

## Séance 8

# Opérations sur des données en NoSQL



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

# Rappels

- Analyse des différents types de **consistance de données**
  - Consistance à la lecture et à l'écriture dans les nœuds
  - Consistance de réPLICATION et mise à jour entre nœuds
  - Consistance avec soi-même et utilisation de sessions
- **Techniques** pour assurer un certain niveau de consistance
  - Quorum de lecture et d'écriture et niveau de réPLICATION
  - Version stamp pour choisir donnée la plus à jour

# Objectifs

- Recherche de données grâce à **Elasticsearch**
  - Déploiement et population d'index
  - Requêtes de recherche de données
- Opérations et analyse de données avec **Map-Reduce**
  - Définition et description des opérations Map et Reduce
  - Exemple de Map-Reduce sur une base de données
  - Plateforme Pig pour programmes sur Hadoop



# Elasticsearch

# Elasticsearch (1)

- Moteur de recherche plein texte avec indexation automatique

*Développé comme une surcouche de Lucene*

- Plusieurs **caractéristiques** du moteur

- Répartition du travail sur plusieurs machines
- Interface REST pour accéder à ses fonctionnalités



# Elasticsearch (2)

- Construction automatique d'**index** comme les bases NoSQL

*Utilisation du sharding et réPLICATION des données*

- **Requête de création** de nouveaux index (~ base de données)

*Ajout de documents dans l'index*

- **Interrogation d'un index** pour récupérer un document

*Requête GET vers Elasticsearch comme avec CouchDB*

# Elasticsearch vs SGBD

- Structure similaire entre **Elasticsearch et les SGBDs**
  - Moteur scalable gérant des *Po* de données structurées ou non
  - Augmentation des performances en utilisant la dénormalisation
  - Moteur de recherche en temps réel et distribué

| Elasticsearch | SGBD            |
|---------------|-----------------|
| Index         | Base de données |
| Shard         | Shard           |
| Mapping       | Table           |
| Champ         | Champ           |
| Objet JSON    | Tuple           |

# Démarrage d'Elasticsearch

- Elasticsearch tourne comme **un service** sur le port 9200

*Appeler son interface REST vous donne un message d'accueil*

```
$ service elasticsearch start

$ curl 127.0.0.1:9200
{
  "name" : "X3GtW1g",
  "cluster_name" : "elasticsearch_combefis",
  "cluster_uuid" : "a4ySj4h4RpGLyiCw5F3kGg",
  "version" : {
    "number" : "6.0.1",
    "build_hash" : "601be4a",
    "build_date" : "2017-12-04T09:29:09.525Z",
    "build_snapshot" : false,
    "lucene_version" : "7.0.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

# Ajout de données

- Création d'un **nouvel index** avec une requête PUT

*On peut ensuite y stocker un document*

```
$ curl -X PUT 127.0.0.1:9200/school
{"acknowledged":true,"shards_acknowledged":true,"index":"school"}  
  
$ curl -X PUT 127.0.0.1:9200/school/students/1 -H "Content-Type: application/json" -d '{"firstname": "Julien", "lastname": "Kessels"}'
{"_index":"school","_type":"students","_id":"1","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1}
```

# Lecture de données

- **Lecture d'un document** en passant par une requête GET
  - Possibilité de ne récupérer que certains champs avec un filtre
  - Possibilité de ne récupérer que la source avec `_source`

```
$ curl 127.0.0.1:9200/school/students/1
{"_index": "school", "_type": "students", "_id": "1", "_version": 3, "found": true, "_source": {"firstname": "Julien", "lastname": "Kessels"}}

$ curl "127.0.0.1:9200/school/students/1?_source_include=lastname"
{"_index": "school", "_type": "students", "_id": "1", "_version": 7, "found": true, "_source": {"lastname": "Kessels"}}

$ curl "127.0.0.1:9200/school/students/1/_source?_source_include=lastname"
{"lastname": "Kessels"}
```

# Recherche de données (1)

- Recherche de données avec la route spéciale `_search`

*On peut faire une recherche en filtrant sur les clés du document*

- Recherche dans tous les shards et renvoi du résultat groupé

```
$ curl "127.0.0.1:9200/school/_search?q=firstname:Julien"
{"took":2,"timed_out":false,"_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":2,"max_score":0.2876821,"hits":[{"_index":"school","_type":"students","_id":"2","_score":0.2876821,"_source":{"firstname": "Julien", "lastname": "Lapraillle"}},{"_index":"school","_type":"students","_id":"1","_score":0.2876821,"_source":{"firstname": "Julien", "lastname": "Kessels"}}]}}

