

# Pratique guidée sur MongoDB Atlas

Dans ce document nous allons travailler sur [MongoDB Atlas](#), un service hébergé de MongoDB pour créer une base de données, charger des documents et exécuter des requêtes.

Dans un second temps nous verrons comment travailler sur des bases de données plus importantes avec `mongosh` dans le terminal.

À la fin de cette session, vous devriez avoir une bonne compréhension du langage de requête de MongoDB (MQL).

Objectifs :

- Vous vous connectez à un serveur MongoDB Atlas
- Vous créez une base de données et vous insérez des documents
- Vous construisez des requêtes relativement complexes sur une base de données MongoDB

---

## Plusieurs façons de travailler avec MongoDB

Sur votre machine locale

- [MongoDB Community Edition](#) Auto-hébergé, s'exécute sur votre ordinateur
- [MongoDB Compass](#) - l'outil GUI pour la visualisation
- `mongosh` le CLI (commande line interface) dans le terminal.

Le MongoDB Shell, `mongosh`, est un environnement JavaScript et Node.js **REPL** (Read-Eval-Print Loop) pour interagir avec des serveurs MongoDB.

Vous pouvez également travailler sur MongoDB avec votre langage de script préféré : [python](#), [go](#), ruby, PHP, Java, etc...

Aujourd'hui, nous travaillons sur la version hébergée Atlas.

- [MongoDB Atlas](#) fournit un service hébergé dans le Cloud avec des offres gratuites ou payantes. **Vous n'avez rien à installer.**

MongoDB Atlas donne un cluster hébergé sur lequel nous pouvons créer une base de données, importer des **collections** d'exemples et comprendre comment effectuer des opérations CRUD dans MongoDB.

**CRUD** signifie Create, Read, Update et Delete : les quatre opérations de base pour un SGBD

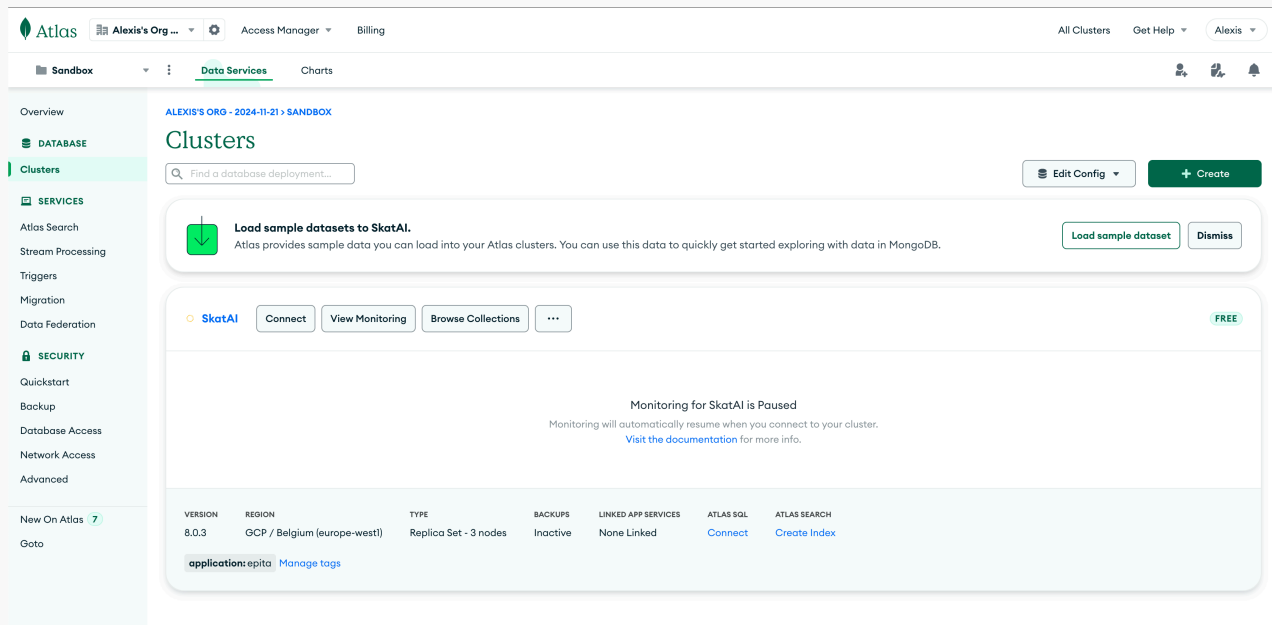
---

## Atlas

Commençons par créer un compte sur Atlas <https://www.mongodb.com/cloud/atlas/register>

