

bases de données  
avancées  
SASUCGG2

# Les bases non relationnelles: NoSQL

Présenté par:

Amal HALFAOUI (Epse GHERNAOUT)

[Amal.halfaoui@univ-tlemcen.dz](mailto:Amal.halfaoui@univ-tlemcen.dz)

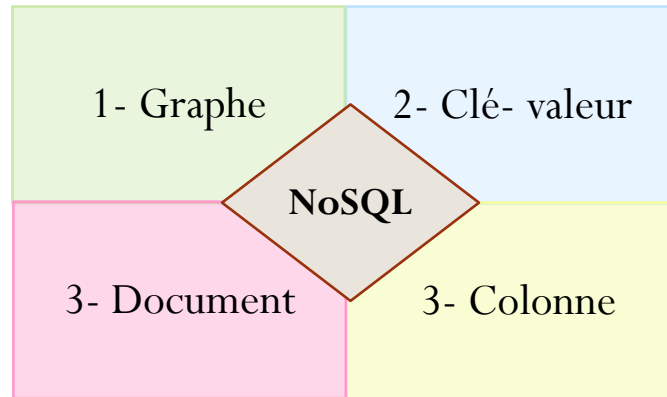
[amal.halfaoui@gmail.com](mailto:amal.halfaoui@gmail.com)

# Rappel: SGBDR vs NoSQL

Relationnel	NoSQL
Schéma défini au départ	Schémaless
Structure rigide	Structure dynamique
Gestion des valeurs Null	Pas de valeurs Null
Transactions ACID	Transactions BASE
Scalabilité Verticale	Scalabilité Horizontale sans limites
Insiste sur la Cohérence des données	Insiste sur la Disponibilité et la Performance
Mauvaise gestion de gros volumes de données	Conçu pour les gros volumes de données
Langage SQL	Différents Langages selon le modèle utilisé

# Rappel: Familles des bases NoSQL

## 1-Classement par schéma de données



## 2- Classement par Usage

- Performance
- Structures spécifique
- Structure souples
- volumétrie

# Rappel: Schéma Orienté document

## Principe:

- Basé sur le modèle clé-valeurs → par contre, **valeur** = **document**
- Le document a une structure arborescente → les données sont semi structurées (**JSON** ou **XML**)
- Les documents sont structurés mais **aucune définition de structure préalable** n'est nécessaire.
  - Document composé de clés/valeurs
  - le type de donnée peut être simples (Int, String, Date) ou sous forme d'une liste de données
  - les données peuvent être imbriqués → schéma arborescent

# Format de données structuré JSON

# JSON: JavaScript Object Notation

- JSON reprend la syntaxe de création d'objets en JavaScript.
- C'est un format de représentation logique de données → permet de stocker de l'information de manière organisée
- il est indépendant de tout langage de programmation. Il peut ainsi être interprété par tout langage à l'aide d'un parser.
- Un fichier au format JSON a pour extension **".json"**.

# JSON: JavaScript Object Notation

## Éléments JSON:

- La structure de base est un couple "**nom**": **valeur**
- 2 types de valeurs

**1- Valeur simple atomique** (Primitifs) : nombre, booléen, chaîne de caractères, null.

```
"_id": 1 → entier  
"nom": "Bruchez ", → string  
"récompense": true → boolean
```

## 2- Valeur complexe :

- **liste de valeurs** : tableau représenté par des crochet

```
"interet": [ "modélisation", "bases de données " ]
```

- **Objets** : listes de couples "**nom**": **valeur** entrés entre accolades, séparés par des virgules.

```
"adresse": {  
  "rue": "76 b bd barbes",  
  "code_postal": 75018,  
  "ville": "Paris",  
  "pays": "France"  
}
```

# JSON: JavaScript Object Notation

## Éléments JSON:

### 2- Valeur complexe :

- **Imbrication de données:** des tableaux de tableaux, des tableaux d'objets contenant eux-mêmes des tableaux, etc. (l'imbrication est sans limite)

```
{  
  "nom cours": "NoSQL",  
  "Module": "Bases de données avancées",  
  "etudiants": [  
    {"nom": "Belfadel",  
     "prenom": "wafa"},  
    {"nom": "Mekhezzem",  
     "prenom": "reda"},  
    {"nom": "Fandi",  
     "prenom": ["Amina", "Manel"] }  
  ]  
}
```