$ curl "127.0.0.1:9200/school/_search?q=firstname:Benoit"
{"took":2,"timed_out":false,"_shards":{"total":5,"successful":5,"skipped":0,"failed":0},"hits":{"total":0,"max_score":null,"hits":[]}}
```

# Recherche de données (2)

- Possibilité de rechercher sur **plusieurs index** à la fois

*Séparer les index à explorer avec des virgules*

- Possibilité de **recherche générale** pas associée à une clé

```
$ curl "127.0.0.1:9200/school,library/_search?q=Julien"
{"took":7,"timed_out":false,"_shards":{"total":10,"successful":10,"skipped":0,"failed":0},"hits":{"total":3,"max_score":0.2876821,"hits":[{"_index":"school","_type":"students","_id":"2","_score":0.2876821,"_source":{"firstname": "Julien", "lastname": "Lapraille"}},{"_index":"library","_type":"books","_id":"1","_score":0.2876821,"_source":{"title": "Comment arnaquer les Julien ?", "author": "X"}},{"_index":"school","_type":"students","_id":"1","_score":0.2876821,"_source":{"firstname": "Julien", "lastname": "Kessels"}}]}}
```

# Fonctionnalité d'Elasticsearch

- Définition de **rivières** pour connexion à source de données

*Permet un remplissage automatique de l'index*

- Définition de **facette** pour calculer de l'information

*Retourne des informations agrégées supplémentaire*

# Module Python elasticsearch

## ■ Module Python `elasticsearch` pour interroger Elasticsearch

*Passerelle `thrift` à démarrer avec `hbase thrift start`*

```
1 from elasticsearch import Elasticsearch
2 es = Elasticsearch()
3
4 result = es.search(index="school", q="Julien")
5 for hit in result['hits']['hits']:
6     print(hit['_source'])
```

```
{'lastname': 'Lapraille', 'firstname': 'Julien'}
{'lastname': 'Kessels', 'firstname': 'Julien'}
```

# Map-Reduce

REDUCED  
REDUCED

REDUCED  
REDUCED  
REDUCED

REDUCED  
REDUCED

REDUCED  
REDUCED

£10

REDUCED  
REDUCED

REDUCED  
REDUCED

£10  
£5

£10  
£5  
£3  
£2  
£1

# Cluster de machines

- Utilisation de **clusters** change manière de stocker les données  
*Mais change également la manière de faire des opérations avec*
- Modification de la manière de **penser les calculs**
  - Sur le serveur de DB ou une machine cliente, si DB centralisée
  - Réparti sur toutes les machines du cluster, sinon
- Attention au **cout de transfert** des données entre nœuds

*Calculs sur la machine où sont les données nécessaires*

# Map-Reduce

- **Organisation du processing** des données pour exploiter cluster

*Plusieurs machines sur un cluster en restant proche des données*

- Première implémentation **framework MapReduce** de Google

*Version open source dans Hadoop et implémentations dans DB*

- Origine du pattern de la **programmation fonctionnelle**

*Opérations map et reduce applicables à des collections*

# Exemple en Python

- Fonction `map` applique une fonction à une liste d'éléments

*Application parallèle possible de la fonction à chaque élément*

```
1 data = [2, -7, 4, 0, -3]
2 squared = list(map(lambda x: x ** 2, data))
3 print(squared)
```

- Fonction `reduce` renvoie un résultat à partir d'une liste

*Calcul incrémental possible du résultat final*

```
1 result = reduce(lambda x, y: x + y, squared)
2 print(result)
```

[4, 49, 16, 0, 9]

78

# Exemple (1)

- Stockage d'une **commande d'un client** dans un agrégat
  - Notamment articles commandés (description, quantité et prix)
  - Les données sont shardées sur plusieurs nœuds d'un cluster
- Les analystes aimeraient connaitre **revenu total d'un produit**

*Par exemple, sur les sept derniers jours écoulés*
- **Nécessité de visiter** toutes les machines du cluster

*Examiner toutes les commandes, sommer les prix du produit*

## Exemple (2)

- Exemple d'un agrégat pour une **commande de deux articles**

*Chaque article possède un nom, une quantité et un prix unitaire*

