BIG DATA ET SCIENCE DE DONNÉES

PRINCIPES DU NoSQL

Master 1 Intelligence Artificielle

Université de M'sila, Département d'Informatique

Dr Mehenni Tahar

2020-2021

Motivations du NoSQL

1969-1970 : arrivée du modèle relationnel

- **Rappel des principes** :
- Repose sur des relations entre les valeurs des données (indépendamment de leur emplacement en mémoire)
- Manipulation à travers une algèbre et un langage de haut niveau
- Dissocie représentation-et-interrogation du stockage, et sera quand même efficace!
- **Mais des « contraintes » :**
- Toutes les lignes d'une Relation ont les mêmes colonnes (valeur *NULL* si absence de données)
- Modifications par séquences atomiques pour que la BdD soit toujours totalement cohérente
- Conception d'un schéma de base qui s'impose à toute la BdD
- Interrogation par *jointures* de nombreuses (petites) Relations

Motivations du NoSQL

1969-1970 : arrivée du modèle relationnel

- On ne peut pas mettre n'importe quoi dans une BdD Relationnelle !
- Toutes les données doivent respecter le schéma initial…
- Le langage Relationnel SQL est adapté pour extraire des informations selon des conditions sur leurs valeurs
- Requêtes OLTP (OnLine Transaction Processing) : **OK**
  …mais n'est pas adapté pour faire des calculs de statistiques complexes sur ces valeurs, ni sur des données volumineuses!
- Requêtes OLAP (OnLine Analytical Processing) : **problème…**

Motivations du NoSQL

Evolution des technologies de BdD traditionnelles

Travail de CODD sur la représentation des données - 1969 → Modèle Relationnel 1970 → Avènement des SGBD relationnels - 1980 → Le stockage devient bon marché, les historiques de data abondent → Les besoins en Data Analytics deviennent fréquents → BdD OLAP → BdD OLAP en mémoire et distribuées → Approche Big data

- Big Data Analytics
- Big Data Engineering

Deux types de requêtes / d'utilisation :

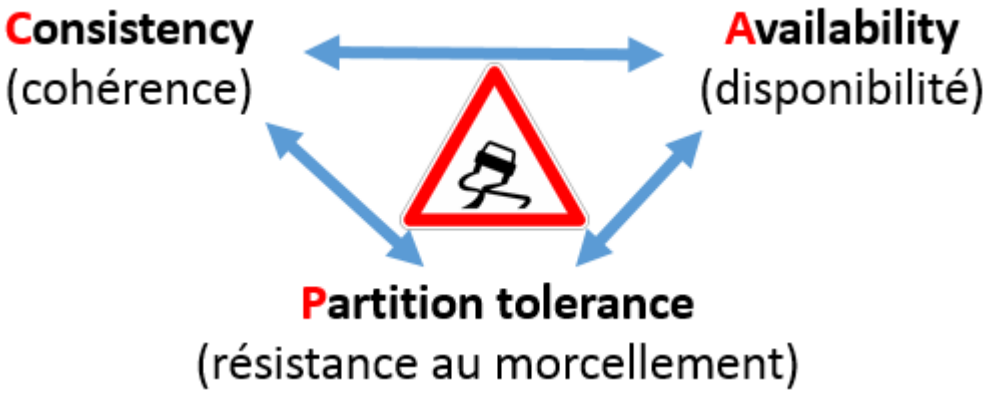
Requêtes OLTP : approche transactionnelle classique des BdD Relationnelles

Requêtes OLAP : besoin en analyse de données, peu favorable aux BdD Relationnelles

