Université de Ngaoundéré
Institut Universitaire de Technologie

Base de Données non structurées (NoSQL)

Chapitre 1: Pourquoi NoSQL?

Pr. Dayang Paul
Maître de conférence

Année académique
2024/2025

INTRODUCTION

Depuis presque toujours, les bases de données relationnelles constituent le choix par défaut pour le stockage des données, en particulier dans le monde des applications d'entreprise. Si vous êtes développeurs et que vous démarrez un nouveau projet, votre seul choix sera probablement la base relationnelle à utiliser.

Après une telle période de dominance, l'engouement actuel pour les bases de données NoSQL est une surprise. Dans ce chapitre, nous examinerons pourquoi les bases de données relationnelles sont devenues si dominantes, et pourquoi nous pensons que l'essor actuel des bases de données NoSQL n'est pas un feu de paille

I- La valeur des bases de données relationnelles

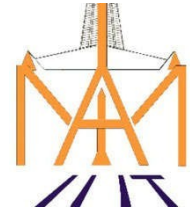
Les bases de données relationnelles font désormais partie intégrante de notre culture informatique, au point qu'il est facile de les considérer comme acquises. Il est donc utile de revenir sur les avantages qu'elles procurent.

1- ACCES AUX DONNEES PERSISTANTES

La valeur la plus évidente d'une base de données est probablement la conservation de grandes quantités de données persistantes. pour conserver les données, nous les écrivons dans un magasin de sauvegarde, généralement vu comme un disque.

La base de données offre plus de flexibilité qu'un système de fichiers en stockant de grandes quantités de données d'une manière qui permet à un programme d'application d'accéder rapidement et facilement à de petites parties de ces informations.

I- La valeur des bases de données relationnelles



2- CONCURRENCE

Dans les applications d'entreprise, de nombreuses personnes regardent le même ensemble de données en même temps, et peuvent éventuellement modifier ces données. La plupart du temps, ces personnes travaillent sur des zones différentes de ces données, mais il arrive qu'elles opèrent sur la même partie des données. Par conséquent, nous devons nous préoccuper de la coordination de ces interactions pour éviter des choses comme la double réservation.

La simultanéité est notoirement difficile à mettre en œuvre, avec toutes sortes d'erreurs qui peuvent piéger même les programmeurs les plus prudents. Comme les applications d'entreprise peuvent avoir beaucoup d'utilisateurs et d'autres systèmes travaillant tous simultanément, il y a beaucoup de place pour que de mauvaises choses se produisent. Les bases de données relationnelles permettent d'y remédier en contrôlant tous les accès à leurs données par le biais de transactions.

I- La valeur des bases de données relationnelles

3- INTEGRATION

Les applications d'entreprise vivent dans un riche écosystème qui exige que de multiples applications, écrites par différentes équipes, collaborent afin d'accomplir des tâches. Ce type de collaboration inter-applications est délicat car il implique de repousser les frontières organisationnelles humaines. Les applications doivent souvent utiliser les mêmes données et les mises à jour effectuées par une application doivent être visibles pour les autres.

Un moyen courant d'y parvenir est l'intégration de bases de données partagées, où plusieurs applications stockent leurs données dans une seule base de données.

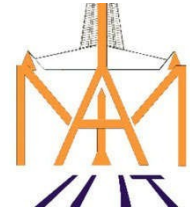
I- La valeur des bases de données relationnelles

4- UN MODELE STANDARD

Les bases de données relationnelles ont réussi parce qu'elles offrent les principaux avantages que nous avons décrits précédemment d'une manière (généralement) standard. Par conséquent, les développeurs et les professionnels des bases de données peuvent apprendre le modèle relationnel de base et l'appliquer à de nombreux projets

Bien qu'il existe des différences entre les différentes bases de données relationnelles, les mécanismes de base restent les mêmes : les dialectes SQL des différents fournisseurs sont similaires, les transactions fonctionnent pratiquement de la même manière.

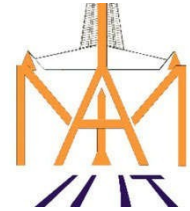
II- Décalage d’impedance



Les bases de données relationnelles offrent de nombreux avantages, mais elles sont loin d'être parfaites. Même à leurs débuts, elles ont suscité de nombreuses frustrations. Pour les développeurs d'applications, la plus grande frustration a été ce que l'on appelle communément le décalage d'impédance :la différence entre le modèle relationnel et la structure de données en mémoire.

