Not Only SQL

Introduction

-SGBDR

Le schéma d'une base relationnelle est prévu pour répondre à toutes les requêtes possibles mais en contrepartie de cette généricité il est très structuré : non duplication de données, jointures entre les tables, colonnes des tables normalisées, contraintes d'intégrité, etc.

Ces contraintes des formes normales, les schémas relationnels imposent la **non-duplication des données** qui, elle-même, **impose** de faire des **jointures entre plusieurs tables**.
L'exécution d'une jointure peut entraîner la lecture d'une table (ou d'un index). Ces scans dépendent, en effet, de la quantité de données. La **performance** de l'exécution des jointures peut donc se **dégrader** au fur et à mesure que la quantité de données augmente.
Les contraintes d'acidité liées aux bases relationnelles, notamment la contrainte de cohérence et d'isolation des transactions, imposent au SGBDR de déposer des verrous sur les données lorsqu'une transaction s'exécute. L'attente de la libération de ces verrous ralentit la base de données. En effet, plus le nombre de transactions est élevé, plus les performances se dégradent, ce qui est dû, entre autres, à l'attente de **libération des verrous**.
Au-delà d'une certaine volumétrie et d'un certain nombre de transactions, il faut donc donner plus de ressources au SGBDR afin de garantir les performances.

Les deux méthodes pour donner plus de ressources sont dites : scalabilité horizontale et scalabilité verticale.

La scalabilité des SGBDR peut être horizontale (multiplication du

nombre de serveurs comme dans des solutions type Oracle RAC) mais uniquement en lecture, à cause des contraintes de cohérence.

Le modèle de scalabilité est vertical lorsqu'il est lié directement à la configuration matériel, le coût des serveurs n'augmente pas de façon linéaire mais exponentielle, surtout lorsqu'on passe une certaine gamme. La scalabilité verticale peut donc coûter très cher. En outre, cette technique de scalabilité est limitée par la configuration maximale des serveurs.

Les temps de réponse ne sont pas constants donc non prédictibles car ils dépendent de la quantité de données et du nombre de requêtes. Cet état de fait rend certaines applications incompatibles avec le modèle relationnel.

Lien sur temps de réponse et normalisation :

http://blog.developpez.com/sqlpro/p10070/langage-sql-norme/base_de_donnees_et_performances_petites

La croissance réelle du NoSQL est dû à 3 acteurs principaux
-Les premiers responsables sont les développeurs. Leur mauvaise maîtrise de la conception des schémas relationnels fait que leur requêtes sont toujours très longues dès que celles-ci comportent des jointures.
« c'est embêtant les jointures, cela demande un effort intellectuel chronophage »
-Mais aussi à celle de mysql, ce sgbdr n'a que tardivement su gérer les jointures et il les traitait tellement mal que les performances se dégradaient rapidement.
-L'explosion des données et la nécessité du data-mining.

Effectivement les technologies en se développant ont permis de récolter de plus en plus de données, des données dans des domaines diverses.
Ex : quand une personne paie par carte bleue elle renseigne sa banque sur le prix des biens qui sont vendus/achetés, la date et l'heure de consommation, le lieux, la fréquence, le type (selon le type de commerce).

Propriétés et classification des bases NoSQL

Contrairement aux SGBDRs, les bases NoSQL ne possèdent pas les caractéristiques **ACID** (Atomicité, Cohérence, Isolation, Durabilité) car la performance a un prix.
Comme tout système réparti, on appréhende mieux les propriétés des bases NoSQL sous l'angle du théorème **CAP** (Consistency, Availability, Partition tolerance).

Le théorème CAP (ou théorème de Brewer) stipule qu'aucun système distribué ne peut pas être à la fois :

Cohérent : c'est-à-dire le fait que la modification d'une donnée prenne instantanément effet
Hautement disponible : c'est-à-dire être capable de répondre à toutes les requêtes
Tolérant au partitionnement : c'est-à-dire garantir une continuité du fonctionnement en cas d'ajout/suppression de partition (ou noeud) du système distribué.

Les bases NoSQL ont la particularité d'être plutôt hautement disponibles et tolérantes au partitionnement, mais ne garantissent généralement pas une cohérence stricte des données, comme on le verra ultérieurement.
Les bases NoSQL n'ont donc pas vocation à remplacer les bases de données relationnelles.

