Bases de données **NoSQL** : **MongoDB**

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Plan

Plan

7. Opérations sur les tableaux d'un document
   - `$push`
   - `$each`
   - `$position`
   - `$pop`
   - `$pull`
   - `$in`
   - `$slice`
   - `$sort`
   - `$[]`
   - `$`
   - `$[element]`
   - `$elemMatch`

Plan

Plan

MongoDB

MongoDB

- Traduit souvent en **Énorme** ou **Gigantesque**
- Système de gestion de base de données faisant partie de la mouvance **NoSQL**
- Orienté documents
- Créé en 2007 par **MongoDB, Inc**
- Open-source
- Développé principalement en **C++**

MongoDB

MongoDB : caractéristiques

- Disponibilité de plusieurs fonctionnalités **SQL** : `COUNT`, `GROUP BY`, `ORDER BY`, `SUM`...
- Possibilité d'accéder aux données via une console **JavaScript**
- Drivers disponibles pour plusieurs langages de programmation : **JavaScript** (**Node.js**), **Java**, **PHP**, **Python**, **Ruby**...
- Données stockées sous format **JSON** (**J**ava**S**cript **O**bject **N**otation) ou **BSON** (format binaire de **JSON**)
- Disponible en tant que service dans le cloud : **AWS** (**A**mazon **W**eb **S**ervices), **Azure** (**Microsoft**), et **GCP** (**G**oogle **C**loud **P**latform) proposent des services MongoDB gérés.
- Utilisée dans une variété d'applications : web, mobiles, systèmes de gestion de contenu (CMS), applications de géolocalisation...

MongoDB

MongoDB : Quelques chiffres

- SGBD NoSQL le plus populaire en 2017, 2021, 2022, 2023 (**DB-Engines**)
- Classé cinquième dans le classement des SGBD **SQL** et **NoSQL** (**DB-Engines**)
- SGBD NoSQL le plus utilisé en **France**
- Plus de 100 millions de téléchargements en 2020
- ...

MongoDB

MongoDB : utilisé par

- De nombreuses grandes entreprises (géants de la technologie) : **Google**, **Facebook**, **Adobe** et **Cisco**
- des entreprises dans des domaines variés : **MTV**, **Disney**, **Doodle**, **eBay**, **Air France**, **Le Figaro**, **Bouygues Telecom**, **SourceForge.net**, **pagesjaunes**, **New York Times**...

MongoDB

SQL vs MongoDB

- Base = Base
- Table = Collection
- Enregistrement (tuple) = Document
- En BDR, tous les tuples d'une table ont les mêmes champs (mais les valeurs peuvent être différentes (les valeurs sont affectées à des colonnes)
- Dans une collection **MongoDB**, les documents peuvent ne pas avoir un champ partagé (pas de colonnes dans un document **MongoDB**).

MongoDB

Pour utiliser **MongoDB** : deux options principales

- **Installation locale** : télécharger et installer **MongoDB**
- **MongoDB Atlas (Cloud)** : opter pour une version gérée dans le cloud

MongoDB

Pour utiliser **MongoDB** : deux options principales

- **Installation locale** : télécharger et installer **MongoDB**
- **MongoDB Atlas (Cloud)** : opter pour une version gérée dans le cloud

ensuite

Télécharger, installer et utiliser **MongoDB Shell**

MongoDB

Première solution : installation locale

- Allez sur le lien `https://www.mongodb.com/try/download/community`, cliquez sur `Select Package`, choisissez
  - la version courante (`7.0`),
  - `msi` comme `Package` et
  - **Windows** comme `Plarform`
  - puis cliquez sur `Download` ↓
- Installez le fichier téléchargé.

MongoDB

Deuxième solution : cloud

- Allez à `https://www.mongodb.com/atlas/database`
- Cliquez sur `Try Free`
- créez un compte
- Créez un cluster (shared cluster)

MongoDB

MongoDB Shell

- Téléchargez **MongoDB Shell** : (`https://www.mongodb.com/try/download/shell`)
- Démarrez l'installation depuis le fichier téléchargé

MongoDB

MongoDB Shell

- Téléchargez **MongoDB Shell** : (`https://www.mongodb.com/try/download/shell`)
- Démarrez l'installation depuis le fichier téléchargé

Depuis un terminal, exécutez la commande suivante pour vérifier l'installation

```
mongosh --version
```

MongoDB

Database Tools : pour exécuter les commandes d'importation et exportation

- Aller à `https://www.mongodb.com/try/download/database-tools`
- Télécharger et installer
- Définir une variable d'environnement

MongoDB

Pour établir la connexion avec une installation locale, exécutez

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

MongoDB

Pour établir la connexion avec une installation locale, exécutez

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

Ou

```
mongosh "localhost:27017"
```

MongoDB

Pour établir la connexion avec une installation locale, exécutez

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

Ou

```
mongosh "localhost:27017"
```

Ou

```
mongosh
```

MongoDB

Pour utiliser un numéro de port différent, exécutez

```
mongosh --port 28015
```

MongoDB

Pour établir la connexion avec le cloud, exécutez

```
mongosh "mongodb+srv://cluster0.x2m2jma.mongodb.net/" --username
ton_nom
```

MongoDB

Pour établir la connexion avec le cloud, exécutez

```
mongosh "mongodb+srv://cluster0.x2m2jma.mongodb.net/" --username
ton_nom
```

Explication

- `mongodb+srv` : URL de connexion à un cluster **MongoDB Atlas**
- `cluster0.x2m2jma.mongodb.net` : adresse du cluster

MongoDB

Pour établir la connexion avec le cloud, exécutez

```
mongosh "mongodb+srv://cluster0.x2m2jma.mongodb.net/" --username
ton_nom
```

Explication

- `mongodb+srv` : URL de connexion à un cluster **MongoDB Atlas**
- `cluster0.x2m2jma.mongodb.net` : adresse du cluster

Pour spécifier la version de l'API, exécutez

```
mongosh "mongodb+srv://cluster0.x2m2jma.mongodb.net/" --apiVersion 1
--username ton_nom
```

MongoDB

En se connectant la chaîne suivante s'affiche

```
Atlas atlas-cwaiqg-shard-0 [primary] test>
```

MongoDB

En se connectant la chaîne suivante s'affiche

```
Atlas atlas-cwaiqg-shard-0 [primary] test>
```


Explication

- `Atlas` : service cloud permettant de déployer et gérer des clusters **MongoDB**.
- `atlas-cwaiqg-shard-0` : nom du cluster **MongoDB** auquel on est connecté. Chaque cluster **Atlas** a un nom unique.
- `primary` : nœud primaire (primary) du cluster. Dans un cluster MongoDB avec réplication, il y a généralement un nœud primaire pour les opérations d'écriture et plusieurs nœuds secondaires (secondaries) servant de copies de lecture.
- `test` : nom de la base de données.

MongoDB

MongoDB utilise les nœuds secondaires dans les cas suivants

- **Lecture de données** : Pour répartir la charge de lecture sur plusieurs nœuds, améliorant ainsi les performances globales du système.
- **Tolérance aux pannes** : En cas de panne du œud primaire, un œud secondaire peut être automatiquement promu en tant que nouveau œud primaire. Cela garantit que le cluster continue de fonctionner et d'accepter des opérations d'écriture même en cas de panne d'un œud.
- **Répartition de charge** : En utilisant un répartiteur de charge (**load balancer**), vous pouvez distribuer le trafic entre les œuds secondaires et le nœud primaire, équilibrant ainsi la charge sur l'ensemble du cluster.
- ...

MongoDB

MongoDB utilise les nœuds secondaires dans les cas suivants

- **Lecture de données** : Pour répartir la charge de lecture sur plusieurs nœuds, améliorant ainsi les performances globales du système.
- **Tolérance aux pannes** : En cas de panne du œud primaire, un œud secondaire peut être automatiquement promu en tant que nouveau œud primaire. Cela garantit que le cluster continue de fonctionner et d'accepter des opérations d'écriture même en cas de panne d'un œud.
- **Répartition de charge** : En utilisant un répartiteur de charge (**load balancer**), vous pouvez distribuer le trafic entre les œuds secondaires et le nœud primaire, équilibrant ainsi la charge sur l'ensemble du cluster.
- ...