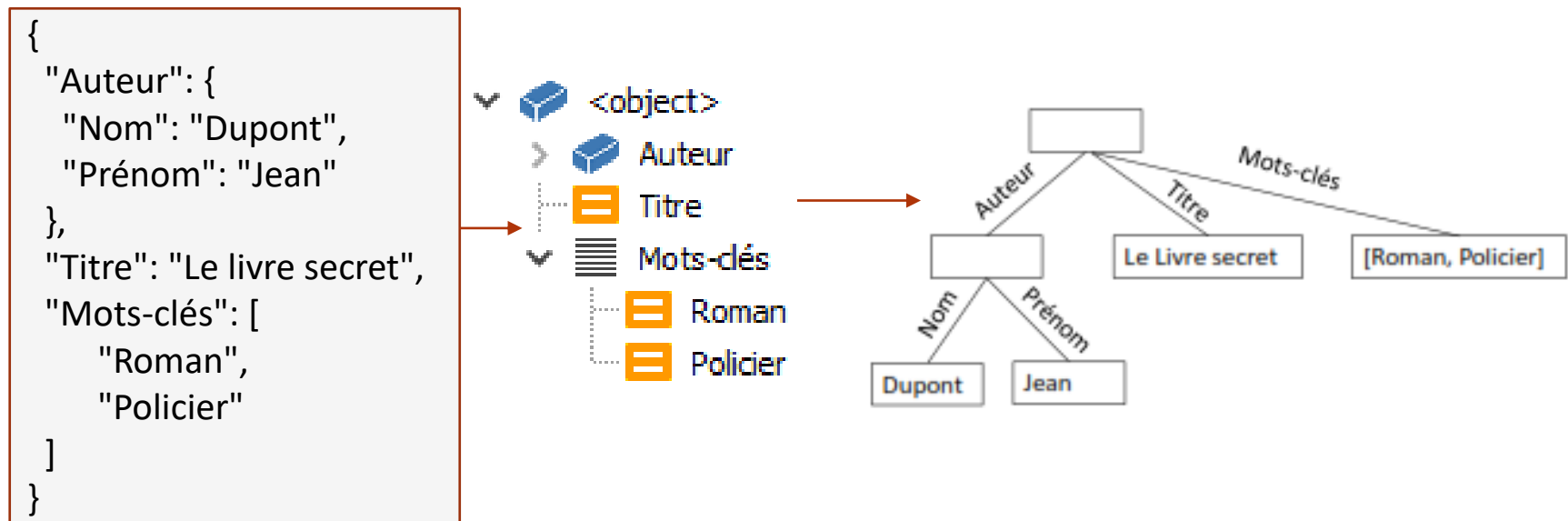
# JSON: JavaScript Object Notation

## Règles syntaxiques:

- Il ne doit exister qu'un seul élément père par document contenant tous les autres : **un élément racine**.
- Tout **fichier JSON bien formé** doit être :
  - soit un objet commençant par { et se terminant par },
  - soit un tableau commençant par [ et terminant par ].Cependant ils peuvent être vides, ainsi [] et {} sont des JSON valides.
- Les **séparateurs utilisés entre deux paires/valeurs sont des virgules**.
- Un objet JSON peut contenir d'autres objets JSON.

## Représentation arborescente

- Les documents structurés JSON peuvent être vus comme des structures arborescentes
- Exemple: Arbre représentant un objet avec une composition d'agrégats et un tableau de valeurs (simples)



L'arborescence a été représentée par l'outil:

<http://www.mitec.cz/jsonv.html>



téléchargeable



# Présentation

- Mongodb signifie en anglais humongous qui peut être traduit par « énorme »
- Initialement développé par 10gen en 2007 puis rebaptisé en 2013 MongoDB, Inc.
- Base de données Orientée document: Stockage/récupération des données sous forme de documents au format BSON (Binary JSON)
- Base de données écrite en C++;
- Dernière version stable (open source): 6 (2023) téléchargeable: <https://www.mongodb.com/try/download/community>
- Outils GUI client :
  - **Compass:**
  - **Robot 3t**

## Caractéristiques

- Les documents sont gérés dans des collections.
  - Tous les documents d'une collection ne sont pas nécessairement identiques
- Chaque document est identifié par une clé unique dans une collection
  - Qui peut être comparée à la notion de clé primaire d'une table relationnelle
  - La clé d'un document porte un nom fixe : **\_id**.  
Elle peut être fournie ou générée automatiquement (**ObjectId**)
- Une BD est une collection de collections
- MongoDB autorise aussi le stockage d'objets larges ou de documents binaires (avec BSON (taille limitée à 16), ou GridFS pour les documents de plus grande taille).

## Analogie avec le modèle relationnel

- Chaque *database* contient des *collections*.
- Chaque *collection* contient des *documents*.
- Chaque *document* est au format *JSON* et contient donc des *propriétés* et ses *valeurs*

Relationnel	Mongo
Base de données	database
table	collection
enregistrement	document
Attribut (atomique)	propriété (chaîne, entier, tableau,

# Schémas NoSQL : Orienté document

## Exemple:

Auteur:

ID	NOM	PRENOM	FONCTION	INTERRET
1	Bruchez	Rudi	Consultant	Bases de données
2	Soutou	Christian	Maitre de conférences	Modélisation, Bases de données
3	Coelho	Paulo	(null)	(null)

Clé : objectid

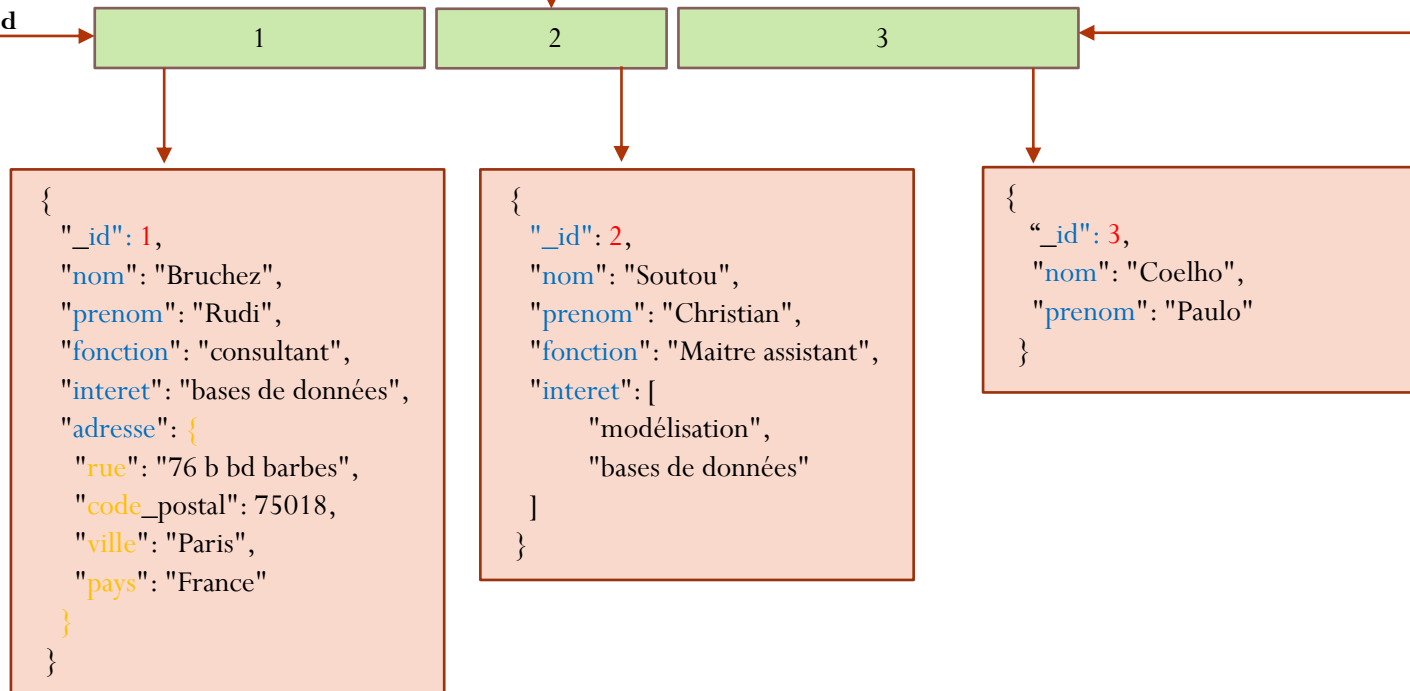


Table auteur = collection auteur

Chaque enregistrement est un document

# Modélisation d'une base Mongodb avec Json



# Modélisation en dénormalisant le schéma

Deux alternatives de modélisation:

1-L'imbrication de données

2- Les références

Quand est ce que imbriquer ? Et quand est ce que référencer?  
Ou les deux ?

