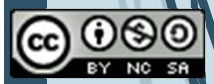
MongoDB – Manipulations



Faiza Ghozzi Jedidi & Sahar Kallel

2 Histoire

- **Hu*mongo*us (monstre / énorme)**
- **MongoDB, un système "NoSQL", l'un des plus populaires**
  - Orienté Documents (avec CouchDB)
  - Modèle de données semi-structuré (encodage JSON) ;
  - Documents sérialisés : BSon objects
  - Implémenté en C++
  - pas de schéma (complète flexibilité) ;
  - un langage d'interrogation original (et spécifique) ;
  - Indexation d'attributs (BTree)
  - pas (ou très peu) de support transactionnel.

3 Histoire

- **MongoDB, construit dès l'origine comme un système scalable et distribué.**
  - distribution par partionnement (sharding) ;
  - technique adoptée : découpage par intervalles (type BigTable, Google) ;
  - Stockage : *GridFS*
  - tolérance aux pannes par réplication (*Replica Set*)

4 Motivations

- **Problèmes avec SQL**
  - Schéma rigide
  - Scalabilité difficile (Conçu pour les technologies des années 90)
  - Nécessitent des jointures non intuitives.

- **Avantages de mongoDB**
  - Interface facile avec les langages commun (Java, Javascript, PHP, etc.)
  - Technologies BD peut s 'exécuter partout (VM's, cloud, etc.)
  - Garde les fonctionnalités essentielles des SGBDR tout en interrogeant à partir des systèmes NoSQL clé-valeur.

5

Les compagnies utilisant MongoDB

mongoDB

Modèle de données

- **Document-Based (max 16 MB)**
- **Les documents sont sous le format BSON, consistant en des paires champs-valeur**
- **Chaque document est stocké dans une collection**
- **Collections**
  - Des ensembles d'index en commun
  - Correspondent aux tables relationnelles.
  - Les documents n'ont pas une structure uniforme

-docs.mongodb.org/manual/

7 JSON

- "JavaScript Object Notation"
- Easy for humans to write/read, easy for computers to parse/generate
- Objects can be nested
- Built on
  - name/value pairs
  - Ordered list of values

- http://json.org/

8 BSON

- "Binary JSON"
- Binary-encoded serialization of JSON-like docs
- Also allows "referencing"
- Embedded structure reduces need for joins
- Goals
  - Lightweight
  - Traversable
  - Efficient (decoding and encoding)

http://bsonspec.org/

9 Exemple

```
{
"_id" :     "37010"
"city" :    "ADAMS",
"pop" :    2660,
"state" :  "TN",
“councilman” : {
            name: “John Smith”
            address: “13 Scenic Way”
          }
}
```

10 BSON Types

Type	Numéro
Double	1
String	2
Object	3
Array	4
Binary data	5
Object id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

Le numéro peut être utilisé avec l'opérateur $type pour interroger en fonction du type.

11

The _id Field

- **Par défaut, chaque document contient un champ _id. Ce champ a quelques caractéristiques :**
  - Valeurs comme des clés primaires pour la collection.
  - Valeur unique, immutable, et peut être any non-array type.
  - Le type par défaut est ObjectId, qui est "small, likely unique, fast to generate, and ordered."
  - Trier en function du ObjectId est équivalent à trier en fonction du temps de creation.

12

mongoDB vs. SQL

mongoDB	SQL
Document	Tuple
Collection	Table/View
PK: _id Field	PK: Any Attribute(s)
Uniformity not Required	Uniform Relation Schema
Index	Index
Embedded Structure	Joins
Shard	Partition

CRUD

Create, Read, Update, Delete

14

Installation MongoDB

Pour installer mongoDB, allez sur ce lien et choisir le système d'exploitation approprié:
http://www.mongodb.org/downloads

Tout d'abord, extraire les fichiers (de préférence vers le C drive).

Enfin, créer les répertoires dans C:\ pour mongoDB

"md data"

"md data\db"

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/

15

Débuter avec MongoDb

- Ouvrir le répertoire mongodb/bin et exécuter mongod.exe pour lancer le serveur de base de données.

- Pour se connecter au serveur, ouvre un autre prompt window et met toi sur le même répertoire, lance mongo.exe.