Remarque

Les applications peuvent être configurées pour lire depuis le nœud primaire ou un nœud secondaire en fonction de leurs besoins.

MongoDB

Lister les commandes possibles sur les bases de données

```
db.help()
```

MongoDB

Du **JSON** au **BSON**

- **JSON** : **B**inary **JSON**
- **MongoDB** enregistre toutes nos données, saisies sous format **JSON**, dans le format binaire **BSON**.

MongoDB

Du **JSON** au **BSON**

- **JSON** : **B**inary **JSON**
- **MongoDB** enregistre toutes nos données, saisies sous format **JSON**, dans le format binaire **BSON**.

Exemple en JSON

```
{
   "name": "John",
   "age": 30
}
```

Même exemple en BSON

```
\x16\x00\x00\x00
\x02name\x00John\x00
\x10age\x002A\x00
\x00
```

MongoDB

Remarques

- **BSON** est plus efficace pour le stockage et la récupération des données que **JSON** en raison de sa représentation binaire, qui réduit la surcharge due à la conversion des types de données en texte.
- **BSON** est généralement plus compact que JSON, ce qui peut réduire les besoins en termes d'espace de stockage et améliorer les performances de lecture/écriture dans le cas de MongoDB.
- **BSON** prend en charge un ensemble plus large de types de données, y compris des types spécifiques à **MongoDB**, tels que `ObjectId` (utilisé comme identifiant unique), `Binary` (pour les données binaires), `Decimal128` (pour les décimaux)...

MongoDB

MongoDB : types de données

- Number
- Boolean
- String
- Array
- Object
- Null

MongoDB

Création d'une base de données : deux solutions

- Préciser le nom de la base à la connexion et la base sera créée
- Utiliser la commende `use`

MongoDB

Création d'une base de données : deux solutions

- Préciser le nom de la base à la connexion et la base sera créée
- Utiliser la commende `use`

Exemple avec une base locale

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

MongoDB

Création d'une base de données : deux solutions

- Préciser le nom de la base à la connexion et la base sera créée
- Utiliser la commende `use`

Exemple avec une base locale

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

Exemple avec Atlas

```
mongosh "mongodb+srv://cluster0.x2m2jma.mongodb.net/nom_base"
--apiVersion 1 --username ton_nom
```

MongoDB

Ou après connexion en exécutant la commande suivante

```
use nom_base
```

MongoDB

Ou après connexion en exécutant la commande suivante

```
use nom_base
```

Remarque

Si la base de données n'existe pas, elle sera créée.

MongoDB

Pour afficher le nom de la base courante, exécutez

```
db
```

MongoDB

Pour afficher le nom de la base courante, exécutez

```
db
```

Pour lister les bases de données existantes

```
show databases
```

MongoDB

Pour afficher le nom de la base courante, exécutez

```
db
```

Pour lister les bases de données existantes

```
show databases
```

Ou

```
show dbs
```

MongoDB

Par défaut

- il existe deux bases de données `admin` et `local`
- si on ne se connecte pas à une base de données, on utilise par défaut une base de données appelée `test`

MongoDB

Pour supprimer une base de données courante

```
db.runCommand({ dropDatabase: 1 })
```

MongoDB

Pour supprimer une base de données courante

```
db.runCommand({ dropDatabase: 1 })
```

Ou le raccourci

```
db.dropDatabase()
```

MongoDB

Rappel

- Une collection : l'équivalent d'une table en **SQL**.
- Un document : l'équivalent d'un tuple en **SQL**.

MongoDB

Rappel

- Une collection : l'équivalent d'une table en **SQL**.
- Un document : l'équivalent d'un tuple en **SQL**.

Création d'une collection : deux solutions

- Directement en appelant la méthode `createCollection()`, ou
- Indirectement en demandant l'insertion d'un document dans une collection inexistante

MongoDB

Première solution

```
db.createCollection("adresse")
```

MongoDB

Première solution

```
db.createCollection("adresse")
```

Deuxième solution

```
db.personnes.insert({nom: "Wick", prenom: "john"})
```

MongoDB

Pour supprimer une collection

```
db.nomCollection.drop()
```

MongoDB

Pour supprimer une collection

```
db.nomCollection.drop()
```

Ou

```
db.nomCollection.remove()
```

MongoDB

Lister les collections existantes

- `show collections`, ou
- `show tables`, ou
- `db.getCollectionNames()`

MongoDB

Pour renommer une collection

```
db.ancienne_collection.renameCollection("nouvelle_collection")
```

MongoDB

Remarque

Chaque document possède un `_id` attribué par l'utilisateur ou par **MongoDB** (`ObjectId`). Le champ `_id` constitue l'index de la collection.

MongoDB

Remarque

Chaque document possède un `_id` attribué par l'utilisateur ou par **MongoDB** (`ObjectId`). Le champ `_id` constitue l'index de la collection.

`ObjectId` est composé de 12 octets :

- 4 octets pour le timestamp
- 5 octets pour une valeur aléatoire unique par machine
- 3 octets pour un compteur auto-incrémental

MongoDB

`ObjectId` : avantages

- **Garantie d'unicité** : Chaque ObjectId généré est unique dans le monde entier.
- **Facilité d'utilisation** : Les ObjectIds sont générés automatiquement par **MongoDB**, ce qui simplifie la création de nouveaux documents sans avoir à gérer manuellement des identifiants uniques. Cela réduit la charge de travail du développeur.
- **Précision temporelle** : Les premiers 4 octets d'un `ObjectId` contiennent un timestamp qui indique la date et l'heure de création du document. Cela peut être utile pour suivre l'ordre d'insertion des documents.

MongoDB

`ObjectId` : avantages

- **Garantie d'unicité** : Chaque ObjectId généré est unique dans le monde entier.
- **Facilité d'utilisation** : Les ObjectIds sont générés automatiquement par **MongoDB**, ce qui simplifie la création de nouveaux documents sans avoir à gérer manuellement des identifiants uniques. Cela réduit la charge de travail du développeur.
- **Précision temporelle** : Les premiers 4 octets d'un `ObjectId` contiennent un timestamp qui indique la date et l'heure de création du document. Cela peut être utile pour suivre l'ordre d'insertion des documents.

`ObjectId` : inconvénients

- **Taille de stockage** (12 bytes) : Ils peuvent augmenter la taille des index et donc la consommation de stockage.
- **Exposition d'informations** : Ils contiennent des informations l'identifiant de la machine ce qui peut potentiellement exposer des informations sensibles sur le système.
- **Lecture humaine** : Ils sont conçus pour être uniques et non pour être lisibles par l'homme.

MongoDB

Ajout d'un document : plusieurs syntaxes

- `db.nomCollection.insert(...)` [Dépréciée]
- `db.nomCollection.insertOne(...)`
- `db.nomCollection.insertMany(...)`

Exemple avec `insert` [Dépréciée]

```
db.personnes.insert(
        {
            nom: 'wick',
            prenom: 'john',
            age: 45,
            sportif: true
        }
)
```

Exemple avec `insert` [Dépréciée]

```
db.personnes.insert(
        {
            nom: 'wick',
            prenom: 'john',
            age: 45,
            sportif: true
        }
)
```

Résultat

```
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("653a00a92fd9276eac1b4521") }
}
```

Exemple avec `insert` [Dépréciée]

```
db.personnes.insert(
        {
            nom: 'wick',
            prenom: 'john',
            age: 45,
            sportif: true
        }
)
```

Résultat

```
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("653a00a92fd9276eac1b4521") }
}
```


Remarque

Lors de la saisie dans le **shell**, après avoir ouvert un objet avec des accolades { , vous pouvez appuyer sur `Entrée` pour démarrer une nouvelle ligne dans l'éditeur sans exécuter la commande. La commande s'exécutera lorsque vous appuierez sur `Entrée` après avoir fermé les accolades.

