

# BASES DE DONNÉES NOSQL

PAR : PR. KAMEL BOUKHALFA  
2020

26/09/2020

1

## HISTORIQUE

- **Modèle hiérarchique (1950)**
  - Relations parent-enfants
  - Arbre hiérarchique
  - **Implémentation:** le moteur IMS (Information Management System) d'IBM
- **CodasyI (Conference on Data Systems Languages) (1959)**
  - Etablissement d'un langage d'interrogation Cobol (COmmon Business Oriented Language)
  - Langage navigationnel
  - Etablissement d'un modèle de données : modèle réseau
- **Modèle relationnel d'Edgar Frank**
  - Basé que la théorie des ensembles
  - Isoler l'accès aux données de l'implémentation physique
  - Proposer un langage déclaratif, algébrique indépendant des algorithmes de manipulation des données

26/09/2020

2

## QUELQUES RÈGLES DE CODD

- **Règle 0** : Toutes les fonctionnalités du SGBDR doivent être disponibles à travers le modèle relationnel et le langage d'interrogation.
- **Règle 1** : Toutes les données sont représentées par des valeurs présentes dans des colonnes et des lignes de tables.
- **Règle 3** : Une cellule peut ne pas contenir de valeur, ou exprimer que la valeur est inconnue, à l'aide du marqueur NULL. Il s'agit d'un indicateur spécial, distinct de toute valeur et traité de façon particulière.
- **Règle 5** : Le SGBDR doit implémenter un langage relationnel qui supporte des fonctionnalités de manipulation des données et des métadonnées, de définition de contraintes de sécurité et la gestion des transactions.

26/09/2020

3

## QUELQUES RÈGLES DE CODD

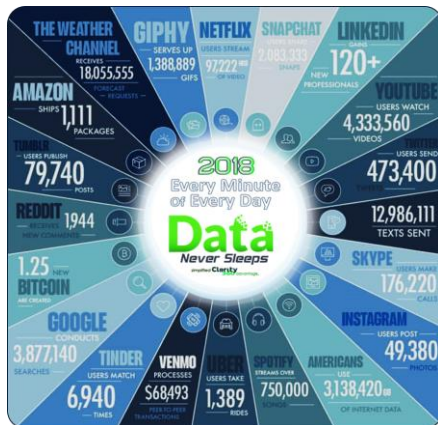
- **Règle 10** : *Indépendance d'intégrité* : les contraintes d'intégrité doivent être indépendantes des programmes clients et doivent être stockées dans le catalogue du SGBDR. On doit pouvoir modifier ces contraintes sans affecter les programmes clients.
- **Règle 11** : *Indépendance de distribution* : la distribution ou le partitionnement des données ne doivent avoir aucun impact sur les programmes clients.
- **Règle 12** : *Règle de non-subversion* : aucune interface de bas niveau ne doit permettre de contourner les règles édictées. Dans les faits, cette règle implique qu'il n'est possible d'interroger et de manipuler le SGBDR qu'à travers son langage relationnel.

26/09/2020

4

## CONTEXTE

- Nous traitons de plus en plus de données volumineuses



5

## POURQUOI CE VOLUME DE DONNÉES

- Tendance à tous numériser et sauvegarder pour en tirer des connaissances
- Numérisation des administrations : APC (état civil), Justice (Casier judiciaire), etc.
- Capteurs : température, tension, vent, humidité, etc.
- Smartphones
- Réseaux sociaux, web, etc.
- Données de localisation : GPS, parcours, chemins, etc.
- Données vitales : battements de cœur, respiration, effort, etc.
- Données médicales : rapports, radio, médicaments, etc.
- Objets connectés grâce aux protocoles NFC, Bluetooth, Zigbee : Montres, machines, équipements, etc.
- Etc.

26/09/2020

6

## BIG DATA

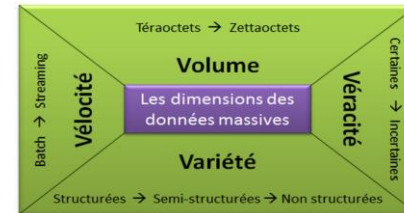
- Les big data (**mégadonnées** **données massives**) désigne des **ensembles de données** devenus si **volumineux** qu'ils **dépassent** l'intuition et les **capacités humaines d'analyse** et même celles des **outils informatiques classiques** de gestion de base de données ou de l'informatique,
- L'explosion quantitative (et souvent redondante) de la donnée numérique contraint à de nouvelles manières de voir et analyser le monde
- De nouveaux problèmes apparaissent

26/09/2020

7

## CARACTERISTIQUES DU BIG DATA

- **Les quatre V (« 4V »)**
  - **Volume** : *taille excessive des données*
  - **Vélocité** : *vitesse avec laquelle ces données sont générées/traitées*
  - **Variété** : *diversité des formats/structures des données*
  - **Véracité** : *problème de fiabilité/précision des données massives*



## CARACTÉRISTIQUES DU BIG DATA

### Volume

- Le volume des données stockées est en pleine expansion
- Les données numériques créées dans le monde : 1,2 zettaoctet/an en 2010, 1,8 en 2011, 2,8 en 2012 et s'élèveront à 40 en 2020.
- En 2013 : Twitter 7 téraoctets de données chaque jour, Facebook 10 téraoctets.
- En 2014: Facebook Hive génèrait 4 000 To de data par jour

### Variété

- Données relationnelles, semi-structurées, non structurées, texte, images, données géo-référencées, etc.

### Vélocité

- Fréquence à laquelle les données sont à la fois générées, capturées, partagées et mises à jour
- **Véracité, Valeur** : certaines / incertaines

26/09/2020

9

## EXEMPLE D'UTILISATION

- **Biologie** : Séquencement du génome humain
- **Prédiction** : changements climatiques, dégradation des sols, gaspillage, etc.
- **Militaire** : mouvement des trafiquants, drones,
- **Santé** : prédiction, propagation des épidémies (corona Virus)
- **Politique** : analyse d'opinions
- Etc.

26/09/2020

10

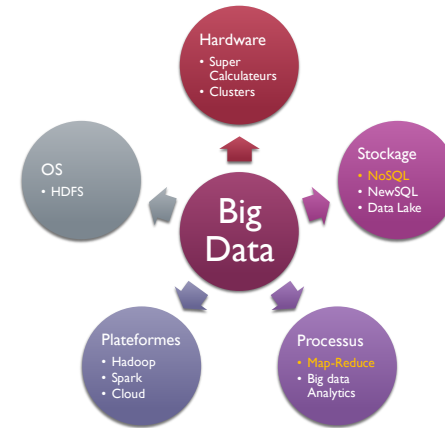
## INFLUENCE DU BIG DATA

- Comment gérer ce volume de données très important
  - Capturer
  - Stocker
  - Rechercher
  - Partager
  - Analyser
  - Visualiser

26/09/2020

11

## INFLUENCE DU BIG DATA

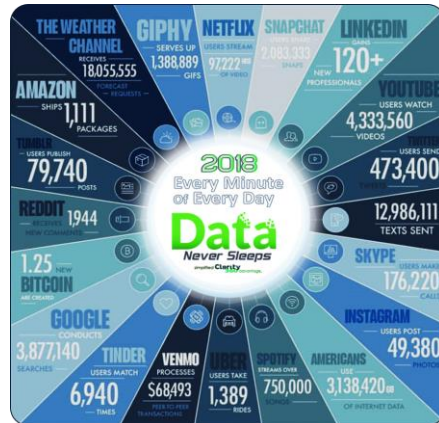


26/09/2020

12

## BIG-DATA → HARDWARE

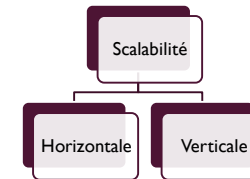
- Nous traitons de plus en plus de données volumineuses
- Les machines actuelles ne suffisent plus pour traiter ce volume de données
- Solution : booster les machines
- Scalabilité



13

## SCALABILITÉ

- La capacité d'un produit à **s'adapter** à un changement d'**ordre de grandeur** de la demande (**montée en charge**), en particulier sa **capacité à maintenir ses fonctionnalités** et ses **performances** en cas de **forte demande**,
- La **scalabilité** est tirée du domaine du génie logiciel pour définir le changement d'échelle dans le cas d'accroissement de charge de données par un système informatique sans pour cela faire impacter ses performances. (solutions hardware et logiciel)



26/09/2020

14

## SCALABILITÉ VERTICALE

- Utiliser un ordinateur qui offre de nombreuses possibilités d'ajout de pièces, sur lequel il est possible de mettre une **grande quantité de mémoire**, de **nombreux processeurs**, plusieurs **cartes mères** et de **nombreux disques durs**.
- Effectuée en faisant une évolution hardware (CPU plus rapide, plus de RAM, Disques volumineux, etc).
- Limitée par le nombre de CPU, la RAM et les capacités maximum des disques configurées sur une seule machine
- Par exemple un **ordinateur Sun Enterprise** peut contenir jusqu'à **64 processeurs**, **16 cartes mères**, **64 Go de mémoire** et **des baies de stockage**. L'ensemble tout équipé peut coûter jusqu'à **1 million de dollars**

26/09/2020

15

## SCALABILITÉ VERTICALE - SUPERCALCULATEURS

- Un **superordinateur** ou **supercalculateur** est un ordinateur conçu pour atteindre les plus hautes performances possibles avec les techniques connues lors de sa conception, en particulier en ce qui concerne la vitesse de calcul.
- La science des superordinateurs : « calcul haute performance » (*High-Performance Computing* ou HPC).
  - Partie *hardware* (conception électronique de l'outil de calcul)
  - Partie *software* (adaptation logicielle du calcul à l'outil).
  - Vitesse en FLOPS (*floating-point operations per second* : opération en virgule flottante par seconde).

