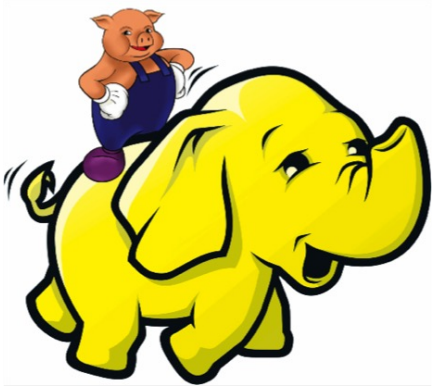
ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Base de Données Avancées (Ecosystème Big Data)
(4IIR — Spring 2025)

Chapitre 06 :



Couche d'Abstraction – Apache Pig

Une approche simplifiée pour le traitement de données massives…

Dr. Ghezlane Halhoul Merabet
G.Halhoulmerabet@emsi.ma

Plan du Chapitre

1. Les **Défis** du Traitement de Données avec **MapReduce**
2. **Apache Pig :** Une Solution de Haut Niveau aux **défis** de **MapReduce**
3. **Pig :**
   - Architecture
   - Modèle de données
   - Environnement d'Exécution
   - Structure Fondamentale d'un script Pig Latin
   - Opérateurs Essentiels d'un Script Pig Latin
   - Opérateurs de Traitement de Données
   - Fonctionnalités Avancées
   - Exécution et Paramétrage d'un Script.pig

Les Défis du Traitement de Données avec MapReduce

Le code **WordCount.java** (utilisé en **TP-02 : MapReduce**) :

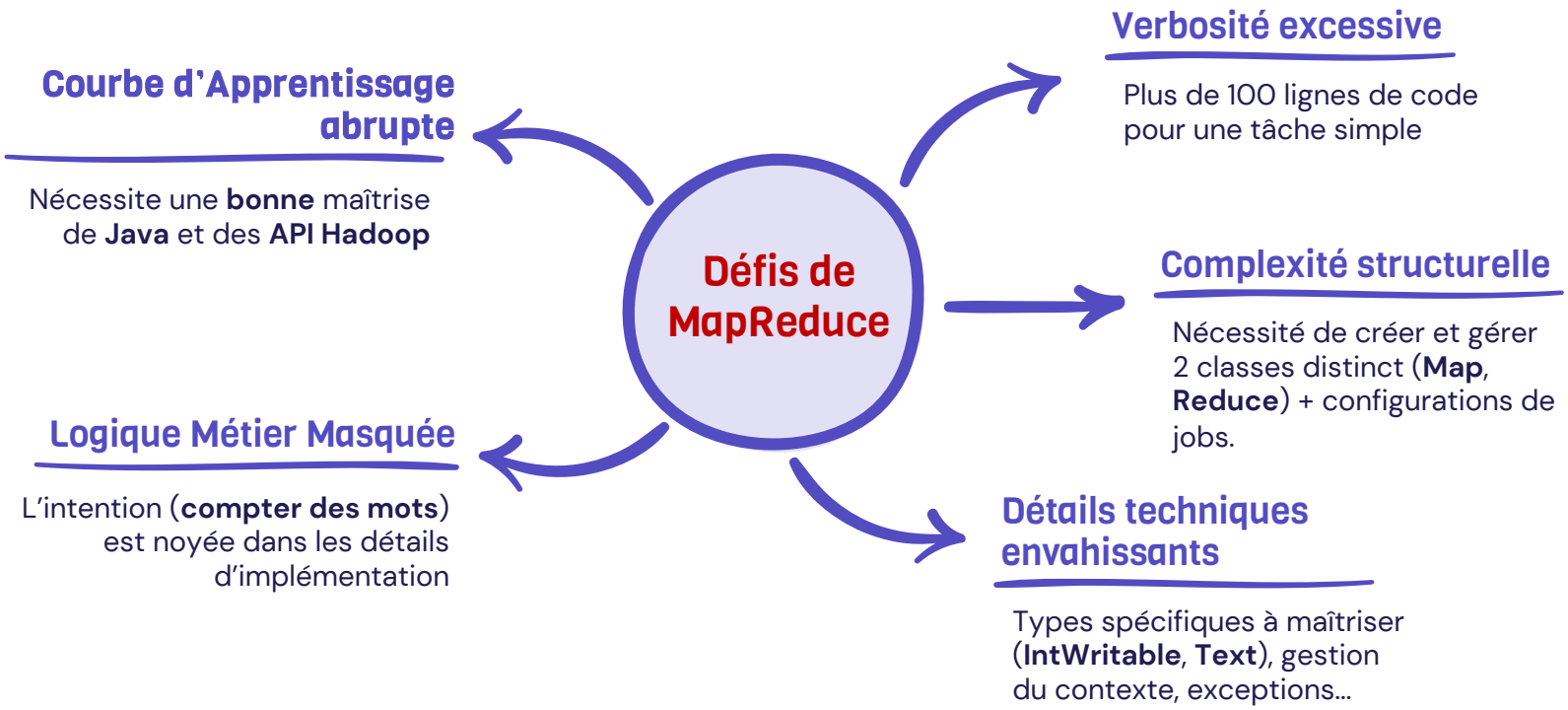
```java
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable offset, Text lineText, Context context)
        throws IOException, InterruptedException {
      String line = lineText.toString();
      if (!caseSensitive) {
        line = line.toLowerCase();
      }
      Text currentWord = new Text();
      for (String word : WORD_BOUNDARY.split
        if (word.isEmpty() || patternsToSkip
            continue;
        }
        currentWord = new Text(word);
        context.write(currentWord, one);
      }
    }
}
```

```java
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text word, Iterable<IntWritable> counts, Context context)
        throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable count : counts) {
        sum += count.get();
      }
      context.write(word, new IntWritable(sum));
    }
}
```

