

Introduction bases de données non relationnelle (NoSQL)

Institut ICOM
MALIA

2024 – 2025

Juba AGOUN, PhD

Sommaire

- I. Qu'est-ce que NoSQL?
- II. Histoire des bases de données
- III. Big Data
- IV. Rappel sur les bases de données relationnelles
- V. Pourquoi le NoSQL (Limites du modèle relationnel)
- VI. Schémas de donnée dans les bases NoSQL
 - I. Catégories des modèles NoSQL
 - II. Représentation en arbre

Echauffement

Qu'est-ce que NoSQL?

- En 2009 le terme « NoSQL » a été inventé lors d'un événement open-source sur les bases de données distribuées.
- Catégorie de SGBD qui s'affranchis du modèle relationnel. Mouvance apparue par le biais des "grands du Web", popularisée en 2010.
 - Le terme est vague ou incorrect (certains moteurs NoSQL utilisent des variantes du langage SQL, par exemple Cassandra).

HOW TO WRITE A CV



NoSQL = Not Only Structured Query Language

Qu'est-ce que NoSQL?

- Désigne une famille de bases de données dont le style et la technologie varient considérablement.
- Partagent un trait commun:
 - Non-relationnelle
 - Diffèrent des système de gestion de base de données relationnelle (SGBDR) standard à lignes et à colonnes
- La majorité des bases de données NoSQL sont conçues pour gérer un type différent de problèmes d'échelle qui sont apparus avec le mouvement "big data".

Un peu d'histoire

Histoire des bases de données

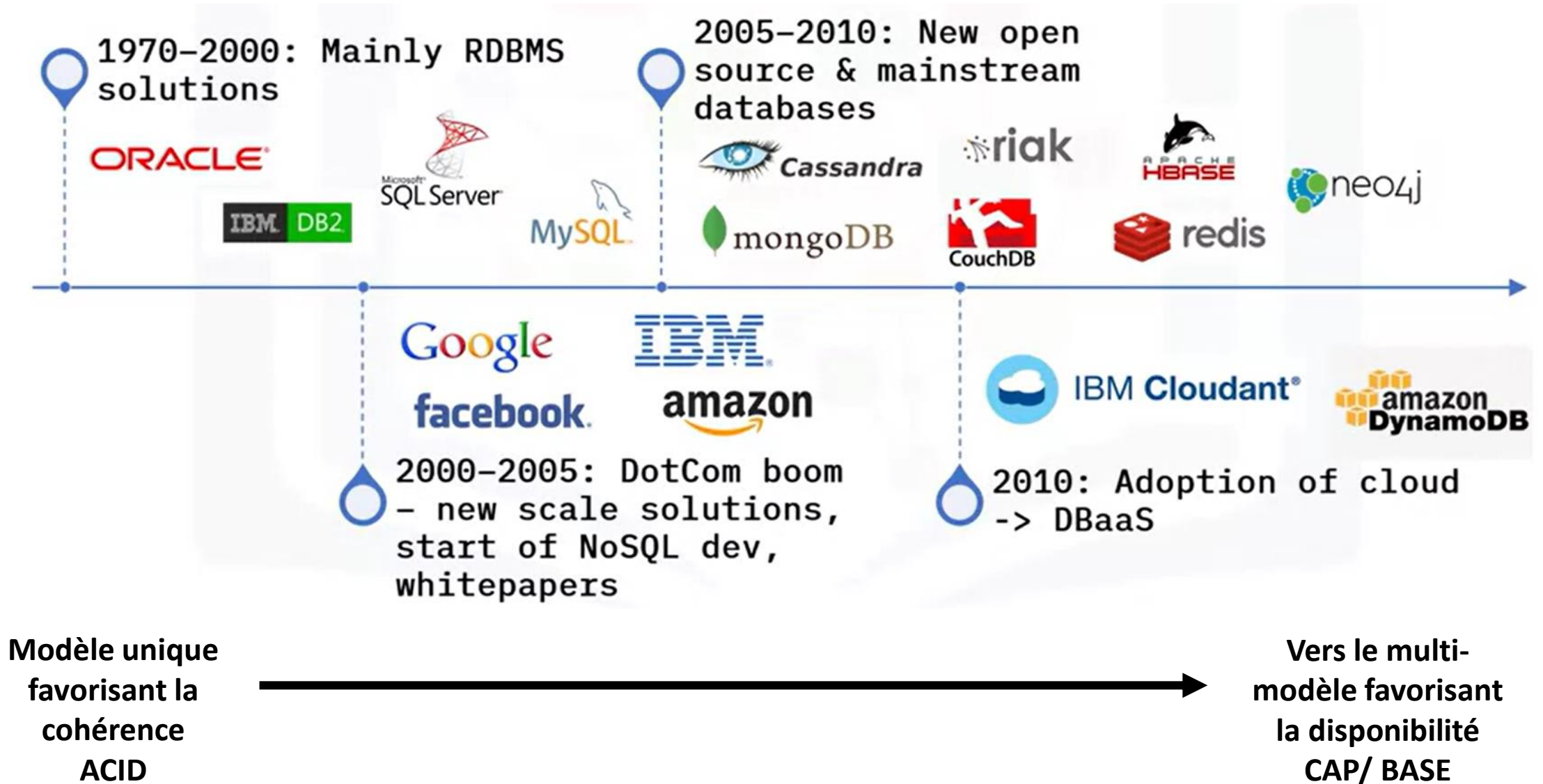
Depuis les années 1970, dominance du modèle relationnel

Émergence du web et du phénomène "Big Data" :

- ⦿ Grandes plateformes ou applications web gérant des millions d'utilisateur.trice.s et/ou d'objets
- ⦿ Explosion du volume de données à stocker et à traiter
- ⦿ Données de plus en plus complexes et hétérogènes

Limites des SGBD relationnels (utilisant le langage SQL) pour ces nouveaux usages, à cause du mécanisme de jointures, des contraintes d'intégrité et des transactions

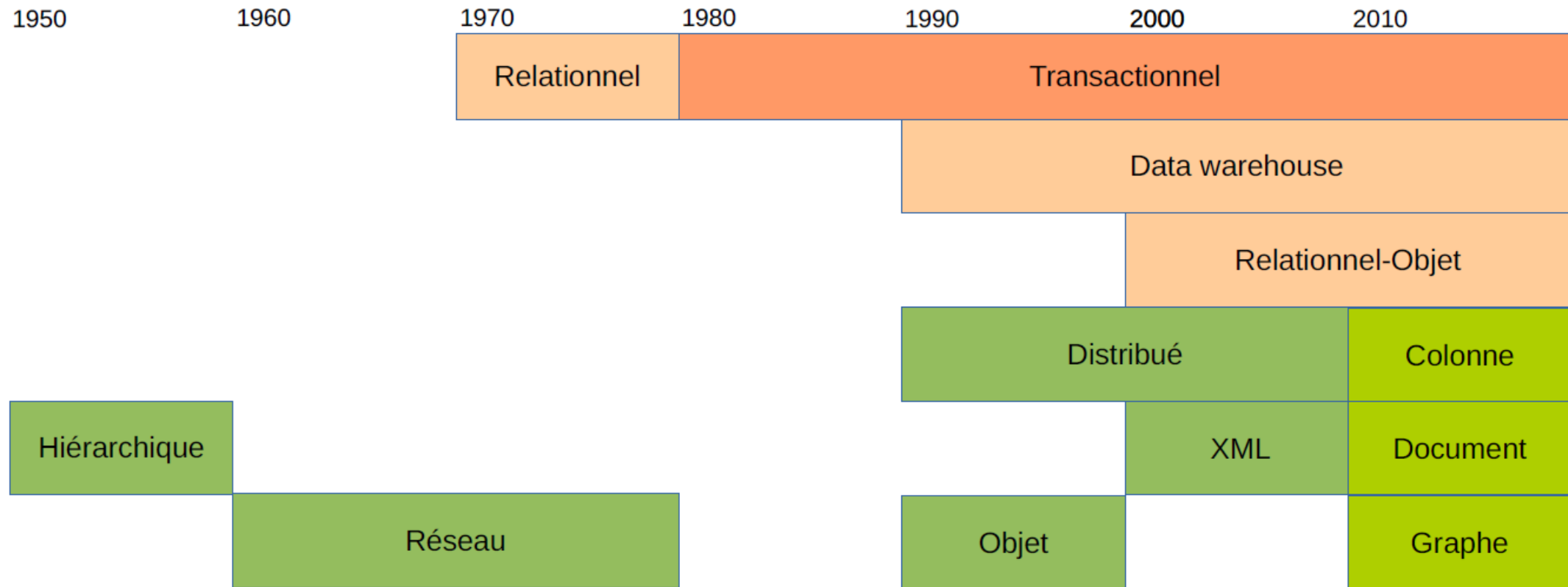
Histoire des bases de données



- 1998- Carlo Strozzi utilise le terme NoSQL pour sa base de données relationnelle légère et open-source.
- 2000- Lancement de la base de données graphique Neo4j
- 2004- Lancement de Google BigTable (orienté colonne)
- 2005- Lancement de CouchDB (orienté document)
- 2007- Publication d'un document de recherche sur Amazon Dynamo (Clé-Val)
- 2008- Facebook met en open source le projet Cassandra (clé-val)
- 2009- Le terme NoSQL a été réintroduit – Première version de MongoDB

DB-Engines Ranking classe les systèmes de gestion de bases de données en fonction de leur popularité: https://db-engines.com/en/ranking_trend

Vu d'ensemble historiquement



Les bases de données relationnelles

Base de données relationnels

Le modèle relationnel s'est largement imposé en recherche et en industrie dans le monde des bases de données.

L'objectif de Codd (père fondateur) est d'offrir un modèle :

- **Simple:** Se base le concept de domaine, relation et quelques opérateurs de dérivations
- **Rigoureux:** Une variante de la théorie des ensembles, donc des mathématiques discrètes
- **Complet:** Un système prouvé équivalent à la logique du premier ordre

Depuis 1972, ce modèle s'est érigé comme meilleur modèle pour structurer les données mais se fait détrôner par le modèle de conception Entité-Association.

Base de données relationnels

Notions essentielles :

➤ Univers U , Attributs A_1, \dots, A_n

Un univers U est un ensemble fini et non-vidé de noms, dits attributs

➤ Domaine $\text{Dom}(A)$ d'un attribut A

Le domaine d'un attribut A ($\text{Dom}(A)$) est l'ensemble des valeurs possibles associé à A

➤ Schéma d'une relation dont le nom est R .

Un schéma d'une relation dont le nom est R est un sous-ensemble non-vidé de l'univers U

$U = \{ \text{NomFilm}, \text{Realisateur}, \text{Acteur}, \text{Producteur}, \text{NomCinema}, \text{Horaire} \}$

$\text{Dom}(\text{NomFilm}) = \text{Dom}(\text{Realisateur}) = \text{Dom}(\text{Acteur}) = \text{Dom}(\text{Producteur}) = \text{Dom}(\text{NomCinema}) = \text{String}$

$\text{Dom}(\text{Horaire}) = \{h.m \mid h \in [0, \dots, 24], m \in [0, \dots, 59]\}$

Schéma de la relation

$\text{Film} = \{ \text{NomFilm}, \text{Realisateur}, \text{Acteur}, \text{Producteur} \}$

$\text{Projection} = \{ \text{NomFilm}, \text{NomCinema}, \text{Horaire} \}$

Base de données relationnels

Notions essentielles :

➤ *n*-uplet sur un ensemble *E* d'attributs

Un *n*-uplet *n* sur *S* est un ensemble $\{A_1 : v_1, \dots, A_n : v_n\}$ où $v_i \in \text{Dom}(A_i)$. / Avec $S = \{A_1, \dots, A_n\}$ le schéma d'une relation

➤ Relation (ou “table”) sur un schéma de relation

Une relation (table) *r* sur un schéma de relation *S* est un ensemble d'*n*-uplets sur *S*. On dit aussi : *S* est le schéma de *r*

➤ Schéma d'une BD

Un schéma *S* d'une base sur un univers *U* est un ensemble non-vide d'expressions de la forme $N(S)$ où *S* est un schéma de relation et *N* un nom de relation.