Exemples :

1939 : Z2 (Konrad Zuse, Allemagne): 5 FLOPS

2016 : TaihuLight (NRCCPC, Chine), 40 960 Mproc : 93.01 Péta FLOP ( $10^{15}$  Flops)

2018 : Behold Summit (IBM/NVIDIA, USA), 36864 : 200 Péta FLOPS,

26/09/2020

16



## SCALABILITÉ VERTICALE

### ■ Avantage

- Augmentation des performances avec la même machine

### ■ Inconvénients

- Limitation techniques/technologiques des ressources : RAM, Proc, DD, etc.
- Augmentation du prix de la solution
- Accès très limité aux pays sous développés

### ■ Solution : Scalabilité Horizontale


26/09/2020

17

## SCALABILITÉ

### Vertical Scaling

  
1 CPU / 1 GB RAM  
~ \$10/mo

  
2 CPU / 2 GB RAM  
~ \$20/mo

  
4 CPU / 8 GB RAM  
~ \$80/mo

### Horizontal Scaling

  
1 CPU / 1 GB RAM  
~ \$10/mo

  
2 x (1 CPU / 1 GB RAM)  
~ \$20/mo

  
4 x (1 CPU / 1 GB RAM)  
~ \$40/mo

26/09/2020

18

## SCALABILITÉ HORIZONTALE

- Ajouter des ordinateurs « **Commodity Hardware** » pour faire face à une demande accrue d'un service.
- La méthode la plus courante est la répartition de charge par utilisation d'un **Cluster de serveurs**.
  - Nécessite une **distribution** de la BD et la **réplication**
  - Limitée par les mises à jour et les communications réseaux

26/09/2020

19

## CLUSTERS

- Regrouper plusieurs ordinateurs indépendants appelés nœuds (*node* en anglais), afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur
  - Augmenter la disponibilité ;
  - Faciliter la montée en charge ;
  - Permettre une répartition de la charge ;
  - Faciliter la gestion des ressources (processeur, mémoire vive, disques durs, bande passante réseau).



26/09/2020

20

## BIG DATA → PROCESSUS

- Le big data a influencé sur la manière de traitement des données
- Le traitement séquentiel ne suffit plus pour traiter les données volumineuses
- Il faut un nouveau paradigme de programmation qui prend en charge
  - La distribution des données
  - La parallélisation du traitement
  - La réplication des données
  - La montée en charge
- **Solution : Map-Reduce**

26/09/2020

21

## MAP-REDUCE

- MapReduce est un **modèle de programmation** conçu spécifiquement pour **lire, traiter et écrire des volumes de données très importants**.
- Les programmes adoptant ce modèle sont automatiquement **parallélisés** et exécutés sur des **clusters (grappes)** d'ordinateurs.
- MapReduce consiste en deux fonctions **map()** et **reduce()**.

26/09/2020

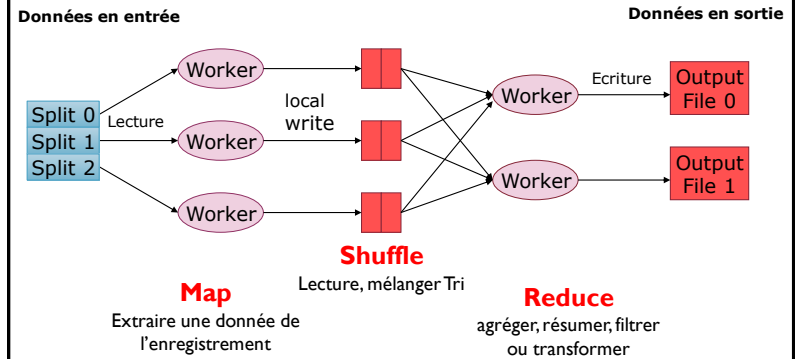
22

## PROBLÈME TYPIQUE TRAITÉ PAR MR

- Lecture d'un volume important de données
- **Map**: Extraction d'une information utile dans chaque enregistrement
  - Entrée : Texte
  - Sortie : <clé1, val1>, <clé2, val2>, ..., <clé<sub>n</sub>, val<sub>n</sub>>
- **Shuffle** Mélanger et Trier : {<clé1, val1>}, {<clé2, val2>}, ..., {<clé<sub>n</sub>, val<sub>n</sub>>}
- **Reduce**: Aggréger, résumer, filtrer ou transformer
  - Entrée : {<clé<sub>i</sub>, val<sub>i</sub>>}
  - Sortie : une seule valeur <clé i, Aggr(val<sub>i</sub>)>
- Ecrire les résultats

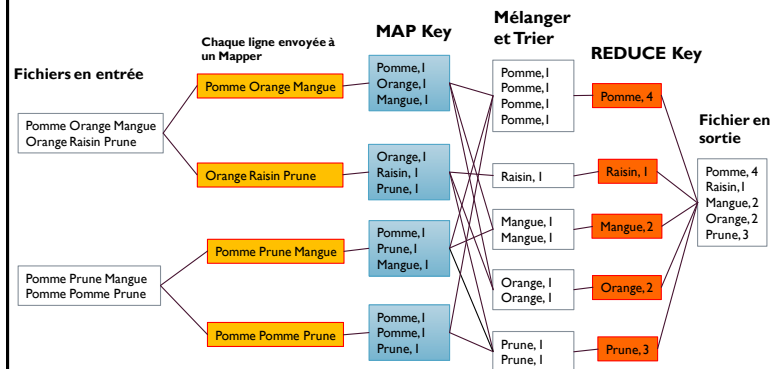
23

## MAPREDUCE



24

## EXAMPLE: WORD COUNT



25

## MAP-REDUCE DANS HADOOP

```

public class WordCount {
    public static class TokenizerMapper
        extends Mapper<LongWritable, Text, Text, IntWritable>{
        public void map(LongWritable key, Text value, Context context);
        /* Votre code pour map() ici */
    }
}

public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context);
    /* Votre code pour reduce() ici */
}

public static void main(String[] args) {
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
}
  
```

Classe interne héritant de Mapper dont on redéfinit la fonction map

Classe interne héritant de Reducer

Configuration des jobs avant lancement

26/09/2020

26

## EMERGENCE DES BD NOSQL

- Développement des centres de données
- Nouveaux paradigmes de traitement : Map-reduce

### → Nécessité de nouveaux types de BD : Not Only SQL en 2009

- Nouvelle génération de BD non relationnelles, distribuées, open source et scalable horizontalement
- Quelques exemples de solutions NoSQL

- Clé-valeur : Riak, Redis



- Orientée colonne : Cassandra, HBase



- Orientée document : MongoDB, CouchDB, Elasticsearch



- Orientée graphe : Neo4j, HypergraphDB



26/09/2020

27

## COMPARATIF RELATIONNEL - NOSQL

### Les bases de données relationnelles

#### Avantages

- La technologie est mature, le SQL est un langage standard et normalisé
- On a une garantie que les transactions sont atomiques, cohérentes, isolées et durables (principe ACID)
- La possibilité de mettre en œuvre des requêtes complexes
- Un large support est disponible et il existe également de fortes communautés.

#### Inconvénients:

- La modification du modèle établi peut être coûteuse
- L'évolutivité des performances est privilégiée de manière verticale (augmentation des ressources du serveur) bien qu'une évolutivité horizontale soit possible, cette dernière reste plus coûteuse (environnement type cluster)
- Sur un très grand volume de données (centaines-milliers de Teraoctets) le modèle peut atteindre des limites en terme de performance
- Pour certains éditeurs, le prix de licence est élevé.

