Les 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

MongoDB

MongoDB

- système de gestion de base de données **NoSQL** orientée documents
- créé en 2007
- open-source
- développé en C++
- disponibilité de plusieurs fonctionnalités **SQL** (`COUNT`, `GROUP BY`, `ORDER BY`, `SUM`...)
- possibilité d'accéder aux données via une console **JavaScript**
- des drivers disponibles pour plusieurs langages de programmation (**Java**, **JavaScript**, **PHP**, **Python**, **Ruby**...)
- données stockées sous format **JSON** (JavaScript Object Notation)
- SGBD NoSQL le plus populaire en 2017 (et le plus utilisé en **France**)
- utilisé par MTV, Disney, Doodle, Adobe, eBay...

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

Téléchargement, installation et mise en place

- Allez sur le lien `https://www.mongodb.com/try/download/community` (onglet `Community server`), choisissez la version `4.4.1`, `msi` comme `Package` et **Windows** comme `Plarform`.
- Installez le fichier téléchargé.
- Sous la racine du disque dur (`C:\` sous **Windows**), créez l'arborescence `data\db` : c'est l'arborescence par défaut qui sera cherchée par **MongoDB**.

MongoDB

Ajouter **MongoDB** au path de **Windows**

- Copiez le chemin absolu du répertoire `bin` de **MongoDB** dans `Programmes files`
- Dans la barre de recherche, cherchez `Système` ensuite cliquez `Paramètres système avancés`
- Choisissez `Variables d'environnement` ensuite dans `Variables utilisateur` cliquez sur `Nouvelle`
- Saisissez comme nom de variable `PATH` et comme valeur le chemin absolu du répertoire `bin` de **MongoDB** dans `Programmes files`

MongoDB

Connexion personnalisée

- Ouvrez une console (invite de commandes)
- Créez l'arborescence `data/db` (on peut la créer n'importe où sur le disque dur)
- Dans la console, exécutez `start -B mongod --port numeroPort --dbpath cheminDeData/db`
- Cliquez sur la touche `Entrer`
- Exécutez `mongo --port numeroPort`

MongoDB

(Autres) commandes **MongoDB**

- `mongod` : démarrer le serveur
- `mongo` : démarrer une connexion client
- `mongoimport` : pour importer une base de données
- `mongoexport` : pour exporter une base de données
- ...

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

MongoDB : types de données

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

MongoDB

Connexion et création d'une base de données

- Après connexion : `use nomBD`
- En se connectant : `mongo --port numeroPort nomBD` (numéro de port par défaut : `27015`)
- Ou sans préciser le numéro de port : `mongo nomBD`

MongoDB

Connexion et création d'une base de données

- Après connexion : `use nomBD`
- En se connectant : `mongo --port numeroPort nomBD` (numéro de port par défaut : `27015`)
- Ou sans préciser le numéro de port : `mongo nomBD`

Remarque

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

MongoDB

Connexion et création d'une base de données

- Après connexion : `use nomBD`
- En se connectant : `mongo --port numeroPort nomBD` (numéro de port par défaut : `27015`)
- Ou sans préciser le numéro de port : `mongo nomBD`

Remarque

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

Exemple de création et d'utilisation d'une base de données `ma_base`

```
use ma_base
```

MongoDB

Suppression de la base de données courante

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

MongoDB

Lister les bases de données existantes

```
show dbs
```

MongoDB

Lister les bases de données existantes

```
show dbs
```

Par défaut

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

MongoDB

Rappel

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

MongoDB

Rappel

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

Création d'une collection

Deux solutions :

- Directement : `db.createCollection('adresse')`
- En insérant un document : `db.personne.insert({nom: "Wick", prenom: "john"})`

MongoDB

Suppression d'une collection

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

MongoDB

Lister les collections existantes

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

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.