Le modèle de données relationnel organise les données dans une structure de tables et de lignes, ou plus précisément de relations et de tuples. Dans le modèle relationnel, un **tuple** est un ensemble de paires nom-valeur et une **relation** est un ensemble de tuples. (La définition relationnelle d'un tuples est légèrement différente de celle des mathématiques et de nombreux langages de programmation avec un type de données tuples, où un tuples est u n e séquence de valeurs).

II- Décalage d’impedance



Cette base sur les relations apporte une certaine élégance et simplicité, mais elle introduit également des limitations. En particulier, les valeurs d'un tuple relationnel doivent être simples. elles ne peuvent contenir aucune structure, telle qu'un enregistrement imbriqué ou une liste. Cette limitation ne s'applique pas aux structures de données en mémoire, qui peuvent adopter des structures beaucoup plus riches que les relations.

Par conséquent, si vous souhaitez utiliser une structure de données en mémoire plus riche, vous devez la traduire en une représentation relationnelle pour la stocker sur le disque.

II- Décalage d’impedance

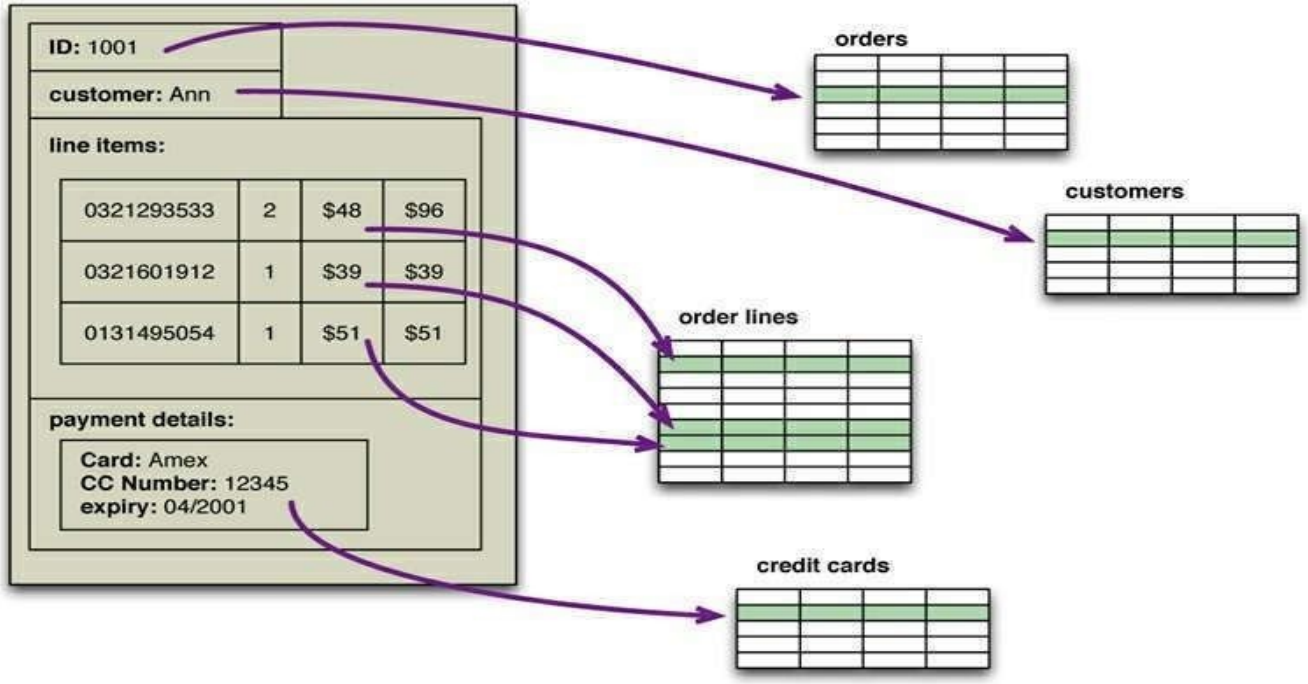
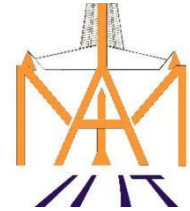


Figure 1.1. Une commande, qui ressemble à u n e structure agrégée unique dans l'interface utilisateur, est divisée en plusieurs lignes provenant de plusieurs tables dans une base de données relationnelle.