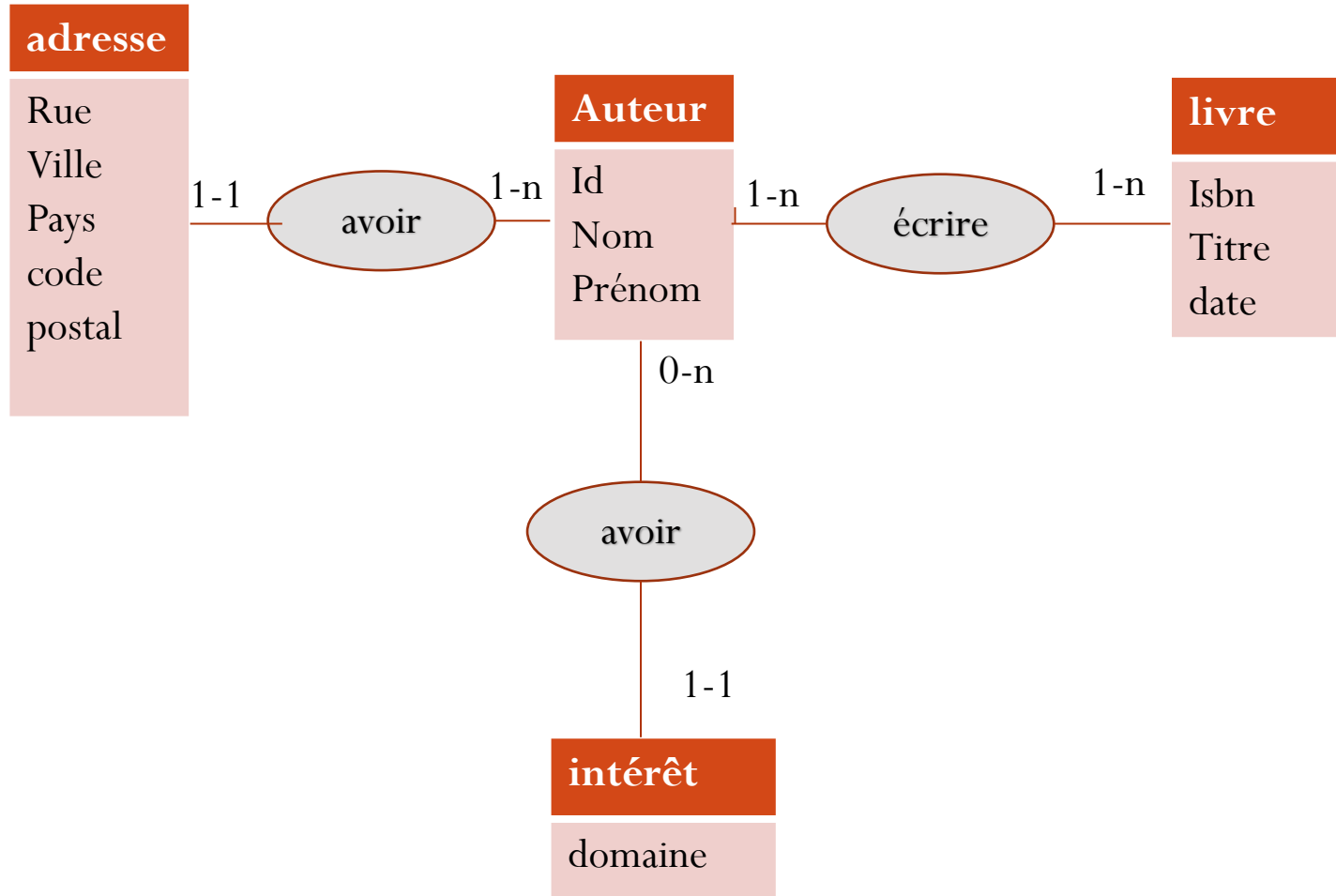
**Ne jamais construire votre schéma comme dans le relationnel :**

- Nombre de collections = nombre de tables ;
- Chaque document d'une collection référence un autre document d'une autre collection

# Modélisation en dénormalisant le schéma

- Comment les données seront-elles stockées ?
- Comment votre application va-t-elle récupérer et interroger des données ?
- Votre application exige-t-elle de nombreuses lectures (read heavy) ou de nombreuses écritures (write heavy) ?

# Reprenons notre exemple



# Imbrication des données

## Exemple: Relation 1-1 (cas de l'adresse)

Cas de l'adresse attribut atomique. L'adresse est partie intégrante de l'auteur

```
{
  "id": "a1",
  "nom": "Bruchez",
  "prenom": "Rudi",
  "fonction": "consultant",
  "interet": "bases de données",
  "adresse": {
    "rue": "76 b bd barbes",
    "code_postal": 75018,
    "ville": "Paris",
    "pays": "France"
  }
}
```

# Imbrication des données

Exemple : Relation 1-n (si plusieurs adresses)

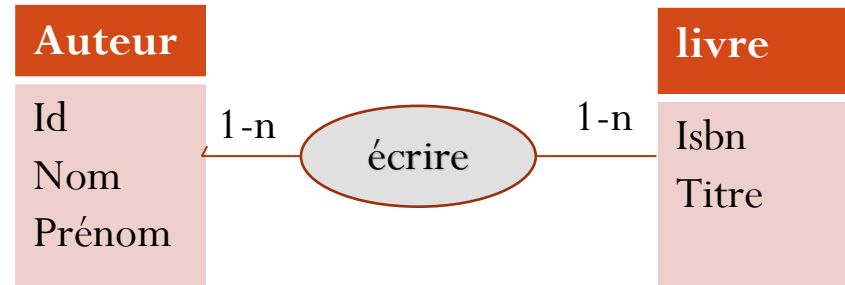
## Dénormalisation en JSon

Imbriquer les entités d'adresses dans les données du client → le nombre des adresses n'est pas très grand et il ne va pas croître sans limite.

```
{
  "id": "a1",
  "nom": "Bruchez",
  "prenom": "Rudi",
  "fonction": "consultant",
  "interet": "bases de données",
  "adresse":
  [
    {
      "rue": "76 b bd barbes",
      "code_postal": 75018,
      "ville": "Paris",
      "pays": "France"
    },
    {
      "rue": "76 b bd ",
      "code_postal": 7800,
      "ville": "marseille",
      "pays": "France"
    }
  ]
}
```