Le problème CAP

Base toujours perçue cohérente, même pendant des mises-à-jour

Disponibilité garantie en l'absence de pannes

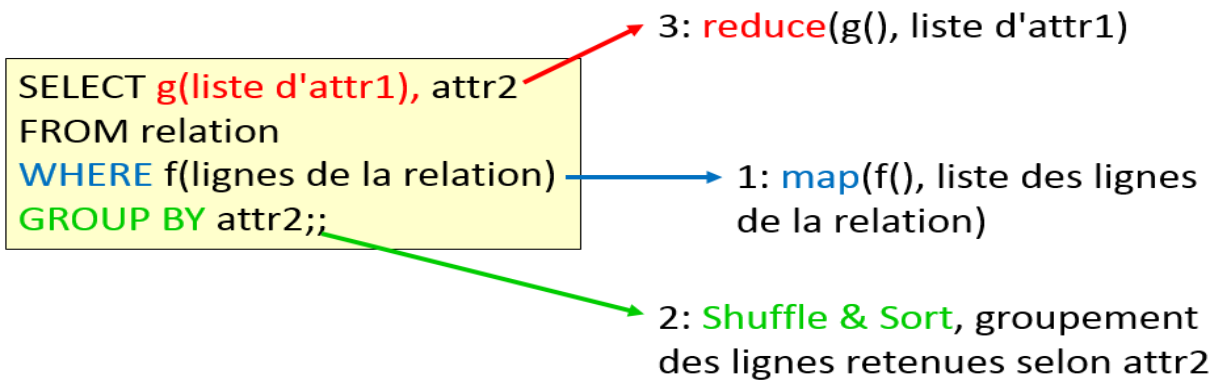


Résistance aux pannes en distribué

- **En mode distribué à large échelle :**
  - On n'a jamais toutes les data à jour en même temps **OK**
  - On ne peut pas différer des requêtes chaque fois qu'on fait une maj (on ne traiterait jamais de requêtes!) **OK**
  - On ne peut pas arrêter de fonctionner dès que des parties de la BdD sont en pannes **NON**

Principes fondateurs du NoSQL

- C'est Google qui a tout d'abord été confronté au stockage et à la gestion de très gros volumes de données (GFS en 2003 et MapReduce en 2004).
- Résolution d'une requête SQL type en Map-Reduce



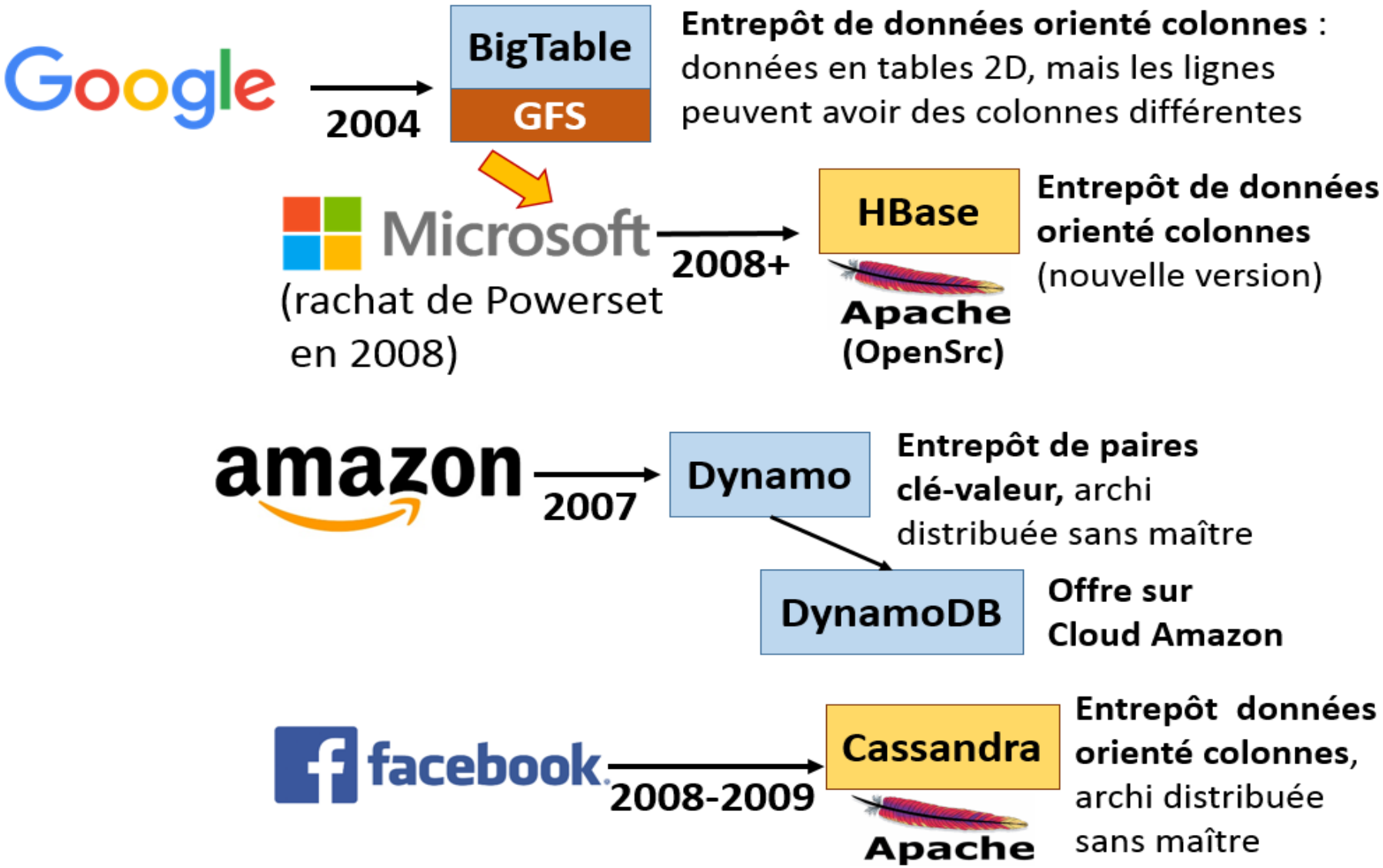
- Une opération Map-Reduce permet de refaire un "Select...From...Where...Group by...".
- Elle peut fonctionner même si les éléments stockés ne sont pas dans une BdD relationnelle.
- Dans ce cas, un SGBD relationnel ne peut pas être employé, alors qu'un mécanisme de Map-Reduce au dessus d'un système de fichiers distribués sera utilisable.

