Story 2 - Introduction base de données

Prérequis

Notions basiques BDD

Contexte

Connaître les différents types de BDD et types de données.

Mots clés

▼ **Donnée**

Caractères, chiffres, lettres, mots brutes, sans contexte (chiffres, lettres, caractère, …). → **Base de donnée (case)**

▼ **Information**

Ensemble de données structurées et compréhensibles. (Adresse, n° sécurité sociale, …). Peut être vrai ou faux. → **Système d'information (Requête)**

▼ **Connaissance**

Relations entre les informations. Règles / Algorithme / Modèle. Implique plusieurs informations.
→ **Base de connaissances (Ensemble de règle dans une base de connaissance)**
Lorsqu'on traite les données pour en tirer des patterns dans un contexte particulier

▼ **méta-connaissance**

Connaissance des connaissances.

▼ **Champ (attribut)**

> Caractéristique ou attribut unique (Tête de colonne).
> Différent de champ de valeur qui correspond à une donnée (case).

▼ **Enregistrement (record)**

> Ensemble de valeur de champs associés (ligne). (= tuple)

▼ **Table**

> Collections d'enregistrement (données similaires).

▼ **Données structurées**

> Données typés bien définies.
>
> **Données formatées dans des champs prédéfinies afin d'être facilement interprété par un SGBD.**

Les données structurées sont des informations qui ont été formatées et transformées en un modèle de données bien défini. Les données brutes sont mappées dans des champs prédéfinis qui peuvent ensuite être extraits et lus facilement via SQL. Les bases de données relationnelles SQL, constituées de tables avec des lignes et des colonnes, sont le parfait exemple de données structurées.

▼ **Données semi-structurées**

> Données définies qui peuvent avoir certaines incohérences.
> **Ensemble de données structurées et non structurées.**
> **exemple image (non structurée) et ses métadonnées (structurée).**

Les données semi-structurées sont un type de données qui présentent des caractéristiques cohérentes et définies. Il ne se confine pas dans une structure rigide telle que celle nécessaire aux bases de données relationnelles. Les propriétés organisationnelles telles que les métadonnées ou les balises sémantiques sont utilisées avec des données semi-structurées pour les rendre plus gérables ; cependant, il contient encore une certaine variabilité et des incohérences.

▼ **Données non structurées**

> Données brutes, qui peuvent contenir plusieurs informations à l'intérieur d'une donnée. **exemple : Texte, image, JSON.**

Données présentes sous forme brute. Il peut s'agir d'un paragraphe d'un livre contenant des informations pertinentes ou d'une page Web. Un exemple de données non structurées pourrait également être des fichiers journaux qui ne sont pas faciles à séparer.

▼ **SGBD** (=Systèmes de Gestion de Bases de Données)

Logiciel destiné au stockage et à la manipulation de bases de données. **Plutôt sous-entendu relationnelle.**

▼ **Base de données**

> Collection de données que l'on peut traiter (CRUD).
> **+ Données structurées, semi-structurées ou non structurées indexées pour faciliter leur recherche.**

C'est un ensemble de données qui sont stockées sur un support informatique, et structurées de manière à pouvoir facilement consulter et modifier leur contenu.

▼ **Base de données relationnelle**

> Base de données qui contient des données structurées, regrouper par thème, et que l'on peut interconnecter **par un champ commun**.

La disposition des données est sous forme de tableau et on peut créer autant de tableaux que l'on veut et les faire interagir entre eux.

Ce qui fait la grande force et la spécificité de ce moteur de base de données c'est la possibilité de créer la structure de la base de données. En SQL, on appelle cela des schémas. Les schémas permettent de définir comment les données seront stockées, le type de données, ce qui est obligatoire et/ou optionnel, etc.

▼ **SQL (Stuctured Query Language)**

> Langage de programmation qui permet de communiquer avec une base de données relationnelle.

(Structured Query Data) est un langage, un langage de base de données. Vous pouvez ajouter, modifier ou supprimer des données. Bien entendu, vous pouvez également lire les données en appliquant des critères de restriction.

▼ **MySQL**

> SGBD (système de gestion de base de donnée) : logiciel de gestion de base de donnée relationnelle.