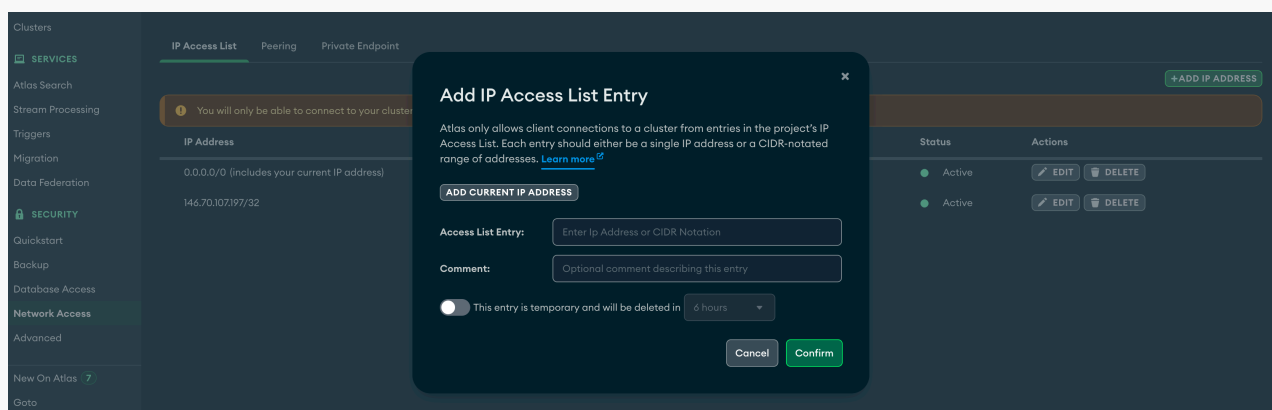
Ensuite :

- Créez un cluster et un projet (j'ai appelé mon projet `Sandbox`)



## Important

- Copiez votre mot de passe de base de données.
- Vérifiez que votre adresse IP est autorisée. Visitez `Network Access` (navigation de gauche) et ajouter la à la liste
- Pour autoriser l'accès depuis n'importe quelle IP ajouter 0.0.0.0/0. C'est déconseillé pour la production mais plus facile pour notre contexte.



## Ensuite créez une base de données MongoDB

- Allez dans `clusters`
- Créez une base de données : ajoutez le nom `songsdb` et le nom de collection `songs`,

Note: ne pas confondre



- le projet : `Sandbox`
- la base de données: `songsdb`
- la collection (table): `songs`

Puis insérez le Document suivant



```
JavaScript
{
  "_id":{
    "$oid":"6745ab6f0e0bbdab062667c7"
  },
  "title": "Happy",
  "artist": "Pharrell Williams",
  "year": 2013,
  "mood": "joyful"
}
```

## Insert Document

To collection music

VIEW  

```
1 {
2   "_id":{
3     "$oid":"6745ab6f0e0bbdab062667c7"
4   },
5   "title": "Happy",
6   "artist": "Pharrell Williams",
7   "year": 2013,
8   "mood": "joyful"
9 }
10
```

Cancel Insert

Et insérez-en un autre

```
{
  "_id":{
    "$oid":"abcdeab6f0e0bbdab062667aa"
  },
  "title": "Highway to hell",
  "artist": "AC/DC",
  "year": 1981,
  "mood": "energetic"
}
```

Dans les 2 cas, la valeur de l'`_id` est arbitraire. L'outil vous en propose une par défaut avec une autre valeur.

Nous avons donc 2 documents !

2 documents, wow c'est méga excitant 🤪🤪🤪 !

