]**);

```
(10,[prenom#saad,nom#ali])
(20,[prenom#amina,nom#salmi])
(30,[prenom#rayane,nom#khaldi])
(40,[prenom#walid,nom#farok])
```

R

```
10;[nom#ali,prenom#saad]
20;[nom#salmi,prenom#amina]
30;[nom#khaldi,prenom#rayane]
40;[nom#farok,prenom#walid]
```

grunt> R1 = **FOREACH** R **GENERATE** code, **details#'nom'** as **firstname** , **details#'prenom'** as **lastname**;

grunt> **DESCRIBE** R1;

```
R1: {code: int,firstname: bytearray,lastname: bytearray}
```

grunt> R1 = **FOREACH** R **GENERATE** code, **(chararray) details#'nom'** as **firstname** , **(chararray) details#'prenom'** as **lastname**;

grunt> **DESCRIBE** R1;

```
R1: {code: int,firstname: chararray,lastname: chararray}
```

```
(10,ali,saad)
(20,salmi,amina)
(30,khaldi,rayane)
(40,farok,walid)
```

R1

Pig: Traitement de données [Foreach…Generate]

3. Faire des **calculs** sur les champs de **chaque tuple** d'un bag via des opérateurs arithmétiques et des fonctions prédéfinies (**MAX**, **MIN**, **AVG**, **COUNT**, …)

Exemple: Calcul arithmétique

grunt> R = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING** PigStorage(';') **AS** (year:chararray, temp:int, region:int);

grunt> R1 = **FOREACH** R **GENERATE** year , **temp+2** **AS** **tempplus2**;

grunt> **DESCRIBE** R1; **R1:** {year: chararray, **tempplus2** : int}

Exemple: La moyenne de température par année et region.

grunt> R = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING** **PigStorage**(';') **AS** (year:chararray, temp:int, region:int);

grunt> FR = **FILTER** R **BY** temp != 999;

grunt> GR = **GROUP** FR **BY** **(year, region);**

GR: {group: (year: chararray, region: int), FR: {(year: chararray, temp: int, region: int)}}

grunt> M= **FOREACH** GR **GENERATE** group **AS** **year_region,** **AVG** (FR.**temp**) **AS** **moyenne**;

M: {**year_region**: (year: chararray, region: int), **moyenne**: double}

Pig: Traitement de données [Imbrication d'opérations]

• L'opérateur **FOREACH** permet d'effectuer **plusieurs traitements** sur le **tuple courant** avant de **générer un tuple** en sortie: *destination* = **FOREACH** *source* { *opération*$_1$;
...
opération$_n$;
GENERATE *expression*;
}

R

```
(2011,23,3)
(2011,17,9)
(2013,23,9)
(2011,20,3)
(2013,17,9)
.....
```

Exemple:

```
R = LOAD  '/user/cloudera/tp2/input/temp.csv'
      USING  PigStrorage (';')
       AS (year : int, temp: int, region: int);
G = GROUP  R BY year;
DESCRIBE G ;
```

G: {**group**: int, **R**: {(**year** : int, **temp**: int, **region**: int)}}

G:
(**2011**, {(**2011**, 23, 3),(**2011**,17,9),(**2011**,20,3)})
(**2013**, {(**2013**, 23, 9),(**2013**,17,9)})
.....

F:
(**2011**, {(3),(9)})
(**2013**, {(9)})
.....

```
F  = FOREACH  G {
                    LC1 = FOREACH  R GENERATE  region;
                    LC2 = DISTINCT  LC1;
                    GENERATE  group,  LC2;
                    };
DUMP  F;
```


Pig: Utilisation de Pig en ligne de commandes

1. Créer le dossier local: **/home/cloudera/pig_lab/input/**
   Copier le fichier **temp1.csv** dans ce dossier.
2. Lancer Pig en **mode local** avec la commande pig depuis le dossier courant crée dans la question 1: .....]$ **cd /home/cloudera/pig_lab/input**
   ...**input**]$ **pig -x local**
   On obtiendra le shell pig: **grunt>**
3. Charger un fichier:
   grunt> *R* = **LOAD** 'temp1.csv' **USING PigStorage**(';')
   AS (***year***:chararray, ***temp***:int, ***region***:int);

R est un bag (son contenu est temporaire) sur lequel on peut appliquer des commandes (describe, dump, ...)