Un *n*-uplet possible sur le schéma de Projection :

<“Jugez — moi coupable”, “Gaumont Alesia 3”, 13.35i>

Film :	NomFilm	Réalisateur	Acteur	Producteur
	nf1	r1	a1	p1
	nf1	r1	a2	p1
	nf2	r2	a1	p2
	nf3	r2	a1	p2

Base de données relationnels

Notions essentielles :

- Une *base de données (relationnelle)* B sur un schéma de base S (avec univers U) est un ensemble de relations finies r_1, \dots, r_n où chaque r_i est associée à un nom de relation N_i et est telle que si $N_i(S) \in S$, alors r_i a S comme schéma.
- On peut aussi imposer des contraintes sur les données. Par exemple : les dépendances fonctionnelles, qui fixent, entre autres, les clés des relations
- Ces contraintes, dites d'intégrité, font aussi partie de la spécification du format des données de la base

Base de données relationnels

On notera en particulier qu'avec ces concepts :

- **Le schéma** : on peut exprimer des règles de cohérence a priori et déléguer leur contrôle au système
- **La normalisation** : on peut supprimer la redondance par un mécanisme de décomposition et retrouver l'information consolidée par les jointures
- **La transaction** : le système assure le maintien d'états cohérents au sein d'environnements concurrents et susceptibles de pannes

Avantage du modèle relationnel

Atomicité

Cohérence

Isolation

Durabilité

Propriété ACID

A : Une transaction représente une unité de travail qui est validée intégralement ou totalement annulée. C'est tout ou rien.

C : La transaction doit maintenir le système en cohérence par rapport à ses règles fonctionnelles. Durant l'exécution de la transaction, le système peut être temporairement incohérent, mais lorsque la transaction se termine, il doit être cohérent, soit dans un nouvel état si la transaction

I : Comme la transaction met temporairement les données qu'elle manipule dans un état incohérent, elle isole ces données des autres transactions de façon à ce qu'elle ne puisse pas lire des données en cours de modification.

D : Lorsque la transaction est validée, le nouvel état est durablement inscrit dans le système.

Limites du modèle relationnels

Imbrication

On ne peut pas imbriquer les informations :

En relationnel (“première forme normale”)

Rappel:

Une relation est dite de première forme normale, si tous les attributs de la relation contiennent une valeur atomique.

<i>Auteur</i>	<i>Titre</i>	<i>Langue</i>
Mozart	La Flûte Enchantée	Allemand
Mozart	Don Juan	Italien
Mozart	Les noces de Figaro	Italien
Bizet	Carmen	Français
Bizet	Djamileh	Français

Redondance sur l’auteur

Limites du modèle relationnels

Si on imbrique :

<i>Auteur</i>	<i>Opéra</i>	
Mozart	<i>Titre</i>	<i>Langue</i>
	La Flûte Enchantée	Allemand
	Don Juan	Italien
	Les noces de Figaro	Italien
Bizet	<i>Titre</i>	<i>Langue</i>
	Carmen	Français
	Djamileh	Français

On ne respect plus la norme “Première Forme Normale”, primordiale pour le modèle relationnel

Limites du modèle relationnels

Rigidité

**Le modèle relationnel une base a un nombre fixé de tables,
une table a un nombre fixe d'attributs etc**

Conçu pour les données structurées (i.e., tabulaires)

Limites du modèle relationnels

En résumé

Evolutivité

Quelle que soit la qualité de l'analyse, les besoins et donc les données évoluent et les schémas doivent intégrer ces changements. Mais le modèle relationnel est peu évolutif.

Efficacité

La masse des données à analyser et à gérer est de plus en plus importante et on voit apparaître : réseaux sociaux, Web, capteurs, ... d'où :

- informatique décisionnelle pour l'analyse de grands jeux de données
- Le phénomène Big data pour la gestion et l'analyse de masses de données.

Vers de nouveau modèle

Le big Data

Big Data : modélisation, stockage et analyse d'un ensemble de données volumineuses, croissantes et hétérogènes, dont l'exploitation permet la prise de décisions ou la découverte de nouvelles connaissances

Les "3V", caractéristiques du Big Data :

- **Volume** : (e.g., plusieurs zettaoctets/an générés sur le web)
- **Vélocité** : ou fréquence de génération des données, (e.g., 4000 To/jour pour Facebook en 2016 ou 7000 To/seconde pour le radiotélescope Square Kilometre Array)
- **Variété** : ou hétérogénéité (e.g., images, texte, données géo-démographiques)

Émergence du NoSQL

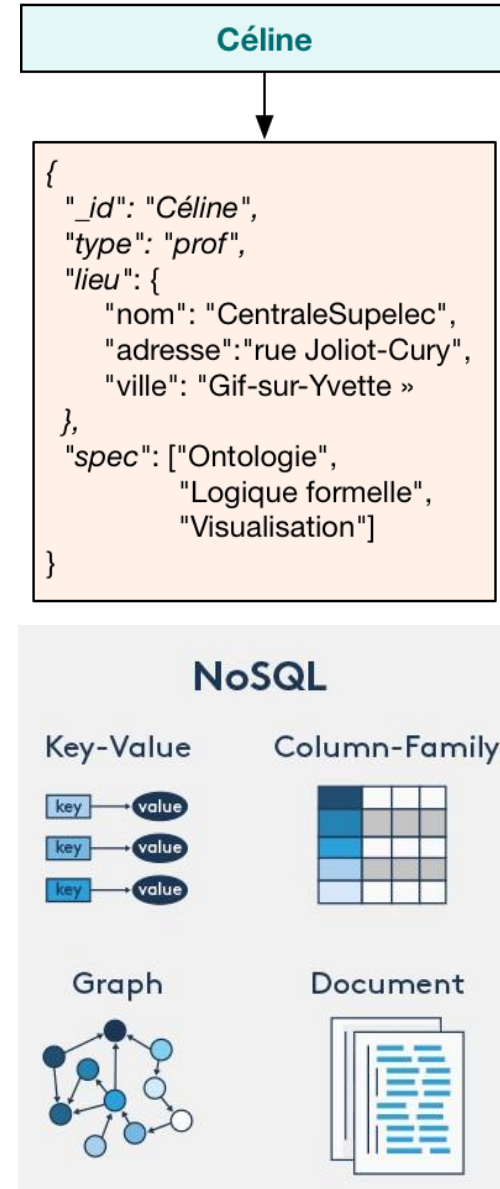
Agrégat

Un concept qui signifie une collection d'objets liés par une entité racine. L'agrégat permet de créer une unité d'information complexe qui est gérée de façon atomique

Centralité

Une BD sert d'un dépôt central a plusieurs services et a chaque fois un agrégat est construit quand il s'agit d'un modèle relationnel. Contrairement aux modèles relationnels, les agrégats sont déjà formé dans une structure NoSQL

On peut classer les modèles par leur niveau de centralité (*clé-valeur, document, colonnes...*)



Émergence du NoSQL

Défaut d'impédance (*impedance mismatch*)

Le passage du relationnel à l'objet s'effectue avec une perte d'énergie et une résistance trop forte

```
$nom = $_REQUEST['nom'];
$prenom = $_REQUEST['prenom'];
$ville = $_REQUEST['ville'];
$pays = $_REQUEST['pays'];
$sexe = $_REQUEST['sexe'];
$age = $_REQUEST['age'];
```

```
$where = "";
$where .= isset($nom) ? (isset($where) ? " OR " : " WHERE ") . "( nom = '$nom' ) " :
""
$where .= isset($prenom) ? (isset($where) ? " OR " : " WHERE ") . "( prenom = '$prenom' ) " : ""
$where .= isset($ville) ? (isset($where) ? " OR " : " WHERE ") . "( ville = '$ville' ) " : ""
$where .= isset($pays) ? (isset($where) ? " OR " : " WHERE ") . "( pays = '$pays' ) " : ""
$where .= isset($sexe) ? (isset($where) ? " OR " : " WHERE ") . "( sexe = '$sexe' ) " : ""
$where .= isset($age) ? (isset($where) ? " OR " : " WHERE ") . "( age = '$age' ) " : ""
```

```
$sql = "SELECT nom, prenom, ville, pays, sexe, age
FROM Contact
$where
ORDER BY nom, prenom DESC ";
```

```
$req = mysql_query($sql) or die('Erreur SQL !<br />'. $sql . '<br />'.mysql_error());
```

```
$data = mysql_fetch_array($req);
```

```
mysql_free_result ($req); ??
```

*ons construire une chaîne de caractère pour bâtir la
a envoyée telle quelle au serveur »*

cert
dan

```
<?
$mongo = new Mongo('mongodb://localhost:27017', array("timeout" =>
2000));
$db = $mongo->selectDB('CRM');

$db->Contact->find(
array('$or' => array(
    array('nom' => $nom),
    array('prenom' => $prenom),
    array('ville' => $ville),
    array('pays' => $pays),
    array('sexe' => $sexe),
    array('age' => $age)
)),
array('nom', 'prenom', 'ville', 'pays', 'sexe', 'age')
);
?>
```

e de
client.

Émergence du NoSQL

Les NULL

Du fait de la prédéfinition d'un schéma rigide sous forme de table, où chaque colonne est prédéfinie, un moteur relationnel doit indiquer d'une façon ou d'une autre l'absence de valeur d'une cellule.

Id	Nom	Titre
001	xx	Mme
002	Xx	NULL
003	Xx	NULL
004	xx	NULL

*Le marqueur NULL indique
clairement l'absence de
valeur, ce n'est donc pas une
valeur*

```
SELECT *  
FROM Contact  
WHERE Titre = 'Mme' OR Titre <> 'Mme' OR Titre IS NULL;
```

*Nul besoin de gérer ces
valeurs lorsqu'elles n'ont pas
de sens dans le document*

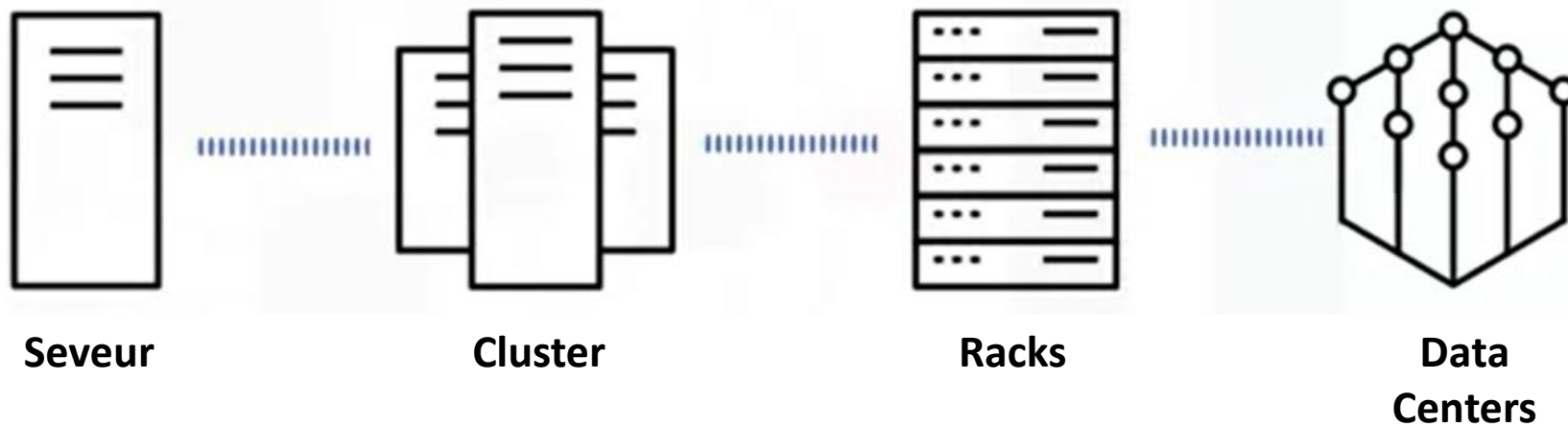
```
{  
  "_id":001,  
  "Nom":"xx",  
  "Titre":"Mme"  
}  
{  
  "_id":002,  
  "Nom":"Xx"  
}  
{  
  "_id":003,  
  "Nom":"Xx"  
}  
{  
  "_id":004,  
  "Nom":"xx"  
}
```

Pourquoi NoSQL ?

- Devenu populaire grâce aux géants de l'internet (G,A,F,..). qui traitent d'énormes volumes de données. Le temps de réponse du système devient lent avec l'utilisation des SGBDRs.
- Permettre une meilleure adaptation autre structure de données (semi-structurées, non- semi-structurées de type binaire ou multimédia).
- Une meilleure représentation (plus naturel) des entités du domaine métier
- Meilleure répartition de la charge de la base de données sur plusieurs hôtes lorsque la charge augmente. Cette méthode est connue sous le nom de « Scaling out ».