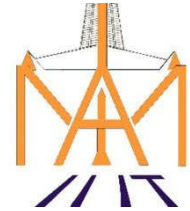
II- Décalage d’impedance



Le décalage d'impédance est une source majeure de frustration pour les développeurs d'applications et, dans les années 1990, beaucoup pensaient qu'il conduirait au remplacement des bases de données relationnelles par des bases de données qui répliquent les structures de données en mémoire sur le disque. Cette décennie a été marquée par l'essor des langages de programmation orientés objet et, avec eux, par celui des bases de données orientées objet, tous deux destinés à devenir l'environnement dominant du développement logiciel au cours du nouveau millénaire.

Cependant, alors que les langages orientés objet ont réussi à s'imposer dans le domaine de la programmation, les bases de données orientées objet sont tombées dans l'oubli. Les bases de données relationnelles ont relevé le défi en mettant l'accent sur leur rôle de mécanisme d'intégration, soutenu par un langage de manipulation des données généralement standard (SQL) et un fossé professionnel croissant entre les développeurs d'applications et les administrateurs de bases de données.

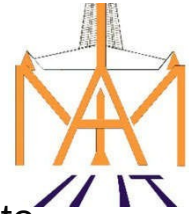
III- Bases de données d'application et d'intégration



Les raisons exactes pour lesquelles les bases de données relationnelles ont triomphé des bases de données font toujours l'objet d'un débat occasionnel au pub pour les développeurs d'un certain âge. Mais à notre avis, le facteur principal était le rôle de SQL comme mécanisme d'intégration entre les applications. Dans ce scénario, la base de données joue le rôle de une **base de données d'intégration :** plusieurs applications, généralement développées par des équipes distinctes, stockent leurs données dans une base de données commune. Cela améliore la communication, car toutes les applications fonctionnent sur un ensemble cohérent de données persistantes.

L'intégration de bases de données partagées présente des inconvénients. Une structure conçue pour intégrer de nombreuses applications finit par être plus complexe - en fait, souvent beaucoup plus complexe - que ce dont une seule application a besoin. En outre, si une application souhaite apporter des modifications à son stockage de données, elle doit se coordonner avec toutes les autres applications utilisant la base de données.

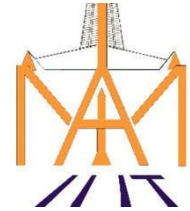
III- Bases de données d'application et d'intégration



Les applications ont des besoins différents en termes de structure et de performances, de sorte qu'un index requis par une application peut entraîner des problèmes d'insertion pour une autre. Le fait que chaque application constitue généralement une équipe distincte signifie également que la base de données ne peut généralement pas faire confiance aux applications pour mettre à jour les données d'une manière qui préserve l'intégrité de la base de données et doit donc en assumer la responsabilité au sein de la base de données elle-même.

Les préoccupations en matière d'interopérabilité peuvent désormais se déplacer vers les interfaces de l'application, ce qui permet de mettre en place de meilleurs protocoles d'interaction et de fournir un support pour les modifier. Au cours des années 2000, nous avons assisté à une nette évolution vers les services Web [Daigneau], où les applications communiquaient via HTTP. Le service Web a permis une nouvelle forme de mécanisme de communication largement utilisé, un défi à l'utilisation des bases de données partagées SQL. (La plupart de ces travaux ont été réalisés sous la bannière de l'"architecture orientée services" - un terme remarquable pour son manque de signification cohérente).

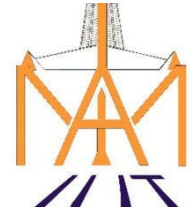
IV- L'attaque des clusters



Au début du nouveau millénaire, le monde de la technologie a été frappé par l'éclatement de la bulle Internet des années 1990. Bien que de nombreuses personnes se soient interrogées sur l'avenir économique d'Internet, les années 2000 ont vu plusieurs grandes propriétés web augmenter considérablement leur taille.

Cette augmentation d'échelle s'est produite dans de nombreuses dimensions. Les sites web ont commencé à suivre l'activité et la structure de manière très détaillée. De grands ensembles de données sont apparus : liens, réseaux sociaux, activité dans les journaux, données cartographiques. Cette croissance des données s'est accompagnée d'une croissance du nombre d'utilisateurs, les plus grands sites Web devenant de vastes domaines accueillant régulièrement un grand nombre de visiteurs.

