Gestion de Données à Grande Échelle

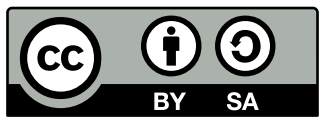
Partie II.1 – Introduction au SGBD MongoDB

Équipe pédagogique BD

Ugo Comignani

ugo.comignani@grenoble-inp.fr

Ensimag ISI 3ème année – 2021-2022



Grenoble INP
ensimag

Organisation du cours

1. Fondements de la gestion répartie des données
   - Distribution des données et évaluation de requêtes
   - Coherence, transactions et duplication
2. Stockage et traitement des masses de données
   - Décisionnel et entrepôts de données
   - Approches NoSQL
   - SGFD, MapReduce
3. Données géographiques et temporelles
   - Systèmes d'Information Géographiques
   - Séries temporelles

Organisation du cours

1. Fondements de la gestion répartie des données
   - Distribution des données et évaluation de requêtes
   - Coherence, transactions et duplication
2. Stockage et traitement des masses de données
   - Décisionnel et entrepôts de données
   - Approches NoSQL
   - SGFD, MapReduce
3. Données géographiques et temporelles
   - Systèmes d'Information Géographiques
   - Séries temporelles

Organisation du cours (II)

Pour la partie *Introduction à MongoDB* :

- **Prérequis :**
  - Bases de données relationnelles
  - Clefs étrangères, normalisation
  - SQL
- **Organisation :**
  - 1h30 de CM introduction générale à MongoDB
  - 1h30 de TP d'introduction à MongoDB

1 Première partie

MongoDB : un SGBD Orienté Documents

1. Caractéristiques techniques de MongoDB
2. Le modèle de données
3. Références entre documents
4. Survol des caractéristiques techniques

Caractéristiques techniques de MongoDB

- Système de Gestion de Bases de Données orienté documents (NoSQL)
- Développé depuis 2007
- Distribué sous licence SSPL (*Server Side Public License*), dérivée de la GNU GPL
- Au 5ème rang des SGBD les plus utilisés au monde (selon https://db-engines.com/en/ranking, février 2021)
- Utilisé par de grandes entreprises (voir sur le site de MongoDB)

Principales caractéristiques techniques

- Architecture client-serveur
- Modèle de données s'appuyant sur JSON
- Modèle semi-structuré : pas de schéma (schéma intégré dans les données)
- Interrogation complexe : projections, sélections (filtres), simili-jointures (*lookup*), agrégations, map-reduce...
- Sérialisation BSON (*binary JSON*)
- Distribution des données (*sharding*)
- Gestion d'index : *B-tree*
- Réplication distribuée (*replicas*)
- Support multi-langages : C, C++, Dart, Erlang, Go, Haskell, Java, JavaScript, .NET (C# F#, PowerShell, etc.), Perl, PHP, Python, Ruby, Scala
- ...

Le modèle de données

Le modèle de données de MongoDB

Dans le monde relationnel
Relation (table)
Tuples
Attributs atomiques
Schéma de relation

Le modèle de données de MongoDB

Dans le monde relationnel	Dans le monde MongoDB
Relation (table)	Collection
Tuples	Document
Attributs atomiques	Attributs atomiques + attributs de type tableau + attributs de type Objet JSON
Schéma de relation	Pas de schéma fixe. Schéma intégré dans les documents (**chaque document d'une collection peut avoir un schéma différent**)

Rappel : les BD relationnelles

Élèves	prénom	nom	e-mail	filière
	Luke	Skywalker	skywalker@imag.fr	MMIS
	Dark	Vador	vador@imag.fr	IF
	Han	Solo	solo@falcon.com	IF
	Leia	Solo	princess@falcon.com	MMIS
	Jabba	The Hut	jabba@imag.fr	SEOC

Rappel : les BD relationnelles

Élèves	prénom	nom	e-mail	filière
	Luke	Skywalker	skywalker@imag.fr	MMIS
	Dark	Vador	vador@imag.fr	IF
	Han	Solo	solo@falcon.com	IF
	Leia	Solo	princess@falcon.com	MMIS
	Jabba	The Hut	jabba@imag.fr	SEOC

dont le schéma est...

Élèves : {prénom : String, nom : String, e-mail : String,
filière :{IF, ISI, MMIS, SEOC}}.

Dans le monde Mongo...