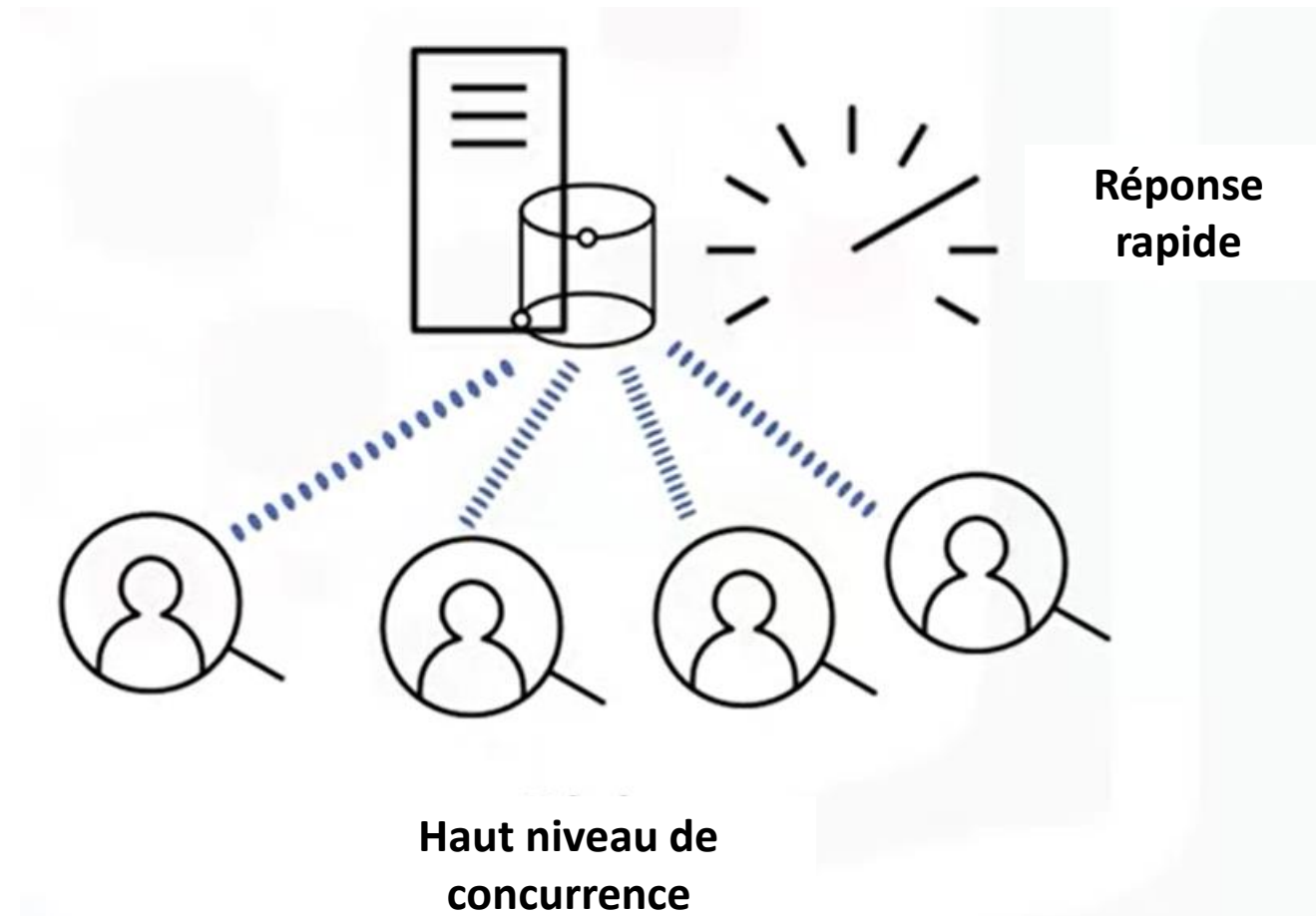
Pourquoi NoSQL ?

- Scalability / monter en échelle en horizontale



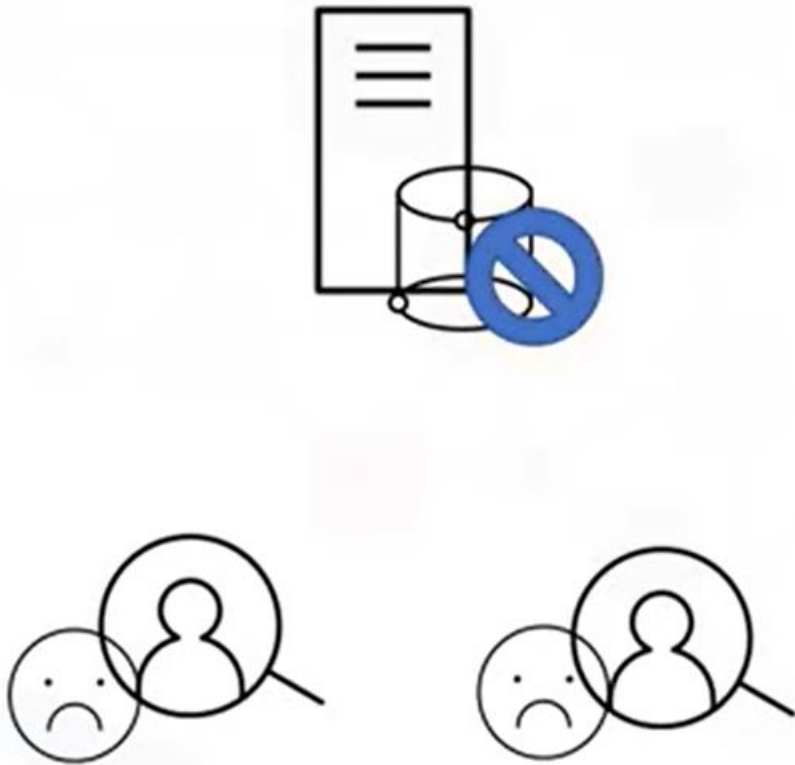
Pourquoi NoSQL ?

- Performance

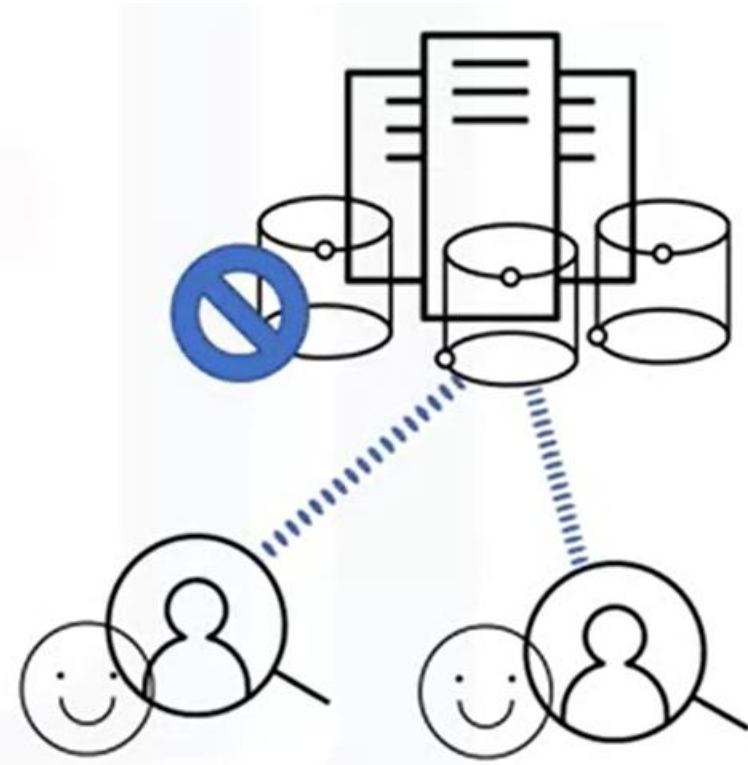


Pourquoi NoSQL ?

- Haute disponibilité

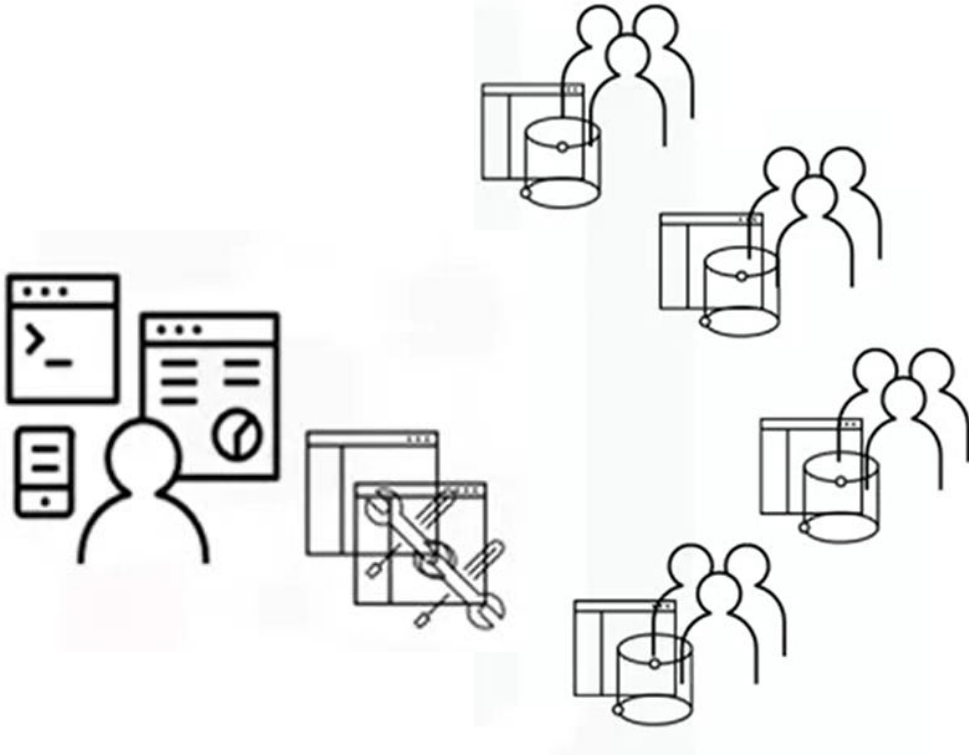


BDD traditionnelle

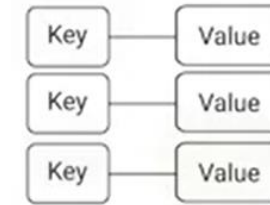


Pourquoi NoSQL ?

- Schéma de données flexible avec une variété de structure de stockage



**Ajout de nouvelle fonctionnalité
non contraignante**



**Résolution de problème de manière
plus pertinente que le modèle
relationnelle**

Du relationnelle au non-relationnelle

Les bases relationnelles = applications gérant des informations structurées et régulières : applications de “gestion”, Web, mobiles:

- **Une modélisation normalisée.**
- **Un langage (SQL) très bien défini, normalisé lui aussi.**
- **De très bonnes performances, obtenues automatiquement.**
- **Une gestion robuste de la concurrence d'accès**

Catégories de SGBD NoSQL

En 2002, Seth Gilbert et Nancy Lynch du MIT (Massachusetts Institute of Technology) ont publié un papier visant à apporter une démonstration de ce principe. Gilbert et Lynch analysent ce trio de contraintes sous l'angle des applications web et concluent qu'il n'est pas possible de réaliser un système qui soit à la fois ACID et distribué.