4. Appliquer des commandes à l'alias (bag) obtenu par **Load**:
   grunt> DESCRIBE *R*
   R: {year: chararray,temp: int, region: int}
   grunt> DUMP *R*;
   (2011,17,4)
   (2011,14,4)
   .....

Pour avoir un extrait (1%) aléatoire des tuples de R:
grunt> e= sample R 0.01;
grunt> Dump e;

Pig: Utilisation de Pig en ligne de commandes

5. Filtrer des tuples à partir du résultat de **Load**:

grunt> *FR* = **FILTER** *R* **BY** temp != 999 **AND** region **IN** (1, 4, 9);

grunt> DUMP *FR*;

….

(2014,7,9)

….

Pour avoir juste les 5 premiers tuples de FR:
grunt> L= Limit FR 5;
grunt> Dump L;

6. Faire un groupement de tuples:

grunt> GR = GROUP *FR* **BY** year;

grunt> describe *GR*;

GR: {**group**: chararray, ***FR*** : {year: chararray, temp: int, region: int}}

grunt> DUMP *GR*;

(2011, {(2011,32,1),(2011,251), …})

…..

Pig: Utilisation de Pig en ligne de commandes

7. Appliquer un traitement à chaque groupe (**tuple de GR**) pour avoir la température maximale:

grunt> *MaxT* = **FOREACH** *GR* **GENERATE group, MAX** (*FR*.temp);

grunt> DUMP *MaxT* ;

(2011,71)

(2012,50)

….

grunt> STORE *MaxT* **INTO** 'res_max/' ;

grunt> sh cat res_max/* ;

On peut laisser le séparateur des champs par défaut \t
ou préciser un séparateur explicite, par exemple ';' :
grunt> STORE *MaxT* INTO 'res_max/' using PigStorage(';') ;

On peut aussi préciser le format du fichier, par exemple le format Json:
grunt> STORE *MaxT* INTO 'res_max/' using JsonStorage( ) ;

8. Dans le shell de Linux: $ **ls** ***res_max/***
9. Dans le shell de Linux: $ **cat** ***res_max/****
10. Pour obtenir le **nombre** de températures par année:

grunt> *NbT* = **FOREACH** *GR* **GENERATE group**, **COUNT**(***FR***) **AS** *Nb*;

11. Débugger la séquence d'instructions pour avoir le contenu d'un bag (*MaxT*)

grunt> ILLUSTRATE *MaxT* ;

12. On peut regrouper toutes les transformations dans un script **.pig** qu'on peut lancer avec:

- la commande **pig** dans le shell Linux par exemple: **$ pig temp.pig**
- les commandes **exec** ou **run** dans le shell Pig: **grunt> exec temp.pig**

Pig: Traitement de données - EXERCICE

Soit **bank.csv** un fichier **csv** dont chaque ligne contient *l'identifiant d'une banque*, *l'identifiant d'un client*, *le montant annuel versé*, *le montant annuel retiré* et *l'année*, par exemple: **ma123; 1293; 100530; 89000; 2019**

Ecrire des scripts **Pig** pour avoir:

1. **Les id des clients débiteurs au moins une fois (*somme versée < somme retirée*)**
2. **Le nombre de clients par banque.**
3. **Le nombre de clients débiteurs par banque**
4. **Les clients débiteurs pour l'ensemble des opérations (*somme des versements < somme des retraits*)**
5. **Les banques ayant au moins un débiteur sur l'ensemble de ses opérations.**
6. **Des tuples dont chacun contient deux champs: l'identifiant d'une banque (atom) et la liste des identifiants de ses clients (bag)**

Pig: Traitement de données [Join]

• L'opérateur **JOIN** permet de faire des jointures entre deux bags ou plus:

