MONGODB (partie 2)

Suppresion (rappel)

- remove()

  Supprimer un document: db.collection.remove()

  - Supprime un seul document contenant userid “Andrew” : db.users.remove({userid:”Andrew”,justOne:true})
  - Supprime tous les documents avec l'attributs age >25 : db.users.remove({Age:{$gt:25}})
  - Supprime tous les documents de la collection (vide la collection équivalent de truncate en sql) : db.users.remove({})
    !! Ne confondez pas avec la suppression d'une collection: db.collection.drop()

    et ne pas faire de bêtises avec db.dropDatabase() (devinez à quoi cela sert ;))

Solution Exercice :

Insérer dans la collection movies un nouveau film.

db.movies.insert({_id:'movie:44444',title:'bigdata',genre:'newstyle',year:2019,summary:'plein de blabla',country:'FR',...

Afficher la liste des films en base.

db.movies.find()

// ou plus fin, on va le voir plus loin

db.movies.aggregate([{$project:{title:1}}])

Afficher le premier film en base.

db.movies.findOne()

Rajouter {"oscar":4} pour le film intitulé Million Dollar Baby.

db.movies.update({title :'Million Dollar Baby'}, {title :'Million Dollar Baby' ,oscar: 4}})

Rajouter un oscar pour tous les films.

db.movies.update({}, {oscar: 4}})

Supprimer le film intutilé vertigo de la collection.

db.movies.remove{title :'vertigo'}

Recherche

On peut rechercher un attribut selon le type BSON ($type) :

db.collection.find({ name: { $type : 2 } })

on cherche l'attribut 'name' de type string ,les numéros des types sont ici.

Solution Exercice :

Déterminer le nombre de films en base

db.movies.count()

Afficher à partir du cinquième film en base

db.movies.find().skip(5)

Limiter l'affichage à 20

db.movies.find().limit(20)

Afficher les films du numéro 5 jusqu'au numéro 20

db.movies.find().limit(20).skip(5)

Trier les films dans l'ordre descendant

db.movies.find().sort({ title: -1 })

Les index

les index permettent d'accélérer les requêtes en évitant au moteur de base de données de devoir scanner tous les documents de la collection afin de trouver ceux correspondant à la requête.

Voici les commandes les plus courantes :

Créer un index
```
db.users.createIndex({title: 1})
```

Créer un index et préciser que la valeur doit être unique
```
db.users.createIndex({title: 1}, {unique: true})
```

L'index ci-dessus considérera une absence de valeur comme une valeur. Il n'y a donc qu'un « title » qui pourra être vide ensuite mongo considérera les autres « title » sans valeur comme dupliqués et refusera de les ajouter.

Pour empêcher ce comportement, on peut utiliser l'option sparse.

```
db.users.createIndex({title: 1}, {unique: true, sparse: true})
```

pour rechercher en mode fulltext et non en exact match (vu plus bas), il est possible d'utiliser l'index de type text
```
db.users.createIndex({firstname: "text"})
```

il ne peut y avoir qu'un seul index text par collection cependant, on peut les combiner
```
db.users.createIndex({firstname: "text", lastname: "text"})
```

afficher les index d'une collection
```
db.collection.getIndexes()
```

supprimer un index (le nom de l'index est fourni par getIndexes)
```
db.collection.dropIndex("indexName")
```

Lorsqu'on recherche un document donné, il est possible de connaître la stratégie effective permettant à MongoDB de le retrouver.

Il suffit d'utiliser la commande explain().

Dans ce cas MongoDB renvoie un document supplémentaire qui détaille sa recherche avec notamment les index utilisés, le nombre de documents parcourus ou encore le temps global de la requête.

C'est très pratique quand on veut optimiser ses requêtes.

Exemple **:** db.collection.find({}).explain()

Requêtes agrégées

MongoDB propose un système de requêtage avancé afin d'obtenir l'information désirée.

Celui-ci permet notamment de filtrer les résultats, de les compiler ou même de les transformer. Il y a trois façons d'utiliser les requêtes agrégées :

