mongodb

19 octobre 2018

Présentation

- Développé par 10gen (`www.10gen.com`)
- Système de stockage orienté document
- Distribué (Auto-Partitionnement pour scalabilité horizontale)
- Répliqué (Maître / Esclave)
- Utilisation de la RAM pour les données
- `http://docs.mongodb.org/manual/`

Concepts de base

- Architecture de stockage
  - Serveur : maître / esclaves
  - Bdd
  - Collections (tables)
  - Documents BSON (données)
  - Fields (colonnes)
- Les documents d'une même collection supportent des structures hétérogènes (schemaless)

Concepts de base

- Architecture de stockage
  - Index : Les champs d'une collection sont indexables (l'id du document l'est par défaut)
  - Curseur : Les requêtes d'interrogation renvoient un curseur de documents, itérable
  - Procédures stockées : Possibilité de stocker des procédures JS

Concepts de base

- Utilisation de JavaScript
  - Instructions dans le terminal (mongo shell)
  - Requètage (Requêtes complexes, aggrégations)
  - MapReduce (cf cours Hadoop)

BSON

- Binary JSON (bsonspec.org)
- (R)appel : (JavaScript Object Notation)

```
{
   "nom" : "toto"
   "age" : "42"
   "adresse" : {
        "rue" : "12 rue tabaga"
        "ville" : "Saint-Priest-la-Prugne"
        "cp" : "42830"
   }
}
```

BSON

- Richesse de types
  - String (UTF-8)
  - Double
  - Object
  - Array
  - Boolean
  - Date
  - ...

Fortement typé et sensible à la casse

Les documents

- ”item” : 1
- ”item” : ”1”
- ”Item” : 1

sont tous différents

BSON

- Richesse de drivers
  - PHP
  - Java
  - Ruby
  - C++
  - C#
  - ...

Array

Un document peut contenir, dans un champ, un tableau d'éléments.

tableau

```
film = {titre:"Fargo",annee:1996,
        genre:["comedie","thriller"]}
```

Documents imbriqués

Un document peut contenir, dans un champ, un autre document.

document

```
film = {titre:"Jaws",annee:1975,genre:"drame",
        real:{nom:"Spielberg", prenom:"Steven"}}
```

Indexation

Les performances de la base de données dépend pour beaucoup d'une indexation correcte et judicieuse des données.
mongoDB gère plusieurs types d'index :

- simple
- multi-champs : compound index
- une sous-partie des données : Partial index
- géo-index : 2d, 2dsphere
- indexation de champ textuel : Text index
- hash-index (pour le partitionnement)

Administration

- Serveur : `mongod`
- Shell : `mongo`
- Base : use *nomdb*
  La base est "créée" en mémoire puis "persistée" lors de la première écriture.
  La variable `db` contient la bdd utilisée.
  Sélectionner une base : `use dbname`

Administration

- Collection :
  - implicite : `db.collec.insert(data)`
  - explicite : `db.createCollection(nomcoll,options)`
    options parmi taille définie, id non auto, règles de validations, etc.
- Outils :
  `help, db.help(), db.stats(),`
  `db.collec.stats(), ...`

Requètage

5 opérations détaillées :

- Insert
- Remove
- Select
- Update
- Upsert

Attention

Les jointures entre collections ne sont pas possibles (ou presque !, de manière simple...)

Insert / Delete

```
film1 = {titre:"Z",annee:1969,genre:"drame"}
film2 = {titre:"Jaws",annee:1975,genre:"drame"}
deuxfilms = [{titre:"Starwars 4",annee:1977,genre:"sci-fi"},
  {titre:"Starwars 5",annee:1980,genre:"sci-fi"}]
db.films.insertOne(film1)
db.films.insert(film2)
db.films.insertMany(deuxfilms)
db.films.deleteOne({annee:1977})
db.films.deleteMany({genre:"drame"})
db.films.remove({titre:"Z"})
db.films.remove({})
```

Insert / Delete

2 arguments :

- Un critère de sélection (peut être nul)
- Un document *WriteConcern* pour spécifier ou non d'attendre la réplication/journalisation

Select

En paramêtre, les critères de recherche, les colonnes souhaitées

```
db.films.find()
db.films.find({}, {titre:1, annee:1})
db.films.find({}, {_id:0})
db.films.findOne()
db.films.find({genre:"drame"})
db.films.find({genre:null})
db.films.find({titre:"Z",genre:"drame"})
```

Selection sur tableau

Possibilité de restreindre sur un tableau

- requête l'ensemble des valeurs
- requête le $(pos + 1)^e$ élément (*champtab.pos*)
- *$elemMatch* permet de mettre plusieurs critères à vérifier par chaque élément
- *$size* permet de contrainte la taille du tableau

Selection sur tableau

```
// le tableau contient l'element en parametre
db.films.find({genre:"drame"})

// le tableau contient le genre principal est comedie
db.films.find({"genre.0":"comedie"})

// le tableau matche exactement celui passe en parametre
db.films.find({genre:["drame","comedie"]})

// le tableau contient les elements
db.films.find({genre:{$all : ["drame","comedie"]}})

// les films de 2 genres
db.films.find({"genre":{$size : 2}})
```

