Découverte de MongoDB

Table des matières

I. Contexte

Durée : 2h

Environnement de travail : Repl.it

Pré-requis : Aucun

Contexte

MongoDB est une base de données NoSQL qui stocke les données sous forme de documents JSON.

Elle est très utilisée aujourd'hui, particulièrement lorsque les volumes de données à stocker sont très importants et changent souvent. Un cas d'utilisation typique est l'analyse de statistiques d'utilisation d'une application, d'un site web, ou des données remontées par des objets connectés. MongoDB est aussi utilisée pour gérer des catalogues de produit et proposer des systèmes de recommandations, sur les sites de vente en ligne.

Dans ces cas où la cohérence des données n'est plus la priorité absolue, la rapidité d'accès aux données offerte par MongoDB est un véritable avantage.

II. Présentation et installation de MongoDB

MongoBD est une base de données *open source* NoSQL orientée document. Elle stocke des données au format JSON (en fait BSON, qui est une version binaire de JSON).

Le serveur MongoDB est organisé en plusieurs *databases* :

- Chaque *database* contient des *collections*.
- Chaque *collection* contient des *documents*.
- Chaque *document* est au format JSON et contient donc des propriétés.

Comparaison SQL / MongoDB

SQL	MongoDB
base de données et/ou schéma	base de données
table	collection
enregistrement	document
attribut (atomique)	propriété (chaîne, entier, tableau, objet)

Fondamental

MongoDB est particulièrement adaptée à la gestion de données imbriquées (ou arborescentes).

Schema-less

C'est une base *schema-less*, aussi une collection peut contenir des documents de structures différentes et il n'est pas possible de définir la structure *a priori* d'une collection. La structure d'une collection n'est donc définie que par les document qui la composent, et elle peut évoluer dynamiquement au fur et à mesure des insertions et suppressions.

Identification clé / valeur

Chaque document est identifié par un identifiant nommé « *_id* » unique pour une collection, fonctionnant comme une **clé primaire artificielle**.

Méthode **Installation**

MongoDB est disponible sur Windows, Mac OS X et Linux : https://docs.mongodb.com/manual/installation

L'installation présentée ici est uniquement destinée à un contexte d'apprentissage, elle permet l'installation d'un serveur sur une machine Linux (ou Mac OS X) sans privilèges utilisateur et l'exploitation de ce serveur avec un client textuel *CLI* situé sur la même machine.

On installera *MongoDB Community Edition*.

Méthode **Test**

Une fois MongoDB installé, lancer le client CLI MongoDB sur le serveur local (commande `mongo` sous Linux) et exécuter le code suivant pour vérifier le bon fonctionnement de la base :

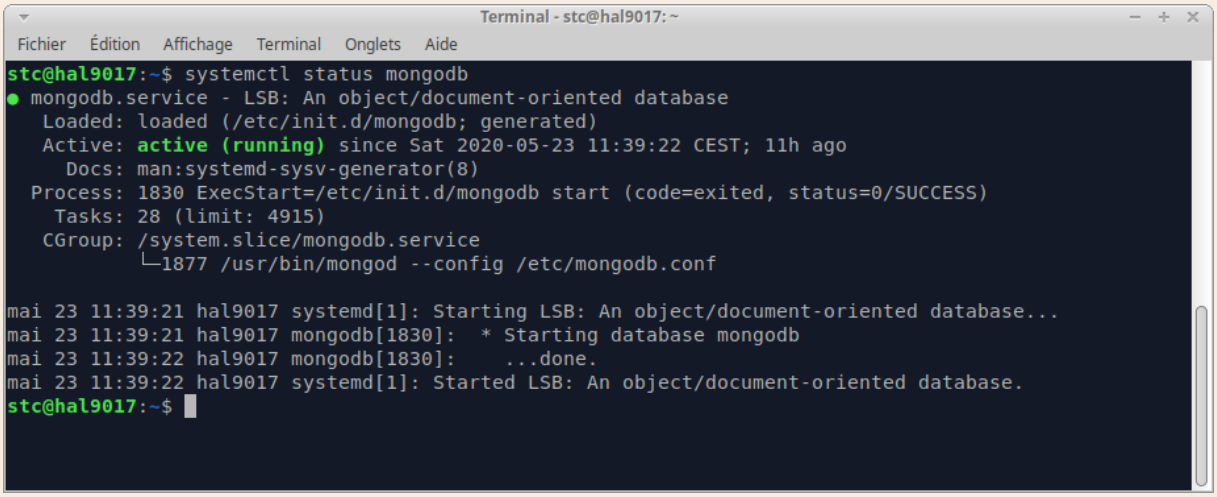
```
1 db.test.insertOne({ "test":"Hello World !" })
2 db.test.find({}, {_id:0})
```

Le résultat attendu est le suivant :

```
{ "test" : "Hello World !" }
```

Complément **Statut du serveur MongoDB**

Sous Linux on peut vérifier le statut du serveur grâce à la commande `systemctl status mongodb`.



Complément **Architecture**

MongoDB fonctionne *a minima* sous la forme d'un serveur auquel il est possible de se connecter avec un client textuel (*mongo shell*).

MongoDB peut être distribuée sur plusieurs serveurs (partitionnement horizontal ou *sharding*) et accédée à travers de multiples couches applicatives (langages, API, etc.)