La collection **Élèves** :

```
{
  "_id" : ObjectId("5d5a63890cbf550e058ccca3"),
  "prénom" : "Dark",
  "nom" : "Vador",
  "e-mail" : "vador@imag.fr",
  "filière" : "IF"
}
{
  "_id" : ObjectId("5d5a63890cbf550e058ccca6"),
  "prénom" : "Han",
  "nom" : "Solo",
  "e-mail" : "solo@falcon.com",
  "filière" : "IF"
}
...
```

Dans le monde Mongo...

- **Élèves** est une collection
- Chaque élève est un document
- Chaque champ correspond à un pseudo-attribut (nom-valeur). Exemple : `"prénom": "Dark"`
- Les élèves peuvent avoir des champs complètement différents (mais il ne vaut mieux pas pour des raisons de cohérence et de performance...)
- Chaque document a un identifiant unique `"_id"`, généré par Mongo s'il n'est pas précisé
- Quelques types possibles pour les champs : `String`, nombres (`byte`, `int32`, `int64` et `double`), `Boolean`, `array`, `ISODate`, document (JSON embarqué) ou `ObjectId` (référence vers un autre document)

Références entre documents

Contraintes de référence en relationnel

Élèves	prénom	nom	e-mail	filière
	Dark	Vador	vador@imag.fr	IF
	Obi-Wan	Kenobi	kenobio@imag.fr	MMIS
	Han	Solo	solo@falcon.com	IF

Notes	cours	prénom	nom	note
	sport	Dark	Vador	20
	sport	Jabba	The Hut	3
	pilotage	Han	Solo	15

Contraintes de référence en relationnel

Élèves	prénom	nom	e-mail	filière
	Dark	Vador	vador@imag.fr	IF
	Obi-Wan	Kenobi	kenobio@imag.fr	MMIS
	Han	Solo	solo@falcon.com	IF

Notes	cours	prénom	nom	note
	sport	Dark	Vador	20
→	sport	Jabba	The Hut	3
	pilotage	Han	Solo	15

Dans la plupart des bases de données, les documents sont liés les uns aux autres. Exemple : les élèves de **Notes** sont les mêmes que ceux de **Élèves**. En relationnel : Contrainte de référence

Intégration vs. Référence

En MongoDB, deux solutions possibles :

- intégration (*embedding*)
- référence

Intégration (embedding)

```
{
  "_id" : ObjectId("5d5a63890cbf550e058ccca3"),
  "prénom" : "Dark",
  "nom" : "Vador",
  "e-mail" : "vador@imag.fr",
  "filière" : "IF",
  "notes" : [
    { "cours" : "Sport", "note" : 20 },
    { "cours" : "Méditation transcendantale", "note" : 3 },
    { "cours" : "Bases de Données", "note" : 13 }
  ]
}
...
```

Intégration (embedding)

```
{
  "_id" : ObjectId("5d5a63890cbf550e058ccca3"),
  "prénom" : "Dark",
  "nom" : "Vador",
  "e-mail" : "vador@imag.fr",
  "filière" : "IF",
  "notes" : [
    { "cours" : "Sport", "note" : 20 },
    { "cours" : "Méditation transcendantale", "note" : 3 },
    { "cours" : "Bases de Données", "note" : 13 }
  ]
}
...
```

Ici, les notes d'un étudiant sont directement intégrées comme « sous-collection » de l'élève. $\rightarrow$ on *dénormalise* le schéma (non FN1).

Adapté essentiellement lorsque l'association entre entités est de type *oneToOne* ou *oneToMany*.

Référence

Autre solution, la référence...

Autre solution, la référence...

Collection `eleves` :

```
{"_id" : ObjectId("5d5a63890cbf550e058ccca3"),
  "prénom" : "Dark",
  "nom" : "Vador",
  "e-mail" : "vador@imag.fr",
  "filière" : "IF"}
...
```

Collection `notes` :

```
{"_id" : ObjectId("5d5a67360cbf550e058ccccd"),
  "cours" : "Sport",
  "numEleve" : ObjectId("5d5a63890cbf550e058ccca3"),
  "note" : 20}
{"_id" : ObjectId("5d5a67360cbf550e058cccd1"),
  "cours" : "Méditation transcendantale",
  "numEleve" : ObjectId("5d5a63890cbf550e058ccca3"),
  "note" : 3}
...
```

Ici, `numEleve` « référence » l'attribut `_id` de la collection `eleves`.