Note :

Ce code révèle plusieurs **défis** inhérents à **MapReduce**...

Les Défis du Traitement de Données avec MapReduce



Apache Pig :

Une Solution de Haut Niveau aux défis de MapReduce

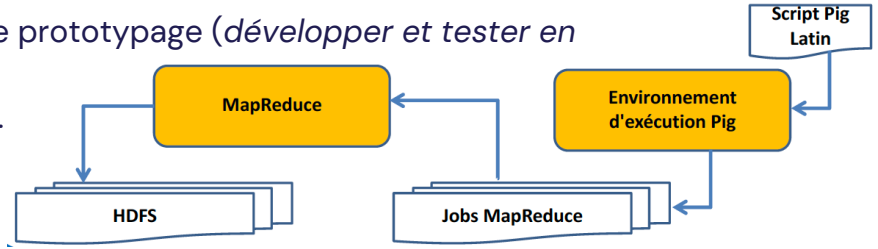
Contexte et Origine :

- **Développé par Yahoo! en 2006** pour surmonter les **limitations** de MapReduce
- **Intégré au projet Apache en 2007** et largement adopté depuis
- **Pig** est une couche d'abstraction au-dessus de **Hadoop** qui simplifie le traitement des données

Objectif fondamental : rendre l'analyse de données massives accessible aux non-experts en Java

Pig est constitué de deux éléments clés :

- **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 **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.



Apache Pig : Architecture

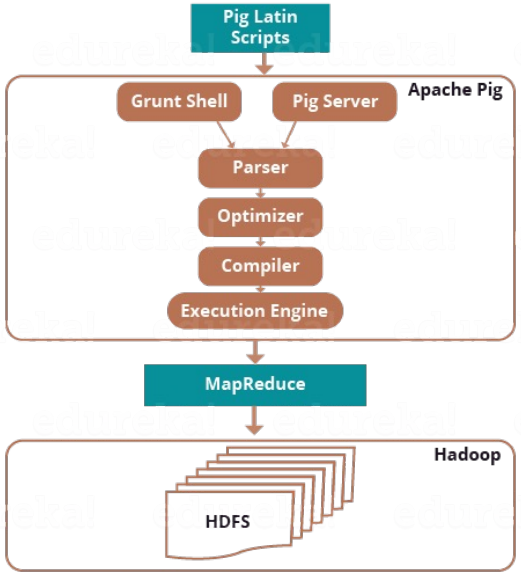
Composants Essentiels

1. Script Pig Latin et Grunt Shell

- Le **script Pig Latin** est un fichier texte (extension .pig) contenant une séquence d'opérations
- Le **Grunt Shell** est une interface interactive pour l'exécution directe de commandes

2. Pig Server et Parser

- Le **Pig Server** coordonne l'exécution des commandes Pig Latin
- Le **Parser** analyse la syntaxe du script et génère un **graphe acyclique dirigé** (**DAG**) représentant les flux de données



L'architecture d'Apache Pig.

3. Optimizer et Compiler

- L'**Optimizer** restructure le plan logique pour améliorer les performances
- Le **Compiler** transforme ce plan optimisé en une séquence de jobs MapReduce

4. Execution Engine

- Soumet les jobs **MapReduce** au cluster Hadoop
- Surveille leur exécution et collecte les résultats

Apache Pig : Architecture

Flux de transformation : Pig-to-MapReduce

1. **Analyse lexicale et syntaxique**
   Le Parser traite le Script Pig Latin
2. **Construction d'un plan logique**
   Création d'un graphe acyclique dirigé (DAG)
3. **Optimisation du plan logique**
   (fusion d'opérations, réduction des transferts de données)
4. **Compilation en plan physique**
   Traduction en jobs MapReduce
5. **Exécution séquentielle**
   Lancement des jobs sur le cluster Hadoop
6. **Collection et présentation des résultats**

Exemple de Script –Pig Latin :

```
-- Script Pig Latin original
logs = LOAD '/data/logs' AS (line:chararray);
errors = FILTER logs BY line MATCHES '.*ERROR.*';
DUMP errors;
```

Ce **Script** traverse l'architecture comme suit :