MongoDB

Pour insérer un nouveau document, on peut utiliser la méthode `insertOne`

```
db.personnes.insertOne(
        {
            nom: 'dalton',
            prenom: 'jack',
            niveau: 'master',
        }
)
```

MongoDB

Pour insérer un nouveau document, on peut utiliser la méthode `insertOne`

```
db.personnes.insertOne(
        {
            nom: 'dalton',
            prenom: 'jack',
            niveau: 'master',
        }
)
```


Résultat

```
{
  acknowledged: true,
  insertedId: ObjectId("653a05402fd9276eac1b4522")
}
```

MongoDB

Pour insérer un document avec un identifiant personnalisé

```
db.personnes.insertOne(
        {
            _id: 1,
            nom: 'maggio',
            prenom: 'elena',
            niveau: 'master',
        }
)
```

MongoDB

Pour insérer un document avec un identifiant personnalisé

```
db.personnes.insertOne(
        {
            _id: 1,
            nom: 'maggio',
            prenom: 'elena',
            niveau: 'master',
        }
)
```

Résultat

```
{ acknowledged: true, insertedId: 1 }
```

MongoDB

Pour insérer plusieurs documents, on utilise `insertMany`

```
db.personnes.insertMany([
        {
            _id: 2,
            nom: 'baggio',
            prenom: 'roberto',
            salaire: 1700,
        },
    {
            _id: 3,
            nom: 'dupont',
            prenom: 'sophie',
            bourse: 500,
        }
])
```

MongoDB

Pour insérer plusieurs documents, on utilise `insertMany`

```
db.personnes.insertMany([
        {
            _id: 2,
            nom: 'baggio',
            prenom: 'roberto',
            salaire: 1700,
        },
    {
            _id: 3,
            nom: 'dupont',
            prenom: 'sophie',
            bourse: 500,
        }
])
```

Résultat

```
{ acknowledged: true, insertedIds: { '0': 2, '1': 3 } }
```

MongoDB

Remarque

`insertOne` et `insertMany` génèrent une erreur si l'`_id` spécifié existe dans la collection.

MongoDB

Pour vérifier que l'ajout a eu lieu

```
db.nomCollection.find()
```

MongoDB

Pour modifier un document, on peut utiliser

- `update()` : pour modifier un ou plusieurs documents selon une ou plusieurs conditions [Dépréciée]
- `updateOne()` : pour modifier uniquement le premier document de la sélection.
- `updateMany()` : pour modifier plusieurs documents.
- `replaceOne()` : pour remplacer le premier document de la sélection.
- `findOneAndUpdate()` : permet de modifier le premier document qui correspond au filtre spécifié et retourner le document originel
- `findOneAndReplace()` : permet de remplacer le premier document qui correspond au filtre spécifié et retourner le document originel

MongoDB

Remarque

`updateOne` et `updateMany()` acceptent au moins deux paramètres :

- le·s élément·s concerné·s par la modification
- les modifications
- quelques options

MongoDB

Exemple

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $set: {
                prenom: 'bill',
            }
        }
)
```

MongoDB

Exemple

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $set: {
                prenom: 'bill',
            }
        }
)
```


Remarque

L'opérateur `$set` permet de spécifier les champs à modifier ainsi que les nouvelles valeurs.

MongoDB

Quelques opérateurs atomiques

- `$set` : pour modifier la valeur d'un champ
- `$unset` : pour supprimer un champ
- `$inc` : pour incrémenter la valeur d'un champ numérique
- `$mul` : pour multiplier l'ancienne valeur d'un champ numérique par la valeur spécifiée
- `$rename` : pour renommer un champ
- `$min` : pour modifier la valeur d'un champ numérique spécifié avec la valeur fournie si cette dernière est inférieure à la valeur actuelle (et inversement pour `$max`)
- ...

MongoDB

L'opérateur `$set` permet d'ajouter la propriété si elle n'existe pas

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $set: {
                genre: 'homme',
            }
        }
)
```

MongoDB

L'opérateur `$set` permet d'ajouter la propriété si elle n'existe pas

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $set: {
                genre: 'homme',
            }
        }
)
```


Remarque

Le document ayant l'identifiant `20` aura une nouvelle clé `genre` avec la valeur `homme`.

MongoDB

Exemple avec `$rename` pour renommer une clé

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $rename: {
                genre: 'sexe',
            }
        }
)
```

MongoDB

Exemple avec `$rename` pour renommer une clé

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $rename: {
                genre: 'sexe',
            }
        }
)
```


Remarque

La clé `genre` devient `sexe` pour le document ayant l'identifiant `2`.

MongoDB

Exemple avec `$unset` pour supprimer une clé

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $unset: {
                sexe: 'homme'
            }
        }
)
```

MongoDB

Exemple avec `$unset` pour supprimer une clé

```
db.personnes.updateOne(
        {
            _id: 2
        },
        {
            $unset: {
                sexe: 'homme'
            }
        }
)
```


Remarque

Le champ `sexe` n'existe plus pour le document ayant l'identifiant `2`.

MongoDB

Exemple avec `$inc` pour incrémenter l'age de 5

```
db.personnes.updateOne(
        {
            nom: 'wick'
        },
        {
            $inc: {
                age: 5
            }
        }
)
```

MongoDB

La méthode `updateOne` ne modifie que le premier document qui remplit la condition

```
db.personnes.updateOne(
        {
            niveau: 'master'
        },
        {
            $set: {
                nom: 'abruzzi'
            }
        }
)
```

MongoDB

Pour modifier tous les documents qui remplissent la condition, on utilise `updateMany`

```
db.personnes.updateMany(
        {
            niveau: 'master'
        },
        {
            $set: {
                nom: 'wayne'
            }
        }
)
```

MongoDB

Si aucun document ne remplit la condition de `updateOne` (ou `updateMany`), aucun changement ne sera effectué

```
db.personnes.updateOne(
        {
            prenom: 'steven'
        },
        {
            $set: {
                nom: 'segal'
            }
        },
)
```

On peut effectuer une insertion avec `updateOne` si aucun document ne remplit la condition en ajoutant `upsert: true`

```
db.personnes.updateOne(
        {
            prenom: 'steven'
        },
        {
            $set: {
                nom: 'segal'
            }
        },
        {
            upsert: true
        }
)
```

On peut effectuer une insertion avec `updateOne` si aucun document ne remplit la condition en ajoutant `upsert: true`

```
db.personnes.updateOne(
        {
            prenom: 'steven'
        },
        {
            $set: {
                nom: 'segal'
            }
        },
        {
            upsert: true
        }
)
```

Remarque

Un nouveau document sera inséré avec le `prenom: steven` et le `nom: segal`.

MongoDB

Il est possible d'appliquer plusieurs modifications à un document

```
db.personnes.updateOne(
    {
        prenom: 'steven'
    },
    {
        $set: { "nom": "Gerard" },
        $unset: { "genre": 1 }
    }
)
```

MongoDB

Pour des modifications plus complexes et dynamiques, comme référencer les valeurs actuelles des champs dans les documents, on doit utiliser le pipeline d'agrégation (et remplacer {} par [])

```
db.personnes.updateOne(
    {
        prenom: 'steven'
    },
    [
        {
            $set: {
                "nomComplet": {
                    $concat: ["$nom", " ", "$prenom"]
                }
            }
        },
        {
            $unset: [ "nom", "prenom"]
        }
    ]
)
```

MongoDB

Pour remplacer un document par un autre, on utilise `replaceOne`

```
db.personnes.replaceOne(
    { nom: "segal" },
    { nom: "seagal", age: 66, ville: "Los Angelos" }
)
```

MongoDB

Pour remplacer un document par un autre, on utilise `replaceOne`

```
db.personnes.replaceOne(
    { nom: "segal" },
    { nom: "seagal", age: 66, ville: "Los Angelos" }
)
```


Remarque

Le premier ayant la valeur `segal` comme nom

- aura `seagal` comme `nom`
- n'aura plus de champ `prenom`
- aura un nouveau champ `age` avec une valeur `66`
- aura un nouveau champ `ville` avec une valeur `Los Angelos`

MongoDB

Pour supprimer un document, on peut utiliser

- `remove()` [Dépréciée depuis **MongoDB 3.2**]
- `deleteOne()` : permet de supprimer le premier document qui correspond au filtre spécifié
- `deleteMany()` : permet de supprimer tous les documents qui correspondent au filtre spécifié
- `findOneAndDelete()` : permet de supprimer et retourner le premier document qui correspond au filtre spécifié

MongoDB

Pour supprimer un document

