Big Data

Bases de Données NOSQL

Bases de Données NOSQL

- Bases de données NOSQL (Not Only SQL) :
  - Ce n'est pas (comme son nom le laisse supposer) No SQL (pas de SQL)
  - Privilégier donc N**O**SQL plutôt que N**o**SQL
- Bases de données non-relationnelles et largement distribuées
- Permet une analyse et une organisation rapides et ad-hoc des données de très grands volumes et de types de données disparates
- Appelées également
  - *Cloud Databases*
  - *Non-Relational Databases*
  - *Big Data Databases*
  - …
- Développées en réponse à l'augmentation exponentielle des données générées, enregistrées et analysées par les utilisateurs modernes et leurs applications

Atouts

- Principaux atouts
  - Évolutivité
  - Disponibilité
  - Tolérance aux fautes

- Caractéristiques
  - Modèle de données sans schéma
  - Architecture distribuée
  - Utilisation de langages et interfaces qui ne sont pas uniquement du SQL

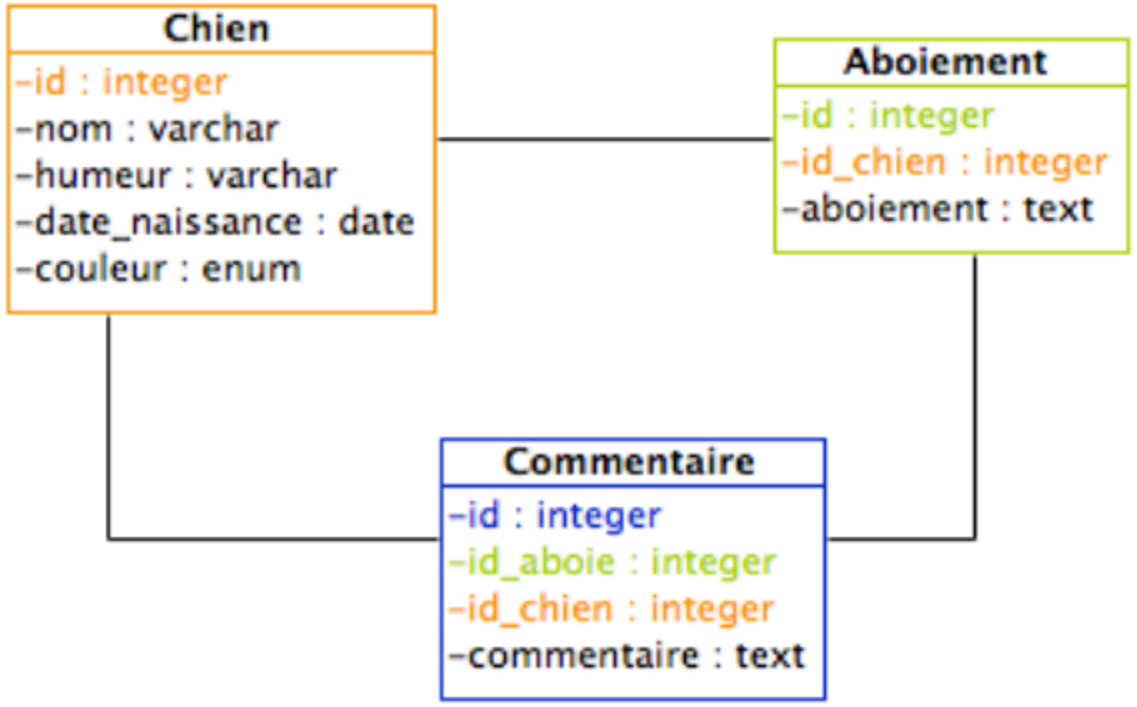
NOSQL et Big Data

- De point de vue métier, utiliser un environnement BigData et NOSQL fournit un avantage compétitif certain

- Importance des données:
  - « **Si vos données ne croissent pas, alors votre entreprise ne le fait pas non plus**«

Types de Bases de Données NOSQL

- Types des bases de données NOSQL
  - Clef/valeur
  - Orientées colonnes
  - Orientées documents
  - Orientées graphes
- Propriétés des BDR
  - ACID : *Atomicity, Consistency, Isolation and Durability*
  - Utilisation de SQL



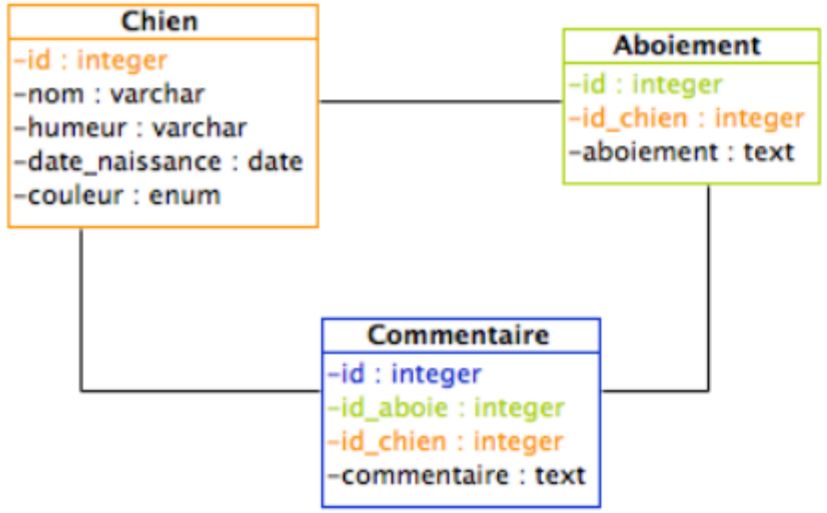
Id	Nom	Humeur	Date_naissance	Couleur
	12Stella	Heureuse	2007-04-01	NULL
	13Wimma	Faim	NULL	Noire
	9Ninja	NULL	NULL	NULL

Types de Bases de Données NOSQL
.1 Clef/Valeur)*Key/Value Store*(

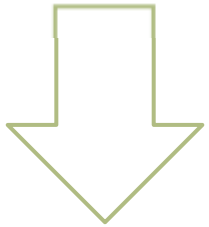
- L'un des types les plus simples, sorte de Hashmap distribuée
  Conçues pour sauvegarder les données sans définir de schéma
- Toutes les données sont sous forme de clef/valeur
  - La valeur peut être une chaîne de caractères, un objet sérialisé, blob…
  - La donnée est opaque au système: il n'est pas possible d'y accéder sans passer par la clef
- Absence de typage a un impact sur le requêtage: toute l'intelligence portée avant par les requêtes devra être portée par l'applicatif qui interroge la BD
- Communications se résumant surtout aux opérations PUT, GET et DELETE
- Objectif: fournir un accès rapide aux informations, conserver la session d'un site web…
- *Exemple* : DynamoDB (Amazon), Azure Table Storage (ATS), Redis, BerkeleyDB, Voldemort (LinkedIn(