▼ **Base de données non relationnelle = NoSQL (Not Only SQL).**

Base de données qui peut accepter des données semi-structurées ou non-structurées. Il existe plusieurs configurations pour différents usages :

▼ **NoSQL orienté agrégats**

Une des caractéristiques retrouvées dans de nombreux SGBD NoSQL est
l'utilisation d'« agrégats de données » constitués d'un ensemble de données souvent « consultées/modifiées en même temps » et qui peuvent
être déployées sur des serveurs indépendants.
À titre d'exemple considérons une application de commerce-en-ligne conçue de manière à accéder fréquemment aux informations d'un client (adresses, etc) et à l'historique de ses achats (par exemple pour lui proposer des réductions).

- Un SGBD relationnel typique modélisera ce système en créant une table des informations clients et une table des achats puis en effectuant une jointure entre ces dernières à chaque opération. Cette architecture pourra poser problème lorsque le nombre de clients/achats deviendra important et sera difficile à répartir entre plusieurs serveurs.
- A contrario une architecture SGBD NoSQL typique aura tendance à modéliser ce problème en un ensemble d'agrégats constitué des informations d'un client et de ses achats. Cette architecture est plus facilement scalable. En effet ces agrégats interagissent peu entre eux et peuvent facilement être répartis sur un cluster de serveurs. Cette architecture pourra par contre poser problème si pour une raison quelconque on est amené à effectuer des requêtes qui ne correspondent pas aux cas d'utilisation immédiatement considérés. Par exemple une demande de calcul du nombre total de clients ou d'achats pourra être moins efficace que dans un système relationnel qui conserve l'ensemble des clients (resp. achats) dans une table unique.

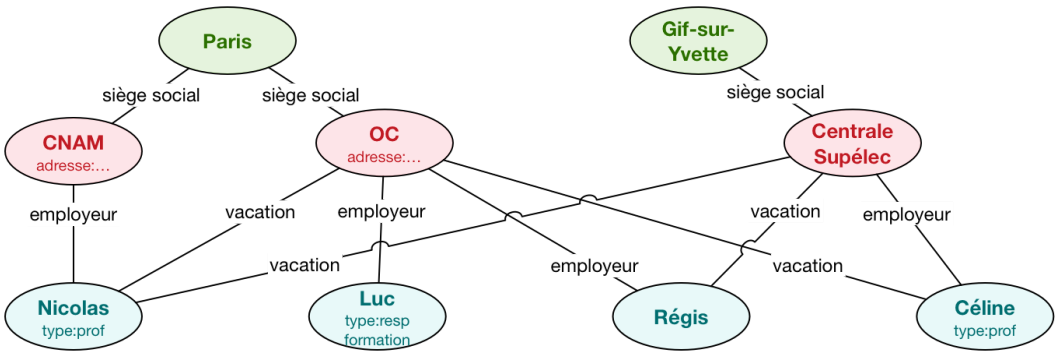
▼ **NoSQL orienté-graphes**

Les bases de données orientées graphe permettent de stocker les données sous forme de graphe et de faciliter l'écriture des requêtes

en supprimant la plupart des jointures. Par
rapport aux bases relationnelles, l'efficacité de ces bases est variable en fonction des systèmes et tâches et des configurations.
Ces données sont typiquement celles des réseaux sociaux, réseaux de transport, topologies, ou systèmes de recommandation de documents.
On distingue habituellement les triplestores des bases de données orientées-graphe. Les bases de données graphe fonctionnent avec différents types de graphe (ex. : pondérés, clusters,
graphe et tables mixtes) et offrent souvent de meilleures performances
pour les traversées de graphes.

Prenons l'exemple d'un réseau social : dans certains cas, il devient très complexe de calculer la distance entre deux personnes non directement connectées. Et c'est ce type d'approche que résolvent les bases orientées Graphe.

Dans la base orientée graphe, les données stockées sont : les nœuds, les liens et des propriétés sur ces nœuds et ces liens. Les requêtes que l'on peut exprimer sont basées sur la gestion de chemins, de propagations, d'agrégations, voire de recommandations. Toutefois, contrairement aux solutions précédentes la distribution des nœuds sur le réseau n'est pas triviale.