```
1  {
2      "id": 101,
3      "customer": "Damien",
4      "items": [
5          {"name": "pralines",
6              "quantity": "2",
7              "price": "29.98"
8          },
9          {"name": "socks",
10             "quantity": "1",
11             "price": "5.99"
12     }],
13     "billing address": "...",
14     "shipping address": "...",
15     "payment details": "..."
16 }
```

# Map (1)

- Fonction Map appliquée à un seul agrégat

*Produit un ensemble de paires clé-valeur comme résultat*

- Toutes les applications de Map sont indépendantes

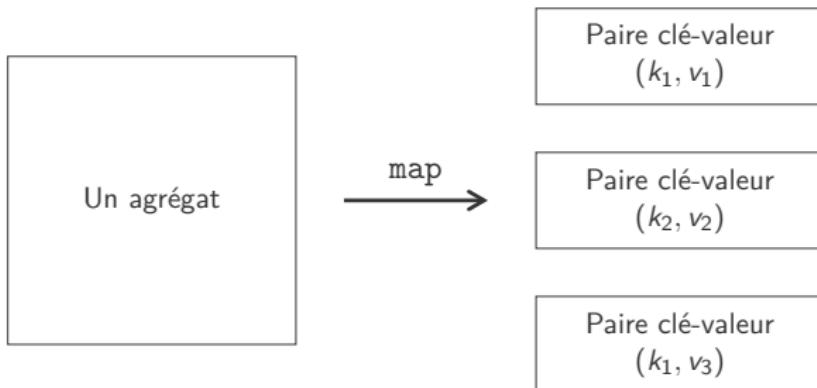
*Parallélisme massif exploitant la localité spatiale*

```
1 {  
2   "pralines": {  
3     "quantity": "2",  
4     "price": "29.98"  
5   },  
6   "socks": {  
7     "quantity": "1",  
8     "price": "5.99"  
9   }  
10 }
```

# Map (2)

- Grand niveau de parallélisme et exploitation **localité données**

*Opérations peuvent être simples voire très complexes*



# Reduce (1)

- Fonction **Reduce** appliquée aux résultats de Map

*Produit un seul résultat unique, et non plus une collection*

- Consomme toutes les valeurs produites avec la **même clé**

*Réduction de plusieurs résultats en un seul*

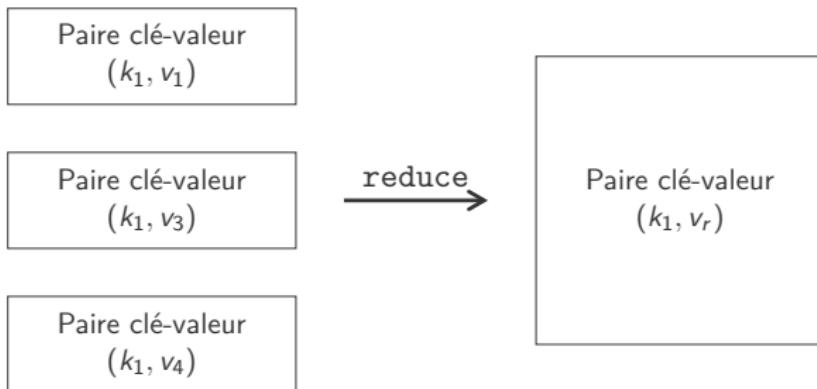
```
1 {  
2   "pralines": [{  
3     "quantity": "2",  
4     "price": "29.98"  
5   }, {  
6     "quantity": "1",  
7     "price": "11.99"  
8   }, {  
9     "quantity": "3",  
10    "price": "32.97"  
11  }]  
12 }
```

