

Bases de données noSQL

Polytech Marseille

Simon Vilmin

simon.vilmin@univ-amu.fr

2024 - 2025

Plan du jour

Introduction

Un peu d'histoire : 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 modèles noSQL

- généralités

- Clé-valeur

- Colonnes

- Documents

- Graphes

Pour finir

À propos du cours

Contenu :

- quelques CM sur les notions principales
- TP de Powershell pour maîtriser les bases de la ligne de commande
- TP d'implémentation d'une BD noSQL très simple
- TP sur pymongo et des données réelles

Évaluation via un devoir maison :

- des questions sur les concepts généraux des bases de données noSQL
- un exercice de Powershell
- des questions d'analyse de données avec mongoDB et pymongo
- résumé d'un article de recherche

Ressources : voir [AMeTICE](#)

Sources principales

Cours d'autres enseignants :

- Farouk Toumani, [page perso](#)
- Bernard Espinasse, [lien cours](#)
- Philippe Declercq, [lien cours](#)

Cours en ligne :

- [OpenClassroom](#) (qu'on utilisera en TP)
- [b3dpedia.fr](#)

Livres :

- « Designing Data-Intensive Applications », M. Kleppmann, 2017, O'Reilly
- « Graph Databases », I. Robinson, J. Webber, E. Eifrem, 2015, O'Reilly
- « NoSQL Distilled », P. Sadalage, M. Fowler, 2013, Pearson
- « Les bases de données NoSQL », R. Bruchez, 2015, Eyrolles



Remarque : aussi des *articles scientifiques* référencés en temps voulu

Un peu d'histoire : du hiérarchique au noSQL

Introduction

Un peu d'histoire : du hiérarchique au noSQL

- Retour sur quelques SGBD

- Le modèle relationnel


- Arrivée des modèles noSQL

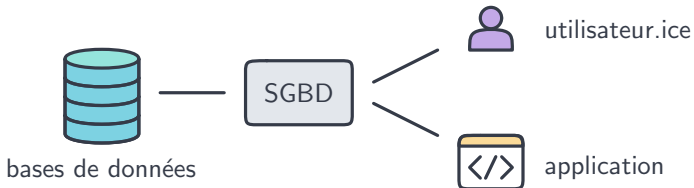
Distribution et données semi-structurées

Les modèles 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

423 systems in ranking, October 2024

Rank			DBMS	Database Model	Score		
Oct 2024	Sep 2024	Oct 2023			Oct 2024	Sep 2024	Oct 2023
1.	1.	1.	Oracle +	Relational, Multi-model ⓘ	1309.45	+22.85	+48.03
2.	2.	2.	MySQL +	Relational, Multi-model ⓘ	1022.76	-6.73	-110.56
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model ⓘ	802.09	-5.67	-94.79
4.	4.	4.	PostgreSQL +	Relational, Multi-model ⓘ	652.16	+7.80	+13.34
5.	5.	5.	MongoDB +	Document, Multi-model ⓘ	405.21	-5.02	-26.21
6.	6.	6.	Redis +	Key-value, Multi-model ⓘ	149.63	+0.20	-13.33
7.	7.	↑ 11.	Snowflake +	Relational	140.60	+6.88	+17.36
8.	8.	↓ 7.	Elasticsearch	Multi-model ⓘ	131.85	+3.06	-5.30
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model ⓘ	122.77	-0.28	-12.10
10.	10.	↓ 9.	SQLite	Relational	101.91	-1.43	-23.23

Figure – Source : <https://db-engines.com/en/ranking>

Les SGBD utilisés aujourd'hui

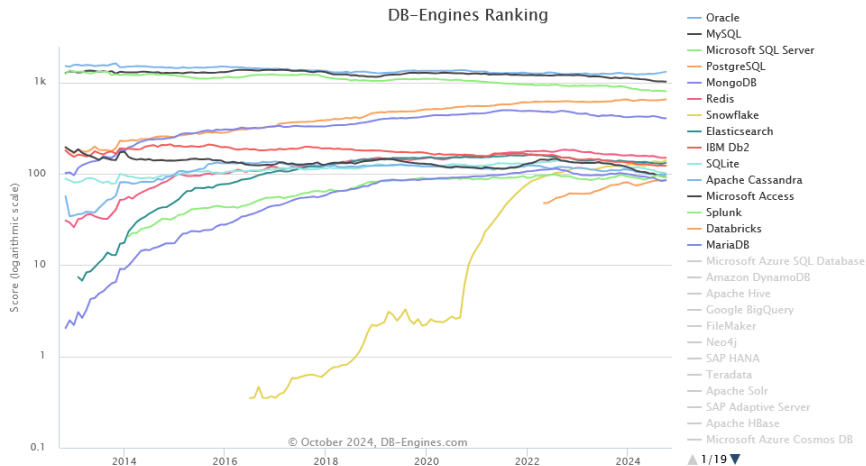
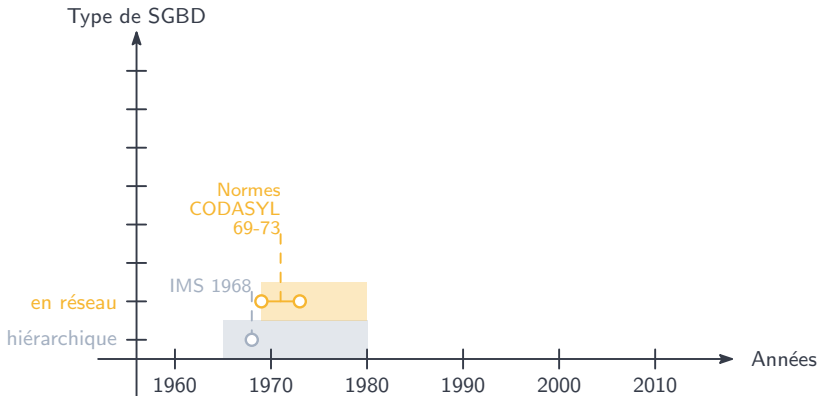