1. Le **Parser** crée un DAG pour les opérations **LOAD → FILTER → DUMP**
2. L'**Optimizer** peut restructurer ce plan (par exemple, en limitant les données à charger)
3. Le **Compiler** transforme ce plan en job **MapReduce** (dans ce cas, un seul job Map sans phase Reduce)
4. L'**Execution Engine** soumet ce job au cluster et affiche les résultats

Pig : Modèle de données

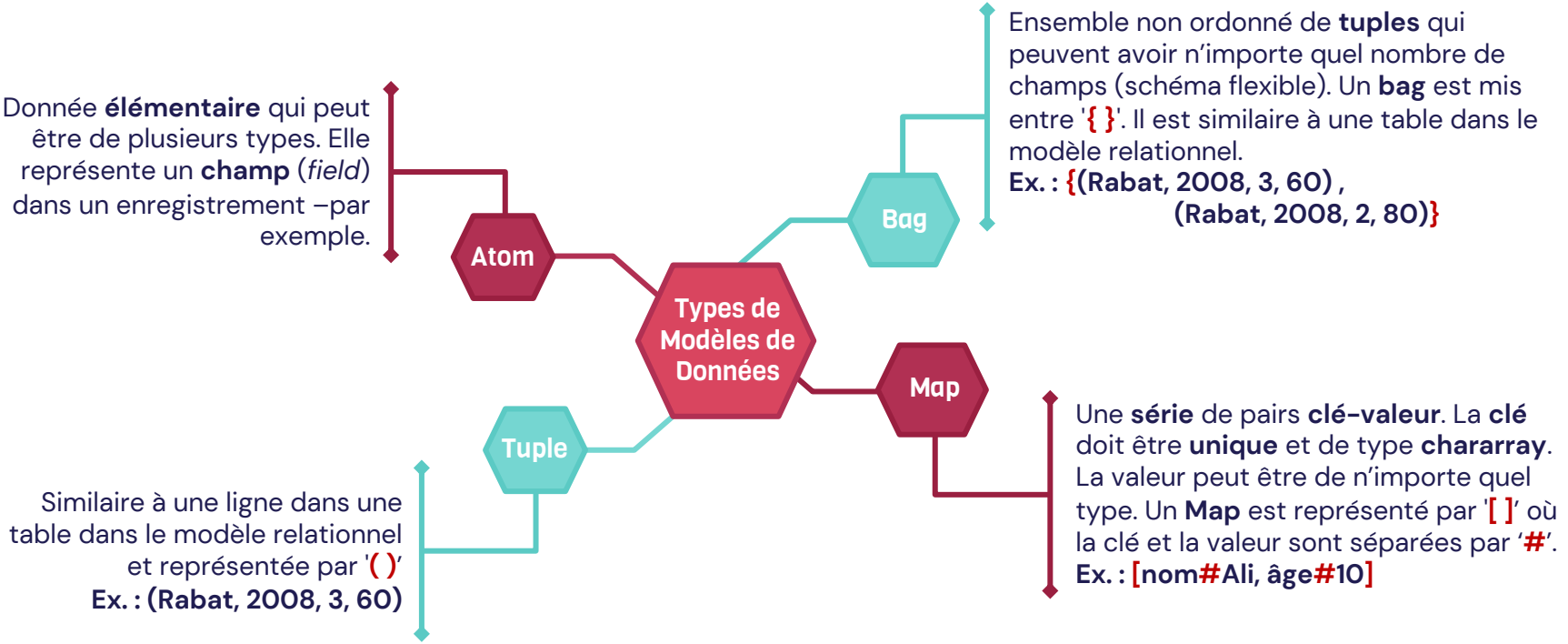
1. Types de Données Primitifs

Pig Latin supporte plusieurs types **primitifs** :

Type de Donnée	Description
int	Entier signé de 32 bits
long	Entier signé de 64 bits
float	Nombre à virgule flottante de 32 bits
double	Nombre à virgule flottante de 64 bits
chararray	Chaîne de caractères UTF-8
bytearray	Tableau d'octets (format par défaut lors du chargement)
boolean	Valeur booléenne (True/False)

