

Bases de données noSQL

Polytech Marseille, SN, 4A

Simon Vilmin

simon.vilmin@univ-amu.fr

2023 - 2024



À propos du cours

Objectifs :

- connaître les grands principes se cachant derrière le noSQL
- connaître les principaux modèles

Évaluation :

- examen final
 - des questions sur les concepts généraux
 - une partie « dissert » sur une question *posée à l'avance*

Ressources :

- supports sur AMeTICE

Sources principales

- Cours de Farouk Toumani
<https://perso.limos.fr/~fatouman/index.html>
- Cours de Bernard Espinasse
<https://pageperso.lis-lab.fr/bernard.espinasse/index.php/>
- Cours de Philippe Declercq
<http://declercq.e-monsite.com/pages/enseignement/>
- Graph Databases, Ian Robinson, Jim Webber, and Emil Eifrem, 2015
[graph-databases-2nd/9781491930885](#)
- Les bases de données NoSQL et le Big Data, Rudi Bruchez, 2015
[les-bases-de-donnees-nosql](#)

Plan

Intro

Du hiérarchique au noSQL

- Retour sur quelques SGBD

- Le modèle relationnel

- Arrivée des modèles noSQL

Distribution et données semi-structurées

- Données semi-structurées

- Distribution

Les 4 cavaliers du noSQL

- généralités

- Clé-valeur

- Documents

- Colonnes

- Graphes

Pour finir

Du hiérarchique au noSQL

Intro

Du hiérarchique au noSQL

- Retour sur quelques SGBD

- Le modèle relationnel


- Arrivée des modèles noSQL

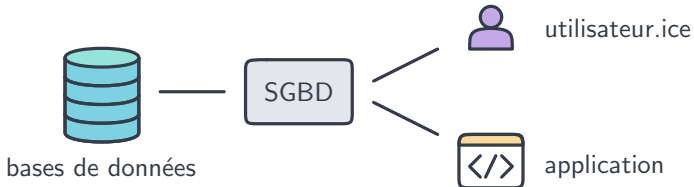
Distribution et données semi-structurées

Les 4 cavaliers du noSQL

Pour finir

Systèmes de gestion des bases de données (SGBD)

 **Définition :** (reformulée de [Wikipedia](#)) un *Système de Gestion de Bases de Données (SGBD)* est un logiciel servant à stocker, à manipuler ou gérer, et à partager des données dans une *base de données*, en garantissant la validité des données et en cachant la complexité des opérations.



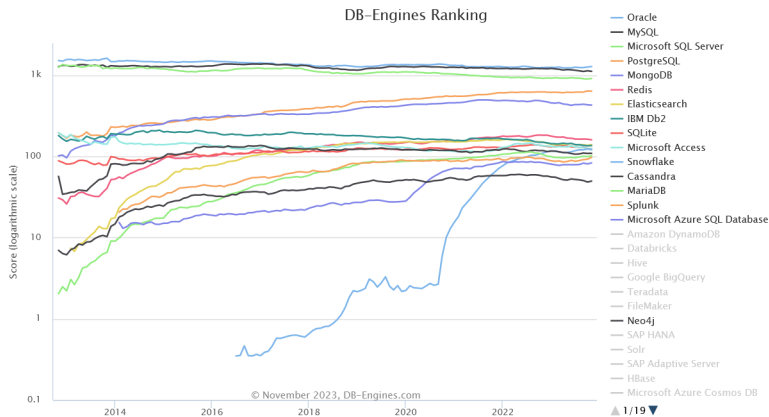
Les SGBD utilisés aujourd'hui

416 systems in ranking, November 2023

Rank			DBMS	Database Model	Score		
Nov 2023	Oct 2023	Nov 2022			Nov 2023	Oct 2023	Nov 2022
1.	1.	1.	Oracle	Relational, Multi-model	1277.03	+15.61	+35.34
2.	2.	2.	MySQL	Relational, Multi-model	1115.24	-18.07	-90.30
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	911.42	+14.54	-1.09
4.	4.	4.	PostgreSQL	Relational, Multi-model	636.86	-1.96	+13.70
5.	5.	5.	MongoDB	Document, Multi-model	428.55	-2.87	-49.35
6.	6.	6.	Redis	Key-value, Multi-model	160.02	-2.95	-22.03
7.	7.	7.	Elasticsearch	Search engine, Multi-model	139.62	+2.48	-10.70
8.	8.	8.	IBM Db2	Relational, Multi-model	136.00	+1.13	-13.56
9.	9.		SQLite	Relational	124.58	-0.56	-10.05
10.	10.		Microsoft Access	Relational	124.49	+0.18	-10.53
11.	11.		Snowflake	Relational	121.00	-2.24	+10.84
12.	12.		Cassandra	Wide column, Multi-model	109.17	+0.34	-8.96
13.	13.	13.	MariaDB	Relational, Multi-model	102.09	+2.43	-2.82
14.	14.	14.	Splunk	Search engine	97.32	+4.95	+3.10
15.	15.		Microsoft Azure SQL Database	Relational, Multi-model	83.17	+2.24	-0.49
16.	16.		Amazon DynamoDB	Multi-model	82.24	+1.32	-3.16
17.	17.		Databricks	Multi-model	77.22	+1.40	+16.33
18.	18.		Hive	Relational	68.64	-0.54	-13.25
19.			Google BigQuery	Relational	59.31	+2.74	+5.18
20.			Teradata	Relational, Multi-model	57.33	-1.23	-7.90
21.	21.	21.	FileMaker	Relational	52.44	-0.88	-1.87
22.			Neo4j	Graph	49.70	+1.26	-7.60