▼ NoSQL sans-schéma

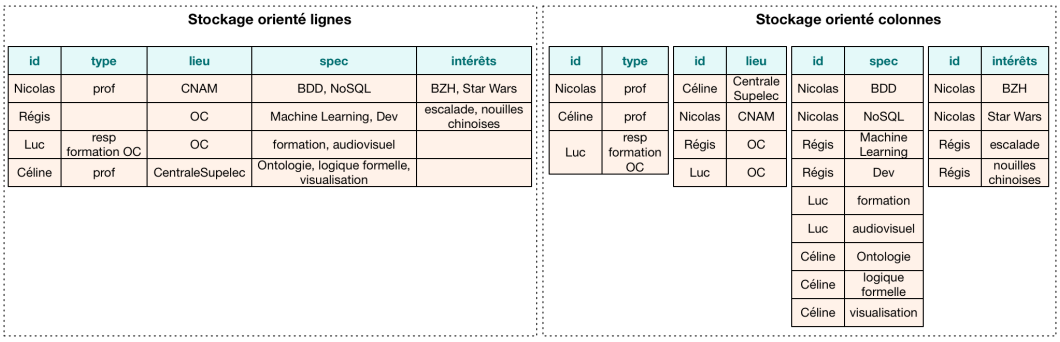
La première étape de la création d'une base de données relationnelle est de définir son schéma c'est-à-dire l'ensemble des tables la

composant et l'ensemble des champs de ces tables. Cette étape crée une certaine rigidité dans l'implémentation, implique d'avoir une vision assez claire des évolutions de l'application et peut poser problème si la structure des données recueillies change dans le temps. Les systèmes NoSQL sans-schéma peuvent ignorer cette étape et stocker des données hétérogènes au fur et à mesure de leur alimentation. Cette utilisation permet une grande flexibilité et des capacités d'adaptation au niveau de la base de données. La contrepartie est que les applications qui liront la base de données devront être capables d'intégrer des données plus hétérogènes et de structure plus complexe.

▼ Orientés colonnes

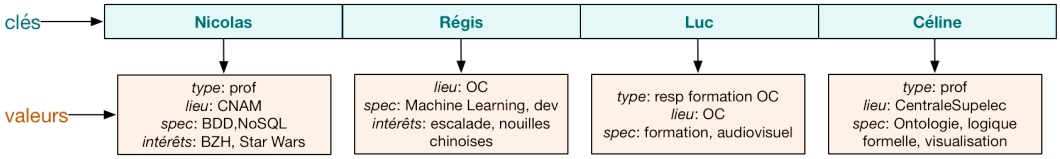
Traditionnellement, les données sont représentées en ligne, représentant l'ensemble des attributs. Le stockage orienté colonne change ce paradigme en se focalisant sur chaque attribut et en les distribuant. Il est alors possible de focaliser les requêtes sur une ou plusieurs colonnes, sans avoir à traiter les informations inutiles (les autres colonnes).

Cette solution est très adaptée pour effectuer des traitements sur des colonnes comme les agrégats (comptage, moyennes, co-occurences...). D'une manière plus concrète, elle est adaptée à de gros calculs analytiques. Toutefois, cette solution est beaucoup moins appropriée pour la lecture de données spécifiques comme pour les clés/valeurs.



▼ Orientées clé/valeur

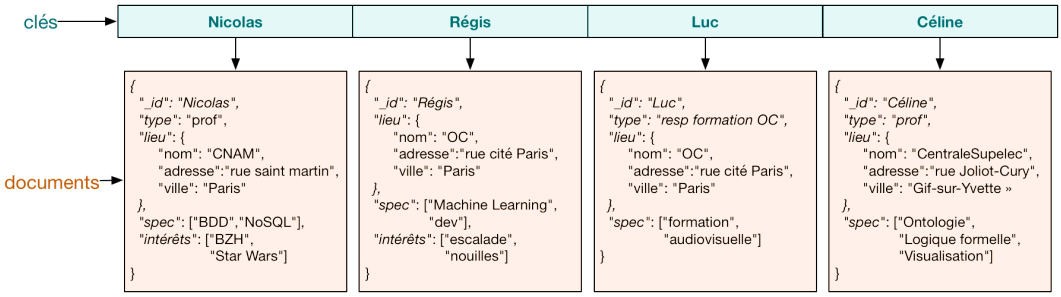
Tout repose sur le couple Clé/Valeur. La clé identifie la donnée de manière unique et permet de la gérer. La valeur contient n'importe quel type de données.



