Introduction

Dans un système de base de données relationnelles, les informations sont stockées par ligne dans des tables. Ces tables sont mises en relation en utilisants des clés primaires et étrangères.

Par exemple, prenons une base de données qui contient des *Acteurs* et des *Films* :

Table Acteur:

id	**nom**	**prenom**
1	DENIRO	Robert
2	STONE	Sharon

Table **Film**:

id	**titre**
1	Les affranchis
2	Casino

Table de jointure **Acteur_Film** qui permet d'associer les acteurs qui ont joué un rôle dans un film:

film_id	**acteur_id**
1	1
2	1
2	2

L'information est rangée dans des tables et si l'on souhaite récupérer le casting d'un film, il faut faire une requête SQL avec des jointures pour récupérer les bonnes lignes des tables.

Dans Mongodb. l'information est modélisée sur un document au format JSON (Javascript Object Notation). Je détaillerai dans la suite du cours ce format (vous en avez eu un aperçu le cours précédent).

Les documents JSON ont ce format :

{_id: "Casino", acteurs : [{nom:"STONE", prenom:"Sharon"}, {nom:"DENIRO", prenom:"Robert"}]}

{_id: "Les affranchis", acteurs : [nom:"DENIRO", prenom:"Robert"}]}

Fini les jointures, les documents contient tout ce qu'il faut !

Dénormaliser le schéma de la base de données pour favoriser les performances à la lecture. Les requêtes seront faites pour décider du format des documents.

Ce format se base sur 2 types d'éléments:

- la paire clé/valeur, par exemple "nom": "STONE"
- le tableau, par exemple "Agrumes" : ["orange", "citron", "mandarine", "clémentine"]

Un objet contient donc des paires clés/valeurs et ou des tableaux séparés par des virgules.

{"nom": "STONE", "prenom" : "Sharon",

"parfums préférés" : ["chanel5", "Smalto"]}

Sachant que la valeur d'une paire clé/valeur peut être un objet et que le tableau peut contenir des objets, on peut ordonner les informations sur plusieurs niveaux:

{"acteur": { "nom": "DENIRO", "prenom" : "Robert" },

"roles" : [

{"personnage" : "mafieux", "film" : "les affranchis"},

{"personnage" : "chauffeur de taxi","film" : "Taxi drivers"}

]

}

Les documents ne sont pas directement stockés dans la base de données en JSON mais dans un format binaire appelé BSON.

Les formats des donnés possible sont des chaînes de caractère, des booléens, des nombres, des dates, null…

mongorestore

Il vous arrivera souvent de devoir importer des données , voir une base entière afin de la traiter tranquillement sans interférer avec la base en production.

Donc vous intégrerez un dump, exemple pour restaurer une seule base de données et une seule collection il vous faudra utiliser la commande *mongorestore.*

1) créer un dossier films et copier le fichier bson

La commande à exécuter pour restaurer notre jeux de données est la suivante:

Dans une autre fenêtre de commande windows:

chemin_mongodb/bin/mongorestore.exe chemin/films/dump.bson

sous Linux on lance mongod puis dans une fenêtre de commande
mongorestore chemin/films/dump.bson

Utilisation

Après installation, lancez le démon ''mongod'' via une invite de commande.

Puis dans une autre invite, tapez ''mongo'' pour lancer l'interpréteur de commande.

Après l'exécution de la commande mongorestore, on aura une base de données films et une collection movies.

Commande à exécuter pour se connecter à la base:

use films

Commande à exécuter pour voir les collections de la base:

show collections

Ajout d'un document

Il nous faut en premier lieu une base de données, appelons la ''Videotheque''.

Pour créer cette base de données, il suffit de dire à MongoDb que l'on souhaite l'utiliser et comme elle n'existe pas, il va la créer tout seul:

use Videotheque

Pour voir toutes les base de données existant sur le système il suffit de taper :

show dbs

Dans une base de données relationnelle on crée des tables pour y mettre nos données. Dans mongodb, notre base de données contient des collections dans lesquelles on ajoute nos documents. *Une collection est donc un ensemble de documents de même nature.*

La commande permettant de créer des collection:

db.<nom de la collection>.insert(<document>)

La création de la collection est implicite, elle se fait à l'insertion du premier document.