$\Leftrightarrow$ clef étrangère dans le monde relationnel.

Intégration vs. Référence

Intégration :

- Modèle non normalisé ($\rightarrow$ redondances possibles)
- À privilégier pour des relations de type *oneToOne* ou *oneToMany*
- Requêtes d'interrogation plus simples (pas de jointure)
- Meilleures performances pour les opérations de lecture

Intégration vs. Référence

Intégration :

- Modèle non normalisé ($\rightarrow$ redondances possibles)
- À privilégier pour des relations de type *oneToOne* ou *oneToMany*
- Requêtes d'interrogation plus simples (pas de jointure)
- Meilleures performances pour les opérations de lecture

Référence :

- Modèle normalisé (similaire aux SGBD relationnelles)
- À privilégier pour des relations de type *manyToMany*
- Requêtes d'interrogation plus complexes (il faut résoudre les références et effectuer les jointures)
- Plus flexible que le modèle par intégration

Intégration vs. Référence

Intégration :

- Modèle non normalisé ($\rightarrow$ redondances possibles)
- À privilégier pour des relations de type *oneToOne* ou *oneToMany*
- Requêtes d'interrogation plus simples (pas de jointure)
- Meilleures performances pour les opérations de lecture

Référence :

- Modèle normalisé (similaire aux SGBD relationnelles)
- À privilégier pour des relations de type *manyToMany*
- Requêtes d'interrogation plus complexes (il faut résoudre les références et effectuer les jointures)
- Plus flexible que le modèle par intégration

Le modèle à privilégier dépend donc de l'application...

Survol des caractéristiques techniques

Opérations basiques

- Insertion
- Suppression
- Mise-à-jour
- Recherche
- Agrégation
- Gestion d'index

Indexation

L'indexation des documents, qui permet une recherche performante selon des critères concernant certains champs, s'appuie sur une structure de type arbre B (*B-tree*).

Indexation

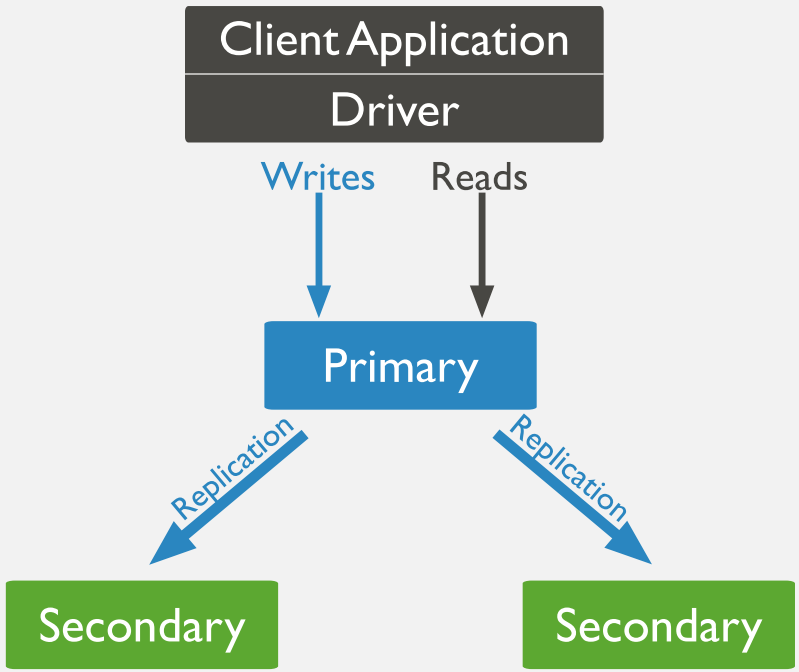
L'indexation des documents, qui permet une recherche performante selon des critères concernant certains champs, s'appuie sur une structure de type arbre B (*B-tree*).