26/09/2020

28

## COMPARATIF BDR - NOSQL

### Les bases de données NoSQL

- **Avantages :**
  - L'évolutivité se fait de manière horizontale (pour augmenter les performances on ajoute des nouvelles machines)
  - Les données sont distribuées sur plusieurs machines (sharding) de ce fait on évite les goulots d'étranglements lors de la récupération des données (fortes performances de lecture)
  - La représentation des données est notable par l'absence de schéma (schemaless)
  - La majorité des solutions est Open Source, néanmoins il existe des Support Pro pour répondre aux besoins des entreprises.
- **Inconvénients**
  - Il n'existe pas de langage d'interrogation standardisé : chaque éditeur a mis en place le sien
  - La mise en œuvre d'un environnement fortement transactionnel (fort besoin d'écriture) où le séquençement des écritures est primordial, reste complexe puisque l'architecture est distribuée compliquant l'atomicité et la cohérence des transactions
  - L'écriture de requêtes complexes est difficile à mettre en œuvre
  - L'offre NoSQL est segmentée en plusieurs familles où chacune répond à un besoin précis.

26/09/2020

29

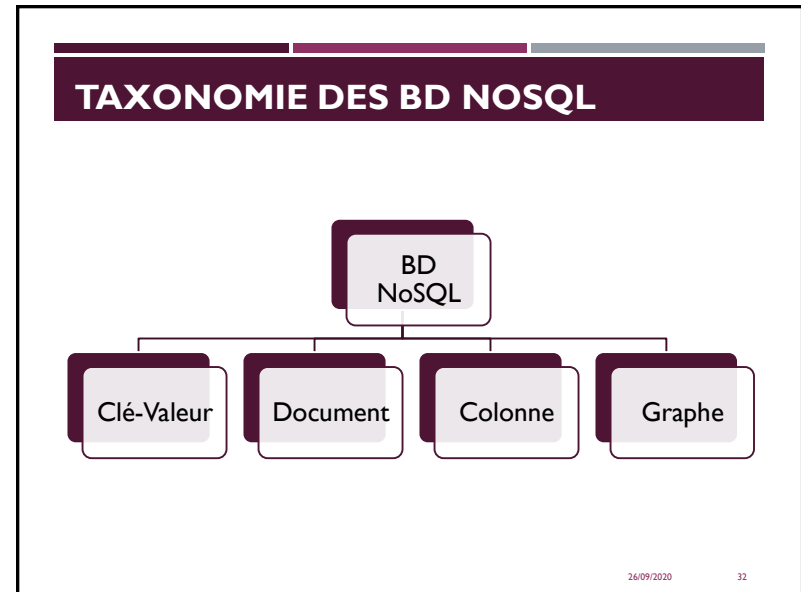
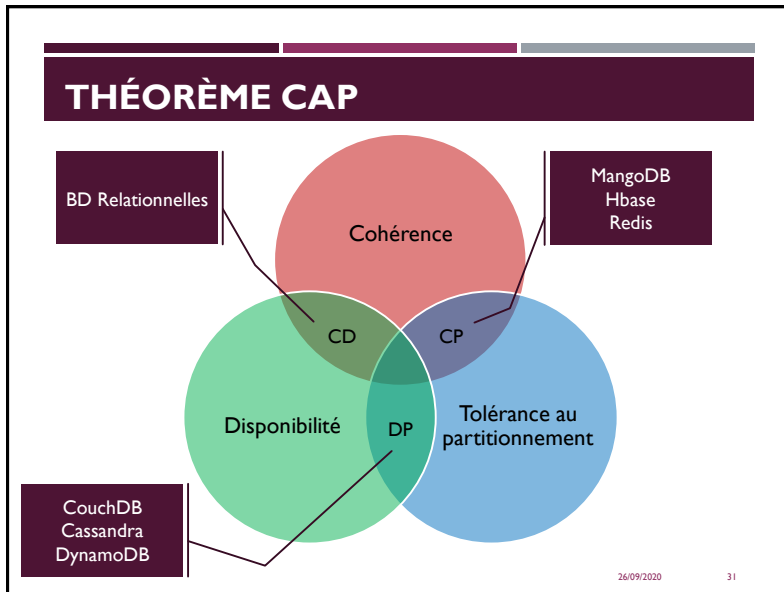
## THÉORÈME CAP (ERIC BREWER)

Un système distribué peut supporter uniquement deux caractéristiques parmi les trois suivantes :

- **Cohérence**
  - Tous les nœuds sont à jour sur les données au même moment
  - Tous les nœuds du système voient exactement les mêmes données au même moment
- **Disponibilité**
  - La perte d'un nœud n'empêche pas le système de fonctionner et de servir l'intégralité des données.
  - Garantie que toutes les requêtes reçoivent une réponse
- **Tolérance au partitionnement**
  - Chaque nœud doit pouvoir fonctionner de manière autonome
  - Aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement
  - En cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome

26/09/2020

30





## BD CLÉ-VALEUR

- Une base de données clé-valeur est un type de base de données **non relationnelle** qui utilise une méthode **clé-valeur** simple pour **stocker des données**.
- Une base de données clé-valeur stocke les données sous forme de **paires clé-valeur** dans lesquelles une clé sert d'identifiant unique.
- Les clés et les valeurs peuvent se présenter sous toutes les formes, des **objets simples** aux **objets composés complexes**.
- Les bases de données clé-valeur sont **hautement divisibles** et permettent une **mise à l'échelle** horizontale à des échelles que d'autres types de bases de données ne peuvent pas atteindre.
- Par exemple, **Amazon DynamoDB** alloue des partitions supplémentaires à une table si une partition existante est à pleine capacité et qu'un espace de stockage supplémentaire est nécessaire.

26/09/2020

33

## OPÉRATIONS

- Les BD Clé-Valeur, sont principalement faites pour le stockage temporaire,

### Quatre opérations principales

- **Création**
  - Créer un nouveau couple (clé,valeur).
  - Selon la base choisie, la valeur peut être n'importe quel objet.
- **Lecture**
  - Lire un objet en connaissant sa clé
- **Modification**
  - Mettre à jour l'objet associé à une clé
- **Suppression**
  - supprimer un objet connaissant sa clé

26/09/2020

34

## USAGE

- La plus simple et flexibles des BD NOSQL
- Associe des clés à des valeurs
- Solution aux limitations des BD relationnelles
- Les valeurs sont identifiées et accédées via la clé

Clé	Valeur
User20	Ami1,Ami2,Ami3

### Exemples

- **Réseau social** : à partir d'un utilisateur (la clé), je veux obtenir une liste de ses amis (la valeur).
- **Catalogue de livres** : le numéro ISBN (la clé) donne accès à tous les détails sur le livre (la valeur).
- **Journal d'activités** : la date d'un événement (la clé) indexe les détails de ce qui s'est passé à ce moment (la valeur).
- Pas de schéma
- La valeur stockée est gérée au niveau applicatif. Elle peut être
  - Entier, Chaîne de caractère, JSON, XML, HTML, Image, vidéo, etc.

26/09/2020

35

## CLÉS-VALEURS - AVANTAGES

- **Modélisation flexible**
  - Ne nécessite aucune structuration spécifique des données. Utiliser n'importe quel structure qui satisfait les besoins de l'application.
- **Haute performance**
  - Opérations couteuses non nécessaires (jointure, union, etc.), la recherche se fait uniquement sur la clé.
- **Scalabilité**
  - Mise à l'échelle en utilisant des commodity hardwares. La mise à l'échelle ne nécessite aucune re-conception de la BD.
- **Haute disponibilité**
  - Implémentée sur une architecture distribuée (cluster) avec une haute tolérance aux pannes
- **Inconvénients** : plusieurs objets difficilement représentés par une paire clé-valeur.

26/09/2020

36

## EXEMPLES DE BD CLÉ-VALEUR

- **DynamoDB** : une brique dans la solution d'Amazon
- **Azure Table Storage** : Microsoft
- Riak
- Redis



26/09/2020

37

## FORCES-LIMITES

- **Forces**
  - Leur simplicité, scalabilité, disponibilité
  - Très bonnes performances dans la mesure où les lectures et écritures sont réduites à un accès disque simple
- **Faiblesses**
  - Pas de requêtes sur le contenu des objets stockés
  - Non-conservation des relations entre les objets (elles ne sont pas faites pour les contextes où la modélisation métier est complexe)
- **Pourquoi une base NoSQL orientée clé valeur**
  - Elles sont beaucoup utilisées en tant que cache, pour conserver les sessions d'un site web et plus généralement pour toutes les données que l'on ne souhaite conserver que pendant un certain laps de temps, pouvant aller de quelques secondes à quelques jours.
- **Exemple** : gestion de panier d'achat (Amazon), collecte d'événements (jeu en ligne).

