Quelques éléments de NoSQL

Xavier Gendre

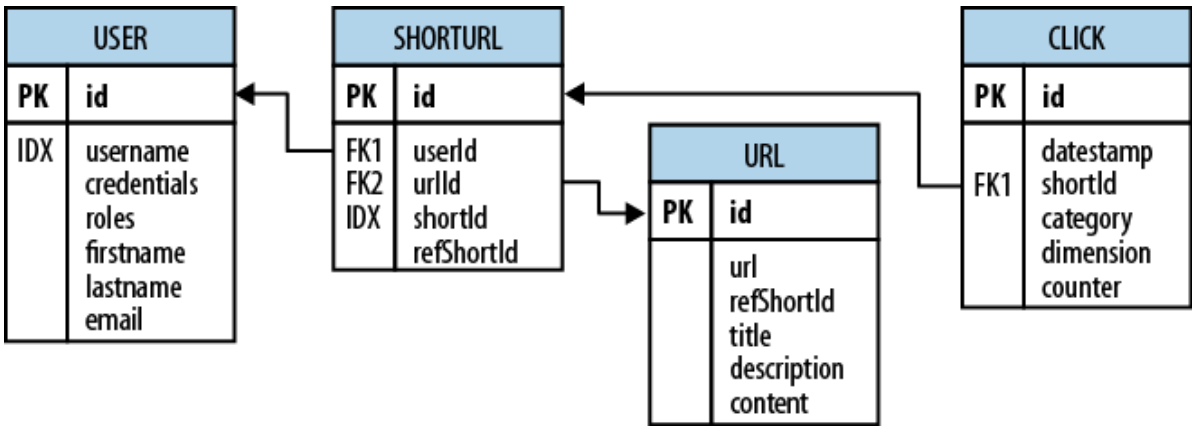
Propriétés ACID

Les propriétés **ACID** (*Atomicité*, *Cohérence*, *Isolation* et *Durabilité*) garantissent la fiabilité d'une transaction informatique (par exemple, avec un serveur de base de données).

- **Atomicité** : une transaction se fait intégralement ou pas du tout,
- **Cohérence** : une transaction amène le système d'un état valide à un autre état valide,
- **Isolation** : une transaction s'exécute comme si elle était la seule sur le système (*i.e.* pas de dépendance entre les transactions),
- **Durabilité** : une transaction confirmée demeure enregistrée même à la suite d'une panne.

Ces propriétés font partie des fondements des systèmes de gestion de bases de données relationnelles.

Un exemple de passage à l'échelle



Le projet consiste à réaliser un réducteur d'URL. Pour quelques milliers d'utilisateurs, un simple serveur de bases de données relationnelles tel qu'une plateforme LAMP suffit : données normalisées,clés étrangères, utilisation d'index, …

Début de la célébrité

En passant à quelques dizaines de milliers d'utilisateurs :

- la pression sur le serveur de base de données augmente,
- il faut ajouter des serveurs applicatifs dits **secondaires** (aucune difficulté, ils partagent leur état avec un serveur central dit **primaire**),
- la ressource CPU et les lectures/écritures commencent à être un problème sur le serveur primaire.

Architecture Primaire-Secondaire

- Plus de serveurs secondaires permettent plus de **lectures** en parallèle.
- Le serveur primaire gère les **écritures** (il y en a souvent beaucoup moins que des lectures).
- Les serveurs secondaires répliquent le serveur primaire.
- Les serveurs secondaires se **répartissent la charge** des lectures.

Le succès est au rendez-vous

En passant à quelques centaines de milliers d'utilisateurs :

- le nombre de lectures est encore un goulot d'étranglement,
- les serveurs secondaires commencent à peiner pour répondre aux requêtes,
- il faut mettre en place un système de **cache** (*Memcached*, *Redis*, …) pour se décharger des requêtes identiques vers un système de mémoire rapide :
  - la **consistence** n'est plus garantie,
  - une invalidation du cache sur un serveur devient **critique**.

La gloire internationale

Si le nombre d'utilisateurs augmente encore :

- les écritures commencent aussi à devenir un goulot d'étranglement,
- le serveur primaire souffre de la charge d'écriture,
- il faut évoluer **verticalement** : le serveur primaire doit être renforcé (le coût est important).