Ajout d'un document

Deux syntaxes :

- `db.nomCollection.insert({clé1: 'val1', clé2: 'val2' ... })`
- ou bien `obj={clé1 :'val1', clé2: 'val2' ... })` ensuite `db.nomCollection.save(obj)` (`save` ajoute l'élément s'il n'existe pas, sinon elle le modifie en écrasant le contenu précédent)

MongoDB

Exemple avec `insert`

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

MongoDB

Exemple avec `insert`

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

Exemple avec `save`

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

MongoDB

Exemple en définissant l'objet avant

```
objet = {
    nom: 'dalton',
    prenom: 'jack',
    niveau: 'master',
}

db.personne.save(objet)
```

MongoDB

Exemple avec identifiant personnalisé

```
db.personne.save(
        {
            _id: 10,
            nom: 'dalton',
            prenom: 'jack',
            niveau: 'master',
        }
)
```

MongoDB

`save` VS `insert`

- `save` et `insert` permettent l'insertion et l'insertion multiple.
- `save` ajoute ou modifie un document.
- `insert` ajoute seulement de nouveaux documents.
- `save` effectue une modification si l'`_id` spécifié existe dans la collection.
- `insert` génère une erreur si l'`_id` spécifié existe dans la collection.

MongoDB

Pour vérifier que l'ajout a eu lieu

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

MongoDB

Pour faire plusieurs insertions au même temps

```
db.nomCollection.insert( [ {clé1 : 'val1', clé2 :
'val2' ... }, {cléN : 'valN', cléM : 'valM' ... } ]
)
```

MongoDB

Exemple

```
db.personne.insert(
        [
                {
                    nom: 'wick',
                    prenom: 'john',
                    age: 45,
                    sportif: true
                },
                {
                    _id: 12,
                    nom: 'dalton',
                    prenom: 'jack',
                    niveau: 'master',
                }
        ]
)
```

MongoDB

La modification

- `update()` : pour modifier un ou plusieurs documents selon une ou plusieurs conditions.
- `save()` : pour remplacer toutes les valeurs d'un document selon l'identifiant par les valeurs indiquées dans la méthode. Si l'identifiant n'existe pas il sera ajouté.
- `updateOne()` : pour modifier uniquement le premier enregistrement de la sélection (par défaut).
- `updateMany()` : pour modifier plusieurs documents.
- `replaceOne()` : pour remplacer le premier élément de la sélection.

MongoDB

La méthode `update`

`db.nomCollection.update()` prend au moins deux paramètres :

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

MongoDB

Exemple

```
db.personne.update(
        {
            _id: 12
        },
        {
            prenom: 'bill',
        }
)
```

MongoDB

Exemple

```
db.personne.update(
        {
            _id: 12
        },
        {
            prenom: 'bill',
        }
)
```


Remarque

Le document ayant l'identifiant 12 a désormais un seul champ (hormis l'identifiant) : `prenom`. Les champs `nom` et `niveau` ont été supprimés.

Pour modifier, on peut utiliser un des opérateurs suivants

- `$set` : pour modifier la valeur d'un champ
- `$unset` : pour supprimer un champ
- `$inc` : pour incrémenter la valeur d'un champ
- `$mul` : pour multiplier l'ancienne valeur d'un champ par la valeur spécifiée
- `$rename` : pour renommer un champ
- `$min` : pour modifier la valeur d'un champ si elle est supérieure à la valeur spécifiée par min (et inversement pour max)
- ...

Pour modifier, on peut utiliser un des opérateurs suivants

- `$set` : pour modifier la valeur d'un champ
- `$unset` : pour supprimer un champ
- `$inc` : pour incrémenter la valeur d'un champ
- `$mul` : pour multiplier l'ancienne valeur d'un champ par la valeur spécifiée
- `$rename` : pour renommer un champ
- `$min` : pour modifier la valeur d'un champ si elle est supérieure à la valeur spécifiée par min (et inversement pour max)
- ...

Syntaxe

```
db.nomCollectiondb.update(
  {clé1: 'val1' ... },
  {$set: {cléN: 'valN' ... }}
)
```

MongoDB

Exemple

```
db.personne.update(
        {
            _id: 12
        },
        {
            $set: {
                prenom: 'peter',
                nom: 'white',
                age: 45
            }
        }
)
```

MongoDB

Exemple

```
db.personne.update(
        {
            _id: 12
        },
        {
            $set: {
                prenom: 'peter',
                nom: 'white',
                age: 45
            }
        }
)
```

Constats

- Le `prenom` a été modifié.
- Les deux champs `nom` et `age` ont été ajoutés.

MongoDB

Exemple avec `save`

```
db.personne.save(
        {
            _id: 12,
            prenom: 'james',
            nom: 'hamilton',
            genre: 'h'
        }
)
```

MongoDB

Exemple avec `save`

```
db.personne.save(
        {
            _id: 12,
            prenom: 'james',
            nom: 'hamilton',
            genre: 'h'
        }
)
```


Remarque

Les champs existant avant modification non mentionnés dans `save` seront supprimés.

MongoDB

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

```
db.personne.update(
        {
            prenom: 'john'
        },
        {
            $set: {
                nom: 'abruzzi'
            }
        }
)
```

MongoDB

Pour modifier tous les documents qui remplissent la condition, on ajoute le paramètre `multi: true`

```
db.personne.update(
        {
            prenom: 'john'
        },
        {
            $set: {
                nom: 'abruzzi'
            }
        },
        {
            multi: true
        }
)
```

MongoDB

Ou en utilisant `updateMany`

```
db.personne.updateMany(
        {
            prenom: 'john'
        },
        {
            $set: {
                nom: 'abruzzi'
            }
        }
)
```

MongoDB

Si aucun document ne remplit la condition de `update`, aucun changement ne sera effectué

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

MongoDB

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

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

MongoDB

Et si on veut supprimer un champ d'un document

```
db.personne.update({nom: "bob"}, {$unset:{prenom:1}})
```

MongoDB

Et si on veut supprimer un champ d'un document

```
db.personne.update({nom: "bob"}, {$unset:{prenom:1}})
```

Explication

- On commence par sélectionner les documents dont le champ `nom` contient comme valeur `bob`
- Si on ne précise aucun critère de sélection, le premier document de la collection `personne` sera concerné par la modification
- Pour ce(s) document(s), on supprimera le champ `prenom` s'il existe.

MongoDB

Et si on veut incrémenter un champ d'un document

```
db.personne.update({nom: "bob"}, {$inc:{age:20}})
```

MongoDB

Et si on veut incrémenter un champ d'un document

```
db.personne.update({nom: "bob"}, {$inc:{age:20}})
```

Explication

- On commence par sélectionner les documents dont le champ `nom` contient comme valeur `bob`
- Si on ne précise aucun critère de sélection, le premier document de la collection `personne` sera concerné par la modification
- Pour ce(s) document(s), on incrémentera de 20 l'`age` pour s'il existe. Sinon, le champ `age` sera créé avec la valeur `20`.

MongoDB

On peut aussi faire plusieurs modifications avec une seule requête

```
db.personne.update({nom: "bob"}, {$inc:{age:20},
$unset:{prenom:1} })
```

MongoDB

On peut aussi faire plusieurs modifications avec une seule requête

```
db.personne.update({nom: "bob"}, {$inc:{age:20},
$unset:{prenom:1} })
```

Explication

Avec cette requête, on incrémente l'age et on supprime le champ prenom pour le premier document ayant un champ nom qui a comme valeur `bob`

MongoDB

Et si on veut renommer un champ

```
db.personne.updateMany( {}, { $rename: { nom: 'name'
} } )
```

MongoDB

Et si on veut renommer un champ

```
db.personne.updateMany( {}, { $rename: { nom: 'name' } } )
```

Explication

- On commence par sélectionner tous les documents
- Pour ces documents, le champ `nom` sera renommé `name`

MongoDB

Supprimer un document

```
db.personne.remove({nom: "bob"})
```

MongoDB

Supprimer un document

```
db.personne.remove({nom: "bob"})
```

Explication

- On commence par sélectionner les documents dont le champ `nom` contient comme valeur `bob`
- Ensuite tous ces documents seront supprimés

MongoDB

On peut également utiliser

- `deleteOne()`
- `deleteMany()`

MongoDB

Récupérer tous les documents d'une collection

```
db.personne.find()
```

MongoDB

Récupérer tous les documents d'une collection

```
db.personne.find()
```

Rechercher selon des critères

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

MongoDB

Récupérer tous les documents d'une collection

```
db.personne.find()
```

Rechercher selon des critères

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

Compter le nombre de documents

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

MongoDB

Trier le résultat de recherche dans l'ordre croissant

- `db.personne.find().sort({name: 1})`

MongoDB

Trier le résultat de recherche dans l'ordre croissant

- `db.personne.find().sort({name: 1})`

Trier le résultat de recherche dans l'ordre décroissant

```
db.personne.find().sort({name: -1})
```