26/09/2020

38

## BD GRAPHES

26/09/2020

39

## BD ORIENTÉES GRAPHE

- Une base de données graphe est une **base de données spécifiquement dédiée au stockage de structures de données de type graphe.**
- Concepts de base : Nœud, arc
- une base graphe correspond à **tout système de stockage fournissant une adjacence entre éléments voisins sans indexation**
  - Tout voisin d'une entité est accessible directement par un pointeur physique
- Types de graphes modélisés
  - Homogène, hétérogène
  - Orienté, non-orienté
  - Simple, Hypergraphe, etc.

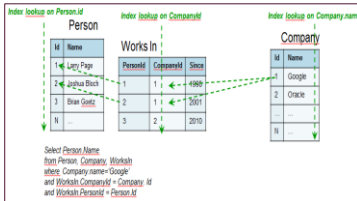
26/09/2020

40

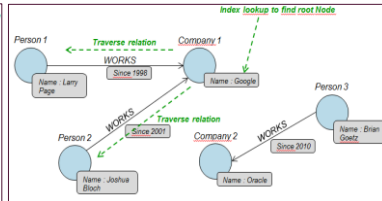
# BD GRAPHE

- Caractéristiques
  - Traiter des **données fortement connectées**
  - Gérer facilement un **modèle complexe et flexible**
  - Offrir des performances exceptionnelles pour les lectures locales, par **parcours de graphe**.

Modélisation relationnelle  
Trois jointures



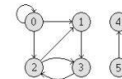
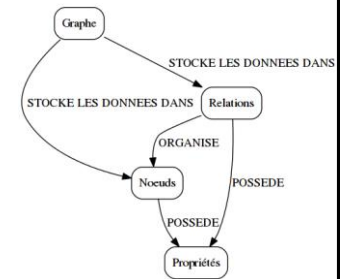
Modélisation graphe  
Parcours de liens



26/09/2020 41

# GRAPHE

- GRAPHE ORIENTÉ**
- Un graphe orienté **G** est une paire **(S,A)** où **S** est un ensemble fini de **sommets** et **A** est une relation binaire sur **S** et constitue l'ensemble des arêtes de **G**.
- Graphe non orienté : paires non ordonnées, pas de boucles



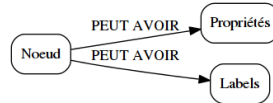
$$G = (S,A) = ((0, 1, 2, 3, 4, 5), \{(0,0), (0, 1), (0, 2), (1, 3), (2, 1), (2, 3), (3, 2), (5, 4)\})$$

26/09/2020 42

## CONCEPTS

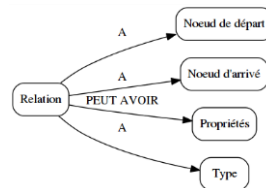
### ▪ Nœud

- Type d'objet dans lequel sont stockées les données avec Neo4j.
- Il peut porter un label et des propriétés.
- Chaque nœud dispose de pointeurs physiques menant directement à ses nœuds voisins



### ▪ Relation

- Un objet reliant 2 nœuds
- Dans le cas de Neo4j elle est forcément uni-directionnelle
- Elle peut également porter des propriétés



26/09/2020

43

## BD NEO4J

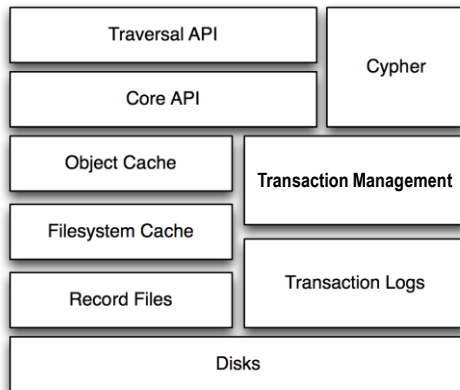
- Neo4j est une base de données **orientée graphe**
- libre et écrite en **Java**
- Développée par **Neo Technology** (société suédoise, siège aux US),
- Les premières lignes de codes datent de l'année **2000** et la version 1.0 est sortie en **2010**.
- Une des premières bases de données orientées graphes
- L'une des plus évoluées et robustes.



26/09/2020

44

## EXEMPLE NEO4J



26/09/2020

45

## NEO4J- PRINCIPALES CARACTÉRISTIQUES

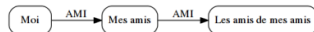
- ❑ **Transaction** : c'est une base de données transactionnelle, respectueuse des principes ACID ;
- ❑ **Haute disponibilité** : via la mise en place d'un cluster ;
- ❑ **Volumétrie** : stocker et requêter des milliards de nœuds et de relations ;
- ❑ **Cypher** : un langage de requête graphe déclaratif, simple et efficace ;
- ❑ **Schemaless** : pas de schéma préétabli.

26/09/2020

46

## LANGAGE CYPHER

- Cypher est un langage déclaratif permettant de **requêter** et **mettre à jour le graphe**.
- Inspiré du SQL : WHERE, ORDER BY, LIMIT...
- Objectif : permettre à l'utilisateur de définir des motifs (pattern), qui seront par la suite recherchés dans tout le graphe.
  - **Exemple** : si je veux les amis de mes amis, il faut décrire le motif suivant :



- En Cypher

**(moi) -[:AMI]-> (mesAmis) -[:AMI]-> (amisDeMesAmis)**

26/09/2020

47

## NŒUDS EN CYPHER

- Les nœuds sont représentés avec des parenthèses, ce qui ressemble à des cercles : ( )
- Si on veut identifier le nœud dans une requête (dans une clause WHERE par exemple), il suffit de lui donner un nom : **(Noeud\_1)**
- Pour spécifier un label, il suffit de l'ajouter comme ceci : **(monNoeud:monLabel)**
- **Exemples**
  - ( ) : n'importe quel nœud ;
  - (n:Personne) : un nœud identifié dans la variable n avec le label Personne ;
  - (n:Personne:Acteur) : un nœud identifié dans la variable n avec le label Personne et Acteur.

26/09/2020

48



## RELATIONS EN CYPHER

- Les relations sont représentées par deux tirets avec un '>', ce qui ressemble à une flèche : -->
- Si on veut identifier la relation dans une requête, on peut lui donner un nom : `-[maRelation]->`
- Pour spécifier le type de la relation, il suffit de l'ajouter : `-[maRelation:MON_TYPE]->`
- **Exemples**
  - `(a)--(b)` : n'importe quelle relation entre le nœud a et b (peu importe la direction) ;
  - `(a)-[:AMI]->(b)` : relation de type AMI depuis le nœud a vers le nœud b ;
  - `(a)-[r:AMI|CONNAIT]->(b)` : relation identifiée dans la variable r de type AMI ou CONNAIT depuis le nœud a vers le nœud b.

26/09/2020

49

## CAS PRATIQUE

26/09/2020

50

## LE SCOTLANDYARD (STÉPHANE CROZAT)

- Le Scotlandyard est un jeu de plateau multijoueur. Le plateau de jeu représente les stations de taxi, métro et bus.
- Chaque joueur dispose d'un pion qu'il déplace à l'aide de tickets (de taxi, métro ou de bus) distribués en début de partie.
- Un des joueurs, "**Mister X**" se déplace mais sans montrer sa position aux autres joueurs.
- A chaque déplacement, Mister X indique aux joueurs quel moyen de transport il a utilisé. La position de Mister X apparaît régulièrement (tous les 5 tours).
- Le but du jeu est qu'un des joueurs tombe sur la même case que Mister X avant le 25e tour.