En plus de ces considérations, le modèle relationnel pose aussi des difficultés :

- les jointures deviennent des goulots d'étranglement,
- optimiser demande de **dénormaliser**,
- les **procédures stockées** (instructions SQL précompilées) chargent trop le CPU,
- il faut abandonner certains index,
- …

Que faire ?

Et si le nombre d'utilisateurs augmente encore après ça ?

. . .

Évolution verticale vs. Évolution horizontale

L'idée de l'évolution horizontale revient à répartir les données dans plusieurs bases de données plutôt que de les garder toutes sur le même serveur :

- il faut contrôler les accès,
- il faut contrôler la partition des données,
- il faut dénormaliser pour permettre les jointures.

Évolution horizontale

Les bases de données relationnelle sont conçues pour tourner sur une seule machine : pour passer à l'échelle, il faut une plus grosse machine (*i.e.* évolution verticale).

Passer à l'échelle horizontalement en investissant dans un groupe de "petites" machines est moins cher et plus efficace. Un tel groupe est appelé un **cluster**.

Les machines d'un cluster sont généralement peu fiables mais l'ensemble reste fiable.

Cloud

Ce qui est généralement désigné par le "cloud" cache justement ce type de cluster de machines, ce qui explique pourquoi les bases de données relationnelles ne s'intègrent pas bien dans cet univers.

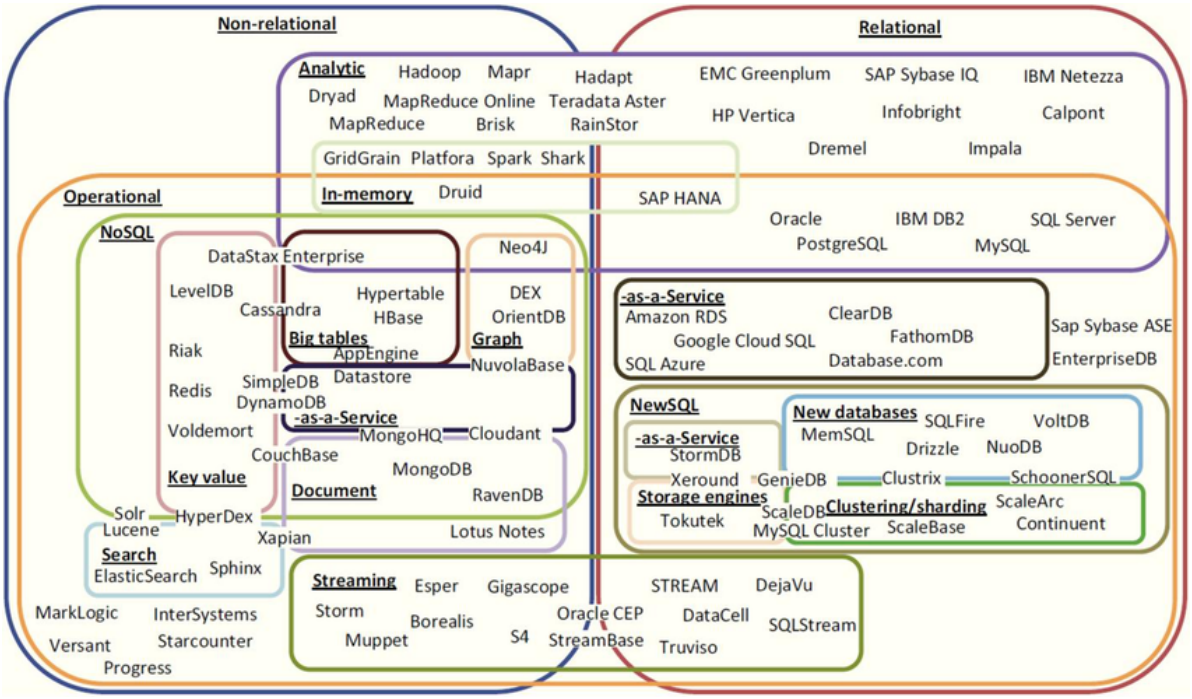
Google et Amazon ont tous deux été parmi les premiers à adopter ces grands clusters. Ils ont évité les bases de données relationnelles dans leurs mises en place. Leurs efforts ont été une grande inspiration pour la mouvance **NoSQL**.