Types de Bases de Données NOSQL

.1Clef/Valeur *(Key/Value Store(*



Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL



Clef

Chien_12

Valeur

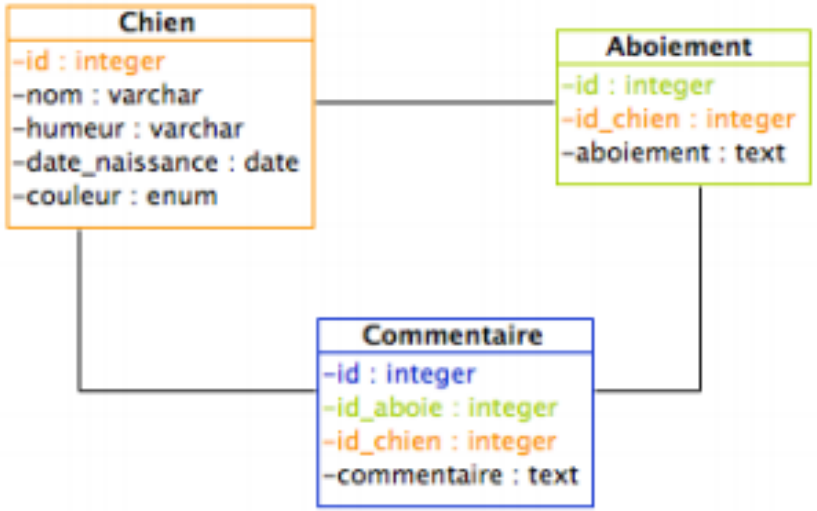
Nom_$#_Stella~~Humeur_
_#$Heureuse~~Date_naissance_
…01-04-2007_#$

Types de Bases de Données NOSQL
.2 Orientée Documents (*Document Database*(

- Étendent le paradigme clef/valeur, avec des « documents » plus complexes à la place des données simples, et une clef unique pour chacun d'eux
- Documents de type JSON ou XML
- Chaque document est un objet, contient un ou plusieurs champs, et chaque champs contient une valeur typée (string, date, binary ou array(
- Permettent de stocker, extraire et gérer les informations orientées documents)données semi-structurées(
- Avantage : pouvoir récupérer, via une seule clef, un ensemble d'informations structurées de manière hiérarchique
  - Dans les bases relationnelles, cela impliquerait plusieurs jointures
- *Exemples* : Mongo DB (SourceForge), Couch DB (Apache), RavenDB (destiné aux plateformes .Net/Windows, interrogation via LINQ(

Types de Bases de Données NOSQL

.2Orientée Documents (*Document Database(*



Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL

Clef

Chien_12

Document (V1)

```
{
        type :      « Chien »,
        nom :       « Stella »,
        humeur :    « Heureuse »,
        date_naissance : 2007-04-01
}
```

Document (V2)

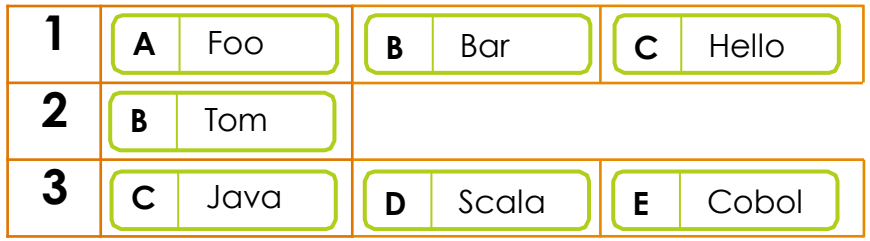
```
{
        type :      « Chien »,
        nom :       « Stella »,
        humeur :    « Heureuse »,
        date_naissance : 2007-04-01
        aboiement : [
                { texte : « j'ai mangé de la pâtée »
                  commentaires : [
                        { id_chien : « chien_4 »,
                          texte : « on s'en fout! »
                        }
                ]
        }
]
}
```

Types de Bases de Données NOSQL

.3 Orientées Colonnes (*Column Store*(

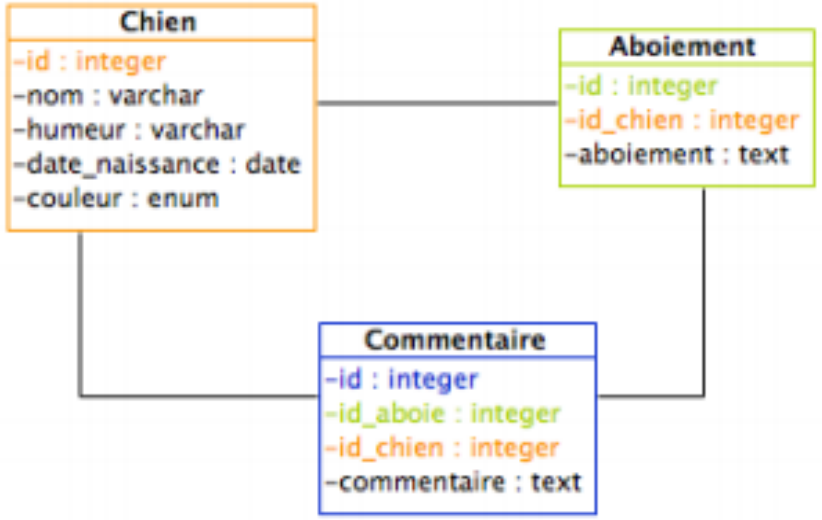
- Évolution de la BD clef/valeur
- Ressemble aux SGBDR, mais avec un nombre de colonnes dynamique, différent d'un enregistrement à un autre (pas de colonnes portant les valeurs NULL(
- Offrent de très hautes performances et une architecture hautement évolutive
- *Exemples*: Hbase (Hadoop), Cassandra (Facebook, Twitter,(BigTable (Google(

	A	B	C	D	E
1	Foo	Bar	Hello		
2		Tom			
3			Java	Scala	Cobol



Types de Bases de Données NOSQL

.3Orientées Colonnes (*Column Store(*



Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL

Requête: clef/famille:titre[/time]

Exp: "chien_12"/"Chien":"Nom" ➔ Stella

Colonnes

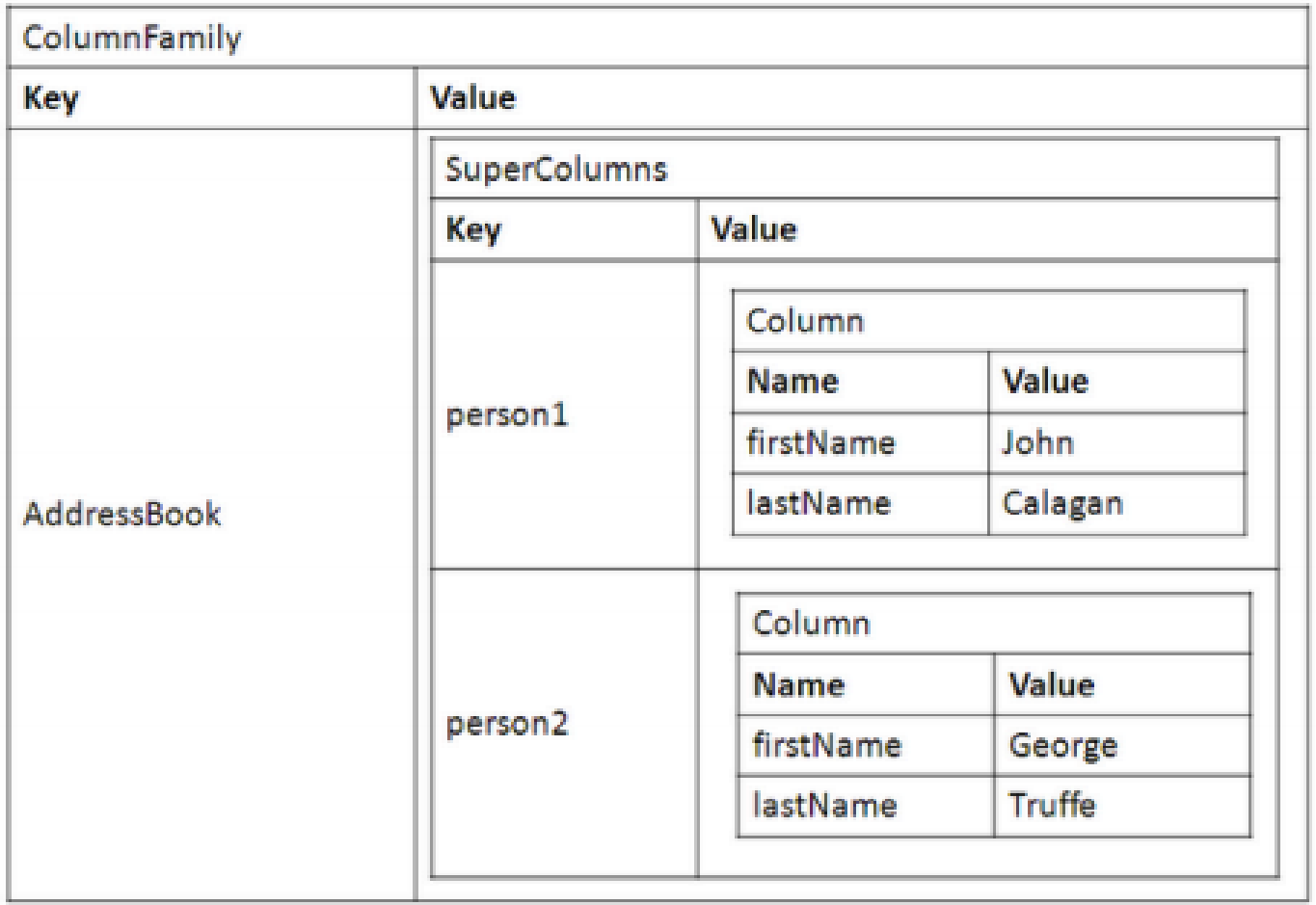
Clef

Chien_12

Famille	Titre	Temps	Valeur
Chien	Date_naissance	15	2007-04-01
Chien	Humeur	11	En Colère
Chien	Humeur	45	Heureuse
Chien	Nom	25	Stella
Chien	Couleur	34	Noire
Aboiement	Text	11	J'ai mangé de la pâtée

Types de Bases de Données NOSQL

.3Orientées Colonnes (*Column Store*(



ColumnFamily	
Key	**Value**
AddressBook	SuperColumns

SuperColumns	
Key	**Value**
person1	Column

Column	
Name	**Value**
firstName	John
lastName	Calagan

SuperColumns	
Key	**Value**
person2	Column

Column	
Name	**Value**
firstName	George
lastName	Truffe

Exemple Cassandra : Colonne / Super Colonne / Famille de Colonnes

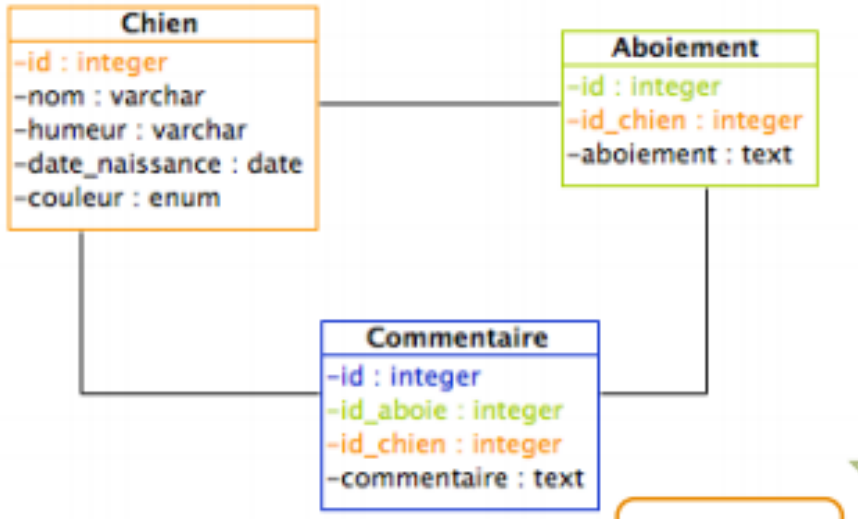
Types de Bases de Données NOSQL

4. Orienté Graphes (*Graph Database*)

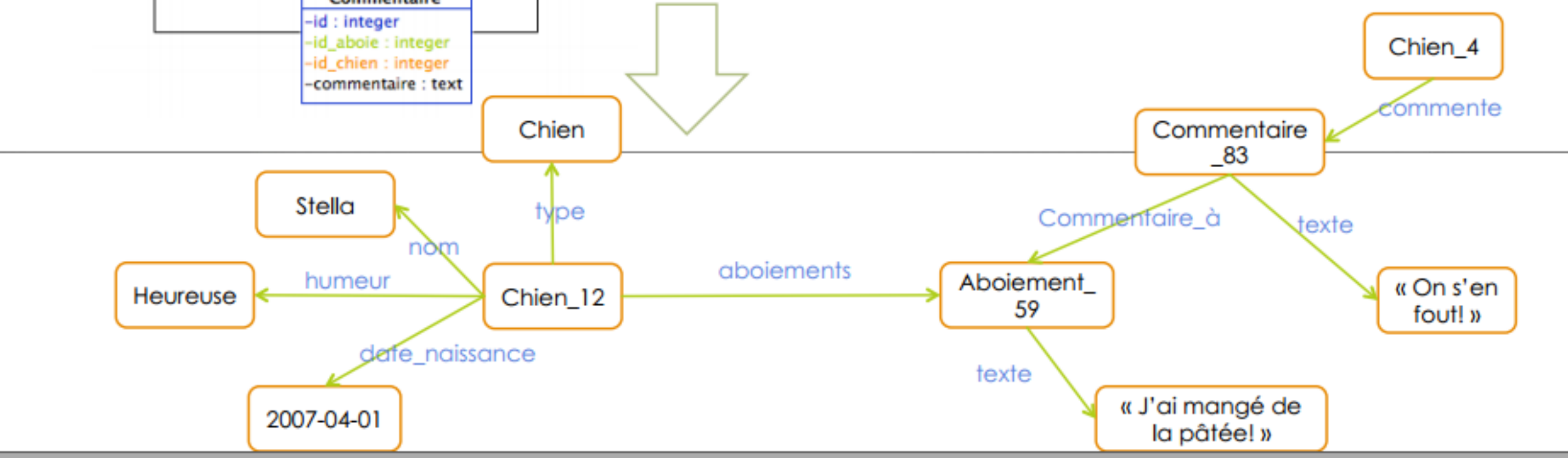
- Basées sur les théories des graphes
- S'appuie sur les notions de nœuds, de relations et des propriétés qui leur sont rattachées
- Conçues pour les données dont les relations sont représentées comme graphes, et ayant des éléments interconnectés, avec un nombre indéterminé de relations entre elles
- Adapté aux traitements des données des réseaux sociaux
- *Exemples* : Neo4J et InfiniteGraph, OrientDB

Types de Bases de Données NOSQL

.4Orienté Graphes (*Graph Database(*



Id	Nom	Humeur	Date_naissance	Couleur
12	Stella	Heureuse	2007-04-01	NULL
13	Wimma	Faim	NULL	Noire
9	Ninja	NULL	NULL	NULL



Comparaison entre Types de BD NOSQL

Modèle	Performance	Évolutivité	Flexibilité	Complexité	Fonctionnalités
Clef/Valeur					Variable
Orienté Colonnes					Minimale
Orienté Document					Variable
Orienté Graphes					Théorie des Graphes

NOSQL vs. BDR

BIG DATA

CHP3 – BASES DE DONNÉES NOSQL

NOSQL et BDR (1/3)