Explorons cette base de données MongoDB.

---

## Requêtes dans MongoDB

Dans MongoDB, écrire une requête se résume à écrire du `JSON`

json	requête
<code>{}</code>	retourne tous les documents
<code>{ field : value }</code>	où field = value
<code>{ field : { \$lt : value } }</code>	où field <= value (lt : less than)

Donc si nous voulons trouver tous les documents dans notre collection `songs`, écrivez simplement `{}` dans le champ de requête

Cela retourne les 2 chansons, les 2 documents.

musicdb.music

STORAGE SIZE: 32KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 32KB

Find Indexes Schema Anti-Patterns 0 Aggregation Search Indexes

Generate queries from natural language in Compass [↗](#) INSERT DOCUMENT

Filter [↗](#) {} Reset Apply Options ▶

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('674c08a3f3d84fb68136be86')
title: "Happy"
artist: "Pharrell Williams"
year: 2013
mood: "joyful"
```

```
_id: ObjectId('674c08c9f3d84fb68136be87')
title: "Highway to hell"
artist: "AC/DC"
year: 1981
mood: "energetic"
```

Et la requête `{ year: { $lt: 2000 } }` retourne la chanson qui a été publiée avant 2000.

Filter [↗](#) {year : { \$lt : 2000 } } Reset Apply Options ▶

QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('674c08c9f3d84fb68136be87')
title: "Highway to hell"
artist: "AC/DC"
year: 1981
mood: "energetic"
```

## Plussssss de données ! (moaaaaaarrrr data)

2 documents, c'est bien mais nous avons besoin de plus de données pour pouvoir vraiment jouer.

Importons maintenant les jeux de données d'exemple fournis par ATLAS.

Allez dans `Clusters > dots > load sample dataset`

# Clusters

[Connect](#)
[View Monitoring](#)
[Browse Collections](#)
[Edit Configuration](#)
[Command Line Tools](#)
[Load Sample Dataset](#)
[Terminate](#)


## Enhance Your Experience

For production throughput and richer metrics, upgrade to a dedicated cluster now!

[Upgrade](#)

● R 0.07

● W

Last 5 hours

0.2/s



18.0



Puis cliquez sur le nom de votre projet (pour moi `Sandbox`) et `Browse collections` pour voir les bases de données disponibles.

Vous obtenez une nouvelle base de données `movies_mflix` avec 5 collections.

sample_mflix							
LOGICAL DATA SIZE: 115.94MB STORAGE SIZE: 102.46MB INDEX SIZE: 19.84MB TOTAL COLLECTIONS: 6							
Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
<code>comments</code>	41079	11.14MB	285B	6.45MB	1	1.91MB	1.91MB
<code>embedded_movies</code>	3483	71.9MB	21.14KB	75.45MB	1	204KB	204KB
<code>movies</code>	21349	32.54MB	1.56KB	20.34MB	2	17.54MB	8.77MB
<code>sessions</code>	1	540B	540B	20KB	2	40KB	20KB
<code>theaters</code>	1564	341.63KB	224B	164KB	2	112KB	56KB
<code>users</code>	185	28.88KB	160B	36KB	2	48KB	24KB

Regardez la collection `embedded_movies` qui contient 1525 documents et observez la structure d'un document

## Un document est un enregistrement JSON

Il peut avoir :

- du **JSON imbriqué** : regardez les dictionnaires imbriqués `imdb` et les champs `tomatoes`.
- des **arrays** : regardez genres, cast, languages, writers, ...

La clé primaire d'une collection est toujours "\_id".