Complément

https://docs.mongodb.org/manual

III. Exercice : Appliquer la notion

Soient les données suivantes représentant des films de cinéma :

```
db.Cinema.drop()

db.Cinema.insertOne(
{
nom:"Goodfellas",
annee:1990,
realisateur:{nom:"Scorsese", prenom:"Martin"},
acteurs:
  [
  {nom:"De Niro", prenom:"Robert"},
  {nom:"Liotta", prenom:"Ray"},
  {nom:"Pesci", prenom:"Joe"}
  ]
})

db.Cinema.insertOne(
{
nom:"The Godfather",
annee:1972,
realisateur:{nom:"Coppola", prenom:"Francis Ford"},
acteurs:
  [
  {nom:"Pacino", prenom:"Al"},
  {nom:"Brando", prenom:"Marlon"},
  {nom:"Duvall", prenom:"Robert"}
  ]
})

db.Cinema.insertOne(
{
nom:"Million Dollar Baby",
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
  {nom:"Swank", prenom:"Hilary"},
  {nom:"Eastwood", prenom:"Clint"}
  ]
})

db.Cinema.insertOne(
{
nom:"Gran Torino",
annee:2008,
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
  {nom:"Vang", prenom:"Bee"},
  {nom:"Eastwood", prenom:"Clint"}
  ]
```

```
})

db.Cinema.insert(
{
nom:"Unforgiven",
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
  {nom:"Hackman", prenom:"Gene"},
  {nom:"Eastwood", prenom:"Clint"}
  ]
})

db.Cinema.insertOne(
{
nom:"Mystic River",
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
   {nom:"Penn", prenom:"Sean"},
   {nom:"Bacon", prenom:"Kevin"}
  ]
})

db.Cinema.insertOne(
{
nom:"Honkytonk Man",
realisateur:{nom:"Eastwood", prenom:"Clint"},
annee:1982,
acteurs:
  [
   {nom:"Eastwood", prenom:"Kyle"},
   {nom:"Bloom", prenom:"Verna"}
  ]
})

db.Cinema.find()
```

L'objectif est d'initialiser une base MongoDB avec ce script, puis d'écrire les requêtes MongoDB permettant de répondre aux questions suivantes.

Question 1

Créer une nouvelle base MongoDB nommé *« Culture »*.

Indice :

Pour créer une base de données, utiliser l'instruction `use db_name`.

Question 2

Exécuter le script permettant d'initialiser la collection `Cinema`.

Question 3

Afficher la liste des films en supprimant le champ `_id`.

Indice :

On ajoute les arguments suivants à la méthode `find():{}, {_id:0}`.

IV. Créer des bases de données et des collections

Mise en situation

MongoDB, comme tous les autres systèmes de gestion de bases de données, permet de créer plusieurs bases de données. L'équivalent d'une table en relationnel, dans la terminologie MongoDB, serait une collection.

Mais la comparaison s'arrête là : en effet, une collection n'a pas de structure pré-définie. Si en général, les documents qui la composent sont de même nature, il n'y a aucune contrainte technique qui vérifie que c'est le cas.

Fondamental **Créer une base et une collection**

MongoDB est une base *schema-less*, la création des bases et des collections est dynamique.

- Les bases sont créées à leur premier accès si elles n'existent pas (commande `use`).
- Les collections sont créées lors d'une première insertion de données si elles n'existent pas (commande `insert`).

Syntaxe **Se connecter à une base de donnés (existante ou non)**

```
use db_name
```

Syntaxe **Insérer des données dans une collection (existante ou non)**

```
db.col_name.insertOne(JSON)
```

Exemple **Créer une première base de données**

Pour créer une base de données il faut exécuter une instruction `use` puis donner un ordre d'insertion d'un premier document JSON avec `insertOne`.

Exemple **Création d'une base de donnée hello et insertion**

```
use hello
db.world.insertOne( { "message":"Hello World !"} )
db.world.find()
```

Syntaxe **Catalogue de données**

- On peut voir la liste des bases de données avec :

  `show dbs`
- On peut voir la liste des collections de la base de données en cours avec :

  `show collections`
- On peut voir le contenu d'une collection `col_name` avec :

  `db.col_name.find()`

Complément

https://docs.mongodb.com/manual/mongo

V. Exercice : Appliquer la notion

On souhaite créer une base de données d'utilisateur pour gérer leurs préférences concernant les films qu'ils ont vus au cinéma.

Question 1

Sélectionner ou créer la base de données *« Culture »*.

Question 2

Créer la collection *« User »* et ajouter le document suivant qui signifie que la personne de pseudo *« Heisenberg »* aime le film *« Million Dollar Baby »* avec trois étoiles.

```
db.User.insertOne(
{
 "pseudo":"Heisenberg",
 "liked" :
   [
     {"film":"Million Dollar Baby","star":3}
   ]
}
)
```

Question 3

Afficher la liste de vos bases de données et la liste des collections de la base *« Culture »*.

```
show dbs
show collections
```

VI. Insertion des documents

Mise en situation

MongoDB, contrairement à une base de données relationnelle, n'identifie pas seulement les données par leur contenu *a priori*, mais appose automatiquement un identifiant unique pour chaque donnée insérée, peu importe si ce sont des doublons ou non.

L'insertion de données se fait toujours au sein d'une collection, mais vous êtes libres d'insérer autant de documents JSON que vous le souhaitez d'un seul coup.

Syntaxe

L'insertion de données dans une base MongoDB se fait avec les instructions :

- `db.col_name.insertOne(`*`objet JSON`*`)` pour insérer un seul document,
- `db.col_name.insertMany(`*`tableau d'objets JSON`*`)` pour insérer plusieurs documents.

Si la collection n'existe pas, elle est créée dynamiquement.