MongoDB

Sauter quelques documents (ne pas les afficher)

- `db.personne.find().skip(2)`

MongoDB

Sauter quelques documents (ne pas les afficher)

- `db.personne.find().skip(2)`

Limiter le nombre de documents à afficher

- `db.personne.find().limit(2)`

MongoDB

Sauter quelques documents (ne pas les afficher)

- `db.personne.find().skip(2)`

Limiter le nombre de documents à afficher

- `db.personne.find().limit(2)`

Trier et limiter le nombre de documents à afficher

- `db.personne.find().sort({name: -1}).limit(2)`

MongoDB

Sauter quelques documents (ne pas les afficher)

- `db.personne.find().skip(2)`

Limiter le nombre de documents à afficher

- `db.personne.find().limit(2)`

Trier et limiter le nombre de documents à afficher

- `db.personne.find().sort({name: -1}).limit(2)`

Afficher seulement le premier document

- `db.personne.findOne()`

MongoDB

Afficher le résultat au format JSON

- `db.personne.find().pretty();`

MongoDB

Afficher le résultat au format JSON

- `db.personne.find().pretty();`

Et si on veut seulement afficher quelques champs

- `db.personne.find({},{nom:1})`
- ça affiche le nom de toutes les personnes ainsi que leurs identifiants (qui sera affiché automatiquement)

MongoDB

Afficher le résultat au format JSON

- `db.personne.find().pretty();`

Et si on veut seulement afficher quelques champs

- `db.personne.find({},{nom:1})`
- ça affiche le nom de toutes les personnes ainsi que leurs identifiants (qui sera affiché automatiquement)

Et si on ne veut pas afficher les `_id`

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

MongoDB

Appeler une fonction pour chaque document de la sélection

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

MongoDB

Appeler une fonction pour chaque document de la sélection

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


Explication

- Pour chaque document de la sélection (`forEach`), on appelle une fonction qui affiche le nom et le prénom.

MongoDB

Peut-on utiliser les expression régulières ?

chercher les personnes dont le nom commence par w :

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

MongoDB

Peut-on utiliser les expression régulières ?

chercher les personnes dont le nom commence par w :

- `db.personne.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.personne.find({name:/k$/})`

chercher les personnes dont le nom commence par e ou par h :

- `db.personne.find({name:/^[eh]/})`

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

- `db.personne.find({name:/^[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.employes.find({prenom: { $regex: /john/}})`

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

- `db.employes.find({prenom: { $regex: /john/, $options: 'i' }})`

MongoDB

Listes des 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.personne.insert({ _id : 5, nom : 'wick', sport: [ 'foot', 'hand', 'tennis'] })`

MongoDB

Ajouter un nouveau sport au tableau

- `db.personne.update( { _id: 5 }, { $push: { "sport": "basket" } } )`

Comment ajouter plusieurs sports avec une seule requête ? Ainsi :

- `db.personne.update( { _id: 5 }, { $push: { sport: ['hockey','sky'] } } )`

MongoDB

Ajouter un nouveau sport au tableau

- `db.personne.update( { _id: 5 }, { $push: { "sport": "basket" } } )`

Comment ajouter plusieurs sports avec une seule requête ? Ainsi :

- `db.personne.update( { _id: 5 }, { $push: { sport: ['hockey','sky'] } } )`

Non, ça rajoute un tableau dans notre tableau

MongoDB

Ou comme-ça ?

```
db.personne.update( { _id: 5 }, { $push: { sport:
{'hockey','sky'} } } )
```

MongoDB

Ou comme-ça ?

```
db.personne.update( { _id: 5 }, { $push: { sport:
{'hockey','sky'} } } )
```

Non, ça génère une erreur

MongoDB

Solution

- `db.personne.update( { _id: 5 }, { $push: { "sport": { $each: ['basket','sky'] } } } )`

Remarque

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

MongoDB

Et si on veut ajouter un élément à une position précise

- `db.personne.update( { _id: 5 }, { $push: { "sport": { $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

Comment supprimer le premier élément d'un tableau ?

- `db.personne.update( { _id: 5 }, { $pop: { sport: -1 } } )`

Comment supprimer le dernier élément d'un tableau ?