- Choix de NOSQL en opposition aux bases de données relationnelles conduits par les contraintes du marché et les besoins techniques
- **BigData**
  - Adaptation des BD NOSQL au BigData
    - Vitesse (*Velocity*) : beaucoup de données qui arrivent rapidement, à partir de plusieurs sources
    - Variété (*Variety*) : données structurées, semi-structurées ou non-structurées
    - Volume (*Volume*) : données très nombreuses (To et Po)
    - Complexité (*Complexity*) : données stockées et gérées à plusieurs endroits et data centers.
- **Disponibilité continue des données**
  - Le manque de disponibilité peut être fatal pour une entreprise
  - BD NOSQL utilisent une architecture hautement distribuée : pas de SPOF
  - Redondance des données et traitements : tolérance aux fautes
  - D'où : disponibilité continue à travers les data centers, et le cloud
  - Toute mise à jour ou modification est faite sans déconnecter la base

NOSQLet BDR (2/3)

- **Indépendance de l'emplacement**
  - Possibilité de consulter et modifier une BD sans savoir où est-ce que ces opérations ont réellement lieu
  - Toute fonctionnalité d'écriture propagée à partir d'un emplacement, pour être disponible aux utilisateurs à partir d'autres sites
  - Difficile à appliquer aux BDR, surtout pour l'écriture
- **Modèles de données flexibles**
  - Une des raisons majeures
  - Dans le modèle relationnel, les relations entre les tables sont prédéfinies, fixes et organisées dans un schéma strict et uniforme
    - ✓✓ Problèmes d'évolutivité et de performance en gérant de grands volumes de données
  - Les BD NOSQL peuvent accepter tout type de données (structurées, semi-structurées ou non-structurées) plus facilement
  - Dans les BDR, les performances posent problème, surtout quand des lignes « larges » sont utilisées et les actions de modification sont nombreuses

NOSQL et BDR (3/3)

- **Business Intelligence et Analyse**
  - Capacité des bases NOSQL à exploiter les données collectées pour dériver des idées
  - Extraction d'informations décisionnelles à partir d'un grand volume de données, difficile à obtenir avec des bases relationnelles
  - Bases NoSQL permettent le stockage et la gestion des données des applications métier, et fournissent une possibilité de comprendre les données complexes, et de prendre des décisions

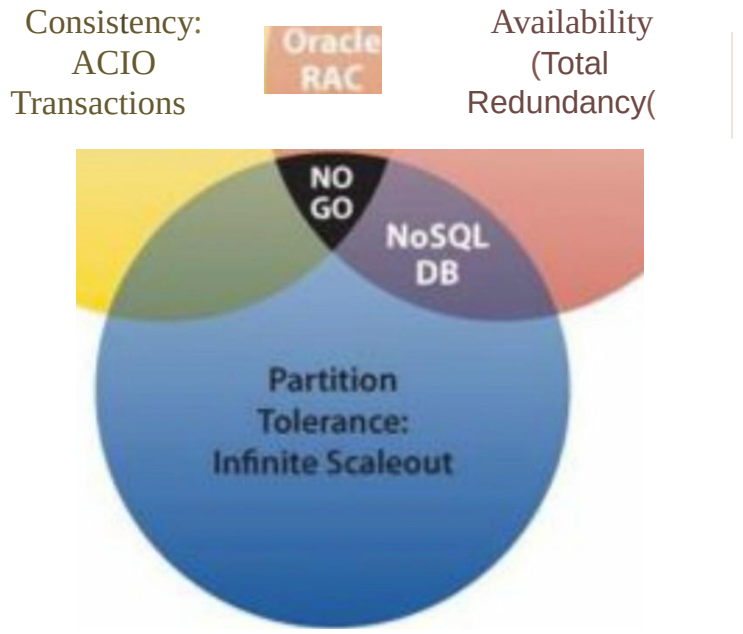
- **Capacités transactionnelles modernes**
  - Nouvelles définition du principe de transaction
  - Utilisation des propriétés BASE au lieu des propriétés ACID

NOSQL et BDR : Propriétés ACID

- **Propriétés ACID** : Atomicité, Cohérence, Isolation et Durabilité
  - Propriétés des transactions des BDR
  - *Atomicité* : Soit toutes les instructions sont exécutées, soit aucune
  - *Consistance* : toute transaction amène la BD d'un état valide à un autre ➔➔ renforcée par les contraintes d'intégrité et les clefs étrangères
  - *Isolation* : Même si plusieurs transactions peuvent être exécutées par un ou plusieurs utilisateurs simultanément, une transaction ne devrait pas voir les effets des autres transactions concurrentes
  - *Durabilité* : Une fois la transaction enregistrée dans la base (*commit*), ces changement sont persistants.
- Toutes les BDR supportent les transactions ACID
- Mais :
  - Données de plus en plus volumineuses + Besoin de systèmes évolutifs
  - Besoin de BD distribuées sur le réseau, pour une évolutivité horizontale

NOSQL et BDR : Théorème CAP

- Destiné à évaluer les systèmes de stockage distribués
- Théorème CAP : « Il est impossible de satisfaire les trois propriétés CAP en même temps«
- **Propriétés CAP** : *Consistency, Availability, Partition tolerance*
  - *Consistance*: Si j'écris une donnée dans un nœud et que je la lis à partir d'un autre nœud dans un système distribué, je retrouve ce que j'ai écris sur le premier.
  - *Haute disponibilité* : à tout moment, pour chaque requête, la réponse est garantie. Même en cas de panne, les données restent accessibles
  - *Tolérance au partitionnement* : les données peuvent être partitionnées sur différents supports sans souci de localisation. Les activités continuent sans interruption lors de la modification du système (en cas d'ajout ou suppression de nœuds) et en cas de chute du réseau



NOSQL et BDR : Théorème CAP

- **Pourquoi est-il impossible de satisfaire les 3 propriétés CAP en même temps?**
- Soit un système distribué. On est entrain de modifier une donnée sur le nœud N1 et d'essayer de la lire à partir du nœud N2
  - N2 peut retourner la dernière bonne valeur dont il dispose, ce qui viole la Consistance
  - N2 attend que la nouvelle valeur lui parvienne. Comme c'est un système distribué, les chances d'un échec de transmission sont assez importantes, ce qui provoquera une attente infinie de N2. D'où une violation de la Disponibilité
  - Si on veut satisfaire à la fois la consistance et la disponibilité, le système de stockage ne doit pas être partitionné. D'où violation de la Tolérance au partitionnement.
- Pour les BD NOSQL, il n'y a plus de jointures
  - ➔➔ La propriété de consistance n'est plus assurée de la même manière
- Consistance dans NOSQL: consistance immédiate et éventuelle des données à travers les nœuds de la base distribuée

NOSQL et BDR : Propriétés BASE

- **BASE** : Basically Available, Soft-state, Eventual consistency
- Basically Available
  - Le système garantit la disponibilité, comme définie dans le théorème CAP
- Soft-State
  - L'état du système peut changer dans le temps, même sans nouvelles entrées, et ce à cause du principe de consistance éventuelle
- Eventual Consistency
  - Les modification arriveront éventuellement à tous les serveurs, si on leur donne suffisamment de temps
- BASE est plus flexible que ACID, accepte certaines erreurs, dont l'occurrence est assez rare

NOSQL et BDR : Recapitulatif

BRD		NOSQL
Consistance forte	vs	Consistance éventuelle
Grandes quantités de données	vs	Enorme quantité de données
Évolutivité possible	vs	Evolutivité facile
SQL	vs	Map-Reduce
Bonne disponibilité	vs	Très haute disponibilité

Synthèse

- Bases de données NOSQL
  - Performances sur de gros volumes de données
  - Performances sur des données non structurées
  - Evolutivité très importante, même pour de faibles volumes
- Cependant:
  - Technologie assez jeune ➔➔ Manque d'outils la supportant
  - Encore en évolution, pas de standards
  - Pas de langage de requêtage commun comme SQL, mais divers:
    - ✓✓ Requêtes spécifiques au langage (Java, Python…)
    - ✓✓ Requêtes spéciale pour la base (Cassandra Query Language)
    - ✓✓ API basée sur Map Reduce, ou sur graphes d'objets
  - On doit faire plus de travail au niveau du code, ce qui peut influer sur la performance

Quand utiliser NOSQL?