# Imbrication des documents

## Relation n-n (Livre auteur)



### 1- soit créer la collection auteur et imbriquer les livres

#### collection auteur:

```
{ "id": "a1", "nom": "Soutou", "prenom": "christian",
  Livres: [ { "isbn": "2212320434", "titre": "Sql pour Oracle" },
             { "isbn": "1234567", "titre": "modélisation des bases de données" } ]
{ "id": "a2", "nom": "Bruchez", "prenom": "Rudi",
  livres: [ { "isbn": "2212141556", "titre": "Les bases de données NoSQL et le big data" } ]
...
}
```

### 2- soit créer la collection livre et imbriquer les auteur

# Imbrication des données

En général, utilisez l'imbrication dans les cas suivants :

- les données imbriquées sont une partie **intégrante** des données d'un document.
- Il existe des relations de type **contient** entre des entités.
- Il existe des relations de type **un-à-plusieurs** entre des entités par contre les données incorporées **changent rarement**.
- Des données incorporées ne croîtront pas **sans limite**.

# Imbrication des données

Ne pas utiliser quand :

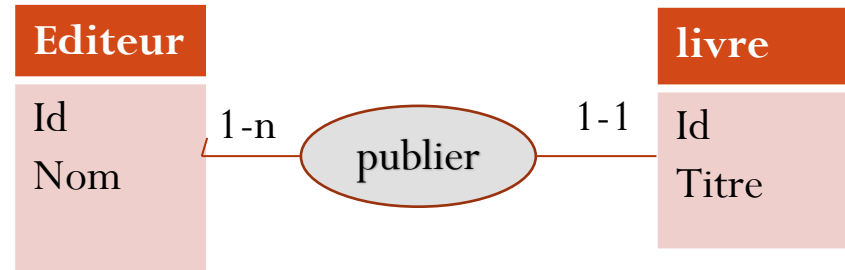
- Les données imbriquées sont illimité: L'augmentation de la taille du document a une incidence sur les possibilités de lecture et de mise à jour du document, à l'échelle.
- les données imbriquées sont souvent utilisées dans d'autres documents et changent fréquemment.

Dans ce cas privilégier la référence



# Référence des documents

## Relation 1-n (Livre Editeur)



collection Editeur:

```
{
  "id": "mspress",
  "nom": "Microsoft Press",
  "livres": [ 1, 2, 3, ..., 100, ..., 1000 ]
}
```

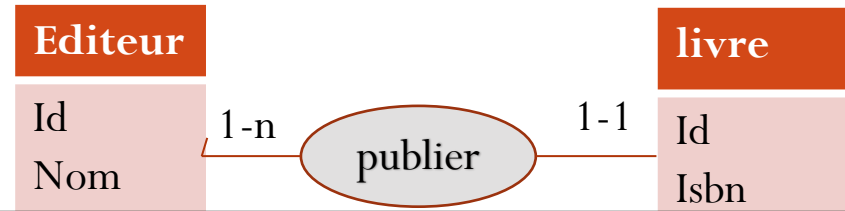
Collection livres:

```
{ "id": "1", "nom": "Azure Cosmos DB 101", "pub-id": "mspress" }
{ "id": "2", "nom": "Azure Cosmos DB for RDBMS Users", "pub-id": "mspress" }
{ "id": "3", "nom": "Taking over the world one JSON doc at a time" }
...
{ "id": "100", "nom": "Learn about Azure Cosmos DB", "pub-id": "mspress" }
...
{ "id": "1000", "nom": "Deep Dive in to Azure Cosmos DB", "pub-id": "mspress" }
```

**C'est intéressant de garder les livres par éditeur, mais attention aux tableaux de références de grande taille illimitée !**

# Référence des documents

## Relation 1-n (Livre Editeur)



La croissance de la relation permet de déterminer dans quel document doit être stockée la référence

**Rq:** On peut garder seulement la collection livre en imbriquant l'éditeur si ce dernier n'est pas important dans notre application (l'application est centrée livres pas d'autres informations que son nom!!)

collection Editeur:

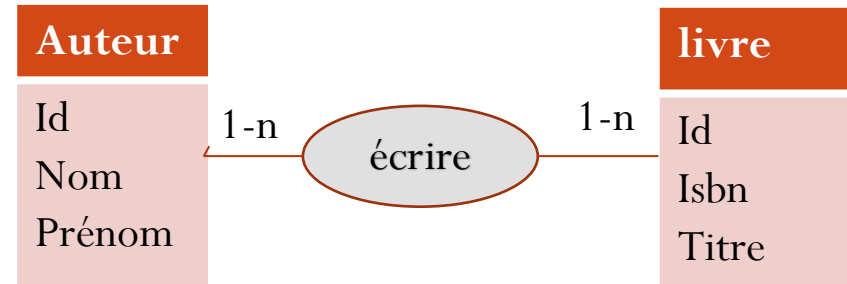
```
{ "id": "mspress",
  "nom": "Microsoft Press",
  "livres": [1, 2, 3, ..., 100, ..., 1000]
}
```

Collection livres:

```
{ "id": "1", "name": "Azure Cosmos DB 101", "pub-id": "mspress", "Editeur": "Microsoft Press" }
{ "id": "2", "name": "Azure Cosmos DB for RDBMS Users", "pub-id": "mspress" }
{ "id": "3", "name": "Taking over the world one JSON doc at a time" }
...
{ "id": "100", "name": "Learn about Azure Cosmos DB", "pub-id": "mspress" }
...
{ "id": "1000", "name": "Deep Dive in to Azure Cosmos DB", "pub-id": "mspress" }
```

# Référence des documents

## Relation n-n (Livre auteur)



### 1- générer un modèle qui ressemble au relationnel

#### collection auteur:


```
{ "id": "a1", "nom": "Soutou", "prenom": "christian" }
{ "id": "a2", "nom": "Bruchez", "prenom": "Rudi" }
{ "id": "a3", "nom": "Andersen", "prenom": "Thomas" }
```

#### collection livre:

```
{ "id": "b1", "isbn": "2212320434", "titre": "Sql pour Oracle" }
{ "id": "b2", "isbn": "2212141556", "titre": "Les bases de données NoSQL et le big data" }
{ "id": "b3", "titre": "Azure Cosmos DB for RDBMS Users" }
{ "id": "b4", "titre": "Taking over the world one JSON doc at a time" }
```