NoSQL

Le terme NoSQL est un néologisme dont la définition fait débat. Une signification souvent évoquée est *Not only SQL* mais cela reste discutable.

Quelques caractéristiques :

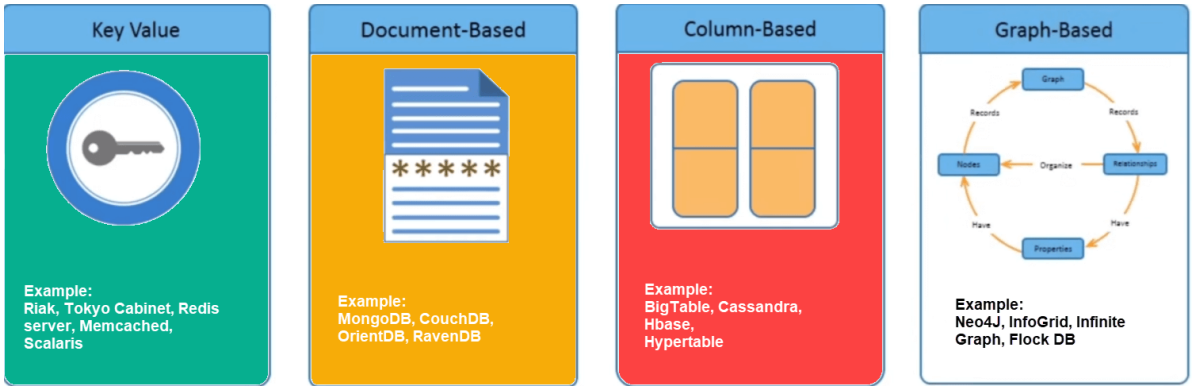
- pas/peu d'utilisation du modèle relationnel,
- un bon comportement sur des architectures de type cluster,
- souvent pensé pour passer à l'échelle,
- des solutions libres et/ou open source,
- des interfaces dans plusieurs langages.



Catégories de bases NoSQL

- Clé/Valeur
- Document

- Colonne
- Graphe



Comment choisir ?

Le choix doit se faire selon les besoins :

- **Relationnel** : schéma, relations, transactions, …
- **Clé/Valeur** : traitement à flux constant de petites quantités de lectures et d'écritures, …
- **Document** : modèle de données naturel, simple à manipuler, adapté au Web, **CRUD**, …
- **Colonne** : bonne gestion de la taille, forte charge d'écritures, haute disponibilité, répartition sur plusieurs centres, …
- **Graphe** : relations, algorithmes sur les graphes, …

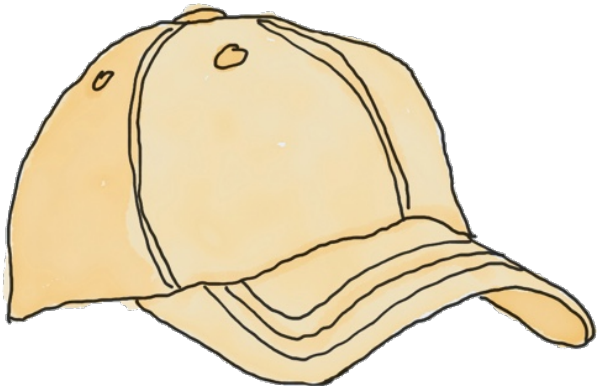
Opérations CRUD

Les opérations **CRUD** sont des opérations standards qu'une base de données peut proposer :

- **Creation** : insertion de données,
- **Read** : récupération de données (recherche, filtre, …),
- **Update** : modification de données,
- **Deletion** : suppression de données.

Théorème CAP de Brewer

Ce résultat initialement conjecturé par l'informaticien Eric Brewer deviendra un théorème en 2002 quand Seth Gilbert et Nancy Lynch en donneront une preuve formelle.



Un système informatique distribué ne peut garantir au plus que deux des contraintes suivantes :

- **Cohérence** (*Consistency*) : toutes les parties du système voient exactement les mêmes données au même moment,
- **Disponibilité** (*Availability*) : toutes les requêtes sont garanties de recevoir une réponse,
- **Tolérance au partitionnement** (*Partition Tolerance*) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (*i.e.* en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome).