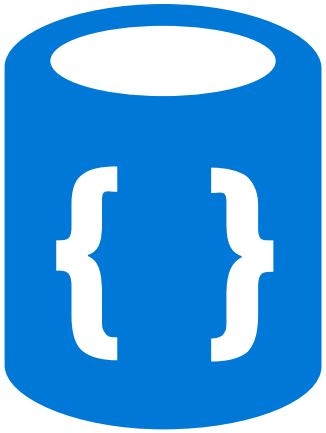


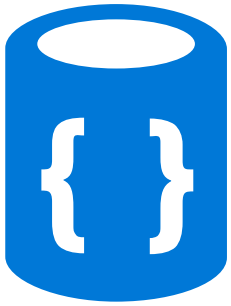
Base de données NoSQL

Nouveau paradigme, MongoDB

Agence nationale pour la formation professionnelle des adultes

- Objectifs pédagogiques :

- Se familiariser au nouveau paradigme des bases de données **NoSQL**
- Maîtriser les concepts autours du **NoSQL**
- Construire/manipuler une base **NoSQL** avec **MongoDB**



Paradigme NoSQL

Différence avec les SBGDR

- Années 2000 → **forte augmentation des données** sur le « Web 2.0 » notamment avec l'émergence des GAFAMs et des réseaux sociaux (Google, Apple, Amazon, Microsoft…)

- Les SGBD relationnels ont présenté des **limitations** face à ces importants volumes de données.

- Nouvelle façon de gérer les données

  → Émergence des bases de données **NoSQL** (**N**ot **O**nly **SQL**)

  qui sont **non relationnelles**.

- SGBD NoSQL :
- Popularisation du terme « NoSQL » en 2009
- Type de base de données qui s'écarte du fonctionnement relationnel

→ nouveau **paradigme ***

- Objectifs NoSQL :
- Se soustraire de certaines contraintes du SGBDR
- Dénormaliser les modèles → dupliquer les données
- Favoriser les performances de requêtage au détriment de l'intégrité
- Favoriser l'évolutivité (« scalability » **) des systèmes

- « Scalability »
- Capacité des systèmes à **s'adapter** à une **modification de son contexte**
- Exemple : beaucoup plus d'utilisateur du jour au lendemain

* Un paradigme est une représentation, vision du monde, modèle, courant de pensées, point de vue

** La scalabilité est la capacité d'un système informatique à s'adapter d'un point de vue dimensionnel, tant vers des tailles inférieures que vers des tailles supérieures

Dans un **SGBDR**, les transactions sont dites **ACID**

- **Atomicité** : Toutes les opérations d'une transaction sont effectuées, autrement aucune n'est réalisée.

- **Cohérence :** Le contenu de la base doit être cohérent (valide) par rapports aux contraintes d'intégrités établies.

- **Isolation :** Les modifications d'une transaction ne sont visibles que quand celle-ci a été validée afin d'éviter les problèmes de concurrence.

- **Durabilité :** Une fois la transaction validée, l'état de la base est permanent (quel que soit l'incident qui survient, panne, coupure, etc.)

Base NoSQL → moins contraignantes pour favoriser les performances sur de grandes quantités de données. Elles sont **BASE**

- **B**asically **A**vailable : Le système garantit un taux de disponibilité de la donnée via plusieurs mécanismes (notamment la duplication de l'information sur plusieurs serveurs → SGBD distribué)

- **S**oft-state : La base n'est pas nécessairement cohérente à tout instant

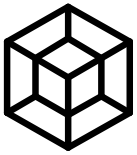
- **E**ventually consistent : La base atteindra, à terme, un état cohérent

MongoDB

Base de données orientée **document**

- base de données **NoSQL** orientée **document**
- stocke les données sous forme de **documents** dans un **format similaire au JSON** (BSON, *forme binaire* du JSON)
- **Open Source et Multiplateforme** qui offre des performances et une évolutivité (« scalability ») élevée.
- Fortement lié à Javascript → « Shell Mongosh » peut être utilisé pour interagir avec MongoDB
- Documentation :

  →https://www.mongodb.com/developer/



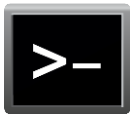
Installation du serveur en local

```
winget install -e --id MongoDB.Server
```