Depuis quelques années, une grande variété d'implémentations a vu le jour (HBase, Cassandra, Voldemort, Redis, MongoDB, CouchDB et bien d'autres). L'approche la plus commune pour les comparer repose sur la classification de l'évangéliste NoSQL Ben Scofield, identifiant 4 grandes familles de bases NoSQL :

bases de données clé/valeur distribuées ou non
bases de données orientées document
bases de données orientées colonne
bases de données orientées graphe

Toutes ces solutions sont conçues pour traiter un volume considérable de données tout en étant performantes car scalables horizontalement, mais elles ne répondent pas toutes aux mêmes cas d'usage.

Les bases de données clé/valeur

Ce type de bases de données peut être vu comme une gigantesque table associative clé/valeur éventuellement partitionnée sur plusieurs instances.
Le modèle s'en rapprochant le plus est celui du cache.
Les structures de données ne sont pas contraintes par un schéma.

Selon l'implémentation, la clé peut être attribuée ou générée aléatoirement lors de l'ajout de l'enregistrement.
Réputées performantes et capables de gérer un volume important de données, les bases clé/valeur sont par exemple adaptées à :

la collecte d'évènements (jeux en ligne),
la gestions de traces (mesure d'audience)
la gestions de profils utilisateurs des sites de forte audience (commerce électronique)

Sur un site de réseau social, à partir d'un utilisateur (la clé) je peux obtenir une liste de ses amis (la valeur)

Dans un catalogue de livres, le numéro ISBN (la clé) donne accès à tous les détails sur le livre (la valeur)

Dans un journal d'activités, la date de l'événement (la clé) indexe les détails de ce qui s'est passé à ce moment(la valeur).

La clé représente une information précise et atomique alors que la valeur peut être complexe et représenter un tableau ou une liste, elle même constituée de clés qui pointent sur des sous valeurs et ainsi de suite.

Le coût de la performance

Les bases de données relationnelles fournissent tout un dispositif garantissant l'intégrité et la cohérence des données. Ce n'est pas le cas des bases de type clé/valeur.

Considérons le cas simple d'une base gérant des informations Client et Adresse avec une relations 1-N.

Dans un modèle relationnel, les clés primaire et étrangère permettent de modéliser de manière triviale la modélisation de la relation client / adresses.

Dans un modèle clé/valeur, on stocke les entités mais pas les relations entre elles. On pourrait faire des ajustements en dénormalisant le modèle, mais l'application devra alors maintenir elle-même la cohérence des données en cas de mise à jour ou suppression.

Plus la modélisation métier est complexe, moins une base de type clé/valeur est adaptée. En effet un des objectifs des entrepôts de clé-valeur (c'est-à-dire des bases de données qui stockent exclusivement des paires clé/valeur) est la simplicité. Il n'y a vraiment plus de notion de schéma de données dans un tel système (schem-less).

Les différents sgbd :

Dynamo

Une base clé/valeur distribuée utilisée pour gérer le panier d'achat du géant du commerce en ligne Amazone.
A Dynamo est hautement disponible et tolérant au partitionnement mais ne garantit pas de cohérence stricte des données. La propagation des mises à jour, sur un nombre
déterminé de réplicats, s'appuie sur une communication peer-to-peer.
Dynamo est une solution propriétaire composante du cloud d'Amazon (AWS).

Redis

Multiple dans ses usages, Redis est également perçue comme une sorte de cache ou mémoire partagée sur TCP, ce qui le distingue d'une solution de persistance alternative aux bases de données relationnelles.
Redis n'est pas tolérant aux pannes.

Redis est singulier dans sa manière de gérer les données en

écriture. Stockées d'abord en mémoire, elles ne sont déversées sur disque que périodiquement.
Réputé pour ses performances (plusieurs dizaines de milliers d'insertions par seconde sur une machine de série), Redis est particulièrement adapté pour construire des systèmes analytiques temps réel à grande échelle (monitoring d'applications web par exemple).
A la différence des autres moteurs de type clé/valeur, Redis supporte plusieurs types (chaîne de caractère, listes ordonnées ou non, table de hachage) permettant de mieux organiser ses données ainsi qu'un grand nombre d'opérations pour les exploiter (déterminer des intersections ou union de liste, incrémenter/décrémenter un compteur etc.).
Enfin, Redis dispose d'un support natif du modèle publish/subscribe. Une utilisation conjointe avec Socket.IO (voir vos cours sur java) permet par exemple de construire aisément un compteur temps réel de visiteurs d'un site web.