- Si l'**évolutivité** est une préoccupation
  - Mais attention, ce n'est pas toujours un MUST : Flickr et Wikipedia utilisent des RDBMS
- Si l'**absence de schéma** est une préoccupation
- Si être **IN et FASHION** est une préoccupation
  - Pour un nombre important de personnes, l'utilisation d'un nouveau paradigme ou technologie est un MUST!

Cassandra

BIG DATA

BASES DE DONNÉES NOSQL

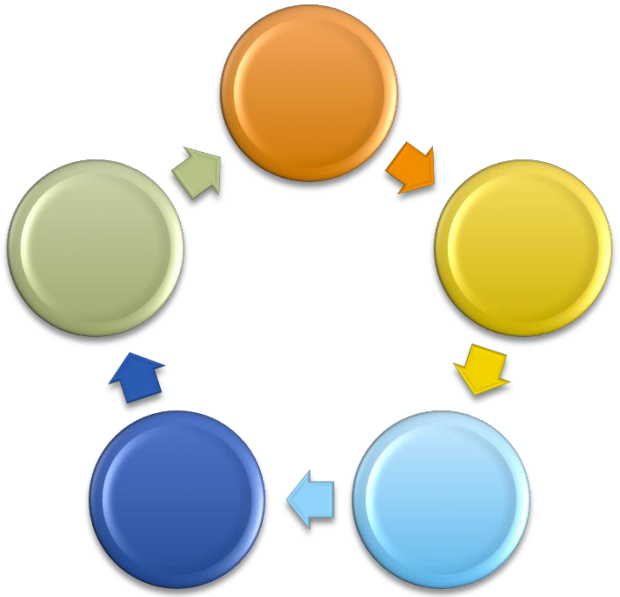
Cassandra

- Base de données :
  - Distribuée
  - À haute performance
  - Extrêmement évolutive
  - Tolérante aux fautes (pas de SPOF)
  - Non-relationnelle
- Peut être utilisée à la fois comme :
  - Base de données temps réel
  - Base de données pour une lecture intensive pour les systèmes décisionnels
- À la base, c'est une combinaison de BigTable de Google et DynamoDB de Amazon, incubée dans Facebook, et hébergée comme solution open source dans Apache.

Architecture de Cassandra (1/2)

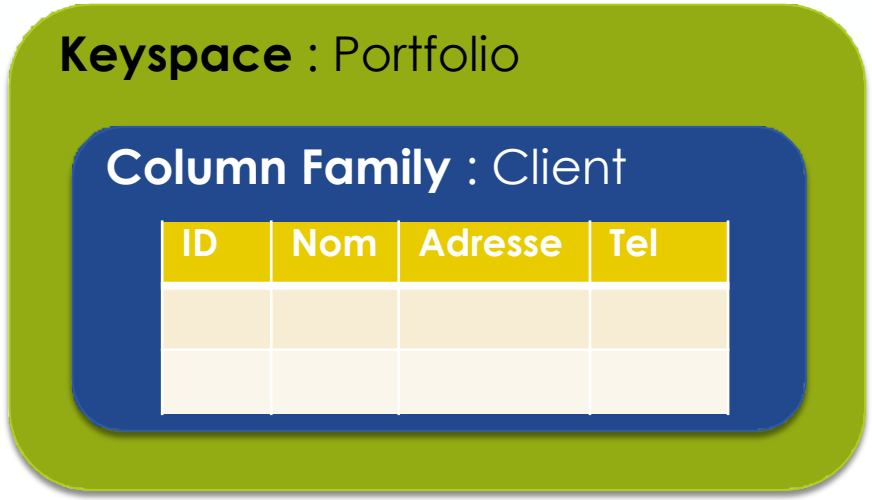
- Système distribué, P2P

  Composés de plusieurs nœuds identiques
  - Pas de notion de NameNode ou nœud maître…
- Les données sont partitionnées par défaut à travers les différents nœuds du cluster
- Les données sont répliquées pour assurer une tolérance aux fautes maximale
  - L'utilisateur contrôle le nombre de répliques qu'il désire avoir pour ses données
- Lecture et écriture à partir de n'importe quel nœud, indépendamment de l'emplacement des données
- Utilisation du protocole **Gossip** pour la communication entre les différents nœuds du cluster
  - Échange de données entre les nœuds chaque seconde



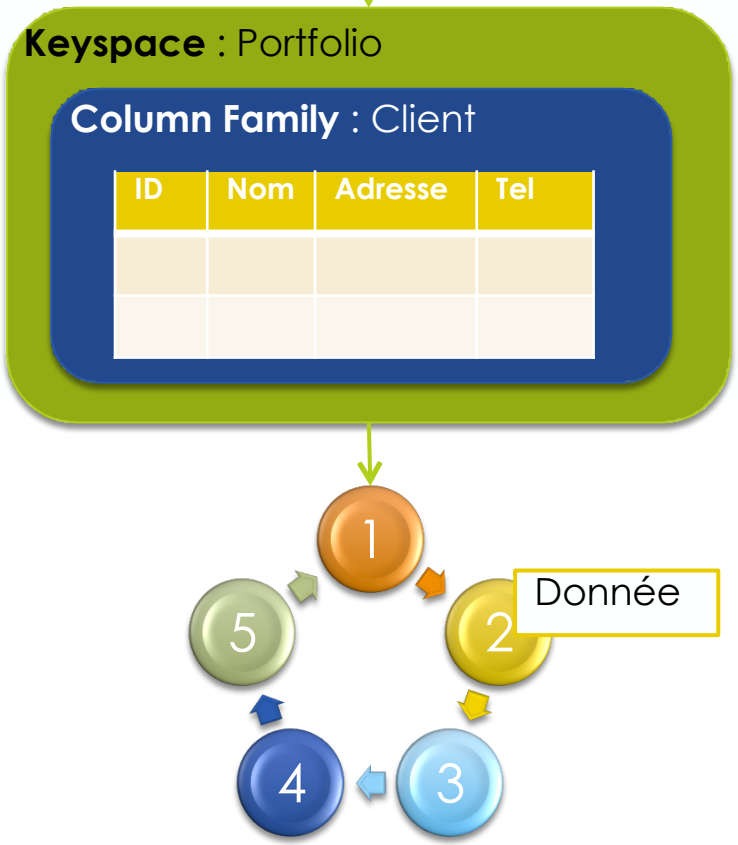
Architecture de Cassandra (2/2)

- Structure orientée colonnes, à l'image de BigTable
- Vocabulaire:
  - **Keyspace** (équivalent de *database*)
  - **Column Family** (équivalent de *table*)
    - ✓✓ Dans la nouvelle version du langage CQL, appelée Table
    - ✓✓ Schéma plus flexible et dynamique qu'une table
  - **Colonne** (équivalent à *enregistrement*)
    - ✓✓ Indexée par une clef
    - ✓✓ D'autres champs peuvent être également indexés, mais à la demande