#### collection livre\_auteur:

```
{ "authorId": "a1", "bookId": "b1" }
{ "authorId": "a2", "bookId": "b2" }
{ "authorId": "a3", "bookId": "b4" }
{ "authorId": "a3", "bookId": "b3" }
```

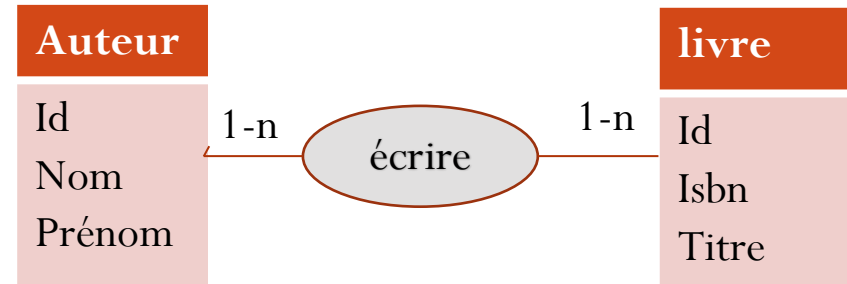
Déconseillée si   
l'application n'est pas  
centrée sur la relation

- le fait de charger soit un auteur avec ses livres soit un livre avec son auteur nécessite toujours **au moins deux requêtes supplémentaires** sur la base de données.
- solution non optimale car on n'a pas dénormaliser

Supprimer la collection `livre_auteur` → Dans quel document placer la référence (Auteur, Livre ou les deux)?

# Référence des documents

## Relation n-n (Livre auteur)



## 2- Un tableau de référence dans les deux collections

### collection auteur:

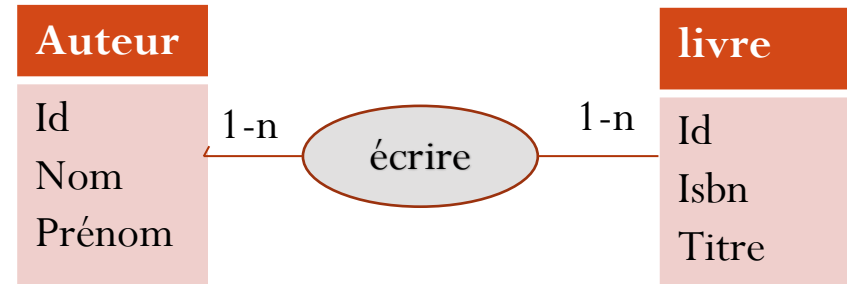
```
{ "id": "a1", "nom": "Soutou", "prenom": "christian", "livre": [ "b1", "b4" ] }  
{ "id": "a2", "nom": "Bruchez", "prenom": "Rudi", "livre": [ "b2" ] }  
{ "id": "a3", "nom": "Andersen", "prenom": "Thomas", "livre": [ "b3" ] }  
{ "id": "a4", "nom": "Brouard", "prenom": "Frédéric", "livre": [ "b4" ] }
```

### collection livre:

```
{ "id": "b1", "isbn": "2212320434", "titre": "Sql pour Oracle", "auteur": [ "a1" ] }  
{ "id": "b2", "titre": "Les bases de données NoSQL et le big data", "auteur": [ "a2" ] }  
{ "id": "b3", "titre": "Azure Cosmos DB for RDBMS Users", "auteur": [ "a3" ] }  
{ "id": "b4", "isbn": "2212141556", "titre": "Modélisation des bases de données", "auteur": [ "a1", "a4" ] }
```

# Solution hybride

## Relation n-n (Livre auteur)



### 3- Un tableau d'objets : références + quelques informations de la table référencée

**Exemple** de l'application à développer (centrée livre)

Les bases de données **NoSQL** et le BigData

Comprendre et mettre en oeuvre

Editeur(s) : Eyrolles  
Date de parution : 24/04/2015  
Auteur(s) : Rudi Bruchez

En savoir plus sur l'auteur

[Consultez la page Rudi Bruchez](#)

- Utiliser le tableau de référence seulement dans la collection livre et faire apparaitre quelques informations nécessaires de la table auteur et au besoin afficher le détail

# Solution hybride

## Relation n-n (Livre auteur)

### 3- Un tableau d'objets : références + quelques informations

#### collection livre:

```
{  "id": "b2",  
  "name": "Les bases de données NoSQL et le big data",  
  "isbn": "", "date_parrution": "24/04/2015",  
  "couverture_livre": "http://....png"  
  "authors": [  
    {"id": "a2", "name": "Rudi_Bruchez", "thumbnailUrl": "http://....png"}  ]  
}
```

#### collection auteur:

```
{  "id": "a2", "nom": "Bruchez", "prenom": "Rudi", "fonction": "consultant",  
  "interet": ["bases de données"], "adresse": {"rue": ,..}  
  "Image_Auteur": "http://....png"  
}
```

# Solution hybride

## Relation n-n (livre auteur)

### 3- Un tableau d'objets : références + quelques informations

Exemple de l'application à développer (application centrée sur la relation)

	<p>Les bases de données NoSQL et le Big Data: Comprendre et mettre en oeuvre. 24 avril 2015 de Rudi Bruchez</p> <p>Broché <b>EUR 32,00</b> ✓prime Plus que 7 ex. Commandez vite ! Plus de choix d'achat <b>EUR 32,00</b> (6 d'occasion &amp; neufs)</p> <p>Format Kindle <b>EUR 21,99</b></p> <p>★★★★☆ 4</p>
	<p>Bases de données NoSQL et Big Data : Concevoir des bases de données pour le Big Data - Cours et travaux pratiques 2 décembre 2014 de Philippe Lacomme et Sabeur Aridhi</p> <p>Broché <b>EUR 37,00</b> ✓prime Plus que 8 ex. Commandez vite ! Plus de choix d'achat <b>EUR 37,00</b> (4 d'occasion &amp; neufs)</p>

- Faire apparaître la table association avec quelques informations nécessaires et au besoin afficher le détail soit de l'auteur soit du livre

# Solution hybride

## Relation n-n (Livre auteur)

### 3- Un tableau d'objets : références + quelques informations

Exemple de l'application a développer

#### Détails sur le produit

**Broché:** 322 pages

**Editeur :** Eyrolles; **Édition :** 2 (24 avril 2015)

**Collection :** Blanche

**Langue :** Français

**ISBN-10:** 2212141556

**ISBN-13:** 978-2212141559

**Dimensions du produit:** 22,7 x 2 x 19 cm

**Moyenne des commentaires client :** ★★★★★ 4 commentaires client

**Classement des meilleures ventes d'Amazon:** 27.544 en Livres (Voir les 100 premiers en Livre n°7 dans Livres > Informatique et Internet > Bases de données

n°67 dans Livres > Informatique et Internet > Programmation et langages

#### En savoir plus sur l'auteur

› Consultez la page Rudi Bruchez d'Amazon



#### Biographie

Consultant informatique inc SQL, ainsi que des services :