Voldemort

Développée et utilisée par LinkedIn, Voldemort est une base de type clé/valeur distribuée développée en Java. Elle expose une API très simple, pour ne pas dire rudimentaire comparée à Redis (de type put/get/delete).

Les données sont :
organisées en stores, entité analogue à la notion de table, au sein desquels une clé est unique mais où la valeur associée peut être versionnée répliquées et partitionnées automatiquement sur plusieurs noeuds de sorte que chaque noeud ne soit responsable que d'un sous ensemble des données, faisant de Voldemort une solution tolérante aux pannes.

Riak

Edité par Basho Technologies, Riak est une base hybride orientée clé/valeur ou document inspirée de Dynamo.
Son architecture distribuée masterless ne contient pas de **SPOF** (point unique de défaillance).
Riak implémente nativement un moteur MapReduce (les jobs peuvent être écrits en Erlang ou en JavaScript) et supporte la recherche full-text via Riak Search.

Bases de données orientées document.

Ce type de base trouvera naturellement sa place dans les systèmes de gestion documentaire (CMS) tant le modèle s'y prête bien mais est également adapté dans les cas d'usage suivants :

archivage / stockage de traces applicatives
statistiques et analyse temps-réel
plateforme de commerce électronique (souvent en association avec un SGBDR)
prototypage

Elles ont vocation à **stocker des données semi-structurées**, c'est-à-dire dont la structure n'est pas contrainte par un schéma mais dont le contenu est néanmoins formaté.
L'unité de stockage est un document, généralement au format JSON ou XML.
D'une certaine façon, on peut définir un document comme un ensemble de couples propriété/valeur, dont la seule contrainte est de respecter le format de représentation.
En ce sens, on peut considérer la base de donnée orientée document comme une sorte d'évolution de la base de donnée clé/valeur distribuée.
Voici par exemple comment pourrait se représenter un document JSON :

```
{
"titre": "Formation Big Data",
"datePublication" : Date("2016/09/01"),
"auteur": "Needemand",
"tags": [ "bigdata", "nosql" ],
"commentaires": [ {
        "auteur": "John Doe",
        "commentaire": "Qui suis je ?"
        }, {
        "auteur": "Gros Minet",
        "commentaire": "pffft il est où titi ?"
        }
]
}
```

Dans un modèle relationnel, on imagine bien le nombre substantiel de tables impliquées pour normaliser ces données et l'effort à produire pour les restituer (requête avec jointures).
Les bases de données orientées document poussent donc à la dénormalisation.

CouchDB

CouchDB est une base de données NoSQL développée en Erlang, orientée document (au format JSON) et exposant une API REST HTTP.
CouchDB partage avec Lotus Notes le concept de base orientée document avec vue.
Une vue est un moyen d'effectuer des requêtes sur la base de documents. Elle est définie dans un document JSON au format particulier contenant l'identifiant de la vue ainsi que les fonctions Map/Reduce écrites en JavaScript et implémentant les règles de recherche.
Les vues sont indexées et mises à jour automatiquement à mesure que des documents sont ajoutés, modifiés ou supprimés.
CouchDB est conçu pour ne pas introduire de verrou lors des accès concurrents. En d'autres termes, les accès en lecture ne bloquent pas les accès en écriture et réciproquement.
La scalabilité horizontale est assurée par réplication.

MongoDB

Développé par la société MongoDB(ex 10gen), MongoDB est le compétiteur direct de CouchDB. Contrairement à ce dernier, MongoDB n'est pas MVCC et implémente un moteur de requêtage plus direct et expressif que les vues dans CouchDB.
MongoDB permet d'indexer les propriétés des documents afin d'optimiser une recherche.
Par ailleurs les documents, regroupés en collections (équivalent NoSQL des tables), sont stockés au format BSON (Binary JSON).
Une autre singularité de MongoDB est son support de l'indexation géospatiale en 2D depuis la version 1.4, à savoir la possibilité de répondre à des requêtes géographiques (par exemple trouver la station de métro la plus proche de ma position actuelle).
MongoDB supporte le **sharding** manuel et automatique, c'est-à-dire le partitionnement horizontal des données.