- http://docs.mongodb.org/manual/tutorial/getting-started/

16

CRUD: Utiliser le Shell

- Pour vérifier la base auquelle tu es connecté **db**
- Afficher les bases de données **show dbs**
- Connecter a une BD ou créer une nouvelle **use <name>**
- Afficher la liste des collections **show collections**

- Note: db's are not actually created until you insert data!

17

Notions

- **Base de données**
  - Commande : use maBD ;
- **Collections de documents**
  - Création : db.createCollection('users');
  - Utilisation : db.users. <commande> ;
    - Commandes : find(), save(), insert(), delete(), update(), aggregate(), distinct(), mapReduce()…
  - Equivalent SQL : **FROM**

18 Authentication

- How to create user in database?

```
$ use paris
$ db.createUser({
  "user": “myuser",
  "pwd": “myuserpw",
  "roles": ["readWrite", "dbAdmin"]
})
```

- Database user roles:
  - read, readWrite, dbAdmin, dbOwner, userAdmin
  - userAdminAnyDatabase
- Login:

```
db.auth(“myuser", “myuserpw")
```

19

Notions

- **Documents**
  - Pas de schéma
  - Json :

```
{
"_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
"name": "James Bond",
"login": "james",
"age": 50,
"address": {"street": "30 Wellington Square",  "city": "London"},
"job" : ["agent", "MI6"]
}
```

CRUD: Insertion de données

```
db.<collection>.insert(<document>)
<=>
INSERT INTO <table>
VALUES(<attributevalues>);
```

db.<collection>.insert({<field>:<value>})

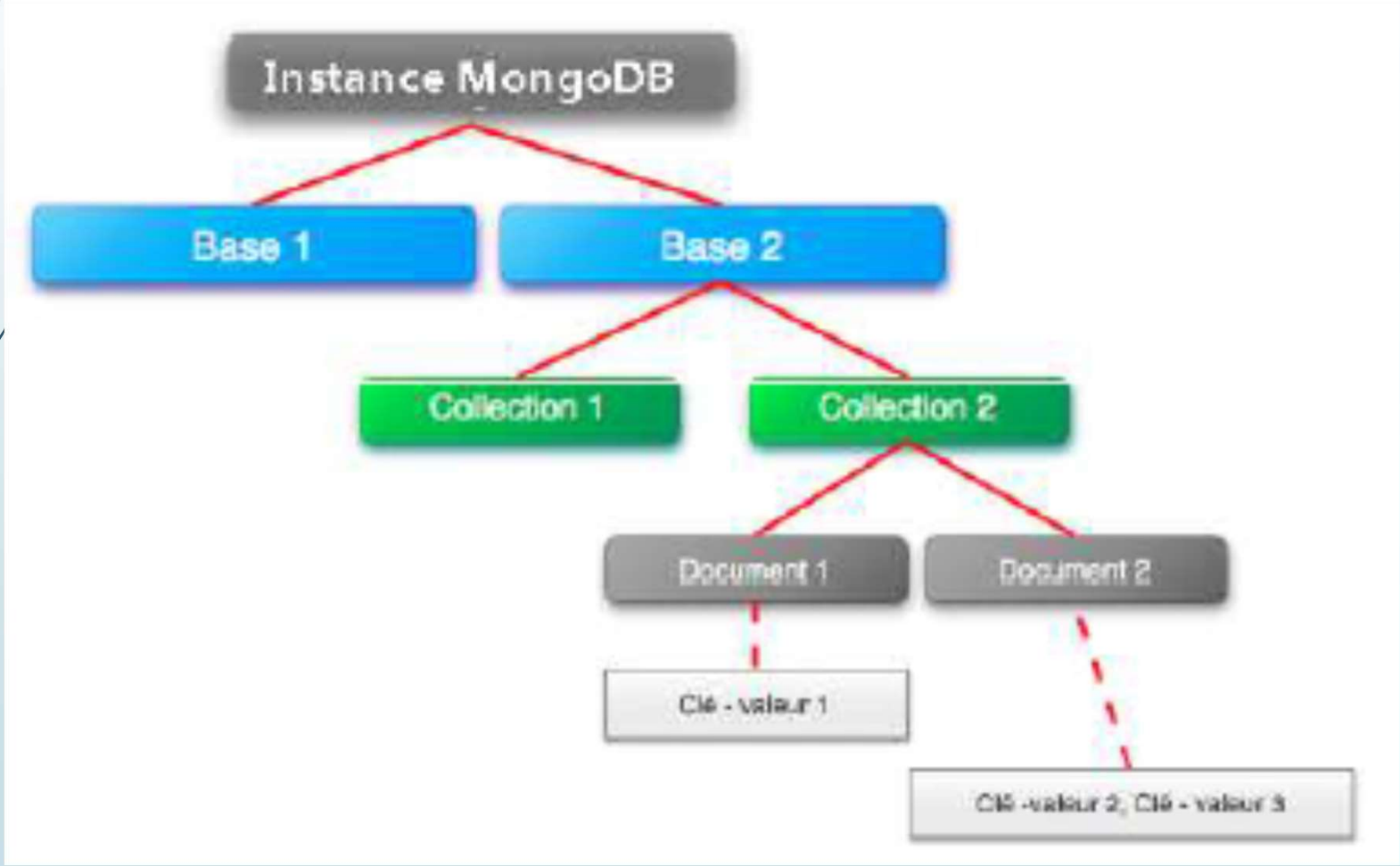
- Insérer un document
- Insérer un document avec un nom de champ nouveau pour la collection supporté par le modèle BSON.