Théorème CAP

Un système informatique de calcul distribué ne peut garantir à un instant donné qu'au plus deux des trois contraintes suivantes :

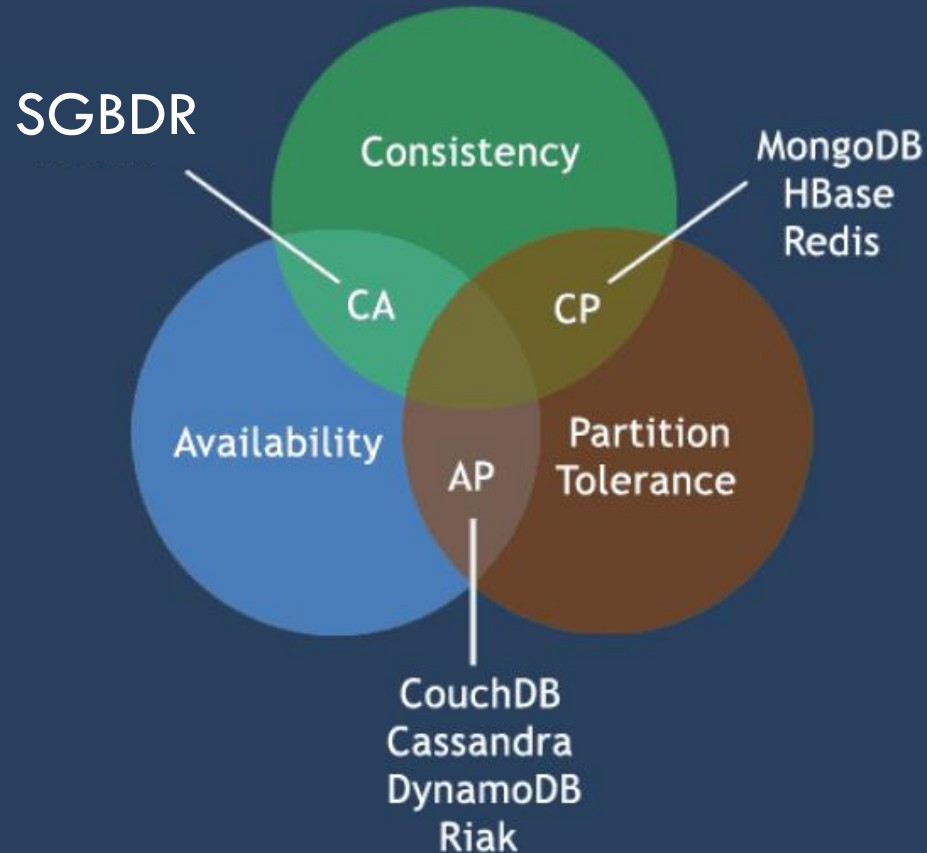
- Cohérence (Consistency) : tous les nœuds du système voient exactement les mêmes données au même moment
- Disponibilité (Availability) : la perte de nœuds n'empêche pas le système de fonctionner correctement (chaque client peut toujours lire et écrire)
- Tolérance au partitionnement (Partition tolerance) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (en cas de morcellement en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome)

Corollaire

Les transactions ACID sont impossibles dans un environnement distribué

Théorème CAP

Théorème CAP



BASE Concept

Basically-Available,
Soft-state,
Eventually-consistent

Dans un système distribué il
pratiquement impossible de
maintenir une cohérence et une
bonne disponibilité

On privilégie la *cohérence finale*

Schémas de donnée dans les bases NoSQL

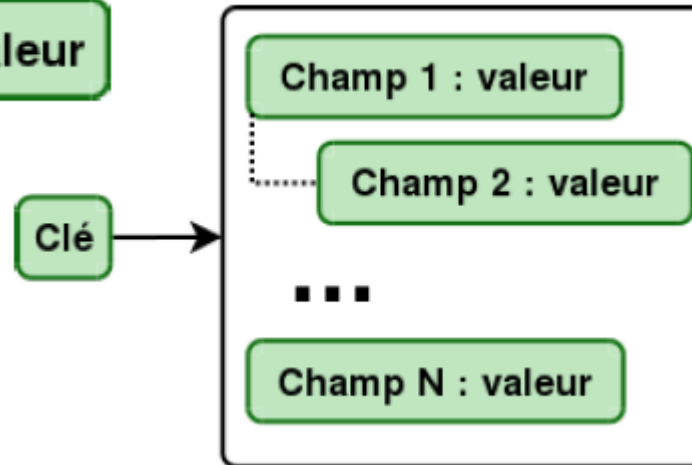
Catégories de SGBD NoSQL

Selon la littérature il existe quatre catégories de modèle de données NoSQL:

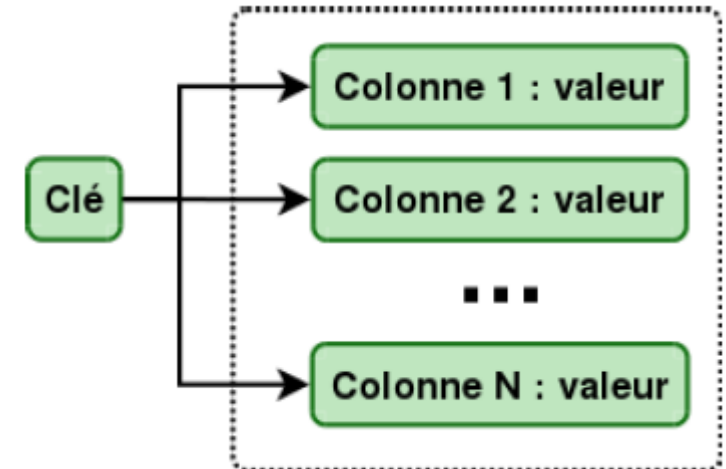
➤ **Clé-valeur**



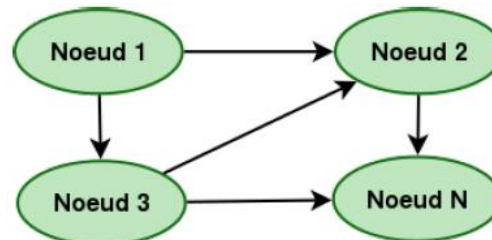
➤ **Orienté documents**



➤ **Orienté colonnes**



➤ **Orienté Graphe**



Il est à rappeler que les bases de données NoSQL sont dites sans schéma (Schemaless ou schema-free)

Catégories de SGBD NoSQL

Autres modèles :

- **RDF ou Triple Store**
- **Base de données XML**
- **Base de donnée orienté objet (ORM)**

Mais qu'est-ce qui lie les bases de données NoSQL entre elles ?

La majorité d'entre elles ont leurs racines dans la communauté open source.

Nombre d'entre elles ont été utilisées et exploitées dans le cadre d'un logiciel libre

Clé-valeur

- Récupération d'un agrégat à partir de sa clé
Fonctionne comme une table de correspondance
- Stockage d'agrégats sous la forme clé-valeur
La clé joue le rôle d'identifiant unique de chaque agrégat

- Traduit en SQL:

```
HSET "actor:1" first_name "Chris" last_name "Pratt" date_of_birth 1979  
HSET "actor:2" first_name "Zoe" last_name "Saldana" date_of_birth 1978  
HSET "actor:3" first_name "Dave" last_name "Bautista" date_of_birth 1969
```

```
create table ENREGISTREMENT(Cle varchar(2048) not null primary key, Valeur blob);
```

Exemples de base de données



Memcached



Riak



Redis

Orienté documents

- Récupération du document ou d'une **partie de document**

À partir de requêtes sur les champs de l'agrégat

- Stockage d'agrégat sous la forme de document

Chaque document est identifié de manière unique par un ID

- Traduit en SQL:

```
create table COMMANDE ( NCOM varchar(12) not
                        CLIENT row( NCLI
                                    NOM
                                    ADRESSE
                                    LOCALITE
                        DETAILS row( QCOM dec
                                    PRODUIT
```

```
{
  "ncom": "1",
  "client":{
    "ncli":"02",
    "nom":"K. Hardono",
    "adresse":"rue du capitaine hadook VirtualCity",
    "localite":"locality"
  },
  "details":{
    "qcom":0002122,
    "produit":[{
      "npro":02354856987458,
      "libelle":"livre",
      "prix":
    }]
  }
}
```



Orienté colonnes

- Stockage sur disque des colonnes au lieu des lignes

On peut voir le stockage comme un map à deux niveaux

- Structure clé-valeur avec identifiant de ligne comme clé

Le second niveau contient les informations sur les colonnes

- Expression en SQL :

```
create table LIGNES ( CLE integer not null primary key,  
                     ELEMENT row( NOM varchar(256), VALEUR varchar(256) array);
```



Les bases de données orientées colonnes optimisées pour une extraction rapide de colonnes de données.



Ce type de stockage réduit considérablement la charge I/O globale du disque



Orienté colonnes

Exemple

« Orientée ligne »

Id	Nom	Prénom
1	Brico	Juda
2	Diote	Kelly

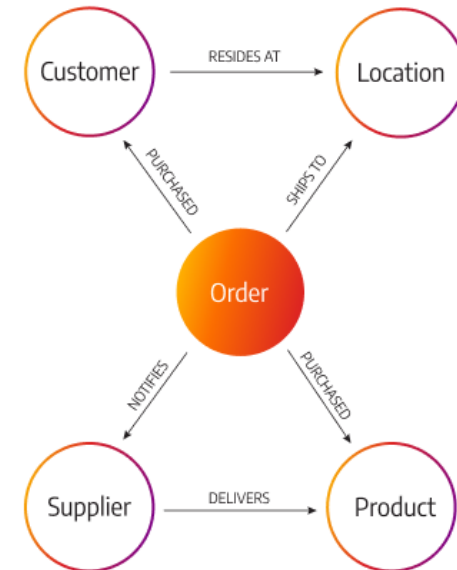
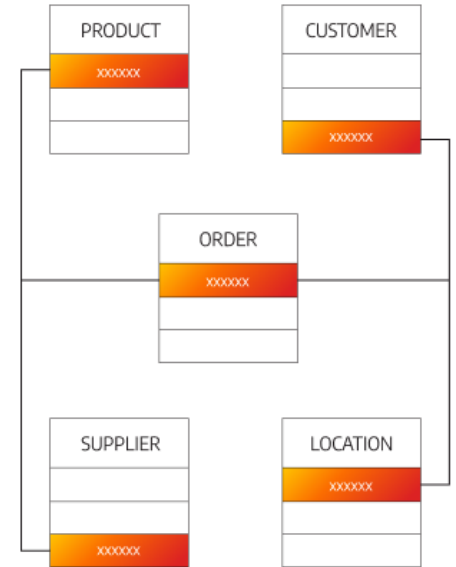


« Orientée Colonne »

Ligne « row »	Colonne « column »	Valeur « value »
1	Nom	Brico
1	Prénom	Juda
2	Nom	Diote
2	Prénom	Kelly

Orienté Graphe

- Possibilité de relation entre les agrégats
Avec possibilité de mise à jour automatique
- Utile pour des petits enregistrements avec beaucoup de liens
Ensemble de nœuds connectés par des arêtes
- Le modèle graphe est considéré comme faisant partie de la famille NoSQL, même s'il supporte mal la fragmentation en raison de la forte connectivité



Les schémas de données dans les bases NoSQL

Les arbres comme modèle de données
JSON et JSON Path

Les schémas de données dans les bases NoSQL

- Le modèle relationnel repose sur une stabilité du schéma des données
- Il est rare que dans un système relationnel dans lequel un schéma est resté invariant depuis sa première mise en production
- Dans les BDD NoSQL on cherche à assouplir cette contrainte, généralement en proposant une approche dite **sans schéma (schema-less)** , ou à **schéma relaxé**



Aucune vérification ou contrainte de schéma n'est effectuée par le moteur



Schéma implicite

Les moteurs purement paires clé-valeur peuvent prétendre à une appellation **sans schéma**

- Il est fortement recommandé de conserver une structure homogène dans la même collection dans le cas du modèle orienté document et colonne


Les schémas de données dans les bases NoSQL

- Le modèle relationnel repose sur une stabilité du schéma des données
- Il est rare que dans un système relationnel dans lequel un schéma est resté invariant depuis sa première mise en production
- Dans les BDD NoSQL on cherche à assouplir cette contrainte, généralement en proposant une approche dite **sans schéma (schema-less)**, ou à **schéma relaxé**

 **Aucune vérification ou contrainte de schéma n'est effectuée par le moteur**

Schéma implicite

Les moteurs purement paires clé-valeur peuvent prétendre à une appellation **sans schéma**

 Danger si plusieurs applications sur la même base de données Elles doivent se mettre d'accord sur le schéma des données

- Il est fortement recommandé de conserver **une structure homogène** dans la même collection dans le cas du modèle orienté document et colonne

Les schémas de données dans les bases NoSQL

- Autorise le stockage de données non uniformes
 - Ce qui élimine le besoin d'avoir des valeurs NULL
- La gestion des données met au centre le développeur en plus de l'administrateur des bases de données
 - La responsabilité de la qualité des données incombe donc au développeur
- Opérations beaucoup plus diverses et variées en NoSQL
 - Grand nombre d'ajouts et de mises à jour
 - Opérations sur d'autres entités que des lignes de tables
- Les bases NoSQL ne sont pas économes en espace disque
- NoSQL pas adapté pour données avec beaucoup de relations



Rien n'empêche de bâtir ses bases NoSQL selon au moins quelques principes relationnels



Les moteurs NoSQL possèdent un schéma implicite et semi-structuré

Les schémas de données dans les bases NoSQL

Les arbres comme modèle de données
JSON et JSON Path

Données arborescentes

- Données stockées dans un arbre :
 - différent du modèle relationnel
 - données de base (\approx feuilles) : textuelles (mais interprétables)
 - nœuds « de structure » (\approx internes) : différents types selon le modèle
- Deux standards (modèles) majeurs :
 - **JSON**
 - XML

Nœuds :

- Données de base (atomiques)
 - texte (string)
 - nombre (number)
 - null
- Tableau (array)
- Dictionnaire (object)

Syntaxe

"toto"

2.5

null

[... , ... , ...]