$bag_{résultat}$ = **JOIN** *bag1* **BY** *champ1* , *bag2* **BY** *champ2* *[,bag3* **BY** *champ3, ...];*

$bag_{résultat}$ contiendra des tuples dont chacun est la concaténation d'un tuple de *bag1* et d'un autre de *bag2 (un de bag3, ...)* **s'ils ont la même valeur de *champ1* et *champ2 (champ3, ...)*.**

• Jointure sur plusieurs champs : *bag3* = **JOIN** *bag1* **BY** *(ch11, ch12)* , *bag2* **BY** *(ch21, ch22)* ;

• **Exemple: temp.csv** (year, temp, **region**)

region.csv (**region**, longitude, latitude, rayon)

```
(2011,32,1)
(2013,24,3)
(2015,19,4)
(2011,14,5)
```
R1

```
(1,12983,-12265,100)
(3,52983,-39102,70)
(4,42983,-19285,120)
(6,74927,-29837,90)
```
R2

grunt> R1 = **LOAD** '/user/cloudera/pig_lab/input/temp.csv' **USING** PigStorage(';') **AS** (**year**:int , **temp**:int , **region**:int);

grunt> R2 = **LOAD** '/user/cloudera/pig_lab/input/region.csv' **USING** PigStorage(';') **AS** (**region**:int , **longitude**:int , **latitude**:int , **rayon**:int);

grunt> R3 = **JOIN** *R1* **BY** *region*, *R2* **BY** *region;*

R3: {**R1::**year: int , **R1::**temp: int, **R1::**region: int , **R2::**region, **R2::**longitude: int , **R2::**latitude: int , **R2::**rayon;int}

grunt> R4 = **FOREACH** R3 **GENERATE R1::** year , **R1::**temp, **R2::**longitude, **R2::**latitude;

Pig: Traitement de données [Join]

• L'opérateur **JOIN** permet de faire des jointure **externes** entre deux bags ou plus:

bag$_{résultat}$ = **JOIN** *left_bag* **BY** *champ1* **[LEFT|RIGHT|FULL] [OUTER]** , *right_bag* **BY** *champ2;*

bag$_{résultat}$ contiendra des tuples issus d'une jointure externe (*comme en SQL*) entre les tuples de *left_bag* et *right_bag* .

R1
```
(2011,32,1)
(2013,24,3)
(2015,19,4)
(2011,14,5)
```

```
(1,12983,-12265,100)
(3,52983,-39102,70)
(4,42983,-19285,120)
(6,74927,-29837,90)
```
R2

• **Exemple:**

temp.csv (year, temp, **region**)

region.csv (**region**, longitude, latitude, rayon)

grunt> R1 = **LOAD** '/user/cloudera/pig_lab/input/temp.csv'
USING PigStorage(';') **AS** (**year**:chararray , **temp**:int , **region**:int);

grunt> R2 = **LOAD** '/user/cloudera/pig_lab/input/region.csv'
USING PigStorage(';') **AS** (**region**:int , **longitude**:int , **latitude**:int , **rayon**:int);

grunt> R3 = **JOIN** *R1* **BY** *region* **LEFT** , *R2* **BY** *region;*

R3: { R1::year:int , R1::temp:int , R1::region:int, R2::region:int, R2::latitude:int, R2::longitude:int, R2::rayon:int}

```
(2011,32,1,1,12983,-12265,100)
(2013,24,3,3,52983,-39102,70)
(2015,19,4,4,42983,-19285,120)
(2011,14,5,,,,)
```


Pig: Traitement de données [Join]

