Les bases de données NoSQL basées sur JSON

L. KERKENI

IUT de Laval - Dpt. MMI

2019-2020

Plan

Introduction

Pourquoi utiliser une base de données JSON plutôt que relationnelle ?

1. JSON est un format très répandu.
2. Il n'y a pas de structure imposée.
3. On stocke des documents regroupant des informations.
4. Les BDD orientées documents sont prévues pour une gestion de données massives.

Plan

MongoDB

- Qu'est-ce que MongoDB ?
  - Une base de données orientée documents
  - Format interne en binaire JSON (BSON)
  - Une base de données massives distribuée
  - Une base de données flexible (i.e. pas de schéma)
  - Une base de données efficace en Big Data
  - Une base de données gratuite et open source

MongoDB vs eXist

- 2 bases de données orientées document
- Des formats internes différents (JSON vs XML)
- Des architectures différentes (Distribuée vs Unique)
- Des limites différentes :
  - eXist : 2^{34} documents maximum
  - MongoDB : 1 document pèse 16MB maximum
- Donc une modélisation des données différentes
- MongoDB plus “à la mode” qu’eXist en raison de l’explosion du format JSON.

La structuration de MongoDB

- Une instance de MongoDB peut contenir plusieurs serveurs distribués
- Un serveur MongoDB peut contenir plusieurs bases de données
- Une base de données peut contenir plusieurs collections
- Une collection peut contenir plusieurs documents
- Un document contient plusieurs données (limite de 16MB) et un identifiant unique généré par MongoDB

Modélisation

- Selon l'usage que vous souhaitez faire de la base de données
  - Soit vous visez la rapidité et vous acceptez la redondance d'informations
  - Soit vous êtes intransigeants sur la conception et vous acceptez des réponses plus lentes
- Il faut aussi se projeter et concevoir vos collections/sous-collections en fonction de vos documents et de leurs tailles.

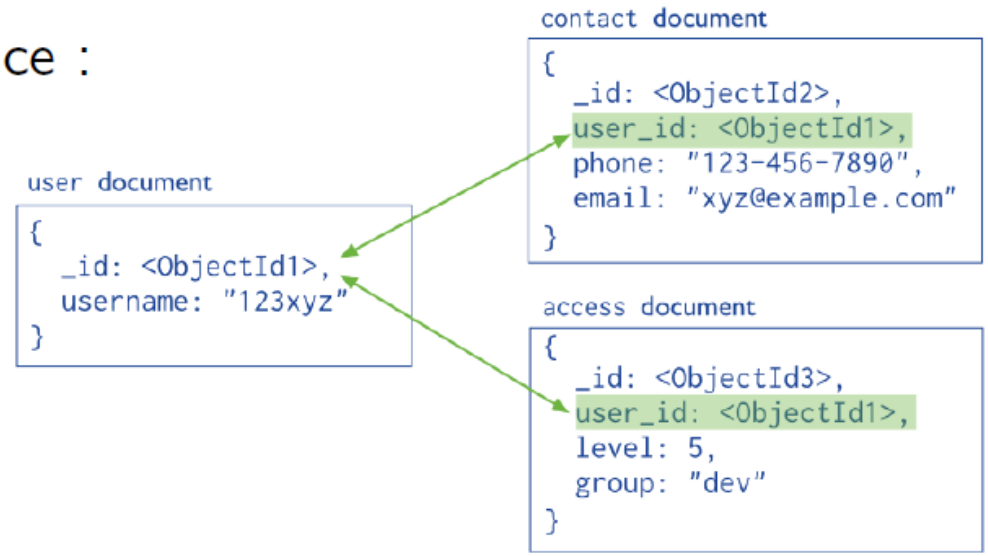
Modélisation : Relations inter-documents

Relation entre les documents de différentes collections :

- Par référence : l'identifiant d'un document (son "_id") est utilisé comme valeur attributaire dans un autre document
  - se rapproche du modèle de données normalisées
  - nécessite des requêtes supplémentaires côté applicatif
- Par inclusion ("embedded") : un "sous-document" est utilisé comme valeur
  - philosophie "non-relationnelle" (pas de jointure)
  - meilleures performances en lecture
  - redondance possible

Modélisation : Relations inter-documents

Relation par référence :



Relation par inclusion (embedded) :

Modélisation d'une BD

Considérations pour modéliser une BD au niveau physique :

- Accès par une clé (identifiant d'un document)
- Schéma optionnel $\Rightarrow$ ajout d'un champ à tout moment
- Pas de jointure $\Rightarrow$ redondances possibles
- Inclusion $\Rightarrow$ moins de lectures
- Opérations atomiques sur un seul document, mais pas sur plusieurs $\Rightarrow$ état incohérent temporaire (souvent tolérable)

