

Bases de données NoSQL



P.Mathieu

LP DA2I Lille

<http://www.iut-a.univ-lille.fr>

prenom.nom@univ-lille.fr

8 septembre 2019

1 NoSQL

2 MongoDB

NoSQL (Not Only SQL)

- De plus en plus d'applications créent des volumes massifs de données
- Les structures doivent évoluer rapidement
- Les données sont souvent semi-structurées voir non structurées
- Faciliter le lien avec la programmation orientée objets

Différents types de SGBD

- Orientés documents
chaque clé est associée à une structure complexe appelée Document (MongoDB, CouchDB)
- Orientés Graphes
adaptées au stockage de graphes sociaux (Neo4J, Giraph)
- Orientés Clé-Valeurs
Chaque information est stockée selon un modèle clé-valeur (Redis, Berkeley DB, DynamoDB)
- Orientés colonnes
Spécialement optimisées pour de larges ensembles de données (Cassandra, HBase)

JSON (JavaScript Object Notation) est un format de données textuelles dérivé de la notation des objets du langage JavaScript.

Utilisé principalement pour l'échange de données entre applications (web-services)

Un document JSON comprend deux types d'éléments structurels :

- des ensembles de paires `clé:valeur` , entre accolades `{ }`
- des listes ordonnées de valeurs, entre crochets `[]`

Types de données : chaînes (entre guillemets), nombres, booléens (`true`, `false`), `null`, objet `Json`

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user: 'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user: 'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

Seule contrainte : pas 2 fois la même clé dans un même document

1 NoSQL

2 MongoDB

- Base de données NoSQL **orientée Documents**, open-source, écrite en C++
- Mode Client Serveur
- Téléchargeable sur [mongodb.org](https://www.mongodb.org)
- Manuel sur <https://docs.mongodb.com/manual/>
- Données (document) représentées au format BSON
- Langage d'interrogation "maison"
- Pas de système transactionnel ni d'intégrité référentielle
- Case sensitive

`{"foo" : 3}` est différent de `{"foo" : "3"}`

`{"foo" : 3}` est différent de `{"Foo" : 3}`

RDBMS	MongoDB
Database	Database
Table	Collection
Ligne	Document
colonne	Field
Jointure	<code>\$lookup</code> , ou Embedded Documents
Primary Key	<code>_id</code>

```
{nom: "paul", naiss: 1989, ville: Lille,  
  tel : {fixe: 0311111111 port:0611111111}} email:["paul@gmail.com","paul@ovh.com"]}  
{nom: "pierre", naiss: 1997, ville: Lille,  
  tel : {fixe: "0311111111", port:"0611111111"}, email:["pierre@gmail.com"]}  
{nom: "jean", naiss: 1999, ville: Lille,  
  tel : {fixe: "0355555555", port:"0655555555"}, email:["jean@gmail.com"]}  
{nom: "lucie", naiss: 1998, ville: Lille,  
  tel : {fixe: "0333333333", port:"0633333333"}, email:["lucie@gmail.com"]}  
{nom: "lucas", naiss: 2000, ville: Lille,  
  tel : {fixe: "0399999999", port:"0699999999"}, email:["lucas@gmail.com"]}  
{nom: "iris", naiss: 1999 , ville: Lille,  
  tel : {fixe: "0322222222", port:"0622222222"}, email:["iris@gmail.com"]}  
{nom: "elsa", naiss: 2002, ville: Lille,  
  tel : {fixe: "0377777777", port:"0677777777"}, email:["elsa@gmail.com","elsa@yahoo.fr"]}  
{nom: "farid", naiss: 2002, ville: Lille,  
  tel : {fixe: 0344444444 port:0644444444}, email:["farid@yahoo.fr"]}
```

- L'objectif d'un SGBDR est d'éviter toute redondance et de faciliter aussi bien la lecture que la mise à jour (donc MCD, Tables et intégrité référentielle)
- L'objectif qu'un SGBD NoSQL est d'éviter toute opération coûteuse (jointure), et de faciliter les recherches
- Un SGBDR nécessite un schéma (éventuellement modifiable)
- Un SGDB NoSQL ne possède aucun schéma. Une même collection peut contenir des documents de schémas complètement différents

les avantages des uns sont les inconvénients des autres !