JavaScript

```
{
  "_id": {
    "$oid": "573a1390f29313caabcd5293"
  },
  "plot": "Young Pauline is left a lot of money when her wealthy uncle dies. H",
  "genres": [
    "Action"
  ],
  "runtime": {
    "$numberInt": "199"
  },
  "cast": [
    "Pearl White",
    "Crane Wilbur",
    "Paul Panzer",
    "Edward José"
  ],
  "num_mflix_comments": {
    "$numberInt": "0"
  },
  "poster": "https://m.media-amazon.com/images/M/MV5BMzgxODk1Mzk2Ml5BMl5BanBnX",
  "title": "The Perils of Pauline",
  "fullplot": "Young Pauline is left a lot of money when her wealthy uncle die",
  "languages": [
    "English"
  ],
  "released": {
    "$date": {
      "$numberLong": "-1760227200000"
    }
  },
  "directors": [
    "Louis J. Gasnier",
    "Donald MacKenzie"
  ],
  "writers": [
    "Charles W. Goddard (screenplay)",
    "Basil Dickey (screenplay)",
    "Charles W. Goddard (novel)",
    "George B. Seitz",
    "Bertram Millhauser"
  ],
  "awards": {
    "wins": {
      "$numberInt": "1"
    },
    "nominations": {
      "$numberInt": "0"
    }
  }
}
```

```

    },
    "text": "1 win."
  },
  "lastupdated": "2015-09-12 00:01:18.647000000",
  "year": {
    "$numberInt": "1914"
  },
  "imdb": {
    "rating": {
      "$numberDouble": "7.6"
    },
    "votes": {
      "$numberInt": "744"
    },
    "id": {
      "$numberInt": "4465"
    }
  },
  "countries": [
    "USA"
  ],
  "type": "movie",
  "tomatoes": {
    "viewer": {
      "rating": {
        "$numberDouble": "2.8"
      },
      "numReviews": {
        "$numberInt": "9"
      }
    },
    "production": "Pathè Frères",
    "lastUpdated": {
      "$date": {
        "$numberLong": "1441993579000"
      }
    }
  }
}

```

L'interface utilisateur ATLAS est bien faite mais je préfère travailler avec du code plutôt qu'une interface web.

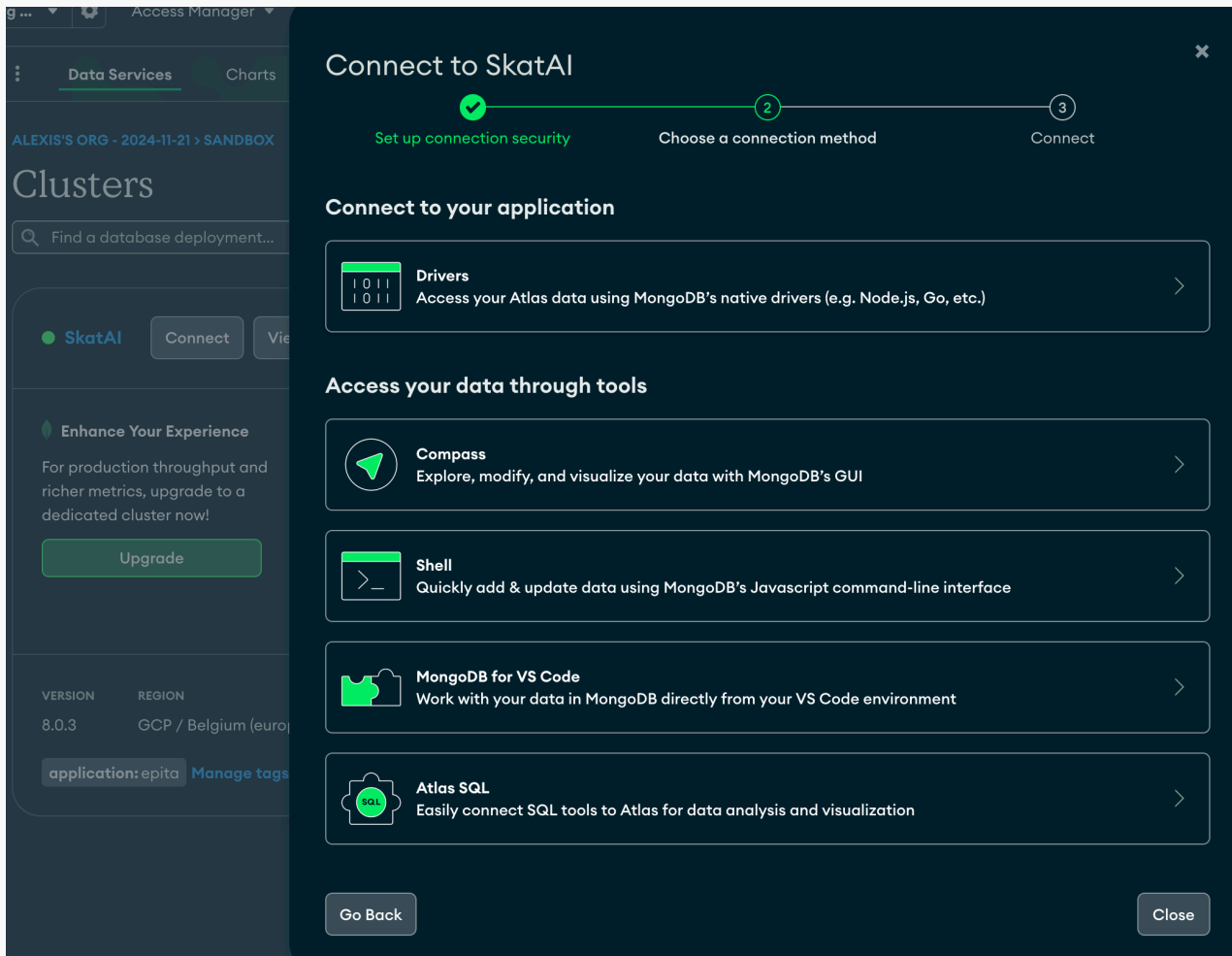
Passons à python et à `mongosh` pour explorer cette base de données de films et apprendre à faire des requêtes dans MongoDB.