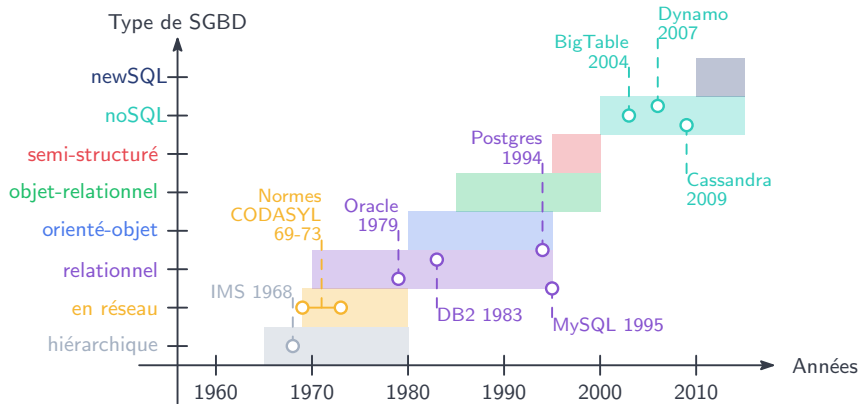
Source : <https://db-engines.com/en/ranking> (étude faite sur 416 SGBD)

Les SGBD utilisés aujourd'hui



Source : https://db-engines.com/en/ranking_trend

Une histoire (très) partielle des SGBD



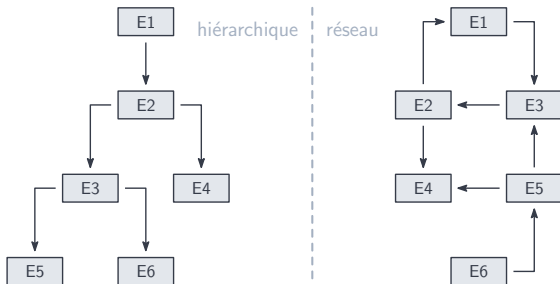
Frise *très partielle* basée (en plus des autres sources) sur :

- What Goes Around Comes Around, Stonebraker et al., 2006
- What's New with NewSQL ?, Pavlo et al., 2016

Modèle relationnel origins

Contexte années 60-70 : deux modèles de stockage des données

- *hiérarchique* : entités/données sous forme d'arbre
- *en réseau* : entités/données connectées en graphe



i Note : là-dessus, au début des années 70, *Edgar F. Codd* pose les bases du *modèle relationnel* en utilisant la *théorie des ensembles*

Modèle relationnel en bref

- base de données (BD) = collection de *tables* (ou *relations*)
- *langage déclaratif de requêtes* (standardisé) SQL
- *système de jointure* entre les tables offrant la possibilité de construire des requêtes complexes
- *système d'intégrité référentielle* (contraintes) assurant la validité des liens logiques entre les données

livres	nolivre	noauteur	prix	titre
	2574	3	10	Iris, c'est votre bleu
	3204	2	8	Le rhume
	2812	2	9	Mémoires trouvés dans une baignoire

écrire (clé étrangère)


auteurs	noauteur	nom	prenom	attribut
	1	Breton	André	
	2	Lem	Stanislas	tuple
	3	Dreyfus	Ariane	


les règles de Codd

Codd grave dans le marbre des règles régissant le modèle relationnel, par ex :

- « *indépendance* des modèles physique et logique, il y aura »
- « un langage de requête *déclaratif*, tu utiliseras »
- « des données *fortement structurées*, tu stockeras »
- « sous forme de tableau, ta base de données sera »
- « une forte *cohérence transactionnelle*, tu garantiras »

Transaction ?

 **Définition** : une *transaction* est une séquence d'opérations de lecture ou de mise à jour sur une base de données.

 **Important** : Les transactions doivent préserver la *cohérence* de la BD au niveau logique (contraintes, etc ...)

Les *SGBD relationnels (SGBDR)* assurent une *gestion des transactions* sur le principe **ACID** :

- **A**tomicity : une transaction est accomplie en entier ou pas du tout
- **C**onsistency : passage d'un état cohérent à un autre état cohérent
- **I**solation : indépendance des transactions concurrentes
- **D**urability : les modifications sont enregistrées dans la BD

Longue vie au modèle relationnel

Depuis 50 ans :

- multitudes de SGBDR : PostgreSQL, MySQL, Oracle, ...
- BD relationnelles très largement implantées, encore aujourd'hui

Les BD relationnelles :

- sont adaptées à des données bien structurées
- gèrent *très bien* les traitements **OLTP** (**O**nLine **T**ransactional **P**rocessing)
gestion de ressources, lectures, écritures, ...
- *un peu moins bien* les traitements **OLAP** (**O**nLine **A**nalytical **P**rocessing)
statistiques, analyse de tendances, ...



Remarque : Dans les années 80-90, de nouveaux modèles naissent, mais avec *peu de succès* ...

- BD *objet-relationnel* ou *semi-structurées* (à base de documents XML)

Tout allait bien ... quand soudain

! **Important** : Dans les années 2000, les volumes de données *explosent* dû à l'évolution des capacités matérielles et l'essor d'Internet

En particulier, des géants du Web (Google, Facebook, Twitter, Amazon, LinkedIn, ...) débarquent :

1. avec des données *hétérogènes* et *peu structurées* en quantité *faramineuses*
2. avec des besoins toujours plus grands d'*analyse*, de *gestion* et de *stockage*

i **Note** : c'est le début du *Big Data*

Hmmm ... Big Data dis-tu ?

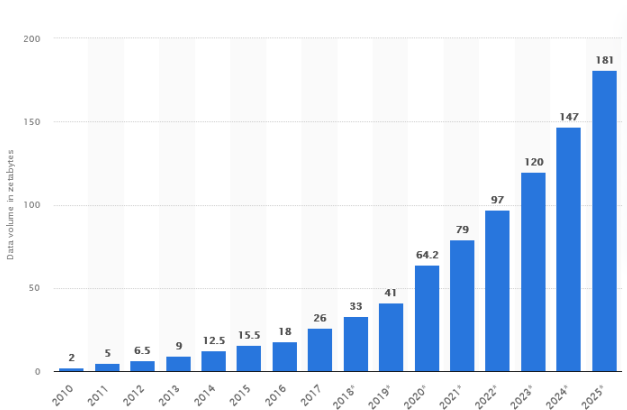
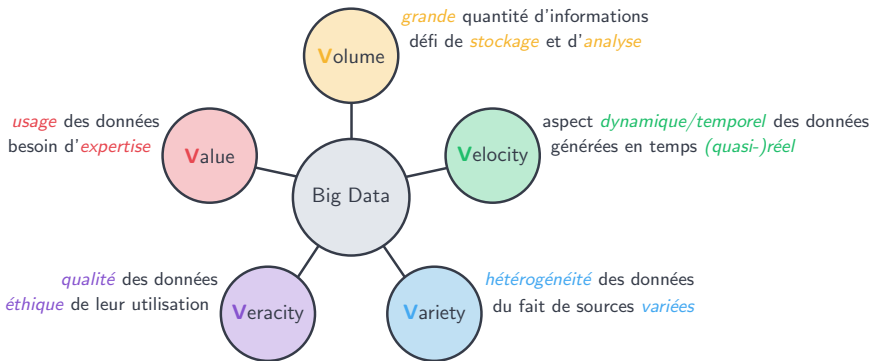


Figure – volume de données générées chaque année, (source : [statista](#), 2021)

i Note : 1 zetaoctet = 1 000 000 000 teraoctets !