- le mapreduce
- count, distinct et group
- le pipeline

mapReduce

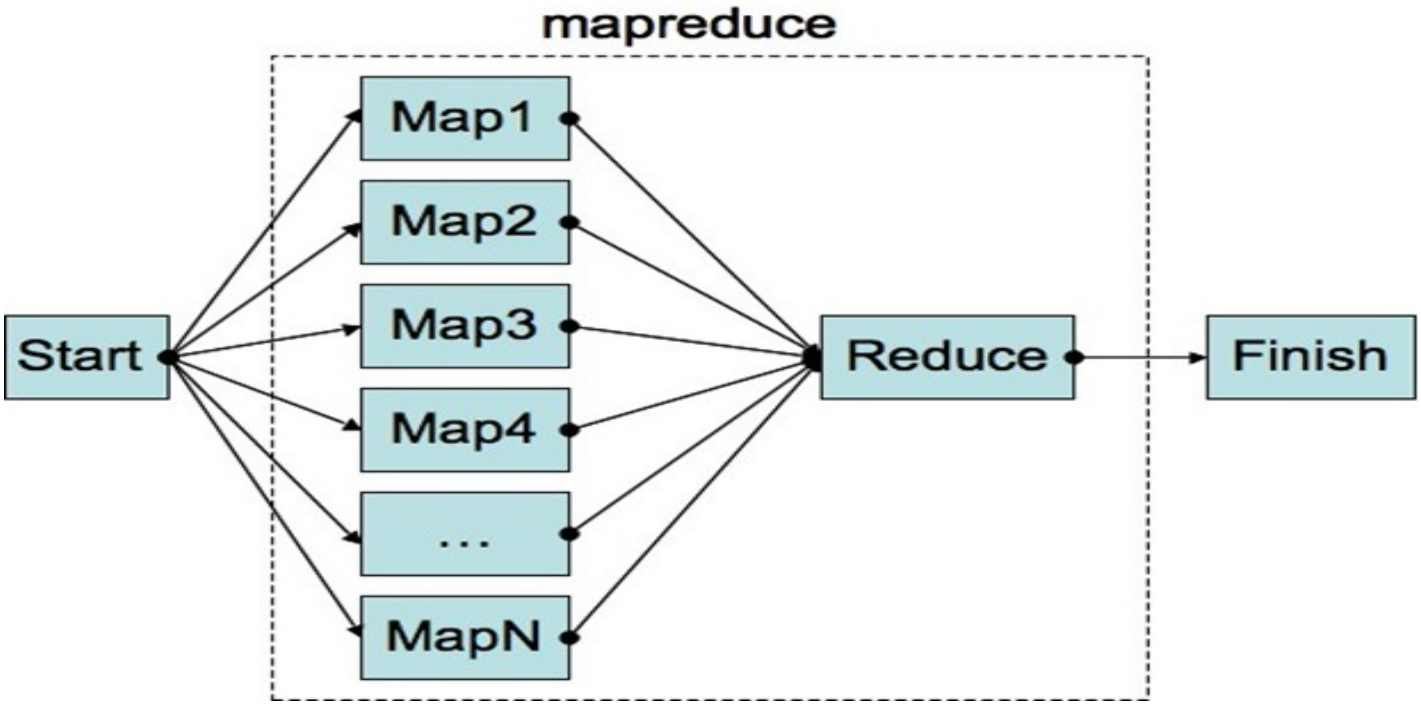
Donné à titre informatif, hors programme

Paradigme de traitement de données pour condenser de grands volumes de données en résultats agrégés utiles.

En termes très simples , la commande MapReduce prend deux entrées primaires , la fonction de mappage et la fonction de réducteur .

Un Mapper débutera par la lecture d'une collection de données pour construire une map avec les champs que nous souhaitons traiter.

Puis cette paire clé, valeur est introduite dans un réducteur , qui traitera les valeurs.



Exemple:

Affichage du nombre des films en base mongo par genre:

var map =function() {emit(this.genre,1);};

var reduce = function(key, values) { return Array.sum(values)};

var count = db.movies.mapReduce(map, reduce, {out: "movies_result"});

db.movies_result.find();

résultat :

{ "_id" : "Action", "value" : 11 }

{ "_id" : "Comédie", "value" : 4 }

{ "_id" : "Fantastique", "value" : 2 }

{ "_id" : "Guerre", "value" : 2 }

{ "_id" : "Horreur", "value" : 4 }

{ "_id" : "Science-fiction", "value" : 8 }

{ "_id" : "Suspense", "value" : 2 }

{ "_id" : "Thriller", "value" : 5 }

Needemand

{ "_id" : "Western", "value" : 3 }

{ "_id" : "crime", "value" : 6 }

{ "_id" : "drama", "value" : 17 }

{ "_id" : "romance", "value" : 1 }

Le pipeline

Le pipeline est un framework qui permet -- comme son nom l'indique -- d'enchaîner des actions pour transformer et traiter les résultats de requêtes spécifiques.

La commande utilisée est : db.collection.aggregate()

Exemple en utilisant *match* et *group* :

```
db.orders.aggregate([
   {$match: { genre: "Action" }},
   {$group: {
      _id: "$cust_id"
   }}
])
```

Exemple en utilisant *project* et *sort* :

```
db.users.aggregate([
   {$project: { name: { $toUpper: "$_id" } , _id: 0 }},
   {$sort: { name: 1 }}
])
```

Autre exemple :

```
db.users.aggregate([
   { $unwind : "$likes" },
   { $group : { _id : "$likes" , number : { $sum : 1 } } },
   { $sort : { number : -1 } },
   { $limit : 5 }
])
```

Issue de https://gist.github.com/alain-andre/8150256

- $project - Sélectionnez, remodeler - 1:1
- $match - Filtrer - n:1
- $group - Agréger - n:1
- $sort - Trier - 1:1
- $skip - Sauter - n:1
- $limit - Limiter - n:1
- $unwind - Découper un champ (array) - 1:n

$project

Permet de remodeler une collection.