Pig : Modèle de données

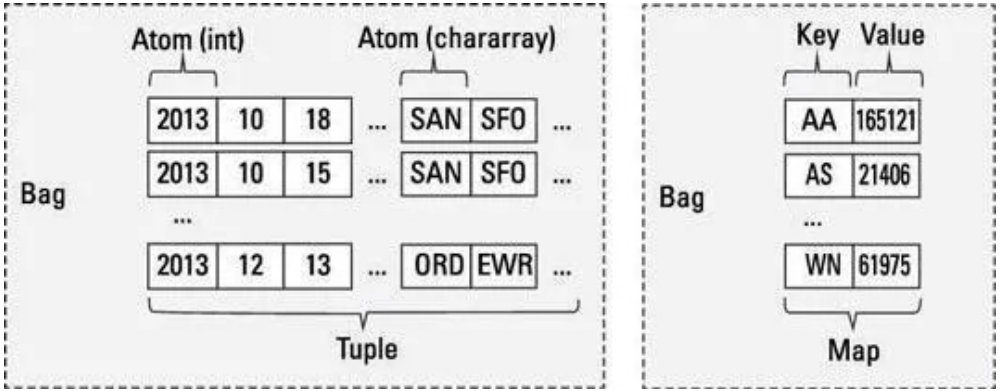
2. Types de Données Complexes#1



Pig : Modèle de données

2. Types de Données Complexes#2

Exemple de types de données : Atom, Bag, Tuple et Map




```
-- Données clients
clients = {
    (1, 'Dupont', 'Jean', 45, 'Paris', {('Ordinateur', 1200), ('Téléphone', 800)}),
    (2, 'Martin', 'Sophie', 32, 'Lyon', {('Tablette', 500), ('Casque', 150), ('Enceinte', 200)})
}
-- Structure du tuple client: (id, nom, prénom, âge, ville, achats)
-- où 'achats' est un bag contenant des tuples (produit, prix)
```

Pig : Environnement d'Exécution

1. Modes d'Exécution

Apache Pig propose deux modes d'exécution distincts :

Mode Local	Mode MapReduce
Le **mode local** exécute Pig dans une seule machine virtuelle Java (JVM) : ○ **Objectif :** Prototypage, développement et tests sur des jeux de données réduits ○ **Activation :** Option `-x local` lors du lancement de Pig ○ **Utilisation recommandée :** Datasets < 1 Go, phase de développement **Ex. :** `pig -x local -f analyse_test.pig`	Le **mode MapReduce** (ou distribué) exécute Pig sur un cluster Hadoop complet : ○ **Objectif :** Traitement de production sur des volumes importants ○ **Activation :** Option `-x mapreduce` (mode par défaut) ○ **Utilisation recommandée :** Datasets > 1 Go, environnement de production **Ex. :** `pig -f analyse_production.pig`

Pig : Environnement d'Exécution

2. Interaction avec Pig#1

Shell interactif (Grunt)

Le **Grunt shell** est l'interface interactive pour Pig :

- **Lancement :** Commande **pig** sans paramètre de script
- **Fonctionnalités :** Exécution directe de commandes Pig Latin
  - Exploration interactive des données
  - Accès aux commandes système via préfixes (**`fs`**, **`sh`**)

Exemple :

```
grunt> R = LOAD 'temp.csv' USING PigStorage(';')
               AS (year:chararray, temp:int, region:int);
grunt> DESCRIBE R;
grunt> DUMP R;
```

Pig : Environnement d'Exécution

2. Interaction avec Pig#2

Scripts Pig Latin (.pig)

Les **scripts Pig Latin** permettent d'automatiser et de réutiliser des traitements :

- **Extension :** `.pig`
- **Exécution :**
  - Via la commande `pig script.pig`
  - Via les commandes `exec` ou `run` dans Grunt

Paramétrage :

```
Pig -param annee=2023 -param fichier='/data/ventes.csv' script.pig
```

Pig : Structure Fondamentale d'un script Pig Latin

Flux de traitement typique

Un **programme Pig Latin** suit généralement un flux en trois étapes principales qui reflètent le cycle **ETL** (**Extract**, **Transform**, **Load**) :

1. LOAD – Chargement des données :

Pour charger des données à partir de **HDFS** en leur donnant un schéma.

```
donnees = LOAD 'chemin/vers/fichier' [USING fonction] [AS schema];
```

Exemple :

```
clients = LOAD '/data/clients.csv' USING PigStorage(',')
          AS (id:int, nom:chararray, age:int, ville:chararray);
```

Pig : Structure Fondamentale d'un script Pig Latin

Flux de traitement typique

2. TRANSFORM/PROCESS – transformation avec opérateurs

- Pour Opérateurs relationnels : **FILTER**, **ORDER**, **DISTINCT**, **JOIN**, **GROUP**, **FOREACH**, **UNION**,…
- Ces transformations sont traduites en plusieurs tâches **Map/Reduce**

Exemple :

```
-- Filtrage
clients_paris = FILTER clients BY ville == 'Paris';

-- Projection et création de champs calculés
profil = FOREACH clients GENERATE
         nom, age, (age >= 18 ? 'Adulte' : 'Mineur') AS statut;

-- Groupement et agrégation
par_ville = GROUP clients BY ville;
stats_ville = FOREACH par_ville GENERATE
              group AS ville,
              COUNT(clients) AS nb_clients,
              AVG(clients.age) AS age_moyen;
```

Pig : Structure Fondamentale d'un script Pig Latin

Flux de traitement typique

3. DUMP/STORE – affichage ou stockage des résultats

Afficher le résultat sur l'écran ou le stocker dans un fichier **HDFS.**

Exemple :

```
-- Affichage à l'écran
DUMP resultat;

-- Stockage dans un fichier
STORE resultat INTO '/resultats/stats_clients' USING PigStorage(';');
```