Caractérisation par les 5V



! Important : « Big Data » est un terme dont la traduction recommandée est *mégadonnées*

Coup dur pour les BD relationnelles

❌ **Problème** : le modèle relationnel, *très structuré*, n'est pas compatible avec des données *hétérogènes* et *peu structurées* :

- pas possible d'*imbriquer les informations*
- schéma de BD et de relations *très rigide*

💡 **Idée** : besoin de modèles de stockage *plus flexibles* permettant de traiter des collections d'objets reliés *ensemble* !

❌ **Problème** : les SGBDR peuvent traiter de très gros volumes de données grâce à des *entrepôts de données*, mais là, c'est trop ...

💡 **Idée** : il faut *distribuer* les données et les calculs sur des machines : mise à l'échelle dite *horizontale*

Émergence de nouveaux modèles

Les grands acteurs du Web proposent alors de nouveaux modèles de SGBD :

- Google avec BigTable (2004)
- Amazon avec Dynamo (2007)
- Facebook avec Cassandra (2008)

! **Important** : c'est la naissance des systèmes (ou SGBD) *noSQL* !

Ces SGBD

- sont souvent *open-source*
- manipulent des données *non* ou *semi-structurées* (JSON, XML)
- implémentent *nativement* des mécanismes de *distribution* des données sur plusieurs nœuds (machines, clusters)

i **Remarque** : depuis, une multitude de systèmes sont apparus : MongoDB, Neo4J, Redis, HBase, Cassandra, ...

noSQL = fin le relationnel ?

! **Important** : Les bases noSQL ne *remplacent pas* les BD relationnelles. Elles sont une *alternative* apportant des solutions dans *certains contextes*

À l'heure actuelle :

- grande diversité de solutions techniques de stockage
- les entreprises utilisent plusieurs solutions (parfois une combinaison relationnel + noSQL)

i **Note** : Le terme « noSQL » est né en 2009 (Carl Strozzi) et à fait débat. En général on l'associe à « *not only SQL* »



Rappel :

- années 2000, croissance *exponentielle* de volumes de données *hétérogènes*
- les SGBDR ne sont *pas adaptés* aux traitements d'*immenses flux de données peu structurées*
- les SGBD noSQL naissent alors, avec comme caractéristiques :
 - stockage des données *non-relationnel*, donc *flexible*
 - prise en charge *native* de la *montée en charge horizontale* (distribution)
- les différents modèles *cohabitent*



Attention : noSQL ne veut pas dire « no SQL »

Distribution et données semi-structurées

Intro

Du hiérarchique au noSQL

Distribution et données semi-structurées

Données semi-structurées

Distribution

Les 4 cavaliers du noSQL

Pour finir

Centralité de la donnée vs. agrégats

Deux conceptions fondamentalement différentes de la donnée :

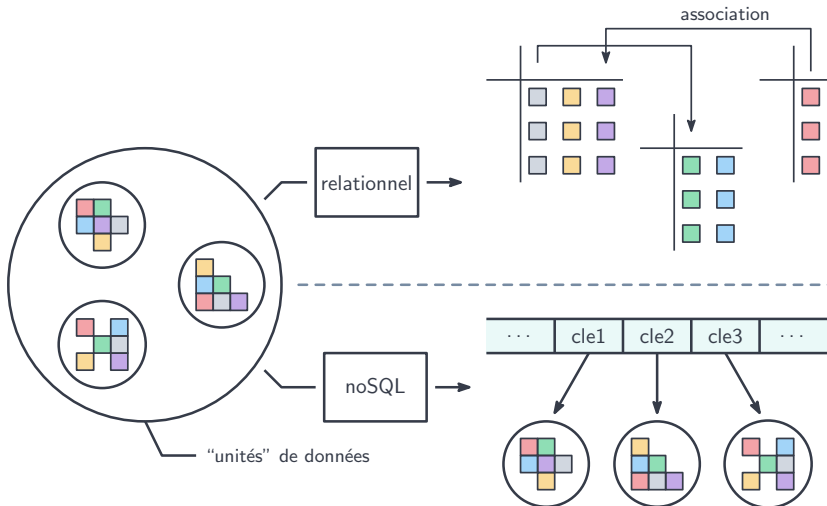
Relationnel : les données sont « *découpées* » et réparties dans des tables. Ce découpage rend une BD relationnelle accessible à une *grande variété d'applications différentes*. La donnée est *centralisée*

noSQL : stockage d'*agrégats*. Un *agrégat* est une collection d'objets liés par une entité racine (\simeq une clé). Un agrégat représente une *unité d'information complexe* traitée et stockée de façon *atomique*

! Attention : alerte esprit *critique* ! Les systèmes noSQL sont très différents les uns des autres : on ne peut pas forcément opposer tous les SGBD noSQL d'un côté, et les SGBDR de l'autre

- ex : dans les *BD graphes*, cette notion d'agrégation est moins vraie

Centralité vs. agrégats schématiquement



Conséquences

Quels impacts ?