L'insertion associe un identifiant (*_id*) à chaque document de la collection.

Exemple **Insertion d'un objet simple**

```
db.Cinema.insertOne(
{
  nom:"Invictus",
  annee:2009
})
```

Exemple **Insertion d'un objet avec objet imbriqué**

```
db.Cinema.insertOne(
{
  nom:"Gran Torino",
  annee:2008,
  realisateur:{
    nom:"Eastwood",
    prenom:"Clint"
  }
})
```

Exemple **Insertion d'un objet avec tableau imbriqué**

```
db.Cinema.insertOne(
{
   "nom":"Honkytonk Man",
   "realisateur":{
      "nom":"Eastwood",
      "prenom":"Clint"
   },
   "annee":1982,
   "acteurs":[
      {
         "nom":"Eastwood",
         "prenom":"Kyle"
      },
      {
         "nom":"Eastwood",
         "prenom":"Clint"
      }
   ]
}
)
```

Exemple **Insertion de plusieurs objets**

```
db.Cinema.insertMany(
[
{
   "nom":"Mystic River",
   "realisateur":{
      "nom":"Eastwood",
      "prenom":"Clint"
   },
   "annee":2003,
   "acteurs":[
      {
         "nom":"Penn",
         "prenom":"Sean"
      },
      {
         "nom":"Robins",
         "prenom":"Tim"
      },
      {
         "nom":"Bacon",
         "prenom":"Kevin"
```

```
        }
    ]
},
{
    "nom":"Bird",
    "realisateur":{
        "nom":"Eastwood",
        "prenom":"Clint"
    },
    "annee":1982,
    "acteurs":[
        {
            "nom":"Whitaker",
            "prenom":"Forest"
        }
    ]
}
]
)
```

Complément

https://docs.mongodb.org/manual/core/crud[1]

https://docs.mongodb.com/manual/tutorial/insert-documents[2]

VII. Exercice : Appliquer la notion

On souhaite créer une base de données d'utilisateurs pour gérer leurs préférences concernant les films qu'ils ont vus au cinéma. Pour chaque utilisateur on gérera un pseudonyme, et une liste de films préférés avec une note allant de une à trois étoiles.

Question

Dans la collection `User` de la base de données `Culture`, en utilisant l'instruction `insertMany()` ajoutez trois utilisateurs tels que :

- « *Stph* » aime « *Goodfellas* » avec 3 étoiles et « *Million Dollar Baby* » avec 3 étoiles.
- « *Luke* » aime « *The Godfather* » avec 2 étoiles.
- « *Tuco* » aime « *Million Dollar Baby* » avec 3 étoiles.

Indice :

Insertion des documents (cf. p.)

VIII. Trouver des documents

Mise en situation

La recherche de documents dans une base de données MongoDB est très similaire à la récupération de données dans une base relationnelle avec la commande `SELECT`.

En effet, comme avec `SELECT`, il est possible de poser des conditions que doivent remplir les documents JSON pour être sélectionnés : récupérer tous les films réalisés par Tarantino, ou toutes les musiques de plus de 10 minutes, etc.

[1] https://docs.mongodb.org/manual/core/crud/
[2] https://docs.mongodb.com/manual/tutorial/insert-documents/

De plus, il est possible de restreindre l'affichage des données à quelques champs des documents JSON, par exemple pour n'afficher que le nom d'un film.

Ces opérations s'appellent restriction et projection.

Syntaxe **Recherche dans les données**

La recherche de données dans une base MongoDB se fait avec l'instruction `db.collection.find(`*`objet JSON, objet JSON`*`)`, avec :

- Le premier document JSON définit une restriction ;
- Le second document JSON définit une projection et est **optionnel**.

Exemple **Restriction**

```
db.Cinema.find({"nom":"Honkytonk Man"})
```

Retourne les documents JSON tels qu'ils ont à la racine un attribut `nom` avec la valeur « *Honkytonk Man* », donc tous les films qui se nomment « *Honkytonk Man* ».

Exemple **Restriction et projection**

```
db.Cinema.find({"nom":"Honkytonk Man"}, {"nom":1, "realisateur":1} )
```

Retourne les documents JSON tels qu'ils ont à la racine un attribut `nom` avec la valeur `Honkytonk Man`. Seuls les attributs situés à la racine `nom` et `realisateur` sont projetés, en plus de l'attribut « *`_id`* » qui est projeté par défaut.

Méthode

Il est possible d'utiliser de restreindre ou projeter en utilisant la profondeur des objets JSON pour accéder aux attributs situés dans des objets imbriqués, on utilise l'opérateur « `.` » (point).

Exemple **Restriction utilisant l'imbrication**

```
db.Cinema.find({"realisateur.nom":"Eastwood", "realisateur.prenom":"Clint"}, {"nom":1,
"acteurs.nom":1, "acteurs.prenom":1})
```

Retourne pour chaque film de Clint Eastwood, le nom du film ainsi que la liste des acteurs.

Syntaxe **Récupérer un document JSON intégral avec ObjectId()**

L'instruction `db.collection.find(ObjectId("uuid"))` renverra le fichier JSON correspondant à la clé `uuid`.

Méthode **Vider une collection**

```
db.Cinema.drop()
```

Complément **Supprimer des enregistrement**

Pour supprimer des enregistrement d'une collection `col_name`, on utilise la commande `db.col_name.drop()` sur le modèle de `find()`.

Exemple **Supprimer**

```
db.Cinema.drop({"nom":"Honkytonk Man"})
```

Supprime tous les films nommés « *Honkytonk Man* ».