---

## Se connecter via votre langage



Atlas vous permet de vous connecter à votre cluster



## Reset et gestion du mot de passe

Si vous avez oublié le mot de passe de votre cluster

## Se Connecter avec Mongosh

Sauver la string de connection dans une variable environnement

```
mongodb+srv://username:mot_de_passe@cluster_id.mongodb.net/
```

Par exemple

```
export MONGO_ATLAS_URI=mongodb+srv://alexis:mot_de_passe@skatai.w932a.mongodb.net/
```

Bash

Puis se connecter avec

```
mongosh ${MONGO_ATLAS_URI}
```

Bash

## Se connecter en Python

Nous avons besoin du package [pymongo](#)

```
pip install pymongo
```

Bash

La string de connexion est

```
connection_string = "mongodb+srv://alexis:<db_password>@skatai.w932a.mongodb.net"
```

Python

Note : Si vous travaillez en python, il est préférable de mettre la string de connexion comme variable d'environnement ( `$MONGO_ATLAS_URI` ), dans un fichier `.env` . puis la charger avec `dotenv`

```
import os
from dotenv import load_dotenv

load_dotenv()

MONGO_ATLAS_URI = os.getenv('MONGO_ATLAS_URI')
```

Python

Et ensuite d'instancier le client avec

```
from pymongo import MongoClient

client = MongoClient(MONGO_ATLAS_URI)
```

Python

Une fois que nous avons un client, nous pouvons nous connecter à la base de données

```
db = client["sample_mflix"]
```

Python

puis instancier une collection

```
collection = db["movies"]
```

Python

La collection est de la classe `pymongo.synchronous.collection.Collection` et possède de nombreuses méthodes :

```
aggregate()      count_documents()  create_search_indexes()  distinct()      drop_search_index()  find_one_and_delete()
aggregate_raw_batches()  create_index()      database                drop()           estimated_document_count()  find_one_and_replace()
bulk_write()      create_indexes()    delete_many()           drop_index()     find()               find_one_and_update()
codec_options     create_search_index()  delete_one()            drop_indexes()   find_one()           find_raw_batches()
function(pipeline: 'Pipeline', session: 'Optional[ClientSession]'=None, let: 'Optional[Mapping[str, Any]]'=None, comment: 'Optional[Any]'=None, **kwargs: 'Any')
```

## Utiliser le terminal avec `mongosh`

Comment installer `mongosh` : <https://www.mongodb.com/docs/mongodb-shell/install/>

Le langage utilisé dans le shell MongoDB (`mongosh`) est du **JavaScript**.