MongoDB supporte plusieurs types d'index :

- Champ simple
- Champ composé
- Tableau (*array*)
- Index géospatial (2d geohash)
- Index textuel
- Index fondé sur une fonction de hachage (permet la recherche par égalité mais pas la recherche sur intervalle de valeurs. Permet également le *sharding* efficace)

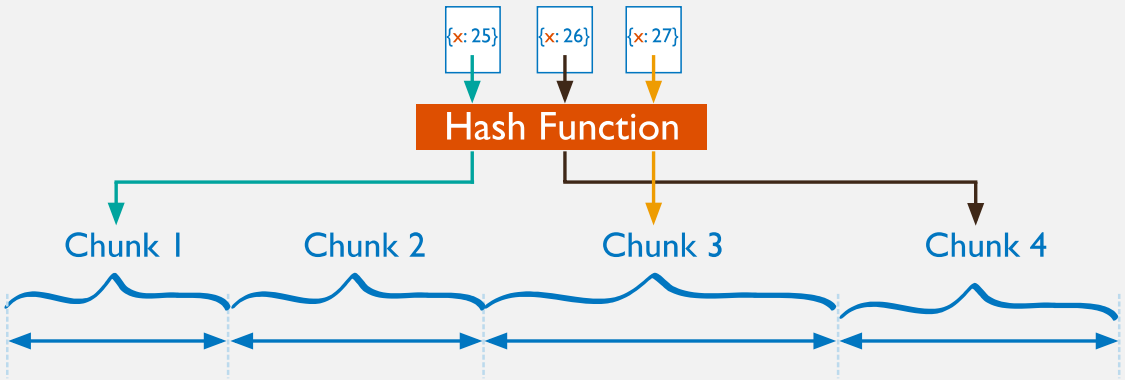
Réplication, principes généraux

- Replicas : Groupe d'instances Mongo maintenant le même jeu de données
- 1 seul replica primaire, autant de secondaires que nécessaires
- Seul le primaire peut écrire. Opérations logguées dans le `oplog`
- Réplication asynchrone entre les replicas secondaires (du `oplog` primaire)
- Tolérant à la perte d'un replica (y compris primaire)



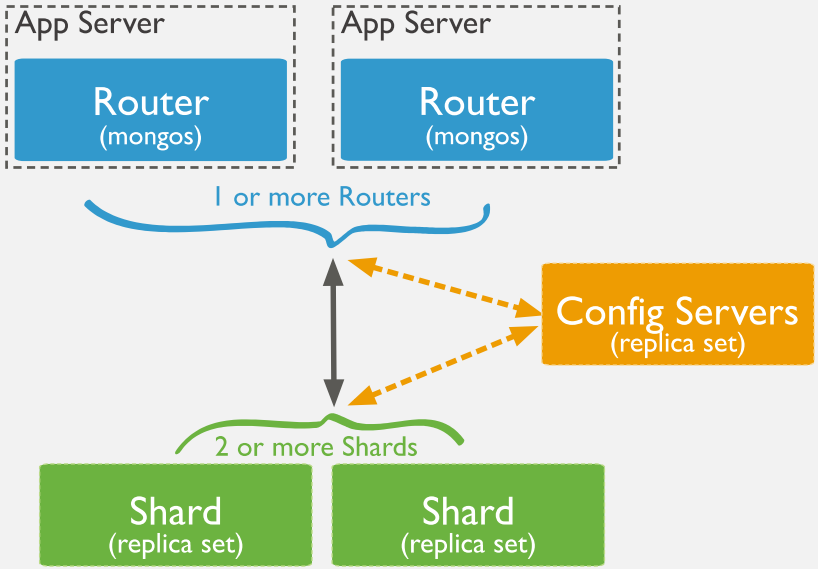
Source https://docs.mongodb.com/manual/replication/ CC-by-SA

- Permet de mettre en œuvre le passage à l'échelle horizontal : en ajoutant des machines.
- **Principe** : stockage distribué d'une collection :
  - les documents d'une collections sont divisés en fragments (*chunks*), à partir d'une clef de *shard*
  - Cette clef peut être n'importe quel champ indexé dans la collection
  - Chaque *chunk* est distribué sur une machine différente dans le *cluster*



Source https://docs.mongodb.com/manual/sharding/ CC-by-SA

Architecture générale de sharding



Source https://docs.mongodb.com/manual/sharding/ CC-by-SA

Chaque Shard peut être constitué d'un groupe de replicas (donc plusieurs machines)

Deux mécanismes de sharding :

- Partitionnement par intervalle
- Partitionnement par fonction de hachage

2 Deuxième partie

MongoDB en Pratique

5. Insertion / Suppression / Mise-à-jour

6. Interrogation

Se connecter à une base de données

- Fonctionnement de Mongo : client/serveur
- Client : langage de programmation avec *bindings* Mongo ou Mongo shell

Se connecter à une base de données

- Fonctionnement de Mongo : client/serveur
- Client : langage de programmation avec *bindings* Mongo ou Mongo shell