db.<collection>.insertMany([….]}

- Pour insérer des documents multiples, utiliser un tableau.

CRUD: Insertion de données

- Rappel sur la hiérarchie des documents



CRUD: Insertion de données

- Exemple :

```
> db.films.insert({"titre": "Oblivion"})
```

Il existe donc **un objet (javascript)** implicite "db" auquel on soumet des demandes d'exécution de certaines méthodes.

CRUD: Insertion de données

- **Pour insérer un autre document :**

```
> db.films.insert({"produit":"Grulband", "prix":230,
                   "enStock":true})
```

  - La structure de ce document n'a rien à voir avec le précédent : pas de schéma imposé (et donc pas de contrainte) dans MongoDB.
  - Ce qui revient à reporter les problèmes (contrôles, contraintes, tests sur la structure) vers l'application.
- **On peut affecter un identifiant explicitement**

```
> db.films.insert({"_id":"1", "produit":"Kramölk", "prix":10,
                   "enStock":true})
```

CRUD: Insertion de données

```
> db.films.insert({"_id": 1, "title": "Jurassic Park"})
```

```
> db.films.save({"_id": 1, "title": "Jurassic Park"})
```

- **Le comportement diffère quand l'identifiant "_id" est passé**
  - Pour **insert**, si un document de la collection aurait le même identifiant "_id", une erreur sera lancée
  - Pour **save**, si un document de la collection avait le même identifiant "_id", il serait mis à jour par les nouvelles valeurs.

CRUD: Insertion de données

- **Insertion :** (utilisation de javascript)

```
> collection = db.contactinfo
> doc = { "nom" : "toto",
        "info" : {
         "twitter" : "@toto5646",
         "email" : "toto@upmc.fr"
         },
        "amis" : 87687,
        "date_login" : new Date()
      }
> collection.insert(doc)
```

Importation et Exportation des documents

- Importer les données JSON avec **mongoimport** :

```
> mongoimport -d TestDB -c films --file films.json --jsonArray
```

```
> mongoimport --db TestDB --collection films --file films.json --jsonArray
```

- Exporter les données d'une collection avec **mongoexport** :

```
> mongoexport -d TestDB -c films --type csv --fields
_id,year,title,summary,director,genre,actors --out data.csv
```

```
> mongoexport -d TestDB -c films --type json --out data.json
```

Exercice 1

- Placez-vous sur une base ADBD.
- Créez une collection « test ».
- Insérez un document dans la collection contenant vos données.
- Affichez le contenu de la collection « test ».
- Comptez le nombre de documents dans la collection.

Exercice 1 (solution)

- Placez-vous sur une base TSD.

```
> use ADBD
```

- Créez une collection « test ».

```
> db.createCollection("test")
```

- Insérez un document dans la collection contenant vos données.

- Affichez le contenu de la collection « test ».

- Comptez le nombre de documents dans la collection.