Dit autrement, le sharding fait référence au découpage des collections de données en plus petites unités (selon un critère donné, par exemple la valeur d'un attribut) et leur distribution sur un réseau de machines esclaves. On augmente ainsi la capacité de stockage tout en répartissant le traitement sur des machines de moyenne puissance.
MongoDB supporte un mécanisme de réplication maître/esclave garantissant une plus grande intégrité des données ainsi qu'une tolérance aux pannes.

Bases de données orientées colonne.

Ces dernières ont dans leur grande majorité été inspirées par Google BigTable, l'implémentation propriétaire du géant de la recherche en ligne dont les principes ont été publiés dans un célèbre papier (vous pouvez le rechercher, très intéressant à lire). Dont HBase et Cassandra sont les 2 implémentations libres les plus populaires à ce jour.

Dans une base de données relationnelles, une table est un ensemble de données organisées en colonnes dont le nombre est fixe quel que soit le nombre d'enregistrements. Lorsque l'on insère un enregistrement dans cette table, on spécifie une valeur pour chaque colonne, null étant une valeur par défaut. Voici un exemple de représentation d'un carnet d'adresse téléphonique :

id	Nom	tél	codepostal
1	max	63685757	34000
2	fred	7848484	null
3	louis	null	null

A contrario,
dans le modèle orienté colonne, le nombre de colonnes peut varier pour chaque enregistrement.
Voici la représentation du même carnet d'adresse selon ce modèle :

1			
	Nom	tél	codepostal
	max	652525252	34000

2		
	nom	tél
	fred	78888888

3	
	Nom
	louis

La dé-normalisation du modèle est donc fortement encouragée pour tirer pleinement parti des performances des bases orientées colonne.

HBase

HBase est un clone de BigTable développé au sein de l'écosystème Hadoop et utilisé par de nombreuses compagnies, dont Facebook et Yahoo!.
Les données sont stockées dans des fichiers appelés HFiles dans la terminologie HBase et gérés par HDFS, le système de fichier distribué d'Hadoop qui le sous-tend, garantissant l'intégrité des données.
HBase est réputé pour être très performant en lecture (là ou Cassandra est très performant en écriture), mais on lui reproche de présenter des points de défaillance inhérent à l'architecture de HDFS et HBase.

Dans une infrastructure Hadoop ou HBase, il y a 2 types de machines : l**es maîtres (le NameNode HDFS, le Master HBase)** et **les esclaves (les DataNode HDFS, les RegionServers HBase).**
L'infrastructure est inopérante en cas d'indisponibilité d'un composant maître, et particulièrement en cas d'indisponibilité du Namenode.
Composant névralgique de HDFS, ce dernier gère l'espace de nommage et l'arborescence du système de fichiers, les méta-données des fichiers et répertoires. Il centralise la localisation des blocs de données répartis sur le système. Sans Namenode, tous les fichiers peuvent être considérés comme perdus.
Il faut néanmoins tempérer ce risque dans la mesure où les

défaillances d’un Namenode sont extrêmement rares.
HBase est ainsi un peu moins centré sur la disponibilité et plus sur la cohérence des données et la tolérance au partitionnement.