26/09/2020

51

## CRÉATION DU GRAPHE

- Création des nœuds (100 stations)

```
create (s61:STaxi {nom : 's61'})
create (s62:STaxi {nom : 's62'})
create (s63:SBUS {nom : 's63'})
create (s64:STaxi {nom : 's64'})
create (s65:SBUS {nom : 's65'})
create (s66:STaxi {nom : 's66'})
create (s67:SMetro {nom : 's67'})
create (s68:STaxi {nom : 's68'})
create (s69:STaxi {nom : 's69'})
...
```

- Création des relations (lien entre deux stations avec moyen de transport)

```
create (s61)-[:TAXI]->(s62)
create (s61)-[:TAXI]->(s78)
create (s63)-[:BUS]->(s79)
create (s63)-[:BUS]->(s100)
create (s63)-[:TAXI]->(s64)
create (s79)-[:METRO]->(s111)
create (s79)-[:METRO]->(s67)
...
```

26/09/2020

52

## INTERROGATION

- Afficher tout le graphe
  - **Match(n) return n**
- Afficher toutes les stations de bus
  - **MATCH (n:SBus) RETURN n**
- Afficher la station bus s127
  - **MATCH (n:SBus (nom:"s127")) RETURN n**
- Afficher toutes les stations liées directement à s127 quelque soit le moyen de transport
  - **MATCH (n:SBus (nom:"s127"))-(m) RETURN n,m**
- Par taxi seulement
  - **MATCH (n:SBus (nom:"s127"))-[:TAXI]->(m) RETURN n,m**

26/09/2020

53

## SUITE

- Quelles sont les stations atteignables (en 1 trajet) depuis la station s140 en prenant n'importe quel moyen de transport.
  - **match ((nom:'s140'))-[]->(n) return n**
- quelles sont les stations atteignables (en 1 trajet) en bus depuis la station (s139)
  - **match ((nom:'s139'))-[:BUS]->(n) return n**
  - Aucun bus ne s'arrête à la station s139
- Même chose par bus
  - **match ((nom:'s139'))-[:TAXI]->(n) return n**

26/09/2020

54

## SUITE

- Un indic nous a dit que Mister X se situerait à 2 coups (taxi, bus, métro confondus) de la station (s84). Où peut-il être ?
  - `match (s1)-[]->(s2)-[]->(s3{nom:'s84'}) return s1`
  - Pour mieux voir les relations on peut afficher s1, s2 et s3
    - `match (s1)-[]->(s2)-[]->(s3{nom:'s84'}) return s1, s2, s3`

26/09/2020

55

## SUITE – PLUS COURT CHEMIN

- Le chemin le plus court entre s61 et s160, peu importe les moyens de transport utilisés.
  - `match (s61 {nom:'s61'}),(s160 {nom:'s160'}), p=shortestPath((s61)-[*]->(s160)) return p`
  - Quels sont les plus courts chemins entre s67 et s160, sans prendre de métro ?
  - `match (s67 {nom:'s67'}),(s160 {nom:'s160'}), p=allShortestPaths((s67)-[:BUS|TAXI*]->(s160)) return p`
- Nombre de Chemins
- `match (s67 {nom:'s67'}),(s160 {nom:'s160'}), p=allShortestPaths((s67)-[:BUS|TAXI*]->(s160)) return count(p)`
- Distance entre s67 et s160
- `match (s67 {nom:'s67'}),(s160 {nom:'s160'}), p=allShortestPaths((s67)-[:BUS|TAXI*]->(s160)) return length(p)`

26/09/2020

56

## BD ORIENTÉES COLONNE

26/09/2020

57

## BD ORIENTÉES COLONNES

- BD Orientées colonnes se rapprochent le plus des bases de données classiques (SGBDR).
- On y retrouve le principe de "table" avec des lignes et des colonnes
- Les bases de données relationnelles sont optimisées pour le stockage de lignes de données, notamment pour des applications transactionnelles.
- Les bases de données orientées colonnes sont, quant à elles, optimisées pour une extraction rapide de colonnes de données, par exemple pour des applications analytiques.
- les bases de données orientées colonnes sont conçues pour se dimensionner en externe à l'aide de clusters distribués bâtis sur du matériel à bas prix afin d'augmenter le débit, ce qui les rend idéales pour l'entreposage de données et le traitement de Big Data.

26/09/2020

58

## BD ORIENTÉES COLONNES

- Deux principales grosses différences

### 1. Les colonnes sont dynamiques.

Au sein d'une même table deux individus peuvent ne pas avoir le même nombre de colonnes car les valeurs nulles ne sont pas stockées

- Libération de l'espace de stockage, amélioration des performances de traitement car la volumétrie de données à traiter est plus faible.
- On a plus tendance également à ne créer qu'une seule table contenant toutes les données (et donc colonnes) dont on a besoin et non plus une multitude de tables.
- L'absence de 'jointure' entre les tables améliore également les performances.

### 2. L'historisation des données se fait à la valeur et non pas à la ligne comme dans les

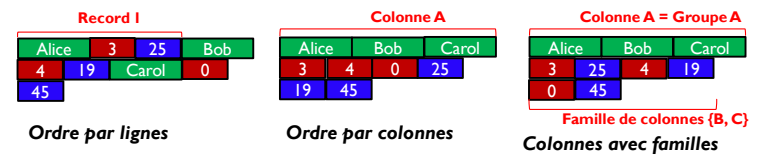
SGBDR cela empêche le stockage d'informations en doublon et de ce fait allège considérablement la base de données et les temps de calcul.

26/09/2020

59

## BD ORIENTÉES COLONNES

- Les bases de données colonnes sont hybrides entre BD relationnels et clés-valeurs
- Les valeurs sont stockées dans des groupes de plusieurs colonnes mais ordonnées selon les colonnes (en opposition à l'ordre par ligne)



**Ordre par lignes**

**Ordre par colonnes**

**Colonnes avec familles**

- Les valeurs sont interrogées par correspondance de clé

26/09/2020

60

## LES BDS ORIENTÉES COLONNE

- Les principaux concepts associés sont les suivants :
  - **Colonne :**
    - Entité de base représentant un champ de donnée
    - Chaque colonne est définie par un couple *clé / valeur*
    - Une colonne contenant d'autres colonnes est nommée *super-colonne*.
  - **Super colonne**
    - Situées dans les familles de colonnes
    - Elle présente une colonne dont les valeurs sont d'autres colonnes
    - La clé représente l'identifiant de la super colonne et la valeur
    - **Famille de colonnes** : est la liste des colonnes qui la compose
    - Il s'agit d'un conteneur permettant de regrouper plusieurs *colonnes* ou *super colonnes*.
      - Les colonnes sont regroupées par ligne et chaque ligne est identifiée par un identifiant unique.
      - Elles sont généralement assimilées aux tables dans le modèle relationnel et sont identifiées par un nom unique.

22/04/2020

61

## LES BDS ORIENTÉES COLONNE

ColumnFamily		La table								
Key	Value									
Nom de la table	SuperColumns		Les lignes de la table							
	Key	Clé de la ligne	Value Valeur de la ligne							
	Clé ligne 1	<table border="1"> <thead> <tr> <th>Column</th> <th>Ensemble de clé/valeur</th> </tr> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>		Column	Ensemble de clé/valeur	Name	Value			
Column	Ensemble de clé/valeur									
Name	Value									
Clé ligne 2	<table border="1"> <thead> <tr> <th>Column</th> <th> </th> </tr> <tr> <th>Name</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>		Column		Name	Value				
Column										
Name	Value									

22/04/2020

62

## EXEMPLE 1

Column family: PERSONNE

	Information personnelles		Informations professionnelles	
	Nom	Matricule	Fonction	Établissement
Key: 001	Belbachir	209-500	mathématicien	USTHB
Key: 002	Benadjji	250-200	Informaticien	USTHB

**PERSONNE:** une famille de colonnes

**Information personnelles & infos professionnelles:** 2 super-colonnes

**Nom, Matricule, fonction, établissements:** colonnes

**Remarque:** les colonnes et super-colonnes sont statiques

22/04/2020

63

## EXEMPLE 2

PERSONNE

	Information personnelles	Distinction		
	Nom	Nobel 1997	Medaille CNRS 1996	
Key: 001	C.C-Tannoudji	mathématicien	USTHB	
Key: 002	P.Deligne	Medaille Fields 1978	Prix Crafoord 1988	Prix Abel 2013

**PERSONNE:** une famille de colonnes

**Distinctions -:** 2 super-colonnes contenant des colonnes dynamiques

22/04/2020

64



## FORCES-LIMITES

- **Forces**
  - Flexibilité
  - Temps de traitement
  - Non-stockage des valeurs null
  - Historisation à la valeur
- **Faiblesses**
  - Non-adaptée aux données interconnectées
  - Non-adaptée pour les données non-structurées
- **Pourquoi une base NoSQL orientée colonnes?**
  - Ces bases sont particulièrement bien adaptées lorsque l'on doit stocker de très nombreux événements qui doivent être mis à jour très régulièrement. Comme par exemple:
  - Le suivi de colis (de nombreux événements dont le statut change : En préparation, en cours de livraison, livré..)
  - La récupération et l'analyse de données en temps réel issues de capteurs, IOT etc.....