Figure – Source : https://db-engines.com/en/ranking_trend

Une histoire partielle des SGBD



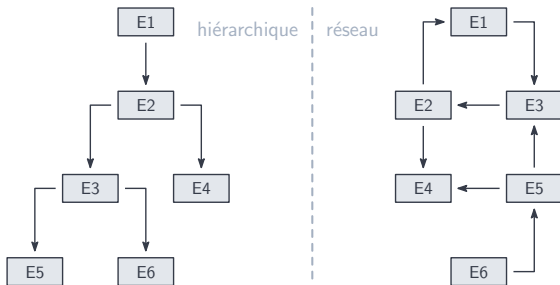
Sources supplémentaires :

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

Modèle hiérarchique, modèle en réseau

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

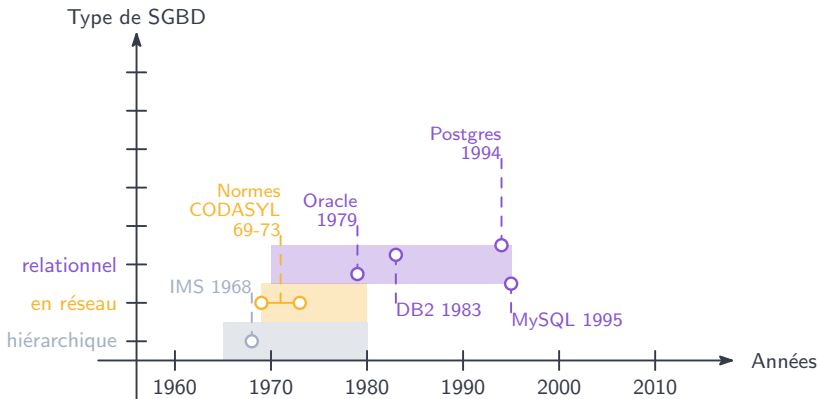
- *hiérarchique* : entités/données sous forme d'arbre (IMS, pour Apollo)
- *en réseau* : entités/données connectées en graphe (CODASYL)



i Remarque :

- *pas d'indépendance* physique/logique
- langage de requêtes *impératif*

Une histoire partielle des SGBD



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 ?

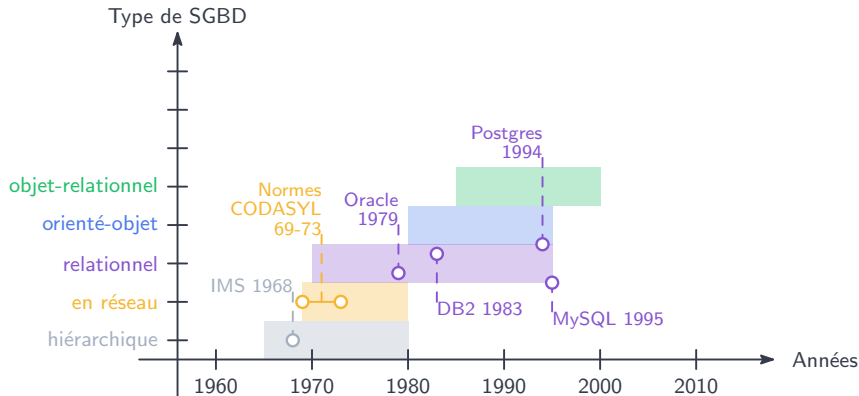
 **Définition** : une *transaction* est une séquence d'opérations de lecture ou de mise à jour sur une base de données.

 **Important** : Les transactions doivent préserver la *consistance* 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 consistant à un autre état cohérent
- **I**solation : indépendance des transactions concurrentes
- **D**urability : les modifications sont enregistrées dans la BD

Une histoire partielle des SGBD



i Remarque : le relationnel c'est super mais *défaut d'impédance*, i.e., difficile de transformer une modélisation du domaine ou d'un logiciel en modélisation relationnelle

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 :

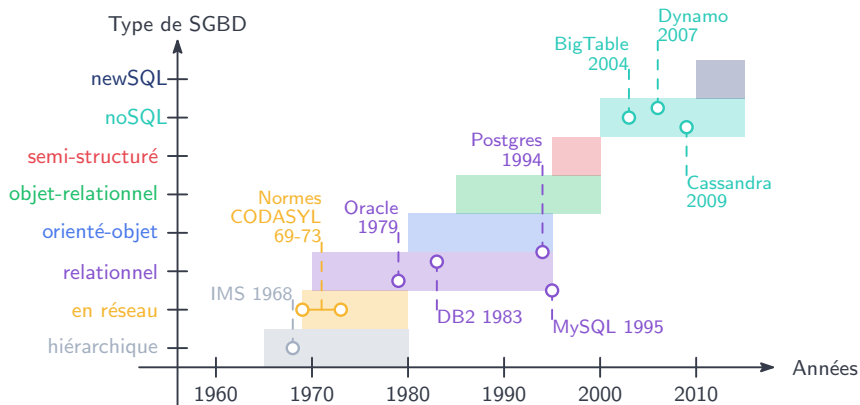
- sont adaptées à des données bien structurées
- gèrent *très bien* les traitements **OLTP** (**O**nLine **T**ransactional **P**rocessing)
gestion de ressources, lectures, écritures, ...
- *un peu moins bien* les traitements **OLAP** (**O**nLine **A**nalytical **P**rocessing)
statistiques, analyse de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

Une histoire partielle des SGBD



i Remarque : On arrive au bout !