Complément

Pour ne pas projeter l'attribut « `_id` » on ajoute `_id:0` à l'objet qui gère les projection.

Complément

https://docs.mongodb.org/manual/tutorial/query-documents/

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

IX. Exercice : Appliquer la notion

Initialiser une base de données avec le script suivant :

```
db.Cinema.drop()

db.Cinema.insertOne(
{
nom:"Goodfellas",
annee:1990,
realisateur:{nom:"Scorsese", prenom:"Martin"},
acteurs:
  [
  {nom:"De Niro", prenom:"Robert"},
  {nom:"Liotta", prenom:"Ray"},
  {nom:"Pesci", prenom:"Joe"}
  ]
})

db.Cinema.insertOne(
{
nom:"The Godfather",
annee:1972,
realisateur:{nom:"Coppola", prenom:"Francis Ford"},
acteurs:
  [
  {nom:"Pacino", prenom:"Al"},
  {nom:"Brando", prenom:"Marlon"},
  {nom:"Duvall", prenom:"Robert"}
  ]
})

db.Cinema.insertOne(
{
nom:"Million Dollar Baby",
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
  {nom:"Swank", prenom:"Hilary"},
  {nom:"Eastwood", prenom:"Clint"}
  ]
```

```
})

db.Cinema.insertOne(
{
nom:"Gran Torino",
annee:2008,
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
  {nom:"Vang", prenom:"Bee"},
  {nom:"Eastwood", prenom:"Clint"}
  ]
})

db.Cinema.insert(
{
nom:"Unforgiven",
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
  {nom:"Hackman", prenom:"Gene"},
  {nom:"Eastwood", prenom:"Clint"}
  ]
})

db.Cinema.insertOne(
{
nom:"Mystic River",
realisateur:{nom:"Eastwood", prenom:"Clint"},
acteurs:
  [
   {nom:"Penn", prenom:"Sean"},
   {nom:"Bacon", prenom:"Kevin"}
  ]
})

db.Cinema.insertOne(
{
nom:"Honkytonk Man",
realisateur:{nom:"Eastwood", prenom:"Clint"},
annee:1982,
acteurs:
  [
   {nom:"Eastwood", prenom:"Kyle"},
   {nom:"Bloom", prenom:"Verna"}
  ]
})

db.Cinema.find()
```

Question 1

Quels sont les films sortis en 1990 ?

Indice :

Trouver des documents *(cf. p.)*

Indice :

db.Cinema.find(*objet JSON*)

Indice :

```
db.col.find({"attribute":"value"})
```

Question 2

Quels sont les films sortis avant 2000 ?

Indice :

https://docs.mongodb.com/manual/tutorial/query-documents

Indice :

On utilisera l'objet `{$lt:value}` à la place de la valeur de l'attribut à tester (*$lt* pour *lesser than*).

Question 3

Quels sont les films réalisés par quelqu'un prénommé Clint avant 2000 ?

Indice :

Utiliser une liste de conditions *attribut:valeur* pour spécifier un *AND* (et logique) :

```
db.col.find({"attribute1":"value1", "attribute2":"value2"})
```

Question 4

Quels sont les noms des films dans lesquels joue Eastwood ?

Indice :

Pour gérer la **projection**, utiliser un second argument de la clause *find()* :

`db.Cinema.find({`*`document JSON de sélection`*` }, {`*`document JSON de projection`*`})`

avec document JSON de projection de la forme : `{"attribut1":1, "attribut2":1...}`

Indice :

Les identifiants sont toujours affichés par défaut. Si on veut les supprimer, on peut ajouter la clause `_id:0` dans le document de projection.

X. Essentiel

Une base de données MongoDB est composée de collections, qui rassemblent en général des documents JSON de même nature : des films, des chanteurs, des utilisateurs, des articles, etc.

En revanche, ce n'est pas une obligation technique : les collections n'ont aucune structure pré-définie, il est donc possible d'insérer tous les documents JSON bien formatés que vous souhaitez au sein d'une collection.

L'opération d'insertion crée automatiquement les bases de données et les collections si elles n'existent pas, et ajoute un identifiant unique à chaque document JSON.

La récupération et le filtrage des données se fait à l'aide des commandes de type `find`, qui reprennent les mêmes concepts que la commande `SELECT` de SQL.

XI. Auto-évaluation

A. Exercice final

Exercice : Quiz - Culture

Exercice

Une base de données *schema-less* est une base de données...

- ☐ Pour laquelle un schéma de données est défini avant d'insérer les données.
- ☐ Pour laquelle un schéma de données est défini automatiquement lors de la première insertion de données.
- ☐ Pour laquelle aucun schéma de données n'est défini.

Exercice

Lorsqu'une base de données contient un nombre très important de données, il est intéressant de répartir ces données sur plusieurs serveurs. On appelle cela le *sharding* (wikipedia.org/wiki/Shard[1]). Cela consiste à :

- ○ Faire un partitionnement vertical, c'est-à-dire sauvegarder certains attributs de chaque objet sur un serveur, d'autres sur un autre serveur, etc. Par exemple on mettrait tous les noms de films sur un serveur et tous les réalisateurs sur un autre.
- ○ Faire un partitionnement par objet : lorsqu'il y a imbrication, on stocke les objets imbriqués sur un autre serveur.
- ○ Faire un partitionnement horizontal : on enregistre une partie des objets sur un serveur, une autre partie sur un autre, etc. Par exemple on mettrait les 1000 premiers objets sur un serveur, les 1000 suivants sur un autre, etc.