Modélisation d'une BD - recommandations

Relation par **inclusion** pour :

- les entités fréquemment lues ensemble (article/commentaires)
- une entité dépendante d'une autre (facture/client)
- des données mises à jour en même temps (nombre d'exemplaires d'un livre et informations sur les emprunteuses)
- besoin de rechercher des documents via un index multi-clés (catégories d'un livre)

Relation par **référence** pour :

- éviter une forte redondance sans "gain" (livre/éditeur)
- des entités fortement connectées "many to many" (livres/auteurs)
- des données hiérarchiques (catégories/sous-catégories/)

Les outils fournis avec MongoDB

- Vous trouverez parmi les fichiers du dossier bin :
  - mongod : Le serveur MongoDB
  - mongo : Le shell MongoDB (avec le langage Javascript)
  - mongodump : Pour sauvegarder (binaire) vos bases de données
  - mongorestore : Pour restaurer (binaire) vos bases de données
  - mongoimport : Pour importer un fichier JSON ou CSV
  - mongoexport : Pour exporter un fichier JSON ou CSV
  - bsondump : Conversion du BSON en JSON

Manipulation en shell (mongo)

- Se placer dans une base de données et la crée si elle n'existe pas : `use` *uneBase*
- Demander l'aide : `help`
- Quitter : `exit`
- Effacer la base dans laquelle on est : `db.dropDatabase()`
- Créer une collection :
  - Avec la commande de création : `db.createCollection(etudiants, {`*options*`})`
  - Avec une insertion : `db.etudiants.insert({...})`
- Afficher les collections : `show collections`
- Vider une collection : `db.maCollection.remove()`
- Supprimer une collection : `db.maCollection.drop()`

Interrogation

- Utilisation de l'opérateur : `find(`*critere*`[,` *projection*`])`
- Rechercher un document :
  - `db.maCollection.find({“nom”: “Dupont”})`
  - `db.maCollection.find({“age”: {$gte: 18}})`
- Rechercher une partie d'un document (projection) :
  - `db.maCollection.find({“age”: {$gte: 18}}, {“nom”: 1})`
  - Dans la projection, la valeur `0` ou `null` signifie exclure, les autres valeurs signifient inclure dans le résultat.
  - ATTENTION : La réponse retournée est toujours un document JSON contenant l'identifiant du document + le résultat de la projection.
- L'utilisation de la fonction `pretty()` permet un affichage plus “friendly”.

Interrogation (2)

- Définir une pagination avec `skip(x)` pour exclure les *x* premiers documents et `limit(y)` pour prendre en compte uniquement les *y* suivants.
  - `db.maCollection.find({“age”: {$gte: 18}}).skip(4).limit(10)`
- Si on souhaite au plus un document en réponse, utilisation de `findOne` au lieu de `find`
- Quelques opérateurs :
  - `$eq, $gt, $gte, $lt, $lte, $ne`
  - `$in, $nin`
  - `$and, $or, $not, $nor`
  - `$exists` qui prend pour valeur `true` ou `false`
  - ainsi que d’autres opérateurs d’évaluation, expressions régulières, sur des données géolocalisées . . .

Interrogation (3)

- Tri d'un résultat :
  - `db.maCollection.find().sort({"nom": 1})`
  - Tri sur le nom ascendant (1), pour l'ordre descendant : -1
  - Nécessite de récupérer tous les résultats avant de trier donc peut prendre plus de temps.
- Compter les résultats : `count()`

Mise à jour

- Utilisation de l'opérateur `update(`*query*`,` *update*`[,` *options*`])`
- Ecraser un document existant en conservant l'identifiant :
  - `db.maCollection.update({“nom”: “Dupont”}, {“date”: 2018})`
- Modifier une valeur d'un document :
  - `db.maCollection.update({“nom”: “Dupont”}, {$set: {“date”: 2018}})`
  - `$set` : modifie ou ajoute une valeur
  - `$unset` : retire la clé
  - `$pull` : retire une valeur d'un tableau
  - `$push` : ajoute une valeur dans un tableau
  - `{$pop: {tab: 1}}` : retire le dernier (1) ou le premier (-1) élément du tableau `tab`

Mise à jour (2)

- Les options :
  - `{multi:true}` : s'applique à tous les documents correspondant à la requête `query`
  - `{upsert:true}` : ajoute un document répondant à la requête `query` s'il n'y a pas de réponse

MongoDB et Javascript

- Le shell supporte le langage Javascript donc il est possible d'exploiter du code.
- Création d'une variable :
  - `col = db.maCollection` qui permet de faire directement `col.find(...)`
  - `dupont = db.maCollection.find({“nom”: “Dupont”})` pour stocker le document ayant pour nom “Dupont”
- Faire une jointure :

```
var liste = db.maCollection.find()
while (liste.hasNext()) {
var livre = liste.next(); var auteur = db.auteurs.findOne({“_id”:
livre.auteur._id});
printjson(livre.titre);
printjson(auteur);
}
```