```
db.personnes.deleteOne({ prenom: "john" })
```

MongoDB

Pour supprimer un document

```
db.personnes.deleteOne({ prenom: "john" })
```

Résultat

```
{ acknowledged: true, deletedCount: 1 }
```

MongoDB

Pour supprimer plusieurs documents

```
db.personnes.deleteMany({ niveau: "master" })
```

MongoDB

Pour supprimer plusieurs documents

```
db.personnes.deleteMany({ niveau: "master" })
```

Résultat

```
{ acknowledged: true, deletedCount: 2 }
```

MongoDB

Pour supprimer et récupérer le document supprimé

```
const doc = db.personnes.findOneAndDelete({ _id: 3 })
```

MongoDB

Pour supprimer et récupérer le document supprimé

```
const doc = db.personnes.findOneAndDelete({ _id: 3 })
```

Pour afficher le document supprimé

```
doc
```

MongoDB

Pour supprimer et récupérer le document supprimé

```
const doc = db.personnes.findOneAndDelete({ _id: 3 })
```

Pour afficher le document supprimé

```
doc
```

Résultat

```
{ _id: 3, nom: 'dupont', prenom: 'sophie', bourse: 500 }
```

MongoDB

Pour réaliser simultanément plusieurs opérations d'écriture, on utilise `bulkWrite`

```
const bulkOps = [
  { insertOne: { document: { nom: "Alice", age: 25 } } },
  { updateOne: { filter: { nom: "Hood" }, update: { $set: { age: 30 } } } },
  { deleteMany: { filter: { nom: "Dalton" } } }
];

db.personnes.bulkWrite(bulkOps);
```

MongoDB

Pour réaliser simultanément plusieurs opérations d'écriture, on utilise `bulkWrite`

```
const bulkOps = [
  { insertOne: { document: { nom: "Alice", age: 25 } } },
  { updateOne: { filter: { nom: "Hood" }, update: { $set: { age: 30 } } } },
  { deleteMany: { filter: { nom: "Dalton" } } }
];

db.personnes.bulkWrite(bulkOps);
```


Suite à cette requête

- un nouveau document sera ajouté
- le premier document ayant un champ `nom` avec la valeur `Hood` aura un nouveau champ `age` avec la valeur `30`
- tous les documents ayant la valeur `Dalton` pour le champ `nom` seront supprimés

MongoDB

`bulkWrite` : avantages

- **Performances améliorées** : Des meilleures performances en réduisant la latence réseau et en optimisant l'utilisation des ressources du serveur.
- **Atomicité** : Toutes les opérations réussissent ou échouent en bloc. Cela garantit la cohérence des données, évitant des états intermédiaires incohérents.
- ...

MongoDB

`bulkWrite` : avantages

- **Performances améliorées** : Des meilleures performances en réduisant la latence réseau et en optimisant l'utilisation des ressources du serveur.
- **Atomicité** : Toutes les opérations réussissent ou échouent en bloc. Cela garantit la cohérence des données, évitant des états intermédiaires incohérents.
- ...

`bulkWrite` : inconvénients

- **Complexité** : code plus complexe, en particulier lorsque si nous devons gérer plusieurs types d'opérations (insertion, mise à jour, suppression) dans un seul appel.
- **Gestion des erreurs** : une erreur dans l'une des opérations peut entraîner l'échec de l'ensemble de l'opération.
- **Moins adapté aux opérations isolées** : Si nous n'avions qu'une seule opération à effectuer, l'utilisation de `bulkWrite` peut sembler excessive et introduire une surcharge inutile.

MongoDB

Récupérer tous les documents d'une collection

```
db.personnes.find()
```

MongoDB

Récupérer tous les documents d'une collection

```
db.personnes.find()
```

Afficher seulement le premier document

```
db.personnes.findOne()
```

MongoDB

Rechercher selon des critères

```
db.personnes.find({ nom: "bob" ... })
```

MongoDB

Rechercher selon des critères

```
db.personnes.find({ nom: "bob" ... })
```

Compter le nombre de documents sélectionnés

```
db.personnes.find({ nom: "bob" ... }).count()
```

MongoDB

Trier le résultat de recherche selon le `nom` dans l'ordre croissant

```
db.personnes.find().sort({ nom: 1 })
```

MongoDB

Trier le résultat de recherche selon le `nom` dans l'ordre croissant

```
db.personnes.find().sort({ nom: 1 })
```

Trier le résultat de recherche dans l'ordre décroissant

```
db.personnes.find().sort({ nom: -1 })
```

MongoDB

Sauter quelques documents (ne pas les afficher)

```
db.personnes.find().skip(2)
```

MongoDB

Sauter quelques documents (ne pas les afficher)

```
db.personnes.find().skip(2)
```

Limiter le nombre de documents à afficher

```
db.personnes.find().limit(2)
```

MongoDB

Sauter quelques documents (ne pas les afficher)

```
db.personnes.find().skip(2)
```

Limiter le nombre de documents à afficher

```
db.personnes.find().limit(2)
```

Trier et limiter le nombre de documents à afficher

```
db.personnes.find().sort({ nom: -1 }).limit(2)
```

MongoDB

Et si on voulait sélectionner les champs à afficher

```
db.personnes.find({}, { nom: 1 })
```

MongoDB

Et si on voulait sélectionner les champs à afficher

```
db.personnes.find({}, { nom: 1 })
```

Explication

La requête précédente affiche le nom de toutes les personnes (l'identifiant sera automatiquement affiché)

MongoDB

Et si on voulait sélectionner les champs à afficher

```
db.personnes.find({}, { nom: 1 })
```

Explication

La requête précédente affiche le nom de toutes les personnes (l'identifiant sera automatiquement affiché)

Et si on ne veut pas afficher les `_id`

```
db.personnes.find({}, { nom: 1, _id: 0 })
```

MongoDB

Explication

- La valeur 1 permet d'inclure le champ dans la sélection.
- La valeur 0 permet d'exclure le champ de la sélection.

MongoDB

Explication

- La valeur 1 permet d'inclure le champ dans la sélection.
- La valeur 0 permet d'exclure le champ de la sélection.

Remarque

Inclure (avec 1) un champ et exclure (avec 0) un autre (dans une même requête) génère une erreur (sauf pour `_id`).

MongoDB

Appeler une fonction pour chaque document de la sélection

```
db.personnes.find().forEach(
        function(perso) {
                print(perso.nom + " " + perso.prenom);
        }
);
```

MongoDB

Appeler une fonction pour chaque document de la sélection

```
db.personnes.find().forEach(
        function(perso) {
                print(perso.nom + " " + perso.prenom);
        }
);
```

Ou une fonction fléchée

```
db.personnes.find().forEach((perso) => {
    print(perso.nom + " " + perso.prenom);
});
```

MongoDB

On peut aussi utiliser une expression régulière pour chercher les personnes dont le nom commence par w

```
db.personnes.find({name: /^w/})
```

MongoDB

On peut aussi utiliser une expression régulière pour chercher les personnes dont le nom commence par w

```
db.personnes.find({name: /^w/})
```

Explication

- les deux / pour indiquer le début et la fin de l'expression régulière
- $\wedge$ pour indiquer par quoi commence le mot cherché

MongoDB

Chercher les personnes dont le nom se termine par `k`

```
db.personnes.find({ nom: /k$/ })
```

MongoDB

Chercher les personnes dont le nom se termine par `k`

```
db.personnes.find({ nom: /k$/ })
```

Chercher les personnes dont le nom commence par `e` ou par `h`

```
db.personnes.find({ nom: /^[eh]/ })
```

MongoDB

Chercher les personnes dont le nom se termine par `k`

```
db.personnes.find({ nom: /k$/ })
```

Chercher les personnes dont le nom commence par `e` ou par `h`

```
db.personnes.find({ nom: /^[eh]/ })
```

Chercher les personnes dont le nom commence par une lettre comprise entre e et w :