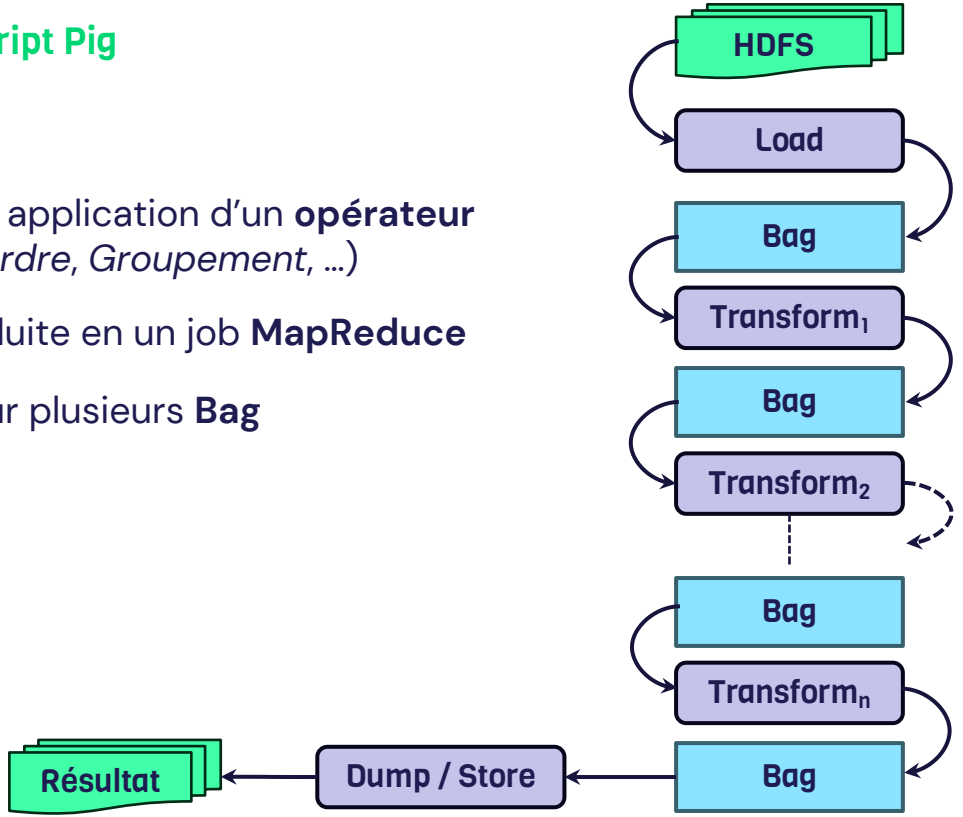
Commentaires en Pig Latin :

/* commentaire sur plusieurs lignes */ ou
-- commentaire en une ligne

Pig : Structure Fondamentale d'un script Pig Latin

Scénario de traitement dans un script Pig

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



Pig : Opérateurs Essentiels de Pig Latin

Opérateurs de chargement (LOAD)

L'opérateur **LOAD** est le point de départ de tout script Pig Latin. Il permet de charger des données depuis diverses sources :

```
relation = LOAD 'source' [USING fonction] [AS schéma];
```

- *Relation* : Un alias (nom) que vous attribuez aux données chargées.
- *Source* : Chemin d'accès vers les données à charger. En **mode local**, il s'agit d'un chemin du système de fichiers; en **mode distribué**, il s'agit généralement d'un chemin HDFS.
  Exemple : '**/user/cloudera/data/clients.csv**'
- *Fonction* : Fonction de chargement optionnelle qui spécifie comment interpréter les données. Si omise, Pig utilise la fonction par défaut **PigStorage** avec le délimiteur **/t**.
  Exemple : **JsonLoader()** (pour un fichier JSON); **TextLoader()** (pour un fichier texte simple)
- *schéma* : Définition optionnelle du schéma qui attribue noms et types aux champs des données. Sans schéma, tous les champs sont chargés comme type **bytearray**
  Exemple : **AS (id:int, nom:chararray, age:int, ville:chararray)**

Pig : Opérateurs Essentiels de Pig Latin

Opérateurs de chargement (LOAD)

Exemple de chargement :

```
-- Chargement simple avec délimiteur par défaut (tab)
donnees = LOAD '/user/data/input.txt';

-- Chargement CSV avec délimiteur spécifique et schéma
clients = LOAD '/data/clients.csv' USING PigStorage(';')
          AS (id:int, nom:chararray, age:int, ville:chararray);

-- Chargement avec tuples imbriqués
etudiants = LOAD '/data/notes.csv' USING PigStorage('|')
            AS (infos:tuple(nom:chararray, prenom:chararray),
            notes:tuple(math:float, phys:float, info:float));
```

Pig : Opérateurs Essentiels de Pig Latin

Opérateurs d'affichage et sauvegarde

DUMP – Affichage des résultats

```
DUMP relation;
```

- *DUMP* : L'instruction qui demande l'affichage des données
- *Relation* : L'alias de la relation à afficher (défini précédemment par un LOAD ou une autre opération)

Exemple d'utilisation:

```
clients = LOAD '/data/clients.csv' USING PigStorage(';') AS (id:int,
nom:chararray, age:int, ville:chararray);
parisiens = FILTER clients BY ville == 'Paris';
DUMP parisiens;
```