```
1 {  
2   "pralines": {  
3     "quantity": "6",  
4     "price": "74.94"  
5   }  
6 }
```

# Reduce (2)

- Monitore toutes les valeurs émises avec **une clé donnée**

*Déplacement des résultats des Map vers les Reduce*



# Partitionnement (1)

- **De base**, une opération Map par nœud et un seul Reduce
  - La fonction Reduce provoque un goulot d'étranglement
  - Diminution parallélisme et augmentation transfert de données
- **Partition des clés** des résultats produits par Map

*Plusieurs instances de Reduce agissant sur un ensemble de clés*

- Exécution de plusieurs **Reduce** en parallèle

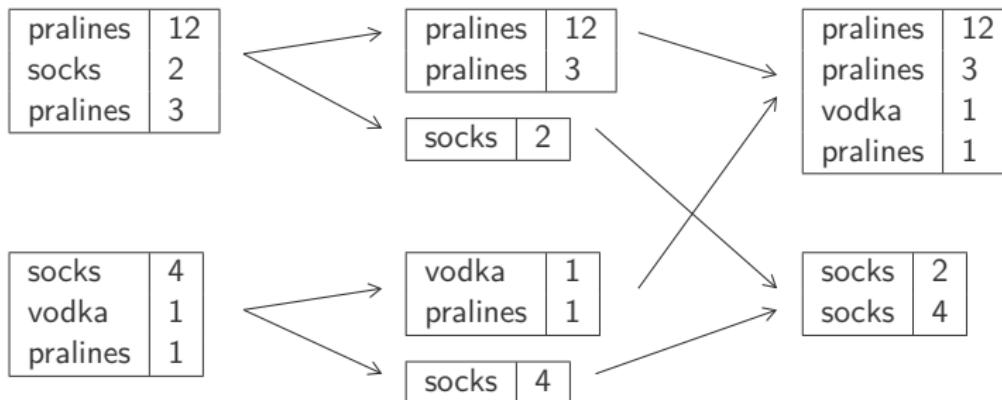
*Fusion des résultats des différents Reduce pour obtenir le final*

# Partitionnement (2)

- Deux reducers pour ensembles différents de clés

*Le premier sur {pralines, vodka}, et le second pour {socks}*

## Sortie du Map      Partitionnement      Reduce



# Combinaison (1)

- Diminution des **données transmises** entre nœuds  
*Données très répétitives à compacter avant envoi*
- Agit comme une **fonction de réduction**, à l'intérieur d'un nœud  
*« Combinable reducer » doit matcher entre inputs et outputs*
- Toutes les fonctions de réduction ne sont **pas combinables**  
*Calcul du nombre de clients ayant acheté chaque produit*

# Combinaison (2)

- **Un combiner** rassemble les paires ayant les mêmes clés

*Sur le même noeud avant envoi vers le reducer distant*

**Sortie du Map**

|          |    |
|----------|----|
| pralines | 12 |
| socks    | 2  |
| pralines | 3  |
| socks    | 4  |
| vodka    | 1  |
| pralines | 1  |

**Combine**

|          |    |
|----------|----|
| pralines | 16 |
| socks    | 6  |
| vodka    | 1  |

# Combinable reducer

- Exemple d'une fonction de réduction qui n'est **pas combinable**
  - Calcul du nombre de personnes ayant acheté chaque article
  - Le résultat produit est différent de l'input reçu

Sortie du Map

|          |    |         |
|----------|----|---------|
| pralines | 12 | Damien  |
| socks    | 2  | Alexis  |
| pralines | 3  | Sylvain |
| socks    | 4  | Alexis  |
| vodka    | 1  | Damien  |
| pralines | 1  | Alexis  |

Reduce

|          |   |
|----------|---|
| pralines | 3 |
| socks    | 1 |
| vodka    | 1 |

# Construire un calcul (1)

- Plusieurs **contraintes** à respecter pour Map-Reduce
  - Un Map ne peut agir que sur un unique agrégat
  - Un Reduce ne peut agir que sur une seule clé
- Il faut **restructurer les opérations** à faire pour fitter le modèle

*Doit fitter la notion d'opération de réduction*
- Exemple qui calcule la **moyenne** d'une donnée

*La moyenne n'est pas une réduction composable*

# Construire un calcul (2)

- Calcul de la **quantité moyenne** par commande d'un article
  - Ne peut pas être calculée comme une somme, doit l'être après
  - Il faut mémoriser montant total et quantité

## Sortie du Combine (pralines)      Réduction

|                     |     |
|---------------------|-----|
| quantité totale     | 600 |
| nombre de commandes | 10  |
| quantité moyenne    | 60  |

|                         |           |
|-------------------------|-----------|
| quantité totale         | 1200      |
| nombre de commandes     | 25        |
| <b>quantité moyenne</b> | <b>48</b> |

|                     |     |
|---------------------|-----|
| quantité totale     | 600 |
| nombre de commandes | 15  |
| quantité moyenne    | 40  |

# Faire un compte

- Prévoir un champ fixé à une valeur 1 pour faire un compte

*Il suffit que le Reduce fasse la somme du champ*

## Sortie du Map (pralines)

|                     |    |
|---------------------|----|
| quantité totale     | 26 |
| nombre de commandes | 1  |

|                     |    |
|---------------------|----|
| quantité totale     | 36 |
| nombre de commandes | 1  |

|                     |    |
|---------------------|----|
| quantité totale     | 44 |
| nombre de commandes | 1  |

## Réduction

|                     |     |
|---------------------|-----|
| quantité totale     | 106 |
| nombre de commandes | 3   |
| quantité moyenne    | 35  |

# Enchainement de Map-Reduce (1)

- Possibilité de **chainer les opérations** de Map-Reduce
  - Pour découper un calcul complexe en plus petites unités
  - Par exemple, comparer ventes d'un produit entre deux années

## Map-Reduce

pralines:2015:12

|          |          |
|----------|----------|
| produit  | pralines |
| année    | 2015     |
| mois     | 12       |
| quantité | 1200     |

pralines:2014:12

|          |          |
|----------|----------|
| produit  | pralines |
| année    | 2014     |
| mois     | 12       |
| quantité | 1000     |

## Map-Reduce

pralines:12

