mongodb

8 septembre 2014

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 requètes 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
  - Requètage (Requètes 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"}}
```

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`
- Collection : pas d'opération de création de collection (implicite : `db.collec.insert(data)`)
- 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

Insert / Delete

```
film = {titre:"Z",annee:1969,genre:"drame"}
db.films.insert(film)
film = {titre:"Jaws",annee:1975,genre:"drame"}
db.films.insert(film)
db.films.remove({titre:"Z"})
db.films.remove({})
```

Insert / Delete

2 arguments :

- Un critère de sélection (peut être nul)
- Un booléen d'avertissement d'erreur

Attention

L'écriture dans la bdd est de type "fire and forget" ;
spécifier si besoin, safe=true :
`db.films.insert(film, safe=True)`,
sinon `db.getLastError()`

Select

En paramêtre, les critères de recherche, les colonnes souhaitées

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

Select

Post-traitements

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

Select

```
// on classe par année décroissante
db.films.find().sort({annee:-1})
// on compte le nb de films du siècle
db.films.count({annee:{$gte:2000}})
// on prend 2 enregistrements à partir du 2nd
db.films.find().skip(1).limit(2)
```

Curseurs

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

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

Update

Opérateurs :

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

Update

```
db.films.update({titre:"Fargo"},
        {$pull : {genre:"comedie"}})
db.films.update({titre:"Jaws"},
        {$addToSet:{genre:{$each:["horreur","Thriller"]}}})
```

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

Requètes avancées

Opérateurs

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

Requètes 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"})
```

Requètes 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"})
```

Requètes avancées

Opération `findAndModify`

- Permet de lancer une requête de sélection, modification, suppression
- Retourne le document avant ou après modification (booléen *new*)
- Opérations remove, update, upsert

findAndModify

```
db.eleve.findAndModify({
    query: {classe:'ING1', passe:true},
    update: {$set: classe:'ING2'}},
    new: true,
    fields: {_id:-1}
});
```

Requètes géo-localisées

- Ensemble d'opérations de requétage géographiques
- Cas de coordonnées GPS 2D (latitude, longitude)
- Basé sur le format geoJSON
  `http://geojson.org/geojson-spec.html`
- 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

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.ensureIndex({ champgeoloc : "2dsphere" })

Requètes géo-localisées

- Proche : geoNear
- Amélioration de $near
- retourne les docs classés par distance

Requète geoNear

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

Requète geoNear : options

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

Requètes géo-localisées

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

Requète within

```
db.bar.find( { geometry :
    { $geoWithin :{ $geometry :
     { $centerSphere : [ [ -0.363138, 43.306637 ] , 0.007 ] } } } }
```

Requètes géo-localisées

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

Requète within

```
db.bus.find( { geometry :
    { $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
- ...

Aggrégation

- fonction `group`
- 5 arguments :
  - `key` : champs du regroupement
  - `reduce` : fonction de regroupement
  - `initial` : valeur initiale de la variable d'aggrégation
  - `cond` : restriction sur les documents
  - `finalize` : fonction appelée, pour chaque aggrégation, avant le retour du résultat

Regroupement

```
// select annee, count(idFilm) from Films group by annee;
db.films.group(
  {key: { annee:true },
   cond: null,
   reduce: function(doc,res) { res.csum += 1; },
   initial: { csum: 0 }
  });
```

Regroupement

```
// select genre, avg(duree) from Films group by genre;
db.films.group(
  {key: {genre:true},
   cond: null,
   reduce: function(doc,res){res.csum += doc.duree;res.cpt+=1;},
   initial: {csum:0,cpt:0},
   finalize: function(res){res.avg = res.csum / res.cpt;}
  });
```

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 garantie de durabilité (fire & forget)
- Pas de transaction, pas de jointure
- Limitation de taille de document 16MB

D'autres notions à approfondir !

- Capped collections
- Recherche géo-spatiale
- Map-Reduce
- ...

mongo et PHP

Installations :

`http://php.net/manual/fr/book.mongo.php`

Connexion

```
<?php
   $mongo = new MongoClient(url);
   $db = $mongo->selectDB($mabdd);
   $collec = $db->selectCollection($macollec);
?>
```

Connexion

url : adresse du serveur,

exemple, en local : mongodb ://127.0.0.1 :27017/

Connexion avec authentification

```
<?php
   $mongo = new MongoClient(url);
   $db = $mongo->mabdd;
   $db->authenticate($login, $pass);

ou bien, dans l'url :
   $mongo = new Mongo("mongodb://${login}:${pass}@localhost");

?>
```

Écritures

- $collec->insert($doc) ;
- $collec->insert($doc, array(safe=>true) ;
- $collec->update($crit, $nvdoc) ;
- $collec->update($crit, $nvdoc, array(upsert=>true)) ;
- $collec->update($crit,
  array('$set'=>array(champ=>val),
  array(multiple=>true))) ;
- $collec->remove($crit, array(safe=>true)) ;

Attention

"Tout" est tableau ! Dans une requète mongo utilisant la valeur *null*, un tableau vide lui correspond :

```
$collec->update(array(), array('$inc' => array(duree=>10)));
```

Insertion

```
<?php
try {
  $doc = array(
    titre => "La classe américaine",
    annee => 1993,
    duree => 70,
    genre => array("comedie","western", "pirate","aventure")
 );

 $collec->insert( $doc, array(safe=>true));
} catch (Exception $ex) {
 echo $ex->getMessage();
}
?>
```

Sélections

- $collec->find() ;
- $collec->findOne($crit) ;
- $collec->find($crit)->sort(array(champ=>1)) ;
- $collec->find($crit)->skip(10)->limit(10) ;

Curseurs

```
<?php
 $cur = $collec->find(array(), array(_id=>false));
 foreach ($cur as $rec) {
  foreach ($rec as $k=>$v) {
    echo "- $k : $v -<br/>"
  }
 }
?>
```

Opérateurs avancés de sélection

- $lt, $lte,..
- $ne, $exists
- $or, $and, $not
- $in, $nin
- $elemMatch
- ...

Opérateurs avancés de mise à jour

- $set, $unset
- $inc
- $push(All), $pull(All), $addToSet
- $remane
- ...

Distinct

retourne un document

```
<?php
 $champ = annee;
 $crit = array(annee=>array('$exists'=>true));

 $res = $collec->distinct($champ, $crit);
 echo json_encode($res);
?>
```

Aggrégation

fonction `group`, 5 arguments, retourne un document.

- `key` : attributs de regroupement
- `initial` : valeur initiale du paramêtre de la fonction d'aggrégation
- `reduce` : fonction d'aggrégation
- `condition` : restriction sur les documents à considérer
- `finalize` : code exécuté avant retour de valeur

Aggrégation

```
<?php
 // fonction créant, par année de sortie,
 // un tableau de films sortis cette année là
 $keys = array(annee=>1);
 $initial = array(tfilms=>array());
 $reduce = 'function(doc,res) {res.tfilms.push(doc.titre);}';
 $crit = array(annee=>array('$exists'=>true));

 $res = $collec->group ($keys, $initial, $reduce, $crit);

 echo json_encode($res['retval']);
?>
```

Attention

- N'oubliez pas les fonctions PHP `json_encode` et `json_decode`
- Attention aux opérateurs type $set et PHP, écrire '$set'
- Attention aux injections de code
- `www.php.net/manual/fr/book.mongo.php`
- `code.google.com/p/rock-php/wiki/rock_mongo`
- `genghisapp.com/`