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 ?

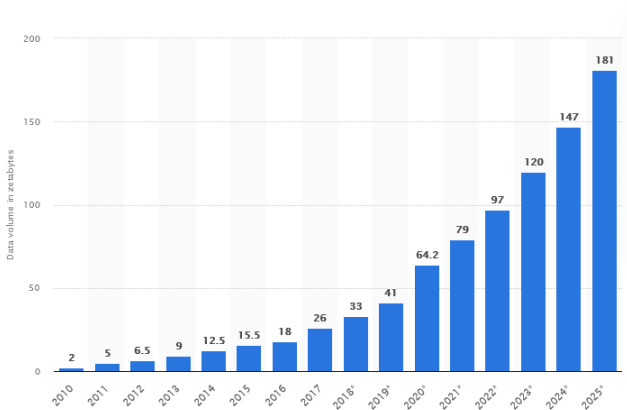
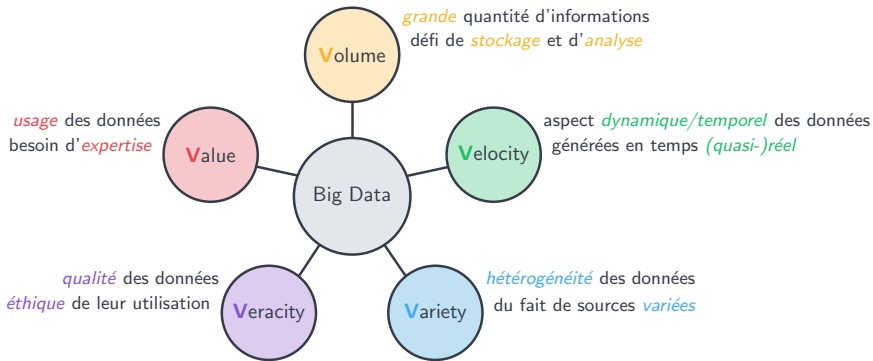


Figure – volume de données générées chaque année, (source : [statista](#), 2021)

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

Caractérisation par les 5V



! Important : « Big Data » peut se traduire par *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 = fini 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 fait débat. Il est parfois associé à « not only SQL », sans consensus.



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

Introduction

Un peu d'histoire : du hiérarchique au noSQL

Distribution et données semi-structurées

- Données semi-structurées

- Distribution

Les modèles noSQL

Pour finir

Centralité de la donnée vs. agrégats

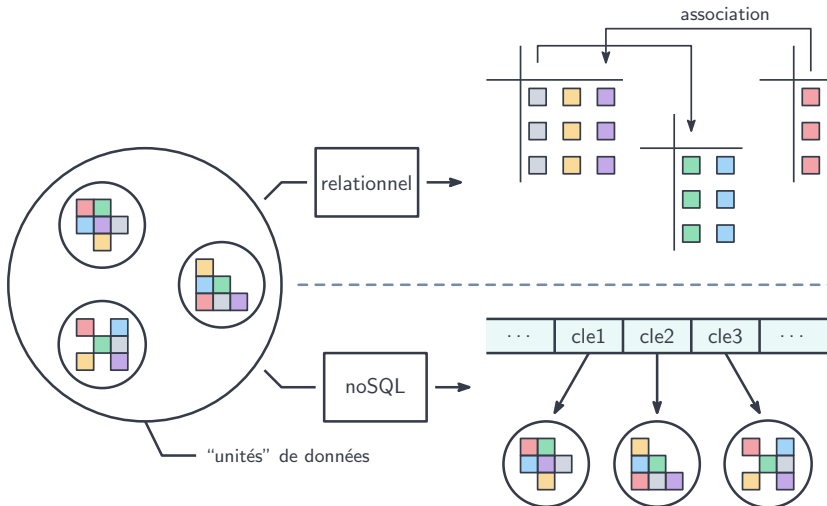
Relationnel : les données sont « *découpées* » et réparties dans des tables. Ce découpage rend une BD relationnelle accessible à une *grande variété d'applications différentes*. La donnée est *centralisée*

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

- **relationnel** :

- possibilité de *reconstituer* des agrégats avec des *jointures* parfois complexes
- une partie du traitement des données peut être fait côté SGBD
- normalisation et absence de redondance

- **noSQL** :