Nous allons créer une collections d'acteurs.

Pour insérer un premier acteur:

db.createCollection('acteurs')

ou directement

db.acteurs.insert({nom:"DENIRO", prenom:"Robert"})

nous avons ainsi enregistré le document :

{nom: "DENIRO", prenom: "Robert"}

Pour afficher les documents dans une collection :

db.acteurs.find()

ce qui nous donne

{ "_id" : ObjectId("57cdd765948a2cc7050c129e"), "nom" : "DENIRO", "prenom" : "Robert" }

Ici on trouve "_id" qui est un identifiant unique pour le document. S'il n'est pas spécifié à l'insertion, MongoDb génère un unique *ObjectId* qui identifie le document.

Si vous réalisez un enregistrement avec un format différent, il n'y a aucun

soucis. MongoDb est ***schemaless,*** ce qui signifie que les documents ne doivent pas tous respecter le même format.

La commande suivante ne posera aucun problème:

db.acteurs.insert({nom :''king kong'',taille:36})

Lorsque vous démarrez sur un nouveau projet, la base de données évolue beaucoup et les modifications du schéma sont courantes.

Néanmoins pour que ce soit maintenable, il faut que votre collection contienne des documents de même type.

Recherche

Quand votre collection contient de nombreux documents la méthode find ne vous affichera pas la totalité des documents, il faudra pour cela taper la commande ''**it**'' pour afficher les pages une à une.

Comme en sql vous avez l'option where

les acteurs portant le prénom robert :

*select * from acteurs where nom='robert'*

avec mongodb cela devient :

db.acteurs.find({prenom:'robert'})

autre requête sql

select * from acteurs where prenom='robert' and nom='deniro'

avec mongodb cela donne :

db.acteurs.find({nom:'robert', prenom :'deniro'})

Pour connaitre le nombre de résultats, vous pouvez appeler la méthode count() après la méthode find(). Par exemple
db.acteurs.find({prenom:'robert'}).count()

On peut aller plus loin sur le critère de recherches en ajoutant des opérandes :

- $gt : plus grand que
- $lt : plus petit que
- $gte : plus grand ou égal à
- $lte: plus petit ou égal à

Les 4 opérandes ci-dessus fonctionnent pour des nombres et des chaines de caractères. Pour les chaines de caractères, l'ordre appliqué est l'ordre alphabétique.

- $or : pour récupérer les documents qui correspondent à 2 critères différents
- $and : pour cumuler des critères, souvent inutile car il est implicite
- $in, $all, $exist, $type et $regex ...

exemple:

db.acteurs.find({taille: {$gte:25, $lt:100 }})

db.acteurs.find({ $or: [{ taille : { $lt : 20 } }, { age: { $gt: 70 } }] })

Remarque: *L'opérateur se met avant les tests sur les attributs. On ne peut pas le combiner à l'intérieur d'un test sur un attribut.*

db.acteurs.find({ genre: { $all: ['drama', 'western'] } })
La personne doit aimer « drama» ET « western ».

Pour le moment, lorsque nous faisons une requête, nous récupérons les documents dans leur ensemble. Si les documents sont volumineux, il peut être intéressant de ne récupérer que les valeurs qui nous intéressent.

MongoDb limite la taille des documents à 16MB, ce qui est une taille important pour un document JSON.

Si vos documents sont plus importants, vous pouvez les découper en plusieurs collections (et vous l'aurez probablement fait avant d'approcher cette limite).

Comme dans une base relationnelle, on utilise les _id pour faire le liens entre plusieurs collections. La différence est qu'il n'y a pas de jointure. Il vous faudra alors faire 2 requêtes.

La méthode find() prend un deuxième paramètre qui va nous servir à récupérer les propriétés qui nous intéressent, dans le cas où vous avez plus de trois 'champs 'dans votre document.

Pour trier les résultats, nous pouvons utiliser la méthode **sort()**.