Select

Version $>$ 3.2 : paramètre optionnel : *readConcern*

- local (par défaut) : retourne la version la plus récente
- majority : retourne la version la plus répliquée

```
db.films.find({titre:"Z",genre:"drame",
        readConcern: { level: "majority"}})
```

Select

Post-traitements

- Tri : `sort`
- Décompte : `count`
- Sous-ensembles : `limit`, `skip`

Select

```
// on classe par annee decroissante
db.films.find().sort({annee:-1})

// on compte le nb de films du siecle
db.films.count({annee:{$gte:2000}})

// on prend 2 enregistrements a partir du 2nd
db.films.find().skip(1).limit(2)
```

Curseurs

```
var cur = db.films.find();
cur.forEach( function(doc) { print(tojson(doc))})

var tab = cur.toArray();
print tab[2];
```

Update (simple)

2 arguments :

- Un critère de sélection (peut être nul)
- La valeur de remplacement

Attention

La valeur de remplacement remplace tout le document (voir exemples)

Update

```
db.films.updateOne({titre:"Jaws"},
    {titre:"Les dents de la mer"})

est different de

db.films.updateOne({titre:"Jaws"},
        {$set : {titre:"Les dents de la mer"}})
```

Update

Fonctions :

- updateOne()
- updateMany()
- replaceOne()

Update

Opérateurs :

- $set / $unset
- $inc
- $push / $pushAll
- $pull / $pullAll
- $addToSet
- $rename
- ...

Update

```
db.films.updateOne({titre:"Fargo"},
      {$pull : {genre:"comedie"}})

db.films.updateOne({titre:"Jaws"},
      {$addToSet:{genre:{$each:["horreur","Thriller"]}}})

db.films.updateOne({titre:"Z"},
      {$rename:{"real":"dir"}})

db.films.replaceOne({titre:"Jaws"},
    {titre:"Les dents de la mer",annee:1975,
     genre:"horreur",dir:"S. Spielberg"})

db.films.updateMany({},
      {$set : {dateSortie:new Date()}})

db.films.updateMany({},
      {$unset : {dateSortie:""}})
```

Update

La fonction update contient 2 paramêtres supplémentaires : `db.collec.update( critère, remplacement, upsert, multi)`

- : `upsert`, met à jour une donnée si elle existe, l'insère sinon
- : `multi`, répercute la mise à jour sur tous les documents vérifiant le critère

Upsert

L'instruction `save` permet aussi un upsert (lorsqu'il n'y a pas de restriction)
db.films.save(titre :"jeu d'enfant", annee :1988)

Upsert

```
db.films.update({titre:"Vanilla sky"},
      {titre:"Abre los ojos"}, {upsert:true})

db.films.update({genre:"drame"},
       {$inc:{duree:1000}}, false, {upsert:true})
```

Requêtes avancées

Opérateurs

- `$lt, $lte, $gt, $gte, $elemMatch`
- `$in, $nin`
- `$or, $and`
- `$exists`
- `$ne`
- ....

Requêtes avancées

```
db.films.distinct("annee",{annee:{$gt:1980}})

db.films.find({annee:{$lt:1970}})

db.films.find({genre:{$in:["drame", "comedie"]}})

db.films.find({$or:[{genre:"drame"}, {annee:1975}]})

db.films.find({nationalite:{$exists:true}})

db.films.find({"real.nom":"Spielberg"})
```

Requêtes avancées

Opérateur `$where`

- Permet de simplifier l'écriture d'une restriction
- S'écrit en JavaScript
- À n'utiliser qu'en dernier recours (n'utilise pas les index)

$where

```
db.films.find({$where:"this.annee>1960 && this.annee<1970"})
```

Requêtes avancées

Requêtes sur des champs texte

- Nécessite un index spécifique
- Opérateurs $text, $search
- Classer les résultats par pertinence $meta

Index

```
// index sur deux champs : titre et description
// cherchera des motifs sur l'ensemble de ces deux champs
db.films.createIndex({titre:"text", description:"text"})
```

Requêtes sur champ texte

```
// contient un de ces 3 mots
db.films.find({$text:{$search:"aventure tresor jungle"}})

// idem avec classement par pertinence
db.films.find({$text:{$search:"aventure tresor jungle"}},
        {score:{$meta:"textScore"}})

// contient un des 2 premiers et pas le 3e
db.films.find({$text:{$search:"aventure tresor -jungle"}})

// contient "aventure" ou bien le terme "jungle amazonienne"
db.films.find({$text:
       {$search:"aventure \"jungle amazonienne\""}})
```

Requêtes géo-localisées

- Ensemble d'opérations de requêtage géographiques
- Géré par l'index **2dsphere**
- Cas de coordonnées GPS 2D (latitude, longitude)
- Basé sur le format geoJSON
  `http://geojson.org/geojson-spec.html`
- Formes : Point, LineString, Polygon, centerSphere
- Opérations de base : proche, dans, coupe