{ "a" : ... , "truc" : ... }

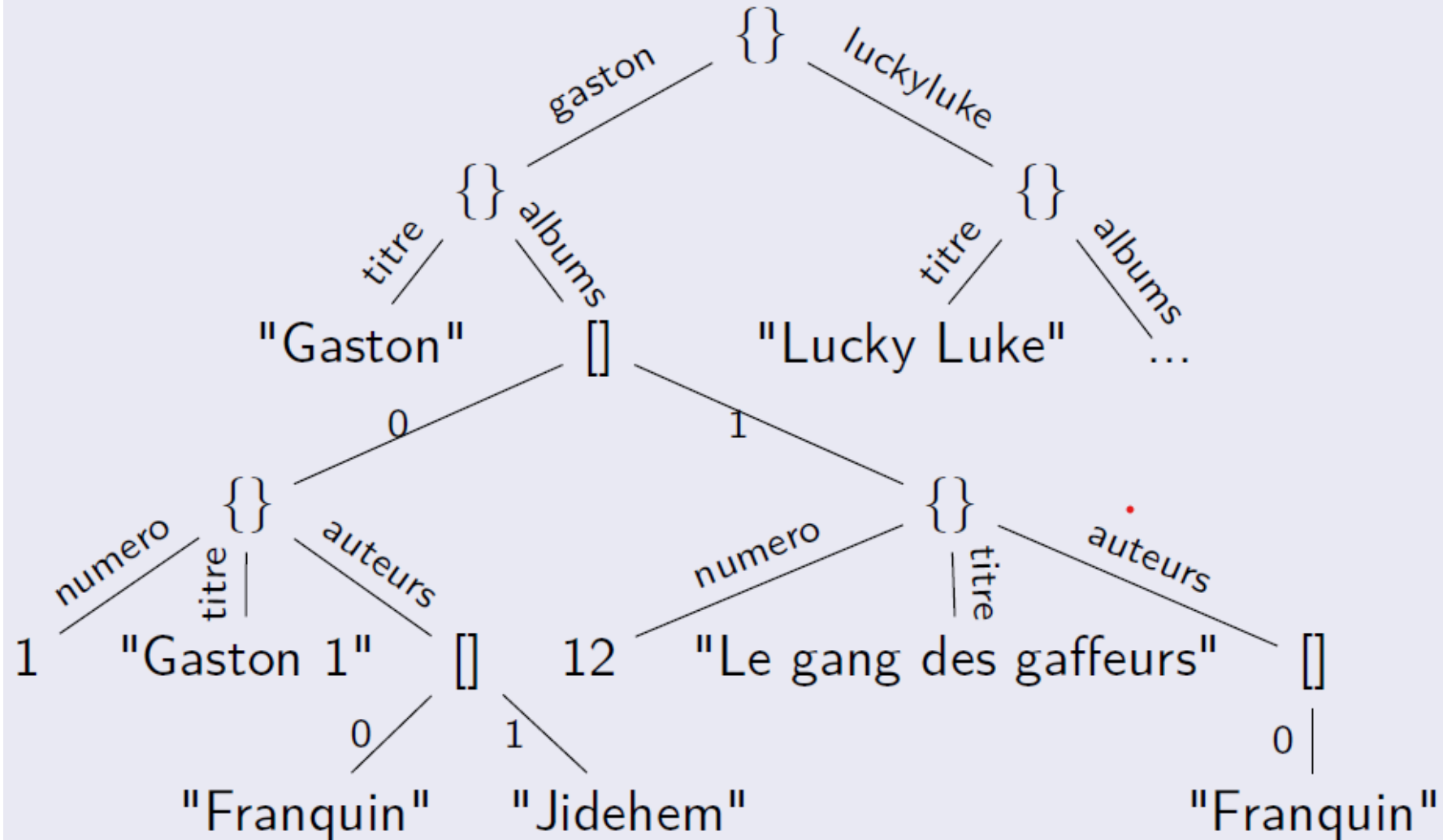
Exemple de document JSON

Collection d'albums de BD

```
{
  "gaston":{
    "tie":"Gaston",
    "albums":[
      {
        "numero":1,
        "titre":"Gaston 1",
        "auteurs":[
          "Franquin",
          "Jidehem"
        ]
      },
      {
        "numero":12,
        "titre":"Le gang des gaffeurs",
        "auteurs":[
          "Franquin"
        ]
      }
    ]
  },
  "lucky Luke":{
    "titre":"Lucky Luke",
    "albums":[
      {
        "numero":1,
        "titre":"La mine d'or de DickDigger",
        "auteurs":[
          "Morris"
        ]
      }
    ]
  }
}
```

Exemple avec arbre

Extrait de la collection précédente



Récupération de l'information dans un arbre

Deux possibilités :

- Programmatiquement dans un langage générique, e.g., en Python ou en Javascript
- En utilisant un langage dédié, plus compact : JSONPath et SQL/JSONPath

Approche programmatique

- Parfois compacte pour des recherches simples :

Recherche en Python

```
-> data = . . .  
-> data ["gaston"] ["albums"] [0] ["auteurs"] [0]  
Le premier auteur du premier album de Gaston
```

Recherche en JS

```
-> data = . . .  
-> data.gaston.albums[0].auteurs[0]  
Le premier auteur du premier album de Gaston
```

Approche programmatique : plusieurs valeurs

➤ Extraction simple en Python:

*Extraire le **titre** de chaque **série***

Impératif

```
ex2 = []  
for key in data:  
    ex2.append(data[key]["titre"])  
print(ex2)
```

Fonctionnel

```
ex2 = list(map(lambda item: data[item]['titre'], data))
```

Approche programmatique : plusieurs valeurs

- Extraction simple en Python:
*Extraire le titre de chaque **album***

Impératif

```
ex3=[]  
for key in data:  
    for album in data[key]['albums']:  
        ex3.append(album['titre'])  
print(ex3)
```


Approche programmatique : Extraire le titre des séries dont on possède l'album numéro 10 ou plus

- Extraction simple en Python:

Extraire le titre des séries dont on possède l'album numéro 10 ou plus

Impératif

```
ex4 = []
for key in data:
    for album in data[key]['albums']:
        if album["numero"]>=10:
            ex4.append(album['titre'])
            break
print(ex4)
```

Langage dédié : JSONPath

Langage de requêtes:

- Pour chercher de l'information dans des documents JSON
- ou comme critère de filtrage de documents JSON

Langage de chemins:

- Expression = spécification de chemin dans un arbre JSON
- Sémantique :

expression + nœud de départ
 \rightsquigarrow ensemble de nœuds

Langages de chemins : analogie avec les chemins de fichier dans un shell bash

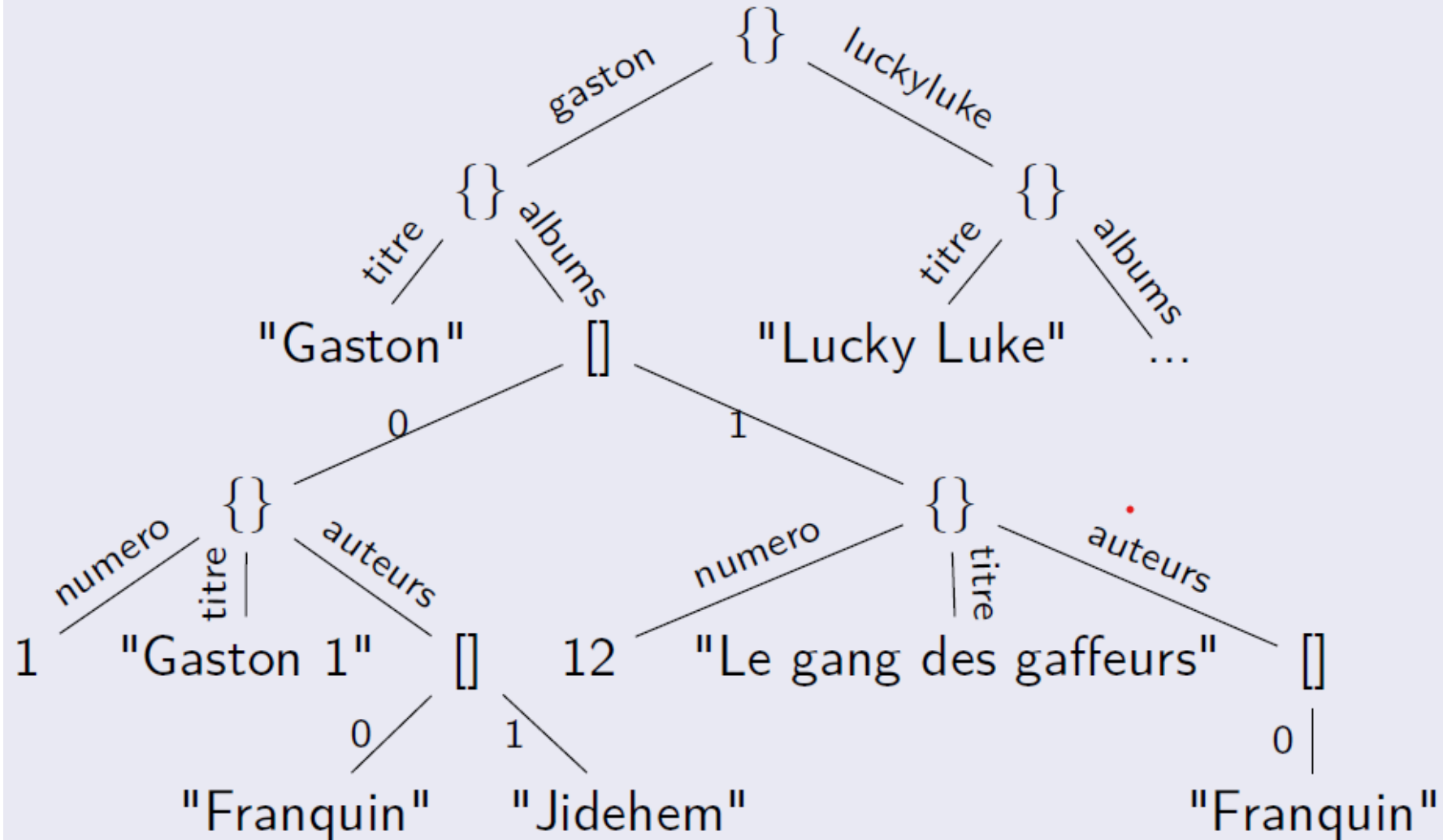
Noms de fichier dans les commandes shell :

- Nom absolu, e.g. `/home/jagoun/bdnosql/cm01.pptx`
 - un seul fichier possible
- Avec *wildcard*¹ e.g. `bdnosql/*.tex`
 - plusieurs fichiers possibles

[1] Un **métacaractère** (en [anglais](#), *wildcard* ou *joker*) est un type de [caractère informatique](#) utilisé lors de la recherche d'un [mot](#) ou d'une expression incomplète sur un réseau informatisé, [ordinateur](#) ou [internet](#). Le métacaractère remplace généralement la fin ou le début du mot recherché. Les caractères de remplacement les plus couramment utilisés sont : *, ?, %.