```
db.personnes.find({ nom: /^[e-w]/ })
```

MongoDB

Autres symboles utilisés en ER

- `x?` : pour indiquer que la lettre x est facultative. Elle peut y être 0 ou 1 fois.
- `x+` : pour indiquer que la lettre x est obligatoire. Elle peut y être 1 ou plusieurs fois.
- `x*` : pour indiquer que la lettre x est facultative. Elle peut y être 0, 1 ou plusieurs fois.
- `x{2,4}` : pour indiquer que la lettre x doit se répéter au moins deux fois et au plus 4 fois.
- `.` : un caractère quelconque
- | : le ou logique

MongoDB

On peut aussi utiliser `$regex`

```
db.personnes.find({ prenom: { $regex: /john/ }})
```

MongoDB

On peut aussi utiliser `$regex`

```
db.personnes.find({ prenom: { $regex: /john/ }})
```

On peut aussi désactiver la sensibilité à la casse avec `$options`

```
db.personnes.find({ prenom: { $regex: /john/, $options: "i" }})
```

MongoDB

Quelques opérations

- `$push` : pour ajouter un élément au tableau
- `$pop` : pour supprimer le premier ou le dernier élément d'un tableau
- `$pull` : pour supprimer une ou plusieurs valeurs d'un tableau
- `$pullAll` : pour supprimer tous les éléments d'un tableau
- `$position` : à utiliser avec `push` pour indiquer la position d'insertion dans un tableau
- `$slice` : à utiliser avec `push` pour préciser les éléments à garder dans un tableau
- `$sort` : à utiliser avec `push` pour ordonner les éléments d'un tableau
- ...

MongoDB

Considérons le document suivant

```
db.personnes.insertOne({
    _id: 5,
    nom: 'wick',
    sports: [ 'foot', 'hand', 'tennis']
})
```

MongoDB

Pour ajouter un nouveau sport au tableau

```
db.personnes.updateOne(
    {
        _id: 5
    },
    {
        $push: {
            "sports": "basket"
        }
     }
)
```

MongoDB

Peut-on ajouter plusieurs sports avec une seule requête ?

```
db.personnes.updateOne(
    {
        _id: 5
    },
    {
        $push: {
            sports: ['hockey', 'sky']
        }
    }
)
```

MongoDB

Peut-on ajouter plusieurs sports avec une seule requête ?

```
db.personnes.updateOne(
    {
        _id: 5
    },
    {
        $push: {
            sports: ['hockey', 'sky']
        }
    }
)
```

Non, ça rajoute un tableau dans un tableau

MongoDB

Ou comme-ça ?

```
db.personnes.updateOne(
        { _id: 5 },
        {
            $push: {
                sports: { 'hockey', 'sky' }
                }
        }
)
```

MongoDB

Ou comme-ça ?

```
db.personnes.updateOne(
        { _id: 5 },
        {
            $push: {
                sports: { 'hockey', 'sky' }
                }
        }
)
```

Non, ça génère une erreur

MongoDB

Solution

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $push: {
            sports: {
                $each: ['basket', 'sky']
            }
        }
    }
)
```

MongoDB

Solution

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $push: {
            sports: {
                $each: ['basket', 'sky']
            }
        }
    }
)
```


Remarque

`$push` : ajoute naturellement l'élément après le dernier élément du tableau

Pour ajouter un élément à une position précise

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $push: {
            sports: {
                $each: ['volley'],
                $position: 2
            }
        }
    }
)
```

Pour ajouter un élément à une position précise

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $push: {
            sports: {
                $each: ['volley'],
                $position: 2
            }
        }
    }
)
```


Explication

- Ceci rajoute l'élément `volley` à la position 2 du tableau `sport` (la première position est d'indice 0)
- Les autres éléments seront décalés

MongoDB

Pour supprimer le premier élément d'un tableau

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $pop: {
            sports: -1
        }
    }
)
```

MongoDB

Pour supprimer le premier élément d'un tableau

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $pop: {
            sports: -1
        }
    }
)
```

Pour supprimer le dernier élément d'un tableau

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $pop: {
            sports: 1
        }
    }
)
```

MongoDB

Pour supprimer un élément quelconque d'un tableau

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $pull: {
            sports: "foot"
        }
    }
)
```

MongoDB

Pour supprimer un élément quelconque d'un tableau

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $pull: {
            sports: "foot"
        }
    }
)
```


Explication

L'élément `foot` sera supprimé du tableau `sports` quelle que soit sa position.

MongoDB

Pour supprimer plusieurs éléments avec une seule requête

```
db.personnes.updateOne(
    { _id: 5 },
    {
        $pull: {
            sports: {
                $in: ['hockey', 'basket']
            }
        }
    }
)
```

MongoDB

Considérons le document suivant

```
db.personnes.insertOne({
        _id : 6,
        nom : 'wick',
        sports : [ 'foot', 'hand', 'tennis', 'hockey', 'sky', 'volley']
})
```

MongoDB

Pour sélectionner les 4 premiers sports de la personne ayant l'identifiant 6

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: 4
        }
    }
)
```

MongoDB

Pour sélectionner les 4 premiers sports de la personne ayant l'identifiant 6

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: 4
        }
    }
)
```


Résultat

```
{ _id: 6, nom: 'wick', sports: [ 'foot', 'hand', 'tennis', 'hockey' ] }
```

MongoDB

Pour sélectionner les 4 derniers sports de la personne ayant l'identifiant 6

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: -4
        }
    }
)
```

MongoDB

Pour sélectionner les 4 derniers sports de la personne ayant l'identifiant 6

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: -4
        }
    }
)
```


Résultat

```
{
  _id: 6,
  nom: 'wick',
  sports: [ 'tennis', 'hockey', 'sky', 'volley' ]
}
```

MongoDB

Pour sélectionner les 4 derniers sports de la personne ayant l'identifiant 6

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: -4
        }
    }
)
```

MongoDB

Pour sélectionner les 4 derniers sports de la personne ayant l'identifiant 6

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: -4
        }
    }
)
```


Résultat

```
{
  _id: 6,
  nom: 'wick',
  sports: [ 'tennis', 'hockey', 'sky', 'volley' ]
}
```

MongoDB

L'opérateur `$slice` peut accepter deux paramètres : le premier étant la position de départ et le second étant le nombre d'éléments

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: [2, 3]
        }
    }
)
```

MongoDB

L'opérateur `$slice` peut accepter deux paramètres : le premier étant la position de départ et le second étant le nombre d'éléments

```
db.personnes.findOne(
    { _id: 6 },
    {
        sports: {
            $slice: [2, 3]
        }
    }
)
```


Résultat

```
{
  _id: 6,
  nom: 'wick',
  sports: [ 'tennis', 'hockey', 'sky' ]
}
```

MongoDB

Pour trier les éléments d'un tableau

```
db.personnes.updateOne(
    { _id: 6 },
    {
        $push: {
            sports: {
                $each: [],
                $sort: 1
            }
        }
    }
)
```

MongoDB

Pour trier les éléments d'un tableau

```
db.personnes.updateOne(
    { _id: 6 },
    {
        $push: {
            sports: {
                $each: [],
                $sort: 1
            }
        }
    }
)
```

Pour trier dans l'ordre décroissant

```
$sort: -1
```

MongoDB

Commençons par créer la nouvelle collection suivante

```
db.etudiants.insertMany([
    {_id: 1, nom: 'wick', notes: [10, 15, 12], age: 19 },
    {_id: 2, nom: 'bob', notes: [18, 8, 12], age: 35 },
    {_id: 3, nom: 'wolf', notes: [7, 6, 13], age: 25 },
    {_id: 4, nom: 'green', notes: [18, 16, 9], age: 22 }
])
```

MongoDB

Commençons par créer la nouvelle collection suivante

```
db.etudiants.insertMany([
    {_id: 1, nom: 'wick', notes: [10, 15, 12], age: 19 },
    {_id: 2, nom: 'bob', notes: [18, 8, 12], age: 35 },
    {_id: 3, nom: 'wolf', notes: [7, 6, 13], age: 25 },
    {_id: 4, nom: 'green', notes: [18, 16, 9], age: 22 }
])
```


Résultat

