Base de données NoSQL – TD 2
MongoDB

Master 2 BI&BD - 2021–2022 - 18/10/2021

1 Modélisation en Document

Un modèle de données efficaces doit supporter les besoins de votre application. La décision la plus importante est de choisir entre un modèle de document imbriqué ou un modèle de document normalisé.

1.1 Modèle : document imbriqué

Dans `mongoDB`, vous pouvez imbriquer des documents dans un seul et même document. Ce modèle est généralement connu comme un modèle "dénormalisé", et prend avantage de la richesse des documents `mongoDB`.

Prenons l'exemple de la collection `livres` que nous avons vu précédemment.

```
{ _id: "1",
  titre: "La Guerre et la paix",
  annee: 1869,
  auteur:{
          prenom: "Leo",
          nom: "Tolstoy"
         }
}
```

— Ici `auteur` est un document imbriqué,
— la modification de document se fait de manière atomique (Parent et enfant),
— cependant, on ne peut pas créer de document enfant avant le document parent.

Le modèle "document imbriqué" permet de stocker toutes les informations liées dans un seul et même document. Dans le cas où vous avez des relations un à plusieurs, le sous-documents sera défini comme un tableau.

```
{  prenom: "J. K.",
   nom: "Rowling"
   livres : [
            {titre: "Harry Potter à l'école des sorciers", annee: 1997},
            {titre: "Harry Potter et les Reliques de la Mort", annee: 2007}
            ]
   }
```

En général, imbriquer des documents permet de lire rapidement les informations. Pour l'utilisateur cela permet de réduire le nombre de requêtes à écrire. Cependant, ce modèle n'est pas recommandé si vous avez des documents qui grandissent après leur création (Ex. Utilisateur / Publications d'un réseau social).

1.2 Modèle : document normalisé

Le modèle "document normalisé" décrit les relations en utilisant des références entre documents. Ceci est comparable au concept de clé primaire / clé étrangère dans les systèmes de gestion de base de données relationnelle (SGBDR).

```
Auteur
       { _id : 12,
         prenom: "J. K.",
         nom: "Rowling""
       }
Livres
       { _id: 1,
         titre: "La Guerre et la paix",
         annee: 1869,
         auteur_id: 12
       },

       { _id: 2,
         titre: "Harry Potter et les Reliques de la Mort",
         annee: 2007,
         auteur_id: 12
       }
```

Ce modèle est à privilégier :

— si l'imbrication détériore les performances (temps d'exécution plus long),
— pour représenter des relations de type `M...N` très complexe,
— pour modéliser de grandes masses de données hiérarchisées.

L'utilisation des références permet une plus grande souplesse que l'imbrication de document. Par contre du côté client, les requêtes peuvent être plus complexes. Dans notre exemple ci-dessus, des opérations de jointure seront nécessaires pour afficher le titre du livre et le nom de l'auteur dans une seule requête. La gestion des données est un peu plus complexe, mais elle se fait de manière plus claire.

2 L'agrégation en mongoDB

`db.nom_de_la_collection.aggregate(pipeline, options)` est utilisé pour faire les agrégations dans mongoDB.

- **`pipeline`** : est une suite d'opérations à exécuter les unes à la suite des autres. Il existe plusieurs opérateurs, nous ne verrons que quelques-uns durant ce TD. Pour plus d'informations, allez sur `https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/`. Dans un `pipeline`, la sortie du premier opérateur devient l'entrée du second et ainsi de suite.
- **`options`** : ne sont pas obligatoires. Par exemple vous pouvez définir `allowDiskUse` un booléen, qui permet de définir si vous souhaitez que mongoDB écrive dans le `_tmp` du `dbPath` de `mongoDB`.

2.1 Regroupement et somme

`$group` et `$sum` sont les opérateurs de regroupement et de somme.
Exemple : soit la collection **`sales`** suivant :

```
{_id : 1, item : "abc", price : 10, qty : 2, date : ISODate("2014-01-01T08:00:00Z")}
{_id : 2, item : "jkl", price : 20, qty : 1, date : ISODate("2014-02-03T09:00:00Z")}
{_id : 3, item : "xyz", price : 5, qty : 5, date : ISODate("2014-02-03T09:05:00Z")}
{_id : 4, item : "abc", price : 10, qty : 10, date : ISODate("2014-02-15T08:00:00Z")}
{_id : 5, item : "xyz", price : 5, qty : 10, date : ISODate("2014-02-15T09:05:00Z")}
```