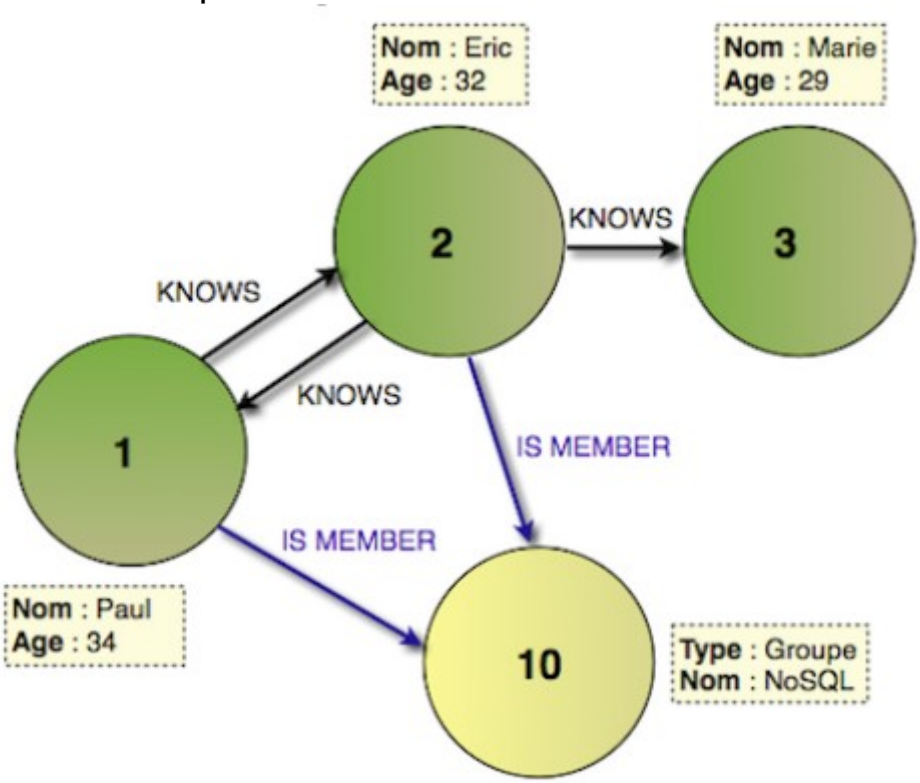
Cassandra

Initialement créé par Facebook, Cassandra est le compétiteur le plus sérieux de HBase.
Contrairement à ce dernier, son architecture distribuée et décentralisée le rend hautement disponible car les noeuds d’un cluster Cassandra jouent tous le même rôle (pas d’architecture maître/esclave). L’indisponibilité d’un noeud ne met pas en péril l’infrastructure.
En écriture, les données sont d’abord stockées en mémoire (Memtables) et ne basculent sur disque que lorsque la structure en mémoire a atteint un seuil défini par configuration.
Cassandra a ainsi la particularité d’être plus performante en écriture qu’en lecture au point de la qualifier de base de données orientée écriture.
Tolérant au partitionnement et hautement disponible, Cassandra ne garantit pas de cohérence stricte (strict consistency) des données. On parle plutôt de cohérence éventuelle signifiant par là que les mises à jour sont propagées dans l’ensemble des réplicats mais que cela prendra plus ou moins de temps. On parle également de cohérence personnalisable car le niveau de consistance peut être ajusté par configuration.
Cassandra supporte la réplication de données sur plusieurs datacenters répartis géographiquement.

bases de données orientées graphes

Fondées sur la théorie des graphes, elles sont conçues pour modéliser des structures de données relationnelles aussi diverses qu’un réseau social, un réseau ferroviaire ou l’organisation d’une entreprise .
Les bases de données relationnelles sont tout à fait adaptées pour gérer ce type de structures mais elles atteignent leurs limites sur le champ de la performance quand la profondeur du graphe et le volume de données à gérer sont importants.

Cela est dû aux structures récursives comme par exemple des arborescences de fichiers ou encore des graphes sociaux, on se retrouve alors avec des jointures lourdes.
Chaque opération sur une relation dans un réseau résulte en une opération de jointure dans le SGBDR, implémentée comme une opération ensembliste entre l'ensemble des clés primaires de deux tables - une opération lente et sans capacité à monter en charge alors que le nombre de t-uples de ces tables augmente.
Sur ce point, les bases de données orientées graphe surclassent les SGBDRs.
Dans le monde des bases orientées graphe, tout est affaire de noeuds, de relations et de propriétés. Les propriétés portent aussi bien sur les noeuds que les relations.
Ce modèle est très adapté pour gérer des données sociales ou spatiales, mais également financières où les relations modéliseraient les dépôts ou retraits.



Considérons l'exemple ci-dessus, Il représente la réalité sociale suivante :

Paul (34 ans) et Eric (32 ans) se connaissent et sont membres du groupe NoSQL
Eric connait Marie, 29 ans

Neo4J

Neo4J est sans conteste l'implémentation (développée en Java) la plus connue, distribuée sous la forme de 3 éditions. La seule édition gratuite (Community Edition) n'offre ni outils de monitoring, ni haute disponibilité.