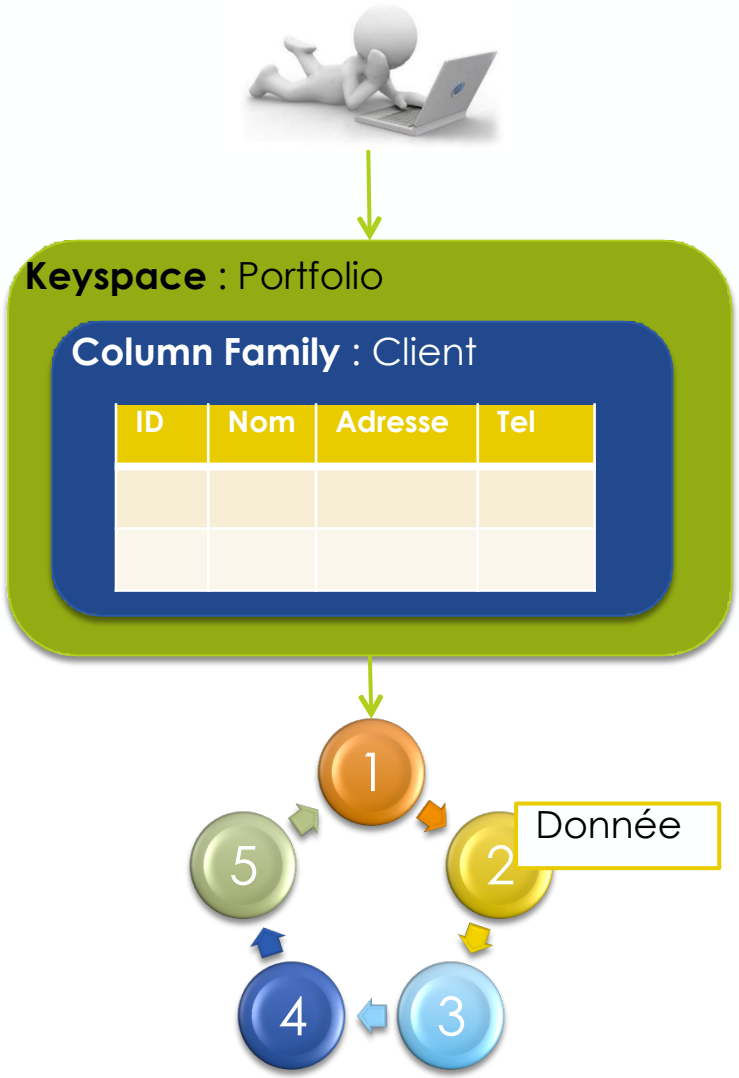
Partitionnement dans Cassandra (1/2)

- Partitionnement facile des données à travers les différents nœuds participants du cluster
- Chaque nœud est responsable d'une partie de la base de données
- Les données sont insérées par l'utilisateur dans une famille de colonnes
- Elles sont ensuite placées sur un nœud, selon sa clef de colonne



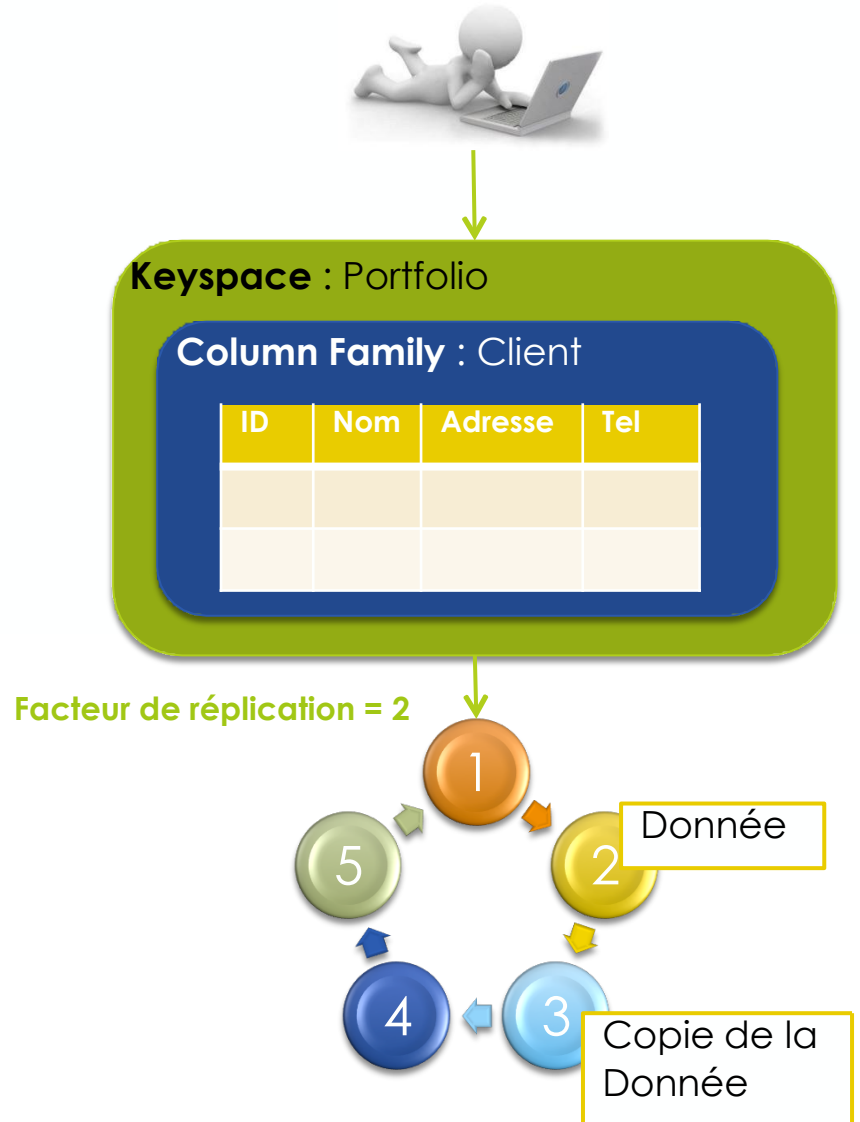
Partitionnement dans Cassandra (2/2)

- Stratégies de partitionnement
  - **Partitionnement aléatoire**
    - ✓✓ Par défaut, recommandé
    - ✓✓ Données partitionnée le plus équitablement possible à travers les différents nœuds
    - ✓✓ Utilisation de MD5 (ou Murmur3) pour le hachage de chaque clef de famille de colonnes
  - **Partitionnement ordonné**
    - ✓✓ Sauvegarde les clefs de familles de colonnes par ordre à travers les nœuds d'un cluster
    - ✓✓ Peut provoquer des problèmes, surtout pour la répartition des charges (des nœuds avec des données plus volumineuses que d'autres)
- Stratégie spécifiée dans un fichier de configuration *cassandra.yaml*
- Si la stratégie d'une base est modifiée, il faut recharger toutes les données



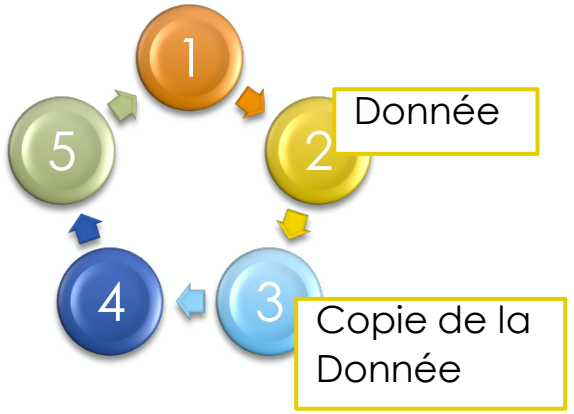
Réplication dans Cassandra (1/3)