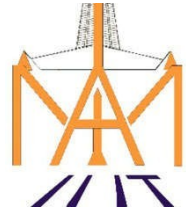
IV- L'attaque des clusters



Pour faire face à l'augmentation des données et du trafic, il fallait des ressources informatiques supplémentaires. Pour gérer ce type d'augmentation, deux choix s'offrent à vous : augmenter ou diminuer. Le passage à l'échelle supérieure implique des machines plus grandes, davantage de processeurs, de stockage sur disque et de mémoire. Mais les machines plus grandes deviennent de plus en plus chères, sans compter qu'il existe de réelles limites à mesure que votre taille augmente.

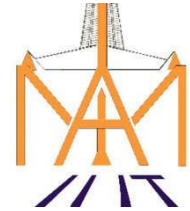
L'alternative consiste à utiliser un grand nombre de petites machines dans un cluster. Un cluster de petites machines peut utiliser du matériel de connnodité et s'avère moins cher à ce type d'échelle. Il peut également être plus résilient : si les pannes de machines individuelles peuvent survenir, le cluster global peut être construit de manière à continuer à fonctionner malgré ces pannes, offrant ainsi une grande fiabilité.

IV- L'attaque des clusters



Lorsque les grandes propriétés ont évolué vers les clusters, cela a révélé un nouveau problème : les bases de données relationnelles ne sont pas conçues pour être exécutées sur des clusters. Les bases de données relationnelles en grappe, telles que Oracle RAC ou Microsoft SQL Server, fonctionnent sur le concept d'un sous système de disque partagé. Elles utilisent un système de grappe qui écrit sur un sous-système de disque hautement disponible, mais cela signifie que la grappe a toujours le sous-système de disque comme point de défaillance unique. Les bases de données relationnelles peuvent également être exécutées sur des serveurs distincts pour différents ensembles de données, ce qui revient à partager la base de données. Bien que cela permette de séparer la charge, tout le sharding doit être contrôlé par l'application qui doit savoir à quel serveur de base de données s'adresser pour chaque élément de données.

IV- L'attaque des clusters

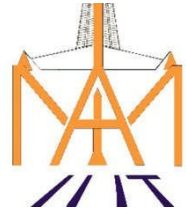


Ces problèmes techniques sont exacerbés par le coût des licences. Les bases de données relationnelles commerciales sont généralement tarifées sur la base d'un serveur unique, de sorte que l'utilisation d'une grappe augmente les prix et entraîne des négociations frustrantes avec les services d'achat.

Cette inadéquation entre les bases de données relationnelles et les clusters a conduit certaines organisations à envisager une autre voie pour le stockage des données. Deux entreprises en particulier - Google et Amazon – ont eu une grande influence. Toutes deux étaient à l'avant-garde de l'exploitation de grands clusters de ce type ; en outre, elles capturaient d'énormes quantités de données. Ces éléments leur ont donné le mobile.

V- L'émergence de NoSQL

V-1Historique



C'est avec une merveilleuse ironie que le terme "NoSQL" est apparu à la fin des années 90 comme le nom d'une base de données relationnelle open-source [Strozzi NoSQL]. Dirigée par Carlo Strozzi, cette base de données stocke ses tables sous forme de fichiers ASCII, chaque tuple étant représenté par une ligne avec des champs séparés par des tabulations. Le nom vient du fait que la base de données n'utilise pas SQL comme langage d'interrogation. Au lieu de cela, la base de données est manipulée par des scripts shell qui peuvent être combinés dans les pipelines UNIX habituels.

L'usage de "NoSQL" que nous connaissons aujourd'hui remonte à une rencontre organisée le 11 juin 2009 à San Francisco par Johan Oskarsson, un développeur de logiciels basé à Londres. L'exemple de BigTable et de Dynamo avait inspiré un grand nombre de projets expérimentant un stockage alternatif des données, et les discussions à ce sujet étaient devenues une caractéristique des meilleures conférences sur les logiciels à cette époque.

V- L'émergence de NoSQL

V-1 Historique

Le terme "NoSQL" s'est répandu comme une traînée de poudre, mais il n'a jamais fait l'objet d'une définition solide. L'appel initial [NoSQL Meetup] pour le meetup demandait des "bases de données non relationnelles, distribuées et à code source ouvert". Les exposés [NoSQL Debrief] ont porté sur Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB et MongoDB, mais le terme ne s'est jamais limité à ce septuor initial. Il n'existe pas de définition généralement acceptée, ni d'autorité pour en fournir une, aussi tout ce que nous pouvons faire est de discuter de certaines caractéristiques communes des bases de données qui ont tendance à être appelées "NoSQL« .