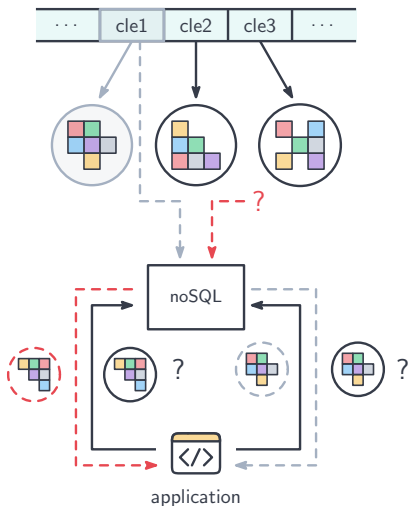
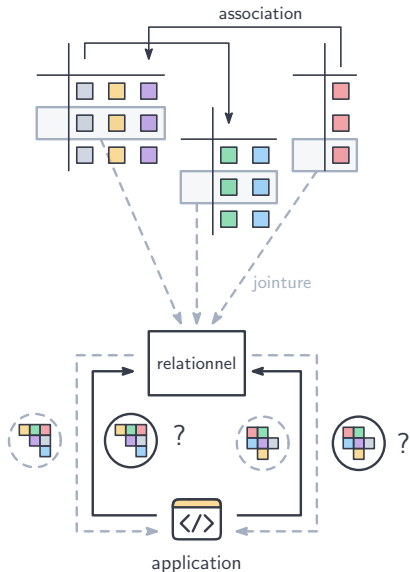
- **relationnel** : possibilité de *reconstituer* tous types d'agrégats au prix de *jointures* parfois coûteuses ...
- **noSQL** : extrêmement efficace pour récupérer les agrégats déjà stockés, mais moins pour en construire d'autres ...

Du coup :

- les BD relationnelles peuvent servir plein d'applications différentes
- les BD noSQL sont très adaptées à des besoins spécifiques

i Remarque : On peut choisir de mettre en place plusieurs moteurs, mais ça implique de la *redondance*, et de gérer l'*échange* et la *transformation* de ces données.

Schématiquement



Agrégats et données semi-structurées

? Question : On a dit que les BD noSQL manipulaient des données *semi-structurées* ... mais de quoi s'agit-il ?

- format de données structurées : tables, respectent un format bien défini
- format de données *semi-structurées* : structures hiérarchiques de paires (champs, valeur)
- en pratique : *JSON*, *YAML*, *BSON*, ...

i Note : ces types de documents semi-structurés sont un moyen de stocker des *agrégats*

Le JSON

JSON : JavaScript Object Notation, première version sortie en 2002 (json.org)

JSON en quelques points :


- format de données issu de la représentation des objets en javascript
- permet de représenter de l'information de *manière hiérarchique*
- *très facile à utiliser* avec n'importe quel langage de programmation
- très utilisé dans les applications web, les web-services, et *les BD noSQL* !
- fichiers au format `.json`

i Remarque : Permet entre autres de *sérialiser* des objets : sauvegarder leur état à un moment donné dans un fichier textuel

Un exemple

```
{  
  "nom" : "Vian",  
  "prenom" : "Boris",  
  "naissance" : 1920,  
  "deces" : 1959,  
  "livres" : [  
    {"titre" : "L'herbe rouge", "annee": 1950, "film": true},  
    {"titre" : "L'arrache-coeur", "annee": 1953, "film": false},  
    {"titre" : "L'ecume des jours", "annee" : 1947, "film": true}  
  ]  
}
```

Éléments de syntaxe

 **Note** : JSON utilise deux structures : *objets* et *tableaux*


 **Syntaxe** : un *objet* est une *collection de paires* "nom": valeur séparées par des virgules. Cette collection est déclarée entre accolades :

```
{  
  "nom1": valeur1,  
  "nom2": valeur2,  
  ...  
}
```

 **Syntaxe** : un *tableau* est une *liste ordonnées* de valeurs. La liste est déclarée entre crochets :

```
[  
  valeur1,  
  valeur2,  
  ...  
]
```

Imbrications

 **Question :** C'est quoi une « valeur », du coup ?

 **Réponse :** Ça peut être :

- une chaîne de caractères
- un nombre
- une des constantes `null`, `true`, `false`
- un *objet* ou un *tableau* !

 **Important :**

- on peut imbriquer des objets et des tableaux \implies structure *hiérarchique*
- *pas de schéma prédéfini*, on peut mettre « n'importe quoi » dans une collection ou un tableau

Exemple

```
[
  {
    "type" : "livre",
    "titre" : "les champs magnetiques",
    "auteur.e" : [
      {"prenom": "Andre", "nom" : "Breton"},
      {"prenom": "Philippe", "nom" : "Soupault"}
    ],
    "annee" : 1920
  },
  {
    "type": "film",
    "titre": "Dune",
    "realisateur.ice": {"prenom": "Denis", "nom": "Villeneuve"},
    "annee": 2022,
    "budget": {"millions": 165, "devise": "dollars"}
  }
]
```


Exercice

⚙️ **Exercice** : le code JSON suivant est-il correct ?

```
[
  {
    "prenom": "Emie",
    "nom": "Molette",
    "contact": [
      "mail": "pouet[at]json.gouv",
      "telephone": null
    ]
  },
  {
    "nom": "Tongue",
    "prenom": "Tim",
    "date-naissance": "01/01/0001",
    "mail": ["timtongue[at]mail.fr"]
  }
]
```

Un comparatif plus visuel avec XML

i Note : XML est un autre format de données semi-structurées, plus ancien :

- *plus riche que le JSON*, mais ... beaucoup *moins léger* et *intuitif*!

```
{
  "nom" : "Vian",
  "prenom" : "Boris",
  "naissance" : 1920,
  "deces" : 1959,
  "livres" : [
    {
      "titre" : "L'herbe rouge",
      "annee" : 1950,
      "film": true
    }
  ]
}
```

```
<?xml version="1.0" encoding="
    UTF-8" ?>
<root>
  <nom>Vian</nom>
  <prenom>Boris</prenom>
  <naissance>1920</naissance>
  <deces>1959</deces>
  <livres>
    <titre>L'herbe rouge</titre>
    <annee>1950</annee>
    <film>true</film>
  </livres>
</root>
```

Retour au distribué

i Remarque : les systèmes noSQL sont des *systèmes distribués* dédiés à la gestion de grandes masses de données

Système distribué :

- système qui permet de coordonner de nombreuses machines
- communiquant par l'échange de messages
- deux architectures :
 - *maître-esclave* : une machine « maître » qui répartit les demandes sur des machines « esclaves »
 - *décentralisé* : toutes les machines jouent le « même rôle »