Connexion à une base de données en utilisant Python :

```
>>> from pymongo import MongoClient
>>> client = MongoClient('mongodb://login:password@mongodb.ensimag.
    fr/bddName')
>>> db = client.ensimag  # Connection to "ensimag" database
```

Se connecter à une base de données

- Fonctionnement de Mongo : client/serveur
- Client : langage de programmation avec *bindings* Mongo ou Mongo shell

Connexion à une base de données en utilisant Python :

```
>>> from pymongo import MongoClient
>>> client = MongoClient('mongodb://login:password@mongodb.ensimag.
    fr/bddName')
>>> db = client.ensimag  # Connection to "ensimag" database
```

Connexion à une base de données en utilisant le shell Mongo :

```
$ mongo "mongodb://login:password@mongodb.ensimag.fr/bddName"
MongoDB shell version v3.6.8
connecting to: mongodb://mongodb.ensimag.fr:27017/login[...]
[...]
>>> use ensimag
switched to db ensimag
```

Se connecter à une base de données

- Fonctionnement de Mongo : client/serveur
- Client : langage de programmation avec *bindings* Mongo ou Mongo shell

Connexion à une base de données en utilisant Python :

```
>>> from pymongo import MongoClient
>>> client = MongoClient('mongodb://login:password@mongodb.ensimag.
    fr/bddName')
>>> db = client.ensimag  # Connection to "ensimag" database
```

Connexion à une base de données en utilisant le shell Mongo :

```
$ mongo "mongodb://login:password@mongodb.ensimag.fr/bddName"
MongoDB shell version v3.6.8
connecting to: mongodb://mongodb.ensimag.fr:27017/login[...]
[...]
>>> use ensimag
switched to db ensimag
```

Remarque : si la base de données n'existe pas, on peut s'y connecter quand même. Elle ne sera effectivement créée que lorsque l'on y insérera des données.

Récupérer une collection

Récupération d'une référence vers une collection en Python :

```
>>> db = client.ensimag  # Connection to "ensimag" database
>>> eleves = db.eleves  # Reference to the collection "eleves"
```

Récupérer une collection

Récupération d'une référence vers une collection en Python :

```
>>> db = client.ensimag  # Connection to "ensimag" database
>>> eleves = db.eleves  # Reference to the collection "eleves"
```

Récupération d'une référence vers une collection en shell Mongo :

```
>>> db.eleves
```

Récupérer une collection

Récupération d'une référence vers une collection en Python :

```
>>> db = client.ensimag  # Connection to "ensimag" database
>>> eleves = db.eleves  # Reference to the collection "eleves"
```

Récupération d'une référence vers une collection en shell Mongo :

```
>>> db.eleves
```

Remarque : si la collection n'existe pas, elle sera créée lors de la première insertion de données.

Par la suite, nous donnerons tous les exemples en shell Mongo (le fonctionnement avec des langages comme Python est similaire).

Insertion / Suppression / Mise-à-jour

Insertion d'un document dans la collection "eleves" en shell Mongo :

```
>>> db.eleves.insert({
... "_id" : ObjectId("5d5bc26f0f38bea8573f29b4"),
... "prénom": "Maître",
... "nom": "Yoda",
... "e-mail": "master.yoda@ensimag.fr",
... "filière": "ISI"
... })
WriteResult({ "nInserted" : 1 })
```

Insertion d'un document dans la collection "eleves" en shell Mongo :

```
>>> db.eleves.insert({
... "_id" : ObjectId("5d5bc26f0f38bea8573f29b4"),
... "prénom": "Maître",
... "nom": "Yoda",
... "e-mail": "master.yoda@ensimag.fr",
... "filière": "ISI"
... })
WriteResult({ "nInserted" : 1 })
```

- Le champ `"_id"` qui sert de clef est facultatif (et est inséré automatiquement s'il n'est pas précisé)
- Si un document de même id existe déjà, la commande échoue
- On peut utilise `save` à la place de `insert`, qui met à jour si un document de même id existe déjà, et insère sinon
- Il existe également `insertOne` et `insertMany`
- Remarque : on peut également importer des données depuis un fichier avec le script `mongoimport`

Suppression

Suppression de documents dans la collection ”eleves” en shell Mongo :

```
>>> db.eleves.remove({"nom": "Yoda"})
WriteResult({ "nRemoved" : 1 })
```