```
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4 } }
```

MongoDB

Pour incrémenter toutes les notes du premier étudiant

```
db.etudiants.updateOne(
      { },
      { $inc: { "notes.$[]": 1 } },
)
```

MongoDB

Considérons le cas d'un tableau d'objets

```
db.personnes.insertOne({
    _id : 7,
    nom : "dalton",
    notes: [
        { programmation: 17, coefficient: 4},
        { OS: 10, coefficient: 2}
    ]
})
```

MongoDB

Pour incrémenter tous les coefficients de la première personne

```
db.personnes.updateOne(
      { _id: 7 },
      { $inc: { "notes.$[].coefficient": 1 } },
)
```

MongoDB

Pour remplacer un élément du tableau (`foot`) par un autre (`football`), on peut utiliser `$`

```
db.personnes.updateOne(
    { _id: 6, sports: "foot" },
    { $set: { "sports.$": "football" } },
)
```

MongoDB

Pour remplacer un élément du tableau (`foot`) par un autre (`football`), on peut utiliser `$`

```
db.personnes.updateOne(
    { _id: 6, sports: "foot" },
    { $set: { "sports.$": "football" } },
)
```

Explication

- L'élément du tableau à remplacer doit figurer dans la première partie (la partie filtre).
- Dans la deuxième partie, on spécifie la nouvelle valeur.

MongoDB

Remarque

Cette syntaxe ne permet de modifier que la première occurrence d'un élément dans un tableau.

MongoDB

Pour le cas d'un tableau d'objets

```
db.personnes.updateOne(
      { _id: 7 , "notes.OS": 10 },
      { $set: { "notes.$.coefficient" : 2 } }
)
```

MongoDB

Pour le cas d'un tableau d'objets

```
db.personnes.updateOne(
      { _id: 7 , "notes.OS": 10 },
      { $set: { "notes.$.coefficient" : 2 } }
)
```

Résultat du `find`

```
{
    _id: 7,
    nom: 'dalton',
    notes: [
        { programmation: 17, coefficient: 5 },
        { OS: 10, coefficient: 2 }
    ]
}
```

MongoDB

Pour remplacer un élément du tableau (`football`) par un autre (`foot`), on peut utiliser `$[element]`

```
db.personnes.updateOne(
    { _id: 6 },
    { $set: { "sports.$[element]": "foot" } },
    { arrayFilters: [ { element: "football" } ] }
)
```

MongoDB

Modifions le tableau `sports` pour la personne ayant l'identifiant 6

```
db.personnes.updateOne(
    { _id: 6 },
    { $set: { sports: ["foot", "foot", "tennis", "hand", "foot"
      ] } },
)
```

MongoDB

Modifions le tableau `sports` pour la personne ayant l'identifiant 6

```
db.personnes.updateOne(
    { _id: 6 },
    { $set: { sports: ["foot", "foot", "tennis", "hand", "foot"
      ] } },
)
```

`$[element]` **permet de remplacer toutes les occurrences de** `foot` **par** `football`

```
db.personnes.updateOne(
    { _id: 6 },
    { $set: { "sports.$[element]": "football" } },
    { arrayFilters: [ { element: "foot" } ] }
)
```

MongoDB

Pour ajouter un nouvel élément au tableau

```
db.personnes.updateOne(
    { _id: 7 },
    {
        $push: {
            notes : { compilation: 15, coefficient: 1 }
        }
    }
)
```

MongoDB

Et pour supprimer

```
db.personnes.updateOne(
    { _id: 7 },
    {
        $pull: {
            notes: { compilation: 15, coefficient: 1 }
        }
    }
)
```

MongoDB

Que fait la requête de suppression suivante ?

```
db.personnes.updateOne(
    { _id: 7 },
    {
        $pull: {
            notes: { programmation: 17 }
        }
    }
)
```

MongoDB

Que fait la requête de suppression suivante ?

```
db.personnes.updateOne(
    { _id: 7 },
    {
        $pull: {
            notes: { programmation: 17 }
        }
    }
)
```

L'objet a été supprimé même s'il n'y avait pas de correspondance complète

MongoDB

Pour éviter la suppression en cas de correspondance partielle

```
db.personnes.updateOne(
    { _id: 7 },
    {
        $pull: {
            notes: {
                $elemMatch: { OS: 10 }
            }
        }
    }
)
```

MongoDB

Pour éviter la suppression en cas de correspondance partielle

```
db.personnes.updateOne(
    { _id: 7 },
    {
        $pull: {
            notes: {
                $elemMatch: { OS: 10 }
            }
        }
    }
)
```

Le document n'a pas été supprimé car l'attribut `coefficient: 2` n'a pas été précisé

MongoDB

Pour chercher un document selon une valeur dans le tableau d'objets

```
db.personnes.find(
    {
        "notes.coefficient": 1
    }
)
```

MongoDB

Comme pour les bases de données relationnelles

- opérateurs de comparaison
- opérateurs logiques
- ...

MongoDB

Pour sélectionner les étudiants âgés de plus de 20 ans

```
db.etudiants.find(
    {
        age: {
            $gt: 20
        }
    }
)
```

MongoDB

Pour sélectionner les étudiants âgés de plus de 20 ans

```
db.etudiants.find(
    {
        age: {
            $gt: 20
        }
    }
)
```



```
[
  { _id: 2, nom: 'bob', notes: [ 18, 8, 12 ], age: 35 },
  { _id: 3, nom: 'wolf', notes: [ 7, 6, 13 ], age: 25 },
  { _id: 4, nom: 'green', notes: [ 18, 16, 9 ], age: 22 }
]
```

MongoDB

Les opérateurs de comparaison

- `$gt` : greater than (supérieur à)
- `$gte` : greater than or equal (supérieur ou égal)
- `$lt` : less than (inférieur à)
- `$lte` : less than or equal (inférieur ou égal)
- `$eq` : equal (égal à)
- `$ne` : not equal (différent de)
- `$in` : dans (un tableau...)
- `$nin` : not in (pas dans)

MongoDB

Pour sélectionner les étudiants dont l'age est entre 30 et 40 ans

```
db.etudiants.find(
    {
        $and: [
            { age: { $gte: 30 }},
            { age: { $lte: 40 }}
        ]
    }
)
```

MongoDB

Pour sélectionner les étudiants dont l'age est entre 30 et 40 ans

```
db.etudiants.find(
    {
        $and: [
            { age: { $gte: 30 }},
            { age: { $lte: 40 }}
        ]
    }
)
```

```
[ { _id: 2, nom: 'bob', notes: [ 18, 8, 12 ], age: 35 } ]
```

MongoDB

Les opérateurs logiques

- `$and` : et
- `$or` : ou
- `$not` : le non logique
- `$nor` : ou exclusif

MongoDB

Afficher les personnes ayant un champ `salaire`

```
db.personnes.find(
    {
        salaire: {
            $exists: true
        }
    }
)
```

MongoDB

Afficher les personnes dont l'age est divisible par 5 (`age % 5 = 0`)

```
db.etudiants.find(
    {
        age: {
            $mod: [ 5, 0 ]
        }
    }
)
```

MongoDB

On peut utiliser `$where` pour exécuter une fonction JavaScript (prédicat)

```
db.etudiants.find(
    {
        $where: function () { return Math.min(...this.notes) >= 10 }
    }
)
```

MongoDB

On peut utiliser `$where` pour exécuter une fonction JavaScript (prédicat)

```
db.etudiants.find(
    {
        $where: function () { return Math.min(...this.notes) >= 10 }
    }
)
```

Résultat

```
[ { _id: 1, nom: 'wick', notes: [ 10, 15, 12 ], age: 19 } ]
```

MongoDB

On peut utiliser `$where` pour exécuter une fonction JavaScript (prédicat)

```
db.etudiants.find(
    {
        $where: function () { return Math.min(...this.notes) >= 10 }
    }
)
```

Résultat