Comme en SGBDR vous aurez besoin d'un regroupement **`$group`** pour effectuer une `$sum` :

```
db.sales.aggregate(
   [{
       $group:
         {
           _id: { day: { $dayOfYear: "$date"}, year: { $year: "$date" } },
           totalAmount: { $sum: { $multiply: [ "$price", "$qty" ] } },
           count: { $sum: 1 }
         }
     }]
)
```

- **`$group`** permet de faire un regroupement. Ici sur le jour de l'année et l'année,
- **`$nom_de_champ`** permet de faire un appel par référence au nom de champ (`$date, $quantity, $price ...`),
- **`$sum`** effectuera l'opération d'agrégation somme,
- **`$multiply`** va exécuter une opération algébrique de multiplication. Il prendra en paramètre le `$price` et le `$qty`,
- **`count`**: `$sum:1` comptera le nombre de ventes par jour.

Le résultat de la requête sera :

```
{ _id : {day : 46, year : 2014 }, totalAmount : 150, count : 2 }
{ _id : {day : 34, year : 2014 }, totalAmount : 45, count : 2 }
{ _id : {day : 1, year : 2014 }, totalAmount : 20, count : 1 }
```

N.B. **`$sum`** renvoie 0 si vous effectuez l'opération sur un champ inexistant.

2.2 Moyenne

L'opérateur `$avg` calculera la moyenne. Avec l'exemple précédent :

```
db.sales.aggregate(
   [
     {
       $group:
         {
           _id: "$item",
           avgAmount: { $avg: { $multiply: [ "$price", "$qty" ] } },
           avgQuantity: { $avg: "$qty" }
         }
     }
   ]
)
```

— **`$avg`** est utilisé pour calculer la moyenne des ventes et la quantité moyenne vendue.

Le résultat de la requête sera :

```
{ "_id" : "xyz", "avgAmount" : 37.5, "avgQuantity" : 7.5 }
{ "_id" : "jkl", "avgAmount" : 20, "avgQuantity" : 1 }
{ "_id" : "abc", "avgAmount" : 60, "avgQuantity" : 6 }
```

2.3 Affichage du résultat d'une agrégation

2.3.1 Projection

L'opérateur `$project` réorganise chaque document dans le `pipeline`, en ajoutant de nouveaux champs ou en supprimant les champs existants.
Soit un document de notre collection **`sales`** :

```
{_id : 1, item : "abc", price : 10, qty : 2, date : ISODate("2014-01-01T08:00:00Z")}
```

L'agrégation avec **`$project`** :

```
db.sales.aggregate( [ { $project : { item : 1 , price : 1 } } ] )
```

Affichera comme résultat :

```
        {
          item : "abc",
          price : 10
        }
```

2.3.2 Sélection

`$match` filtre les documents, `$match` est constitué de conditions de sélection. Seuls les documents satisfaisant le filtre passeront à l'étape suivante du `pipeline`.

1. **Optimisations :**
   - Placez le `$match` un peu plus haut dans les opérations du `pipeline` d'agrégation. Cela permet de limiter le nombre de documents à traiter dans les opérations suivantes,
   - Si vous exécutez le `$match` comme première opération, la requête tirera avantage de l'indexation comme `db.collection.find()` ou `db.collection.findOne()`.
2. **Restrictions :**
   - Vous ne pouvez pas utiliser l'opérateur `$where` dans l'opérateur `$match`,
   - Pour utiliser `$text`, dans `$match`, il doit être à la première position du `pipeline`.

Avec notre collection **`sales`** :

```
db.sales.aggregate( [ { $match : { item : "xyz" } } ] )
```

Ce qui donnera comme résultat :

```
{
  _id : 5,
  item : "xyz",
  price : 5,
  qty : 10,
  date : ISODate("2014-02-15T09:05:00Z")
}
```

2.3.3 Tri

`$sort` permet de trier les documents et de les renvoyer dans le `pipeline` dans l'ordre que vous souhaitez. Il fonctionne comme suit : **`$sort: <champ1>: <ordre>, <champ2>: <ordre> ...`** .