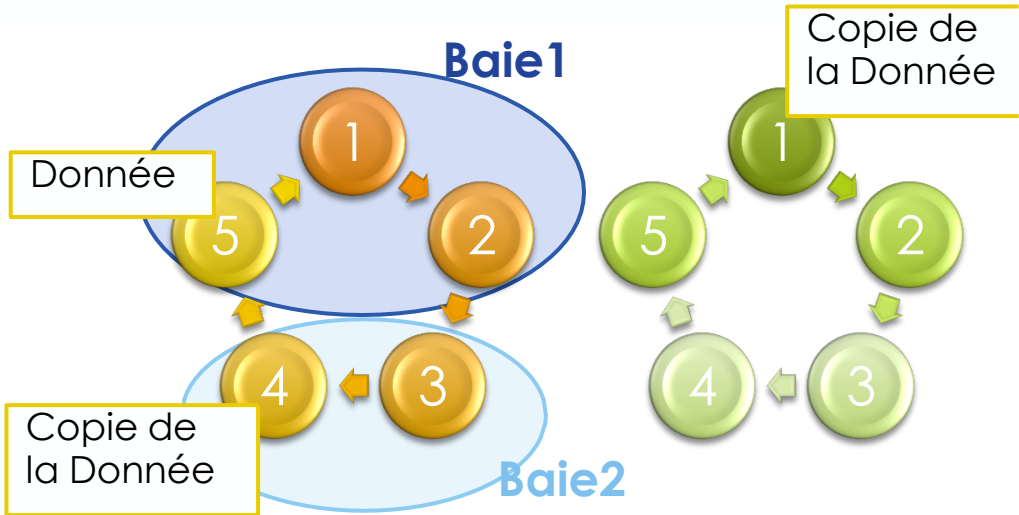
- Pour assurer la tolérance aux fautes et pas de SPOF, il est possible de créer une ou plusieurs copie(s) de chaque colonne à travers les nœuds participants
- L'utilisateur spécifie le **facteur de réplication** désiré à la création du keyspace
- Les données sont insérées par l'utilisateur dans une famille de colonnes
- La colonne est répliquée dans les nœuds du cluster selon le facteur de réplication



Réplication dans Cassandra (2/3)

- Stratégies de réplication
  - **Stratégie Simple** :
    - ✓✓ La colonne originelle est placée sur un nœud déterminé par le partitionneur
    - ✓✓ La réplique est placée sur le nœud suivant de l'anneau (clockwise)
    - ✓✓ Pas de considération pour l'emplacement dans un data- center ou dans une baie (approprié pour les déploiement sur un seul datacenter)
  - **Stratégie par topologie du réseau**
    - ✓✓ Plus sophistiquée
    - ✓✓ Plus de contrôle sur l'emplacement des répliques de colonnes
    - ✓✓ Parcourt le cluster (clockwise) à la recherche d'un nœud dans une baie différente. Si introuvable, stocker la colonne dans un nœud de la même baie
    - ✓✓ Un *groupe de réplication* est spécifié, pour distinguer entre plusieurs datacenters





Réplication dans Cassandra (3/3)

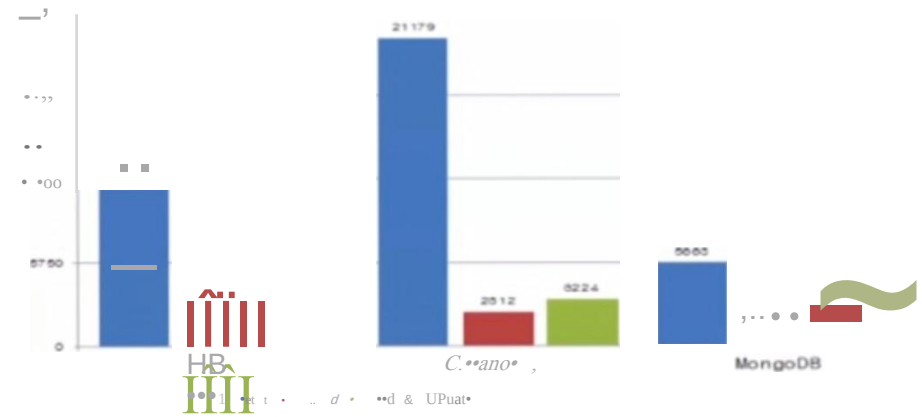
- Mécanisme de réplication
  - Utilisation de **SNITCH**:
    - ✓✓ Définition de la manière dont les nœuds sont groupés dans un réseau
    - ✓✓ Répartition des nœuds entre baies et datacenters
  - Plusieurs types de SNITCH
    - ✓✓ **Simple SNITCH** : utilise la stratégie simple
    - ✓✓ **Rack-Inferring SNITCH** : détermine la topologie du réseau en analysant les adresses IP:
      - Second octet de l'adresse IP: Data Center
      - Troisième octet de l'adresse IP: Baie
    - ✓✓ **Property File SNITCH** : se base sur une description de l'utilisateur pour déterminer l'emplacement des nœuds (*cassandra-topology.properties*)
    - ✓✓ **EC2 SNITCH** : pour déploiement dans Amazon EC2. Utilise l'API AWS pour déterminer la topologie
  - Défini dans le fichier de configuration *cassandra.yaml*

Consistance dans Cassandra

- Architecture *Read and Write Anywhere*
- L'utilisateur peut se connecter à n'importe quel nœud, dans n'importe quel datacenter, et lire/écrire les données qu'il veut
- Les données sont automatiquement partitionnées et répliquées à travers le cluster

Consistance dans Cassandra

- Écriture:
  - Données écrites d'abord dans un *commit log* pour la durabilité
  - Ensuite, écriture en mémoire dans une *MemTable*
  - Une fois la MemTable pleine, les données sont sauvegardées dans le disque, dans une *SSTable*)***S****orted* ***S****trings* ***T****able*(
  - Même si les transactions relationnelles (commits et rollbacks) ne sont pas supportées, les écritures sont atomiques au niveau des colonnes
    - Soit toutes les colonnes sont modifiées, soit aucune ne l'est
- Cassandra est la base de données NOSQL la plus rapide en écriture



Consistance dans Cassandra

- Consistance : à quel point est-ce qu'une donnée est à jour et synchronisée sur toutes ses répliques.
- Extension du concept de consistance éventuelle à une **consistance ajustable**
- Choix possible entre une consistance forte ou éventuelle selon les besoins
- Ce choix est fait par opération, ce n'est pas une stratégie globale pour la base de données (ex, pour changer la stratégie de lecture en *quorum*)
  - SELECT * FROM users USING CONSISTENCY QUORUM WHERE state='TX';
- Consistance gérée à travers plusieurs data centers.