```
[ { _id: 1, nom: 'wick', notes: [ 10, 15, 12 ], age: 19 } ]
```


Remarques

- `this` permet de pointer sur les champs de chaque document
- On ne peut utiliser les fonctions fléchées pour pouvoir accéder aux champs avec `this`

MongoDB

Pour sélectionner les étudiants ayant la valeur 10 dans `notes`

```
db.etudiants.find(
    {
        notes: {
            $eq: 10
        }
    }
)
```

MongoDB

Pour sélectionner les étudiants ayant la valeur 10 dans `notes`

```
db.etudiants.find(
    {
        notes: {
            $eq: 10
        }
    }
)
```

Ou en plus simple

```
db.etudiants.find(
    {
        notes: 10
    }
)
```

MongoDB

On peut utiliser `$all` pour retourner la liste d'étudiants ayant les notes 10 et 12

```
db.etudiants.find(
    {
        notes: {
            $all: [10, 12]
        }
    }
)
```

MongoDB

On peut utiliser `$all` pour retourner la liste d'étudiants ayant les notes 10 et 12

```
db.etudiants.find(
    {
        notes: {
            $all: [10, 12]
        }
    }
)
```


Résultat

```
[ { _id: 1, nom: 'wick', notes: [ 10, 15, 12 ], age: 19 } ]
```

MongoDB

Exercice

Écrire une requête **MongoDB** qui permet de sélectionner les étudiants n'ayant que des notes supérieures ou égale à 10.

MongoDB

Solution

```
db.personnes.find({ notes: { $all: { $gt: 12 } } })
```

MongoDB

Pour réaliser un traitement conditionnel

```
db.etudiants.find({}, {
    age: 1,
    catégorie: {
        $cond: {
            if: { $gte: ["$age", 18] },
            then: "Adulte",
            else: "Mineur"
        }
    }
})
```

MongoDB

Pour réaliser un traitement conditionnel

```
db.etudiants.find({}, {
    age: 1,
    catégorie: {
        $cond: {
            if: { $gte: ["$age", 18] },
            then: "Adulte",
            else: "Mineur"
        }
    }
})
```


Résultat

```
[
  { _id: 1, age: 19, 'catégorie': 'Adulte' },
  { _id: 2, age: 35, 'catégorie': 'Adulte' },
  { _id: 3, age: 25, 'catégorie': 'Adulte' },
  { _id: 4, age: 22, 'catégorie': 'Adulte' }
]
```

MongoDB

Pour réaliser une opération arithmétique sur plusieurs champs

```
db.etudiants.find({
    $expr: {
        $gt: [
            { $avg: "$notes" },
            10
        ]
    }
})
```

MongoDB

Pour réaliser une opération arithmétique sur plusieurs champs

```
db.etudiants.find({
    $expr: {
        $gt: [
            { $avg: "$notes" },
            10
        ]
    }
})
```


Résultat

```
[
  { _id: 1, nom: 'wick', notes: [ 10, 15, 12 ], age: 19 },
  { _id: 2, nom: 'bob', notes: [ 18, 8, 12 ], age: 35 },
  { _id: 4, nom: 'green', notes: [ 18, 16, 9 ], age: 22 }
]
```

MongoDB

Index : définition

- Outil utilisé pour améliorer les performances des opérations de recherche et de requête dans une base de données **MongoDB**
- Par exemple, si on a un champ (autre que `_id`) selon lequel on effectue des recherches très fréquemment
- Pour accélérer la recherche, on peut créer un index sur ce champ
- Par défaut, on a un index sur chaque `_id` d'une collection (impossible de le supprimer)

MongoDB

Index : pourquoi

- **Accélérer les requêtes de recherche** : localisation rapide des documents qui correspondent à une requête donnée.
- **Améliorer les performances des opérations de tri**
- **Optimiser les opérations d'agrégation**
- **Limiter le nombre de documents analysés**
- ...

MongoDB

Index : inconvénients potentiels

- **Espace disque supplémentaire** : Plus on a d'index, plus la base de données peut devenir volumineuse.
- **Surcharge lors des opérations d'écriture** : Lorsque on effectue des opérations d'écritures, les index doivent être mis à jour. Cela peut ralentir les opérations d'écriture, en particulier si on a de nombreux index.
- **Besoins en ressources (mémoire et CPU)** : Avoir de nombreux index actifs $\Rightarrow$ impact sur les performances globales du serveur **MongoDB**.
- ...

MongoDB

Pour consulter la liste d'index sur `personnes`

```
db.personnes.getIndexes()
```

MongoDB

Pour consulter la liste d'index sur `personnes`

```
db.personnes.getIndexes()
```

Résultat

```
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
```

MongoDB

Pour créer un nouvel index sur le champ `nom`

```
db.personnes.createIndex({ nom: 1 })
```

MongoDB

Pour créer un nouvel index sur le champ `nom`

```
db.personnes.createIndex({ nom: 1 })
```

Pour supprimer l'index sur le champ `nom`

```
db.personnes.dropIndex({ nom: 1 })
```

MongoDB

Pour créer un nouvel index sur le champ `nom`

```
db.personnes.createIndex({ nom: 1 })
```

Pour supprimer l'index sur le champ `nom`

```
db.personnes.dropIndex({ nom: 1 })
```

Ou selon le nom de l'index

```
db.personnes.dropIndex("nom_1")
```

MongoDB

Consultons une nouvelle fois la liste d'index

```
db.personnes.getIndexes()
```

MongoDB

Consultons une nouvelle fois la liste d'index

```
db.personnes.getIndexes()
```

Résultat

```
[    executionStages: {
   { v: 2, key: { _id: 1 }, name: '_id_' },
   { v: 2, key: { nom: 1 }, name: 'nom_1' }
]      nReturned: 2,
```

MongoDB

Méthode `agregate()`

- utilisée pour effectuer des opérations d'agrégation de données sur les documents d'une collection
- permettant de transformer, regrouper, trier, filtrer et effectuer diverses autres opérations sur les données stockées dans une collection **MongoDB**

MongoDB

`$project` **permet de remodeler une collection**

```
db.etudiants.aggregate([
    {
        $project: {
            nom: 1,
            notes: 1,
            _id: 0
        }
    },
])
```

MongoDB

`$project` **permet de remodeler une collection**

```
db.etudiants.aggregate([
    {
        $project: {
            nom: 1,
            notes: 1,
            _id: 0
        }
    },
])
```


Le résultat

```
[
  { nom: 'wick', notes: [ 10, 15, 12 ] },
  { nom: 'bob', notes: [ 18, 8, 12 ] },
  { nom: 'wolf', notes: [ 7, 6, 13 ] },
  { nom: 'green', notes: [ 18, 16, 9 ] }
]
```

MongoDB

Dans `$project`, on peut renommer une colonne et utiliser une fonction d'agrégation

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    }
])
```

MongoDB

Dans `$project`, on peut renommer une colonne et utiliser une fonction d'agrégation

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    }
])
```


Le résultat

```
[
  { "_id" : "wick", "moyenne" : 12.333333333333334 }
  { "_id" : "bob", "moyenne" : 12.666666666666666 }
  { "_id" : "wolf", "moyenne" : 8.666666666666666 }
  { "_id" : "green", "moyenne" : 14.333333333333334 }
]
```

MongoDB

Explication

- `$project` : est un pipeline d'agrégation qui permet de remodeler une collection
- On aura ainsi deux nouveaux champs dans les résultats de la requête :
  - `_id` : contiendra la valeur du champ `nom` du document d'origine
  - `moyenne` : contiendra la moyenne des valeurs du champ `notes`
- L'opérateur d'agrégation `$avg` est utilisé pour calculer la moyenne.

MongoDB

Autres pipelines d'agrégation

- `$group` : Permet de regrouper des documents comme un `group by` en SQL
- `$out` : Permet de créer une nouvelle collection à partir d'une autre qui existe déjà
- `$unwind` : Permet de décomposer un tableau en autant de documents que d'élément.
- `$match` : Permet de filtrer les documents selon la condition spécifiée
- `$sample` : Permet de sélectionner aléatoirement un nombre de documents spécifiée dans la requête
- ...

MongoDB

Autres opérateurs d'agrégation

- `$max`, `$min`, `$sum`, `$sqrt`, `$pow`, `$floor`, `$divide`, `$abs` ...
- `$ifNull`
- `$map`, `$reduce`
- `$arrayToObject`, `$dateFromString`, `$dateToString`...
- `$split`, `$slice`, `$size`...
- `$substr`, `$toUpper`, `$toLower`, `$concat`, `$strLenCP`...
- ...

MongoDB

Exercice

Écrire une requête **MongoDB** qui permet d'afficher pour chaque étudiant, son nom, sa note max et sa note min.

MongoDB

Remarque

L'opérateur `$project` peut être utilisé plusieurs fois dans une même méthode `aggregate`.

MongoDB

Pour trier le résultat précédent, on peut utiliser l'opérateur `$sort`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
        $sort: { moyenne: -1}
    }
])
```