▼ Orientée "document"

Les bases orientées documents ressemblent sans doute le plus à ce que l'on peut faire dans une base de données classique pour des requêtes complexes. Le but de ce stockage est de manipuler des documents contenant des informations avec une structure complexe (types, listes, imbrications). Il repose sur le principe du clé/valeur, mais avec une extension sur les champs qui composent ce document.

L'avantage de cette solution est d'avoir une approche structurée de chaque valeur, formant ainsi un document. De fait, ces solutions proposent des langages d'interrogation riches permettant de faire des manipulations complexes sur chaque attribut du document (et sous-documents) comme dans une base de données traditionnelles, tout en passant à l'échelle dans un contexte distribué.



On appelle aussi les bases de données non relationnelles des bases de données NoSQL (Not Only SQL). Leur principale différence avec les bases de données relationnelles est qu'elle s'abstienne de la plupart des contraintes que peut avoir le relationnel, à commencer par les schémas. Vous pouvez stocker les données un peu comme vous le voulez, n'importe comment

même si ce n’est pas conseillé de le faire bien sûr. Il y a plusieurs types de bases de données NoSQL : les orientées documents (MongoDB, CouchDB), les clés-valeurs (Redis, Memcached) et les orientées colonnes (ElasticSearch, BigTable). En clair, chaque document peut avoir d’une certaine façon son propre schéma ce qui rend l’utilisation et la pratique du NoSQL beaucoup plus souple, simple et rapide à utiliser.
Bien entendu, vous faites complètement l’impasse sur l’intégrité de vos données ainsi que de leur sécurité et de leur cohérence même s’il est possible de pallier avec votre langage de programmation.

Un modèle typique en NoSQL est le système clé-valeur, avec une base de données pouvant se résumer topologiquement à un simple tableau associatif unidimensionnel avec des millions — voire des milliards — d'entrées. Parmi les applications typiques, on retrouve des analyses temps-réel, statistiques, du stockage de logs (journaux), etc.

▼ **NoSQL**

(Not Only SQL). Utiliser pour le big data !

▼ **Big Data**

> Ensemble de données diverses dont le volume et la vélocité (vitesse d’arrivée des données) nécessite des capacités de traitement importantes.

Le ***big data***, les mégadonnées ou les données massives, désigne les ressources d’informations dont les caractéristiques en termes de **volume**, de **vélocité** et de **variété** imposent l’utilisation de technologies et de méthodes analytiques particulières pour créer de la valeur, et qui dépassent en général les capacités d'une seule et unique machine et nécessitent des traitements parallélisés.

▼ **Cloud Computing**

> C’est un système de stockage sur des serveurs dans des datacenters distants.

Le cloud computing est la pratique consistant à utiliser des serveurs informatiques à distance et hébergés sur internet pour stocker, gérer et traiter des données, plutôt qu'un serveur local ou un ordinateur personnel.

Problématiques

Quels type de base de donnée utiliser ?

Pour des données structurées on privilégiera les BDD relationnelles si on privilégie la sécurité, la cohérence des données et que la base n’est pas appelé à changer d’échelle. Si les données sont trop volumineuse (dispatchées sur plusieurs serveur), le NoSQL est aussi préférable.

Pour des données semi-structurées ou non structurées on utilisera des BDD non relationnelles.

Comment stocker et organiser nos données (structurées ou non) ?
Grâce à des bases de données adaptées à nos données.

Hypothèses (vraies ou fausses)

Le type de donnée nous permet d'utiliser qu'un seul type de base de données.

Les données **structurées** peuvent être utilisées dans des bases de données **relationnelles ou non**.
Les données **non structurées** ou semi-structurées ne peuvent être utilisées que dans des bases de données **non relationnelle**.