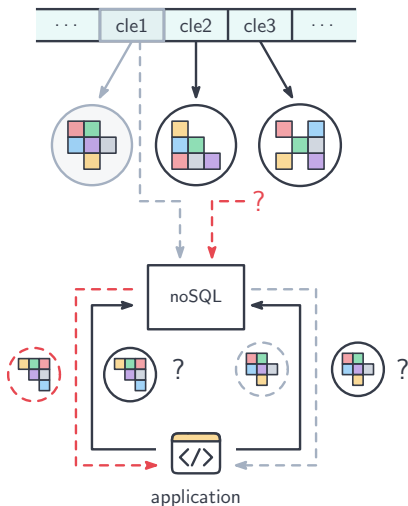
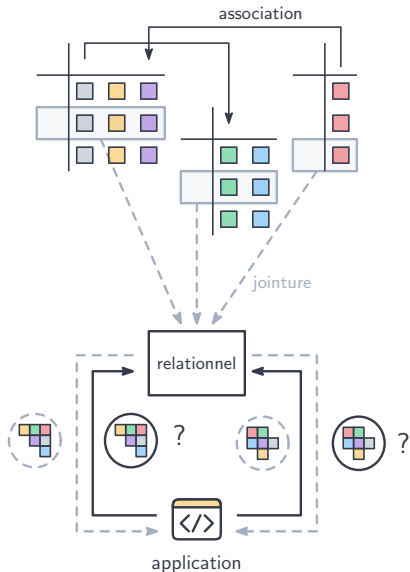
- extrêmement efficace pour récupérer les agrégats déjà stockés mais moins pour en construire d'autres
- le traitement des données est plutôt côté application
- redondance mais « unités d'information » conservées

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 utiliser 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`
- modélise des *relations 1..n*

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"}
  }
]
```

Un comparatif plus visuel avec XML

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

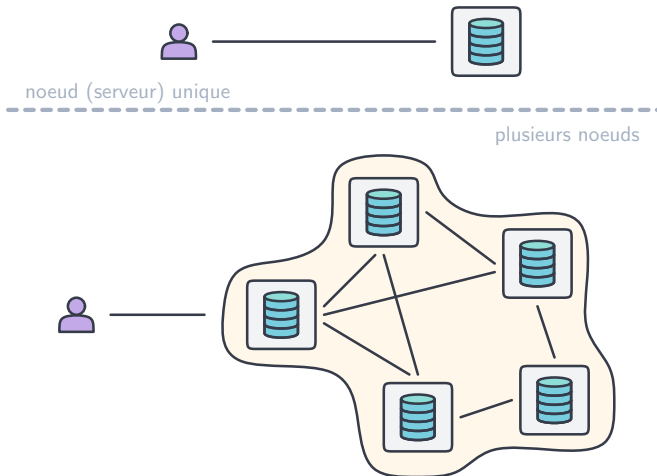
- *plus riche que le JSON*, mais ... beaucoup *moins léger* et *intuitif*!

```
{  
  "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 à la distribution

i Remarque : les systèmes noSQL sont pensés pour être *distribués*





Quelques questions

 **Définition** : un *système distribué* est un système qui permet de coordonner des machines communiquant par l'échange de messages

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

 **Question** : quel(s) rôles jouent les noeuds, quelle architecture ? Sur quel noeud sont exécutés les requêtes ?

 **Question** : comment sont réparties les données ?

Architecture : la question

? **Question** : quel(s) rôles jouent les noeuds, quelle architecture ? Sur quel noeud sont exécutés les requêtes ?

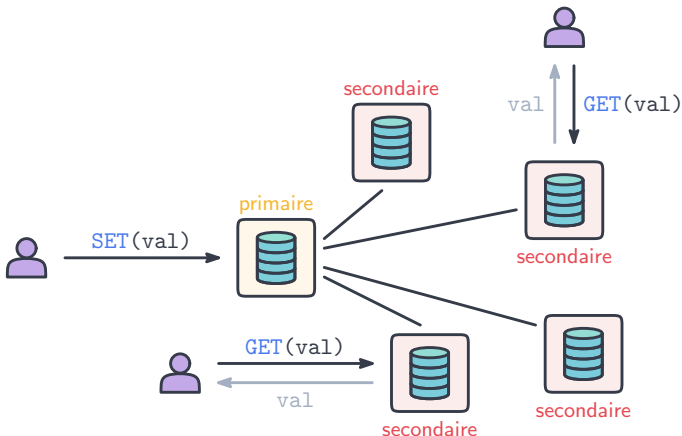
On s'intéresse à deux types de requêtes :

- les requêtes d'écriture, **SET**(data)
- les requêtes de lecture, **GET**(data)

Deux architectures :

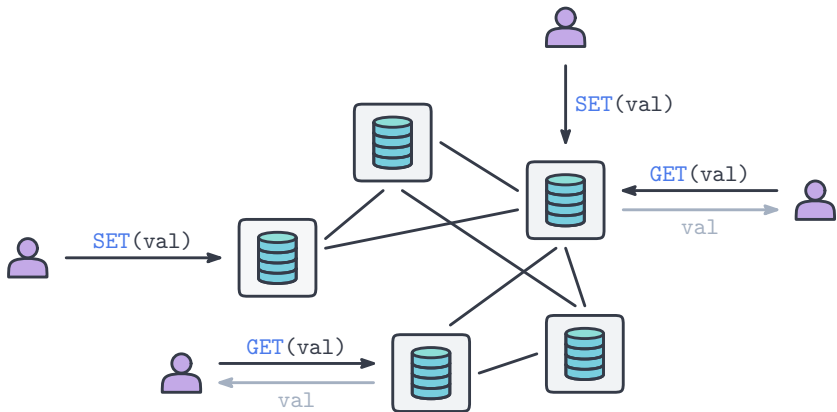
- *primaire-secondaire*, un noeud principal reçoit les écritures et les distribuent aux noeuds secondaires
- *décentralisé*, tous les noeuds ont le « même rôle »

Architecture primaire-secondaire



- un noeud *primaire* et des noeuds *secondaires*
- le primaire se charge des *requêtes d'écriture* et *envoie les données* aux secondaires (c.f. répliquions dans quelques slides)
- tous les noeuds peuvent effectuer des *requête de lecture*

Architecture décentralisée



- tous les noeuds ont le même rôle
- ils peuvent tous exécuter *lecture et écriture*

Répartition : la question

? Question : comment sont réparties les données ?