CRUD: Interrogation

- **Filtrage simple**
  - Appliqué sur les collections.
  - Récupérer **tous les docs**: **db.<collection>.find()**
    - Retourne un curseur qui affiche itérativement sur le shell les 20 premiers résultats.
  - Récupérer **un seul doc**:

    db.<collection>.findOne()

30

CRUD: Interrogation

- **Filtrage simple** `db.users.find( <filtre> , <projection> );`
  - **Filtre** :
    - Motif "JSon" qui doit être contenu dans le document
    - Format "clé/valeur"
    - Peut contenir des valeurs exactes, des opérations, des imbrications, des tableaux
    - Opération : $op (pas de guillemets)
    - Equivalent SQL : *WHERE*
  - **Projection** :
    - "Clé/valeur" à retourner (les autres sont supprimées)
    - Equivalent SQL : *SELECT (sans agrégat)*

CRUD: Interrogation

- Recherche exacte

```
db.users.find( { "login" : "james" } , {"name" : 1, "age" : 1});
```

Dans une imbrication

```
db.users.find( { "address.city" : "London" } );
```

Avec une opération

```
db.users.find( { "age" : { $gt : 40 } } );
```

- $gt, $gte,
- $lt, $lte
- $ne, $in, $nin
- $or, $and,
- $size, …

CRUD: Interrogation

- Exclure un champ
  - db.<collection>.find({<field1>:<value>}, {<field2>: 0})
- Inclure un champ
  - db.<collection>.find({<field>:<value>}, {<field2>: 1})
- Trouver les documents avec ou sans le champ
  - db.<collection>.find({<field>: { $exists: true}})

Voici les opérateurs disponibles avec des exemples associés :

$gt, $gte	>, ≥	Plus grand que (*greater than*)	"a":{"$gt":10}
$lt, $lte	<, ≤	Plus petit que (*less than*)	"a":{"$lt":10}
$ne	≠	Différent de (*not equal*)	"a":{"$ne":10}
$in, $nin	∈, ∉	Fait parti de (ou ne fait pas)	"a":{"$in":[10, 12, 15, 18]}
$or	˅	OU logique	{"$or":[{"a":{"$gt":10}}, {"a":{"$lt":5}}]}
$and	^	ET logique	{"$and":[{"a":{"$lt":10}}, {"a":{"$gt":5}}]}
$not	¬	Négation	"a":{"$not":{"$lt":10}}
$exists	∃	La clé existe dans le document	"a":{"$exists":1}
$size		test sur la taille d'une liste (uniquement par égalité)	"a":{"$size":5}

CRUD: Interrogation

- Expression régulière
- Les expressions régulières (Regex) permettent de retrouver des chaines de caractères qui correspondent à un modèle.
- Les opérations suivantes utilisent des Regex :

```
> db.films.find({"title": /^Re/}, {"title": 1})
```

Commence par "Re"

```
> db.films.find({"title": /Re/i}, {"title": 1})
```

Contient "Re"

Insensible à la case

```
> db.films.find({"title": /Re$/i}, {"title": 1})
```

Fini par "Re"

https://docs.mongodb.com/manual/reference/operator/query/regex/

Manipulation du tableau

```
{
  "_id": 1,
  "name": "Alice",
  "hobbies": ["Lecture", "Peinture"]
},
{
  "_id": 2,
  "name": "Bob",
  "hobbies": ["Golf", "Tennis", "Cuisine"]
},
{
  "_id": 3,
  "name": "Charlie",
  "hobbies": ["Photographie"]
}
```

- db.users.find({ "hobbies": "Lecture" });
- db.users.find({ "hobbies": { $all: ["Tennis", "Golf"] } }); ➔ tous les utilisateurs ayant à la fois "Tennis" et "Golf" dans leur tableau "hobbies".
- db.users.find({ "hobbies": { $in: ["Tennis", "Photographie"] } }); ➔ tous les utilisateurs qui ont "Tennis" ou "Photographie" (ou les deux) dans leur tableau "hobbies".
- db.users.find({ "hobbies": { $size: 2 } }); ➔ tous les utilisateurs dont le tableau "hobbies" contient exactement deux éléments.

CRUD: Interrogation

- db.users.find({ " hobbies.1" : "Tennis"}); ➔ tous les utilisateurs ayant Tennis dans la deuxième place du tableau".
- db.users.find({ " hobbies" : ["Tennis"]}); ➔ tous les utilisateurs ayant exactement Tennis dans le tableau hobbies".