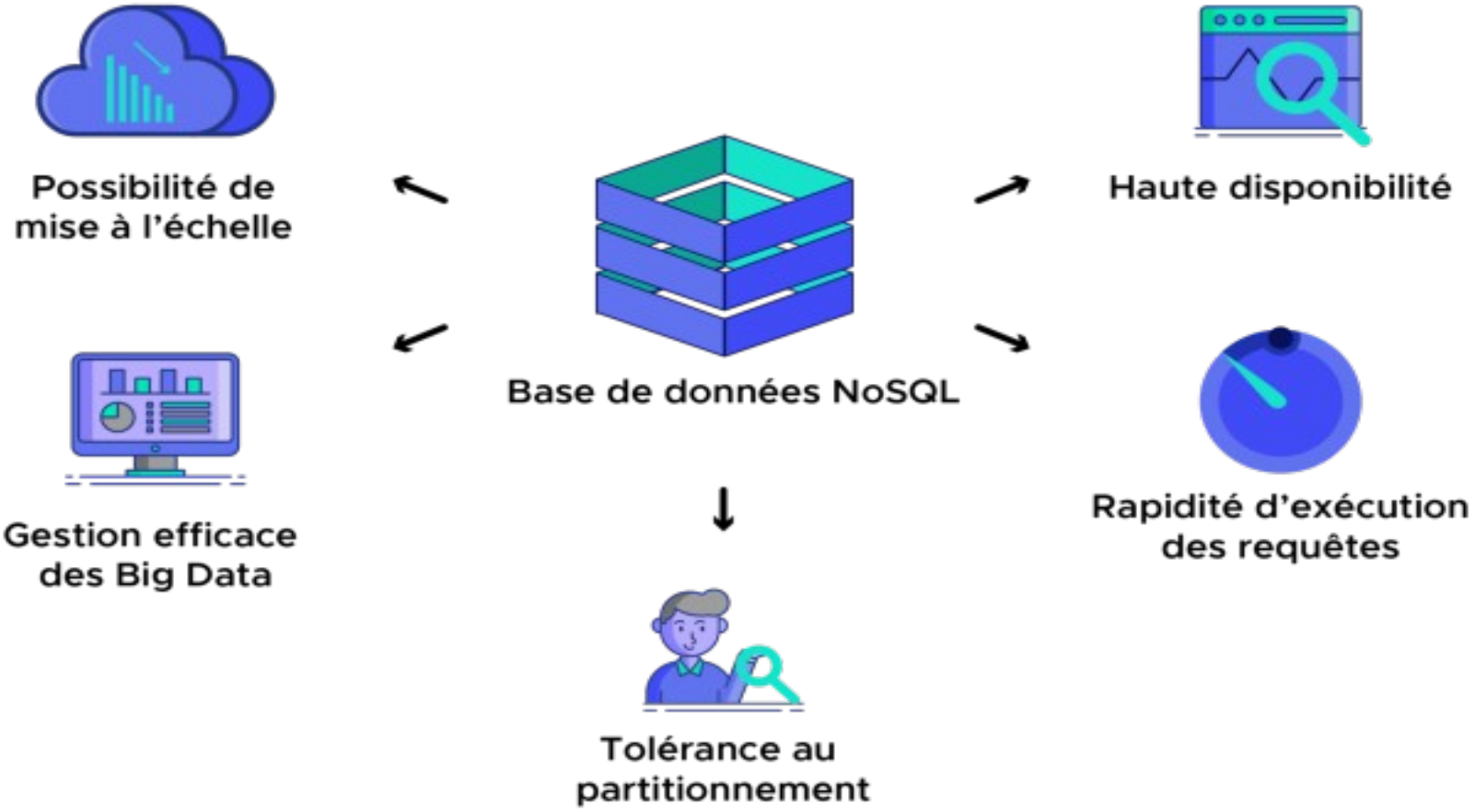
V- L'émergence de NoSQL

V-1 Historique

Pour commencer, il y a le point évident que les bases de données NoSQL n'utilisent pas SQL. Certaines d'entre elles ont des langages de requête, et il est logique qu'ils soient similaires à SQL afin d'en faciliter l'apprentissage. Le CQL de Cassandra est ainsi fait : " exactement comme SQL (sauf là où il ne l'est pas) " [CQL]. Mais jusqu'à présent, aucun n'a implémenté quoi que ce soit qui corresponde à la notion plutôt flexible de SQL standard. Il sera intéressant de voir ce qui se passera si une base de données NoSQL établie décide d'implémenter un SQL raisonnablement standard ; le seul résultat prévisible d'une telle éventualité est un grand nombre d'arguments.