Pig : Opérateurs Essentiels de Pig Latin

Opérateurs de chargement (LOAD)

STORE - Sauvegarde des résultats

```
STORE relation INTO 'destination' [USING fonction];
```

- *Destination* : Chemin où seront sauvegardées les données (dossier, pas fichier)
- *Fonction* : Fonction optionnelle qui détermine le format de sortie (par défaut: PigStorage **/t**)

Exemple :

```
resultats = ORDER clients BY age DESC;
STORE resultats INTO '/output/clients_par_age' USING PigStorage(',');
```

Pig : Opérateurs de Traitement de Données

FILTER - Filtrage de Données

L'opérateur **FILTER** sélectionne uniquement les tuples qui satisfont une condition spécifique.

```
résultat = FILTER relation BY condition;
```

Exemple :

```
(1, 'Dupont', 45, 'Paris')
(2, 'Martin', 32, 'Lyon')
(3, 'Durand', 28, 'Paris')
(4, 'Leroy', 52, 'Marseille')
```

Données d'entrée (clients)

```
jeunes = FILTER clients BY age < 35;
```

Code Pig Latin

```
(2, 'Martin', 32, 'Lyon')
(3, 'Durand', 28, 'Paris')
```

Résultat (jeunes)

Pig : Opérateurs de Traitement de Données

FOREACH...GENERATE - Projection et calcul

Cet opérateur parcourt chaque tuple et génère un nouveau tuple selon les expressions spécifiées.

```
résultat = FOREACH relation GENERATE expression1 [AS alias1], ...;
```

Exemple :

Données d'entrée (clients)

```
(1, 'Dupont', 45, 'Paris')
(2, 'Martin', 32, 'Lyon')
(3, 'Durand', 28, 'Paris')
```

Code Pig Latin

```
profils = FOREACH clients GENERATE
          nom,
          age,
          (age >= 35 ? 'Senior' :
           'Junior') AS categorie;
```

Résultat (profils)

```
('Dupont', 45, 'Senior')
('Martin', 32, 'Junior')
('Durand', 28, 'Junior')
```

FOREACH a ajouté une nouvelle catégorie calculée selon l'âge.

Pig : Opérateurs de Traitement de Données

GROUP... BY - Regroupement de données

GROUP regroupe les tuples d'une relation en fonction d'un ou plusieurs champs.

```
résultat = GROUP relation BY champ;
```

Exemple :

Données d'entrée (clients)

```
(1, 'Dupont', 45, 'Paris')
(2, 'Martin', 32, 'Lyon')
(3, 'Durand', 28, 'Paris')
(4, 'Leroy', 52, 'Marseille')
```

Code Pig Latin

```
par_ville = GROUP clients BY ville;
```

Résultat (par_ville)

```
('Paris', {(1, 'Dupont', 45, 'Paris'), (3, 'Durand', 28, 'Paris')})
('Lyon', {(2, 'Martin', 32, 'Lyon')})
('Marseille', {(4, 'Leroy', 52, 'Marseille')})
```

L'opérateur GROUP a créé un nouveau jeu de données où :

- La première colonne (group) contient la valeur de regroupement (ville)
- La seconde colonne contient un bag avec tous les tuples correspondants

Pig : Opérateurs de Traitement de Données

Exercice d'Application #01:

Soit **bank_data.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

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 : Fonctionnalités Avancées

Opérateurs Arithmétiques et Autres...

- **Opérateurs Arithmétiques :** `+, -, *, /, %`

  Exemple :

```
calcul = FOREACH donnees GENERATE prix * quantite AS total;
```

- **Opérateur de Cast :** Conversion de types — *`(type) champ`*

  Exemple :

```
donnees = LOAD 'input.csv' AS (id, valeur:chararray);
convertis = FOREACH donnees GENERATE id, (int)valeur + 10;
```

- **Opérateur Ternaire :** Évaluation conditionnelle — *`(condition ? val_si_true : val_si_false)`*

  Exemple :

```
profil = FOREACH personnes GENERATE nom, age,
         (age >= 18 ? 'Adulte' : 'Mineur') AS statut;
```

Pig : Fonctionnalités Avancées

Opérateur de Sélection Multiple

- **Case :** Sélection conditionnelle multiple (équivalent du `switch/case`)

  Exemple :

```
categorie = FOREACH personnes GENERATE nom,
            CASE
               WHEN age < 18 THEN 'Mineur'
               WHEN age < 65 THEN 'Actif'
               ELSE 'Senior'
            END AS tranche_age;
```

- **CASE avec expression :** Évaluation d'une expression

  Exemple :

```
niveau = FOREACH notes GENERATE etudiant,
         CASE note
           WHEN 20 THEN 'Excellent'
           WHEN 16 THEN 'Très bien'
           WHEN 14 THEN 'Bien'
           WHEN 12 THEN 'Assez bien'
           ELSE 'Passable'
         END AS appreciation;
```

Pig : Fonctionnalités Avancées

L'Opérateur Flatten (fonction FLATTEN)#1

L'opérateur **FLATTEN** est une fonction qui permet de "dénicher" ou "aplatir" des structures de données complexes (tuple, bag, map) en structures plus simples.