Pour les données distribuées :

- accès efficaces même avec des gros volumes de données
- tolérant aux pannes
- possibilité de montée en puissance par ajout de machines

Théorème CAP

Les trois *propriétés fondamentales* **CAP** des systèmes distribués :

- **C**onsistency (coherence) : tous les nœuds du système voient les mêmes données au même moment
- **A**vailability (disponibilité) : la perte d'un nœud n'empêche pas le système de fonctionner et servir l'intégralité des données
- **P**artition tolerance (résistance au partitionnement) : le système fonctionne même s'il est *partitionné* en plusieurs morceaux

! **Important** : En 2000, Brewer énonce le *Théorème CAP* : dans un *système distribué*, on peut valider *au plus 2 propriétés* parmi les 3.

i **Note** : Ou plutôt si le système est partitionné (et ça va arriver), il faudra choisir la cohérence ou la disponibilité

Le relationnel face au distribué

Dans la plupart des SGBDR :

- des *données liées entre elles* sont placées au *même endroit*
- quand le modèle contient beaucoup de liens, difficile de répartir les données

i Remarque : Dans un système distribué avec une BD relationnelle, le besoin **ACID** des transactions augmente le temps de réponse (besoin de consistance) !

! Attention : le consistency de ACID n'est pas le même que celui de CAP

noSQL et BASE

Pour être efficaces, les systèmes noSQL relâchent ACID au profit de BASE :
disponibilité > cohérence

BASE : **B**asic **A**vailability, **S**oft-state and **E**ventual consistency

Les propriétés **BASE** :

- **B**asic **A**vailability : le système garantit la disponibilité des données
- **S**oft-state : la base peut changer lors des mises à jour ou lors d'ajout/suppression de serveurs. La base NoSQL n'a pas à être cohérente à tout instant
- **E**ventual consistency : la cohérence des données n'est pas assurée à un moment donné, mais elle arrivera avec le temps

Comment les données se répartissent en noSQL ?

Réplication (plutôt maître-esclave) :

- *duplique des données* sur les machines
- pratique pour les lectures intensives (on peut les répartir sur les esclaves)

Sharding (plutôt décentralisé) :

- *partition des données* et répartition sur les noeuds du réseau
- améliore la mise à l'échelle horizontale (distribue les requêtes au bon endroit)

i Remarque :

- en noSQL, on utilise parfois le terme *sharding* pour sous-entendre que le système fait le partitionnement et la répartition de manière *automatique*
- on peut combiner sharding et réplication

Les 4 cavaliers du noSQL

Intro

Du hiérarchique au noSQL

Distribution et données semi-structurées

Les 4 cavaliers du noSQL

- généralités

- Clé-valeur

- Documents

- Colonnes

- Graphes

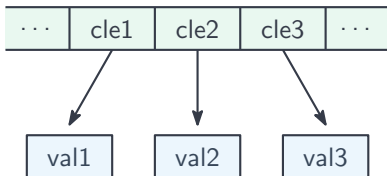
Pour finir

4 modèles de SGBD noSQL

1. *clé-valeur* : ensembles de couples (clé, valeur)
2. *document* : ensembles de couples (clé, document). Les documents sont semi-structurés (XML, JSON)
3. *colonne* : découpage « axé colonnes » d'une table, stocké
4. *graphe* : graphes dirigés où les nœuds représentent des entités avec des propriétés (clé, valeur) et les arcs des propriétés (potentiellement avec des

Clé-valeur : principe

💡 **Idée** : (intuition générale) à partir d'une clé *simple*, *atomique* et *précise*, on veut accéder à une information potentiellement *complexe*



En général :

- *clé* = chaîne de caractères
- *valeur* pouvant être *simple* (nombre, chaînes de caractères, ...), ou *complexe* (liste de valeurs, document JSON, ...)

i Remarque : similaire aux *dictionnaires* de Python (et d'autres)

Exemple de tables de données

livres	clé		prix	titre
	nolivre	noauteur		
	2574	3	10	Iris, c'est votre bleu
	3204	2	8	Le rhume
	2812	2	9	Mémoires trouvés dans une baignoire

clé étrangère

auteurs	clé		nom	prenom
	noauteur			
	1		Breton	André
	2		Lem	Stanislas
	3		Dreyfus	Ariane

- *clés* pour identifier les lignes
- *clés étrangères* pour référencer de l'auteur.ice d'un livre vers son identité

Une version clé-valeurs

clé	valeur
auteur-1-nom	André
auteur-1-prenom	Breton
auteur-2-nom	Lem
auteur-2-prenom	Stanislas
auteur-3-nom	Dreyfus
auteur-3-prenom	Ariane
livre-2574-titre	Iris, c'est votre bleu
livre-2574-auteur-3	Dreyfus
livre-2574-prix	10
...	...

lien auteur-livre



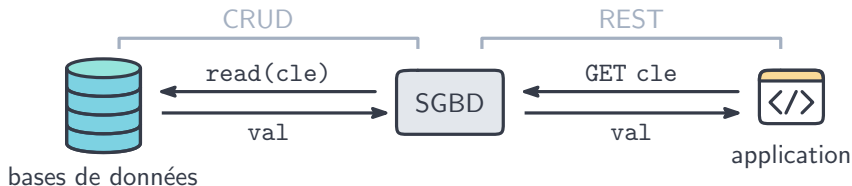
Remarque : plus *simple*, mais on perd la *logique* des données

Opérations

Pour utiliser une BD clé-valeurs, opérations **CRUD** :

- **Create** : `create(key, value)`, créer une nouvelle paire clé valeur
- **Read** : `read(key)`, accède à une valeur par sa clé
- **Update** : `update(key, value)`, met à jour une valeur
- **Delete** : `delete(key)`, supprime une valeur et sa clé

Proposent souvent une interface HTTP **REST** (**RE**presentational **S**tate **transfer** \Rightarrow accessibles depuis n'importe quel langage



Forces et faiblesses

Caractéristiques :

- le système de stockage *ne connaît pas la structure de l'information*
- accès très rapide grâce au hash des clés
- valeurs seulement *accessibles via les clés* (au contraire d'un SGBDR)

Forces :

- ✓ *performances très élevées* en lecture/écriture
- ✓ supporte très bien la *montée en charge* horizontale
- ✓ maintenance très simple

Faiblesses :

- ✗ *requête limité* (seulement sur les clés)
- ✗ tous les traitements sont à *faire par l'application client* (le système ne sait pas ce qu'il stocke)

Usages et SGBD

Usages :

- *grosses* dépôts de données avec besoins de requêtage *très simples*
- profils utilisateurs, données de capteurs, journaux d'évènements ...