Une autre caractéristique importante de ces bases de données est qu'elles sont généralement des projets à code source ouvert.

V- L'émergence de NoSQL



V- 2 Différents types de base de données NoSQL

Les bases de données NoSQL sont donc une catégorie de base de données qui n'est plus fondée sur l'architecture classique des bases relationnelles, et non pas un type à part entière. Quatre grandes catégories se distinguent parmi celles-ci.

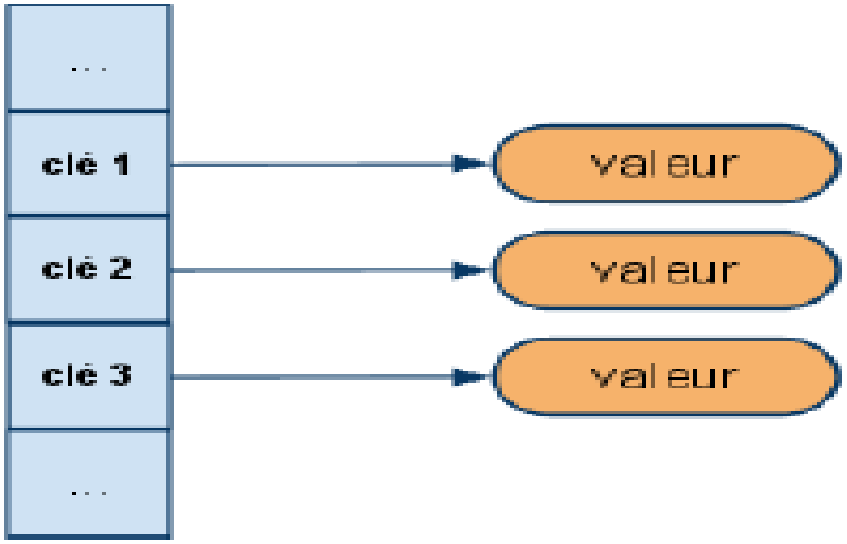
- **Paires clé-valeur**

Les moteurs NoSQL les plus simples manipulent des paires clés valeur, ou des tableaux de hachage, dont l'accès se fait exclusivement par la clé.

V- 2 Différents types de base de données NoSQL

❖ Paires clé-valeur

illustration



Avantages

- Facile à implémenter;
- Le pattern d'accès par la clé ne nécessite pas de moteur de requête complexe;
- performances élevé.

Limites

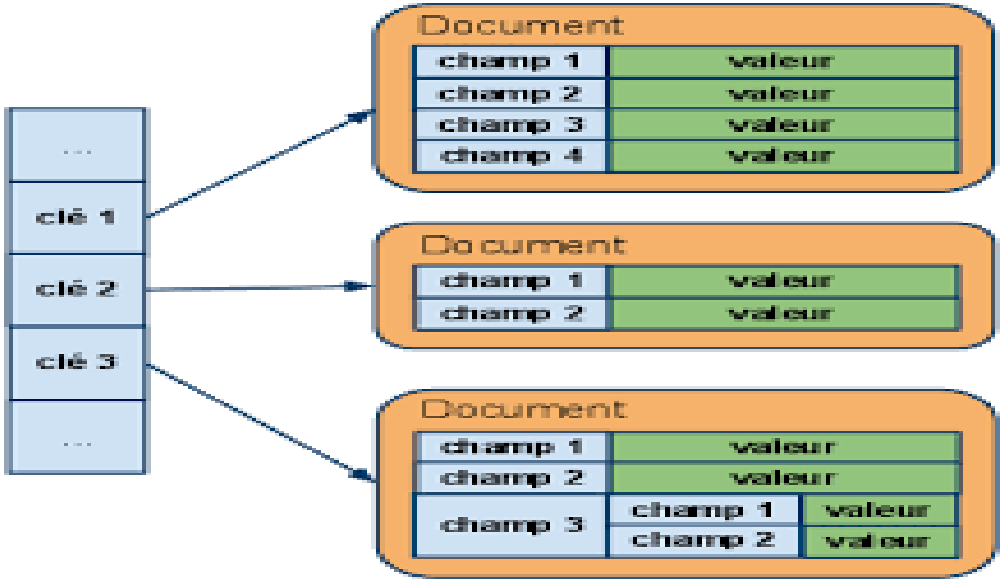
- Manque d'interopérabilite avec les autres systèmes
- Augmentation significative des coûts pour garder le contrôle des versions plus anciens des enregistrements