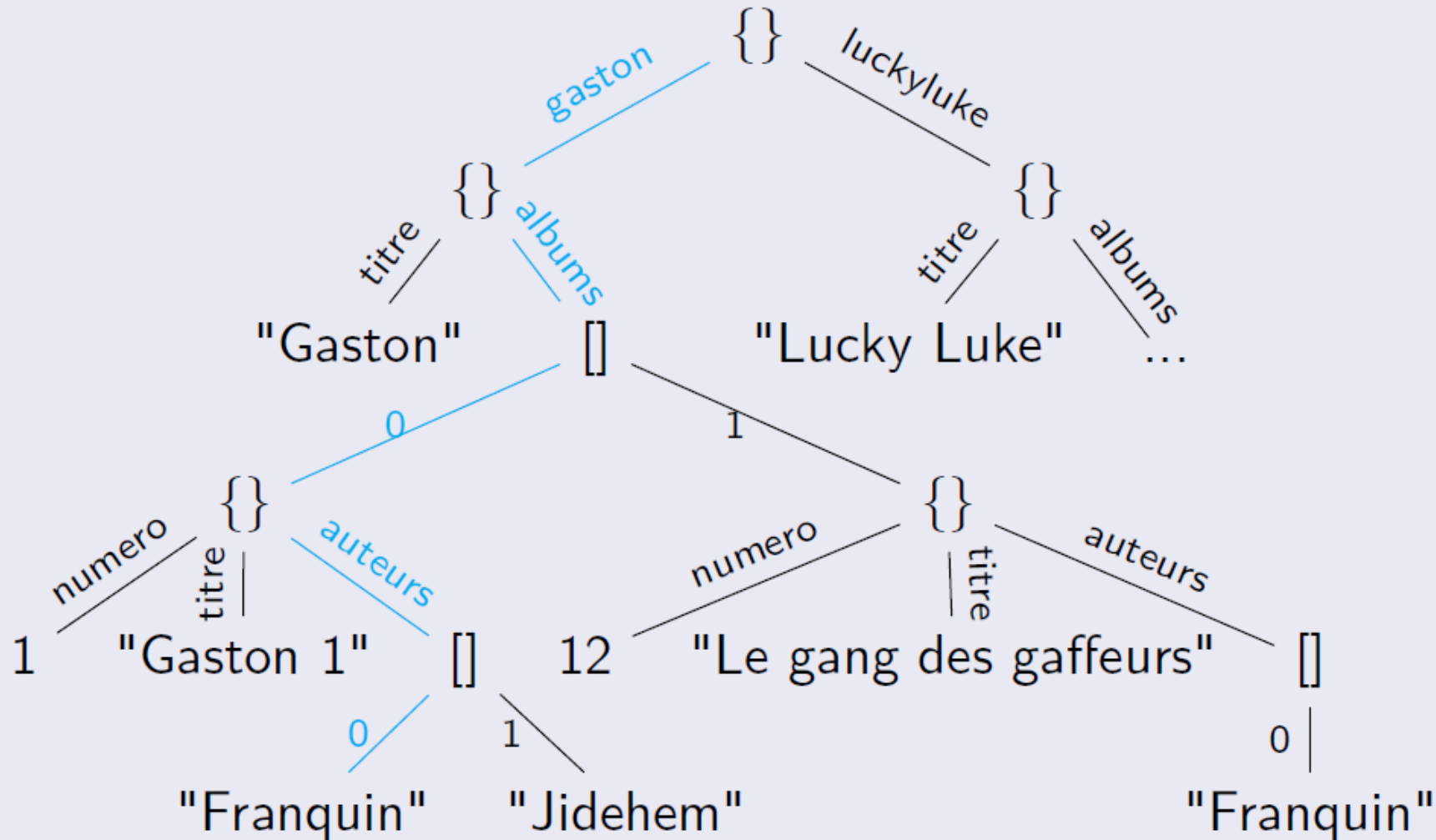
Rappel

Extrait de la collection précédente



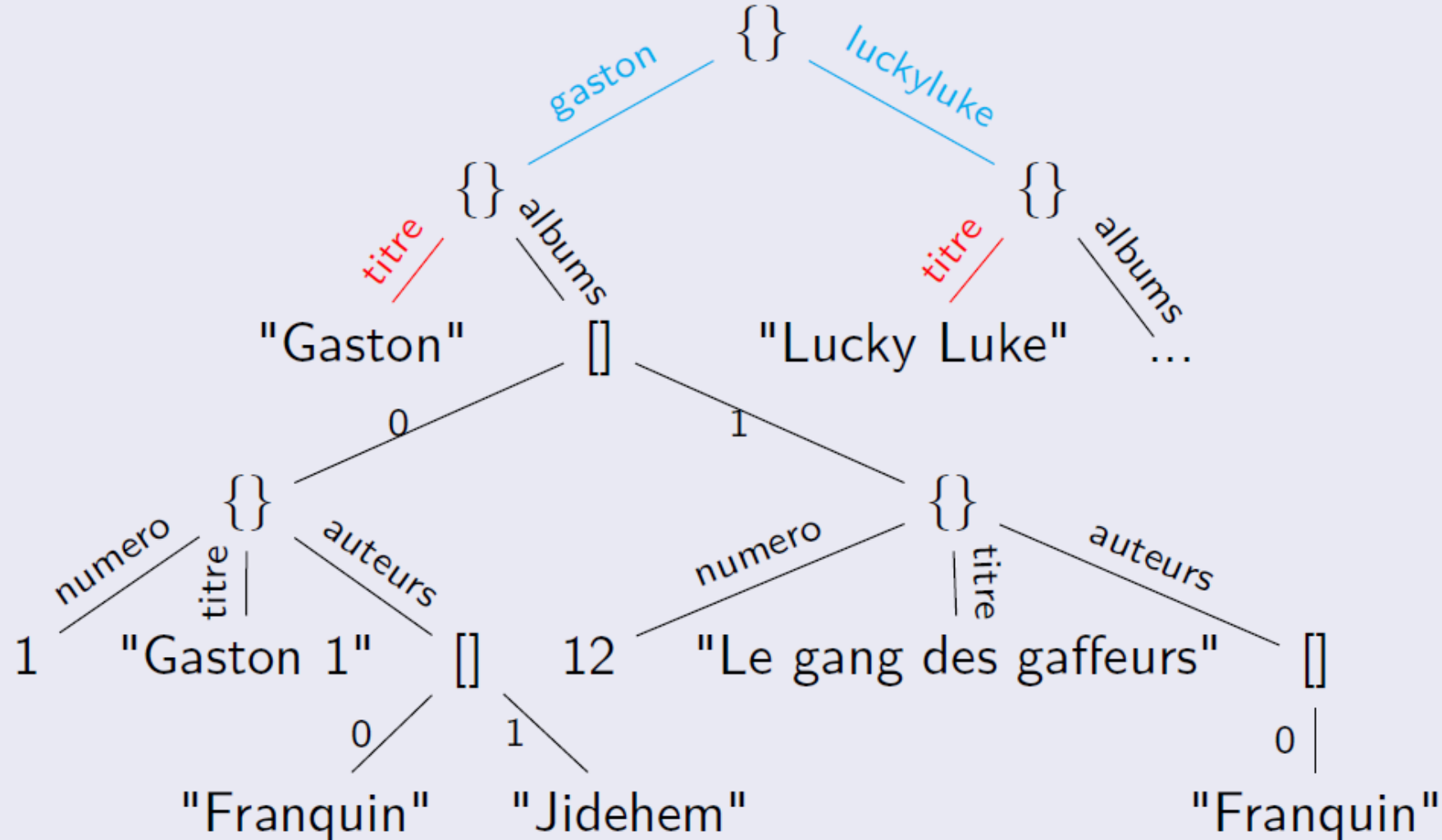
Exemple JSONPath : Le premier auteur du premier album de la série Gaston

```
$.gaston.albums[0].auteurs[0]
```



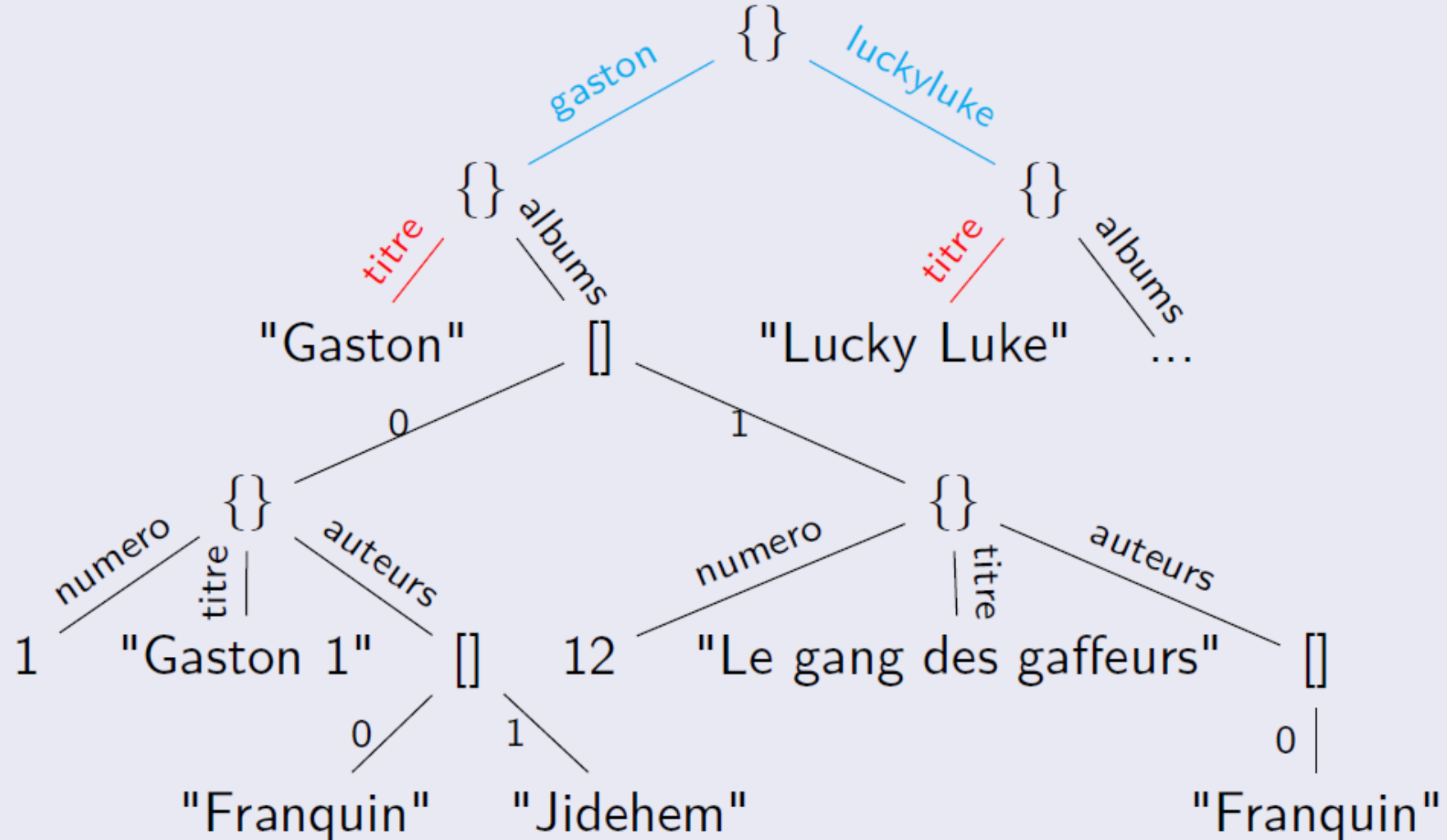
Exemple JSONPath : Le premier auteur du premier album de la série Gaston

`$.*.titre`



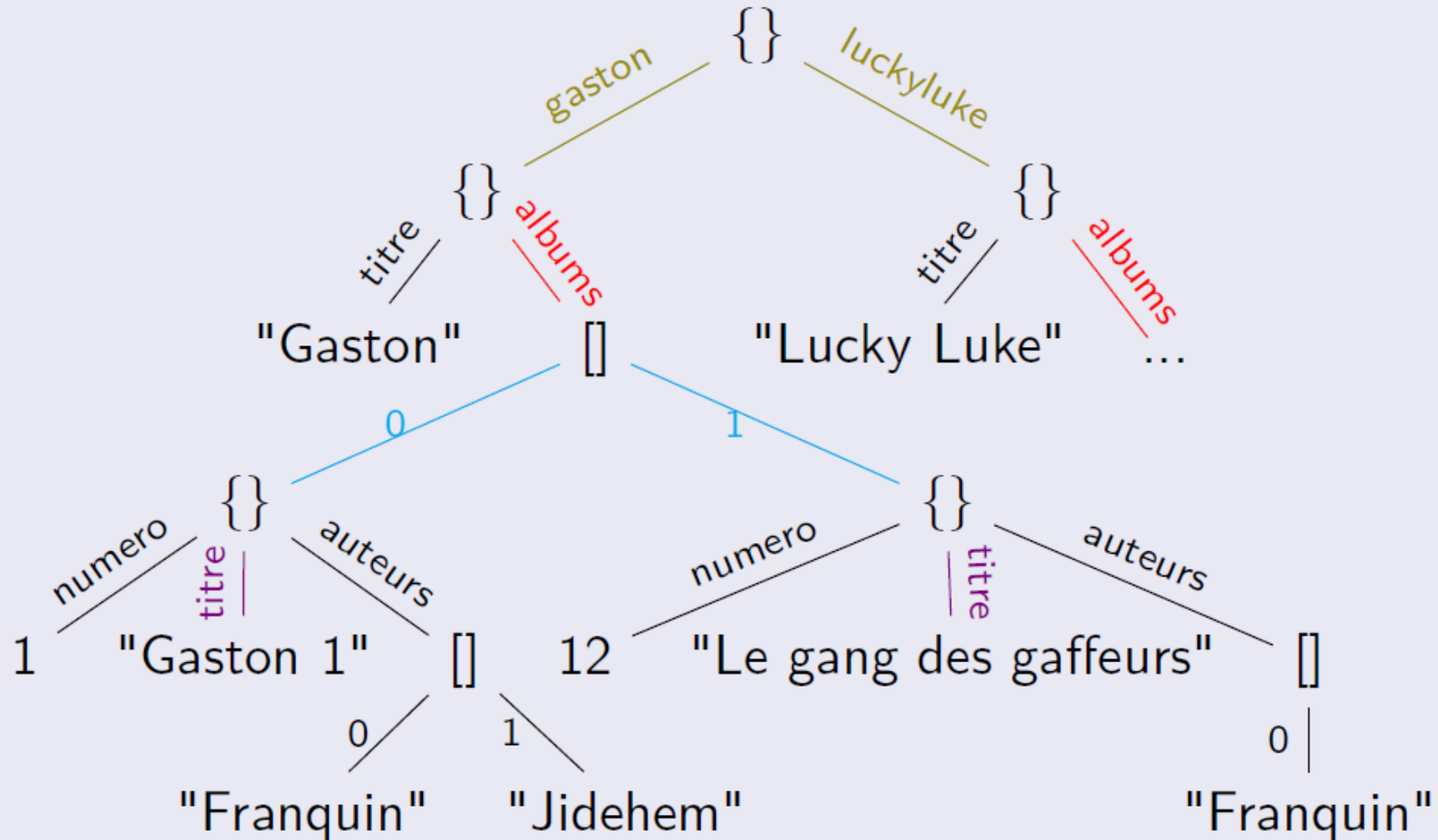
Exemple JSONPath : le titre de chaque série