Image Docker utilisable

→ https://hub.docker.com/_/mongo

→ https://github.com/afpa-learning/mongodb-docker



Client en ligne de commande → MongoDB Shell (mongosh)

```
winget install -e --id MongoDB.Shell
```

Client graphique → Mongo Compass

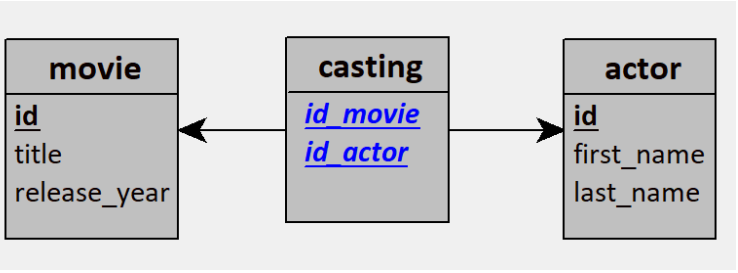
```
winget install -e --id MongoDB.Compass.Community
```

Différences de terminologie :

SGBDR	MongoDB
Table	Collection
Tuple / ligne / enregistrement	Document
Colonne	Champ
Jointure	Document incorporé
Clé primaire	Clé primaire (clé par défaut « _id » créée par MongoDB)

- Dénormalisation :

- Action **d'intégrer des données d'une entité à une autre**
- Réduit le nombre de jointures (clés étrangères)
- Permet d'**accélérer** la lecture des données
- Conduit à de la **redondance de données** et des **opérations de mise à jour plus complexes**



Relationnel → BDD normalisée

```
[
    {
        "_id": "movie:11",
        "title": "La Guerre des étoiles",
        "release_year": 1977,
        "actors": [
            {
                "last_name": "Ford",
                "first_name": "Harrison"
            }
        ]
    },
    {
        "_id": "movie:24",
        "title": "Kill Bill : Volume 1",
        "release_year": 2003,
        "actors": [
            {
                "last_name": "Thurman",
                "first_name": "Uma",
                "birth_date": 1970
            },
            {
                "last_name": "Liu",
                "first_name": "Lucy",
                "birth_date": 1968
            }
        ]
    }
]
```

NoSQL → BDD dénormalisée

Suppression FK → redondance de données

Manipulation de base de données MongoDB

Instruction Javascript

- Créer/utiliser une base de données :

```
// crée la BDD "db_movies" elle n'existe pas déjà
// cette base sera constituée de plusieurs
collections
use db_movies;
```

- Créer une collection :

```
// crée la collection "actors"
// cette collection sera constituée de plusieurs
documents (JSON)
db.createCollection("actors");
```

- Supprimer une collection :

```
// supprime la collection "actors"
db.actors.drop();
```

Action sur une collection → appel d'une méthode

```
db.<nom-collection>.<méthode>
```

- Insertion de document (JSON) :

```
db.actors.insert(<document>)
db.actors.insertOne(<document>)
db.actors.insertMany(<document>)
```

Type : JSON ou tableau de JSON

- Récupération d'un JSON de la BDD :

```
db.actors.find(<query>)
db.actors.findOne(<query>)
```

Type : JSON ou tableau de JSON
doit définir la requête

- Supprimer des documents :

```
db.actors.deleteOne(<filter>)
db.actors.deleteMany(<filter>)
```

Type : JSON ou tableau de JSON
Doit préciser l'élément à supprimer

Insertion de document→ appel d'une méthode

```
db.<nom-collection>.insert(<document>)
```

- Exemple d'insertion :

```
db.actors.insertOne({ first_name: "Morena",
                      last_name: "BACCARIN"
                    })
```

- Les documents peuvent avoir des **structures différentes** :

```
db.actors.insertOne({ first_name : "Chris",
                      last_name : "PRATT",
                      birth_date : "21/06/1979"})
```

Sélection de document (select en SQL) → méthode

```
db.actors.find(<query>)
```

- Sélection avec un critère :

```
db.actors.find({first_name: "Chris"})
```

- Sélection avec plusieurs critères :

```
db.actors.find({ first_name: "Chris",
                 last_name: "PRATT"})
```