|                     |          |
|---------------------|----------|
| produit             | pralines |
| année               | 2015     |
| mois                | 12       |
| quantité            | 1200     |
| quantité précédente | 1000     |
| augmentation        | 20%      |

# Étape 1 : Bilan par année

- Construction du **bilan par année** pour chaque produit

*Le Reduce sort le bilan de chaque mois de chaque année*

## Sortie du Map

pralines:2015:12

|          |          |
|----------|----------|
| produit  | pralines |
| année    | 2015     |
| mois     | 12       |
| quantité | 800      |

pralines:2015:12

|          |          |
|----------|----------|
| produit  | pralines |
| année    | 2015     |
| mois     | 12       |
| quantité | 400      |

## Réduction

pralines:2015:12

|          |          |
|----------|----------|
| produit  | pralines |
| année    | 2015     |
| mois     | 12       |
| quantité | 1200     |

# Étape 2 : Comparaison des années (1)

- Construction des données pour l'année et sa précédente

*Le Map produit deux paires en sortie avec les deux quantités*

## Sortie du Map

pralines:2015:12

|          |          |
|----------|----------|
| produit  | pralines |
| année    | 2015     |
| mois     | 12       |
| quantité | 1200     |

pralines:2014:12

|          |          |
|----------|----------|
| produit  | pralines |
| année    | 2014     |
| mois     | 12       |
| quantité | 1000     |

## Map

pralines:12

|                     |          |
|---------------------|----------|
| produit             | pralines |
| année               | 2015     |
| mois                | 12       |
| quantité            | 1200     |
| quantité précédente | 0        |

pralines:12

|                     |          |
|---------------------|----------|
| produit             | pralines |
| année               | 2014     |
| mois                | 12       |
| quantité            | 0        |
| quantité précédente | 1000     |

# Étape 2 : Comparaison des années (2)

## ■ Comparaison des deux années successives

*Fusion des deux paires clé-valeur incomplète en une seule*

### Sortie du Map

pralines:12

|                    |          |
|--------------------|----------|
| produit            | pralines |
| année              | 2015     |
| mois               | 12       |
| quantité           | 1200     |
| quantité précédent | 0        |

pralines:12

|                     |          |
|---------------------|----------|
| produit             | pralines |
| année               | 2015     |
| mois                | 12       |
| quantité            | 0        |
| quantité précédente | 1000     |

### Réduction

pralines:12

|                     |          |
|---------------------|----------|
| produit             | pralines |
| année               | 2015     |
| mois                | 12       |
| quantité            | 1200     |
| quantité précédente | 1000     |
| <b>augmentation</b> |          |
| <b>20%</b>          |          |

# Enchainement de Map-Reduce (2)

- Décomposition d'un gros calcul en **plusieurs étapes simples**

*Chacune des étapes est un Map-Reduce plus simple*

- **Résultats intermédiaires** peuvent nourrir plusieurs opérations

- Simplifie la programmation et les calculs réalisés
  - Peut être stocké en vue matérialisée

- Pas de contraintes de **langages** pour Map-Reduce

*Certains langages adaptés existent comme Apache Pig, Hive*

# Map-Reduce incrémental

- Recalculer le Map-Reduce à chaque mise à jour coûte cher  
*Éviter de recommencer les calculs à partir de zéro à chaque fois*
- Structurer les calculs pour permettre **mise à jour incrémentale**
  - Facile pour le Map puisque travaillent de manière isolées
  - Reduce/Combine de la partition doit être réexécuté
- **Réduction combinable** recalculée incrémentalement

*Selon le type de changement (additif ou destructif)*



Pig

# Plateforme Pig

- Plateforme de haut niveau Pig pour programmes sur Hadoop
  - Utilisation du langage Pig Latin pour décrire les programmes
  - Décrire des opérations suivant le paradigme Map-Reduce
- Trois propriétés clés offertes par Pig Latin
  - Facilité de programmation parallèle pour analyser des données
  - Opportunités d'optimisation automatique de l'exécution
  - Extensibilité avec création de ses propres fonctions

# Tutoriel Pig (1)

- Petit **exemple simple** d'exécution d'un script Pig

*Source : <https://github.com/rohitsden/pig-tutorial>*

- Lancement de Pig en **mode local** et chargement de données
  - Exécution de commande en Pig Latin via le shell grunt
  - Chargement depuis un fichier CSV avec `PigStorage(,,)`