- `db.personne.update( { _id: 5 }, { $pop: { sport: 1 } } )`

Comment supprimer un élément quelconque d'un tableau ?

- `db.personne.update( { _id: 5 }, { $pull: { "sport": "foot" } } )` : supprime l'élément `foot` du tableau `sport`

MongoDB

Comment supprimer plusieurs éléments avec une seule requête ?

- `db.personne.update( { _id: 5 }, { $pull: { sport: { $in: ['hockey','basket'] } } } )`

MongoDB

Considérons le document créé de la façon suivante :

```
db.personne.insert({
        _id : 6,
        nom : 'wick',
        sport : [ 'foot', 'hand', 'tennis']
})
```

MongoDB

Si on exécute

```
db.personne.update(
    { _id: 6 },
    {$push: {
        sport: {
            $each: [ 'hockey', 'sky', 'volley' ],
            $slice: -5
        }
    }
})
```

MongoDB

Ensuite

```
db.personne.find({_id:6}).pretty();
```

Le résultat sera :

```
{
    "_id" : 6,
    "nom" : "wick",
    "sport" : [
        "hand",
        "tennis",
        "hockey",
        "sky",
        "volley"
    ]
}
```

MongoDB

Considérons le document créé de la façon suivante :

```
db.personne.insert({
        _id : 7,
        nom : 'wick',
        sport : [ 'foot', 'hand', 'tennis']
})
```

MongoDB

Si on exécute

```
db.personne.update(
    { _id: 7 },
    {$push: {
        sport: {
            $each: [ 'hockey', 'sky', 'volley' ],
            $slice: 5
        }
    }
})
```

MongoDB

Ensuite

```
db.personne.find({_id:7}).pretty();
```

Le résultat sera :

```
{
    "_id" : 7,
    "nom" : "wick",
    "sport" : [
        "foot",
        "hand",
        "tennis",
        "hockey",
        "sky"
    ]
}
```

MongoDB

Considérons le document créé de la façon suivante :

```
db.personne.insert({
        _id : 8,
        nom : 'wick',
        sport : ['hand', 'foot',  'tennis']
})
```

MongoDB

Si on exécute

```
db.personne.update(
    { _id: 8 },
    {$push: {
        sport: {
            $each: ['sky', 'volley' , 'hockey'],
            $sort: 1
        }
    }
})
```

MongoDB

Ensuite

```
db.personne.find({_id:8}).pretty();
```

Le résultat sera :

```
{
    "_id" : 8,
    "nom" : "wick",
    "sport" : [
        "foot",
        "hand",
        "hockey",
        "tennis",
        "sky",
        "volley"
    ]
}
```

MongoDB

Pour trier dans l'ordre décroissant

- `$sort: -1`

MongoDB

Et quand il s'agit d'un tableau d'objet ?

Considérons le document suivant :

```
db.personne.insert({
    _id : 10,
    nom : 'wick',
    notes: [
        {'programmation': 17, 'coefficient': 4},
        {'OS': 10, 'coefficient': 2}
    ]
})
```

MongoDB

Comment ajouter un nouvel élément au tableau ?

```
db.personne.update(
        { _id: 10 },
        { $push:
                { notes :
                        {'compilation': 15, '
                           coefficient': 1}
                }
        }
)
```

MongoDB

Et pour supprimer ?

```
db.personne.update(
    { _id: 10 },
    { $pull:
        { notes :
            {'compilation': 15, 'coefficient': 1}
        }
    }
)
```

MongoDB

Considérons le document suivant :

```
db.personne.insert({
    _id : 11,
    nom : 'wick',
    notes: [
        {'programmation': 17, 'coefficient': 4,
          optionnel: false},
        {'OS': 10, 'coefficient': 2}
    ]
})
```

MongoDB

Que fait la requête de suppression suivante ?

```
db.personne.update(
    { _id: 11 },
    { $pull:
        { notes :
            {'programmation': 17, 'coefficient': 4}
        }
    }
)
```

MongoDB

Que fait la requête de suppression suivante ?

```
db.personne.update(
    { _id: 11 },
    { $pull:
        { notes :
            {'programmation': 17, 'coefficient': 4}
        }
    }
)
```