Stratégies d'Écriture

- **Niveau de consistance :** combien de répliques doivent être écrites avec succès avant de retourner acquittement au client
  - **Any** : une écriture doit réussir sur n'importe quel nœud, au moins un.
    - ✓✓ Offre la plus haute disponibilité, mais la plus basse consistance
  - **One (défaut dans CQL)** : une écriture doit réussir sur le commit log et la memtable d'au moins une réplique
  - **Quorum** : une écriture doit réussir sur un certain pourcentage de répliques (pourcentage = (facteur de réplication/2)+1)
    - ✓✓ Meilleure alternative en terme de consistance et de disponibilité
  - **Local-Quorum** : une écriture doit réussir sur un certain pourcentage de nœuds répliques sur le même datacenter que le nœud coordinateur
  - **Each-Quorum** : une écriture doit réussir sur un certain pourcentage de nœuds répliques sur tous les datacenters.
  - **All** : une écriture doit réussir sur tous les nœuds répliques d'une colonne
    - ✓✓ Offre la plus haute consistance, mais la plus basse disponibilité

Stratégies d'Écriture

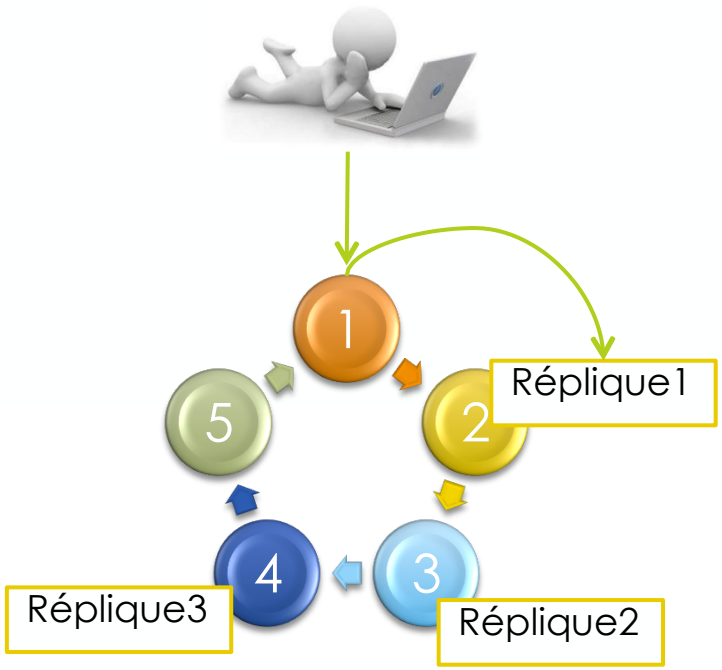
- Cassandra utilise les **Hinted Handoffs**
- Elle tente de modifier une colonne sur toutes les répliques
- Si certains des nœuds répliques ne sont pas disponibles, un indice (*hint*) est sauvegardé sur l'un des nœuds répliques en marche, pour mettre à jour tous les nœuds répliques en panne une fois disponibles
- Si aucun nœud réplique n'est disponible, l'utilisation de la stratégie **ANY** permettra au nœud coordinateur de stocker cet indice. Mais la donnée ne sera lisible que quand l'un des nœuds est disponible de nouveau.

Stratégies de Lecture

- **Niveau de consistance :** combien de répliques doivent répondre avant de retourner le résultat à l'application cliente
- Cassandra vérifie le nombre de répliques pour la donnée la plus récente pour satisfaire une demande de lecture (basée sur le temps)
  - **One (défaut dans CQL)** : obtention d'une réponse à partir de la réplique la plus proche selon le SNITCH
    - ✓✓ Offre la plus faible consistance, mais la plus haute disponibilité
  - **Quorum** : obtention du résultat le plus récent à partir d'un certain pourcentage de nœuds répliques (pourcentage = (facteur de réplication/2)+1)
    - ✓✓ Meilleure alternative en terme de consistance et de disponibilité
  - **Local-Quorum** : obtention du résultat le plus récent à partir d'un certain pourcentage de nœuds répliques sur le même datacenter que le nœud coordinateur
    - ✓✓ Évite la latence due à la communication inter-datacenters
  - **Each-Quorum** : : obtention du résultat le plus récent à partir d'un certain pourcentage de nœuds répliques sur tous les datacenters.
  - **All** : obtention du résultat le plus récent à partir de tous les nœuds répliques
    - ✓✓ Offre la plus haute consistance, mais la plus basse disponibilité

Stratégies de Lecture

- ***Read Repair***
  - Cassandra assure que les données fréquemment lues soient consistantes
  - À la lecture d'une donnée, le nœud coordinateur compare toutes ses répliques en arrière plan.
  - Si ces données ne sont pas consistantes, envoie une demande d'écriture aux nœuds réplicas pour mettre à jour leur donnée et afficher la donnée la plus récente
  - *Read Repair* peut être configuré par famille de colonnes et est activé par défaut.



Gérer les Données et Objets dans Cassandra

- Deux interfaces pour gérer les objets/données
  - Cassandra CLI (Command Line Interface)
  - CQL (Cassandra Query Language)

- CLI
  - Interface originelle conçue pour créer des objets, entrées et manipuler les données

- CQL
  - Utilisée pour créer/manipuler des données en utilisant un langage proche de SQL

Gérer les Données et Objets dans Cassandra

- CQL
  - Objets tels que les keyspaces, familles de colonnes et index sont créés, modifiés et supprimés avec les requêtes usuelles: CREATE, ALTER et DROP
  - Données insérées, modifiées et supprimées avec INSERT, UPDATE et DELETE
  - Données lues avec SELECT
  - **Mais**
    - Ne supporte pas des opérations telles que GROUP BY, ORDER BY (sauf pour les clefs composées, et ordonnées seulement selon la deuxième clef primaire)…
  - Utiliser la clause USING CONSISTENCY pour déterminer le type de consistance forte pour chaque opération de lecture et l'écriture (Any, One, Quorum…)

Récapitulatif : Cassandra (1/2)

- Grande Scalabilité (Gigaoctet/Petaoctet) ➔➔ Appropriée aux Big Data
  - Gain de performances linéaire grâce à l'ajout des nœuds
- Pas de SPOF (Single Point of Failure)
- Réplication et distribution des données facile à travers les data-centers
- Pas besoin de couche de cache séparée
  - Utilisation des mémoires vives de l'ensemble des nœuds
- Consistance des données ajustable
  - Possibilité de choisir pour chaque opération si une donnée est fortement consistante (tous les nœuds sont similaires à tout moment) ou éventuellement consistante
- Schéma de données flexible

Récapitulatif : Cassandra (2/2)

- Compression des données
  - Utilisation de l'algorithme de compression Snappy de Google
  - Compression des données pour chaque famille de colonnes
  - Pas de pénalité pour la performance, et même une amélioration notable, grâce à la diminution du nombre de I/O physique
- Langage CQL très proche de SQL
- Support des langages et plateformes clef
- Pas besoin de matériel ou de logiciel particulier

Sources

- **Sites** (consultés en février 2014)
  - Planet Cassandra : www.planetcassandra.org
  - NOSQL : 5 minutes pour comprendre : http://blog.neoxia.com/nosql-5-minutes-pour-comprendre/ *NEOXIA*
  - NOSQL Europe : Bases de données orientées colonnes et Cassandra http://blog.xebia.fr/2010/05/04/nosql-europe-bases-de-donnees-orientees-colonnes-et-cassandra/ *XEBIA*
  - Une base NOSQL, Cassandra : http://www-igm.univ-mlv.fr/~dr/XPOSE2010/Cassandra *IGM*
  - Why NOSQL – Part 1 – CAP Theorem : http://bigdatanerd.wordpress.com/2011/12/08/why-nosql-part-1-cap-theorem/ *DATANERD*
  - DataStax Cassandra Tutorials : http://www.datastax.com/resources/tutorials/cassandra-overview *DataStax*
- **Présentations**
  - Harri Kauhanen, « NOSQL Databases », *Futurice, Septembre 2010*
- **Rapports**
  - Mouna Laroussi, « Etude et mise en place de la technologie Big Data pour la télécommunication », *Projet de fin d'études, INSAT, 2013*
- **Livres Blancs**
  - Top 5 Considerations when evaluating NOSQL Databases, *MongoDB White Paper, Juin 2013*