Par exemple, si vous faites l'opération suivante:

db.acteurs.find({taille : {$in: [10, 20, 40]}})

Vous voyez que *taille* vaut alternativement 10, 20 et 40.

Maintenant si on trie de manière croissante sur *taille*:

db.acteurs.find({taille : {$in: [10, 20, 40]}}).sort({taille:1})

Pour trier dans l'autre sens :

db.acteurs.find({taille : {$in: [10, 20, 40]}}).sort({taille:-1})

En mettant -1 on trie de manière décroissante.

Pour limiter et filtrer le nombre de résultats de la fonction find()

Afficher un résultat à partir d'une position donnée

```
db.collection.find().skip(5)
```

retourne tous les documents mais à partir du 5 ème

Limiter l'affichage à un nombre donnée

```
db.collection.find().limit(5)
```
retourne 5 documents.

Skip() et limit() utilisés conjointement ont le même comportement que la commande sql 'limit()'

ex skip(2).limit(5) équivaut à limit(5,2)

Exercice :

Déterminer le nombre de films en base

Afficher à partir du cinquième film en base

Limiter l'affichage à 20

Afficher les films du numéro 5 jusqu'au numéro 20

Trier les films dans l'ordre descendant

La mise à jour .

Comme pour la méthode *find()*, le premier paramètre sert de critère de recherche pour les documents à mettre à jour, le second va servir à définir la mise à jour voulue.

Par exemple, si vous faites:

db.acteurs.update({nom: 'stalone'} , {$set : {metier:'boxeur'}})

Ensuite faites :

db.acteurs.find({nom: 'stalone'})

Vous constatez que le document avec pour nom 'stalone' a été mis à jour mais pas les suivants. Car MongoDb ne met à jour que le premier document qu'il trouve avec le critère donné. Si vous voulez tous les mettre à jour :

db.acteurs.update({nom: 'stalone'} , {$set : {metier:'boxeur'}}), {multi:true})

On indique avec le paramètre *multi* que l'on souhaite mettre à jour tous les documents trouvés.

On peut modifier la valeur que l'on a ajouté:

db.acteurs.update({nom: 'stalone'} , {$set : {metier:'cascadeur'}}), {multi:true})

Modifier ou créer un document (upsert) :

db.collection.update({ name: 'Jenkins' }, { $set: { age: 33 } }, { upsert: 1 })

si ne trouve aucun enregistrement correspondant à la condition alors enregistre un nouveau document avec les nouveaux critères.

SUPPRESSION

Supprimer la propriété que l'on vient d'ajouter : $unset.

db.acteurs.update({nom: 'stalone'} , {$unset : {metier:'cascadeur'}}), {multi:true})

Ici nous indiquons que nous voulons supprimer la propriété métier ajoutée au préalable.

Mise à jour d'un tableau

On va ajouter un tableau à un document

db.acteurs.insert({nom:'robert', tab:['blond','brun','roux']})

Vous pouvez le consulter avec la commande suivante:

db.acteurs.find({nom:'robert'})

Maintenant pour ajouter un élément, on utilise l'opérande **$push**:

db.acteurs.update({nom:'robert'}, {$push : {tab : 'chatain'}})

Si vous consultez à nouveau ce document vous constatez que le tableau tab contient:

['blond','brun','roux', 'chatain']

Pour en ajouter plusieurs d'un coup, il existe **$push $each** pour ajouter un tableau de valeur.

db.acteurs.update({nom:'robert'}, {$push : {tab :{$each :
['blond','brun','roux', 'chatain']}}

Avec l'opérande **$pop** on va supprimer le dernier élément:

db.acteurs.update({nom:'robert'}, {$pop : {tab:1}})

Le tableau a perdu son dernier élément "chatain".

Pour supprimer le premier élément:

db.acteurs.update({nom:'robert'}, {$pop : {tab:-1}})

De façon analogue avec la méthode sort, en mettant -1 on supprime les éléments dans l'autre sens.