Si l'on veut remodeler la collection ci-dessous

```
{
"city" : "ACMAR",
"loc" : [
    -86.51557,
    33.584132
],
"pop" : 6055,
"state" : "AL",
"_id" : "35004"
}
```

en

```
{
"city" : "acmar",
"pop" : 6055,
"state" : "AL",
"zip" : "35004"
}
```

Il nous faut faire l'aggregation suivante

```
db.zips.aggregate([
  { $project:
    {
      _id: 0,
      city: {$toLower: "$city" },
      pop: 1,
      state: 1,
      zip: "$_id"
    }
  }
])
```

$group

Permet de faire un group by SQL et ainsi de retrouver le nombre de X disponibles pour chaque Y.

```
{
  "_id" : ObjectId("50906d7fa3c412bb040eb58a"), "student_id" : 4,
  "type" : "homework",
  "score" : 28.656451042441
}
>>> db.grades.aggregate([
  {$group:
    {
      _id:"$type",
      num: {$sum:1}
    }
  }
])
{
  "result" : [
    {
      "_id" : "homework",
      "num" : 200
    },
    {
      "_id" : "quiz",
      "num" : 200
    },
    {
      "_id" : "exam",
      "num" : 200
    }
  ],
  "ok" : 1
}
```

$match

Recherche dans la collection. Par exemple les villes dont la population est supérieure à 100 000.

```
>>> db.zips.aggregate([
  { $match:
    {
      pop: {$gt: 100000}
    }
  }
])
"result" : [
    {
```

```
    "city" : "CHICAGO",
    "loc" : [
      -87.7157,
      41.849015
    ],
    "pop" : 112047,
    "state" : "IL",
    "_id" : "60623"
  },
```

$sort

Permet de liste par ordre croissant(1) ou décroissant(-1).

```
db.zips.aggregate([
  { $sort:
    {
      state: 1,
      city: 1
    }
  }
])
```

$skip, $limit

Ces opérateur de regroupement ne sont pertinents qu'après un $sort, et sont utilisés dans l'ordre donné.

Ceci par exemple ne donnera aucun résultat à cause de l'ordre $limit 5, $skip 10 donné. Si on limite la liste à 5, en sauter 10 ne retourne rien.

```
>>> db.zips.aggregate([
  {$match:
    {
      state:"NY"
    }
  },
  {$group:
    {
      _id: "$city",
      population: {$sum:"$pop"},
    }
  },
  {$project:
    {
      _id: 0,
      city: "$_id",
      population: 1,
    }
```

```
  },
  {$sort:
    {
      population:-1
    }
  },
  {$limit: 5},
  {$skip: 10}
])
```

$first, $last

Retourne le premier ou le dernier élément d'une liste pour chaque groupe donné.

Admettant la collection suivante

```
{ "_id" : 0, "a" : 0, "b" : 0, "c" : 21 }
{ "_id" : 1, "a" : 0, "b" : 0, "c" : 54 }
{ "_id" : 2, "a" : 0, "b" : 1, "c" : 52 }
{ "_id" : 3, "a" : 0, "b" : 1, "c" : 17 }
{ "_id" : 4, "a" : 1, "b" : 0, "c" : 22 }
{ "_id" : 5, "a" : 1, "b" : 0, "c" : 5 }
{ "_id" : 6, "a" : 1, "b" : 1, "c" : 87 }
{ "_id" : 7, "a" : 1, "b" : 1, "c" : 97 }
```

L'aggregation suivante nous retourne { "_id" : 1, "a" : 0, "b" : 0, "c" : 54 }

```
db.fun.aggregate([
    {$match:{a:0}},
    {$sort:{c:-1}},
    {$group:{_id:"$a", c:{$first:"$c"}}}
])
```

$unwind

Éclate un array en autant de documents que d'élément.