Le principe du MapReduce

- Il s'agit d'appliquer à un ensemble de documents un traitement (map) et enfin d'appliquer une fonction d'agrégation (reduce) sur le résultat.
- On utilise Javascript pour créer ces fonctions.
- On transmet le traitement au moteur de MongoDB :
  - `db.maCollection.mapReduce(fonctionMap, fonctionReduce,` *sortie* `[,` *query*`])`
  - La sortie est par exemple l'écran : `{out: {"inline": 1}`

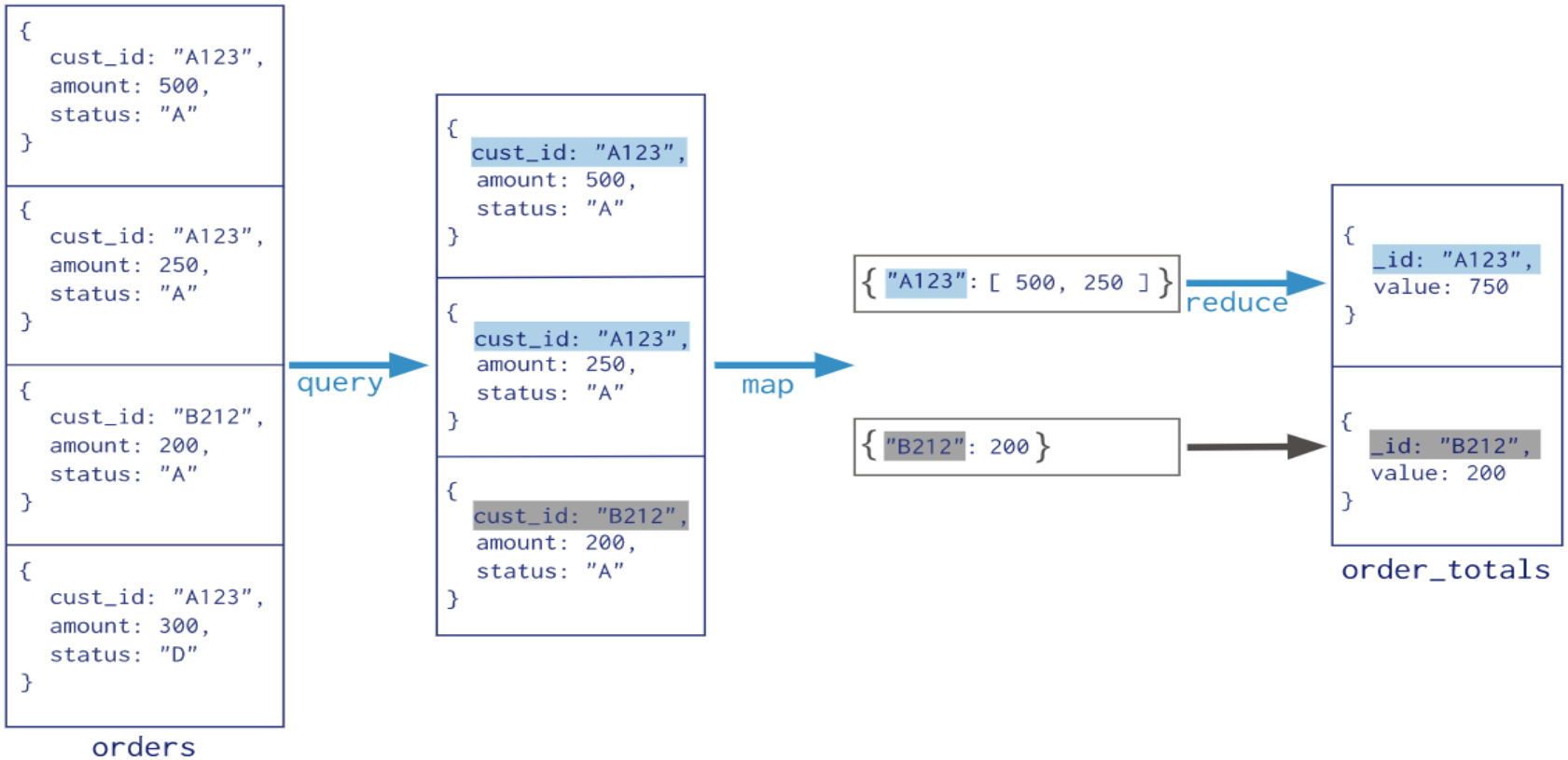
Exemple 1

```
var fonctionMap = function() {
        emit(this.etudiant._id, this.cours);
};
var fonctionReduce = function(etudiantID, cours) {
        var resultat = new Object();
        resultat.etudiant = etudiantID;
        resultat.enseignements = cours;
        return res;
};
```