Réplication :

- *duplique des données* sur les machines
- pratique pour les lectures intensives

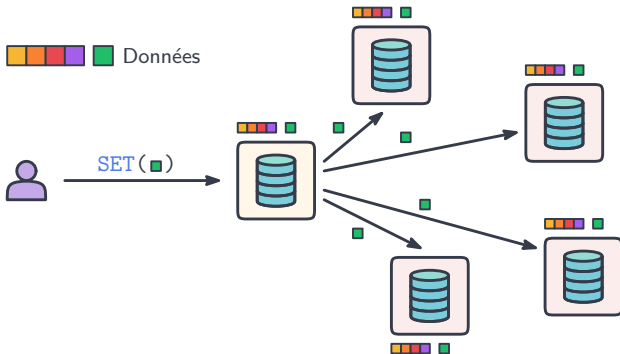
Sharding (plutôt sur du décentralisé) :

- *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 :

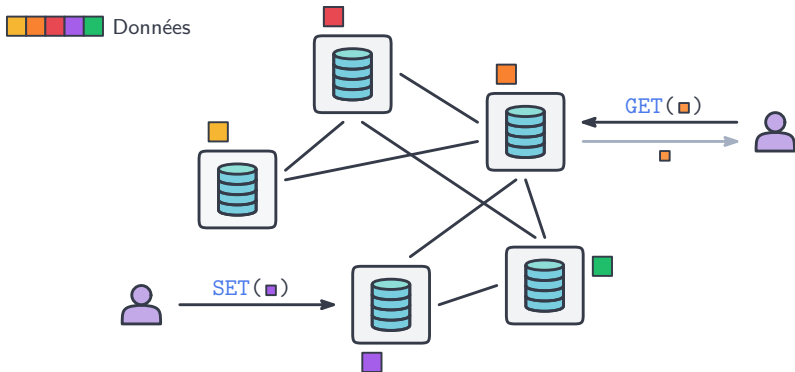
- en noSQL, on utilise parfois le terme *sharding* pour sous-entendre que le système fait le partitionnement et la répartition de manière *automatique*
- on peut combiner sharding et réplication

Réplication



- les données sont *dupliquées* sur plusieurs noeuds
- ici, le primaire envoie les mises à jour aux secondaires

Sharding



- les données sont *partitionnées* et *réparties* sur les noeuds
- les requêtes sont redirigées sur le noeud approprié

Difficultés du distribué

Des *propriétés* qu'on aimerait garantir dans un système distribué :

- *Consistance* : tous les nœuds du système voient les mêmes données au même moment (modulo le sharding et la réplication)
- *disponibilité* : le système doit être opérationnel en cas de panne de noeuds, de partitionnement du réseau, etc...

Des *problèmes* inévitables :

- *latence*, une modification sur un noeud doit être répercutée dans le système
⇒ requiert du temps et met en danger la consistance !
- *partitionnement*, par malheur le réseau est coupé en deux parties ou plus

! **Attention** : cette consistance n'est pas la même que celle des transactions ACID !

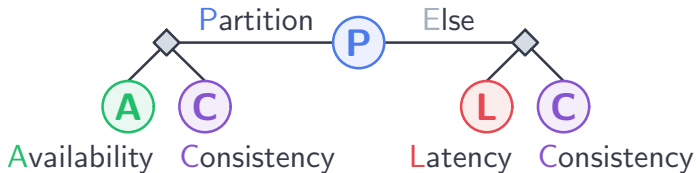
Théorème CAP

! **Important** : En 2000, Brewer énonce le *Théorème CAP* : dans un *système distribué*, si le système est **P**artitionné (et ça va arriver), il faudra choisir la **C**onsistance ou la disponibilité (**A**vailability)

i **Note** : le Théorème CAP a été sujet à moult interprétations peut-être trop simples : en l'absence de partition, quid du problème de latence ?

Alternative PACELC

✓ **Réponse** : l'alternative *PACELC* de Daniel Abadi intègre ce point




! **Attention** : Comme pour CAP, cette alternative est probablement trop simple par rapport à la réalité. C'est plutôt un guide.

noSQL : consistance à terme


Plusieurs niveaux de consistance :

1. *consistance faible* : aucune garantie sur la consistance des données
2. *consistance à terme* : les données seront consistantes *à un moment dans la vie de la BD* (populaire en noSQL)
3. *consistance forte* : tous les noeuds sont d'accord

 **Note** : parfois le terme **BASE** (par opposition à **ACID**) est utilisé pour caractériser les systèmes noSQL

- **Basically Available**, la BD est toujours disponible,
- **Soft state**, la BD peut être inconsistante à un moment donné
- **Eventually**, consistante mais elle sera consistante à termes

Mais cet acronyme est critiqué à cause de son manque de précision.