Elle supprime quand-même l'objet même s'il n'y pas de correspondance complète

MongoDB

Comment faire pour éviter cette problématique ?

```
db.personne.update(
    { _id: 11 },
    { $pull: {
        notes : {
            $elemMatch:
                {'programmation': 17, 'coefficient':
                    4}
        }
     }
} )
```

MongoDB

Comment faire pour éviter cette problématique ?

```
db.personne.update(
    { _id: 11 },
    { $pull: {
        notes : {
            $elemMatch:
                {'programmation': 17, 'coefficient':
                    4}
        }
     }
} )
```

Cette fois-ci, ça ne supprime pas l'objet car l'attribut `optionnel: true` n'a pas été précisé

MongoDB

Pour chercher un document selon une valeur dans son tableau d'objet

```
db.personne.find(
        {"notes.programmation":
                {$ne:17}
        }
)
```

MongoDB

Pour chercher un document selon une valeur dans son tableau d'objet

```
db.personne.find(
        {"notes.programmation":
                {$ne:17}
        }
)
```

Cela permet de chercher toutes les personnes dont la note en programmation est différente de 17.

MongoDB

Comme pour les bases de données relationnelles

- opérateurs de comparaison
- opérateurs logiques
- ...

MongoDB

Commençons par créer la collection suivante (etudiant) :

```
{"_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

Comment sélectionner les étudiants âgés de plus de 30 ans

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

MongoDB

Comment sélectionner les étudiants âgés de plus de 30 ans

```
db.etudiant.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

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

```
db.etudiant.find(
    {$and:
        [
            { age:{$gte:20}},
            { age:{$lte:30}}
        ]
    }
)
```

MongoDB

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

```
db.etudiant.find(
    {$and:
        [
            { age:{$gte:20}},
            { age:{$lte:30}}
        ]
    }
)
```

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

{"_id" : 3, "nom" : "wolf", "notes": [7, 6, 13], "age" : 25 }

{"_id" : 4, "nom" : "green", "notes": [18, 16, 9], "age" : 22 }
```

MongoDB

Les opérateurs logiques

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

MongoDB

Afficher les personnes dont le champ `name` existe

```
db.personne.find(
        {name:
                {$exists:true}
        }
)
```

MongoDB

Afficher les personnes dont le champ `name` existe

```
db.personne.find(
        {name:
                {$exists:true}
        }
)
```

Afficher les personnes dont l'age est divisible par 5

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

MongoDB

C'est quoi ? et pourquoi ?

- 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 (et il est impossible de le supprimer)

MongoDB

Pour consulter la liste d'index sur la collection personne

`db.personne.getIndexes();` ça affiche :

```
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.personne"
        }
]
```

MongoDB

Pour créer un nouvel index sur le champ nom

`db.personne.ensureIndex(nom:1);` ça affiche :

```
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

MongoDB

On peut consulter une nouvelle fois la liste d'index

`db.personne.getIndexes();` ça affiche :

```
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.personne"
        },
        {
                "v" : 2,
                "key" : {
                        "nom" : 1
                },
                "name" : "nom_1",
                "ns" : "test.personne"
        }
]
```

MongoDB

Pour supprimer un index sur le champ nom

```
db.personne.dropIndex(nom:1)
```

On vérifie

`db.personne.getIndexes();` ça affiche :

```
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.personne"
        }
]
```

MongoDB

C'est quoi ? et pourquoi ?

- C'est une requête qui retourne un résultat sous forme d'un tableau
- ça permet d'utiliser des fonctions d'agrégations comme en SQL

MongoDB

Exemple

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

MongoDB

Exemple

```
db.etudiant.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
- ça permet d'utiliser des opérateurs d'agrégations comme en SQL

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`
- ...

MongoDB

Exemple avec `sample`

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

MongoDB

Exemple avec `sample`

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

Le résultat

Il choisit aléatoirement trois documents de la collection

MongoDB

Exemple avec `unwind`

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

MongoDB

Exemple avec `unwind`

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

Le 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

Exemple avec `out`

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

MongoDB

Exemple avec `out`

```
db.etudiant.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

```
{ "_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

Exemple avec `group`

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

MongoDB

Exemple avec `group`

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

Le résultat :

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

MongoDB

Exemple avec `match`

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

MongoDB

Exemple avec `match`

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

Le 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 }
```

MongoDB

Documentation officielle

- `https://docs.mongodb.com/`