V- 2 Différents types de base de données NoSQL

❖ Les moteurs orientés documents

Le format de sérialisation et d'échange de données le plus populaire est aujourd'hui le JSON *(JavaScript Object Notation)* Il permet d'exprimer un document structuré qui comporte des types de données simples, mais aussi des listes et des paires clé-valeur, sous une forme hiérarchique.

illustration



V- 2 Différents types de base de données NoSQL

❖ Les moteurs orientés documents

Avantages

- Idéal pour représenter des données structurées ou semi-structurées
- Schéma flexible
- Bonne évolutivité

Limites

- Pas de jointures native supportée

V- 2 Différents types de base de données NoSQL

❖ Les moteurs orientés documents

La nuance entre moteurs paires clé-valeur et moteurs orientés documents a tendance à s'estomper, parce qu'au fur et à mesure de leur évolution, les premiers intègrent souvent un support du JSON C'est notamment le cas avec Riak qui depuis sa version 2 supporte l'indexation secondaire, donc sur des documents JSON dans la partie valeur de ses données

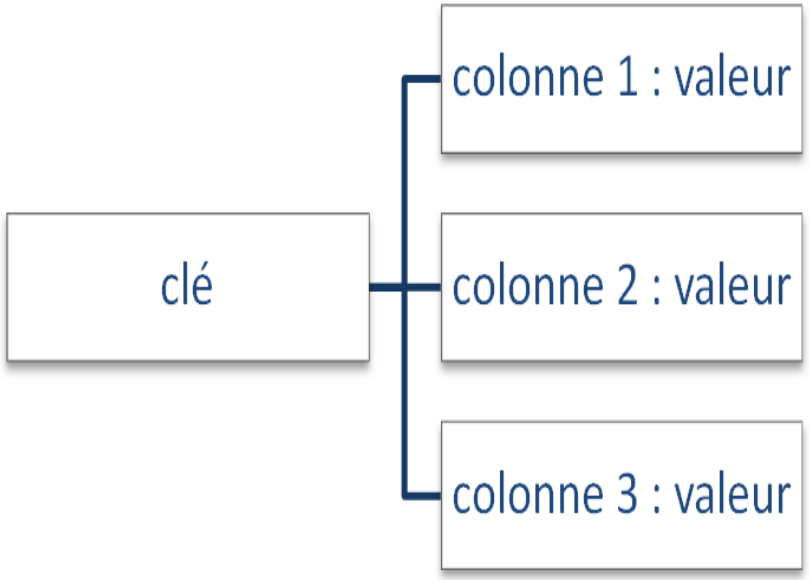
V- 2 Différents types de base de données NoSQL

❖ Les moteurs orientés colonnes

Inspirés par Google BigTable, plusieurs moteurs NoSQL implémentent une structure proche de la table, dont nous avons déjà parlé et que nous avons définie comme une table de hachage distribuée Contrairement aux moteurs orientés documents, les données sont ici clairement représentées en lignes et séparées par colonnes Chaque ligne est identifiée uniquement par une clé, ce qu'on appelle dans le modèle relationnel une clé primaire, et les données de la ligne sont découpées dans des colonnes, ce qui représente un niveau de structuration plus fort que dans les deux modèles précédents.

V- 2 Différents types de base de données NoSQL

- **Les moteurs orientés colonnes**



V- 2 Différents types de base de données NoSQL

❖ Index inversé

Un index inversé est une correspondance entre un terme, ou du contenu, et sa position dans un ensemble de données, par exemple un document ou une page web Google utilise un index inversé pour répondre aux recherches sur son moteu

Pourquoi les considérer comme des moteurs de bases de données ?

Ils permettent de manipuler une structure JSON, de la chercher et de la restituer. Cela permet à l'utilisateur de travailler avec une structure de données semblable à un moteur orienté documents, et de profiter d'excellentes capacités de requêtage grâce au moteur de recherche.

V- 2 Différents types de base de données NoSQL

❖ Les moteurs orientés graphe

Bien que les bases de données de type clé-valeur, colonne, ou document tirent leur principal avantage de la performance du traitement de données, les bases de données orientées graphe permettent de résoudre des problèmes très complexes qu'une base de données relationnelle serait incapable de faire.

