# Les Big data

### Les 5 “V”:

* + **Volume** : énorme!!
	+ **Variabilité** : On stock mais les données ne sont pas toujours pareilles (stockage de différente forme)
	+ **Véracité** : Certaines données sont vraies et d’autres fausses
	+ **Vélocité** : La vitesse à laquelle l’information entre
	+ **Valeur**: La valeur des données

### Une image qui résume tout…


# NoSQL

### Définition

Il n’y a pas de définition définitive de ce que veut dire NoSQL.

Donc il ne faut pas se fier à certains qui disent que ça veut dire : No SQL ou Not Only SQL

Au début des années 2000, avec les bases de données grandissantes et la sortie de BigTable et Dynamo pour conserver leurs données, plusieurs projets expérimentaux font leur apparition.

En 2009, lors d’une exposition à San Francisco, un organisateur a voulu faire un rassemblement de personnes développant de nouveaux types de bases de données, et le mot NoSQL a été trouvé pour le nom du rassemblement. Court et facile à retenir. Parfait pour les discussions Twitter #NoSQL. Le nom a été retenu à partir de ce moment-là.

Il existe plusieurs variances de BD NoSQL. Ces variances ont cependant des points en commun :

1. Elles n’utilisent pas de modèle relationnel
2. Elles sont *open source*
3. Elles sont créées pour sauvegarder un nombre important de données
4. Elles n’ont pas de schéma (on sauvegarde ce que l’on veut avec la structure que l’on veut)
5. Elles s’utilisent facilement dans un *cluster*

NoSQL n’est cependant pas parfait.

1. Il n’y a généralement pas de transaction aussi solide que comme dans les BDRs.
2. Pour les bases de données où il y a plusieurs liens entre les documents/objets/tables, il vaut peut-être mieux utiliser des BDR.
3. Comme il n’y a pas de schéma à définir, il faut faire attention à ce que l’on sauvegarde pour ne pas tout mélanger et rendre la lecture plus difficile

### Quand utiliser NoSQL

De façon générale, on peut penser à deux raisons pour vouloir utiliser NoSQL comme solution de persistance.

1. Pour une sauvegarde massive de données
* La plupart des bases de données NoSQL sont faites pour être facilement utilisables en *cluster*, comparativement aux BD relationnelles (BDR).
1. Pour simplifier la sauvegarde de documents
* Pensons à une application de sauvegarde de factures. Avec NoSQL, on peut créer la base de données en 2 minutes. Avec BDR, on doit faire plusieurs tables et liens pour la sauvegarder.

### Les types de clusters

Lorsqu’il y a trop de données et trop d’achalandage pour une base de données déjà optimisée, on doit penser à une solution pour qu’elle ne croule/plante pas.

Il y a 3 manières pour la rendre plus performante.

**Acheter/louer un serveur de base de données plus performant**

Solution onéreuse, et il y a des limites de performance pour un ordinateur

**Le partitionnement**

Séparer la base de données en plusieurs fragments, sur plusieurs serveurs.

Exemple d’une application sauvegardant des usagers :

Sharding (horizontal partitionning)

* Les usagers dont le prénom commence par la lettre A à F sur le serveur #1
* Les usagers dont le prénom commence par la lettre G à M sur le serveur #2
* Les usagers dont le prénom commence par la lettre N à Z sur le serveur #3

Solution pratique, mais qui peut avoir un problème, car souvent on veut avoir accès à un ensemble de données qui se retrouvent sur plusieurs serveurs. Par exemple : Trouver **tous** les usagers qui habitent au Canada. Aussi, si un des serveurs plante, la base de données devient potentiellement hors service.

Vertical partitionning

* Prendre la colonne id, nom et prénom de la table usager sur le serveur #1
* Reprendre id, plus la date de naissance, le numéro d'assurance sociale, etc. sur le serveur #2
* …

**La réplication**

La réplication traditionnelle est comme suit :

* Il y a un serveur « master », qui est responsable de toutes les écritures dans la base de données
* Plusieurs serveurs esclave (« slaves »), qui sont responsables de toutes les lectures de la base de données.
	+ Ceux-ci essaient de suivre les fichiers de journalisation du master afin de mettre leur propre base de données à jour.

Évidemment, si la base de données contient majoritairement des écritures à la base de données, la réplication ne sert à rien. Aussi, il peut se passer un délai entre la mise à jour des slaves par rapport au master, ce qui peut être problématique.

Un avantage d’avoir une réplication est le fait que les esclaves (« slaves ») sont des copies du master, donc si cette dernière tombe hors ligne, un esclave peut devenir automatiquement un master et la base de données continue à être opérationnelle.

Solution mixe

L’idéal, pour certains contextes, est de combiner le sharding et la réplication.

### NoSQL : Les 4 types de modèle de données

**Key-Value**

Riak, Redis, Project Voldemort, LevelDB, Berkeley DB…

Pour être bref, ce modèle de base de données est un peu comme un Hashtable ultra performant et persistant.

Exemple (Riak)

Bucket bucket = getBucket("shoppingCart"); // Base de données ShoppingCart

bucket.store(“fred”, myProducts).execute(); // Sauvegarde liste de produits. La clé

 est fred

ArrayList obj = bucket.fetch("fred", ArrayList.class).execute(); // Aller chercher

 les produits

**Column-Family**

BitTable (de Google), Cassandra, Amazon SimpleDB, …

Une hashtable, où la valeur est un groupe de colonnes qui contiennent des informations

Exemple

Fred : {

 adresse : {

 Ville : "Montréal",

 Tel : "5145556565"

},

info : {

 nom : "Frédéric Thériault",

 annee : 1981

}

 }

Fred est la clé.

Adresse et info sont 2 colonnes

Ville/tel/nom/annee sont des informations

**Document**

CouchDB, MongoDB, RavenDB, …

Permet de sauvegarder des documents, et de les interroger via MapReduce.

{

nom : "Fred",

age : 35,

status : "active",

cours : ["DBA", "Web", "Linux"]

}

Contrairement à Key-Value, on peut faire des recherches dans les propriétés du document (par exemple, "où le nom est égal à xyz"). Avec Key-value, on peut seulement interroger sa clé (ex : key > 111).

**Graph**

Neo4j, FlockDB, HyperGraphDB, …

Entités et leurs relations

Exemple (Neo4j)

Node fred = graphdb.createNode(); // Création d’un nœud du graphique

fred.setProperty("name", "Frederic");

fred.setProperty("emploi", "Enseignant");

Node jc = graphdb.createNode(); // Création d’un nœud du graphique

jc.setProperty("name", "Jean-Christophe");

jc.setProperty("emploi", "Enseignant");

fred.createRelationship(jc, Collegue); // Frederic est un collègue de Jean-Christophe

jc.createRelationship(fred, Collegue); // Jean-Christophe est un collègue de Frédéric

// Pour aller chercher tous ceux qui ont Frederic comme collègue…

List<RelationShip> relations = fred.getRelationships(INCOMING, Collegue);

for (Relationship relation : relations) {

 ...

}

###