26/09/2020

65

## PRINCIPALES SOLUTIONS

Quelques bases NoSQL orientées colonnes



26/09/2020

66

## CASSANDRA - CARACTÉRISTIQUES

- **Tolérance aux pannes** : les données d'un nœud sont automatiquement répliquées vers d'autres nœuds.
- Si un nœud est hors service les données présentes sont disponibles à travers d'autres nœuds. Les nœuds qui sont tombés peuvent être remplacés sans indisponibilité du service.
- **Décentralisé** :
  - dans un cluster tous les nœuds sont égaux. Il n'y pas de notion de maître, ni d'esclave, ni de processus qui aurait à sa charge la gestion
- **Modèle de données riche** : basé sur la notion de clé/valeur permet de développer de nombreux cas d'utilisation dans le monde du Web.
- **Élastique** : la scalabilité est linéaire. Le débit d'écriture et de lecture augmente de façon linéaire lorsqu'un nouveau serveur est ajouté dans le cluster.
- **Haute disponibilité** : possibilité de spécifier le niveau de cohérence concernant la lecture et l'écriture. L'écriture des données est très rapide comparée au monde des bases de données relationnelles.

26/09/2020

68

## ENTITÉS MANIPULÉES

- **Colonne** : la plus petite unité du modèle de données de Cassandra.
  - Nom jusqu'à 64Ko, Valeur jusqu'à 2 Go, Timestamp : dernière mise à jour
- **Ligne** : composée d'un ensemble de colonnes.
- **Famille de colonnes** : regroupement logique de lignes (Table en relationnel)
- **KeySpace** : regroupement de famille de colonnes (Schéma en relationnel)
- **Famille de colonnes** : regroupement logique de lignes (Table en relationnel)
- **KeySpace** : regroupement de famille de colonnes (Schéma en relationnel)

26/09/2020

69

## CQL – CASSANDRA QUERY LANGUAGE

- Utilisation de l'outil CQLSH : basé sur Python
- Création d'un Keyspace
  - `CREATE KEYSPACE cassandrademocql WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };`
  - Pour utiliser le Keyspace : `USE cassandrademocql;`
  - Pour voir les Keyspaces créés : `SELECT * FROM system.schema_keyspaces;`
  - Pour modifier un Keyspace : `ALTER KEYSPACE cassandrademocql WITH strategy_class=SimpleStrategy AND strategy_options:replication_factor=2;`
  - Pour supprimer un Keyspace : `DROP KEYSPACE cassandrademocql;`

26/09/2020

70

## FAMILLE DE COLONNES

- Création

```
CREATE TABLE Persons (
  familyName varchar,
  firstName varchar,
  age int,
  address varchar,
  PRIMARY KEY(familyName));
```

- Affichage des familles de colonnes

```
SELECT columnfamily_name FROM schema_columnfamilies
WHERE keyspace_name = 'cassandrademocql';
```

- Afficher les colonnes d'une famille de colonnes

```
SELECT column_name FROM schema_columns WHERE keyspace_name =
'cassandrademocql' and columnfamily_name = 'persons';
```

- Ajout/suppression d'une colonne

```
ALTER TABLE Persons ADD phone VARCHAR;
DROP COLUMNFAMILY Persons;
```

26/09/2020

71

## INSTANCES

- Insertion

```
USE cassandrademocql;  
INSERT INTO Persons (familyName, firstName, age, address, phone) VALUES  
( 'BARON', 'Mickael', 36, 'Poitiers', '+33549498073');
```

- Interrogation

```
SELECT * FROM cassandrademocql.Persons;
```

- Update

```
UPDATE cassandrademocql.persons SET firstname = 'Keulkeul' WHERE familyname  
= 'BARON';
```

- Suppression

```
DELETE FROM Persons WHERE familyname='BARON';
```

26/09/2020

72

## BD ORIENTÉES DOCUMENT

26/09/2020

73

## BD ORIENTÉE DOCUMENT

- La base de données orientée documents est une évolution de la base de données clé-valeur
- chaque clé n'est plus associée à une valeur sous forme de bloc binaire mais à un document dont la structure reste libre : XML, JSON
- Rendre la base de données consciente de la structure de la valeur qu'elle stocke
- On peut définir un document comme un ensemble de couples propriété/valeur, dont la seule contrainte est de respecter le format de représentation
- Autres fonctionnalités
  - Ajout, modification, lecture ou suppression de seulement certains champs dans un document
  - Indexation de champs de document permettant ainsi un accès rapide sans avoir recours uniquement à la clé
  - Requêtes élaborées pouvant inclure des prédicats sur les champs

26/09/2020

74

## EXEMPLE- DOCUMENT

```
{
  "titre": « BD NoSQL",
  "datePublication": Date("20/05/2018"),
  "auteur": « K. D.",
  "tags": [ "bigdata", "nosql" ],
  "commentaires": [ {
    "auteur": « Said",
    "commentaire": « Un très bon cours"
  }, {
    "auteur": « Karim",
    "commentaire": « A améliorer"
  }
]
}
```

26/09/2020

75

## BD ORIENTÉE DOCUMENT (1)

- Elles stockent une collection de « **documents** »
- elles sont basées sur le modèle « clé-valeur » mais la valeur est un document en **format semi-structuré hiérarchique** de type **JSON** ou **XML** (possible aussi de stocker n'importe quel objet, via une sérialisation)
- les **documents** n'ont **pas de schéma**, mais une **structure arborescente** : ils contiennent une liste de champs, un champ a une valeur qui peut être une liste de champs, ...
- elles ont généralement une **interface d'accès HTTP REST** permettant d'effectuer des requêtes (plus complexe que l'interface CRUD des BD clés/valeurs)
- Implémentations les plus connues :
  - **CouchDB** (fondation Apache)
  - **RavenDB** (pour plateformes « .NET/Windows » - LINQ)
  - **MongoDB, Terrastore, ...**

26/09/2020

76

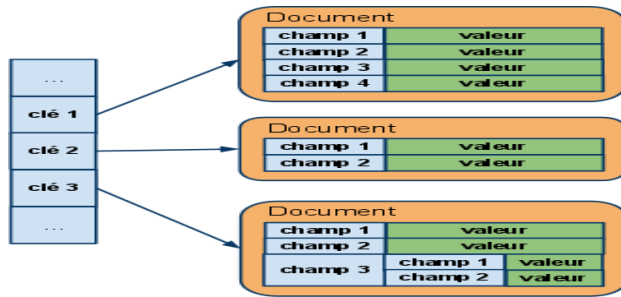
## BD ORIENTÉE DOCUMENT (2)

- Un **document** est composé de **champs** et **des valeurs associées**
- ces **valeurs** :
  - peuvent être **requêtées**
  - sont soit d'un **type simple** (entier, chaîne de caractère, date, ...)
  - soit elles mêmes **composées** de plusieurs couples clé/valeur.
- bien que les documents soient structurés, ces BD sont dites "**schemaless**" : il n'est **pas nécessaire de définir au préalable les champs** utilisés dans un document.
- les documents peuvent être très **hétérogènes** au sein de la BD
- permettent **d'effectuer des requêtes sur le contenu** des documents/objets : pas possible avec les BD clés/valeurs simples
- Elles sont principalement utilisées dans le **développement de CMS** (Content Management System - outils de gestion de contenus).

26/09/2020

77

## BD ORIENTÉE DOCUMENT: « EXEMPLE »



26/09/2020

78

## BD ORIENTÉE DOCUMENT: « FORCES ET FAIBLESSE »

- **Forces :**
  - modèle de données simple mais puissant (expression de structures imbriquées)
  - bonne mise à l'échelle (surtout si sharding pris en charge)
  - pas de maintenance de la BD requise pour ajouter/supprimer des « colonnes »
  - forte expressivité de requêtage (requêtes assez complexes sur des structures imbriquées)
- **Faiblesses :**
  - inadaptée pour les données interconnectées
  - modèle de requête limitée à des clés (et indexes)
  - peut alors être lent pour les grandes requêtes (avec MapReduce)

26/09/2020

79

## BD ORIENTÉE DOCUMENT: «UTILISATIONS PRINCIPALES »

- Les BD NoSQL de type « Document » principalement utilisées pour
  - Enregistrement d'événements
  - Systèmes de gestion de contenu
  - Web analytique ou analytique temps-réel
  - Catalogue de produits
  - Systèmes d'exploitation
  - ...

26/09/2020

80

## BD MONGODB

26/09/2020

81



## QU'EST-CE QUE MONGODB?

MongoDB, un SGBD "NoSQL", l'un des plus populaires

- fait partie des NoSQL dits "documentaires" (avec CouchDB)
- s'appuie sur un modèle de données semi-structuré (encodage JSON) ;
- pas de schéma (complète flexibilité) ;
- **un langage d'interrogation** original (et spécifique) ;
- pas (ou très peu) **de support transactionnel**.

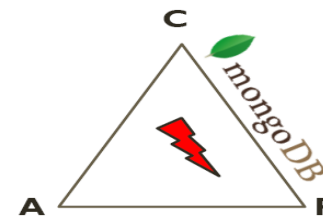
Construit dès l'origine comme un système **scalable** et **distribué**.