Le défi ici n'est pas le nombre d'élément à gérer, mais le nombre de relations qu'il peut y avoir entre tous ces éléments. En effet, il y a potentiellement n^2 relations à stocker pour n éléments. Même s'il existe des solutions comme les jointures, les bases de données relationnelles se confrontent très vite à des problèmes de performances ainsi que des problèmes de complexité dans l'élaboration des requêtes.

V- 2 Différents types de base de données NoSQL

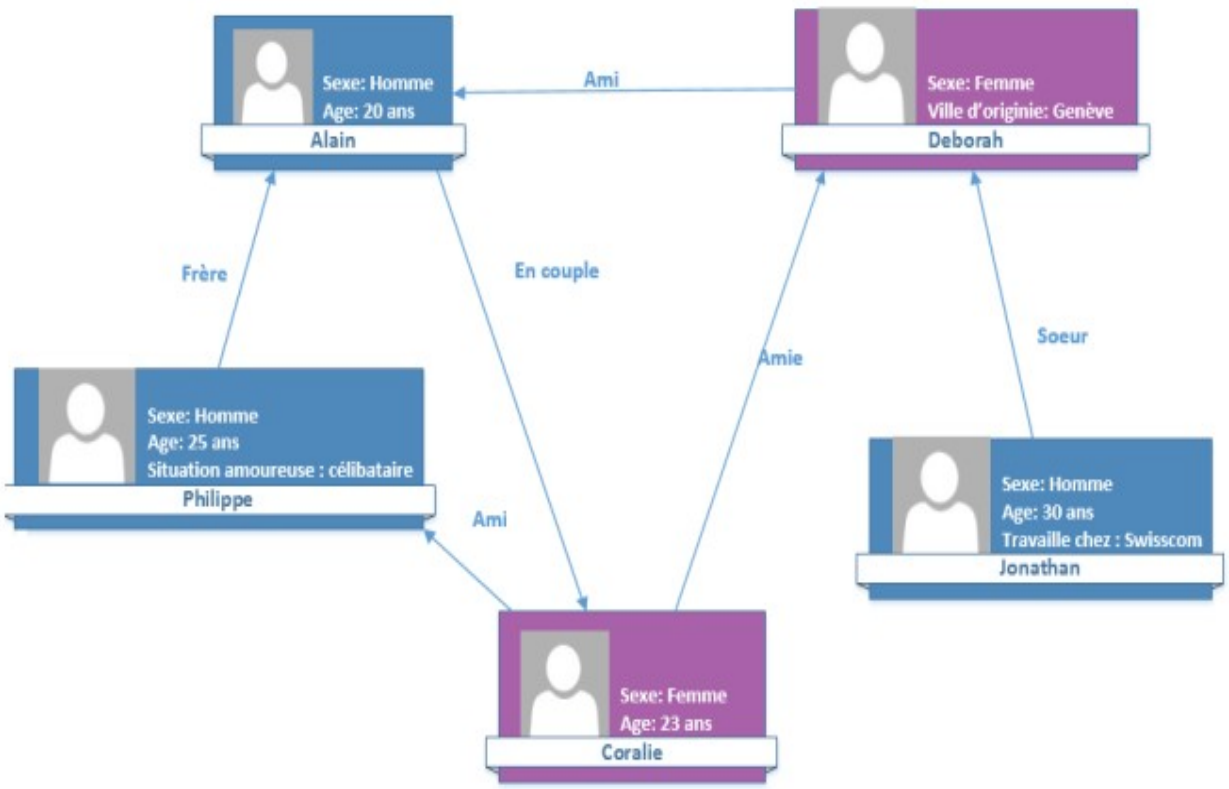
❖ Les moteurs orientés graphe

Comme son nom l'indique, ces bases de données reposent sur la théorie des graphes, avec trois éléments à retenir :

- Un objet (dans le contexte de Facebook nous allons dire que c'est un utilisateur) sera appelé un Nœud.
- Deux objets peuvent être reliés entre eux (comme une relation d'amitié).
- Chaque objet peut avoir un certain nombre d'attributs (statut social, prénom, nom etc.)

V- 2 Différents types de base de données NoSQL

- **Les moteurs orientés graphe**



Interrogation

- **Comment les informations sont-elles structures dans le NoSQL?**
- **Comment manipulons-t-on les données ?**