`mongosh` est une interface JavaScript interactive pour MongoDB, vous permettant d'interagir avec vos instances MongoDB, d'exécuter des requêtes ou d'effectuer des tâches administratives en JavaScript.

`mongosh` prend également en charge de nombreuses fonctionnalités JavaScript y compris l'utilisation de variables, de boucles et de fonctions.

Voici quelques exemples d'utilisation de JavaScript dans `mongosh` :

- **Se connecter à une instance MongoDB :**

Locale

```
mongosh "mongodb://localhost:27017"
```

Bash

Sur Atlas

```
mongosh mongodb+srv://alexis:mot_de_passe@skatai-ipsa.w932a.mongodb.net/
```

Bash

ou

```
mongosh ${MONGO_ATLAS_URI}
```

Bash

Une fois connecté à une instance

- **voir les bases de données :**

```
show dbs
```

JavaScript

- **Changer de base de données :**

```
use myDatabase
```

JavaScript

- **voir les collections :**

```
show collections
```

JavaScript

- **Insérer un document**

JavaScript

```
db.songs.insertOne(  
  {  
    "title": "Green Onions",  
    "artist": "Booket T and the MGs",  
    "year": 1967,  
    "mood": "[blues, soul]"  
  }  
)
```

JavaScript

```
db.users.updateOne(  
  { "artist": "Booket T and the MGs" },  
  { $set: { "artist": "Booker T and the MGs" } }  
~~~~  
  
``js  
  
db.songs.insertOne(  
  {  
    "title": "Happy",  
    "artist": "Pharrell Williams",  
    "year": 2013,  
    "mood": "joyful"  
  }  
)
```

# Insert Document

To collection music

VIEW



```
1 ▼ {
2 ▼   "_id":{
3     "$oid":"6745ab6f0e0bbdab062667c7"
4   },
5   "title": "Happy",
6   "artist": "Pharrell Williams",
7   "year": 2013,
8   "mood": "joyful"
9 }
10
```



Cancel

Insert

Et insérez-en un autre

```
db.songs.insertOne(
  {
    "title": "Highway to hell",
    "artist": "AC/DC",
    "year": 1981,
    "mood": "energetic"
  })
```

JavaScript

- Executer une requête

```
db.songs.find({ year: { $lt: 2000 } })
```

JavaScript

- Mettre à jour un document

```
db.myCollection.updateOne({ name: "Alice" }, { $set: { age: 31 } })
```

JavaScript

- Supprimer un document :

```
db.myCollection.deleteOne({ name: "Alice" })
```

JavaScript

- Utiliser des variables et fonctions JavaScript

```
var name = "Alice";
db.myCollection.find({ name: name }).forEach(printjson);
```

JavaScript

`mongosh` est un outil puissant pour interroger et gérer vos bases de données MongoDB.

## Différence entre les requêtes en python et en javascript

python : double guillemets autour des champs et opérateurs

```
db.movies.find(
  {"runtime": {"$gt" : 180}}, // Filter on movie duration
  { "_id": 0, "title": 1, "runtime": 1, "imdb.rating": 1 } // Projection to include
)
```

Python

`mongosh` : pas besoin de guillemets

```
db.movies.find(
  {runtime: {$gt : 180}}, // Filter on movie duration
  { _id: 0, title: 1, runtime: 1, "imdb.rating": 1 } // Projection to include
)
```

JavaScript

## Filtrage

<https://www.mongodb.com/docs/manual/reference/glossary/>

Le JSON qui spécifie les arguments de filtrage est appelé un **prédicat de requête** (query predicate). C'est une expression qui renvoie un booléen indiquant si un document correspond à la requête spécifiée.

Par exemple, `{ title: { $eq: "Top Gun" } }`, qui renvoie les documents qui ont un champ "title" dont la valeur est "Top Gun".

Un prédicat de requête vide (`{ }`) retourne tous les documents de la collection.

Fonctions principales sur les collections

fonction	retourne
<code>find()</code>	tous les documents
<code>find_one()</code>	le 1er document
<code>distinct("&lt;field&gt;")</code>	liste des valeurs distinctes pour le <code>&lt;field&gt;</code>
<code>count_documents({})</code>	nombre de documents pour la collection ou retournés par le filtre dans le prédicat de requête

Notez aussi

- `find_one_and_replace()`
- `find_one_and_update()` et
- `delete_many()`
- `delete_one()`
- `drop_index()`
- etc ...

Note : Vous pouvez également interroger directement la collection depuis le client avec `db.<collection_name>.find()` sans avoir à instancier un objet `collection`.