Principes fondateurs du NoSQL

Hadoop : un premier Map-Reduce distribué open-source

- Repose sur le système de fichier distribué HDFS (Hadoop Distributed File System), et sur un mécanisme de Map-Reduce distribué exploitant HDFS.
- Environnement de développement de Java.
- Rapidement adopté, Yahoo ! et Facebook en 2012: des dizaines de milliers de machines exploitées par Hadoop.
- A l'exécution, les noeuds de données sont transformés en noeuds de traitement, et accueillent des processus map et des processus reduce.
- La partie la plus complexe d'Hadoop, qui assure le routage des données de sortie des processus map vers les entrées des processus reduce, n'a pas à être re-développée

Les premières BdD NoSQL



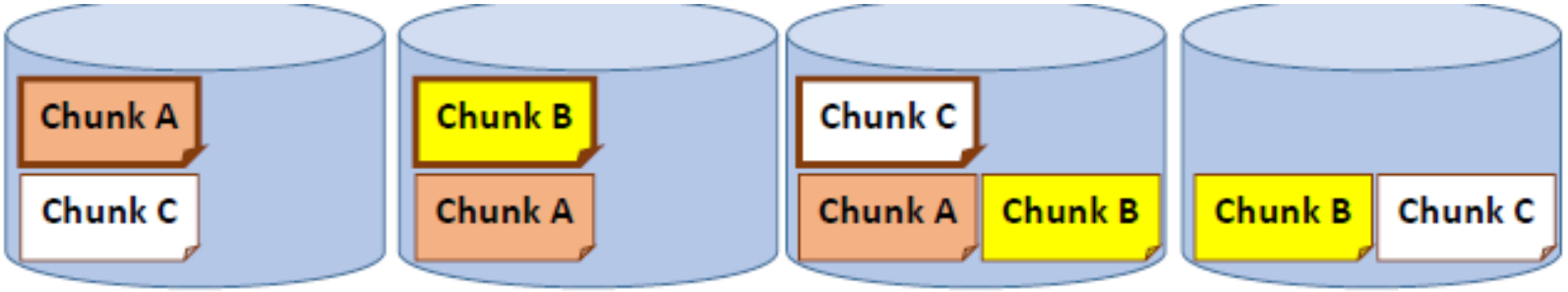
Résumé « NoSQL *vs* SQL »

- **Une grande souplesse dans la nature des données stockées.** Par exemple, les lignes d'un entrepôt orienté colonnes n'ont pas forcément les mêmes colonnes, les valeurs d'un entrepôt clé-valeur peuvent être des fichiers textes (en résumé : pas de schéma!)
- **L'exploitation d'un très gros volume dans des temps raisonnables.** Par exemple, la mise à jour d'une valeur (d'un document) se fera de manière atomique (en exclusion mutuelle avec des lectures de la valeur), mais une suite de mises à jour sur plusieurs documents ne pourra pas se faire de manière atomique. Il sera donc possible de lire un ensemble de valeurs en cours de mise à jour et légèrement incohérentes
- **L'augmentation des performances par la distribution massive du stockage et des traitement**. Par exemple, en utilisant suffisamment de PC pour avoir assez de mémoire, on peut charger toute la base en mémoire, et accélérer le traitement des requêtes supportées en NoSQL.