Les différents types de base de données présentent des contraintes différentes en terme d'opération qui peuvent être conduites sur les données -Etienne

Vrai

Plan d'action

- [ ] Parcourir des ressources

https://apprendre-la-programmation.net/type-de-base-de-donnees-choisir/

https://www.mysql.com/fr/why-mysql/

https://fr.wikipedia.org/wiki/NoSQL

https://github.com/ClementDelgrange/Cours_bases_de_donnees/blob/master/Presentation_BDD.md

Les miennes :

https://openclassrooms.com/fr/courses/4462426-maitrisez-les-bases-de-donnees-nosql/4462433-choisissez-votre-famille-nosql

- [ ] Définir les mots clés

Histoire

Les SGBD relationnels créés dans les années 1970 se sont progressivement imposés jusqu'à devenir le paradigme de bases de données très largement dominant au début des années 1990.

Les différents type.
C'est dans le courant des années 2000 avec le développement de grandes entreprises internet (Google, Amazon, eBay…) brassant des quantités énormes de données et le développement de l'informatique en grappes que la domination sans partage du modèle relationnel a été remise en question car souffrant de limites rédhibitoires pour ces nouvelles pratiques.

Ce sont les grandes entreprises du web amenées à traiter des volumes de données très importants qui ont été les premières confrontées aux limitations intrinsèques des SGBD relationnels traditionnels. Ces systèmes fondés sur une application stricte des propriétés ACID
et généralement conçus pour fonctionner sur des ordinateurs uniques ont rapidement posé des

problèmes d'extensibilité. Afin de répondre à ces limites, ces entreprises ont commencé à développer leurs propres systèmes de gestion de bases de données pouvant fonctionner sur des architectures matérielles distribuées et permettant de traiter des volumes de données importants.

La rencontre de 2009 à San Francisco est considérée comme l'inauguration de la communauté des développeurs de logiciels NoSQL. Des développeurs qui, selon le magazine Computerworld, « racontent comment ils ont renversé la tyrannie des coûteux et lents SGBD relationnels par des moyens plus simples et plus rapides de manipuler des données ». Selon Jon Travis, un des présentateurs de la conférence, « les SGBD relationnels en font trop, alors que les produits NoSQL font exactement ce dont vous avez besoin ».

Avantages / Inconvénients

	BBD relationnelle	BBD non relationnelle	données structurées	données non structurées
Avantages	+ Données structurées + Cohérentes + Sécurisées + Possibilité de gérer des droits d'accès différents à la base de données et aux données	+ Données non/semi-structurées + Performance (vitesse) + Plus flexible	+ Facile à ordonner + Typées	+ Flexible (pas de schéma) + Peuvent être traiter en masse + scalable
Inconvénients	+ Peu scalable	+ Moins sécurisée + Regroupement et croisement de données plus difficiles	+ Traitement rigide	+ Le regroupement et le croisement de données sont plus difficiles à réaliser qu'en SQL + La sécurité et la cohérence des données sont sacrifiées même si une équipe de développeur peut le minimiser, il y aura toujours un risque.

Les différents type de BDD NoSQL

▼ **NoSQL orienté agrégats**

Une des caractéristiques retrouvées dans de nombreux SGBD NoSQL est l'utilisation d'« agrégats de données » constitués d'un ensemble de données souvent « consultées/modifiées en même temps » et qui peuvent être déployées sur des serveurs indépendants.
À titre d'exemple considérons une application de commerce-en-ligne conçue de manière à accéder fréquemment aux informations d'un client (adresses, etc) et à l'historique de ses achats (par exemple pour lui proposer des réductions).