Suppression

Suppression de documents dans la collection "eleves" en shell Mongo :

```
>>> db.eleves.remove({"nom": "Yoda"})
WriteResult({ "nRemoved" : 1 })
```

- On passe une requête de filtrage en argument de `remove` (voir plus loin pour la syntaxe)
- **Attention :** Si aucun argument n'est passé, alors `remove` vide la table (même chose que `DELETE FROM` en SQL)

Mise à jour

Remplacement de l'information concernant un élève, en shell Mongo :

```
>>> db.eleves.update(
... {"nom": "Yoda"},
... {"e-mail": "maitre@yoda.fr"}
... )
```

- Le premier argument précise quel document mettre à jour
- Seul le premier document vérifiant le critère sera modifié (pour les modifier tous, utiliser `updateMany`
- Le second argument spécifie par quoi remplacer le document
- **Attention :** tout le contenu du document sera remplacé ! (dans l'exemple, Yoda perdra donc son nom, son prénom et sa filière)

Mise à jour

Remplacement de l'information concernant un élève, en shell Mongo :

```
>>> db.eleves.update(
... {"nom": "Yoda"},
... {"e-mail": "maitre@yoda.fr"}
... )
```

- Le premier argument précise quel document mettre à jour
- Seul le premier document vérifiant le critère sera modifié (pour les modifier tous, utiliser `updateMany`
- Le second argument spécifie par quoi remplacer le document
- **Attention :** tout le contenu du document sera remplacé ! (dans l'exemple, Yoda perdra donc son nom, son prénom et sa filière)

Pour ne toucher qu'une partie des champs sans tout effacer, utiliser `$set`

```
>>> db.eleves.update(
... {"nom": "Yoda"},
... {$set: {"e-mail": "maitre@yoda.fr"}}
... )
```

Interrogation

Interroger une collection

Récupérer tous les documents d'une collection :

```
>>> db.eleves.find()
{ "_id" : ObjectId("5d5a63890cbf550e058ccca6"), "prénom" : "Han", "
    nom" : "Solo", "e-mail" : "solo@falcon.com", "filière" : "IF" }
{ "_id" : ObjectId("5d5a63890cbf550e058ccca8"), "prénom" : "Luke", "
    nom" : "Skywalker", "e-mail" : "skywalker@imag.fr", "filière" :
    "MMIS" }
[...]
```

Interroger une collection

Récupérer tous les documents d'une collection :

```
>>> db.eleves.find()
{ "_id" : ObjectId("5d5a63890cbf550e058ccca6"), "prénom" : "Han", "
    nom" : "Solo", "e-mail" : "solo@falcon.com", "filière" : "IF" }
{ "_id" : ObjectId("5d5a63890cbf550e058ccca8"), "prénom" : "Luke", "
    nom" : "Skywalker", "e-mail" : "skywalker@imag.fr", "filière" :
    "MMIS" }
[...]
```

Pour filtrer la liste des champs affichés (réaliser une projection) :

```
>>> db.eleves.find({}, {"prénom": 1, "nom": 1})
{ "_id" : ObjectId("5d5a63890cbf550e058ccca6"), "prénom" : "Han", "
    nom" : "Solo" }
{ "_id" : ObjectId("5d5a63890cbf550e058ccca8"), "prénom" : "Luke", "
    nom" : "Skywalker" }
[...]
```

- Attention, les paramètres de projection correspondent au **deuxième** argument
- L'identifiant est ajouté par défaut (ajouter `"_id": 0` pour l'enlever)

Le premier paramètre de `find` permet de filtrer la liste des documents (réaliser une sélection) :

```
>>> db.eleves.find({"nom": "Solo"})
{ "_id" : ObjectId("5d5a63890cbf550e058ccca6"), "prénom" : "Han", "
    nom" : "Solo", "e-mail" : "solo@falcon.com", "filière" : "IF" }
{ "_id" : ObjectId("5d5a63890cbf550e058ccca7"), "prénom" : "Leia", "
    nom" : "Solo", "e-mail" : "princess@falcon.com", "filière" : "
    MMIS" }
```

Opérateurs de sélection

MongoDB fournit de nombreux opérateurs de sélection :