- Sélection avec un objet imbriqué :

```
db.actors.find({ first_name: "Chris",
                 last_name: "PRATT",
                 city.zip_code: "CA 91506"})
```

Affiner la sélection (projection)→ méthode

```
db.actors.find(<query>, <projection>)
```

- Projection :
- permet de choisir les données qui seront sélectionnées par la requête
- Exemple

```
db.actors.find({ fist_name: "Chris",
                 last_name: "PRATT"
               },
               { last_name: true })
```

```
SELECT last_name FROM actors;
```

Sélection du
nom de famille uniquement

Compter les documents -> méthode Javascript standard

```
db.<nom-collection>.find(<query>, <projection>).count()
```

- Exemple

```
db.actors.find({ fist_name: "Chris",
                 last_name: "PRATT"
               },
               { last_name: true }).count()
```

- Restriction sur une sélection :

$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, $nin	∈, ∉	Fait parti de (ou ne doit pas)	"a" : {"$in" : [10, 12, 15, 18] }
$or	∨	OU logique	"a" : {"$or" : [{"$gt" : 10}, {"$lt" : 5}] }
$and	∧	ET logique	"a" : {"$and" : [{"$lt" : 10}, {"$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}

- Exemples :

```
// récupération du nom de tous les acteurs
// nés avant 1980
db.actors.find({ birth_year: { $lt: 1980 },
                 { last_name: true }})
```

Si birth_year < 1980

```
// récupération de tous les acteurs prénommés Alan
// ou habitant dans le département ayant le code postal 19240
db.actors.find({
    $or  : [{first_name: "Alan"},
            {city.zip_code: "19240" }]
})
```

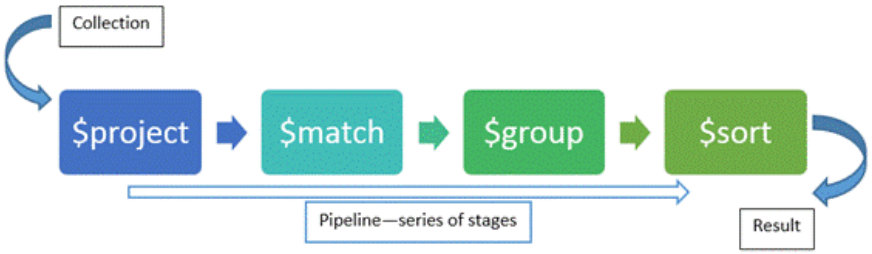
OU logique

- Agrégation avec MongoDB :

→ Les opérations **d'agrégation** traitent les enregistrements de données et renvoient les **résultats calculés**.

- Fonctionnement de l'agrégation:

1. les documents sélectionnés entrent dans un pipeline composé d'étapes (« stages ») de transformation
2. Les documents sont transformés en un résultat agrégé



- Comparaison SQL / MongoDB

SQL	MongoDB
WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM	$sum
COUNT	$count

Agrégation -> méthode

```
db.<nom-collection>.aggregate([stages])
```

- Exemple :

```
// calcul l'âge moyen des acteurs suivant leur genre
// ("femme" / "homme" / "non binaire")
db.actors.aggregate([
    // Stage : regroupement et calcul
    { $group : { "_id " : "$genre",
                 "avg_age": { $avg : "$age" }
               }
    }
])
```

Regroupement par genre

Calcul de la moyenne

Mise à jour de document

```
db.<nom-collection>.update(<query>,<update>)
```

- Mise à jour de données d'un document :

```
db.actor.updateOne({last_name: "PRATT"} , { $set: {fist_name : "Christopher"}}
```

```
db.actor.updateMany({last_name: "PRATT"} , { $set: {fist_name : "Christopher"}}
```

- Mise à jour de la structure d'un document :

```
db.actors.updateMany({ first_name : "Morena",
                       last_name : "BACCARIN"},
                     { $set : {oscar_winner: false} })
```

Création dynamique d'un nouveau champ
« oscar_winner » uniquement pour ce document

Suppression de document

```
db.<nom-collection>.deleteOne(<filter>)
db.<nom-collection>.deleteMany(<filter>)
```

- Exemples :