• **Exemple:**

temp.csv (year, temp, **region**)

region.csv (**region**, longitude, latitude, rayon)

grunt> R1 = **LOAD** '/user/cloudera/pig_lab/input/temp.csv'
USING PigStorage(';') **AS** (**year**:chararray , **temp**:int , **region:**int);

```
(2011,32,1)
(2013,24,3)
(2015,19,4)
(2011,14,5)
```
R1

grunt> R2 = **LOAD** '/user/cloudera/pig_lab/input/region.csv'
USING PigStorage(';') **AS** (**region**:int , **longitude**:int , **latitude**:int , **rayon**:int);

grunt> R3 = **JOIN** *R1* **BY** *region* **RIGHT** , *R2* **BY** *region;*

```
(2011,32,1,1,12983,-12265,100)
(2013,24,3,3,52983,-39102,70)
(2015,19,4,4,42983,-19285,120)
(,,,6,74927,-29837,90)
```

```
(1,12983,-12265,100)
(3,52983,-39102,70)
(4,42983,-19285,120)
(6,74927,-29837,90)
```
R2

grunt> R4 = **FILTER** *R3* **BY** R1::*region* **is not null**;

```
(2011,32,1,1,12983,-12265,100)
(2013,24,3,3,52983,-39102,70)
(2015,19,4,4,42983,-19285,120)
```


Pig: Traitement de données [Join]

• **Exemple:**
temp.csv (year, temp, **region**)
region.csv (**region**, longitude, latitude, rayon)

R1
```
(2011,32,1)
(2013,24,3)
(2015,19,4)
(2011,14,5)
```

R2
```
(1,12983,-12265,100)
(3,52983,-39102,70)
(4,42983,-19285,120)
(6,74927,-29837,90)
```

```
grunt> R1 = LOAD '/user/cloudera/pig_lab/input/temp.csv'
         USING PigStorage(';') AS (year:chararray , temp:int , region:int);
grunt> R2 = LOAD '/user/cloudera/pig_lab/input/region.csv'
         USING PigStorage(';') AS (region:int , longitude:int , latitude:int , rayon:int);
grunt> R3 = JOIN R1 BY region FULL , R2 BY region;
```

R3: { R1::year:int , R1::temp:int , R1::region:int, R2::region:int, R2::latitude:int, R2::longitude:int, R2::rayon:int}

```
(2011,32,1,1,12983,-12265,100)
(2013,24,3,3,52983,-39102,70)
(2015,19,4,4,42983,-19285,120)
(2011,14,5,,,,)
(,,,6,74927,-29837,90)
```

• **Auto-jointure:**

Res
```
Fatah,Amine
FaTah,Amina
Khalid,Saad
```

```
R1 = LOAD '/user/cloudera/pig_lab/input/emp.csv'
         USING PigStorage(';') AS (id:int , name:chararray , mgr:int);
R2 = LOAD '/user/cloudera/pig_lab/input/emp.csv'
         USING PigStorage(';') AS (id:int , name:chararray , mgr:int);
E = JOIN R1 BY id, R2 BY mgr;
Res = FOREACH E GENERATE R1::name , R2::name ;
```

```
(1,Amine, 5)
(2,Saad,3)
(3,Khalid,)
(4,Amina,5)
(5,Fatah,)
```

R2 R1

Nom d'un manager , nom de subordonné

Pig: Traitement de données [Cross]

• L'opérateur **CROSS** permet de faire un produit cartésien entre deux bags:

bag3 = **CROSS** *bag1* , *bag2;*

bag3 contiendra des tuples dont chacun est composé d'un tuple de *bag1* et d'un autre de *bag2*

• Exemple: **users.csv (id, nom, nbquestion, nbreponse)**

```
grunt> R1 = LOAD '/user/cloudera/pig_lab/input/users.csv' USING PigStorage(';')
         AS (id:long, nom:chararray, nbq:int , nbr:int);
grunt> R2 = LOAD '/user/cloudera/pig_lab/input/users.csv' USING PigStorage(';')
         AS (id:long, nom:chararray, nbq:int , nbr:int);
grunt> R3= CROSS R1, R2 ;
```

R3: { R1::id: int,R1::nom: chararay, R1::nbq: int, R1::nbr: int, R2::id: int,R2::nom: chararay, R2::nbq: int, R2::nbr: int }

```
grunt> R4 = FILTER R3 BY R1::nbq == R2::nbq and R1::id != R2::id ;
grunt> R5 = FOREACH R4 GENERATE R1::nom As nom1, R2::nom As nom2, R1::nbq As nbq;
```