 **Astuce** : Le choix de quoi faire dépendra du *use case et des données* !

Les modèles noSQL

Introduction

Un peu d'histoire : du hiérarchique au noSQL

Distribution et données semi-structurées

Les modèles noSQL

- généralités

- Clé-valeur

- Colonnes

- Documents

- Graphes

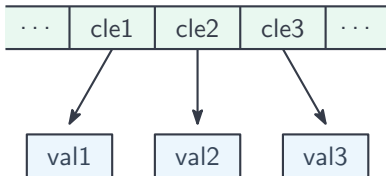
Pour finir

4 modèles de SGBD noSQL

1. *clé-valeur* : ensembles de couples (clé, valeur), i.e. *tableau associatif*
2. *colonne* : tableau associatif multi-dimensionnel
3. *document* : ensembles de couples (clé, document). Les documents sont semi-structurés (XML, JSON)
4. *graphe* : graphes dirigés où les nœuds représentent des entités avec des propriétés (clé, valeur) et les arcs des relations entre entités

Clé-valeur : principe

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



En général :

- *clé* = identifiant, nombre, chaîne de caractères, ...
- *valeur* vue comme un bloc *opaque*, i.e. un *blob*

i Remarque : structure de *tableau associatif* \simeq *dictionnaires* de Python

Exemple de données relationnelles

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

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

clé étrangère

- *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

Propriétés

i Remarque : le système ne « connaît pas » le contenu des valeurs. Tout le traitement des données est laissé à l'*application*

Jeu d'opérations très simple, **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

Quelques propriétés donc :

- lecture/écriture *très rapide*
- *perte* des dépendances entre les données
- *pas forcément de requêtes* sur plusieurs clés à la fois
- *très facile à partitionner* donc *très adapté* à la distribution

Usages et SGBD

Cas d'usages :

- gestion de profils utilisateurs, de session web, de panier (e-commerce),
- gros dépôts de données avec besoins de requêtage très simples

Exemples (*clé*, *valeur*) :

- à partir d'un *identifiant* d'un.e utilisateur.ice, retrouver les paramètres de sa *session* sur un site web
- l'*ISBN* d'un livre donne accès à toutes ses *informations*

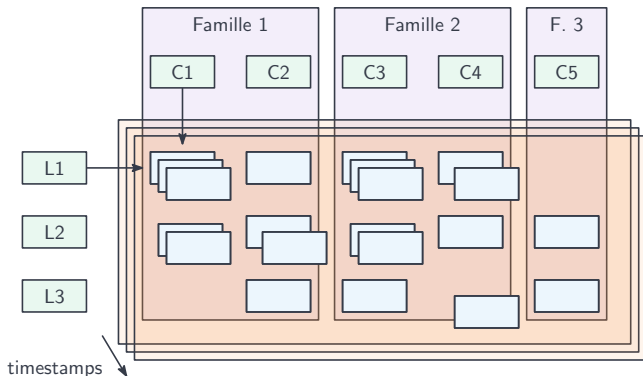
Moteurs clé-valeurs :

- *Redis*, Memcached, Riak, DynamoDB, ...

i Remarque : dans les faits, les moteurs clés-valeurs ajoutent de la structure dans les valeurs et se rapprochent des documents

Colonnes : principe (abstrait)

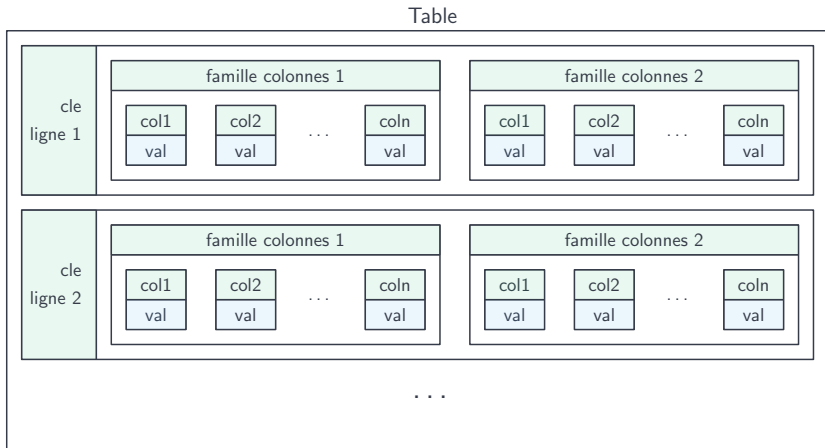
💡 **Idée** : un tableau associatif *multi-dimensionnel* qui permet de sélectionner les données par « *familles de colonnes* » (\simeq groupe d'attributs)



i Remarque : comme pour le modèle clé-valeur, les valeurs sont des *blob*

Exemple de BigTable (Google)

! Important : les autres systèmes peuvent présenter quelques variations



Exemple de BigTable (Google)

i Remarque : BigTable (2004) est l'un des premiers systèmes en colonnes

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 :

- les *timestamp* permettent le *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, on marque *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													



Remarque : les valeurs nulles *disparaissent*

Quelques caractéristiques

i Remarque : ces BD visent à *analyser des immenses stocks de données*

En conséquence, elles sont :

- adaptées à des données tabulaires
- par nature *pensées pour la distribution*

Pour un usage efficace :