Manipulation des documents

- **Interrogation : Pagination**
- La méthode ***skip ()*** contrôle le point de départ de l'ensemble de résultats.
- L'opération suivante ignore les 2 premiers documents des films sorties en 2000 **(5)** et renvoie tous les autres documents **(3)**:

```
> db.films.find({"year":2005}).skip(2)
```

Manipulation des documents

- **Interrogation : Pagination**
- La méthode *limit ()* limite le nombre de documents dans l'ensemble de résultats.
- L'opération suivante renvoie au plus 5 documents dans la collection de films:

```
> db.films.find().limit(5)
```

Manipulation des documents

- **Interrogation : Ordonner le résultat**
- La méthode ***sort ()*** ordonne les documents dans l'ensemble de résultats. (Équivalent à OrderBy de SQL)
- L'opération suivante renvoie les titres des films de la collection films triés par ordre croissant par le champ de title :

```
> db.films.find({},{"title":1}).sort({"title":1})
```

```
1 : croissant
-1 : décroissant
```

Manipulation des documents

Interrogation : distinct

- La méthode ***distinct()*** recherche les valeurs distinctes d'un champ spécifié d'une collection et renvoie les résultats dans un tableau.

```
> db.collection.distinct(champ, [query])
```

- champ (string) : le champ pour lequel on va appliquer le distinct
- query (document) : optionnel, une requête pour filtrer les documents sur lesquels applique le distinct. Si non spécifié, le distinct s'applique sur toute la collection.

Manipulation des documents

- **Interrogation : distinct**
- Afficher les genres distincts de tous les films de la collection « films »

```
> db.films.distinct("genre")
```

- Afficher les noms de famille distincts des acteurs des films sorties en 1976.

```
> db.films.distinct("actors.last_name", {"year":1976})
```

Exercice

- **Dans la collection films de la base de données TestDB :**
  - Retrouver les informations des films d'identifiant movie:103
  - Retrouver tous les films dont le nom de famille du réalisateur est Scott
  - Afficher le nombre de ces films
  - Retrouver les titres des films dont l'un des acteurs a le nom de famille "Weaver"

Exercice (Solution)

- **Dans la collection films de la base de données TestDB :**
  - Retrouver les informations des films d'identifiant movie:103
  - Retrouver tous les films dont le nom de famille du réalisateur est Scott
  - Afficher le nombre de ces films
  - Retrouver les titres des films dont l'un des acteurs a le nom de famille "Weaver"

CRUD: Interrogation

- **Pipelines**
  - Modélisé selon le concept de data processing pipelines.
  - Offrent :
    - des filtres qui opèrent comme des requêtes.
    - des transformations de document qui modifient la forme du résultat.
    - des outils qui groupent et trient par champs
    - des outils pour agrègent le contenu des docs, incluant les tableaux.
  - Permet d'utiliser des **opérateurs** pour des tâches de calcul de moyenne ou de concaténation de mots.

CRUD: Interrogation

- **Agrégation**
  - Opérations qui traitent les enregistrements et retournent le résultat calculé.
  - Exécuter les agrégations dans mongod instance simplifie le code des applications et limite les besoins en ressources.
  - Le retour d'une séquence est l'entrée d'une autre séquence. Chaque séquence est possédée par un opérateur.

CRUD: Interrogation

- La fonction *"aggregate()"* concerne les opérations plus complexes
- La fonction *"aggregate()"* permet de spécifier des chaînes d'opérations, appelées pipeline d'agrégation :
  - db.users.aggregate([{$op1 : {}}, {$op2 : {}}, …]);
  - Quels sont ces opérations ?

- **{$match:{}}** : C'est le plus simple, il correspond au premier paramètre de la requête ***find*** que nous avons fait jusqu'ici. Il permet donc de filtrer le contenu d'une collection.
- **{$project:{}}** : C'est le second paramètre du ***find***. Il donne le format de sortie des documents (projection). Il peut par ailleurs être utilisé pour changer le format d'origine.
- **{$sort:{}}** : Trier le résultat final sur les valeurs d'une clé choisi.