Pairs d'utilisateurs ayant posé le même nombre de questions

Pig: Opérateurs arithmétiques et autres

- **Opérateurs arithmétiques**: +, -, *, /, %
- **Opérateur de Cast**: (*type*) *champ*

Exemple:

```
R= LOAD '/user/cloudera/pig_lab/input/temp.csv' USING PigStorage(';')
                                                AS (year , temp:int , region:int);
Describe R; ⇒ R:{year: bytearray, temp: int, region:int}
T = Foreach R Generate (int) year, temp;
Describe T; ⇒ T:{year: int, temp: int}
```

- **Opérateur ternaire de sélection**: (*condition* ? *val_si_true* : *val_si_false*)

  Exemple: **R:{year:int, temp:int, region:int}**

```
t = Foreach R Generate year, temp, (temp>20 ? 'CHAUD' : 'BEAU TEMPS' );
t = Foreach R Generate year,temp,(temp>20 ? 'CHAUD' :(temp>10?'BEAU TEMPS' : 'FROID') );
```


Pig: Opérateur de sélection multiple

- Opérateur de sélection multiple:

```
CASE
        WHEN condition1 THEN resultat1
        WHEN condition2 THEN resultat2
        ....
        WHEN conditionn THEN resultatn
        ELSE resultat
END
```

```
CASE expression
        WHEN val1 THEN resultat1
        WHEN val2 THEN resultat2
        ....
        WHEN valn THEN resultatn
        ELSE resultat
END
```

Exemple: **R:{year:int, temp:int, region:int}**

```
T = Foreach R Generate year, temp, (Case  When temp>20 Then  'CHAUD'
                                          When temp>10 Then 'BEAU TEMPS'
                                          Else  'FROID'
                                   End)  AS  climat;
```

T: { year: int, temp: int, climat: chararray}

```
T= Foreach R Generate year, temp, (Case  region
                                   When 1 Then  'RABAT'
                                   When 2 Then 'CASABLANCA'
                                   When 3 Then 'FES'
                                   Else  'AUTRE'
                                   End) AS  region;
```

T: { year: int, temp: int, region: chararray}

Pig: L'opérateur flatten (ou la fonction FLATTEN)

- L'opérateur **flatten** modifie la structure d'un champ des tuples d'un bag.
- Il permet de mettre à **plat un champ** qui est d'un type **complexe** (***tuple***, ***bag***, ***map***).
- Pour un champ de type **tuple**, **flatten** le remplace par ses champs.

Exemple: Pour un bag **R: {a:int, t: (b:int, c:int)}**

```
(2020,(2,9))
(2020,(3,5))
(2019,(4,8))
(2019,(6,7))
```

F= F= **Foreach** R **Generate** a, **flatten** (**t**); -- ou **Foreach** R **Generate** $0, **flatten** (**$1**);
⇒ F:{a: int, **t::**b:int, **t::**c:int}

F= **Foreach** R **Generate** $0, **flatten** (**$1**) AS (**b,c**);
⇒ F:{a: int, **b**:int, **c**:int}

```
(2020,2,9)
(2020,3,5)
(2019,4,8)
(2019,6,7)
```


Pig: L'opérateur flatten (ou la fonction FLATTEN)

- Pour un champ de type **map**, **flatten** transforme chaque pair clé-valeur en un tuple dont les champs sont la clé et la valeur.

Exemple: Pour un bag **R: {code:int, details: [ ]}**

R

```
(10,[nom#amini,prenom#anas])
(20,[nom#gandi,prenom#laila])
```

F= **Foreach** R **Generate** **flatten** (**details**);
Dump F;
Describe F; ⇒ F: {**details::key**: chararray, **details::value**: chararray}

F

```
(nom, amini)
(prenom, anas)
(nom, gandi)
(prenom, laila)
```

F= **Foreach** R **Generate** **flatten** (**details**) AS (**cle**, **valeur**);
Describe F; ⇒ F: {**cle**: chararray, **valeur**: chararray}

F= **Foreach** R **Generate** code, **flatten** (**details**) AS (**cle**, **valeur**);
Dump F;
Describe F; ⇒ F: {**code**: int, **cle**: chararray, **valeur**: chararray}

F

```
(10, nom, amini)
(10, prenom, anas)
(20, nom, gandi)
(20, prenom, laila)
```

Pig: L'opérateur flatten (⇔ la fonction FLATTEN)

- Pour un champ de type **bag**, l'opérateur **flatten** permet de créer de **nouveaux tuples** à partir du tuple contenant le champ sur lequel on applique **flatten**.