On s'éloigne donc de la rigueur des BdD relationnelles pour plus de souplesse, plus de volume, et plus de vitesse.

Sharding et réplication

- Le *sharding* est un découpage des différentes lignes d'une table par tranches dans différents *chunks* (au lieu de distribuer les colonnes)
- Un fichier/une table est découpé en morceaux (*chunks*) distribués pour permettre:
  - des accès parallèles plus rapides (répartir la charge)
  - des stockages plus volumineux (et extensibles)
- Les *chunks* sont répliqués sur des noeuds différents pour la tolérance aux pannes



Classification des technologies NoSQL

Classification des différents moteurs de BdD NoSQL selon le type de structures de données qu'ils manipulent.

- **Stockage / Entrepôt de paires clé-valeur**
  - (Redis, Riak)
- **BdD orientés documents**
  - (MongoDB)
- **BdD orientés colonnes**
  - (BigTable, HBase, Cassandra)
- **BdD créées pour des index inversés**
  - (Elasticsearch)
- **BdD orientés graphes**
  - (Neo4J)

Entrepôts (NoSQL) de paires clé-valeur

La solution la plus extensible (« *scalable* »), mais simple/pauvre :

- Statistiquement, la plupart des applications demandent à lire des données à partir de leurs identifiants
  → engendre le besoin de BdD stockant des paires clé-valeur
- Le composant clé de ces bases est leur fonction de hachage
  → distribution et recherche des données dans le système distribué
- Mécanisme final très efficace mais avec peu de fonctionnalités
  → développements dans la couche applicative, en *Map-Reduce*

Remarque : Initialement des *valeurs* binaires et opaques, puis des valeurs structurées hiérarchiques analysables.

- ***Exemples:*** **Redis, Riak**

BdD NoSQL orientées documents

- **On associe des clés à des documents à structure hiérarchique**
  - Les valeurs ne sont plus opaques
  - On peut manipuler les champs des données
  - Manipulation de doc web au format HTML/XML, ou doc JSON
- **On stocke des données prêtes à être interrogées sans jointure:**
  - Exploitation rapide
  - Mais la jointure doit être faite lors de l'écriture
  - Si besoin de croiser des informations : **plus complexe et lent**
- **Remarque 1:** L'utilisation de documents structurées non opaque permet de les analyser et de produire des index inversés
- **Remarque 2:** BdD devenues de très grandes tailles, proches des entrepôts de paires clé-valeur

BdD orientées documents → Entrepôts clé-valeur

- ***Exemple: MongoDB***

BdD NoSQL orientées colonnes

- **Stockent des tables 2D de *clé - ensemble de valeurs***
- **Ressemblent à des tables relationnelles, mais bcp plus souples**
  - Les lignes peuvent avoir des colonnes différentes et en nombres différents
  - Les colonnes d'une ligne peuvent évoluer dynamiquement en nombre et en nom
  - Pas de champ « NULL » contrairement à une table relationnelle (pas de colonne inutile dans une ligne)
- **Requêtes simples** (minimalistes) *vs* BdD SQL, ou traitements complexes en *Map-Reduce*…
- **Remarque :** BdD NoSQL conçues pour stocker des associations ***one-to-many*** comme on en trouve très fréquemment sur le **web** !