- structurer la BD en amont pour l'adapter aux requêtes futures
⇒ retour partiel de l'idée de schéma
- *perte des relations* dans les données

Par rapport aux BD relationnelles :

- schéma relativement flexible (ajout/suppression de colonnes simple)
- les valeurs **NULL** disparaissent
- les valeurs sont des blob

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 :

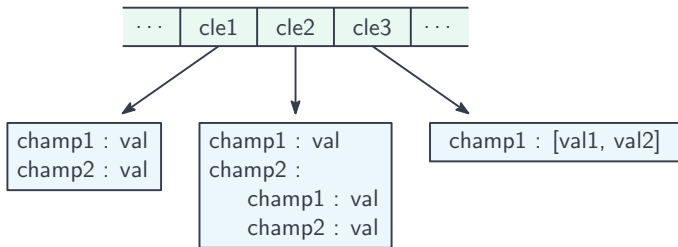
- Netflix : analyse de sa clientèle
- Ebay : optimisation de recherche
- Adobe : Business intelligence

Principaux SGBD colonnes :

- *Cassandra*, HBase, BigTable, ScyllaDB

Documents : principe

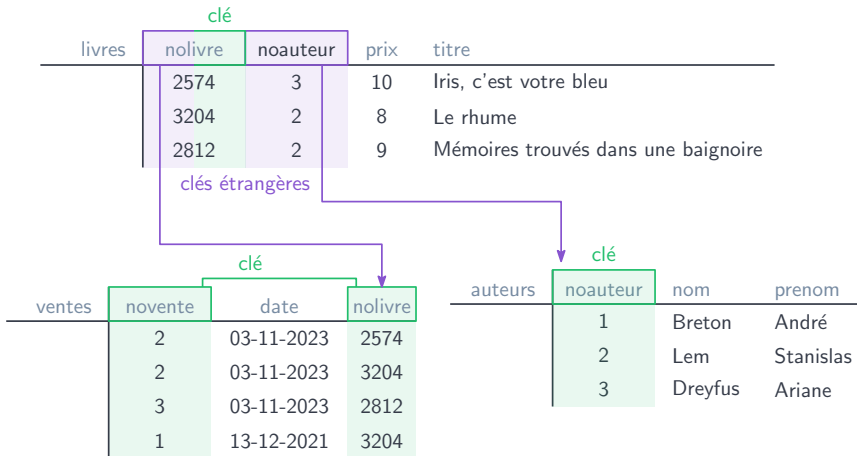
💡 **Idée :** modèle (clé, valeur) où la valeur est un *document* en format *semi-structuré* (ex : JSON ou XML)



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

Quelques caractéristiques

i Remarque : En fait, la structure hiérarchique des documents permet de capturer très efficacement les *relations 1..n* et de *spécialisation*

Modèle plus structuré que les BD clé-valeur et colonnes :

- le système « sait » que la structure des documents est hiérarchique
⇒ possible de requêter les valeurs directement dans les documents
- aucun schéma prédéfini (au contraire des BD colonnes), excepté la contrainte des documents semi-structurés
- reste *très bien adapté* à la distribution

Par contre :

- mécanismes de jointure moins efficaces que dans les SGBDR
- perte des relations entre les données

Usages et SGBD

Usages :

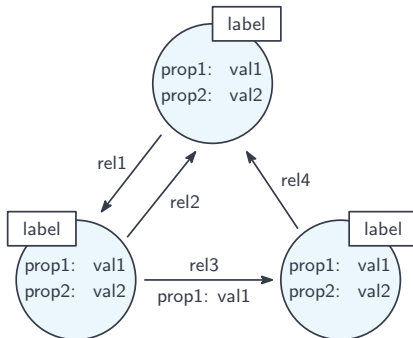
- catalogues de produits,
- enregistrements d'évènements,
- outils de gestion de contenu,
- analyse en temps réel,
- alternative au relationnel si la modélisation relationnelle avait engendrée des problèmes de partitionnement et de réplication

SGBD orientés documents :

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

Graphes : principe

💡 **Idée** : les données et leurs relations sont modélisées par un *graphe dirigé*

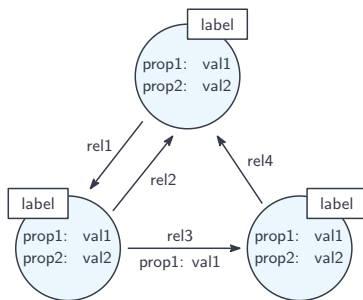


i Remarque : sémantique *labeled property graph* de Neo4J, mais il existe aussi une variante dite RDF

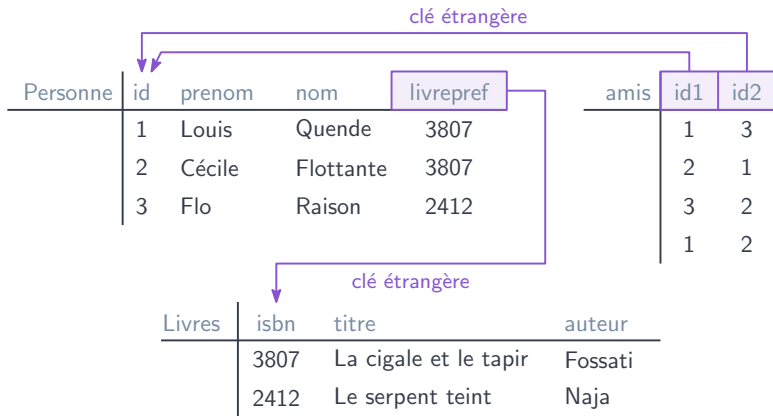
Graphes : principes (suite)