- **{$group:{}}** : C'est l'opération d'agrégation. Il va permettre de grouper les documents par valeur, et appliquer des fonctions d'agrégat. La sortie est une nouvelle collection avec les résultats de l'agrégation.
- **{$unwind:{}}** : Cet opérateur prend une liste de valeur et produit pour chaque élément de la liste un nouveau document en sortie avec cet élément.
- **{$lookup: {}}:** jointure gauche (depuis 3.2)
- **{$out:{} }:** stockage du résultat (depuis 3.2)

50 CRUD: Interrogation

- **Requête : le titre et l'année des films réalisés après 2016.**

```
> db.films.aggregate([
    {$match:{"year":{$gt:2016}}},
    {$project:{"title":1,"year":1}}
])
```

```
> varMatch = {$match:{"year":{$gt:2016}}}
> varProject = {$project:{"title":1,"year":1}}
> db.films.aggregate([varMatch, varProject])
```

Si on souhaite ordonner la liste des films retournés selon l'année :

```
> db.films.aggregate([
    {$match:{"year":{$gt:2016}}},
    {$project:{"title":1,"year":1}},
    {$sort:{"year":1}}
])
```

```
> varSort = {$sort:{"year":1}}
> db.films.aggregate([varMatch, varProject, varSort])
```

52 CRUD: Interrogation

- **Exemple:**
  - **Requête** : créer un document pour chaque instance du tableau

```
> db.users.aggregate([ {$unwind : "$job"}} ]);
```

```
{"_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
"name": "James Bond",
 "login": "james",
"age": 50,
"address": {"street": "30 Wellington Square", "city": "London"},
"job" : ["agent", "MI6"] }
```

```
{"_id":
ObjectId("4efa8d2b7d284dad101e4bc7"),
"name": "James Bond", "login": "james", "age":
50,
"address": {"street": "30 Wellington Square",
"city": "London"},
"job" : "agent"}
```

```
{"_id":
ObjectId("4efa8d2b7d284dad101e4bc7"),
"name": "James Bond", "login": "james", "age":
50,
"address": {"street": "30 Wellington Square",
"city": "London"},
"job" : "MI6"}
```

On peut décomposer les listes des acteurs dans les documents des films, et affecter les titres des films pour chaque acteur.

```
> varUnwind = {$unwind:"$actors"}
> varGroup = {$group:{_id:"$actors",films:{$push:"$title"}}}
> db.films.aggregate([varUnwind, varGroup])
```

- L'opérateur *$group* groupe les documents passés en entré selon la valeur passé pour la clé de groupement "_id" et permet de réaliser une fonction d'agrégation par groupe.
- Le document de sortie contient la clé de groupement et le résultat de fonction d'agrégation.
- Les fonctions d'agrégation qu'on peut utiliser : *$sum, $max, $min, $avg, $push,* etc.

55 CRUD: Interrogation

- **$group :** clé (**_id**) + agrégat ($sum / $avg / …)
  - **Pas de groupement : *valeur unique***

```
> db.films.aggregate([ {$group : {"_id" : "year",
"res": {$sum : 1}}} ]);
```

  - **Groupement par valeur : *$clé***

```
> db.films.aggregate([ {$group:{"_id" : "$year",
"res": {$sum : 1}}} ]);
```

  - **Moyenne des valeurs : *$clé***

```
> db.films.aggregate([{$group:{"_id":"$year",
"moy": {$avg: "$rating"}}} ]);
```

CRUD: Interrogation

- **Exemple** de séquences

```
> db.users.aggregate([
{$match: {"address.city" : "London"} },
{$unwind : "$job" },
{$group : {"_id" : "$job", "moy": {$avg: "$age"}} },
{$match : {"moy" : {$gt : 30}} },
{$sort : { "moy" : -1} } ]);
```

57 CRUD: Interrogation

- $lookup
  - Effectue une jointure externe gauche vers une collection dans la même base de données pour filtrer les documents de la collection "jointe" pour traitement.
  - $lookup ajoute un nouveau champ de type tableau à chaque document d'entrée. Le nouveau champ contient les documents correspondants de la collection "jointe".
  - $lookup passe ces documents remodelés à l'étape suivante.
  - À partir de MongoDB 5.1,$lookup fonctionne sur des collections partitionnées.
  - Pour combiner des éléments de deux collections différentes, utilisez l' $unionWith étape pipeline.