- Download : <https://www.mongodb.org/downloads>

- Serveur :

```
mongod --dbpath /tmp  
mongod --config /usr/local/etc/mongod.conf
```

- Fichiers :

```
configuration file (/usr/local/etc/mongod.conf)  
log directory path (/usr/local/var/log/mongodb)  
data directory path (/usr/local/var/mongodb)
```

- port utilisé par défaut : 27017

- Client :

```
mongo -v  
mongo --quiet
```

- Par défaut 3 bases sont déjà créées (`admin`, `config`, `local`) et on se trouve dans une base `test` vide

```
show dbs
db
```

- Changer de base : `use mabase`
- Le shell est un interpréteur Javascript complet
- Au lancement, exécution du fichier `/.mongorc.js`
- par défaut mono-utilisateur et uniquement connexions locales.

les commandes générales sont préfixées par `db`. Celles d'une collection `matable` sont préfixées par `db.matable`.

```
use mabase
db.createCollection("matable")
show collections
db.matable.insert({nom:'dupond' , prenom:'paul' , age:15})
db.matable.insert({nom:'durand' , prenom:'paul' , age :25})
db.matable.find()
db.matable.find({'prenom':'paul'}, {'nom':1})
db.matable.count()
db.matable.update({nom:'durand'}, {$set:{prenom:"lucie"}})
db.matable.remove({nom:'durand'})
db.matable.remove({})
db.matable.drop()
quit()
```

```
db.matable.find({prenom: 'paul'})  
db.matable.find({prenom:'paul'}, {nom:1})    (uniquement le nom)  
db.matable.find({prenom:'paul'}, {nom:0})    (tout sauf le nom)  
db.matable.find({age: {$gt:18}})            (lt, lte, gt, gte,  
  
db.matable.find({$or: [{key1: value1}, {key2:value2}]})
```

Le \$and est équivalent à une virgule

find peut aussi être accolé à .sort({KEY:1}) ou .limit(n) ou .skip(n) ou .pretty() ou milli

`aggregate` est l'opération principale pour les interrogations complexes

SQL op	Aggregate op
where	<code>\$match</code>
group by	<code>\$group</code>
having	<code>\$match</code>
select	<code>\$project</code>
order by	<code>\$sort</code>
limit	<code>\$limit</code>
sum	<code>\$sum</code>
count	<code>\$count</code>

Voir

<https://docs.mongodb.com/manual/reference/sql-aggregation-comparison>


```
db.matable.aggregate([
  { $match: { nom: "durand" } },
  { $group: { _id: "$prenom", total: { $sum: "$age" } } },
  { $sort: { total: -1 } }
])
```

Opérations d'agrégation : \$sum, \$avg, \$min, \$max, \$first, \$last

```
db.matable.aggregate([{$group: {_id: "$nom", result: {$sum : 1 }}}])
db.matable.aggregate([{$group: {_id: "$nom", result: {$sum : "$age" }}}])
```

Utiliser `db.matable.explain().aggregate` pour avoir des détails sur l'exécution

En mongo il y a 2 méthodes pour créer une relation

- par imbrication
- par référence (avec `aggregate` et `lookup` pour la jointure)

```
db.etudiants.aggregate(  
  [ {$lookup: {from: "notes" , localField: "_id" ,  
                foreignField: "etu_id" , as : "notes"} } ]
```

Comme on est dans un interpréteur Javascript, il est très facile de générer des millions de documents !

```
db.createCollection("users")

for (i=0; i<1000000; i++) {
  db.users.insert({ "i" : i,
                    "username" : "user"+i,
                    "age" : Math.floor(Math.random()*120),
                    "created" : new Date()
                  } );
}

db.stats()
db.users.find({username: "user101101"})
db.users.ensureIndex({ "username" : 1 })
```

- MapReduce est un patron de conception de requetes qui consiste à partitionner les données pour ensuite les traiter en parallèle
- Patron notamment mis en place par Google dans son framework Hadoop
- MapReduce s'appuie principalement sur deux fonctions : `map` qui associe des données à des clés et `reduce` qui collecte les données d'une clé particulière pour effectuer les calculs

mySQL

```
SELECT
  Dim1, Dim2,
  SUM(Measure1) AS MSum,
  COUNT(*) AS RecordCount,
  AVG(Measure2) AS MAvg,
  MIN(Measure1) AS MMin,
  MAX(CASE
    WHEN Measure2 < 100
    THEN Measure2
  END) AS MMax
FROM DenormAggTable
WHERE (Filter1 IN ('A','B'))
   AND (Filter2 = 'C')
   AND (Filter3 > 123)
GROUP BY Dim1, Dim2
HAVING (MMin > 0)
ORDER BY RecordCount DESC
LIMIT 4, 8
```

