# TP1 NoSql

Othman

27 November 2025


FIGURE 1 – Logo de MongoDb

## 1 Introduction

MongoDB est un SGBD orienté document, contrairement au SGBD « classique » il utilise des collections au lieu des tables et les lignes correspondent maintenant à des documents manipulable en tant qu'object.

## 2 Installation

Pour installer MongoDB, il suffit de se rendre sur le site officiel de MongoDB et de télécharger la version *Community Server* adaptée à votre système d'exploitation. Une fois l'installateur lancé, choisissez l'installation complète, qui inclut MongoDB et les outils associés. Après l'installation, assurez-vous que le service MongoDB est bien démarré, puis vérifiez son fonctionnement en exécutant la commande `mongosh` dans un terminal. Vous pouvez ensuite commencer à créer vos bases de données et collections.

## 2.1 MongoDB Compass

Vous pouvez également installer une interface graphique pour MongoDB. *MongoDB Compass* vous permet de vous connecter à un serveur MongoDB local ou distant et de gérer vos bases de données de manière visuelle. L'installateur vous propose normalement de l'installer automatiquement. Si ce n'est pas le cas, vous pouvez le télécharger ici.

## 2.2 Configuration du PATH (facultatif)

Pour pouvoir exécuter `mongosh` ou `mongo` depuis n'importe quel terminal, il peut être nécessaire d'ajouter le chemin d'installation de MongoDB à votre variable d'environnement `PATH`. Sur Windows, vous pouvez le faire via les paramètres système avancés, et sur Linux ou macOS, en modifiant votre fichier `.bashrc` ou `.zshrc`.

## 2.3 Importation des données de travail

Premièrement il faut récupérer le fichier via la commande suivante :

```
curl https://atlas-education.s3.amazonaws.com/sampledata.archive -o sampledata.archive
```

Ensuite il faut importer les données dans mongoDb avec la commande :

```
mongorestore --archive=sampledata.archive
```

Enfin pour vérifier si les données ont bien été importé, il faut :

— Se connecter sur le serveur mongoDb avec :
```
mongosh
```
— Utiliser la bonne base avec :
```
use sample_mflix
```
— Vérifier le nombre de film avec :
```
db.movies.find().count()
```

Une fois ces étapes complétés vous pouvez réaliser les différentes requêtes sur la base de données.

# 3 Quelques commandes à connaître

Pour illustrer les différentes méthodes disponibles avec MongoDB, nous utiliserons la base de données des films que nous venons d'insérer. Nous allons explorer les méthodes fondamentales : `find()`, `update()`, `aggregate()`, ainsi que la gestion et l'utilisation des index.

# 4 Sélections simples avec `find()`

**Méthode** `find(filter, projection)`

- Le premier paramètre est **le filtre** (critères de recherche). C'est l'équivalent de la clause **WHERE** en SQL.
- Le second paramètre est la **projection** : elle spécifie quels champs afficher. On peut le comparer à un **SELECT** en SQL.

## 4.1 Afficher les 5 films sortis depuis 2015

```
db.movies.find({ year: { $gte: 2015 } }).limit(5)
```

**Explication :**

- `$gte` signifie *greater than or equal to*.
- `limit(5)` permet de limiter le nombre de documents retournés.

## 4.2 Trouver tous les films dont le genre est "Comedy"

```
db.movies.find({ genres: "Comedy" })
```

**Explication :** MongoDB vérifie si la valeur `"Comedy"` est présente dans le tableau `genres`. Aucune opération supplémentaire n'est nécessaire : l'opérateur d'égalité fonctionne directement sur les tableaux.

## 4.3 Afficher les films sortis entre 2000 et 2005

```
db.movies.find({ year: { $gte: 2000, $lte: 2005 } },
               { title: 1, year: 1 }).pretty()
```

**Explication :**

- `$lte` signifie *less than or equal to*.
- La projection `{ title: 1, year: 1 }` affiche uniquement le titre et l'année.
- `pretty()` formate le résultat pour une meilleure lisibilité.