Exemple 2

Collection

```
db.orders.mapReduce(
          map      →   function() { emit( this.cust_id, this.amount ); },
          reduce   →   function(key, values) { return Array.sum( values ) },
                       {
          output   →     out: "order_totals"
                       }
                   )
```



Manipulation en Java

- MongoDB fournit un driver spécifique pour Java.
- Attention, ce qui suit concerne les versions récentes du driver !!!
- Se connecter au serveur MongoDB :
  - `MongoClient mgClient = new MongoClient();`
  - `MongoClient mgClient = new MongoClient("serveur");`
  - `MongoClient mgClient = new MongoClient("serveur", port);`
- Fermer la connexion :
  - `mgclient.close();`

Connexion sécurisée

- MongoDB permet une connexion en SSL avec identifiants

```
String util;
String bdd;
char[] mdp;
MongoCredential authentification =
                        MongoCredential.createCredential(util, bdd, mdp);
MongoClientOptions options =
                        MongoClientOptions.builder().sslEnabled(true).build();
MongoClient mgClient = new MongoClient(new ServerAddress("serveur", port),
                        Arrays.asList(authentification),
                        options);
```

Accéder aux données

- Charger une base de données :
  - `MongoDatabase db = mgClient.getDatabase(“maBase”);`
- Créer une collection :
  - `db.createCollection(“nom”, instanceDeCreateCollectionOptions);`
  - Création automatique par MongoDB lors du stockage de documents.
- Charger une collection :
  - `MongoCollection<Document> coll = db.getCollection(“maCollection”);`

Interroger les données

- Utilisation de la fonction `find()`
  - Sans paramètres => récupération de tous les documents de la collection
  - Utilisation d'un filtre de document en paramètre
    - Tous les documents de la collection : `coll.find(new Document())`
    - `coll.find(new Document("age", new Document("$gte", 18)))`
    - `coll.find(new Document("age", new Document("$gte", 18).append("$lt", 30)))`
  - Utilisation des méthodes "helpers" de filtre
    - `coll.find(and(gte("age", 18), lt("age", 30)));`
- Le résultat est de type `FindIterable<Document>`

Opérations sur le résultat

- La projection
  - `coll.find().projection(new Document("nom", 1).append("ville", 1))`
  - `coll.find().projection(fields("nom", "ville"))`
- Le tri
  - `coll.find().sort(Sorts.ascending("nom"))`
  - `coll.find().sort(Sorts.descending("ville", "nom"))`
  - `coll.find().sort(Sorts.orderBy(ascending("ville"), descending("nom")))`
- Le premier : `coll.find().first()`
- La boucle :
  - `coll.find().forEach(`*Block*`)`
  - où *Block* est une liste d'instruction à appliquer sur chaque élément.

La création d'un *Block*

- Création d'une classe de type *Block*
- Redéfinition de la fonction `apply`

```
Block<Document> afficherTitre = new Block<Document>() {
          public void apply(final Document document) {
                 String val = (String)document.get("titre");
                 System.out.println("titre = " + val);
          }
};
```

Manipulation en PHP

- Connexion et requête simple

```
<?php
// connexion
$m = new MongoDB\Driver\Manager();
// Applique un filtre sur la collection
$filter = ['domaine' => 'informatique'];
$query = new MongoDB\Driver\Query($filter);
$cursor = $m->executeQuery('maBase.maCollection', $query);
// traverse les résultats
foreach ($cursor as $document) {
        $doc = (array)$document;
        echo 'titre = '.$doc['titre']."\n";
}
?>
```

Manipulation en PHP (2)

- Connexion et requête avec filtre et projection

```
<?php
// connexion
$m = new MongoDB\Driver\Manager();
// Applique un filtre sur la collection
$filter = ['domaine' => 'informatique'];
// Applique une projection sur la résultat
$proj = ['titre' => 1];
$query = new MongoDB\Driver\Query($filter, $proj);
$cursor = $m->executeQuery('maBase.maCollection', $query);
// traverse les résultats
foreach ($cursor as $document) {
       $doc = (array)$document;
       echo 'titre = '.$doc['titre']."\n";
}
?>
```

Plan

Références

- MongoDB : https ://www.mongodb.com
- Documentation et tutoriels : https ://docs.mongodb.com/
- Vidéos tutorielles :
  http ://mrbool.com/course/introduction-to-mongodb/323
- Drivers :
  - JAVA : https ://mongodb.github.io/mongo-java-driver/