MongoDB

Pour trier le résultat précédent, on peut utiliser l'opérateur `$sort`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
        $sort: { moyenne: -1}
    }
])
```


Le résultat

```
[
  { _id: 'green', moyenne: 14.333333333333334 },
  { _id: 'bob', moyenne: 12.666666666666666 },
  { _id: 'wick', moyenne: 12.333333333333334 },
  { _id: 'wolf', moyenne: 8.666666666666666 }
]
```

MongoDB

L'opérateur `$sort` peut être remplacé par la méthode `sort`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
]).sort({ moyenne: -1 })
```

MongoDB

L'opérateur `$sort` peut être remplacé par la méthode `sort`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
]).sort({ moyenne: -1 })
```


Le résultat

```
[
  { _id: 'green', moyenne: 14.333333333333334 },
  { _id: 'bob', moyenne: 12.666666666666666 },
  { _id: 'wick', moyenne: 12.333333333333334 },
  { _id: 'wolf', moyenne: 8.666666666666666 }
]
```

MongoDB

Pour limiter le nombre de documents dans `aggregate`, on utilise `$limit`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
        $sort: { moyenne: -1}
    },
    {
        $limit: 1
    }
])
```

MongoDB

Pour limiter le nombre de documents dans `aggregate`, on utilise `$limit`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
        $sort: { moyenne: -1}
    },
    {
        $limit: 1
    }
])
```

Le résultat

```
[ { _id: 'green', moyenne: 14.333333333333334 } ]
```

Pour sauter les deux premiers documents, on utilise `$skip`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
        $sort: { moyenne: -1}
    },
    {
        $skip: 2
    }
])
```

Pour sauter les deux premiers documents, on utilise `$skip`

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
        $sort: { moyenne: -1}
    },
    {
        $skip: 2
    }
])
```

Le résultat

```
[
  { _id: 'wick', moyenne: 12.333333333333334 },
  { _id: 'wolf', moyenne: 8.666666666666666 }
]
```

MongoDB

Exercice

Écrire une requête **MongoDB** qui permet d'afficher le nom, et uniquement le nom, de l'étudiant ayant la meilleur moyenne.

MongoDB

Pour ajouter des nouveaux champs tout en gardant les autres, on utilise `$set`

```
db.etudiants.aggregate([
    {
        $set: {
            moyenne: { $avg: "$notes" },
            max: { $max: "$notes" }
        }
    }
])
```

MongoDB

Pour exclure certains champs, on utilise `$unset`

```
db.etudiants.aggregate([
    {
        $unset: ["age", "notes"]
    }
])
```

MongoDB

Pour exclure certains champs, on utilise `$unset`

```
db.etudiants.aggregate([
    {
        $unset: ["age", "notes"]
    }
])
```

Le résultat

```
[
  { _id: 1, nom: 'wick' },
  { _id: 2, nom: 'bob' },
  { _id: 3, nom: 'wolf' },
  { _id: 4, nom: 'green' }
]
```

MongoDB

Exemple avec `sample`

```
db.etudiants.aggregate(
[
        {
            $sample: { size: 3 }
        }
])
```

MongoDB

Exemple avec `sample`

```
db.etudiants.aggregate(
[
        {
            $sample: { size: 3 }
        }
])
```


Résultat

Il choisit aléatoirement trois documents de la collection

`$unwind` **permet de créer un document distinct (aplatir) pour chaque élément du tableau** `notes`.

```
db.etudiants.aggregate(
[
        { $unwind : "$notes" }
])
```

`$unwind` **permet de <u>créer un document distinct</u> (aplatir) pour chaque élément du tableau** `notes`.

```
db.etudiants.aggregate(
[
        { $unwind : "$notes" }
])
```

Résultat

```
{ "_id" : 1, "nom" : "wick", "notes" : 10, "age" : 19 }
{ "_id" : 1, "nom" : "wick", "notes" : 15, "age" : 19 }
{ "_id" : 1, "nom" : "wick", "notes" : 12, "age" : 19 }
{ "_id" : 2, "nom" : "bob", "notes" : 18, "age" : 25 }
{ "_id" : 2, "nom" : "bob", "notes" : 8, "age" : 25 }
{ "_id" : 2, "nom" : "bob", "notes" : 12, "age" : 25 }
{ "_id" : 3, "nom" : "wolf", "notes" : 7, "age" : 35 }
{ "_id" : 3, "nom" : "wolf", "notes" : 6, "age" : 35 }
{ "_id" : 3, "nom" : "wolf", "notes" : 13, "age" : 35 }
{ "_id" : 4, "nom" : "green", "notes" : 18, "age" : 22 }
{ "_id" : 4, "nom" : "green", "notes" : 16, "age" : 22 }
{ "_id" : 4, "nom" : "green", "notes" : 9, "age" : 22 }
```

MongoDB

Exercice

Écrire une requête **MongoDB** qui permet de retourner le nom de l'étudiant ayant la plus mauvaise note.

MongoDB

Utilisons `out` pour stocker le résultat dans une nouvelle collection

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
            $out : "moyennes"
    }
])
```

MongoDB

Utilisons `out` pour stocker le résultat dans une nouvelle collection

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
            $out : "moyennes"
    }
])
```

Vérifier la création de la collection moyennes

```
show collections
```

MongoDB

Utilisons `out` pour stocker le résultat dans une nouvelle collection

```
db.etudiants.aggregate([
    {
        $project: {
            _id: "$nom",
            moyenne: { $avg: "$notes" }
        }
    },
    {
            $out : "moyennes"
    }
])
```

Vérifier la création de la collection moyennes

```
show collections
```

Ou

```
db.moyennes.find()
```

MongoDB

Considérons la collection `books` suivante (Exemple de la documentation officielle

```
db.books.insertMany([
    { _id: 8751, title: "The Banquet", author: "Dante", copies: 2 },
    { _id: 8752, title: "Divine Comedy", author: "Dante", copies: 1 },
    { _id: 8645, title: "Eclogues", author: "Dante", copies: 2 },
    { _id: 7000, title: "The Odyssey", author: "Homer", copies: 10 },
    { _id: 7020, title: "Iliad", author: "Homer", copies: 10 }
])
```

MongoDB

Utilisons `group` pour regrouper selon `author`

```
db.books.aggregate(
[
      {
          $group: {
              _id : "$author",
              books: { $push: "$title" }
          }
      }
])
```

MongoDB

Utilisons `group` pour regrouper selon `author`

```
db.books.aggregate(
[
      {
          $group: {
              _id : "$author",
              books: { $push: "$title" }
          }
      }
])
```


Résultat

```
[
  {
    _id: 'Dante',
    books: [ 'The Banquet', 'Divine Comedy', 'Eclogues' ]
  },
  { _id: 'Homer', books: [ 'The Odyssey', 'Iliad' ] }
]
```

MongoDB

Exemple avec `match`

```
db.books.aggregate(
[
    {
        $match : { author : "Dante" }
    }
]);
```

MongoDB

Exemple avec `match`

```
db.books.aggregate(
[
    {
        $match : { author : "Dante" }
    }
]);
```

Résultat

```
[
  { _id: 8751, title: 'The Banquet', author: 'Dante', copies: 2 },
  { _id: 8752, title: 'Divine Comedy', author: 'Dante', copies: 1 },
  { _id: 8645, title: 'Eclogues', author: 'Dante', copies: 2 }
]
```