- Opérateurs de comparaison – `$gt`, `$gte`, `$lt`, `$lte`, `$ne`

```
>>> db.notes.find({"note": {$lt: 10}})
{ "_id" : ObjectId("5d5a67360cbf550e058cccce"), "cours" : "Sport", "
    numEleve" : ObjectId("5d5a63890cbf550e058ccca6"), "note" : 7 }
{ "_id" : ObjectId("5d5a67360cbf550e058ccccf"), "cours" : "Sport", "
    numEleve" : ObjectId("5d5a63890cbf550e058ccca7"), "note" : 9 }
```

Opérateurs de sélection

MongoDB fournit de nombreux opérateurs de sélection :

- Opérateurs de comparaison – `$gt`, `$gte`, `$lt`, `$lte`, `$ne`

```
>>> db.notes.find({"note": {$lt: 10}})
{ "_id" : ObjectId("5d5a67360cbf550e058cccce"), "cours" : "Sport", "
    numEleve" : ObjectId("5d5a63890cbf550e058ccca6"), "note" : 7 }
{ "_id" : ObjectId("5d5a67360cbf550e058ccccf"), "cours" : "Sport", "
    numEleve" : ObjectId("5d5a63890cbf550e058ccca7"), "note" : 9 }
```

- Test d'existence d'un champ –
  - `$exists: true`(pour tester l'existence d'un champ particulier)
  - `$exists: false`(pour tester l'inexistence d'un champ particulier)
  - `$type: <BSON type>|<alias>`(pour tester le type d'un champ particulier)

```
>>> db.eleves.find({"nom": {$exists: true}})
```

```
>>> db.eleves.find({"nom": {$type: "string"}})
```

Opérateurs de sélection (suite)

- Opérateurs booléens – `$or`, `$and`, `$not`, `$nor`

```
>>> db.notes.find({$and: [{"note": {$gt: 7}}, {"note": {$lt: 10}}]})
```

- Recherche dans un tableau – `$all`, `$in`, `$nin`, `$size`, `$slice`, `"array.X"`

Opérateurs de sélection (suite)

- Opérateurs booléens – `$or`, `$and`, `$not`, `$nor`

```
>>> db.notes.find({$and: [{"note": {$gt: 7}}, {"note": {$lt: 10}}]})
```

- Recherche dans un tableau – `$all`, `$in`, `$nin`, `$size`, `$slice`, `"array.X"`

Eleves dont la première note est inférieure à 10 :

```
>>> db.eleves_embedded.find({"notes.0.note": {$lt: 10}})
```

Opérateurs de sélection (suite)

- Opérateurs booléens – `$or`, `$and`, `$not`, `$nor`

```
>>> db.notes.find({$and: [{"note": {$gt: 7}}, {"note": {$lt: 10}}]})
```

- Recherche dans un tableau – `$all`, `$in`, `$nin`, `$size`, `$slice`, `"array.X"`

Eleves dont la première note est inférieure à 10 :

```
>>> db.eleves_embedded.find({"notes.0.note": {$lt: 10}})
```

Eleves dont au moins une note est inférieure à 10 :

```
>>> db.eleves_embedded.find({"notes.note": {$lt: 10}})
```

Opérateurs de sélection (suite)

- Opérateurs booléens – `$or`, `$and`, `$not`, `$nor`

```
>>> db.notes.find({$and: [{"note": {$gt: 7}}, {"note": {$lt: 10}}]})
```

- Recherche dans un tableau – `$all`, `$in`, `$nin`, `$size`, `$slice`, `"array.X"`

Eleves dont la première note est inférieure à 10 :

```
>>> db.eleves_embedded.find({"notes.0.note": {$lt: 10}})
```

Eleves dont au moins une note est inférieure à 10 :

```
>>> db.eleves_embedded.find({"notes.note": {$lt: 10}})
```

Eleves dont toutes les notes sont inférieures à 10 :

```
>>> db.eleves_embedded.find({$nor: [{"notes.note": {$gte: 10}}]})
```

Agrégation

La méthode `find` est une méthode d'interrogation simple sur une collection. Pour une interrogation plus complexe (regroupement, agrégation, pseudo-jointure), il faut utiliser `aggregate`.

Agrégation

La méthode `find` est une méthode d'interrogation simple sur une collection. Pour une interrogation plus complexe (regroupement, agrégation, pseudo-jointure), il faut utiliser `aggregate`.

`aggregate` fonctionne comme un *pipeline* de traitement de données, enchaînant les étapes (filtrage, regroupement...).