Exemples (*clé*, *valeur*) :

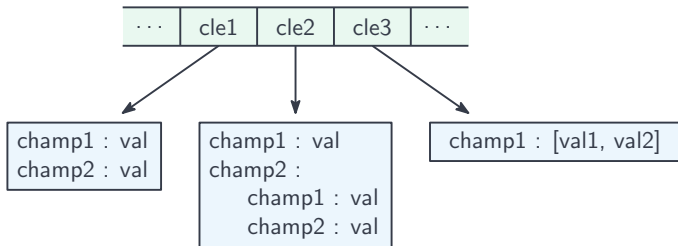
- réseau social : à partir de l'*identifiant* d'un.e utilisateur.ice, retrouver la *liste de ses ami.es*
- librairie : l'*ISBN* d'un livre donne accès à toutes ses *informations*

SGBD clés-valeurs :

- essentiellement *Redis*, memcached, Riak, ...

Documents : principe

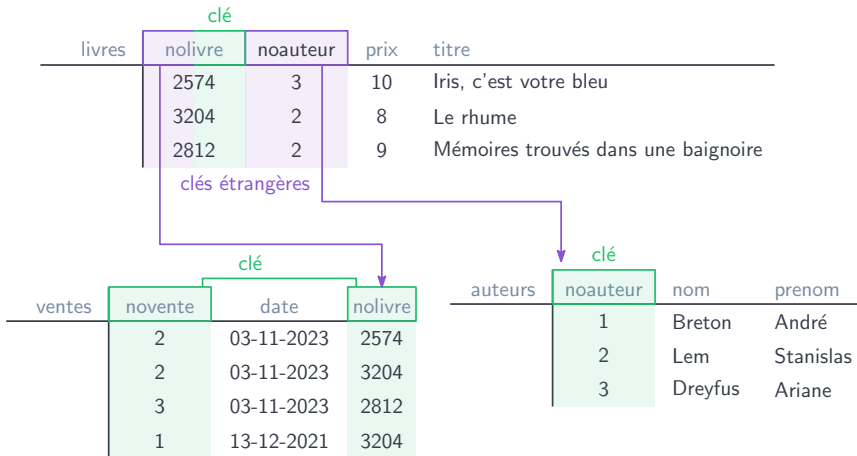
💡 Idée : (intuition générale) les BD noSQL orientées documents sont basées sur le modèle (clé, valeur) où la valeur est un *document* en format *semi-structuré* (ex : JSON ou XML)



i Note :

- suivant le SGBD, possibilité de regrouper avec une structure similaire dans des *collections*
- parallèle avec le relationnel : collection \simeq table, document \simeq ligne

Retour de l'exemple



Une version document

collection ventes

clé (OID)	document
145976258	<pre>"novente" : 2 "date" : "03-11-2023" "livres" : ["nolivre" : 2574 "nolivre" : 3204]</pre>
965130478	<pre>"novente" : 1 "date" : "13-12-2021" "livres" : ["nolivre" : 3204]</pre>

collection livres

clé (OID)	document
751235741	<pre>"nolivre" : 2574 "titre" : "Iris, c'est votre bleu" "auteur" : { "nom" : "Dreyfus" "prenom" : "Ariane" }</pre>
574895877	<pre>"nolivre" : 3204 "titre" : "Le rhume" "auteur" : { "nom" : "Lem" "prenom" : "Stanislas" }</pre>



Remarque : on aurait aussi pu faire une seule collection ventes en y *imbriquant* les livres

Caractéristiques et opérations

Le modèle est *plus structuré* qu'une BD clés-valeurs :

- le système « sait » que la structure des documents est hiérarchique
- possible de requêter les valeurs directement dans les documents

Mais *moins* qu'une BD relationnelle :

- données (documents) très hétérogènes, composites
- pas de schémas de document à définir au préalable
- à priori pas ou peu de liens logiques entre les documents

Opérations CRUD + REST

Forces et faiblesses

Forces :

- ✓ modèle simple mais puissant grâce à la richesse des documents semi-structurés
- ✓ supporte très bien la mise à l'échelle (distribution)
- ✓ pas de maintenance pour ajouter/supprimer des champs

Faiblesses :

- ✗ modèle hiérarchique inadapté pour les données interdépendantes
- ✗ requêtes complexes (avec « jointure ») moins efficaces que dans les SGBDR

Usages et SGBD

Usages :

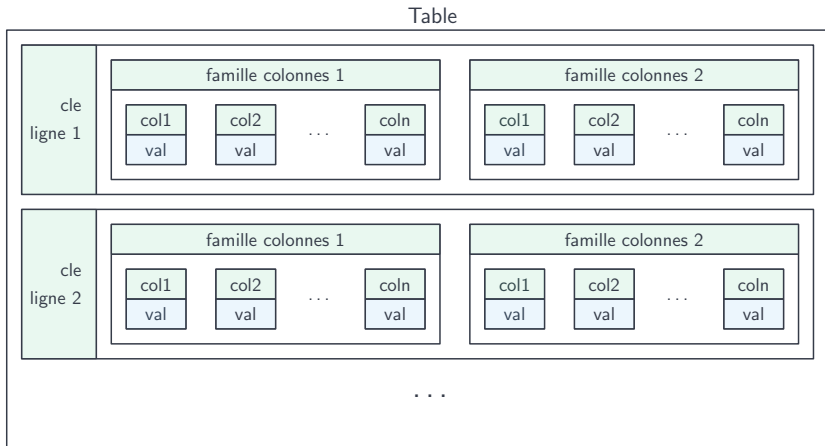
- catalogues de produits,
- enregistrements d'évènements,
- outils de gestion de contenu,
- analyse en temps réel,
- alternative au relationnel si la modélisation relationnelle avait engendrée des problèmes de partitionnement et de réplication

SGBD orientés documents :

- *MongoDB*, Couchbase, DynamoDB, (\pm) Elasticsearch, ...