```
& pig -x local
[...]
grunt> movies = LOAD 'movies_data.csv' USING PigStorage(,,) as (
id ,name ,year ,rating ,duration );
grunt> DUMP movies;
```

# Tutoriel Pig (2)

- Films avec évaluation > 4 avec **filtrage de données**

*Utilisation de l'opération FILTER suivie d'une condition*

```
grunt> movies_gt_four = FILTER movies BY (float)rating>4.0;  
grunt> DUMP movies_gt_four;
```

- Application d'une **transformation** basée sur les colonnes

*Utilisation de l'opération FOREACH pour parcourir*

```
grunt> movie_duration = FOREACH movies GENERATE name, (double)(duration/60);  
grunt> DUMP movie_duration;
```

# Analyse de données en Pig (1)

- Calcul de la **quantité moyenne** par commande d'un article

*Reprise de l'exemple de la section précédente, mais en Pig Latin*

```
>grunt products = LOAD 'products_data.csv' USING PigStorage( , )  
as (id:int, name:chararray, year:int, month:int, quantity:int);  
>grunt DUMP products;  
[...]  
(1,pralines,2015,12,800)  
(1,pralines,2015,12,400)  
(2,pralines,2014,12,1000)
```

# Analyse de données en Pig (2)

- Construction du **bilan par année** pour chaque produit

*Deux étapes avec regroupement suivi de somme*

```
>grunt grouped_by_year = GROUP products BY (name,year,month);
>grunt DUMP grouped_by_year;
[...]
((pralines,2014,12),{(2,pralines,2014,12,1000)})
((pralines,2015,12),{(1,pralines,2015,12,800),(1,pralines
,2015,12,400)})

>grunt count_by_year = FOREACH grouped_by_year GENERATE group,
SUM(products.quantity) AS quantity;
>grunt DUMP count_by_year;
[...]
((pralines,2014,12),1000)
((pralines,2015,12),1200)
```

# Analyse de données en Pig (3)

- Construction des données pour l'année et sa précédente

*Utilisation du CASE pour gérer année actuelle et précédente*

```
>grunt diff_by_year = FOREACH count_by_year GENERATE (group.name,  
group.month) AS key, (CASE group.year WHEN 2015 THEN quantity  
WHEN 2014 THEN 0 END) AS quantity, (CASE group.year WHEN 2015  
THEN 0 WHEN 2014 THEN quantity END) AS prev_quantity;  
>grunt DUMP diff_by_year;  
[...]  
((pralines,12),0,1000)  
((pralines,12),1200,0)
```

# Analyse de données en Pig (4)

## ■ Comparaison des deux années successives

*Regroupement des données et calcul augmentation entre les deux*

```
>grunt grouped_by_key = GROUP diff_by_year BY key;
>grunt DUMP grouped_by_key;
[...]
((pralines ,12),{((pralines ,12),0,1000),((pralines ,12),1200,0)})

>grunt stats_by_year = FOREACH grouped_by_key GENERATE group, SUM
(diff_by_year.quantity) AS quantity, SUM(diff_by_year.
prev_quantity) AS prev_quantity, (stats_by_year.quantity-
stats_by_year.prev_quantity)/10 AS increase;
>grunt DUMP stats_by_year;
[...]
((pralines ,12),1200,1000,20)
```

# Crédits

- Photos des logos depuis Wikipédia
- <https://www.flickr.com/photos/richard-g/303117497>
- <https://www.flickr.com/photos/samsmith/491756802>
- <https://www.flickr.com/photos/hoobandmoonar/9721033200>