- ***Exemples: BigTable, HBase, Cassandra***

BdD NoSQL d'index inversés

- **Un index inversé** est indispensable pour traiter rapidement les requêtes de recherche de documents par mots clés
- Mais l'index inversé **peut devenir TRES** volumineux (plus que les documents analysés) :
  - il faut le compresser
  - mais pas trop pour que la décompression à la volée soit rapide
  - et/ou trouver un format de compression permettant de travailler dans le format compressé
- **Les BdD NoSQL spécialisées en index inversés apportent:**
  - des algorithmes optimisés pour construire ces index
  - des algorithmes de compression/décompression adaptés

- ***Exemple: Elasticsearch***

BdD NoSQL de graphes

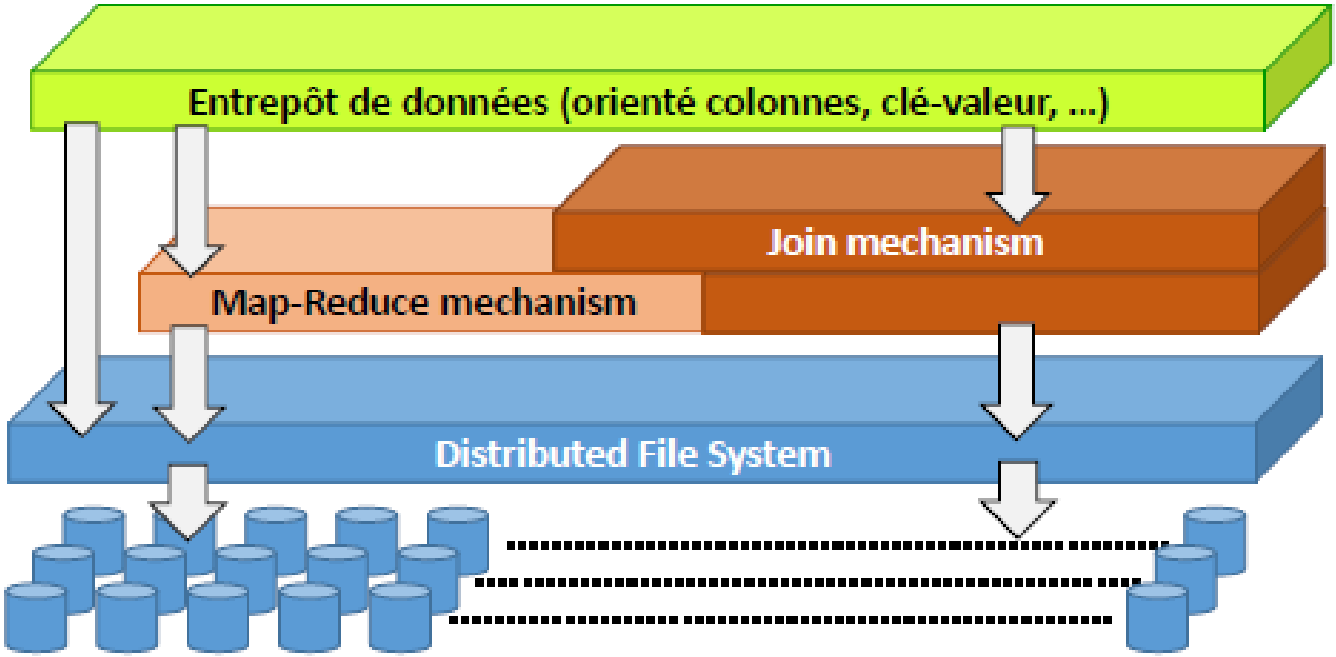
- **Spécialement adaptées pour fouiller le web et les réseaux sociaux**
  - Apportent des stockage de graphes efficaces (par références)
  - Apportent des algorithmes d'analyse de graphes optimisés et adaptés au stockage réalisé
- Les autres bases NoSQL pourraient stocker des graphes mais seraient moins efficaces pour les analyser
- ***Exemple:*** **Neo4j**
  - technologie très complète et très efficace pour stocker/fouiller/analyser des graphes
  - Rapide (codage par ''pointeurs'')
  - Stockage compact
  - Répliquée sur cluster, mais pas distribuée…

Principes d'architecture NoSQL

- **Architecture de principe d'une BdD NoSQL :**

Au début les « entrepôts de données » (NoSQL) n'offraient pas les fonctionnalités de SQL… il fallait les interroger en Map-Reduce…