- `"-1"` pour un tri croissant
- `"1"` pour un tri décroissant

Avec notre collection **`sales`**, avec un tri sur les champs `qty` et `price` la requête sera :

```
db.sales.aggregate([{$sort: {qty:-1, price:1}}])
```

Ce qui donnera comme résultat :

```
{_id : 2, item : "jkl", price : 20, qty : 1, date : ISODate("2014-02-03T09:00:00Z")}
{_id : 1, item : "abc", price : 10, qty : 2, date : ISODate("2014-01-01T08:00:00Z")}
{_id : 3, item : "xyz", price : 5, qty : 5, date : ISODate("2014-02-03T09:05:00Z")}
{_id : 4, item : "abc", price : 10, qty : 10, date : ISODate("2014-02-15T08:00:00Z")}
{_id : 5, item : "xyz", price : 5, qty : 10, date : ISODate("2014-02-15T09:05:00Z")}
```

2.3.4 Limit

`$limit` permet de limiter le nombre de documents à envoyer à l'étape suivante du `pipeline`. Avec notre collection **`sales`** :

```
db.sales.aggregate([{$sort: {qty:-1, price:1}}
                    {$limit:2}])
```

Ce qui donnera comme résultat :

```
{_id : 2, item : "jkl", price : 20, qty : 1, date : ISODate("2014-02-03T09:00:00Z")}
{_id : 1, item : "abc", price : 10, qty : 2, date : ISODate("2014-01-01T08:00:00Z")}
```

2.4 Jointure

Les SGBD documents tel que `mongoDB` ont été faits pour stocker des données dénormalisées et non-structurées. Idéalement, il ne devrait y avoir aucune relation entre collections de documents. Si on a besoin de la même donnée dans deux ou plusieurs documents, cette donnée devra être répétée. Cepedant il y a peu de situations ou de cas d'utilisation où l'on n'a pas besoin de relation entre les données. Dans `mongoDB` l'opérateur de jointure est **`$lookup`**.

`$lookup` est seulement permis dans les opérations d'agrégation. Pour mieux comprendre le concept, voici un exemple :
Supposons que nous allons créer un réseau social avec une collection `utilisateurs`. Cette collection stockera toutes les informations sur l'utilisateur.

```
{   _id: ObjectID(45b83bda421238c76f5c1969)
    nom: "Dupond",
    email: "dupond@mail.com",
    pays: "France"
    DN : ISODate("1992-09-13T00:00:00.000Z")
}
```

Notre réseau social a maintenant besoin d'une collection `publications` qui retracera toutes les publications de l'utilisateur. Une `publication` sera un document contenant le `texte` publié, la `date` de la publication, la référence à l'utilisateur `user_id` et la `note`.

```
{    _id: ObjectID(17c9812acff9ac0bba018cc1),
     user_id: ObjectID(45b83bda421238c76f5c1969),
     date : ISODate("2017-01-30T18:30:00.000Z"),
     texte: "C'est quoi une jointure ?",
     note: "important"
}
{     _id: ObjectID(18c9812acff9ac0bba019cc2),
      user_id: ObjectID(45b83bda421238c76f5c1969),
      date : ISODate("2017-01-30T20:30:00.000Z"),
      texte: "C'est difficile à mettre en oeuvre",
       note: "important"
}
```

Maintenant si l'on souhaite afficher les deux dernières publications de l'utilisateur "Dupond" qui ont une `note "important"`, trier du plus récent au plus ancien.

1. Premièrement, nous aurons besoin d'une opération de $Selection$ qui va filtrer toutes les publications avec `{"note":"important"}` :

```
{ "$match": { "note": "important" } }
```

2. Deuxièmement, l'opération de tri :

```
{ "$sort": { "date": -1 } }
```

3. Troisièmement, étant donné qu'on a besoin que des deux dernières publications :

```
{ "$limit": 2}
```

4. **Quatrièmement, nous pouvons maintenant faire la jointure entre nos collections `publications` et `utilisateurs` avec l'opérateur `$lookup` :**
   - **`localField`** : le champ (`field`) du document,
   - **`from`** : le nom de la collection à joindre,
   - **`foreignField`** : le champ (`field`) à rechercher dans la collection jointe,
   - **`as`** : le nom du champ de sortie à afficher.

```
{ "$lookup": {
             "localField": "user_id",
             "from": "utilisateurs",
             "foreignField": "_id",
             "as": "infoU"
             }
}
```

   Cette action va nous créer un nouveau champ `infoU`. Ce champ sera un tableau contenant les informations concernant l'utilisateur qui remplira les conditions de la jointure.