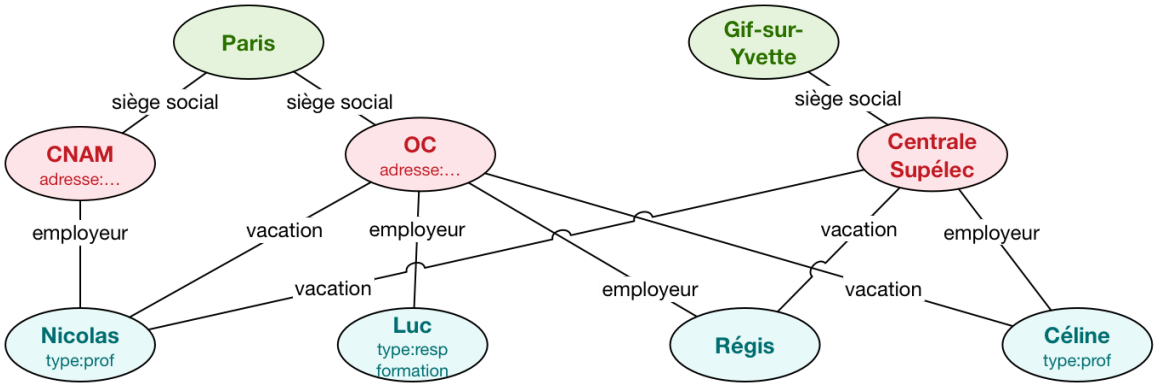
- Un SGBD relationnel typique modélisera ce système en créant une table des informations clients et une table des achats puis en effectuant une jointure entre ces dernières à chaque opération. Cette architecture pourra poser problème lorsque le nombre de clients/achats deviendra important et sera difficile à répartir entre plusieurs serveurs.
- A contrario une architecture SGBD NoSQL typique aura tendance à modéliser ce problème en un ensemble d'agrégats constitué des informations d'un client et de ses achats. Cette architecture est plus facilement scalable. En effet ces agrégats interagissent peu entre eux et peuvent facilement être répartis sur un cluster de serveurs. Cette architecture pourra par contre poser problème si pour une raison quelconque on est amené à effectuer des requêtes qui ne correspondent pas aux cas d'utilisation immédiatement considérés. Par exemple une demande de calcul du nombre total de clients ou d'achats pourra être moins efficace que dans un système relationnel qui conserve l'ensemble des clients (resp. achats) dans une table unique.

▼ **NoSQL orienté-graphes**

Les bases de données orientées graphe permettent de stocker les données sous forme de graphe et de faciliter l'écriture des requêtes en supprimant la plupart des jointures. Par rapport aux bases relationnelles, l'efficacité de ces bases est variable en fonction des systèmes et tâches et des configurations.
Ces données sont typiquement celles des réseaux sociaux, réseaux de transport, topologies, ou systèmes de recommandation de documents.
On distingue habituellement les triplestores des bases de données orientées-graphe. Les bases de données graphe fonctionnent avec différents types de graphe (ex. : pondérés, clusters, graphe et tables mixtes) et offrent souvent de meilleures performances
pour les traversées de graphes.

Prenons l'exemple d'un réseau social : dans certains cas, il devient très complexe de calculer la distance entre deux personnes non directement connectées. Et c'est ce type d'approche que résolvent les bases orientées Graphe.

Dans la base orientée graphe, les données stockées sont : les nœuds, les liens et des propriétés sur ces nœuds et ces liens. Les requêtes que l'on peut exprimer sont basées sur la gestion de chemins, de propagations, d'agrégations, voire de recommandations. Toutefois, contrairement aux solutions précédentes la distribution des nœuds sur le réseau n'est pas triviale.



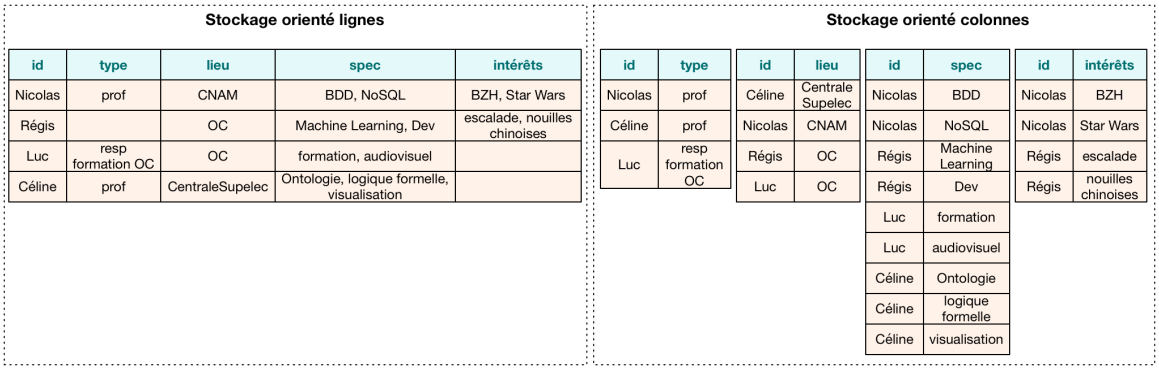
▼ NoSQL sans-schéma

La première étape de la création d'une base de données relationnelle est de définir son schéma c'est-à-dire l'ensemble des tables la composant et l'ensemble des champs de ces tables. Cette étape crée une certaine rigidité dans l'implémentation, implique d'avoir une vision assez claire des évolutions de l'application et peut poser problème si la structure des données recueillies change dans le temps. Les systèmes NoSQL sans-schéma peuvent ignorer cette étape et stocker des données hétérogènes au fur et à mesure de leur alimentation. Cette utilisation permet une grande flexibilité et des capacités d'adaptation au niveau de la base de données. La contrepartie est que les applications qui liront la base de données devront être capables d'intégrer des données plus hétérogènes et de structure plus complexe.