`$.*.titre`



Exemple JSONPath : le titre de chaque album

```
$.*.albums[*].titre
```



Syntaxe JSONPath (extrait)

Chemins

$exprC$	$::=$	\$	racine
		@	nœud courant
		$exprC.champ$	valeur du champ
		$exprC.*$	valeur de tous les champs
		$exprC..*$	valeur de tous les champs, rékurs.
		$exprC[entier]$	case tableau
		$exprC[*]$	toutes les valeurs d'un tableau
		$exprC?(exprB)$	filtre sur cond. bool. (\approx WHERE)

Syntaxe JSONPath (extrait)

Conditions

$exprB ::=$	true	
	false	
	$! exprB$	négation
	$exprB \&\& exprB$	et logique
	$exprB exprB$	ou logique
	$exprC op val$	comparaison avec valeur
	$exprC op exprC$	comparaison avec chemin
	$exists(exprC)$	test d'existence d'un chemin relatif

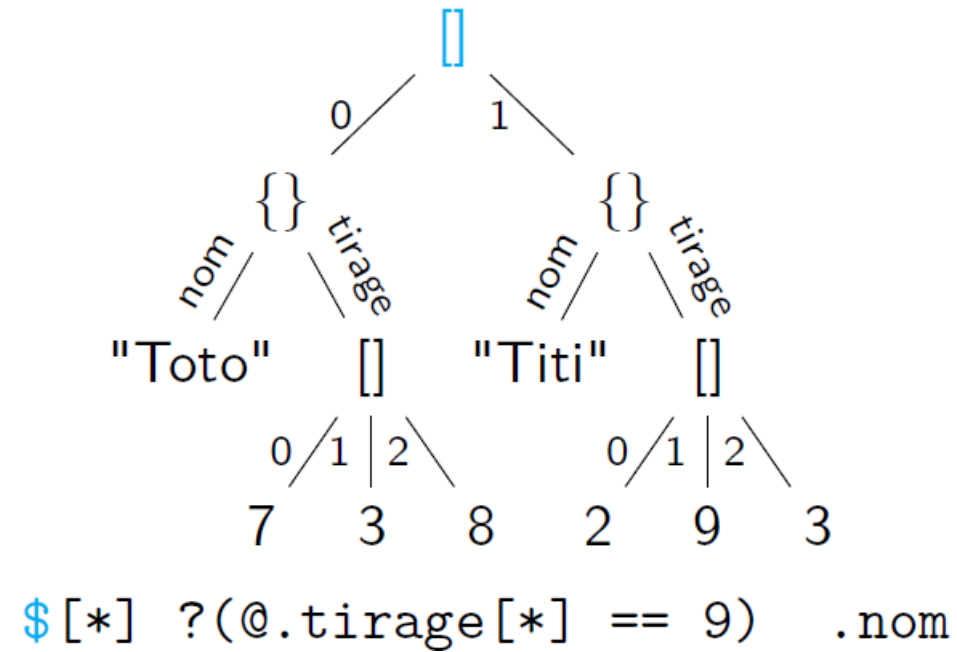
avec $op \in \{==, <>, !=, <, <=, >, >=\}$
et $exprC$ commence par @ ou \$

Exemple JSONPath : le titre des séries dont on possède l'album numéro 10 ou plus

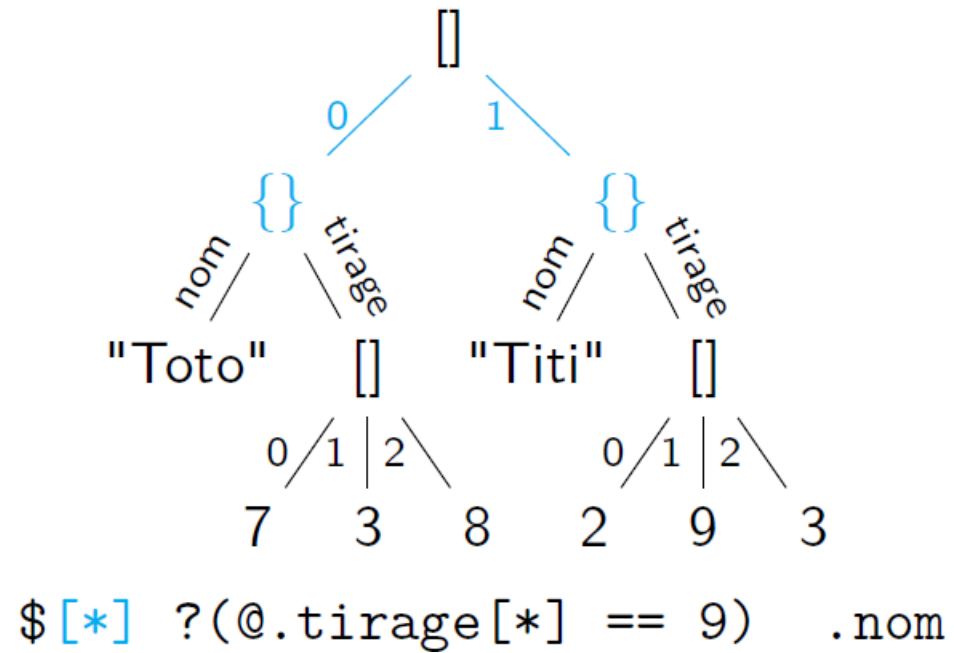
```
$.* ?(@.albums[*].numero >= 10).titre
```