et rapidement il a fallu porter SQL sur ces environnements

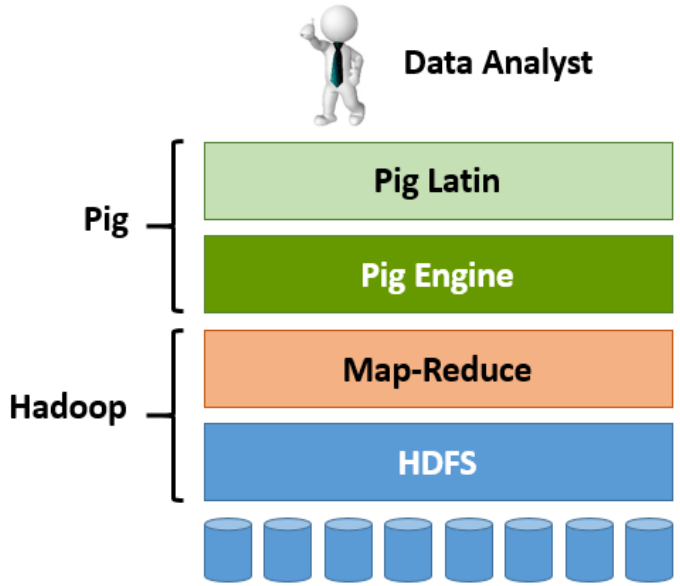


***join* natif,**
ou bien
***join* au dessus**
de *Map-Reduce*

NoSQL → *SQL-like* : PIG

- En 2006 Yahoo! développe **PIG** pour faire « un peu comme SQL » au dessus d'Hadoop :
- pour simplifier le travail de ses équipes de *data-analysts*
  - devient OpenSource en 2007
  - Langage de requêtes et de programmation : « *Pig-Latin* »
- **Exemples de commandes :** FILTER, GROUP BY, JOIN, SORT FOREACH…
- … et chaque commande est traduite

en un ***graphe d'opérations Map-Reduce***



NoSQL → *SQL-like: PIG*

```
grunt> DUMP ETUDIANT;
(Dupond, Alphonse, 10, Metz)
(Dupond, Grégoire, 9, Rennes)
(Durant, Hubert, 12, Gif)
(Talon, Achille, 15, Gif)

grunt> ETUDIANT2 = FOREACH ETUDIANT GENERATE $0, $1, $2+1 ;

grunt>DUMP ETUDIANT2;
(Dupond, Alphonse, 11)
(Dupond, Grégoire, 10)
(Durant, Hubert, 13)
(Talon, Achille, 16)
```

```
grunt> DUMP VEHICULE;
(AA 123 AB, bleue, Twingo)
(AZ 451 GT, noire, C3)
(BC 634 FY, jaune, Twingo)
(DE 398 AA, blanche, DS4)

grunt>DUMP CONSTRUCTEUR;
(Twingo, Renault)
(C3, Citroen)
(DS4, Citroen)

grunt> VOITURE = JOIN VEHICULE BY $2, CONSTRUCTEUR BY $0;

grunt>DUMP VOITURE;
(AA 123 AB, bleue, Twingo, Twingo, Renault)
(AZ 451 GT, noire, C3, C3, Citroen)
(BC 634 FY, jaune, Twingo, Twingo, Renault)
(DE 398 AA, blanche, DS4, DS4, Citroen)
```

```
grunt> VOITURE2 = FOREACH VOITURE GENERATE $0, $1, $2, $4 ;

grunt> VOITURE3 = ORDER VOITURE2 BY $2, $0 DESC ;
(AZ 451 GT, noire, C3, Citroen)
(DE 398 AA, blanche, DS4, Citroen)
(BC 634 FY, jaune, Twingo, Renault)
(AA 123 AB, bleue, Twingo, Renault)
```

NoSQL → SQL-like : Hive

- En 2005 Facebook crée **Hive** : « **langage déclaratif proche de SQL** », open source en 2008
- Pour des *data-analysts* qui ne peuvent pas programmer du *Map-Reduce* (en Java…)
- *Hive* propose un langage ***HiveQL*** qui permet d'écrire des requêtes proches de SQL (avec une forte influence de MySQL).

Les données manipulées par HiveQL sont structurées en tables où les lignes ont des structures identiques, et chaque attribut peut avoir un type primitif (scalaire) classique, ou un type complexe : un tableau, une map ou une structure.

```
// Create new table from result of a SELECT command
hive> INSERT OVERWRITE TABLE user_active
SELECT user.*
FROM user
WHERE user.active = 1;

// Projection on one index of an array attribute
hive> SELECT pv.friends[2]
FROM page_views pv;

// Join 2 tables and write in a new one
hive> INSERT OVERWRITE TABLE pv_users
SELECT pv.*, u.gender, u.age
FROM user u JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2008-03-03';
```