Exercice

Pour créer une collection de données qui, à un auteur (par exemple Jules Verne) associe plusieurs œuvres qui ont chacune un nom et une date de publication (par exemple *« Voyage au centre de la Terre »*, en 1864 et *« Vingt-mille lieues sous les mers »* en 1869, etc.), quelle est la meilleure méthode ?

- ○ Faire une imbrication des œuvres dans les auteurs.
- ○ Faire une imbrication des auteurs dans les œuvres.

Exercice

Avant l'insertion de données avec MongoDB, il faut s'assurer que :

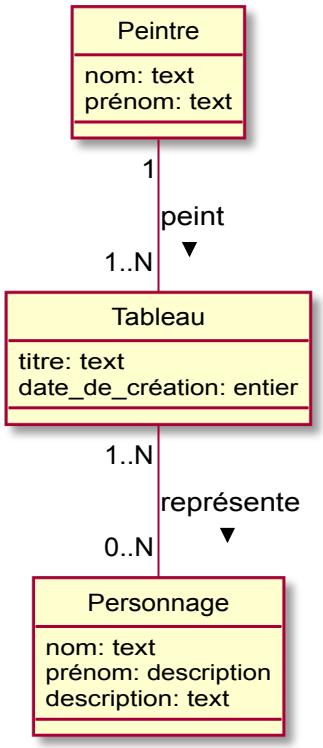
- ☐ La base de données existe
- ☐ La collection existe
- ☐ La donnée à insérer n'est pas identique à une donnée préexistante
- ☐ La donnée peut être identifiable par au moins un attribut unique et non nul (une clé naturelle)

Exercice : Quiz - Méthode

Exercice

Est-il possible de réaliser une imbrication MongoDB respectant ce schéma UML, sans redondance et sans faire de référence entre objets (donc en utilisant une seule collection) ?

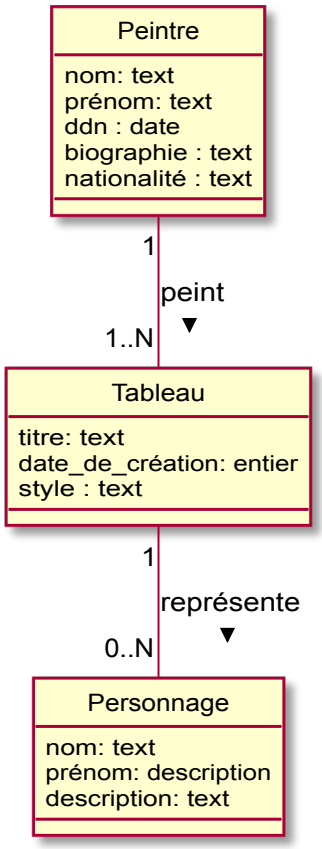
[1] https://en.wikipedia.org/wiki/Shard_(database_architecture)



- Oui
- Non

Exercice

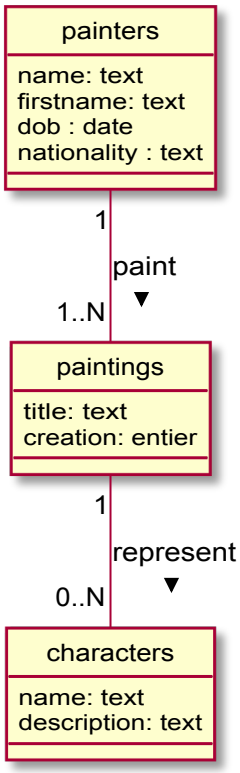
Est-il possible de réaliser une imbrication MongoDB respectant ce schéma UML, sans faire de référence entre objets (donc en utilisant une seule collection) ?



- Oui, en imbriquant les peintres et les personnages dans les tableaux
- Oui, en imbriquant les peintres et les tableaux dans les personnages
- Oui, en imbriquant les personnages et les tableaux dans les peintres
- Non

Exercice

Compléter l'instruction MongoDB suivante afin qu'elle corresponde au schéma UML.



db.Painter.insert(

{

" ": "Vinci",

" ": "Léonard",

" ": "1452-04-15",

" ": "Italien",

" ": [{

" ": "La Joconde",

" ": 1506,

" ": [{

" ": "Mona Lisa",

" ": "La Joconde est le portrait d'une jeune femme, sur fond d'un paysage montagneux aux horizons lointains et brumeux."

}]

}]

}

)

Exercice : Quiz - Code

Soit la base de données MongoDB :

```
db.Livres.insertOne(
{
"nom":"Therese Raquin",
"annee":1867,
```

```
"genre": "roman",
"auteur":{nom:"Zola", prenom:"Emile"},
"personnages":
  [
  {"prenom":"Therese"},
  {"prenom":"Camille"},
  ]
}
)

db.Livres.insertOne(
{
"nom":"Le mariage de Figaro",
"annee":1778,
"genre": "théâtre",
"auteur":{nom:"Beaumarchais", prenom:"P-A"},
"personnages":
  [
  {prenom:"Figaro"},
  {prenom:"Suzanne"},
  ]
}
)

db.Livres.insertOne(
{
nom:"L'avare",
annee:1668,
genre: "théâtre",
auteur:{nom:"Poquelin", prenom:"J-B"},
personnages:
  [
  {"prenom":"Harpagon"},
  {"prenom":"Mariane"},
  ]
}
)
```

Exercice

Compléter l'instruction suivante afin de renvoyer dans cet ordre : le nom, l'année de publication, le nom et prénom de l'auteur et le genre de chaque œuvre.

db.Livres.find({},
{"":1,"":1,"auteur.":1,"auteur.":1,"":1})

Exercice

Compléter l'instruction suivante afin de renvoyer le nom des livres de théâtre, suivi du le nom de leur auteur.

db..find({"":""},{"":1, "".":1})

Exercice

Compléter l'instruction suivante afin de supprimer toutes les données de la collection Livres.

db.().