```
collection.find({})
# ou
db.movies.find({})
```

Bash

Vous pouvez enchaîner ces méthodes avec `limit` et `sort`

```
db.movies.find({runtime: {$gt: 120}}).limit(3)
```

Bash

## Curseur

Le résultat retourné est un **curseur**.

```
cursor = db.movies.find({})
```

JavaScript

Un curseur est un pointeur sur un ensemble de résultats de requête MongoDB.

## Projection

Dans le langage des bases de données, **projeter** signifie sélectionner un sous-ensemble de tous les champs possibles.

En SQL, vous listez simplement les noms des colonnes

SQL

```
select genres, plot from movies;
```

Dans MongoDB, il faut spécifier les champs dans un objet JSON, juste après le prédicat de requête

JavaScript

```
db.movies.find(
  {runtime: {$gt : 180}}, // Filter on movie duration
  { _id: 0, title: 1, runtime: 1, "imdb.rating": 1 } // Projection to include
)
```

Ici la projection est exprimé par : `{_id: 0, title: 1, runtime: 1, "imdb.rating": 1 }`

- `title: 1` : inclut le champ `title`.
- `runtime: 1`, inclut le `runtime`
- `"imdb.rating": 1` : Inclut le champ `imdb.rating`.
- `_id: 0` : Exclut le champ `_id` du résultat (la valeur par défaut est 1 si non spécifié).

La requête retourne

JavaScript

```
{ runtime: 240, title: 'Napoleon', imdb: { rating: 7.4 } },
{ runtime: 281, title: 'Les Misérables', imdb: { rating: 7.9 } },
{ runtime: 245, title: 'Flash Gordon', imdb: { rating: 7.3 } },
{ runtime: 238, title: 'Gone with the Wind', imdb: { rating: 8.2 } },
```

## Exercices

### En python

Exécutons quelques requêtes en Python ou `mongosh` sur la base de données `movies`

Python

```
import os
from pymongo import MongoClient

connection_string = os.getenv('MONGO_ATLAS_URI')
client = MongoClient(connection_string)
db = client["sample_mflix"]
```

Ensuite

- Récupérer le titre et les genres des films qui ont le genre "Action".

Python

```
cursor = db.movies.find(
  {"genres": "Action"}, # Filter: movies with 'Action' in the genres array
  {"_id": 0, "title": 1, "genres": 1} # Projection: include title and genres,
)
```



pour voir les résultats avec python

```
for movie in cursor:  
    print(movie)
```

Python

avec `mongosh` :

une fois connecté

```
cursor = db.movies.find(  
    {"genres": "Action"}, # Filter: movies with 'Action' in the genres array  
    {"_id": 0, "title": 1, "genres": 1} # Projection: include title and genres,  
)
```

JavaScript

## Nombre de documents

La manière la plus rapide de compter les documents est d'utiliser `count_documents`

```
count = db.movies.count_documents({"imdb.rating": {"$gt": 8.0}})
```

Python

Note : en Python, vous pouvez cloner le curseur pour obtenir sa longueur et le nombre de documents retournés. Cloner le curseur ne le consomme pas

```
len(list(cursor.clone()))
```

Python

avec `mongosh`

```
db.movies.countDocuments({ "imdb.rating": { $gt: 8.0 } })
```

JavaScript

## À vous de jouer

avec