- distribution par partitionnement (sharding) ;
- technique adoptée : découpage par intervalles (type BigTable, Google) ;
- tolérance aux pannes par réplication.

26/09/2020

82

## MONGODB : THÉORÈME DE CAP?



MongoDB met l'accent sur la consistance et la tolérance aux pannes

26/09/2020

83

## FONCTIONNALITÉS

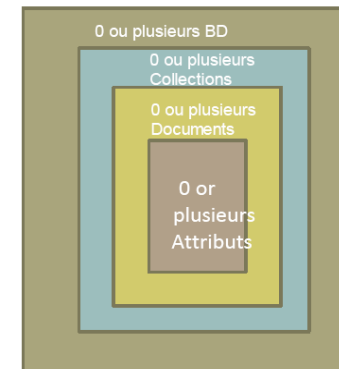
- Stockage orienté Document
- Supporte les index
- Replication & haute disponibilité
- Partitionnement automatique des données
- Langage de requêtes
- Rapidité des Mises à jour
- Supporte Map-Reduce

26/09/2020

84

## MONGODB: HIÉRARCHIE DES OBJETS

- Une instance MongoDB peut avoir zéro ou plusieurs « bases de données »
- Une base de données peut contenir zéro ou plusieurs «collections».
- Une collection peut contenir zéro ou plusieurs «documents».
- Un document peut avoir un ou plusieurs «champs».
- Les index de MongoDB fonctionnent comme leurs homologues SGBDR



26/09/2020

85

## DOCUMENT STORE

BR		MongoDB
Base de donnée	➔	Base de données
Table, Vue	➔	Collection
Tuple	➔	Document (JSON, BSON)
Colonne	➔	Champs
Index	➔	Index
Jointure	➔	Document imbriqués
Clé étrangère	➔	Référence
Partition	➔	Shard

```
> db.user.findOne({age:39})
{
  "_id" :
  ObjectId("5114e0bd42..."),
  "first" : "John",
  "last" : "Doe",
  "age" : 39,
  "interests" : [
    "Reading",
    "Mountain Biking ]
  "favorites": {
    "color": "Blue",
    "sport": "Soccer"}
}
```

26/09/2020

## MONGODB: « PROCESSUS ET CONFIGURATION »

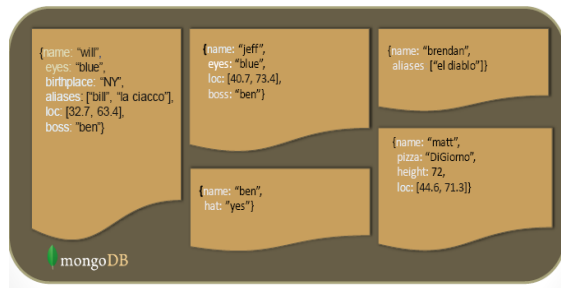
- Le processus Mongod: instance de la base de données
- Le processus Mongos: Sharding processus
  - Analogue à un routeur de base de données.
  - Traite toutes les requêtes.
  - Décide le nombre de processus mongos qui doivent recevoir la requête.
  - Mongos rassemble les résultats et les renvoie au client.
- Le processus Mongo: un shell interactif (le client)
  - Un environnement JavaScript pour l'utiliser avec MongoDB.
- Vous pouvez avoir un processus mongos pour tout le système, peu importe le nombre d'instance « mongod».
- Ou vous pouvez avoir un processus mongos local pour chaque client si vous voulez minimiser la latence du réseau.

26/09/2020

87

## MONGODB: « PAS DE SCHÉMA »

- MongoDB n'a pas besoin de schéma de données prédéfini
- Chaque document d'une collection peut avoir des données différentes



26/09/2020

88

## LE FORMAT JSON

- Les données sont dans des paires nom / valeur
- Une paire nom / valeur consiste en un nom de champ suivi de deux points, suivi d'une valeur:
  - Exemple: nom: "MERAD"
- Les données sont séparées par des virgules:
  - Exemple: nom: "MERAD", prénom: "Manel"
- Les accolades contiennent des objets
  - Exemple: {nom: "MERAD", prénom: "Manel", pays : "Algérie"}
- Un tableau est stocké entre crochets [ ]
  - Exemple [{nom: "MERAD", prénom: "Manel", pays : "Algérie"}, {nom: "LAMI", prénom: "Yanis", pays : "Algérie"}]

26/09/2020

89

## EXAMPLE

```
> db.user.insert({
  first: "John",
  last : "Doe",
  age: 39
})
```

```
> db.user.find ()
{
  "_id" : ObjectId("51..."),
  "first" : "John",
  "last" : "Doe",
  "age" : 39
}
```

```
> db.user.update(
  {"_id" : ObjectId("51...")},
  {
    $set: {
      age: 40,
      salary: 7000}
  }
)
```

```
> db.user.remove({
  "first": /^J/
})
```

26/09/2020

## GESTION DES UTILISATEURS

```
db.auth()
db.changeUserPassword()
db.createUser()
db.dropUser()
db.dropAllUsers()
db.getUser()
db.getUsers()
db.grantRolesToUser()
db.removeUser()
db.revokeRolesFromUser()
db.updateUser()
passwordPrompt()
```

26/09/2020

91

## LES OPÉRATIONS CRUD

### ▪ Create

- `db.collection.insert( <document> )`
- `db.collection.save( <document> )`
- `db.collection.update( <query>, <update>, { upsert: true } )`

```
>db.user.insert({_id:51
Nom: "LAIDI",
Prénom : "Ahmed",
age: 39
})
```

### ▪ Read

- `db.collection.find( <query>, <projection> )`
- `db.collection.findOne( <query>, <projection> )`

```
> db.user.update(
  {"_id" : ObjectId("51...")},
  {
    $set: {
      age: 40,
      salaire: 7000 } } )
```

### ▪ Update

- `db.collection.update( <query>, <update>, <options> )`

### ▪ Delete

- `db.collection.remove( <query>, <justOne> )`

26/09/2020

92

## L'OPÉRATION CREATE

**DB.collection** spécifie la collection où seront enregistrés les documents.

- `db.nom_collection.insert( <document> )`
  - Omettre le champ `_id` pour que MongoDB génère automatiquement la clé
  - Exemple `db.parts.insert ({{type: "tournevis", quantité: 15}})`
  - `db.parts.insert ( { _id: 10, type: "marteau", quantité: 1 } )`
- `db.nom_collection.update ( <requête>, <mise à jour>, { upsert: true } )`
  - Mettra à jour un ou plusieurs enregistrements dans une collection satisfaisant la requête
- `db. nom_collection.save ( <document> )`
  - Met à jour un enregistrement existant ou crée un nouvel enregistrement

26/09/2020

93

## L'OPÉRATION READ

- `db.collection.find( <query>, <projection> ).cursor` modified
  - Fournit des fonctionnalités similaires à la commande **SELECT**
  - `<query>` where condition , `<projection>` attributs dans le résultat.
  - Exemple: `var PartsCursor= db.parts.find({parts:"marteau"}).limit(5)`
  - la requête à un curseur pour limiter le nombre de lignes affichés par page
  - on peut modifier la requêtes en ajoutant `limits`, `skips`, and `sort orders`(Tri).

- Pour avoir le première ligne de résultat on utilise:
  - `db.collection.findOne( <query>, <projection> )`

```
> db.user.find ()
{ "_id" : ObjectId("51"),
  "Nom" : "LAIDI",
  "Prénom" : "Ahmed",
  "age" : 39 }
```

26/09/2020

94

## L'OPÉRATION REMOVE

- `db.nom_collection.remove(<query>, <justone>)`
  - Supprimer tous les enregistrements d'une collection qui correspondent au critère
  - `<justone>` -spécifie de supprimer seulement 1 enregistrement correspondant au critère
  - Exemple: `db.parts.remove (type: / ^ h /)` - supprime toutes les pièces commençant par h
  - `Db.parts.remove ()` -dsupprime tous les documents dans la collections de part

```
> db.user.remove({
  "nom": /^L/ })
```

26/09/2020

95

## CAS PRATIQUE

Un document correspond à un film, qui aura un nom, un réalisateur et une liste d'acteurs qui ont joué dans ce film. Un acteur a un nom et un prénom