Colonnes : principe

💡 **Idée :** (intuition générale) proche d'une table, sauf que l'unité de base est la *colonne* et pas la *ligne*



Colonnes : principe (suite)

! Important : on présente ici le modèle type BigTable (le premier système en colonnes), il y a parfois des petites variations

Autrement dit :

- *colonne* = paire (nom, valeur)
- *famille de colonnes* = liste de colonnes
- les familles sont stockées dans une *ligne* indexée par une *clé*
- un ensemble de lignes forment une *table*

i Remarque :

- en pratique, une colonne est une paire

$$(nom, [(val1, tmp1), (val2, tmp2), \dots, (valn, tmpn)])$$

où les *tmp* sont des *timestamp* \implies *versioning*

- famille de colonnes \simeq groupe de colonnes *sémantiquement reliées* (bouts d'une adresse, informations d'une personne, ...)

Exemple de BD

Utilisateur	clé	infos perso		contact	
	id	nom	prénom	telephone	mail
	1	Nette	Marie	07 00 00 00 00	marie.nette@gmail.com
	2	Fenouil	Emile	NULL	mille.courgettes@legume.fr
	3	Taie	NULL	NULL	clara.taie666@impots.gouv.fr

i Remarque : en relationnel classique, on renseigne *l'absence d'une valeur* par NULL

Une version colonnes

Table Utilisateurs

1	<table><tr><th colspan="2">persos</th></tr><tr><td>nom</td><td>prénom</td></tr><tr><td>Nette</td><td>val</td></tr></table>	persos		nom	prénom	Nette	val	<table><tr><th colspan="2">contact</th></tr><tr><td>telephone</td><td>mail</td></tr><tr><td>07 00 00 00 00</td><td>marie.nette@gmail.com</td></tr></table>	contact		telephone	mail	07 00 00 00 00	marie.nette@gmail.com
persos														
nom	prénom													
Nette	val													
contact														
telephone	mail													
07 00 00 00 00	marie.nette@gmail.com													
2	<table><tr><th colspan="2">persos</th></tr><tr><td>nom</td><td>prénom</td></tr><tr><td>Fenouil</td><td>Emile</td></tr></table>	persos		nom	prénom	Fenouil	Emile	<table><tr><th colspan="2">contact</th></tr><tr><td></td><td>mail</td></tr><tr><td></td><td>mille.courgettes@legume.fr</td></tr></table>	contact			mail		mille.courgettes@legume.fr
persos														
nom	prénom													
Fenouil	Emile													
contact														
	mail													
	mille.courgettes@legume.fr													
3	<table><tr><th colspan="2">persos</th></tr><tr><td>nom</td><td></td></tr><tr><td>Taie</td><td></td></tr></table>	persos		nom		Taie		<table><tr><th colspan="2">contact</th></tr><tr><td></td><td>mail</td></tr><tr><td></td><td>clara.taie666@impots.gouv.fr</td></tr></table>	contact			mail		clara.taie666@impots.gouv.fr
persos														
nom														
Taie														
contact														
	mail													
	clara.taie666@impots.gouv.fr													

i Remarque : les valeurs nulles *disparaissent*

Caractéristiques

- BD assez complexes à appréhender (conception, exploitation)
- très utilisées pour l'analyse de données et dans les traitements massifs
- elles offrent plus de flexibilité que les BD relationnelles :
 - les valeurs peuvent être quelconques
 - possibilité d'ajouter une colonne à n'importe quelle ligne dans n'importe quelle famille de colonnes
 - disparition des **NULL**

i Remarque : On y retrouve l'idée que la donnée est « opaque » pour le système

- plus structuré que du clé-valeurs
- moins structuré que du document

Forces et faiblesses

Forces :

- ✓ bonne mise à l'échelle horizontale
- ✓ supporte des données tabulaires et des données clairsemées
- ✓ grâce à l'indexation lignes/colonnes, de bonnes performances en temps réel sur des requêtes adaptées à la structure des tables

Faiblesses :

- ✗ supporte mal les données interconnectées
- ✗ ajout de ligne coûteux
- ✗ pour des recherches efficaces, il faut modéliser la BD en fonction des requêtes à faire \implies requêtes pré-écrites

Usages et Logiciels

Usage :

- besoins de traiter d'énormes volumes de données de manière distribuée
- analyse de données, jeux de données scientifiques, data mining

Exemples :

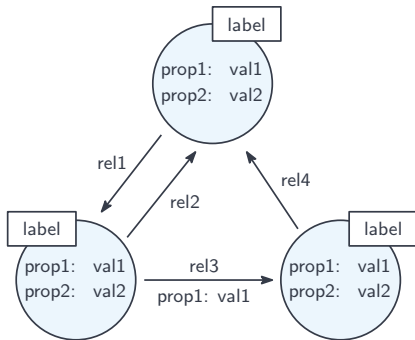
- Netflix : analyse de sa clientèle
- Ebay : optimisation de recherche
- Adobe : Business intelligence

Principaux SGBD colonnes :

- HBase, BigTable, ScyllaDB

Graphes : principe

💡 **Idée :** (intuition générale) organisation des données et leurs relations sous forme de *graphe dirigé*

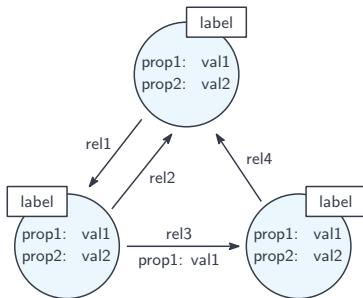


i Remarque : On utilise ici la sémantique *labeled property graph* qu'on retrouve dans Neo4J, mais il existe aussi une variante dite RDF