FlockDB

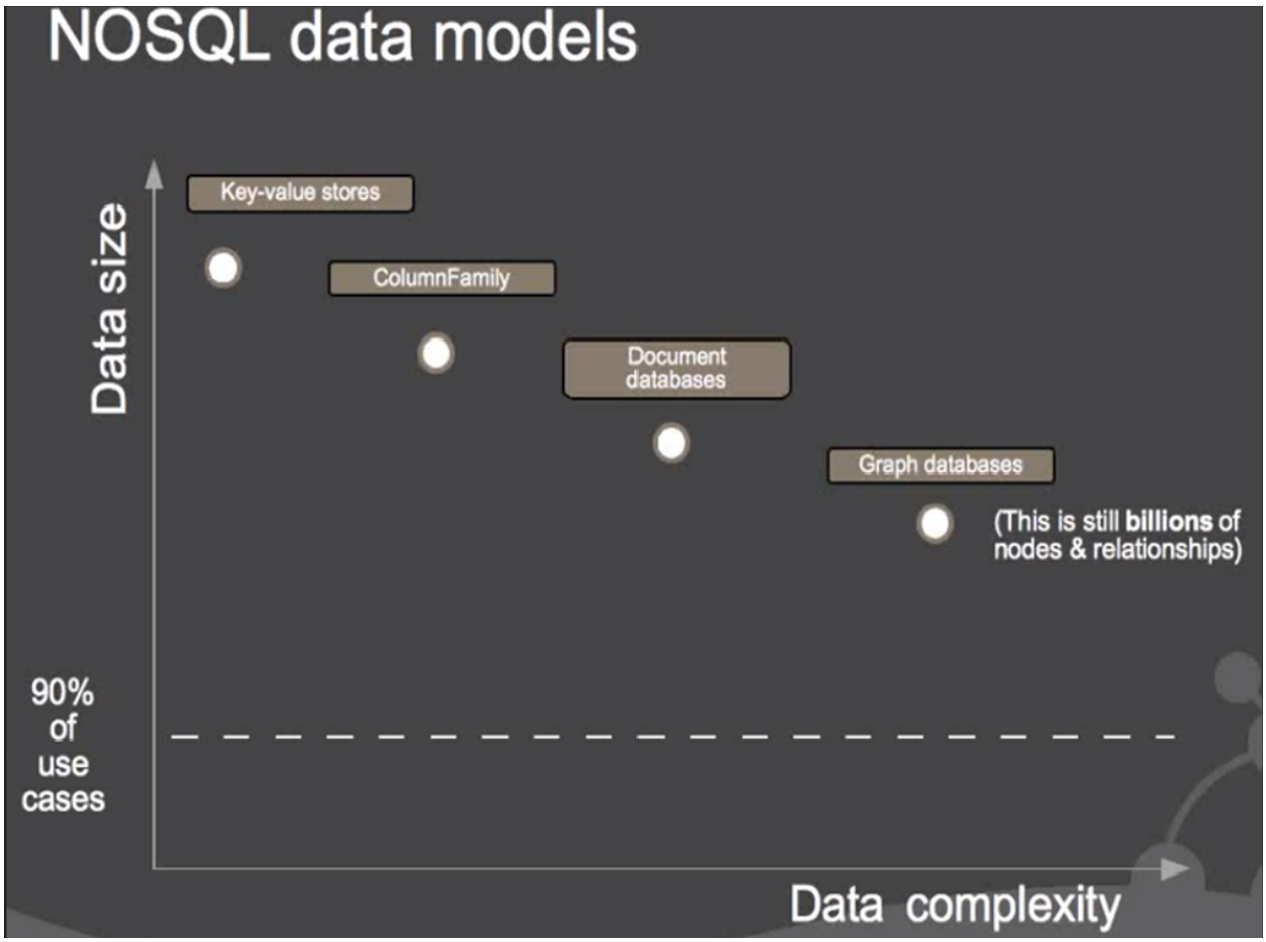
Créé par Twitter et développé en Scala, FlockDB est disponible sous licence libre. Pour information, le cluster FlockDB de Twitter gère plus de 13 milliards de relations et supporte jusqu'à 20 000 écritures par seconde.

OrientDB

OrientDB est une nouvelle venue promise à un bel avenir. Développée en Java, elle est scalable horizontalement et réputée plus performante que Neo4J. Rappelons que la version gratuite de Neo4J n'offre pas de support pour la haute disponibilité.

Comme vous pouvez le constater il existe une multitude de sgbd NoSQL.
Chacun est adapté à un type de demande et de données.

En attendant un petit tableau


NOSQL data models
Data size
Key-value stores
ColumnFamily
Document databases
Graph databases
(This is still billions of nodes & relationships)
90% of use cases
Data complexity