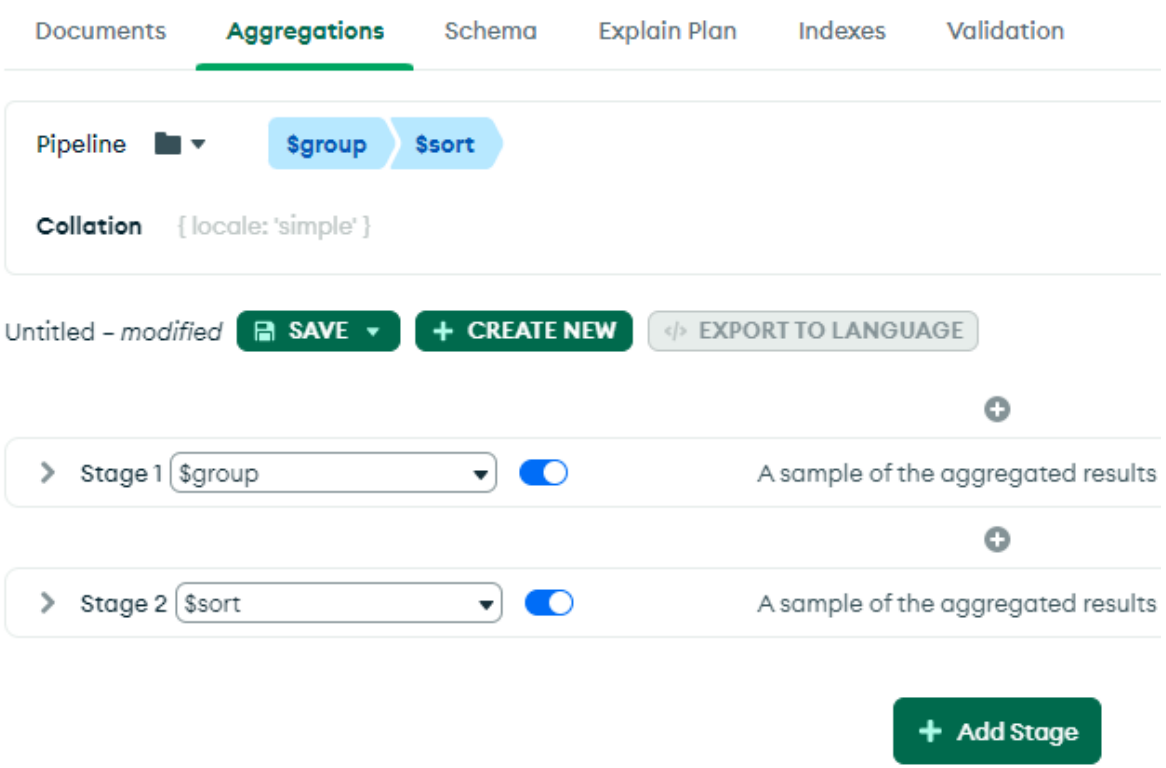
```
// supprimer un seul acteur ayant le nom "BACCARIN"
db.actors.deleteOne({ last_name : "BACCARIN"})
```

```
// supprimer tous les acteurs ayant le nom "BACCARIN"
db.actors.deleteMany({ last_name : "BACCARIN"})
```

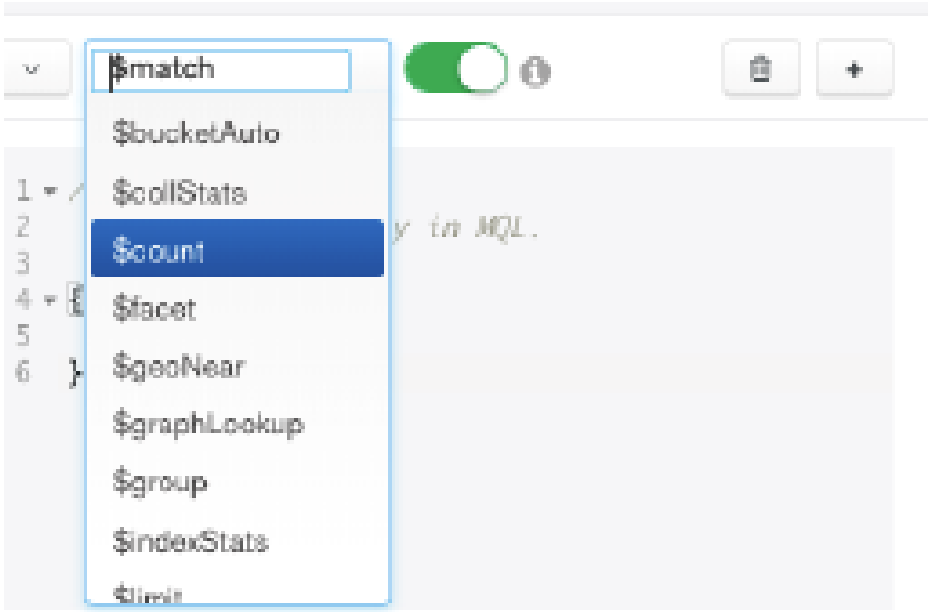
- https://www.mongodb.com/fr-fr
- https://blog.dyma.fr/mongodb-la-base-de-donnees-nosql-la-plus-utilisee/
- https://www.tutorialsteacher.com/mongodb