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

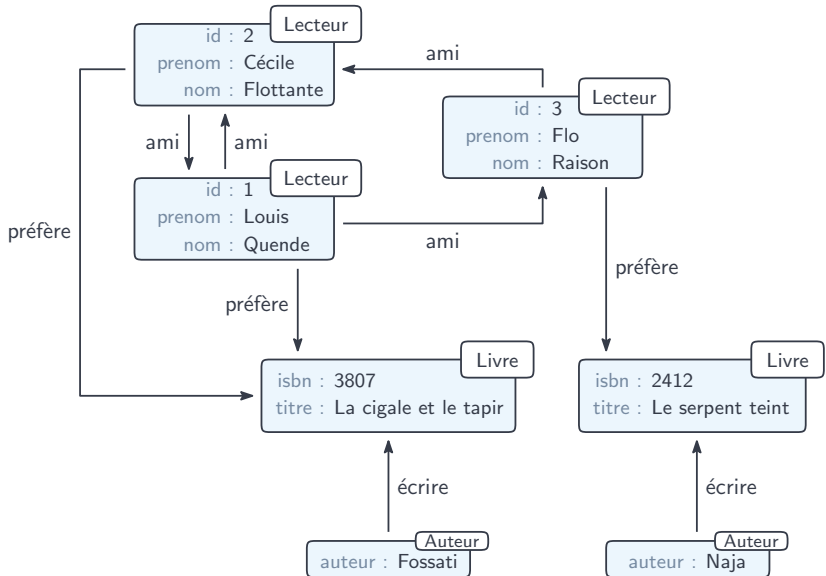
- *noeud* :
 - contient une liste de *propriétés*, i.e. des paires (clé, valeur)
 - peut avoir un ou plusieurs *labels*
- *relations* :
 - *arc entre deux noeuds*
 - avec un nom et possiblement des propriétés



Exemple de BD



Une version graphes

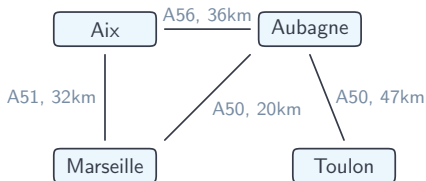
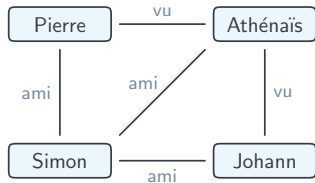


À quel problème répondent les graphes ?

Question : comment analyser et tirer parti des *liens* entre les entités ?

Exemples :

- connexions dans un réseau social
- références de sites web
- réseau routier



Réponse : les graphes modélisent *par nature* ces relations.

Graphes vs. les autres

❌ **Problème** : les modèles relationnels, clé-valeurs, documents ou colonnes ne capturent pas naturellement ces liens.

Pour modéliser des liens entre entités :

- nouvelles relations,
- références dans les données ou dans les contraintes (clés étrangères, id, etc)
- utilisation de jointures *coûteuses* sur des gros volumes de données

Propriétés des bases de données graphes

Quelques propriétés des graphes :

- *capturent les relations* dans les données et les *explorent efficacement*
- permettent des *requêtes complexes* (sélection de sous-graphe, parcours, etc), souvent via à un *langage déclaratif* (e.g. SPARQL, Cypher) à l'image de SQL
- sont *difficiles à partitionner* et donc à distribuer

Cas d'usages :

- systèmes de recommandation
- réseaux de distribution (par ex routiers), transport
- analyse de réseaux sociaux

Des exemples de SGBD graphes :

- *Neo4J*, ArangoDB, Memgraph, JanusGraph, NebulaGraph

Pour finir

Introduction

Un peu d'histoire : du hiérarchique au noSQL

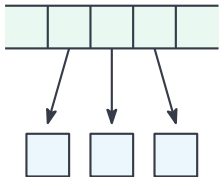
Distribution et données semi-structurées

Les modèles noSQL

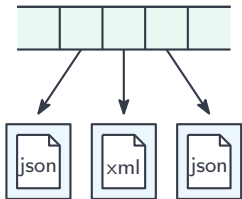
Pour finir

les 4 grands modèles noSQL

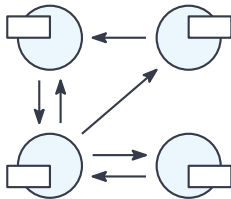
clé-valeur



documents



graphes



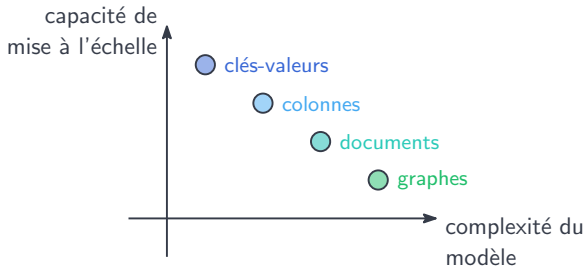
colonnes



Que choisir ?

Quelques questions à se poser

- Quels types de données sont à traiter ?
- 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