```
cursor = db.movies.find( filter, projection).limit(5)  
for movie in cursor:  
    print(movie)
```

Python

Écrivez le filtre et la projection pour les requêtes suivantes et retournez également le nombre de documents avec `db.movies.count_documents(filter)`

- utilisez la projection pour ne retourner que les champs pertinents ou au minimum "title"
- limitez les résultats à 5 documents

1. Trouver les films avec une note IMDb supérieure à 8

- filter : `{"imdb.rating": {"$gt" : 8}}`
  - projection : `{"_id": 0, "title": 1, "imdb.rating": 1}`
2. Films sortis après 2000
  3. Films avec un réalisateur spécifique : "Christopher Nolan". Montrer le titre, le réalisateur et l'année
  4. Récupérer les films avec `tomatoes.viewer.rating > 4.0`, montrant le titre et la note des spectateurs.
  5. Trouver les films qui contiennent "Comedy" et "Drama" dans le tableau `genres`. Utiliser `{ $all: [list of genres]}`
  6. Combiner une requête avec un tri : Récupérer les 5 meilleurs films avec la note IMDb la plus élevée, montrant le titre et la note. (vous devez uniquement récupérer `imdb.rating` avec le type de données `double`)
  7. Requête de films sur une plage d'années : Récupérer les films sortis entre 1990 et 2000, montrant le titre et l'année.
  8. Requête de films avec des champs manquants : Trouver les films où le champ `fullplot` n'existe pas. Utiliser `$exists`.
  9. Trouver tous les genres distincts
    - utiliser `db.movies.distinct("genres")`
  10. Films avec au moins 2 genres
    - utiliser `{"genres": {$size: 2}}`
  11. Films du genre Action, après 1950 avec des notes imdb > 8, trier par année desc, note imdb desc
    - utiliser `{ "year": {$gt: 1950}, "imdb.rating": {$gt: 8}, "genres": "Action" }`
  12. Films avec les deux genres : Action et Drama
    - `$and: [ {"genres": "Action"}, {"genres": "Drama"} ]`
  13. Films avec soit Action soit Drama
    - `$or: [ {"genres": "Action"}, {"genres": "Drama"} ]`
  14. Films après 1950 avec soit imdb.rating > 0 soit awards.wins > 5
    - utiliser : `{ "year": {$gt: 1950}, $or: [ {"imdb.rating": {$gt: 8}}, {"awards.wins": {$gt: 5}} ] }`

---

## Conclusion

Dans cette session vous avez pratiqué :

- La configuration de MongoDB Atlas, un service de base de données hébergé dans le cloud, y compris la création de cluster et la configuration de sécurité
- L'écriture de requêtes MongoDB de base utilisant le format JSON :
  - Syntaxe de base : `{}` pour tous les documents, `{field: value}` pour l'égalité, `{field:`

`{{lt: value}}` pour les comparaisons

- Comment interroger les champs imbriqués et les tableaux dans des documents complexes
- La connexion à MongoDB Atlas en utilisant Python et `pymongo` :
  - Utilisation des opérations de base : `find()`, `find_one()`, `distinct()`, `count_documents()`
  - Implémentation des projections pour sélectionner des champs spécifiques
- Le travail avec des jeux de données d'exemple (particulièrement la base de données movies) pour pratiquer :
  - Le filtrage et le tri des données
  - Le travail avec des champs imbriqués
  - L'utilisation d'opérateurs comme `$gt`, `$lt`, `$all`, `$exists`
  - L'écriture de requêtes combinées avec plusieurs conditions

Dans la prochaine session, nous plongerons plus profondément dans MongoDB et examinerons des façons plus complexes d'interroger les données en utilisant les **pipelines d'agrégation**. Nous aborderons également la conception et la validation des schémas.

---

## Pour aller plus loin

Pour la prochaine fois, vous pouvez :

- explorer davantage les bases de données exemple d'Atlas et pratiquer l'écriture de requêtes
- suivre <https://www.mongodb.com/docs/languages/python/pymongo-driver/current/read/> pour plus de pratique
- Il y a de nombreux cours et tutoriels gratuits dans l'université MongoDB
  - [Intro to MongoDB](#)
  - [CRUD in python](#)

et bien plus encore