+ Suivre

- Dans ce cas, comme dans le relationnel: utilisation des trois collections. Les collections: auteur et livre (sans ou avec références selon l'application ) et la table d'association (référence + les informations à faire apparaître dans l'application ). Au besoin aller au détail des deux



# Solution hybride

## Relation n-n (Livre auteur)

### 3- Un tableau d'objets : références + quelques informations

#### collection livre\_auteur (qq infos du livre + qq infos auteur):

```
{  "couverture_livre": "http://....png"
  "titre": "Les bases de données NoSQL et le big data",
  "date_parrution": "24/04/2015",
  "détail_livre": "b2",
  "authors": [
    { "id": "a2", "name": "Rudi_Bruchez" }
  ]
}
```

#### collection auteur:

```
{
  "id": "a2", "nom": "Bruchez", "prenom": "Rudi", "fonction": "consultant", "interet": ["bases de données"],
  "adresse": { "rue": "76 b bd barbes", ... }, "Nombre_livre": 7,
  "Image_Auteur": "http://....png", "livres": ["b2", ...],
}
```

#### collection livre

```
{ "id": "b2",
  "titre": "Les bases de données NoSQL et le big data", "Broché": 322, "Editeur": "Eyrolles";
  "Collection": "Blanche", "Langue": "Français", "ISBN-10": "2212141556"
  "ISBN-13": "978-2212141559", "Dimensions du produit": "22,7 x 2 x 19 cm" }
```

Pas nécessaire à ajouter seulement si à partir de l'auteur les livres sont affichés

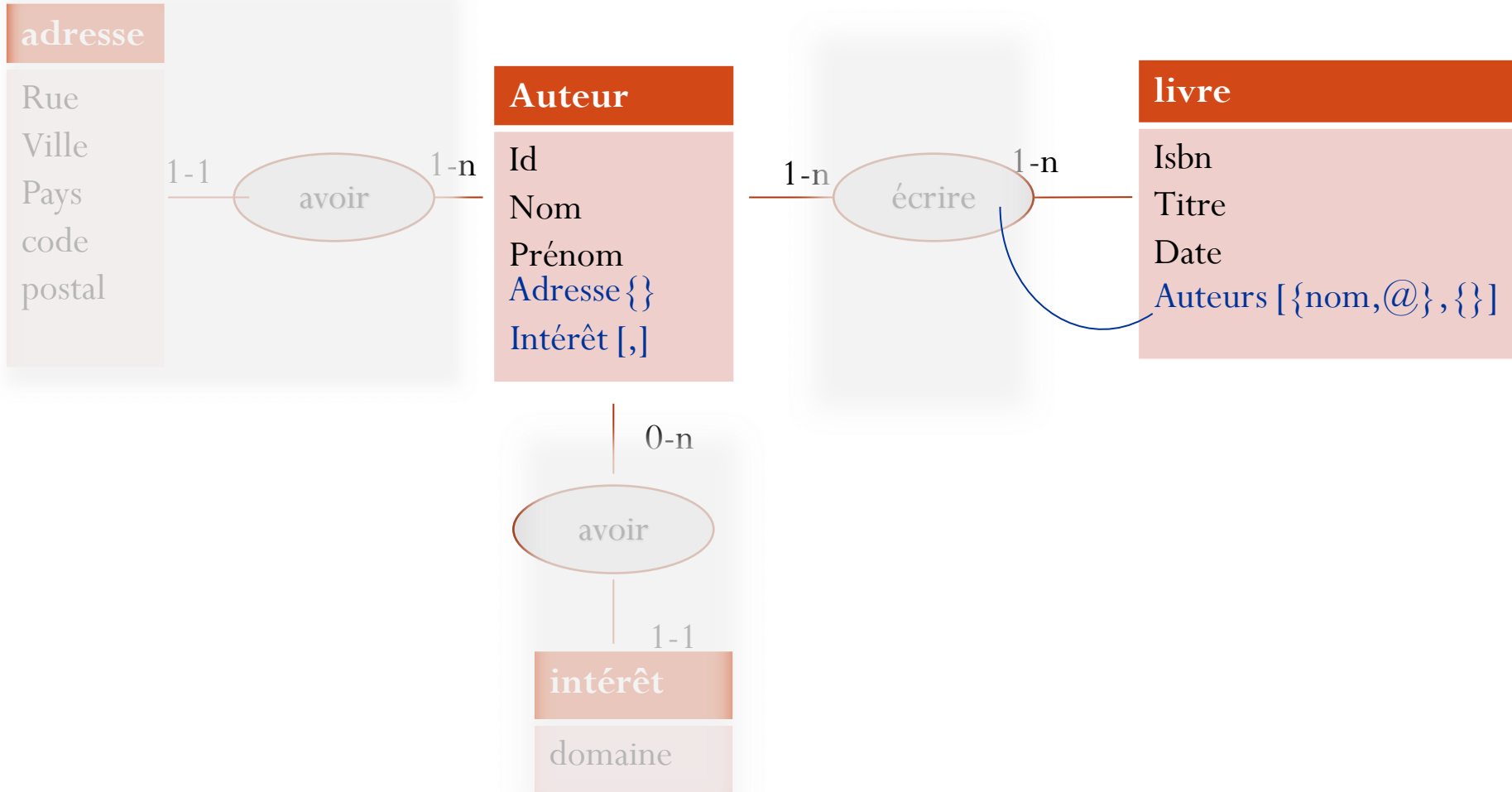
# Modélisation

- la modélisation des données dans un monde sans schéma reste importante (**pas de schéma  $\neq$  pas de modélisation**)
- A retenir :
  - la normalisation offre de meilleures performances en **écriture**  
mais **peut nécessiter davantage d'aller-retour** pour lectures
  - les modèles de données dénormalisés offrent de meilleures performances en **lecture**
  - Les solutions hybrides offrent un bon compromis entre dénormalisation et redondances de données.