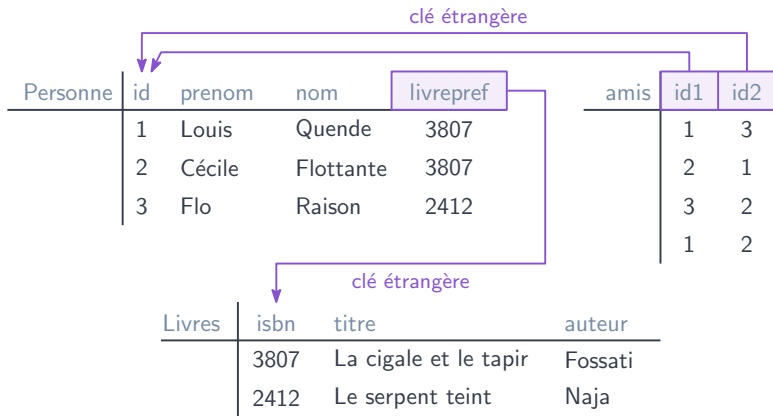
Graphes : principes (suite)

Un labeled property graph contient des *noeuds* et des *relations*

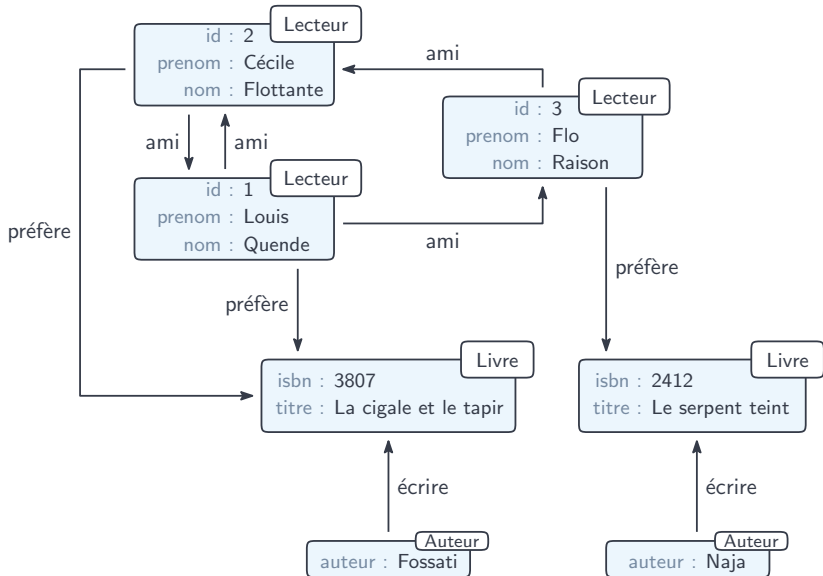
- *noeud* :
 - contient une liste de *propriétés*, i.e. des paires (clé, valeur)
 - peut avoir un ou plusieurs *labels*
- *relations* :
 - *arc entre deux noeuds*
 - avec un nom et possiblement des propriétés



Exemple de BD



Une version Graphes



Forces et Faiblesses

Forces :

- ✓ particulièrement efficace pour traiter des problèmes de « réseaux » (réseaux sociaux, cartographies, plus-court-chemin, ...), plus que des BD relationnelles
- ✓ modèle très expressif, très puissant
- ✓ modèles d'interrogation bien établis et performants (SPARQL, CYPHER)

Faiblesses :

- ✗ capacités de mises à l'échelle moins fortes
- ✗ difficile de répartir les données

Usages et logiciels

Usages :

- moteurs de recommandation,
- données spatiales, cartographiques,
- réseaux sociaux, réseaux de transports, ...

Principaux SGBD graphes :

- *Neo4J*, ArangoDB, Memgraph

noSQL en général

Modèle :

- non relationnel, *pas de schéma* pour les données (ou schéma dynamique)
- données *complexes* ou *semi-structurées* (hiérarchiques)

Stockage distribué :

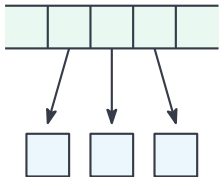
- *partitionnement* horizontal des données sur plusieurs nœuds
- *réplication* des données sur plusieurs nœuds

Propriétés et usages :

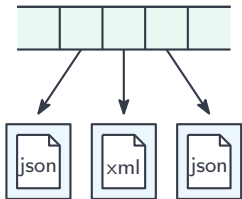
- *Théorème* de CAP : privilégient la disponibilité **A** sur la cohérence **C**
- compromis de ACID pour plus de scalabilité : *modèle* BASE
- peu de gestion de transactions
- mode d'utilisation : *peu d'écritures, beaucoup de lectures*

les 4 grands modèles noSQL

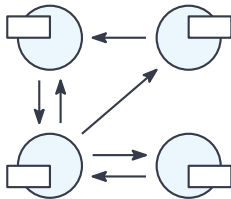
clé-valeur



documents



graphes



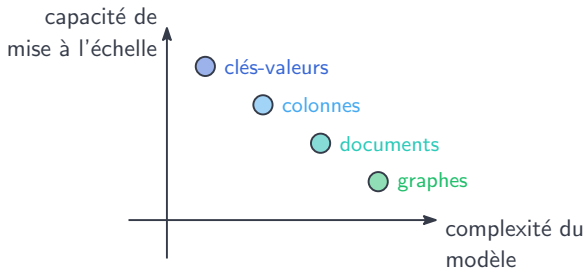
colonnes



Que choisir ?

Quelques questions à se poser

- Quels types de données sont à traiter ? (structurées, semi-structurées, complexes)
- Comment les applications vont utiliser ces données ?
- Quelles sont les fréquences de lecture/écriture ? (OLAP/OLTP)
- Complexité des requêtes ?
- Prévisions d'augmentation du volume de données ? (système distribué)



Quelques soucis

Conception, modélisation d'une BD noSQL :

- en essence, tout repose sur des paires (clé, valeurs)
- accès seulement par des clés
- les valeurs peuvent être complexes
- \Rightarrow *pas encore de modèles/méthodologie éprouvée de conception*

Autres problèmes :

- *Complexité des traitements* : pas de langage de requêtage puissant comme SQL (sauf BD graphes)
- *relâchement de la cohérence (ACID)* peut être critique pour certaines applications
- (personnel) au contraire du modèle relationnel, pas toujours de théorie mathématique sous-jacente