### 4.4 Films de genres “Drama” ET “Romance”

```
db.movies.find({ genres: { $all: ["Drama", "Romance"] } },
               { title: 1, genres: 1 })
```

**Explication :** `$all` impose que les deux valeurs soient présentes dans le tableau `genres`.

### 4.5 Films sans champ `rated`

```
db.movies.find({ rated: { $exists: false } },
               { title: 1 })
```

**Explication :** `$exists: false` permet de sélectionner les documents qui n'ont pas le champ `rated`.

## 5 Agrégations avec `aggregate()`

**Méthode `aggregate([])`** Une agrégation est un **pipeline** composé d'étapes successives, comme :

- `$match` : filtrer les documents (équivalent du WHERE SQL)
- `$group` : regrouper et calculer (équivalent du GROUP BY)
- `$sort` : trier les documents
- `$unwind` : éclater un tableau en plusieurs documents
- `$project` : sélectionner ou créer des champs calculés

### 5.1 Nombre de films par année

```
db.movies.aggregate([
  { $group: { _id: "$year", total: { $sum: 1 } } },
  { $sort: { _id: 1 } }
])
```

**Explication :**

- `_id: "$year"` indique que le regroupement se fait par année.
- `$sum: 1` incrémente un compteur pour chaque film.

## 5.2 Moyenne des notes IMDb par genre

```
db.movies.aggregate([
  { $unwind: "$genres" },
  { $group: { _id: "$genres",
              moyenne: { $avg: "$imdb.rating" } } },
  { $sort: { moyenne: -1 } }
])
```

**Explication :**

- `$unwind` crée un document distinct pour chaque genre d'un film.
- `$avg` calcule la moyenne des notes IMDb.
- `sort -1` classe les genres du meilleur au moins bon.

## 5.3 Nombre de films par pays

```
db.movies.aggregate([
  { $unwind: "$countries" },
  { $group: { _id: "$countries", total: { $sum: 1 } } },
  { $sort: { total: -1 } }
])
```

**Explications :**

- **$unwind** : décompose le tableau `countries` pour traiter chaque pays individuellement.
- **$group** : regroupe les films par pays et calcule le nombre total de films par pays.
- **$sort** : trie les résultats par nombre de films décroissant.

## 5.4 Films triés par note IMDb

```
db.movies.aggregate([
  { $sort: { "imdb.rating": -1 } },
  { $project: { title: 1, "imdb.rating": 1 } }
])
```

**Explication :** `$project` permet d'afficher uniquement les champs nécessaires.

# 6 Mise à jour de documents

**Méthodes principales**

- `updateOne(filter, update)` : modifie un seul document.
- `updateMany(filter, update)` : modifie plusieurs documents.

## 6.1 Ajouter un champ

```
db.movies.updateOne(
  { title: "Jaws" },
  { $set: { etat: "culte" } }
)
```

**Explication :**

- Le premier paramètre `{ title: "Jaws" }` est le **filtre** : il sélectionne le film dont le titre est « Jaws ».
- Le second paramètre `{ $set: { etat: "culte" } }` est l'**opération de mise à jour** :
  - `$set` crée ou met à jour le champ `etat` du document.
  - Ici, le champ `etat` prendra la valeur `"culte"`.

## 6.2 Incrémenter une valeur

```
db.movies.updateOne(
  { title: "Inception" },
  { $inc: { "imdb.votes": 100 } }
)
```

**Explication :** Cette commande utilise la méthode `updateOne` pour modifier un seul document dans la collection `movies`.

- Le filtre `{ title: "Inception" }` sélectionne le film dont le titre est « Inception ».
- L'opérateur `$inc` permet d'incrémenter la valeur d'un champ numérique :
  - Ici, le champ `imdb.votes` sera augmenté de 100.
  - Si le champ n'existe pas, MongoDB le crée et lui assigne la valeur de l'incrément (ici 100).

## 6.3 Supprimer un champ

```
db.movies.updateMany({}, { $unset: { poster: "" } })
```

**Explication :** Cette commande utilise la méthode `updateMany` pour modifier plusieurs documents dans la collection `movies`.

- Le filtre `{}` est vide, ce qui signifie que la mise à jour s'applique à **tous les documents** de la collection.
- L'opérateur `$unset` supprime un champ existant :
  - Ici, le champ `poster` est supprimé de tous les documents.
  - La valeur associée (ici une chaîne vide `""`) est ignorée ; l'important est le nom du champ.