Attention

Les coordonnées sont toujours données dans l'ordre suivant :

`longitude, latitude`

Objets géo-localisés

- Point : longitude/latitude
- Ligne brisée : tableau de longitude/latitude
- Polygone : tableau de longitude/latitude revenant au pt de départ
- Sphère : tableau de longitude/latitude pour le centre et un rayon

Objets géo-localisés

```
{$geometry: {type: "Point" ,
    coordinates: [-0.373856,43.295213]
   }}
{$geometry: {type:"LineString",
    coordinates:[
            [-0.397492,43.274087],[-0.354233,43.279461]
  ]}}
{$geometry: "type": "Polygon",
           "coordinates": [
             [ [100.0,0.0],[101.0,0.0],[101.0,1.0],
               [100.0,1.0],[100.0,0.0] ]
             ]
         }}
```

Index géo-localisés

- Index de géo-localisation : 2DSphere
- Pour les objets formalisés geoJSON
- Création :
  db.collection.createIndex({ champgeoloc : "2dsphere" })

Requêtes géo-localisées

- proche : geoNear
- amélioration de $near (déprécié)
- retourne les docs classés par distance croissante
- déprécié par l'aggrégation (voir ci-après)

Requête système : geoNear

```
db.runCommand({geoNear:"bar",
  near:{type: "Point" , coordinates: [-0.3670,43.3325]},
  spherical:true,
  maxDistance:200/6371000});
```

Requête geoNear : options

- near : point de départ
- minDistance, maxDistance : bornage
- limit, num : limite le nb de docs retournés
- query : doc pour faire une restriction
- ...

Requêtes géo-localisées

- dans : $geoWithin
- formes : cercles, polygone, rectangle
- retourne les docs entièrement contenus dans la zone définie

Requête within

```
db.bar.find(
{localisation:
    {$geoWithin :
        {$geometry :
        {$centerSphere : [ [-0.363138, 43.306637], 0.007]
    }}}},
{nom:1, _id:0} )
```

Requêtes géo-localisées

- croise : $geoIntersect
- formes : cercles, polygone, lignes, rectangles
- retourne les docs partiellement contenus dans la zone définie

Requête within

```
db.bus.find( {localisation :
    { $geoIntersects :{ $geometry :
                        {type : "Polygon" ,
                        coordinates : [ [[-0.374709, 43.297849],
                            [-0.366362, 43.299161],
                            [-0.367027, 43.294023],
                            [-0.374709, 43.297849]] ]}}}},
                {nom:1, _id:0} )
```

Aggrégations

mongodb permet de réaliser des fonctions de regroupement pour :

- dénombrer
- compter
- faire des moyennes
- ...

3 moyens

- group (déprécié)
- map-reduce (déprécié)
- aggregate, basé sur les pipelines linux

Aggrégation

- fonction aggregate
- opérateurs $group, $match, $sum, $avg, $max, $push...

Regroupement

```
// select genre, avg(duree) from Films group by genre;
db.films.aggregate([
 {$group: {_id:"$genre", total:{$avg:"$duree"}}}
]);

// select annee, count(idFilm) as nb from Films
// where annee>2000 group by annee;
db.films.aggregate([
 {$match:{ annee: {$gt:2000} }},
 {$group: {_id:"$annee", nb:{$sum:1}}}
]);
```

Regroupement

```
// select genre, avg(duree) from Films
//group by genre where annee > 2000;
db.films.aggregate([
 {$project:{ annee:1, genre:1, duree:1, _id:0}},
 {$match:{ annee: {$gt:2000} }},
 {$group: {_id:"$genre", nb:{$avg:$duree}}}
]);
```

Aggrégation et geoQuery

- fonction aggregate
- opérateur $geoNear

Regroupement

```
db.bar.aggregate([
{$geoNear: {
   near: { type: "Point",
           coordinates: [12.094556236,78.895275886] },
           distanceField: "dist",
           maxDistance: 42,
           spherical: true
   } } ])
```

Caractéristiques

- Système de stockage orienté document
- Opérations : Réplication, partitionnement horizontal et vertical
- Architecture de serveur Maître / Esclave
- Performance : utilisation de la RAM pour les données, mais dépend de sa capacité ou du nombre de réplications, vérifications (safe) effectuées
- Pas de transaction, pas de jointure (de manière simple...)
- Limitation de taille de document 16MB

D'autres notions à approfondir !

- Capped collections (taille limitée)
- Collation (règles syntaxiques)
- "Jointures" : *$lookup*
- Règles de structuration : Schema validation
- Écouteurs d'événements : Change streams
- Transactions explicites (v4.0)
- Administration Bdd (users, rôles, vues, etc.) :

  `https://docs.mongodb.com/manual/reference/command/`
- Compass (IDE) :

  `https://docs.mongodb.com/compass/current/`
- Un ancien eistien chez mongodb ! : @MBeugnet