Manipulation de base de données MongoDB

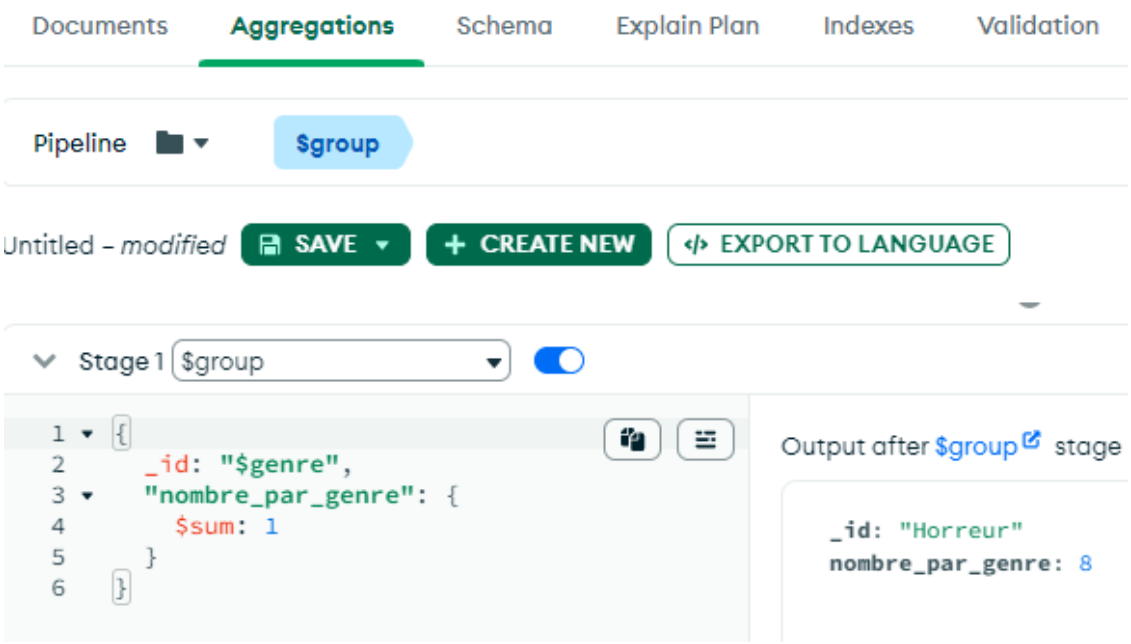
Client graphique – MongoDB Compass



Vous pouvez construire un **pipeline** composé éventuellement de plusieurs **opérateurs** d'agrégation.



Pour chaque **opérateur** vous pouvez définir le **champ** et la **fonction d'agrégation**.



L'interface proposée par **MongoDB Compass** est assez intuitive, elle permet d'obtenir de façon dynamique le résultat de la **requête d'agrégation** que vous aurez composé.

Crédits
Œuvre Collective de l’Afpa

Equipe de conception :
Alexandre Restoueix / Fabrice Le Blanc / Ludovic Esperce
Date de mise à jour : 15/01/2025

Agence nationale pour la formation professionnelle des adultes
Établissement public à caractère industriel et commercial
Tour Cityscope
3 rue Franklin, 93100 Montreuil
824 228 142 RCS BOBIGNY