Agrégation

La méthode `find` est une méthode d'interrogation simple sur une collection. Pour une interrogation plus complexe (regroupement, agrégation, pseudo-jointure), il faut utiliser `aggregate`.

`aggregate` fonctionne comme un *pipeline* de traitement de données, enchaînant les étapes (filtrage, regroupement...).

La moyenne des notes par matière, restreintes aux seules notes ≥ 10 :

```
>>> db.notes.aggregate([{$match: { note: {$gte: 10}}}, {$group: {_id
    : "$cours", moyenne: {$avg: "$note"}}}])
{ "_id" : "Bases de Données", "moyenne" : 14.5 }
{ "_id" : "Sport", "moyenne" : 17.5 }
```

La méthode `find` est une méthode d'interrogation simple sur une collection. Pour une interrogation plus complexe (regroupement, agrégation, pseudo-jointure), il faut utiliser `aggregate`.

`aggregate` fonctionne comme un *pipeline* de traitement de données, enchaînant les étapes (filtrage, regroupement...).

La moyenne des notes par matière, restreintes aux seules notes ≥ 10 :

```
>>> db.notes.aggregate([{$match: { note: {$gte: 10}}}, {$group: {_id
    : "$cours", moyenne: {$avg: "$note"}}}])
{ "_id" : "Bases de Données", "moyenne" : 14.5 }
{ "_id" : "Sport", "moyenne" : 17.5 }
```

La moyenne des notes dans chaque matière par ordre décroissant :

```
>>> db.notes.aggregate([{$group: {_id: "$cours", moyenne: {$avg: "
    $note"}}}, {$sort: {"moyenne": -1}}])
{ "_id" : "Bases de Données", "moyenne" : 13.4 }
{ "_id" : "Sport", "moyenne" : 10.8 }
{ "_id" : "Méditation transcendantale", "moyenne" :
    5.333333333333333 }
```

Agrégation (suite)

Combien y a-t-il de notes dans chaque matière ?

```
>>> db.notes.aggregate([{$group: {_id: "$cours", nombre: {$sum:
    1}}}])
{ "_id" : "Bases de Données", "nombre" : 5 }
{ "_id" : "Sport", "nombre" : 5 }
{ "_id" : "Méditation transcendantale", "nombre" : 3 }
```

Combien y a-t-il de notes dans chaque matière ?

```
>>> db.notes.aggregate([{$group: {_id: "$cours", nombre: {$sum:
    1}}}])
{ "_id" : "Bases de Données", "nombre" : 5 }
{ "_id" : "Sport", "nombre" : 5 }
{ "_id" : "Méditation transcendantale", "nombre" : 3 }
```

Quelle est la moyenne la plus basse de toutes les matières ?

```
>>> db.notes.aggregate([{$group: {_id: "$cours", moyenne: {$avg: "
    $note"}}}, {$group: {"_id": null, moyenne: {$min: "$moyenne"
    }}}])
{ "_id" : null, "moyenne" : 5.333333333333333 }
```

(cette requête requiert une double agrégation)

Simili-jointure

Il n'y a pas dans MongoDB d'opérateur permettant de faire directement une jointure, mais on peut faire des choses similaires en suivant des références grâce à l'opérateur `$lookup`.

Il n'y a pas dans MongoDB d'opérateur permettant de faire directement une jointure, mais on peut faire des choses similaires en suivant des références grâce à l'opérateur `$lookup`.

Donner la liste des étudiants, avec leurs notes :

```
 >>> db.eleves.aggregate([{$lookup: {
        from: "notes",
        localField: "_id",
        foreignField: "numEleve",
        as: "notes"}},
    {$project: {"nom": 1, "prénom": 1, "notes.note": 1}}])

{ "_id" : ObjectId("5d5a63890cbf550e058ccca6"), "prénom" : "Han", "
    nom" : "Solo", "notes" : [ { "note" : 7 }, { "note" : 9 } ] }
{ "_id" : ObjectId("5d5a63890cbf550e058ccca8"), "prénom" : "Luke", "
    nom" : "Skywalker", "notes" : [ { "note" : 8 }, { "note" : 15 },
     { "note" : 12 } ] }
[...]
```

Conclusion

Conclusion

- Nous n'avons couvert qu'une petite partie de MongoDB
- Pour le reste, voir la documentation officielle (https://docs.mongodb.com/manual/)
- Le plus important à retenir : modèle de données + comment l'interroger
- À vous maintenant...