▼ Orientés colonnes

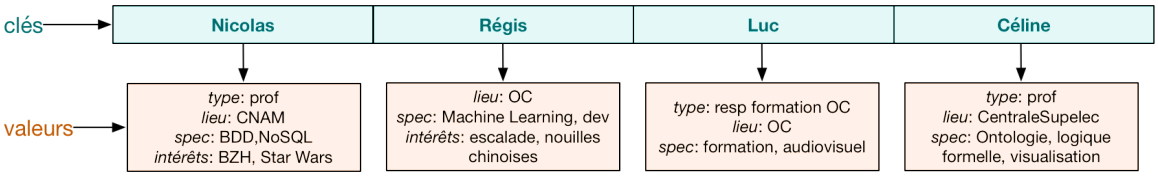
Traditionnellement, les données sont représentées en ligne, représentant l'ensemble des attributs. Le stockage orienté colonne change ce paradigme en se focalisant sur chaque attribut et en les distribuant. Il est alors possible de focaliser les requêtes sur une ou plusieurs colonnes, sans avoir à traiter les informations inutiles (les autres colonnes).

Cette solution est très adaptée pour effectuer des traitements sur des colonnes comme les agrégats (comptage, moyennes, co-occurences...). D'une manière plus concrète, elle est adaptée à de gros calculs analytiques. Toutefois, cette solution est beaucoup moins appropriée pour la lecture de données spécifiques comme pour les clés/valeurs.



▼ Orientées clé/valeur

Tout repose sur le couple Clé/Valeur. La clé identifie la donnée de manière unique et permet de la gérer. La valeur contient n'importe quel type de données.

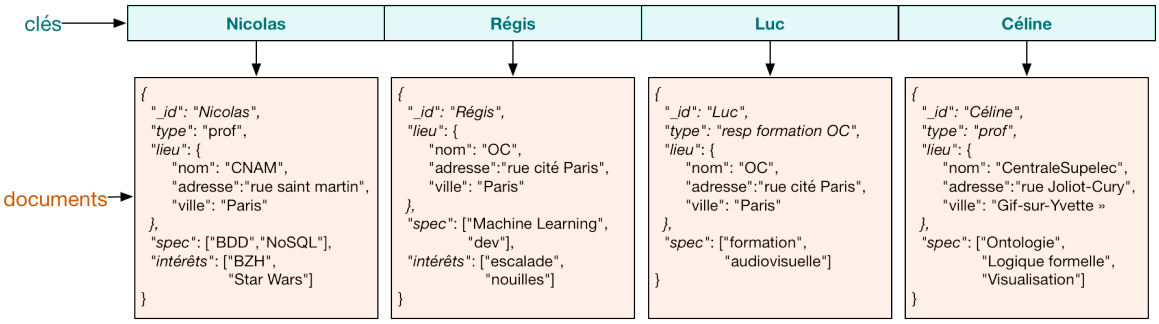


▼ Orientée "document"

Les bases orientées documents ressemblent sans doute le plus à ce que l'on peut faire dans une base de données classique pour des requêtes complexes. Le but de ce stockage est de manipuler des documents contenant des informations avec une structure complexe (types,

listes, imbrications). Il repose sur le principe du clé/valeur, mais avec une extension sur les champs qui composent ce document.

L'avantage de cette solution est d'avoir une approche structurée de chaque valeur, formant ainsi un document. De fait, ces solutions proposent des langages d'interrogation riches permettant de faire des manipulations complexes sur chaque attribut du document (et sous-documents) comme dans une base de données traditionnelles, tout en passant à l'échelle dans un contexte distribué.



Objectifs

Remarques pertinentes