B. Exercice : Défi : Books

On dispose d'une base de données orientée document, implémentée sous MongoDB représentant les livres d'une maison d'édition.

Le code de la base est le suivant :

```
db.Book.insert(
{
"title":"Logiciels et objets libres",
"subtitle":"Un support méthodologique pour les communautés de développement",
"authors":[  {"name":"Ribas", "firstname":"Stéphane"},
    {"name":"Guillaud", "firstname":"Patrick"},
    {"name":"Ubeda", "firstname":"Stéphane"}
  ],
"keywords": ["floss", "community", "development"],
"year":2016
}
)
db.Book.insert(
{
"title":"Numérique : reprendre le contrôle",
"subtitle":"Entre autonomie et souveraineté numérique, des témoignages éclairés.",
"authors":[  {"name":"Nitot", "firstname":"Tristan"},
    {"name":"Cercy", "firstname":"Nina"}
  ],
"keywords": ["privacy", "digital", "floss"],
"year":2016
}
)
db.Book.insert(
{
"title":"Un monde sans copyright et sans monopole",
"authors":[  {"name":"Smiers", "firstname":"Joost"},
    {"name":"van Schijndel", "firstname":"Marieke"}
  ],
"keywords": ["copyright"],
"year":2011
}
)
```

Question 1

Rétro-concevoir un MCD en UML correspondant à ce problème.

Question 2

Proposer un modèle relationnel correspondant à ce problème.

Question 3

Écrire la requête MongoDB permettant de trouver tous les titres des livres qui ont comme mot clé « *floss* ».

Indice :

La requête renvoie le résultat JSON suivant :

```
{ "_id" : ObjectId("59418102bb63d6ab4db45cc3"), "title" : "Logiciels et objets libres" }
{ "_id" : ObjectId("59418102bb63d6ab4db45cc4"), "title" : "Numérique : reprendre le contrôle"
}
```

Question 4

Réécrire la requête correspondante en imaginant que l'on interroge une base relationnelle implémentée selon notre modèle logique de données.

Solutions des exercices

Exercice p. Solution n°1

```
use Culture
```

Exercice p. Solution n°2

Valeur de retour de la dernière instruction

```
{ "_id" : ObjectId("5ec99f4f49cf212646753950"), "nom" : "Goodfellas", "annee" : 1990, "realisateur" : { "nom" : "Scorsese", "prenom" : "Martin" }, "acteurs" : [ { "nom" : "De Niro", "prenom" : "Robert" }, { "nom" : "Liotta", "prenom" : "Ray" }, { "nom" : "Pesci", "prenom" : "Joe" } ] }
{ "_id" : ObjectId("5ec99f4f49cf212646753951"), "nom" : "The Godfather", "annee" : 1972, "realisateur" : { "nom" : "Coppola", "prenom" : "Francis Ford" }, "acteurs" : [ { "nom" : "Pacino", "prenom" : "Al" }, { "nom" : "Brando", "prenom" : "Marlon" }, { "nom" : "Duvall", "prenom" : "Robert" } ] }
{ "_id" : ObjectId("5ec99f4f49cf212646753952"), "nom" : "Million Dollar Baby", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Swank", "prenom" : "Hilary" }, { "nom" : "Eastwood", "prenom" : "Clint" } ] }
{ "_id" : ObjectId("5ec99f4f49cf212646753953"), "nom" : "Gran Torino", "annee" : 2008, "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Vang", "prenom" : "Bee" }, { "nom" : "Eastwood", "prenom" : "Clint" } ] }
{ "_id" : ObjectId("5ec99f4f49cf212646753954"), "nom" : "Unforgiven", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Hackman", "prenom" : "Gene" }, { "nom" : "Eastwood", "prenom" : "Clint" } ] }
{ "_id" : ObjectId("5ec99f4f49cf212646753955"), "nom" : "Mystic River", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Penn", "prenom" : "Sean" }, { "nom" : "Bacon", "prenom" : "Kevin" } ] }
{ "_id" : ObjectId("5ec99f4f49cf212646753956"), "nom" : "Honkytonk Man", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "annee" : 1982, "acteurs" : [ { "nom" : "Eastwood", "prenom" : "Kyle" }, { "nom" : "Bloom", "prenom" : "Verna" } ] }
```