## 6.4 Modifier le réalisateur

```
db.movies.updateOne(
  { title: "Titanic" },
  { $set: { directors: ["James Cameron"] } }
)
```

**Explication :** `updateOne` modifie un seul document. Ici, on sélectionne le film dont le titre est `"Titanic"` et on remplace son champ `directors` par un tableau contenant uniquement `"James Cameron"` au moyen de l'opérateur `$set`.

# 7 Requêtes complexes

## 7.1 Films les mieux notés par décennie

```
db.movies.aggregate([
  { $match: { "imdb.rating": { $exists: true } } },
  { $project: {
      title: 1,
      decade: { $subtract: ["$year", { $mod: ["$year", 10] }] },
      "imdb.rating": 1
  }},
  { $group: { _id: "$decade", maxRating: { $max: "$imdb.rating" } } },
  { $sort: { _id: 1 } }
])
```

**Explication :** Cette commande utilise plusieurs étapes d'agrégation pour trouver les films les mieux notés par décennie :

- **$match : { "imdb.rating" : { $exists : true } }** : filtre les films qui ont un champ `imdb.rating` existant.
- **$project** : crée de nouveaux champs et sélectionne ceux à afficher :
  - `title: 1` : conserve le titre du film.
  - `decade` : calcule la décennie du film avec `$subtract` et `$mod` :
    - `$mod: ["$year", 10]` calcule le reste de la division de l'année par 10.
    - `$subtract: ["$year", ...]` soustrait ce reste pour obtenir l'année du début de la décennie.
  - `"imdb.rating": 1` : conserve la note IMDb.
- **$group : { _id : "$decade", maxRating : { $max : "$imdb.rating" } }** : regroupe les films par décennie et calcule la note maximale ($max) pour chaque groupe.
- **$sort : { _id : 1 }** : trie les résultats par décennie croissante.

## 7.2 Titre commençant par “Star”

```
db.movies.find({ title: /^Star/ }, { title: 1 })
```

Cette commande utilise une expression régulière pour sélectionner les titres commençant par « Star ». La projection `title: 1` affiche uniquement le titre des films.

## 7.3 Films avec plus de 2 genres

```
db.movies.find(
  { $where: "this.genres.length > 2" },
  { title: 1, genres: 1 }
)
```

**Explication :** L’opérateur `$where` permet d’exécuter du JavaScript dans la requête. Ici, on vérifie que le tableau `genres` contient plus de deux éléments. Attention : cette méthode est lente et déconseillée en production.

## 7.4 Films de Christopher Nolan

```
db.movies.find(
  { directors: "Christopher Nolan" },
  { title: 1, year: 1, "imdb.rating": 1 }
)
```

**Explication :** Cette commande utilise `find()` pour sélectionner les films réalisés par Christopher Nolan :

- Le filtre `{ directors: "Christopher Nolan" }` sélectionne tous les documents où le tableau `directors` contient `"Christopher Nolan"`.
- La projection `{ title: 1, year: 1, "imdb.rating": 1 }` indique que seules les informations suivantes seront affichées :
  - `title` : le titre du film,
  - `year` : l’année de sortie,
  - `imdb.rating` : la note IMDb.

# 8 Indexation

## 8.1 Créer un index sur `year`

```
db.movies.createIndex({ year: 1 })
```

`createIndex` crée un index sur le champ `year`. La valeur `1` indique un tri croissant. Cela permet d'accélérer les recherches utilisant le champ `year`.

## 8.2 Voir les index existants

```
db.movies.getIndexes()
```

**Explication :** Assez explicite celle là.

## 8.3 Comparer une requête avec `explain()`

```
db.movies.find({ year: 1995 }).explain("executionStats")
```

L'option `explain("executionStats")` affiche des statistiques d'exécution :
- `totalDocsExamined` : nombre de documents examinés,
- `executionTimeMillis` : temps d'exécution,
- utilisation ou non d'un index.

Cela permet de mesurer précisément les gains apportés par un index.

## 8.4 Supprimer un index

```
db.movies.dropIndex({ year: 1 })
```

Supprime l'index correspondant exactement à la définition `year: 1` .

## 8.5 Index composé

```
db.movies.createIndex({ year: 1, "imdb.rating": -1 })
```

Crée un index composé :
- tri croissant sur `year`,
- tri décroissant sur `imdb.rating`.

Ce type d'index optimise les requêtes triant ou filtrant les films par année puis par note IMDb.