1. Aplatissement d'un TUPLE :

L'aplatissement d'un **TUPLE** consiste à transformer ses champs internes en champs individuels au même niveau.

Exemple :

```
adresses = LOAD 'adresses.csv' AS(
          id:int,
          info:tuple(nom:chararray, prenom:chararray),
          adresse:tuple(rue:chararray, ville:chararray, cp:chararray) );
```

Contenu de 'adresses' :

```
(1, (Dupont, Jean), (123 Rue de Paris, Paris, 75001))
(2, (Martin, Sophie), (45 Avenue des Fleurs, Lyon, 69002))
(3, (Durand, Pierre), (8 Boulevard Central, Marseille, 13001))
```

Code Pig Latin :

```
aplati = FOREACH adresses GENERATE id,
                FLATTEN(info),
                FLATTEN(adresse);
```

Résultat (Aplati) :

```
(1, Dupont, Jean, 123 Rue de Paris, Paris, 75001)
(2, Martin, Sophie, 45 Avenue des Fleurs, Lyon, 69002)
(3, Durand, Pierre, 8 Boulevard Central, Marseille, 13001)
```

Pig : Fonctionnalités Avancées

L'Opérateur Flatten (fonction FLATTEN)#2

2. Aplatissement d'un BAG :

L'aplatissement d'un **BAG** génère un **tuple** séparé pour chaque élément du **bag**.

Exemple :

```
clients = LOAD 'clients.csv' AS(
   id:int,
   nom:chararray,
   achats:bag{t:tuple(produit:chararray, prix:float)});
```

Contenu de 'clients' :

```
(1, Dupont, {(Ordinateur, 1200.00), (Téléphone, 800.50)})
(2, Martin, {(Tablette, 450.00)})
(3, Durand, {(Écran, 350.00), (Clavier, 85.00), (Souris, 45.00)})
```

Résultat (achats_detail) **:**

```
(1, Dupont, Ordinateur, 1200.00)
(1, Dupont, Téléphone, 800.50)
(2, Martin, Tablette, 450.00)
(3, Durand, Écran, 350.00)
(3, Durand, Clavier, 85.00)
(3, Durand, Souris, 45.00)
```

Code Pig Latin :

```
achats_detail = FOREACH clients GENERATE id, nom,
FLATTEN(achats) AS (produit, prix);
```

Pig : Fonctionnalités Avancées

L'Opérateur Flatten (fonction FLATTEN)#3

3. Aplatissement d'un MAP :

L'aplatissement d'un **MAP** génère un **tuple** pour chaque paire **clé-valeur**.

Exemple :

```
utilisateurs = LOAD 'utilisateurs.csv' AS(
        id:int,
        profil:map[chararray]);
```

Contenu de 'utilisateurs' :

```
(1, [nom#Dupont, age#45, ville#Paris])
(2, [nom#Martin, role#admin, service#IT])
(3, [nom#Durand, diplome#Master, specialite#IA])
```

Code Pig Latin :

```
profil_details = FOREACH utilisateurs
GENERATE id, FLATTEN(profil) AS
(attribut, valeur);
```

Résultat (profil_details) :

```
(1, nom, Dupont)
(1, age, 45)
(1, ville, Paris)
(2, nom, Martin)
(2, role, admin)
(2, service, IT)
(3, nom, Durand)
(3, diplome, Master)
(3, specialite, IA)
```

Pig : Fonctionnalités Avancées

L'Opérateur Flatten (fonction FLATTEN)#4

Cas d'utilisation pratiques de 'FLATTEN' :

1. **Normalisation de données :** Transformer des données hiérarchiques en format tabulaire.
2. **Analyse détaillée :** Permettre des analyses au niveau le plus granulaire.
3. **Jointures avancées :** Faciliter les jointures sur des éléments imbriqués.
4. **Traitement de données JSON :** Manipuler des structures JSON complexes en les transformant en format relationnel

Pig : Fonctionnalités Avancées

Fonctions sur Chaînes de Caractères

- **CONCAT —** Concaténation de chaînes : Concatène deux ou plusieurs chaînes.

Exemple :

```
noms = LOAD 'noms.txt' AS (prenom:chararray, nom:chararray);
noms_complets = FOREACH noms GENERATE CONCAT(prenom, ' ', nom);
```

  - **Entrée :** `('Jean', 'Dupont')`
  - **Sortie :** `('Jean Dupont')`

- **INDEXOF —** Position d'un caractère : Retourne la position de la première occurrence d'un caractère/chaîne dans une chaîne.

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
position = FOREACH texte GENERATE INDEXOF(ligne, 'a', 0);
```

  - **Entrée :** `('Hadoop')`
  - **Sortie :** `(1)` (position du 'a' dans 'Hadoop')

Pig : Fonctionnalités Avancées

Fonctions sur Chaînes de Caractères

- **LAST_INDEX_OF —** Dernière position : Retourne la position de la dernière occurrence d'un caractère dans une chaîne.

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
position = FOREACH texte GENERATE LAST_INDEX_OF(ligne, 'a');
```

  - **Entrée :** `('Java')`
  - **Sortie :** `('3')` (position du dernier 'a' dans 'Java')

- **UPPER —** Conversion en majuscules : Convertit une chaîne en majuscules.