58 CRUD: Interrogation

- **Correspondance d'égalité avec une condition de jointure unique**
  - Pour effectuer une equi jointure entre un champ des documents d'entrée et un champ des documents de la collection "jointe", le $lookup a cette syntaxe :

```
db.collection.aggregate([ {
  $lookup:
    {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
    }
} ])
```

CRUD: Interrogation

Exemples : Effectuer une jointure d'égalité simple avec `$lookup`

Créez une collection **`orders`** avec ces documents :

```
db.orders.insertMany( [
{ "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
{ "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
{ "_id" : 3 }
] )
```

Créez une autre collection **`inventory`** avec ces documents :

```
db.inventory.insertMany( [
{ "_id" : 1, "sku" : "almonds", "description": "product 1", "instock" : 120 },
{ "_id" : 2, "sku" : "bread", "description": "product 2", "instock" : 80 },
{ "_id" : 3, "sku" : "cashews", "description": "product 3", "instock" : 60 },
{ "_id" : 4, "sku" : "pecans", "description": "product 4", "instock" : 70 },
{ "_id" : 5, "sku":  null, "description": "Incomplete" },
{ "_id" : 6 }
] )
```

CRUD: Interrogation

- Exemple : Effectuer une jointure d'égalité simple avec $lookup
- L'opération d'agrégation suivante sur la collection **orders** joint les documents de **orders** avec les documents de la collection **inventory** en utilisant les champs **item** de la collection **orders** et le champ **sku** de la collection **inventory**.

```
db.orders.aggregate([
  {
    $lookup:
      {
        from: "inventory",
        localField: "item",
        foreignField: "sku",
        as: "inventory_docs"
      }
  }
])
```

CRUD: Interrogation

Effectuer une jointure d'égalité simple avec `$lookup`

```
SELECT *, inventory_docs
FROM orders
WHERE inventory_docs IN (
  SELECT *
  FROM inventory
  WHERE sku = orders.item
);
```



```
{
"_id" : 1,
"item" : "almonds",
"price" : 12,
"quantity" : 2,
"inventory_docs" : [
{ "_id" : 1, "sku" : "almonds", "description" : "product 1", "instock" : 120 }
]
}
{
"_id" : 2,
"item" : "pecans",
"price" : 20,
"quantity" : 1,
"inventory_docs" : [
{ "_id" : 4, "sku" : "pecans", "description" : "product 4", "instock" : 70 }
]
}
{
"_id" : 3,
"inventory_docs" : [
{ "_id" : 5, "sku" : null, "description" : "Incomplete" },
{ "_id" : 6 }
]
}
```

CRUD: Interrogation

Collection :Customers

```
{
  "_id": "1"
 "name": "John Doe",
 "email": "j@example.com",
 "address": "123 Main Street"
}
```

Collection :Orders

```
{
  "_id":" Num1"
  "order_number": "ORD-2023-001",
  "customer_id": "1 " ,
  "total_amount": 150.00
}
```

CRUD: Interrogation

```
db.orders.aggregate([
  {
    $lookup : {
       from:  "customers", //  Nom de la collection cible
       localField:  "customer_id", //  Champ dans la collection source (orders)
       foreignField:  "_id", //  Champ dans la collection cible (customers)
        as : "customer_info" //  Nom du champ où seront stockées les informations
    }
  }
])
```

```
{
  "_id": “Num1”
  "order_number": "ORD-2023-001",
  "customer_id": “1"
  "total_amount": 150.00,
  "customer_info": [ {
    "_id": “1”,
    "name": "John Doe",
    "email": "johndoe@example.com",
    "address": "123 Main Street"
  }]
}
```

Exercice

- Afficher un document qui contient les informations de l'acteur Jamie Foxx dans la clé "_id" et la liste de ses films associés à la clé "films"

Exercice (Solution)

- Afficher un document qui contient les informations de l'acteur Jamie Foxx dans la clé "_id" et la liste de ses films associés à la clé "films"

CRUD: Mise à jour

```
> db.collection.update(query, update, [{upsert:boolean, multi:boolean}])
```

- query (document) : la requête de recherche
- update (document) : les modifications à apporter ($set, $unset, $inc)
- upsert (boolean) :
  - false (default) : uniquement mise à jour
  - true : mise à jour ou ajout d'un nouveau document si le document n'existe pas
- multi (boolean) :
  - false (default) : maj uniquement de la 1ère occurrence trouvée
  - true : maj de toutes les occurrences.