Exercice p. Solution n°3

```
db.Cinema.find({}, {_id:0})
```

```
{ "nom" : "Goodfellas", "annee" : 1990, "realisateur" : { "nom" : "Scorsese", "prenom" : "Martin" }, "acteurs" : [ { "nom" : "De Niro", "prenom" : "Robert" }, { "nom" : "Liotta", "prenom" : "Ray" }, { "nom" : "Pesci", "prenom" : "Joe" } ] }
{ "nom" : "The Godfather", "annee" : 1972, "realisateur" : { "nom" : "Coppola", "prenom" : "Francis Ford" }, "acteurs" : [ { "nom" : "Pacino", "prenom" : "Al" }, { "nom" : "Brando", "prenom" : "Marlon" }, { "nom" : "Duvall", "prenom" : "Robert" } ] }
{ "nom" : "Million Dollar Baby", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Swank", "prenom" : "Hilary" }, { "nom" : "Eastwood", "prenom" : "Clint" } ] }
{ "nom" : "Gran Torino", "annee" : 2008, "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Vang", "prenom" : "Bee" }, { "nom" : "Eastwood", "prenom" : "Clint" } ] }
{ "nom" : "Unforgiven", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Hackman", "prenom" : "Gene" }, { "nom" : "Eastwood", "prenom" : "Clint" } ] }
{ "nom" : "Mystic River", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "acteurs" : [ { "nom" : "Penn", "prenom" : "Sean" }, { "nom" : "Bacon", "prenom" : "Kevin" } ] }
{ "nom" : "Honkytonk Man", "realisateur" : { "nom" : "Eastwood", "prenom" : "Clint" }, "annee" : 1982, "acteurs" : [ { "nom" : "Eastwood", "prenom" : "Kyle" }, { "nom" : "Bloom", "prenom" : "Verna" } ] }
```

Exercice p. Solution n°4

```
use Culture
```

Exercice p. Solution n°5

```
db.User.insertMany(
[
{
 "pseudo":"Stph",
 "liked" :
   [
     {"film":"Goodfellas","star":3},
     {"film":"Million Dollar Baby","star":3}
   ]
},
{
 "pseudo":"Luke",
 "liked" :
   [
     {"film":"The Godfather","star":2}
   ]
},
{
 "pseudo":"Tuco",
 "liked" :
   [
     {"film":"Million Dollar Baby","star":3}
   ]
}
]
)
```

Exercice p. Solution n°6

```
db.Cinema.find({"annee":1990})
```

Exercice p. Solution n°7

```
db.Cinema.find({"annee":{$lt:2000}})
```

Exercice p. Solution n°8

```
db.Cinema.find({"realisateur.prenom":"Clint", "annee":{$lt:2000}})
```

Exercice p. Solution n°9

```
db.Cinema.find({"acteurs.nom":"Eastwood"}, {"nom":1,"_id":0})
```

Exercice p. 14 Solution n°10

Exercice

Une base de données *schema-less* est une base de données...

- ☐ Pour laquelle un schéma de données est défini avant d'insérer les données.
- ☐ Pour laquelle un schéma de données est défini automatiquement lors de la première insertion de données.
- ☑ Pour laquelle aucun schéma de données n'est défini.

Exercice

Lorsqu'une base de données contient un nombre très important de données, il est intéressant de répartir ces données sur plusieurs serveurs. On appelle cela le *sharding* (wikipedia.org/wiki/Shard[1]). Cela consiste à :

- ○ Faire un partitionnement vertical, c'est-à-dire sauvegarder certains attributs de chaque objet sur un serveur, d'autres sur un autre serveur, etc. Par exemple on mettrait tous les noms de films sur un serveur et tous les réalisateurs sur un autre.
- ○ Faire un partitionnement par objet : lorsqu'il y a imbrication, on stocke les objets imbriqués sur un autre serveur.
- ◉ Faire un partitionnement horizontal : on enregistre une partie des objets sur un serveur, une autre partie sur un autre, etc. Par exemple on mettrait les 1000 premiers objets sur un serveur, les 1000 suivants sur un autre, etc.

En général il est préférable de faire une partition horizontale, qui conserve l'intégrité de chaque objet (chaque objet est entièrement contenu dans une collection), et qui répartit l'ensemble des objets sur plusieurs serveurs.

Exercice

Pour créer une collection de données qui, à un auteur (par exemple Jules Verne) associe plusieurs œuvres qui ont chacune un nom et une date de publication (par exemple *« Voyage au centre de la Terre »*, en 1864 et *« Vingt-mille lieues sous les mers »* en 1869, etc.), quelle est la meilleure méthode ?

- ◉ Faire une imbrication des œuvres dans les auteurs.
- ○ Faire une imbrication des auteurs dans les œuvres.

Les œuvres sont uniques : imbriquer les œuvres dans les auteurs permet d'éviter la duplication des informations concernant les auteurs.

Exercice

Avant l'insertion de données avec MongoDB, il faut s'assurer que :

- ☐ La base de données existe
- ☐ La collection existe
- ☐ La donnée à insérer n'est pas identique à une donnée préexistante
- ☐ La donnée peut être identifiable par au moins un attribut unique et non nul (une clé naturelle)

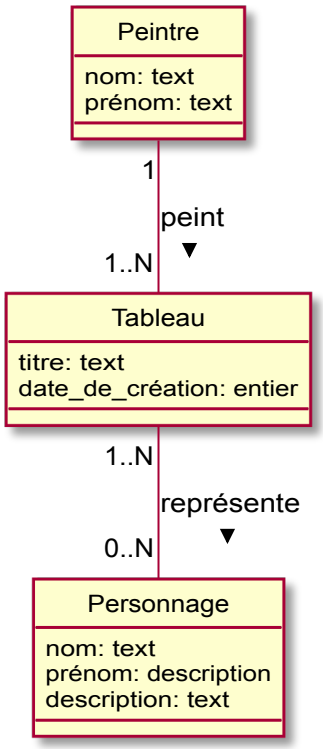
- MongoDB va créer une nouvelle base de données si celle demandée n'existe pas.
- MongoDB va créer une collection si elle n'existe pas.
- MongoDB ne vérifie pas si des données identiques sont déjà insérées dans la collection (on peut donc insérer des données en double).
- MongoDB va automatiquement fournir un identifiant unique artificielle à chaque objet d'une collection qui n'en possède pas.