- ① Grouped dimension columns are pulled out as keys in the map function, reducing the size of the working set.
- ② Measures must be manually aggregated.
- ③ Aggregates depending on record counts must wait until finalization.
- ④ Measures can use procedural logic.
- ⑤ Filters have an ORM/ActiveRecord-looking style.
- ⑥ Aggregate filtering must be applied to the result set, not in the map/reduce.
- ⑦ Ascending: 1; Descending: -1

MongoDB

```
db.runCommand({
  mapreduce: "DenormAggCollection",
  query: {
    filter1: { '$in': [ 'A', 'B' ] },
    filter2: 'C',
    filter3: { '$gt': 123 }
  },
  map: function() { emit(
    { d1: this.Dim1, d2: this.Dim2 },
    { msum: this.measure1, recs: 1, mmin: this.measure1,
      mmax: this.measure2 < 100 ? this.measure2 : 0 }
  ); },
  reduce: function(key, vals) {
    var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };
    for(var i = 0; i < vals.length; i++) {
      ret.msum += vals[i].msum;
      ret.recs += vals[i].recs;
      if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;
      if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))
        ret.mmax = vals[i].mmax;
    }
    return ret;
  },
  finalize: function(key, val) {
    val.mavg = val.msum / val.recs;
    return val;
  },
  out: 'result1',
  verbose: true
});
db.result1.find({ mmin: { '$gt': 0 } }).
sort({ recs: -1 }).
skip(4).
limit(8);
```

Revision 4, Created 2010-03-06
Rick Osborne, rick@osborne.org

```
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;

import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class Test {

    public static void main( String args[] ) {

        // Creating a Mongo client
        MongoClient mongo = new MongoClient( "localhost" , 27017 );

        // Creating Credentials
        MongoCredential credential;
        credential = MongoCredential.createCredential("sampleUser", "myDb",
            "password".toCharArray());
        System.out.println("Connected to the database successfully");

        // Accessing the database
        MongoDatabase database = mongo.getDatabase("myDb");

        // Creating a collection
        database.createCollection("sampleCollection");
        System.out.println("Collection created successfully");

        // Retrieving a collection
        MongoCollection<Document> collection = database.getCollection("myCollection");
        System.out.println("Collection myCollection selected successfully");
    }
}
```

Par défaut Mongo est accessible sans authentification, sans utilisateur et uniquement de la machine locale. Aucun utilisateur n'est créé.

Pour que l'authentification soit nécessaire, lancer `mongod` avec `-auth` ou mettre dans le fichier de config `mongod.conf`

```
security:  
  authorization: enabled
```

Les utilisateurs doivent ensuite être créés dans la ou les bases concernées

MongoDB

Gestion des utilisateurs

```
use admin
db.createUser( {user: "phil", pwd: "phil", roles: [{"role:r
db.getUsers()      (ou show users)
db.changeUserPassword("username", "newPass")
db.revokeRolesFromUser("phil", [{"role:"dbAdmin", db:"test"}])
db.grantRolesToUser("phil", ["dbAdmin"])

db.dropUser("phil")
show roles
```

Si un utilisateur est ajouté à la base `admin`, il hérite automatiquement de ses permissions pour toutes les bases

Deux manières différentes d'activer l'authentification

- En ligne de commande

```
mongo --authenticationDatabase "admin" -u "phil" -p  
show collections
```

- Une fois connecté

```
mongo  
use admin  
db.auth("phil", "phil")  
use test  
show collections
```

Les adresses ip autorisées doivent être déclarées dans le
`mongod.conf`

```
net:  
  bindIp: 127.0.0.1 , 0.0.0.0
```

L'accès se fait par

```
mongo 134.206.153.87 --authenticationDatabase "admin"
```

- mongodump (dump en bson, binaire !)

- mongorestore

- mongoimport

```
mongoimport -d mabase -c matable --type csv --file fichier.csv --head
```

- mongoexport

```
mongoexport --db music --collection artists --out /data/dump/fichier.
```