Admettant la collection suivante, un $unwind sur likes retournerait 6 documents.

```
{
  "_id" : "Barack Obama",
  "likes" : [
    "social justice",
    "health care",
    "taxes"
  ]
},
```

```
{
  "_id" : "Mitt Romney",
  "likes" : [
    "a balanced budget",
    "corporations",
    "binders full of women"
  ]
}
```

Les expressions

$sum

Additionne une valeur ou calcule la somme d'un champ

```
{
  "city" : "CLANTON",
  "loc" : [
    -86.642472,
    32.835532
  ],
  "pop" : 13990,
  "state" : "AL",
  "_id" : "35045"
}

>>> db.zips.aggregate([
 {
  $group:{
   _id: "$state",
   population: {$sum: "$pop"}
  }
 }
])
```

avg

Calcule la moyenne

```
>>> db.zips.aggregate([
  { $group:
    {
      _id: "$state",
      average_pop: {$avg: "$pop"}
    }
  }
])
```

addToSet

Ajoute au champ s'il n'existe pas déjà (permet d'obtenir une liste pour un group by)

```
>>>> db.zips.aggregate([
  { $group:
    {
      _id: "$city",
      postal_codes: {$addToSet: "$_id"}
    }
  }
])
{
  "_id" : "CENTREVILLE",
  "postal_codes" : [
    "22020",
    "49032",
    "39631",
    "21617",
    "35042"
  ]
}
```

$max, $min

Retourne la plus grande ou la plus petite valeur du group by

```
>>> db.zips.aggregate([
  { $group:
    {
      _id: "$state",
      pop: {$max: "$pop"}
    }
  }
])
{
  "_id" : "WI",
  "pop" : 57187
},
{
  "_id" : "WV",
  "pop" : 70185
},
```

Javascript

Il est assez lourd de devoir taper à chaque fois db.users.commande, la console Mongo est aussi un shell JavaScript,

il est donc tout à fait possible d'enregistrer des choses dans des variables !

on enregistre la collection dans une variable

```
t = db.collection
```

nous pouvons maintenant faire comme ceci

```
t.find().pretty()
```

EXERCICES

Exercice 1

Créez une nouvelle base de données nommée Bigdata et vérifiez qu'elle est sélectionnée.

Exercice 2

Créez une nouvelle collection nommée produits et y insérer le document suivant:

- nom: Macbook Pro
- fabriquant: Apple
- prix: 1299.99
- options:
    - Intel Core i5
    - Retina Display
    - Long life battery

Rajoutez un autre document dans produits:

- nom: Macbook Air
- fabriquant: Apple
- prix: 1099.99
- ultrabook: true
- options:
    - Intel Core i7
    - SSD

        - Long life battery

Et enfin un dernier document:

- nom: Thinkpad X230
- fabriquant: Lenovo
- prix: 999.99
- ultrabook: true
- options:
    - Intel Core i5
    - SSD
    - Long life battery

Exercice 3

Effectuez les requêtes de lecture suivantes:

1. Récupérez tous les produits.
2. Récupérez le premier produit
3. Trouvez l'id du Thinkpad et faites la requête pour récupérer ce produit avec son id.
4. Récupérez les produits dont le prix est supérieur à 1200$
5. Récupérez le premier produit ayant le champ ultrabook à true
6. Récupérez le premier produit dont le nom contient Macbook
7. Récupérez les produits dont le nom commence par Macbook

Exercice 4

1. Supprimez les deux produits dont le fabricant est Apple.
2. Supprimez le Lenovo X230 en utilisant uniquement son id.

Exercice 5

Dans cet exercice, nous allons modéliser un système de facturation très simple.

Voici deux factures:

Facture numéro 10012A:

- Date de facture: 2013-07-04
- Client:
    - Nom: Alexandre Lepetit

        - Courriel: mo@exemple.com
- Liste des produits (2 produits):
    - Code: MACBOOKAIR
    - Nom: Macbook Air
    - Prix: 999.99
    - Quantite: 1
    - Code: APPLESUPPORT
    - Nom: AppleCare 1 an
    - Prix: 149.99
    - Quantite: 1
- Total: 1149.98

Facture numéro 10013A:

- Date de facture: 2013-07-05
- Client:
    - Nom: Jacques Berger
    - Courriel: berger.jacques@lampe.com
- Liste des produits (1 produit) :
    - Code: LENOVOX230
    - Nom: Lenovo Thinkpad X230
    - Prix: 899.99
    - Quantite: 1
- Total : 899.99

Effectuez les actions suivantes:

1. Insérer les deux factures dans la base
2. Récupérer la facture avec le numéro est 10013A
3. Modifier la facture 10012A en changeant la date pour le 2013-07-03 et le courriel du contact pour alex@example.com
4. Récupérer la facture avec le produit vendu ayant un code LENOVOX230 utilisez l'opérateur $elemMatch qui s'applique à un tableau
5. Supprimer la facture 10012A

>Reprenez la collection extrait d'une base de

publications scientifiques, The DBLP Computer Science Bibliography

7. Trier les publications de "Toru Ishida" par titre de livre et par

page de début ;

8. Projeter le résultat sur le titre de la publication, et les pages ;

9. Compter le first_namebre de ses publications ;

10.Compter le first_namebre de publications depuis 2011 et par type ;

11.Compter le nombre de publications par auteur et trier le résultat par ordre croissant ;