   N.B étant donné que nous avons une relation 1..N entre `utilisateurs` et `publications`, le champ utilisateur ne contiendra qu'un seul document. L'opérateur `$unwind` peut être utilisé pour décomposer le champ `infoU` en sous-document.

```
{ $unwind: "$infoU"}
```

5. Finalement, le `$project` va nous servir à définir les champs à retourner.

```
{ "$project": {
              "text": 1,
              "date": 1,
              "infoU.nom": 1,
              "infoU.pays": 1
              }
}
```

La requête de jointure :

```
db.post.aggregate([
                  { "$match": { "rating": "important" } },
                  { "$sort": { "date": -1 } },
                  { "$limit": 2},
                  { "$lookup": {
                              "localField": "user_id",
                              "from": "utilisateurs",
                              "foreignField": "_id",
                              "as": "infoU"
                              }
                  },
                  { "$unwind": "$infoU" },
                  { "$project": {
                              "text": 1,
                              "date": 1,
                              "infoU.nom": 1,
                              "infoU.pays": 1
                              }
                  }
                  ]);
```

Le résultat sera :

```
[
  {
    "text": "C’est difficile à mettre en oeuvre",
    "date: ISODate("2017-01-30T20:30:00.000Z"),
    "userinfo": {
      "name": "Dupond",
      "country": "France"
    }
  },
  {
    "text": "C’est quoi une jointure ?",
    "date: ISODate("2017-01-30T18:30:00.000Z"),
    "userinfo": {
      "name": "Dupond",
      "country": "France"
    }
  }
  ...
]
```

3 Étude de cas

On veut implanter une base de données pour la gestion des rencontres des tournois de tennis d'une saison donnée. On pourra ainsi déterminer facilement le classement des joueurs au niveau mondial. Les spécifications vous sont données ci-dessous.

3.1 Spécifications

Chaque joueur a un nom, un sexe et représente un pays. Deux joueurs peuvent former une équipe (de double, mixte ou non). Un tournoi est identifié par son nom et se déroule dans un pays donné à une date prévue. La dotation des gagnants varie selon les tournois et est exprimée dans la monnaie du pays d'accueil. Afin de plus tard afficher les dotations en utilisant la bonne "unité", on souhaite également stocker le nom de la monnaie de chaque pays. À la fin d'un tournoi, un joueur ou une équipe participant à ce tournoi obtient un score qui représente le nombre de tours passés dans le tournoi (1er tour vaut 1 point, 2ème tour vaut 2 points, etc.). On attribue à chaque tournoi un coefficient selon son importance.

Le score final d'un joueur (ou d'une équipe) est obtenu de la manière suivante : $\sum_{i=1}^{n} score * coefficient$ pour les n de l'année. Les joueurs (ou équipes) sont classés par ordre décroissant de leur score final.

Modèle conceptuel (UML)

Le modèle conceptuel suivant est fourni pour la base de données.

