Index

Objectifs du chapitre

- ✓ Créer un Index simple
- ✓ Créer un Index composé
- ✓ Créer un Index unique

Index simple

L’utilisation d’index permet d'accéder à une information plus rapidement.

Une table d’index est comparable à une table des matières d’un livre de cuisine. Cette table permet d’accéder rapidement à une recette dans un livre.

Pour de meilleur performance, l’index d’une collection doit pouvoir tenir en mémoire. Autrement, MongoDB devra lire les données depuis le disque, ce qui pénalisera l’application.

Par défaut, le champ _id est indexé ! Cet index ne peut pas être supprimé.

Création d'un index simple

Considérons ce document :

```
{ "_id" : ObjectId(...),
   "name" : "Alice",
   "age" : 27
}
```

La commande suivante créé un index sur le champ "name"

```
db.friends.createIndex( { "name" : 1 } )
```

La valeur 1 permet d'indiquer dans quel sens l'index doit être parcouru.

Création d'un index simple sur un sous document

Considérons ce document :

```
{"_id": ObjectId(...),
   "name": "John Doe",
   "address": {
        "street": "Main",
        "zipcode": "53511",
        "state": "WI"
   }
```

La commande suivante créé un index sur le champ "zipcode" du sous document "address"

```
db.people.createIndex( { "address.zipcode": 1 } )
```

Création d'un index composé

Quand une requête utilise plusieurs champs pour identifier un document il est nécessaire d'utiliser un index composé.

```
{ "_id" : ObjectId(...),
   "name" : "Alice",
   "age" : 27
}
```

Par exemple, si la collection people comporte beaucoup d'Alice, il est nécessaire d'utiliser également l'age comme index

```
db.people.find( { "name": "Alice", "age": 27 } )
```

La commande suivante créé un index composé sur ces deux champs :

```
db.people.createIndex( { "name": 1, "age": 1 } )
```

Création d'un index unique

Un index unique permet de s'assurer l'unicité d'un document, sans forcement devoir utiliser le champ _id

```
{ "_id" : 123,
  "name" : "Alice"
}
```

```
{ "_id" : 456,
  "name" : "Alice"
}
```

Un index unique sur le champ "name" empêchera la création de documents ayant la même valeur dans le champ "name".

La commande suivante créé un index unique :

```
db.people.createIndex( { "name": 1 }, { unique: true } )
```

Index avancé

Créer un index TTL

Un index TTL (Time-To-Live) permet de limiter dans le temps la vie d'un document. Quand ce document aura atteint sa date de péremption, alors il sera supprimé par MongoDB.

```
db.eventlog.createIndex( { "lastModifiedDate": 1 },
                         { expireAfterSeconds: 3600 } )
```

Dans cet exemple de requête, le document sera supprimé quand le champ "lastModifiedDate" aura une date égale ou inférieur à maintenant, moins une heure.

Par exemple, si nous sommes le 06 mars 2019 à 10h30, le document ci-dessous sera supprimé dans 30 minutes :

```
{"_id": 1, "lastModifiedDate":
                         ISODate("2019-06-10T10:00:00Z")}
```

Framework d’aggregation

Objectifs du chapitre

- ✓ Comprendre l’aggregation framework
- ✓ Connaître les opérateurs disponibles
- ✓ Utiliser différents opérateurs

Comprendre l’aggregation Framework

L’aggregation framework permet d'exécuter des opérations sur un ensemble de document.

Ces opérations de filtrage, groupes, etc, permettent d'obtenir un résultat venant d’un ensemble d’opération sur un groupe de document.

Il est généralement admit que l’aggregation framework est le pendant des opérations GROUP BY du monde SQL.

Comprendre l'aggregation Framework

```
{ "_id": "10280",
  "city": "NEW YORK",
  "state": "NY",
  "pop": 5574,
  "loc": [ -74.016323, 40.710537 ]
}
```

Prenons l'ensemble des villes des Etats-unis avec leurs populations.

Via l'aggregtion framework, on pourra regrouper l'ensemble des villes par état (state) et compter la population par état. Ensuite, on pourra filtrer pour avoir la liste des états ayant une population supérieur à 10 000 personnes.

Connaître les opérateurs disponibles

L'aggregation framework est une pipeline qui enchaîne les opérations.

```
db.zipcodes.aggregate( [
   { $group: { _id: "$state", totalPop: { $sum: "$pop" } } },
   { $match: { totalPop: { $gte: 10*1000*1000 } } }
] )
```

$project	opère la projection d'un document : il est possible d'ajouter/supprimer des champs d'un document et ainsi obtenir un nouveau document
$match	filtre les documents selon certains critères
$limit	Limite le nombre de résultat à un certain nombre de document

Connaître les opérateurs disponibles

$skip	passe les n premiers documents & traite les suivants
$unwind	“explose” un tableau d’un document pour créer autant de document que d'élément du tableau. Chaque nouveaux documents ayant à la place du tableau une seule valeur.
$group	Regroupe les documents selon une clé
$sort	Ordonne les documents selon un critère de tri

Opérateur $match

L'opérateur $match permet de limiter l'ensemble de données.

Il est préférable d'utiliser l'opérateur $match en début du pipeline d'agrégation pour limiter le nombre de document à traiter.

Cette limitation évitera à mongodb de traiter des données inutiles.

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

Opérateur $project

L'opérateur $projet permet de composer un nouveau document, à l'aide de "spécification".

`<champ> : 1` indique la présence du champ
`_id : 0` indique la suprresion du champ _id
`<champ> : <expression>` ajoute un champ avec une valeur ou à partir d'un autre champ

```
db.articles.aggregate(
    [ { $project : { city : 1, country: "USA" } } ]
);
```

Opérateur $unwind

$unwind permet d'exploser le tableau d'un document en une multitude de document.

```
db.items.insert({ "_id" : 1, "item" : "ABC1", sizes: [ "S", "M", "L"] })

db.items.aggregate( [ { $unwind : "$sizes" } ] );

// output

{ "_id" : 1, "item" : "ABC1", "sizes" : "S" }

{ "_id" : 1, "item" : "ABC1", "sizes" : "M" }

{ "_id" : 1, "item" : "ABC1", "sizes" : "L" }
```

Opérateur $group

L'opérateur $group permet de faire un regroupement de document. Lors de ce regroupement, il est possible de faire appel à différents opérateurs d'accumulation.

exemple : $sum, $avg, $first, $last, $max, $min, $push, $addToSet

```
db.zipcodes.aggregate( [
   { $group: { _id: "$state", totalPop: { $sum: "$pop" } } },
   { $match: { totalPop: { $gte: 10*1000*1000 } } }
] )
```