- Commencez par créer une base de données s'appelant Cinema
  - `>use Cinema`
- Insérer le film "The godfather" réalisé par "Francis Ford Coppola" dans lequel a joué "Marlon Brando", "Al Pacino" et "Robert Duvall"
  - `db.Cinema.insert({nom:"The godfather",realisateur:"Francis Ford Coppola",acteurs:[{nom:"Pacino",prenom:"Al"},{nom:"Brando",prenom:"Marlon"},{nom:"Duvall",prenom:"Robert"}]})`
- Marvel prévoit bientôt de faire un film d'avengers avec de jeunes personnages de BD dont Kamala Khan.
- Le nom du réalisateur et des acteurs qui vont jouer dans le film ne sont pas encore connus. Il nous est toutefois demandé d'ajouter cette information dans notre collection sachant que le film s'appellera "Young Avengers". On donnera l'\_id 1 au film.
  - `db.Cinema.insert({_id:"1",nom:"Young Avengers"})`

26/09/2020

96

## CAS PRATIQUE

- Nous venons d'apprendre que le film sera finalement réalisé par "Joe Johnston".
- Ajoutez le nom du réalisateur aux informations
  - `db.Cinema.update({_id:"1"},{$set:{realisateur:"Joe Johnston"}})`
- Après les premiers castings, il a été convenu que le personnage de Kamala Khan sera interprété par l'actrice pakistanaise Sanam Jhung.
- Mettez à jour les informations dans la base.
  - Nous utiliserons l'opérateur \$push qui permet d'ajouter une nouvelle valeur dans un tableau.
  - `$push:{tab:element}` pour ajouter element à la liste des éléments dans tab
  - tab est créé si il n'existe pas
  - `db.Cinema.update({_id:"1"},{$push:{acteurs:{nom:"Jhung",prenom:"Sanam"}}})`
- Ayant du mal à trouver les acteurs adaptés aux personnages, Marvel décide d'abandonner le projet.
- Nous allons donc retirer ce dernier de notre base de données
  - `db.Cinema.remove({_id:"1"})`

26/09/2020

97



## INSERTION DES DONNÉES

```

db.Cinema.insert([
  {nom:"Goodfellas",
    realisateur:"Martin Scorsese",
    acteurs:[
      {nom:"Liam Neeson", "prenom":"Liam"},
      {nom:"Al Pacino", "prenom":"Al"},
      {nom:"Robert De Niro", "prenom":"Robert"},
      {nom:"Joe Pesci", "prenom":"Joe"}
    ],
    "nom": "The godfather",
    "realisateur": "Clint Eastwood",
    "acteurs": [
      {nom: "Swank", "prenom": "Hilary"}
    ]
  },
  {nom:"Heat", realisateur:"Michael Mann", acteurs:[
    {nom:"Deniro", "prenom:"Robert"},
    {nom:"Pacino", "prenom:"Al"}
  ]},
  {nom:"Million Dollar Baby",
    "realisateur": "Clint Eastwood",
    "acteurs": [
      {nom: "Swank", "prenom": "Hilary"}
    ]
  }
]),
BulkWriteResult({
  "writeErrors": [],
  "writeConcernErrors": [],
  "nInserted": 7,
  "nUpserted": 0,
  "nMatched": 0,
  "nModified": 0,
  "nRemoved": 0,
  "upserted": []
})

```

26/09/2020 98

## INTERROGATION

- Lister les noms des films réalisés par Clint Eastwood

```
db.Cinema.find({realisateur: "Clint Eastwood"}, { _id : 0, nom : 1 })
```

```
{ "nom": "Gran Torino" }
{ "nom": "Unforgiven" }
```

- Donnez les films dans lesquels a joué Robert Deniro
- `db.Cinema.find({'acteurs.nom': "Deniro"})`

```
{ "_id" : ObjectId("5e933a7ac5b33823d91d3167"), "nom" : "Goodfellas", "realisateur" : "Martin Scorsese", "acteurs" : [ { "nom" : "Deniro", "prenom" : "Robert" }, { "nom" : "Liotta", "prenom" : "Ray" }, { "nom" : "Pesci", "prenom" : "Joe" } ] }
```

```
{ "_id" : ObjectId("5e933a7ac5b33823d91d3168"), "nom" : "The Godfather: Part II", "realisateur" : "Francis Ford Coppola", "acteurs" : [ { "nom" : "Pacino", "prenom" : "Al" }, { "nom" : "Deniro", "prenom" : "Robert" } ] }
```

```
{ "_id" : ObjectId("5e933a7ac5b33823d91d3169"), "nom" : "Heat", "realisateur" : "Michael Mann", "acteurs" : [ { "nom" : "Deniro", "prenom" : "Robert" }, { "nom" : "Pacino", "prenom" : "Al" } ] }
```

26/09/2020 99

## CAS 2 : CAN 2015

- Fichier can.txt : 9 équipes
- Afficher toutes les équipes
  - `db.equipes.find()`

```
BulkWriteResult({
  "writeErrors": [],
  "writeConcernErrors": [],
  "nInserted": 9,
  "nUpserted": 0,
  "nMatched": 0,
  "nModified": 0,
  "nRemoved": 0,
  "upserted": []
})
```

```
> db.equipes.find()
{ "_id": ObjectId("54c2370b5ec3bd6b8f4d3f20"), "nom": "Guinee Eq",
  "matoriale", "joueurs": [ { "nom": "Ovono", "poste": "gardien",
  "best_player": false, "age": 21 }, { "nom": "Randy", "poste":
  "defenseur", "best_player": false, "age": 28 }, { "nom": "Mangu",
  "poste": "defenseur", "age": 18, "best_player": false }, {
  "nom": "Gomes", "poste": "defenseur", "best_player": false, "age":
  30 }, { "nom": "Bohale", "age": 27, "poste": "defenseur",
  "best_player": false }, { "nom": "Boula", "poste": "milieu",
  "best_player": false, "age": 22 }, { "nom": "Boullia", "poste": "b"
  "best_player": false, "age": 27 }, { "nom": "Boullia", "poste": "b"
  "best_player": false, "age": 27 } ] }
```

26/09/2020

100

## CAS 2 : CAN 2015

- affichons le nom et l'âge de tous les joueurs de l'équipe du Sénégal
  - `db.equipes.find({nom:"Senegal"},{joueurs.nom:1,joueurs.age:1,_id:0})`

```
{ "joueurs": [ { "nom": "Bouna", "age": 32 }, { "nom": "Djilobodji", "age":
  26 }, { "nom": "Kara", "age": 25 }, { "nom": "Sane", "age": 27 }, { "nom":
  "Souare", "age": 24 }, { "nom": "Badji", "age": 24 }, { "nom": "Diop", "age":
  28 }, { "nom": "Gueye", "age": 25 }, { "nom": "Kouyate", "age": 25 }, {
  "nom": "Diouf", "age": 27 }, { "nom": "Dame", "age": 29 } ] }
```

- Afficher les équipes triées selon le classement
  - `db.equipes.find().sort({classement:1})`

26/09/2020

101

## CAS 2 : CAN 2015

- Trier les équipes selon le classement FIFA ainsi que le nombre de bons joueurs (1/classement+nb\_Bon\_joueurs)
- `db.equipes.aggregate({{$project: {_id:0,nom:1,coeff:{$add:{{$divide:[1,"$classement"]},"$bestPlayers"]}}}, {$sort : { coeff:-1 } })`

```
{ "nom" : "Cote d'ivoire", "coeff" : 3.0357142857142856 }
{ "nom" : "Cameroun", "coeff" : 3.018867924528302 }
{ "nom" : "Tunisie", "coeff" : 2.0454545454545454 }
{ "nom" : "Senegal", "coeff" : 2.0285714285714285 }
{ "nom" : "Mali", "coeff" : 1.0204081632653061 }
{ "nom" : "Guinee", "coeff" : 0.02564102564102564 }
{ "nom" : "Cap Vert", "coeff" : 0.025 }
{ "nom" : "Congo", "coeff" : 0.01639344262295082 }
{ "nom" : "Guinee Equatoriale", "coeff" : null }
```

26/09/2020

102

## INDEX

- On peut utiliser des index pour optimiser les requêtes les plus fréquentes.
- Un index peut être créé sur n'importe quel attribut de documents.
- Par défaut dans MongoDB un index est positionné sur le champ `_id`.
- La syntaxe de création d'index : `db.collection.CreateIndex({clé}, {options})`
- **Options :**
  - **unique** : pas de doublon ; **TTL** : le document indexé a une durée de vie ; **name** : le nom de l'index ; **background** : l'indexation s'effectue en tâche de fond.
- Pour les champs, la valeur à positionner est `1` ou `-1` selon l'ordre voulu (ascendant ou descendant) des résultats.

26/09/2020

103

## RÉFÉRENCES

- Bernard ESPINASSE - Introduction aux systèmes NoSQL
- Kathleen Durant-Introduction to NoSQL and MongoDB
- Mickael Baron : Introduction to Cassandra