CRUD: Mise à jour

- remplacer un document par le document en 2ème paramètre

```
> db.films.update({"_id": "movie:24"}, {"title": "Kill Bill"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

- **$set :** Pour mettre à jour la valeur d'une clé d'un film

```
> db.films.update({"_id": "movie:103"}, {$set:{"genre":"drame",
"director.first_name":"Test","rating":5}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

- **$unset :** Pour retirer une clé d'un document

Champ inexistant dans le document

```
> db.films.update({"_id": "movie:103"}, {$unset:{"genre": ""}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

CRUD: Mise à jour

- $inc

```
> db.films.update({"_id":"movie:103"},{$inc:{"rating":2}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

- Mise à jour de liste :

```
> db.films.update({"_id":"movie:28"},$set:{"actors.1": {"first_name":"xyz",
"last_name":"zyz", "birth_date":1955}, "actors.2.first_name":"zyx"}})
```

```
> db.films.update({"year":2010,"actors.first_name":"Winona"},
{$set:{"actors.$.birth_date":1961}})
```

Opérateur de positionnement

CRUD: Mise à jour

- **Mise à jour : Les paramètres des listes**
- **Mise à jour de liste : $push, $pull, $pop**
  - $push : Ajouter un objet dans une liste

```
db.films.update({"_id":"movie:103"},{$push:{actors:{"first_name":"Test",
"last_name":"Test", birth_date:"1961"}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

  - $pull : Retirer un objet d'une liste

```
db.films.update({"_id":"movie:103"},{$pull:{actors:{"first_name":"Test"}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

  - $pop : Retirer le premier ou le dernier objet d'une liste

```
db.films.update({"_id":"movie:103"},{$pop:{actors:-1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Retire le premier objet de la liste

```
db.films.update({"_id":"movie:103"},{$pop:{actors:1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Retire le dernier objet de la liste

CRUD: Mise à jour

- **upsert : Par défaut false (pas d'insertion de nouveau document)**

```
> db.films.update({"year":2023},{$set:{"genre":"drame"}});
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
```

- **Upsert : true (insertion d'un nouveau document)**

```
>
db.films.update({"year":2023},{$set:{"genre":"drame"}},{upsert:true});
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("652fb8ea54b869a960336ff6")
})
```

CRUD: Mise à jour

- **multi : Par défaut false (mise à jour de la première occurrence)**

```
> db.films.update({"year":1990},{$set:{"rating":5}})
WriteResult({ "nMatched" : 6, "nUpserted" : 0, "nModified" : 1 })
```

- **multi : true (mise à jour de tous les documents qui répondent à la condition)**

```
> db.films.update({"year":1990},{$set:{"rating":5}},
{multi:true})
WriteResult({ "nMatched" : 6, "nUpserted" : 0, "nModified" : 6 })
```

CRUD:Suppression

```
> db.collection.remove(requête, [justOne])
```

- Requête (document) : requête de recherche du/des documents à supprimer. Un document vide permet de supprimer tous les documents de la collection.
- justOne (boolean) :
  - false (default) : suppression de tous les documents retournés
  - true : suppression d'un seul document de la liste de documents retournés

CRUD: Suppression

- **Suppression :**

Suppression d'un seul document de film sortie en 2020:

```
> db.films.remove({"year":2000}, true)
```

Suppression de tous les documents des films sorties en 2020:

```
> db.films.remove({"year":2000})
```

Suppression de tous les documents de la collection films:

```
> db.films.remove({})
```

CRUD: Isolation

- **Cohérence: Isolation**
  - Par défaut, toutes les insertions sont atomiques au niveau document.
  - Ceci implique que par défaut toutes les écritures peuvent interférer avec d'autre opérations.
  - On peut isoler les écritures dans les collections unsharded en ajoutant **$isolated:1** dans la requête :

db.<collection>.remove({<field>:<value>, *$isolated: 1*})

Exercice

- **Dans la collection films de la base de données TestDB :**
  - Ajouter dans le film 103 un couple clé-valeur {"sous_titre": "francais"}
  - Enlever du film 103 le couple {"sous_titre": "francais"}
  - Vider la collection films

Exercice (Solution)

- **Dans la collection films de la base de données TestDB :**
  - Ajouter dans le film 103 un couple clé-valeur {"sous_titre": "francais"}
  - Enlever du film 103 le couple {"genre":"crime"}
  - Vider la collection films