# Modélisation

- il existe plusieurs façons de représenter un élément de données sur un écran, il existe plusieurs manières de modéliser vos données.
- Il faudra:
  - comprendre votre application et comment elle produira, utilisera et traitera les données (documents).
  - Quelles sont les données fréquemment interrogées conjointement.
  - Les données qui sont le moins dépendantes des autres
  - Le taux de mise à jours de vos données

## Retour a notre exemple



# Commandes de base CRUD Mongodb

## Lancer le serveur

- créer un répertoire de stockage de données: **C:\data\db**
- Lancer le serveur **mongod**
- Lancer le shell mongo client **mongo**
- Le serveur MongoDB est organisé en plusieurs *databases*

# Commandes de création du catalogue (base,collection)

- connaître les Bases stockées dans le répertoire géré par le serveur

```
> show dbs
```

```
Films      0.000GB
```

```
GuideResto 0.000GB
```

- Afficher le nom de la base actuelle

```
> db
```

```
film
```

- Basculer à la base biblio si cette dernière n'est pas créée elle le sera à la première création de collection

```
> use biblio
```

```
switched to db biblio
```

- Création de la collection:

```
> db.createCollection("livres");
```

```
{ "ok" : 1 }
```

- Visualisation des collections : afficher les noms des collections qui ont été créées

```
> show collections
```

```
livres
```

## Insérer les données (documents dans une collection)

Il est possible d'insérer directement le document ou de le définir puis de l'insérer

```
> document = ({ "isbn" : "9782212134131", "titre" : "Modélisation des bases de données",  
"auteur" : [ "Christian Soutou", "Frédéric Brouard" ] })  
{  
  "isbn" : "9782212134131",  
  "titre" : "Modélisation des bases de données",  
  "auteur" : [  
    "Christian Soutou",  
    "Frédéric Brouard"  
  ]  
}
```

Insérer le document dans la collection livre

```
> db.livres.insert (document);  
WriteResult({ "nInserted" : 1 })
```

**Rq:** si la collection livres n'existait pas déjà elle sera créée.



## Insérer les données (documents dans une collection)

Insérer un seul document dans la collection livre

```
> db.livres.insertOne(document);  
{  
  "acknowledged" : true,  
  "insertedId" : ObjectId("5a99bded567089e00a38a0e4")  
}
```

Insérer plusieurs documents dans la collection restaurant

```
> db.livres.insertMany([{"isbn" : "2212141556", "titre" : "Les bases de  
données NoSQL et le big data", "auteur" : ["Rudi Bruchez"] }, {"isbn" :  
"208134761X", "titre" : "Maktub", "auteur" : ["Paulo Cohelo"]}])  
  
{  
  "acknowledged" : true,  
  "insertedIds" : [  
    ObjectId("5a99c645f7b45ad721c4a5f5"),  
    ObjectId("5a99c64df7b45ad721c4a5f7")  
  ]  
}
```

# Afficher les données des documents(find)

Pour afficher tous les documents de la collection

```
> db.livres.find()
{ "_id" : ObjectId("5a99bc34f7b45ad721c4a4a9"), "isbn" : "2081399253",
  "titre" : "l'Alchimiste", "auteur" : [ "Coelho" ] }
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" : "9782212134131",
  "titre" : "Modélisation des bases de données", "auteur" : [ "Christian
Soutou", "Frédéric Brouard" ] }
```

Pour afficher un seul document

```
> db.livres.findOne()
```

# Afficher des données

Pour afficher tous les documents de la collection

```
> db.livres.find()
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" : "9782212134131",
  "titre" : "Modélisation des bases de données", "auteur" : [ "Christian
Soutou", "Frédéric Brouard" ] }
{ "_id" : ObjectId("5a99c645f7b45ad721c4a5f5"), "isbn" : "2212141556",
  "titre" : "Les bases de données NoSQL et le big data", "auteur" : [ "Rudi
Bruchez" ] }
{ "_id" : ObjectId("5a99c64df7b45ad721c4a5f7"), "isbn" : "208134761X",
  "titre" : "Maktub", "auteur" : [ "Paulo Cohelo" ] }
```

Pour afficher un seul document

```
> db.livres.findOne()
```

## Mise à jour d'un enregistrement (Update)

## Remplacer complètement un document

```
> db.livres.find()
```

```
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" : "9782212134131",  
  "titre" : "Modélisation des bases de données", "auteur" : [ "Christian Soutou",  
    .....  
  ] }
```

```
> db.livres.update({titre:"Modélisation des bases de données"},{annee:2012})
```

```
WriteResult( { "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 } )
```

```
> db.livres.find();
```

```
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "annee" : 2012 }
{ "_id" : ObjectId("5a99c645f7b45ad721c4a5f5"), "isbn" : "2212141556", "titre"
: "Les bases de données NoSQL et le big data", "auteur" : [ "Rudi Bruchez" ] }
{ "_id" : ObjectId("5a99c64df7b45ad721c4a5f7"), "isbn" : "208134761X",
"titre" : "Maktub", "auteur" : [ "Paulo Cohelo" ] }
```

**Attention : On a écrasé tout l'enregistrement ! mais l'Id est resté le même**

# Mise à jour d'un enregistrement (Update)

On corrige l'enregistrement, en rajoutant les champs :

```
>db.livres.find();
```

```
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "annee" : 2012 }  
{ "_id" : ObjectId("5a99c645f7b45ad721c4a5f5"), "isbn" : "2212141556", "titre"  
: "Les bases de données NoSQL et le big data", "auteur" : [ "Rudi Bruchez" ] }  
{ "_id" : ObjectId("5a99c64df7b45ad721c4a5f7"), "isbn" : "208134761X",  
"titre" : "Maktub", "auteur" : [ "Paulo Coelho" ] }
```

```
>db.livres.update({_id:ObjectId("5a99bded567089e00a38a0e4")},{ "isbn":  
"9782212134131", "titre" : "Modélisation des bases de données", "annee":2012,  
"auteur" : [ "Christian Soutou", "Frédéric Brouard" ] })  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.livres.find({titre:"Modélisation des bases de données"})  
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "titre" : "Modélisation des  
bases de données", "annee" : 2012, "isbn" : "9782212134131", "auteur" : [  
"Christian Soutou", "Frédéric Brouard" ] }
```

## Mise à jour d'un enregistrement (Update)

Modifier et/ou rajouter un champ au document (opération **\$set**)

```
>db
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.livres.find({titre:"Modélisation des bases de données"})
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" : "9782212134131", "titre" :
"Modélisation des bases de données", "annee" : 2011, "auteur" : [ "Christian Soutou",
"Frédéric Brouard" ], "avis" : "excellent", "genre": " informatique " }
```