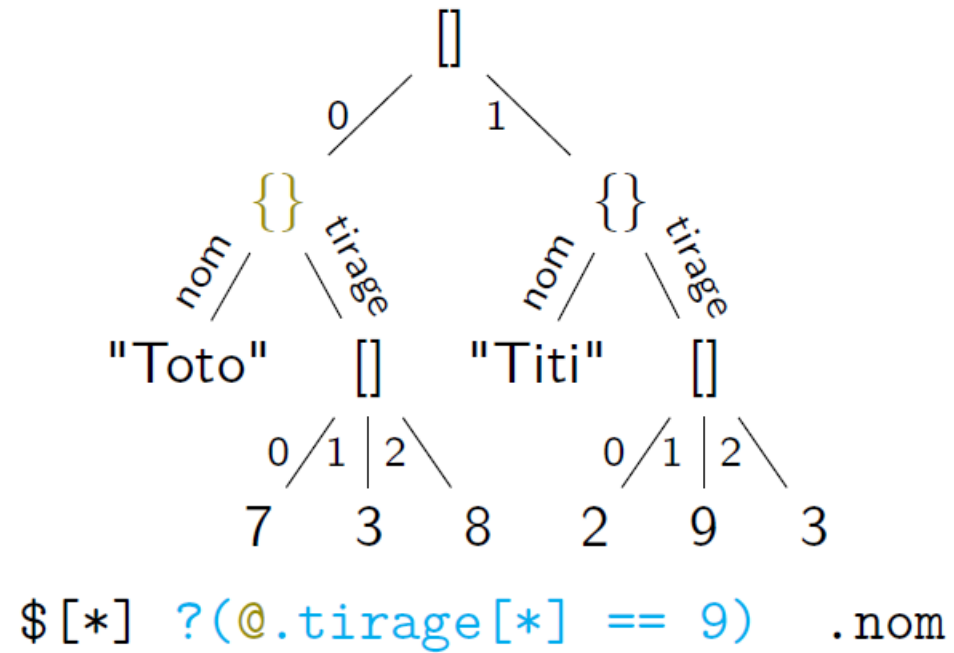
Comparison : exemple



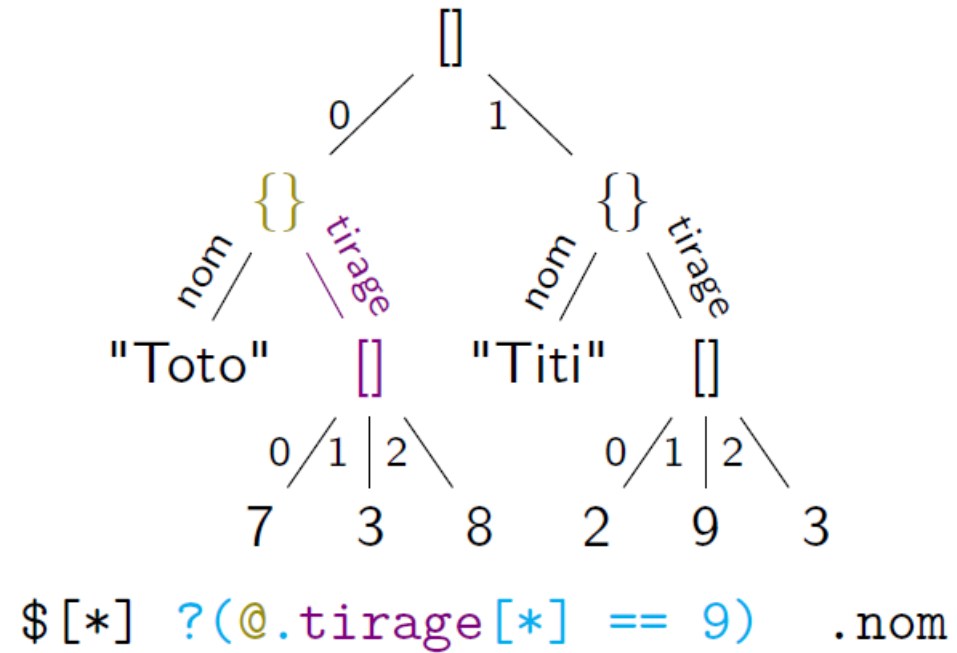
Comparaison : exemple



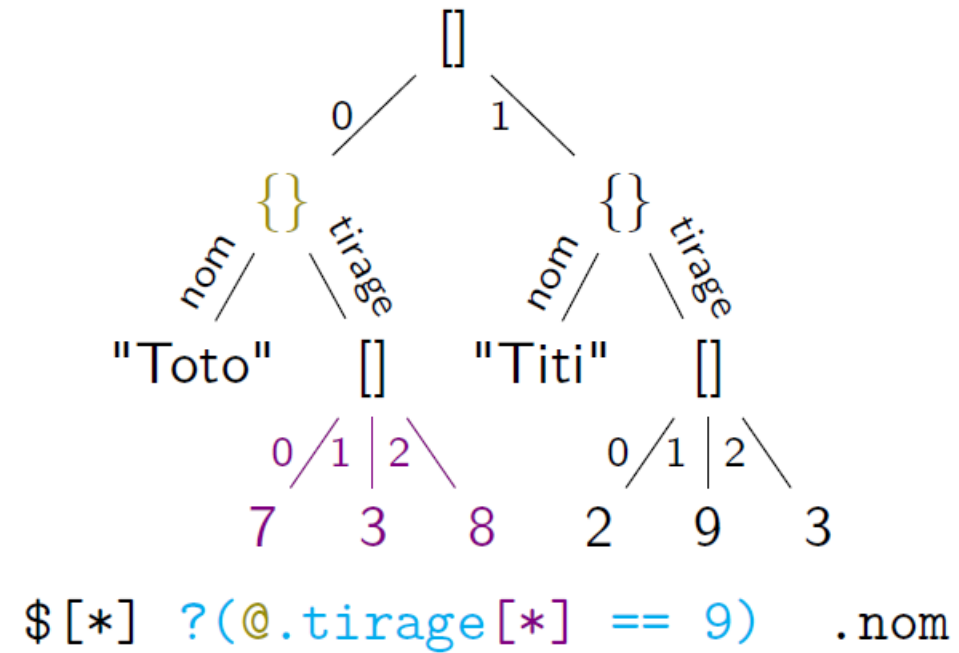
Comparaison : exemple



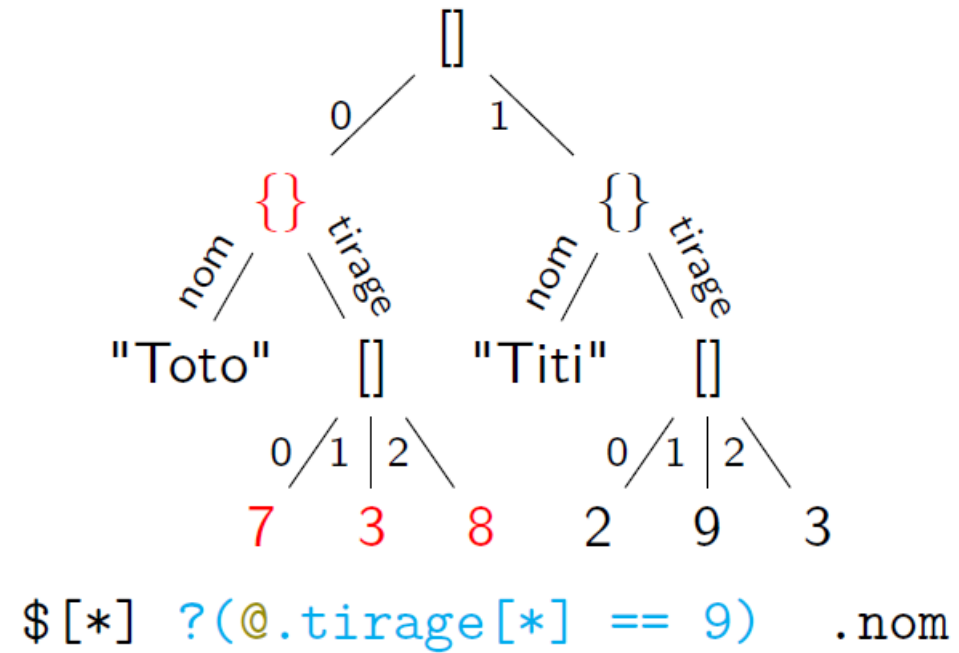
Comparaison : exemple



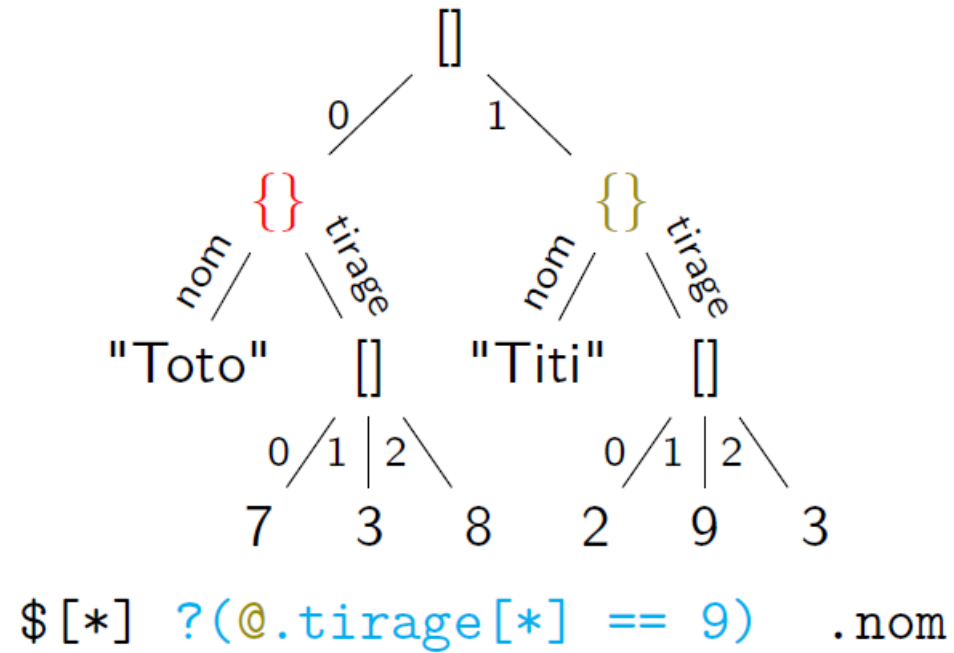
Comparaison : exemple



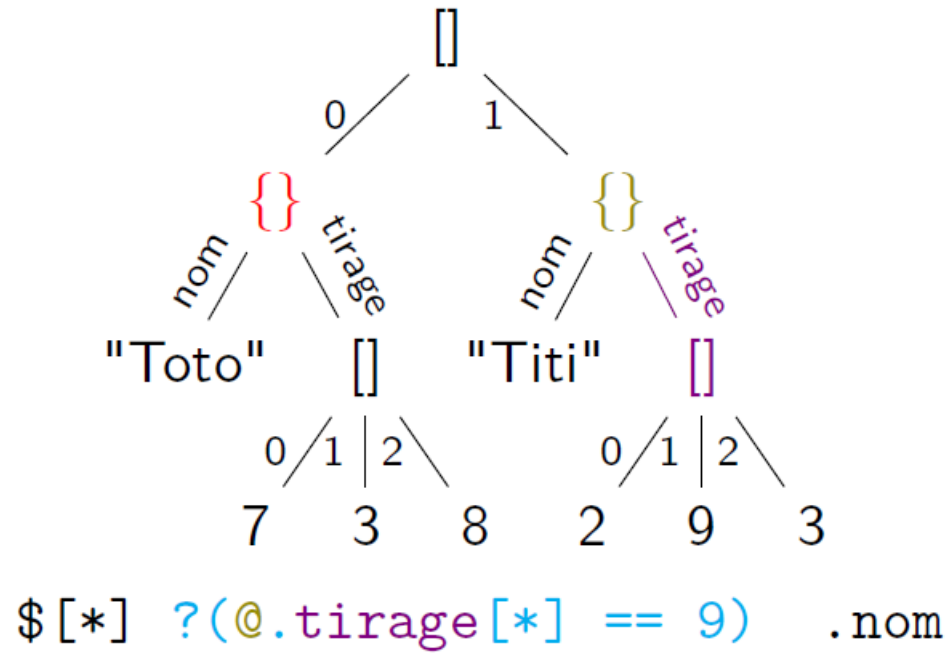
Comparaison : exemple



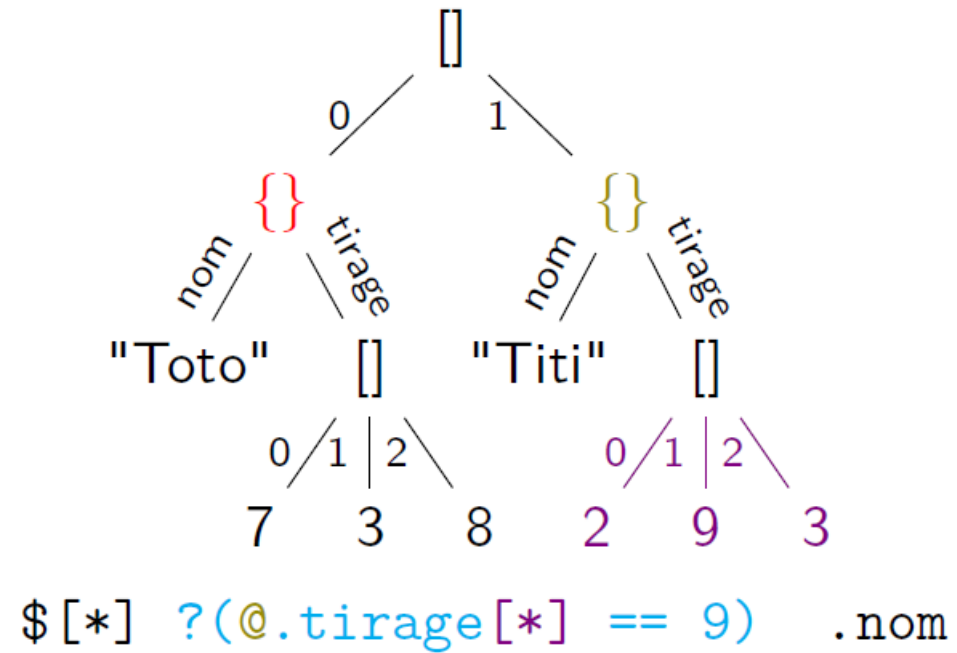
Comparaison : exemple



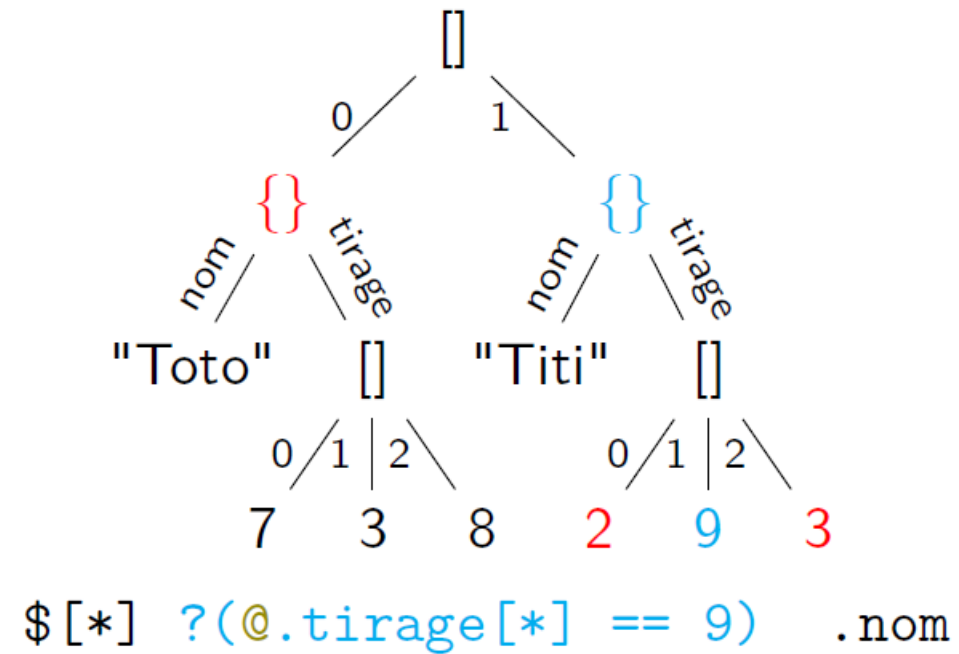
Comparaison : exemple



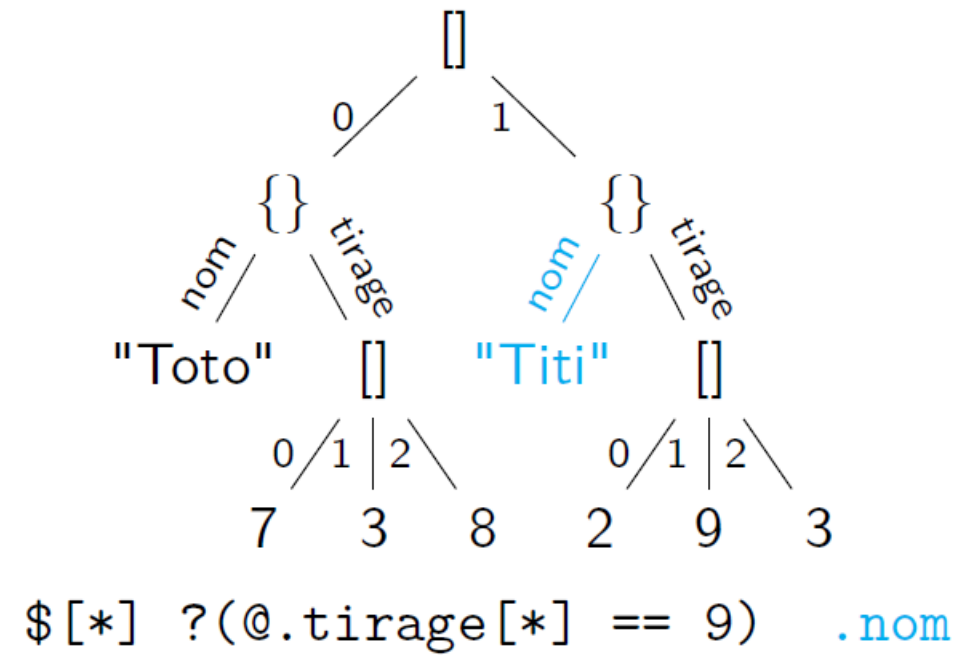
Comparaison : exemple



Comparaison : exemple



Comparaison : exemple



- I. <https://sebastien.combefis.be/files/ecam/nosql/ECAM-NoSQL4MIN-Cours1-Slides.pdf>
- II. Les bases de données NoSQL - Comprendre et mettre en œuvre- Ed 2021