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
maj = FOREACH texte GENERATE UPPER(ligne);
```

  - **Entrée :** `('pig latin')`
  - **Sortie :** `('PIG LATIN')`

Pig : Fonctionnalités Avancées

Fonctions sur Chaînes de Caractères

- **LOWER —** Conversion en minuscules : Convertit une chaîne en minuscules.

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
min = FOREACH texte GENERATE LOWER(ligne);
```

  - **Entrée :** ('PIG LATIN')
  - **Sortie :** ('pig latin')

- **REPLACE —** Remplacement de caractères : Remplace toutes les occurrences d'une sous-chaîne par une autre.

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
remplace = FOREACH texte GENERATE REPLACE(ligne, '-', ' ');
```

  - **Entrée :** ('Big-Data')
  - **Sortie :** ('Big Data')

Pig : Fonctionnalités Avancées

Fonctions sur Chaînes de Caractères

- **SIZE —** Longueur d'une chaîne : Retourne le nombre de caractères dans une chaîne.

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
longueur = FOREACH texte GENERATE SIZE(ligne);
```

  - **Entrée :** `('Hadoop')`
  - **Sortie :** `(6)`

- **STRSPLIT —** Division d'une chaîne : Divise une chaîne selon une expression régulière et retourne un tuple contenant les parties résultantes.

```
(chararray) STRSPLIT(chararray source, chararray regex)
```

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
mots = FOREACH texte GENERATE STRSPLIT(ligne, '[-\\s#]') AS parties;
```

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

Pig : Fonctionnalités Avancées

Fonctions sur Chaînes de Caractères

- **SUBSTRING —** Longueur d'une chaîne : Extrait une partie d'une chaîne entre les positions **start** et **end**.

Exemple :

```
texte = LOAD 'texte.txt' AS (ligne:chararray);
sousChaine = FOREACH texte GENERATE SUBSTRING(ligne, 0, 3);
```

  - **Entrée :** `('Hadoop')`
  - **Sortie :** `('Had')`

- **TOKENIZE —** Découpage en mots : Divise une chaîne de caractères en mots individuels, chacun placé dans son propre **tuple**, et tous ces tuples sont ensuite placés dans un **bag**.

Syntaxe :

```
TOKENIZE(string [, 'delimiter'])
```

Exemple :
Données d'entrée (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
```

Pig : Fonctionnalités Avancées

Fonctions sur Chaînes de Caractères

Chargement des données

```
R = LOAD 'maman.txt' USING TextLoader() AS (L:chararray);
```

Contenu de R après chargement :

```
(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)
```

Application de TOKENIZE :

```
F = FOREACH R GENERATE TOKENIZE(L) AS m;
```

Structure du résultat :

```
F: {m: {tuple_of_tokens: (token: chararray)}}
```

Résultat (contenu de F) :

```
({(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)})
```

Pig : Fonctionnalités Avancées

Les 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)
- …

Exemple :

```
-- Chargement de données numériques
measures = LOAD 'sensor_data.csv' AS (
            sensor_id:int, temp:double, humidity:double,
            pressure:double);

-- Application des fonctions mathématiques
stats = FOREACH (GROUP measures BY sensor_id) GENERATE
            group AS sensor_id,
            COUNT(measures) AS count,
            AVG(measures.temp) AS avg_temp,
            MIN(measures.temp) AS min_temp,
            MAX(measures.temp) AS max_temp,
            SUM(measures.humidity) AS total_humidity,
            SQRT(SUM(measures.pressure)/COUNT(measures)) AS
            rms_pressure;
```

Pig : Fonctionnalités Avancées

Les Fonctions sur des Types Complexes

- 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**.
- chararray **TOSTRING**(*complex input*)
- …

Exemple :

```
-- Données initiales
R = LOAD 'dates.txt' AS (a:int, t:int, c:int);

-- Exploration dans Grunt
Grunt> describe R;
R: {a: int, t:int, c:int}

-- Conversion en bag
Grunt> B = foreach R Generate TOBAG(a, t, c);
Grunt> dump B;
( {(2011),(23),(4)} )
( {(2012),(15),(3)} )
( {(2009),(35),(9)} )
```

```
-- Contenu de R
(2011, 23, 4)
(2012, 15, 3)
(2009, 35, 9)
...
```

Pig : Exécution et Paramétrage d'un Script.pig

Pour lancer un **script pig** :

```
grunt > exec [[-param param_name = param_value] [...] ] file
grunt > run[[-param param_name = param_value] [...] ] file
```

Exemple 01 :

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 > run -param annee=2011 temp_moy_year.pig
```

Résultat :

```
(2011, 31.301972567)
```

Pig : Exécution et Paramétrage d'un Script.pig

Exemple 02 :

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}
```

References

- **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