Avec l'opérande **$addToSet**, on ajoute sans doublon :

db.acteurs.update({nom:'robert'}, {$addToSet : {tab : 'roux'}})

La valeur "roux" était présente dans le tableau donc il n'y pas eu d'ajout.

Enregistrer et insérer

db.acteurs.save({ nom: "scharze", prenom: 'arnold' })

Supprimer un document

La suppression d'un document se fait grâce à la méthode remove.

Par exemple:

db.acteurs.remove({nom:'robert'})

Supprime tous les documents qui ont pour pour valeur 'robert' à la propriété *nom*.

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 ;))

Exercice :

Insérer dans la collection movies un nouveau film.

Afficher la liste des films en base.

Afficher le premier film en base.

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

Rajouter un oscar pour tout les films.

Supprimer le film intutilé vertigo de la collection.

Manipulation de base

Il s'agit de créer une base, une collection, d'y insérer des données et de l'interroger. Les documents fournis correspondent à un extrait d'une base de publications scientifiques, The DBLP Computer Science Bibliography.

Gestion de collection

Voici le résumé des commandes à effectuer.

- Importer la base
- Se connecter à la base DBLP : use DBLP;

- Créer une collection “publis” : db.createCollection('publis');
- Créer le document suivant (astuce pour le client mongo: le mettre sur une ligne; c’est plus facile avec un client graphique type RoboMongo):

```
{
  "type": "Book",
  "title": "Modern Database Systems: The Object Model, Interoperability, and Beyond.",
  "year": 1995,
  "publisher": "ACM Press and Addison-Wesley",
  "authors": ["Won Kim"],
  "source": "DBLP"
}
```

- Insérer le document dans la collection publis avec db.publis.save(...);
- Créer et insérer deux autres publications à partir de cette page de conférence type “*Article*” (Vue “BibTeX”) : http://www.informatik.uni-trier.de/~ley/db/journals/vldb/vldb23.html
- Consulter le contenu de la collection : db.publis.find();
- Importer les données du TP dans MongoDB :
  1. Télécharger le fichier contenant les données : DBLP.json.zip (document fournis)
  2. Décompresser le fichier dblp.json.zip
  3. Dans le même répertoire, lancer l’importation du fichier :

     mongoimport – jsonArray --host localhost:27017 --db DBLP --collection publis < dblp.json

     vous donne après importation : *imported 118026 documents*

Note

Le chemin vers l’exécutable mongoimport est nécessaire, ou la variable d’environnement PATH contenant le chemin vers mongo/bin. L’opération peut prendre quelques secondes (118000 items à insérer)

- Dans la console mongo vérifier que les données ont été insérées

```
use DBLP ;

db.publis.count();
```

Interrogation simple

Exprimez des requêtes simples pour les recherches suivantes :

1. Liste de tous les livres (type “Book”) ;
   vous devez obtenir 11074 résultats
   pour afficher la suite tapez *it*
2. Liste des publications depuis 2011 ;
   vous devez obtenir 29499 résultats
3. Liste des livres depuis 2014 ;
   vous devez obtenir 288 résultats
4. Liste des publications de l’auteur “Toru Ishida” ;
   vous devez obtenir 22 résultats
5. Liste de tous les éditeurs (type “publisher”), **distincts** ;
   vous devez obtenir 539 résultats
6. Liste de tous les auteurs **distincts** ;
   vous devez obtenir 158772 résultats

Indexation (optionnel : si vous allez très vite)

Cette partie n’est pas vue (pas encore) en cours. Elle correspond à la découverte de l’indexation et de l’exécution de requêtes avec MongoDB: à vous d’explorer la documentation si vous voulez vous lancer.

- Pour chaque requête de type find(), regarder le plan d’exécution généré avec ***.explain()*** à la fin de la requête (modifiez votre requête en rajoutant à la fin ,explain(), exemple :
  - *b.acteurs.find({nom:'robert', prenom :'deniro'})*.explain()
- Créer un index sur l’attribut année db.publis.createIndex({ year :1 }); ;
- Refaire les requêtes find() sur l’année en regardant le plan d’exécution généré ;