Exemple: Pour un bag **R:{ a: int, b: {(c:int, d:int)} }** dont le champ **b** est un bag

```
(2019,{(6,7),(4,8)})
(2020,{(3,5),(2,9)})
```

F= F= **Foreach** R **Generate** a, **flatten** (**b**); -- ou **Foreach** R **Generate** $0, **flatten** (**$1**);

⇒ F:{a: int, **b::**c:int, **b::**d:int}

```
(2019,6,7)
(2019,4,8)
(2020,3,5)
(2020,2,9)
```

F= **Foreach** R **Generate** $0, **flatten** (**$1**) AS **(c,d)**;

⇒ F:{a: int, **c**:int, **d**:int}

N.B: Lorsqu'on supprime un niveau d'imbrication dans un bag ou un map, on provoque un produit croisé.

Pig: Fonctions sur chaînes de caractères

- chararray **CONCAT**(chararray c1, chararray c2)
- int **INDEXOF**(chararray ***source***, chararray ***character,*** int ***startIndex***)
- int **LAST_INDEX_OF**(chararray ***source***, chararray ***character***)
- chararray **UPPER**(chararray input)
- chararray **LOWER**(chararray input)
- chararray **REPLACE**(chararray source, chararray toReplace, chararray newValue)
- long **SIZE**(chararray input)
- (chararray) **STRSPLIT**(chararray source, chararray regex)
- chararray **SUBSTRING**(chararray source, int start, int end)
- {(chararray)} **TOKENIZE**(chararray input [, 'délimiteur'])
- …

STRSPLIT: **Source**: 'Formation-Big#Data et BI'
Regex: '[-\\s#]'
➔ Résultat: (Formation, Big, Data, et, BI)

TOKENIZE: **Source**: 'Formation_Big_Data_BI'
Délimiteur: '_'
➔ Résultat: { (Formation), (Big), (Data), (BI) }

Pig: La fonction TOKENIZE

• Le fonction **TOKENIZE** divise une chaîne de caractère en mot dont chacun sera placé dans son propre **tuple** et tous les tuples seront placés dans un **bag**.

Exemple: Fichier maman.txt

```
Quand il faisait nuit, tu étais là
Quand il faisait froid, tu étais là
Quand j'avais faim, Quand j'avais peur
Quand je ne croyais qu'au malheur, tu étais là.
Dans tes yeux, Maman, le ciel est toujours bleu
```

R= **Load** 'maman.txt' **Using TextLoader() AS** (L : chararray) ;

```
(Quand il faisait nuit, tu étais là)
(Quand il faisait froid, tu étais là)
(Quand j'avais faim, Quand j'avais peur)
(Quand je ne croyais qu'au malheur, tu étais là)
```

F= **Foreach** R **Generate TOKENIZE(L) AS** m ;

⇒ **F: { m**: { tuple_of_tokens : (token: chararray) } }

```
({(Quand),(il),(faisait),(nuit),(tu),(étais),(là)})
({(Quand),(il),(faisait),(froid),(tu),(étais),(là)})
({(Quand),(j'avais),(faim),(Quand),(j'avais),(peur)})
({(Quand),(je),(ne),(croyais),(qu'au),(malheur),(tu),(étais),(là)})
```


Pig: Fonctions Mathématiques

- double **ABS**(double input)
- double **SQRT**(double input)
- int **AVG**({(int)} input)
- long **COUNT**({} input)
- int **MAX**({(int)} input)
- int **MIN**({(int)} input)
- long **SUM**({(int)} input)
- …