Exercice p. 15 Solution n°11

Exercice

[1] https://en.wikipedia.org/wiki/Shard_(database_architecture)

Est-il possible de réaliser une imbrication MongoDB respectant ce schéma UML, sans redondance et sans faire de référence entre objets (donc en utilisant une seule collection) ?



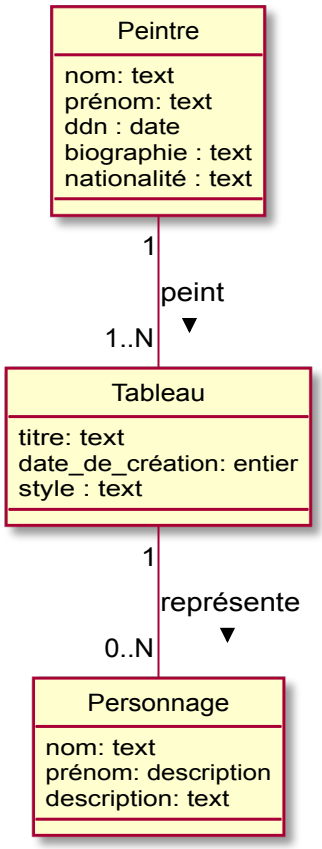
- ○ Oui
- ◉ Non

Étant donné l'association N:M entre les tableaux et les personnages, ce n'est pas possible :

- Soit on imbrique tout dans une collection et il y aura de la redondance,
- Soit on imbrique pas les personnages et il faudra un système de référence pour lier les personnages et les tableaux.

Exercice

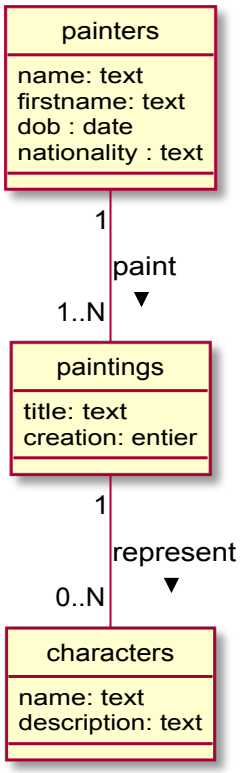
Est-il possible de réaliser une imbrication MongoDB respectant ce schéma UML, sans faire de référence entre objets (donc en utilisant une seule collection) ?



- ○ Oui, en imbriquant les peintres et les personnages dans les tableaux
- ○ Oui, en imbriquant les peintres et les tableaux dans les personnages
- ◉ Oui, en imbriquant les personnages et les tableaux dans les peintres
- ○ Non

Exercice

Compléter l'instruction MongoDB suivante afin qu'elle corresponde au schéma UML.



db.Painter.insert(

{

"name": "Vinci",

"firstname": "Léonard",

"dob": "1452-04-15",

"nationality": "Italien",

"paintings": [{

"title": "La Joconde",

"creation": 1506,

"characters": [{

"name": "Mona Lisa",

"description": "La Joconde est le portrait d'une jeune femme, sur fond d'un paysage montagneux aux horizons lointains et brumeux."

}]

}]

}

)

Exercice p. 18 Solution n°12

Exercice

Compléter l'instruction suivante afin de renvoyer dans cet ordre : le nom, l'année de publication, le nom et prénom de l'auteur et le genre de chaque œuvre.

db.Livres.find({},{"nom":1,"annee":1,"auteur.nom":1,"auteur.prenom":1,"genre":1})

Exercice

Compléter l'instruction suivante afin de renvoyer le nom des livres de théâtre, suivi du le nom de leur auteur.

db.Livres.find({"genre":"théâtre"},{"nom":1, "auteur.nom":1})

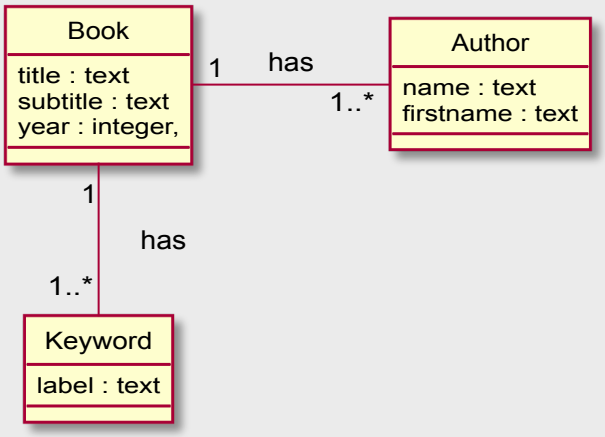
Exercice

Compléter l'instruction suivante afin de supprimer toutes les données de la collection Livres.

db.Livres.drop()

Exercice p. Solution n°13

Exemple



Attention

Les cardinalités minimales des associations côté *Book* doivent être de 1, car il ne peut pas exister d'auteur ou de licence en dehors des livres, par définition de l'imbrication.

Exercice p. Solution n°14

```
Author (#name:varchar, #firstname:varchar, #book=>Book)
Keyword (#label:varchar, #book=>Book)
Book (#id:integer, title:varchar, subtitle:varchar, year:integer)
  avec (title, subtitle, year) clé
```

Exercice p. Solution n°15

```
db.Book.find({"keywords":"floss"}, {"title":1})
```

Exercice p. Solution n°16

```
SELECT b.title
FROM book b
JOIN keyword k
ON k.book=b.id
WHERE k.label='floss'
```