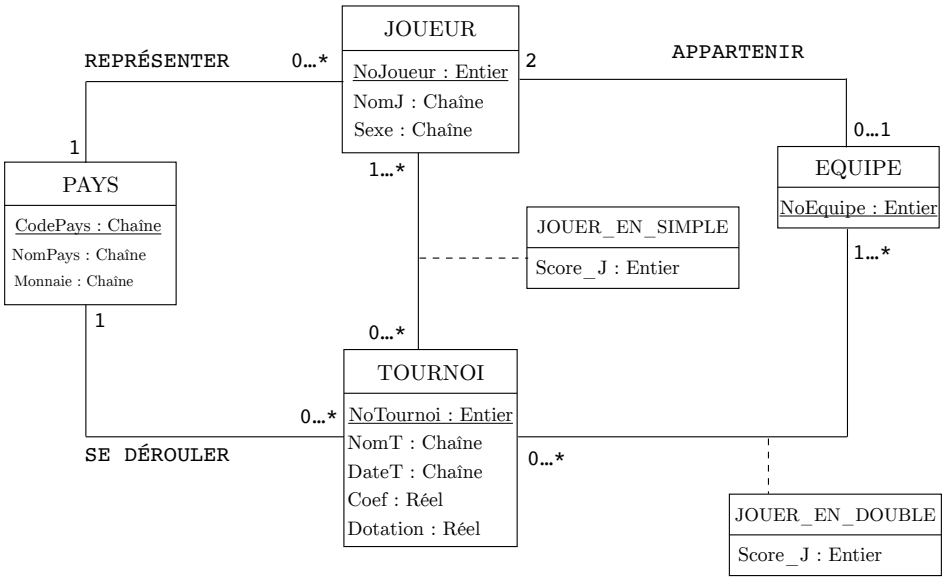

Figure 1 – Modèle conceptuel

3.2 Travail à faire

3.2.1 Modèle logique

1. Traduire le modèle conceptuel UML proposé (FIGURE 1) en modèle logique document. Les attributs de chaque classe seront considérés comme clé des documents à créer.

3.2.2 Modèle physique

1. Créez une base de données mongoDB, nommez-le TD2.
2. Créez les collections correspondant à vos tables du modèle logique.

3.2.3 Remplissage de la base de données

1. Saisir les informations suivantes dans les collections correspondantes.

NoJoueur	NomJoueur	Sexe
10	Roddick	Masculin
20	Ginepri	Masculin
30	Gasquet	Masculin
40	Monfils	Masculin
100	Mauresmo	Féminin
200	Davenport	Féminin

Données 1 – Joueurs

CodePays	NomPays	Monnaie
AUS	Australie	AUD
FRA	France	EU
USA	Etats-Unis	USD

Données 2 – Pays

NoEquipe
1
2

Données 3 – Equipes

NoTournoi	NomTournoi	DateT	Coef	Dotation
1	Roland Garros	10/06/2000	10	700 000
11	Open d'Australie	15/10/2000	5	700 000
111	Flusshing Meadows	10/11/2000	6	1 000 000
1111	Open de Paris-Bercy	10/12/2000	4	300 000

Données 4 – Tournois

2. Certaines informations sont manquantes dans les tables. Les compléter, sachant que :
   - Roddick, Ginepri et Davenport sont américains ;
   - Gasquet, Monfils et Mauresmo sont français ;
   - Roddick et Monfils sont partenaires de double ;
   - Ginepri et Gasquet sont partenaires de double ;
   - Mauresmo et Davenport ne jouent qu'en simple ;
   - Résultats de Roland Garros :
     - Roddick marque 7 points,

    - Ginepri marque 8 points,
    - Mauresmo marque 2 points,
    - Davenport marque 4 points,
    - Roddick et Monfils marquent 9 points,
    - Ginepri et Gasquet marquent 7 points ;
- Résultats de l'Open d'Australie :
    - Roddick marque 8 points,
    - Gasquet marque 1 point,
    - Monfils marque 3 points
- Résultats de Flushing Meadows :
    - Roddick marque 0 point,
    - Roddick et Monfils marquent 7 points,
    - Ginepri et Gasquet marquent 8 points ;
- Résultats de l'Open de Paris Bercy :
    - Roddick marque 0 point.

3.2.4 Requêtes

Exprimer les requêtes suivantes en requête mongo :

Requêtes simples avec `find()`

1. Liste des joueurs, n'affichez que le nom et le pays.
2. Nom des joueuses.
3. Liste des tournois se déroulant en France **ou** dont la dotation est supérieure à 800 000.
4. Nom des joueurs français qui jouent dans une équipe de double.
5. Pour chaque tournoi, son nom, le nom du pays où il se déroule (pas le code pays), sa dotation et la monnaie dans laquelle elle s'exprime.
6. Le nom du tournoi qui a la dotation minimum.
7. Le nom du tournoi qui a la dotation maximum.

Requêtes complexes avec `aggregate()`

1. Affichez la liste des joueurs triée par ordre alphabétique de nom.
2. Affichez la moyenne des scores de chaque joueur de simple (indiquer le nom des joueurs). Triez la moyenne par ordre décroissant.
3. Affichez le score final de chaque équipe de double et classer les équipes de la meilleure à la moins bonne.
4. Affichez, pour chaque joueur, son numéro de joueur, son nom, son pays et son score total en simple.
5. Affichez le nom du joueur qui a joué tous les tournois en simple.

3.3 Travail à rendre par mail

Travail à envoyer par mail à l'adresse : `mohamed-lamine.messai@univ-lyon2.fr`

1. Votre modèle logique de document.
2. Les requêtes que vous avez écrit.
3. Les résultats des requêtes.

N.B. Date limite : la fin de la séance prochaine.