Ajouter et/ou modifier un champ à plusieurs documents (**multi**)

```
>db.livres.update({auteur: "Rudi Bruchez"},{$set: { genre:
"informatique" }},{multi:1})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
```

Ajouter et/ou modifier un champ à toute la collection

```
>db.livres.update({}, {$set: { langue: "Français" }},{multi:1})
WriteResult({ "nMatched" : 3, "nUpserted" : 0, "nModified" : 3 })
```

## Mise à jour d'un enregistrement (Update)

Pour supprimer un champ, il suffit de remplacer par l'opérateur **\$unset**.

```
> db.livres.update({titre:"Modélisation des bases de données"},
{$unset: {avis: "" }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

> db.livres.find({titre:"Modélisation des bases de données"})
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" :
"9782212134131", "titre" : "Modélisation des bases de données", "annee"
: 2012, "auteur" : [ "Christian Soutou", "Frédéric Brouard" ], "genre":
" informatique " ,"date" : 2011 }
```

Pour renommer un champ, l'opérateur **\$rename**.

```
>db.livres.update({titre:"Modélisation des bases de données"},{$rename
: {"avis":"commentaire"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
```

# Mise à jour d'un enregistrement (Update)

Mettre à jour un tableau :

modifier une valeur d'un élément d'un tableau : la notation pointée

**Rq:** l'utilisation d'un **\$set** sur le tableau, va remplacer le tableau existant par un nouvel élément (écraser les ancienne valeurs). Si on veut modifier un élément précis d'un tableau il faudra utiliser la notation pointée

```
>db.livres.update({titre:"Modélisation des bases de données"},  
{$set:{"auteur.0" : "Guardarin"}})
```

```
> db.livres.find({titre:"Modélisation des bases de données"})  
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" :  
"9782212134131", "titre" : "Modélisation des bases de données", "annee"  
: 2012, "auteur" : ["Guardarin", "Frédéric Brouard"], "date" : 2011,  
"avis" : "excellent", "genre" : "informatique", "langue" : "Français"} }
```



# Mise à jour d'un enregistrement (Update)

Mettre à jour un tableau :

ajouter à la fin d'un tableau avec **\$push**

```
> db.livres.update({titre: "Modélisation des bases de données"},  
{$push: {auteur: "Frédéric Brouard"}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.livres.find({titre: "Modélisation des bases de données"})  
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" : "9782212134131",  
"titre" : "Modélisation des bases de données", "annee" : 2012, "auteur" :  
["Guardarin", "Frédéric Brouard", "Frédéric Brouard"], "date" : 2011, "avis" :  
"excellent ", "genre" : "informatique", "langue" : "Français" }
```

**Attention:** l'auteur a été dupliqué, si on veut rajouter à la fin sans doublon, il faudra utiliser l'opérateur **\$addToSet**

**Remarque :** si le tableau n'existe pas alors il le crée avec l'élément mentionné

# Mise à jour d'un enregistrement (Update)

Mettre à jour un tableau :

supprimer une valeur avec l'opérateur **\$pull**

```
> db.livres.update({titre:"Modélisation des bases de données"},  
{$pull:{auteur:"Frédéric Brouard"}})  
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.livres.find({titre:"Modélisation des bases de données"})  
{ "_id" : ObjectId("5a99bded567089e00a38a0e4"), "isbn" : "9782212134131",  
  "titre" : "Modélisation des bases de données", "annee" : 2012, "auteur" : [  
    « guardarin » ], "date" : 2011, "avis" : "excellent ", "genre" : "informatique",  
  "langue" : "Français" }
```

D'autres commandes sur les tableaux voir la documentation officielle de Mongo

<https://docs.mongodb.com/manual>

## supprimer les données d'une collection(remove)

Pour supprimer un document de la collection

```
> db.livres.remove({titre:"Modélisation des bases de données"})  
WriteResult({ "nRemoved" : 1 })
```

Pour supprimer tous les documents de la collection: vider la collection

```
> db.livres.remove({})  
WriteResult({ "nRemoved" : 1 })
```

Pour supprimer la collection entière

```
> db.livres.drop()  
true
```

# Supprimer la base

```
> show dbs
Films      0.000GB
GuideResto 0.000GB
biblio 0.000GB
> db
biblio
> db.dropDatabase()
{ "dropped" : " biblio", "ok" : 1 }
> show dbs
Films      0.000GB
GuideResto 0.000GB
> db
biblio
```

On efface la base courante  
**Attention** : la base est effacée sur disque mais continue d'être référéncée (ne pas oublier de passer à une autre base: `use`)

# Importation de documents

Importation de documents au format JSON

On utilise l'outil **mongoimport** : **n'est pas une commande** du mongo shell

```
mongoimport --db dbName --collection collectionName  
--mode importMode --file fileName.json --jsonArray
```

- **Le démon mongod doit être lancé, et mongoimport s'y connectera**
- L'option **--mode** permet de préciser si on ajoute/mélange/remplace les données déjà présente dans la collection
- L'option **--jsonArray** permet d'importer des tableaux de documents JSON
- La commande **mongoimport** est riche en options de fonctionnement (voir la doc technique de MongoDB)

**Exemple: Ouvrir le shel de windows CMD**

```
C:\Program Files\MongoDB\Server\3.6\bin>mongoimport -d biblio -c auteur --file  
"E:\personnes.json"
```

**Importe le contenu du fichier JSON personnes.json, dans la base biblio dans la collection auteur et les ajoute aux données déjà présentes.**

# Exportation de documents

Exportation de documents au format JSON.

On utilise l'outil **mongoexport** : n'est pas une commande du mongo shell

```
mongoexport --db dbName --collection collectionName  
--out fileName.json --query theQuery
```

**Le démon mongod doit être lancé, et mongoexport s'y connectera**

- Tout fichier cible préexistant est écrasé (pas de *merge à l'écriture*).
- On peut spécifier un filtrage sur les enregistrements avec l'option **-query**, et ne sauvegarder que ceux satisfaisant certains critères

**Exemple: Ouvrir le shel de windows CMD**

```
C:\Program Files\MongoDB\Server\3.6\bin>mongoexport --db biblio --  
collection livres --out "E:\livres.json" --query {genre : "informatique"}
```

Exporte le contenu de la collection livre, de la base biblio, dans le fichier livre.json  
On ne sauvera que les enregistrements dont le genre est 'informatique'.

# **Cours prochain**

## **Interrogation des données**