Pig: Fonctions sur des types complexes

(2011 , 23 , 4)
(2012 , 15 , 3)
(2009 , 35 , 9)
....
R

```
Grunt> describe R;
R: {a: int, t:int, c:int}
Grunt> B= foreach R Generate TOBAG(a, t, c);
Grunt> dump B;
```

B

({ (2011) , (23) ,(4)})
({ (2012) , (15) ,(3)})
({ (2009) , (35) ,(9)})

- long **SIZE**(**map** *input*)
- long **SIZE**(**tuple** *input*)
- long **SIZE**(**bag** *input*)
- bag **TOBAG**(*expression [, expression ...]*)
- map **TOMAP**(*key, value [, key, value ...]*)
- tuple **TOTUPLE**(*expression [, expression ...]*)
- bag **TOP**(int *n* , int *m* , **bag** *source*)

  Retourne un bag contenant les ***n*** premiers tuples du bag ***source*** trié selon le champ de rang ***m***.
- …..

Pig: Exécution et paramétrage d'un script pig

- Pour lancer un script pig:

exec [[**-param** *param_name = param_value*] [...]] *file*

run [[**-param** *param_name = param_value*] [...]] *file*

N.B: Avec **run** les bags sont accessibles dans le shell **Grunt**.

Exemple:

Fichier pig

temp_moy_year.pig

```
/*
Ce script permet d'avoir la moyenne des températures enregistrées durant
l'année fournie en paramère  $annee
*/
R= Load 'temp.csv' Using PigStorage(';') As (year:int, temp:int, cat:int);
F= Filter R By year==$annee;
G= Group F By year;
O= Foreach G Generate group As Year, AVG(F.temp) As Temp_Moyenne;
Dump O;
```

Exécution

```
grunt> exec -param annee=2011 temp_moy_year.pig
```

Résultat

```
(2011,31.301217884551217)
grunt>
```

Pig: Exécution et paramétrage d'un script pig

- **Exemple**:

Fichier pig

temp_moy_year_cat.pig

```
/*
Ce script permet d'avoir la moyenne des températures enregistrées durant l'année
$annee pour la catégorie $categorie fournies comme paramètres
*/
R= Load 'temp.csv' Using PigStorage(';') As (year:int, temp:int, cat:int);
F= Filter R By year==$annee and  cat==$categorie;
G= Group F By (year,cat);
O= Foreach G Generate flatten(group) As (year, cat), AVG(F.temp) As Temp_Moyenne;
Dump O;
```

Exécution

```
grunt> run  -param  annee=2011 -param categorie=3  temp_moy_year_cat.pig
```

Résultat et structure du bag O

```
(2011,3,32.958955223880594)
grunt> describe  O;
O: {year: int,cat: int,Temp_Moyenne: double}
```


Atelier 3: Utilisation de Pig

Le fichier ***vol.csv*** contient les colonnes suivantes séparées par ";":

Année de vol , *mois du vol*, *jour du vol*, *n° de vol*, *aéroport de départ*, *aéroport d'arrivée* et *distance du vol*

Copier ces fichiers dans le dossier local: /home/cloudera/pig_lab/input

1. Ecrire un script Pig pour calculer le nombre de vols en partance d'un aéroport donné en paramètre. Le fichier de données est fourni lui aussi en deuxième paramètre.
2. Ecrire un script Pig pour calculer le nombre de vol en partance de chaque aéroport.
3. Ecrire un script Pig pour calculer la distance totale parcourue par tous les vols.

Atelier 3: Utilisation de Pig

4. Ecrire un script Pig pour calculer le nombre de vols en partance d'un aéroport donné et pour une année donnés en paramètre. Le fichier de données est fourni lui aussi en troisième paramètre.
5. Ecrire un script Pig pour calculer le nombre de vol à destination de chaque aéroport.
6. Ecrire un script Pig pour calculer la distance maximale parcourue.

Références Pig

Site Officiel Apache:

http://pig.apache.org/docs/r0.12.0/

http://pig.apache.org/docs/r0.12.0/basic.html

Tutorial:

https://www.tutorialspoint.com/apache_pig/index.htm

