Qualité et au delà du relationnel

BUT2, CM2, 2022-2023

Gilles Nachouki

1- Etude de la qualité dans les bases de données relationnelles : concevoir un schéma normalisé et connaître la qualité du schéma avec ses avantages et ses inconvénients

2- Aller au delà du modèle relationnel : les bases de données NoSQL.

Au sommaire:

- Bases de données relationnelles :
  1. Modèle de données
  2. Normalisation et qualité
- Bases de données non relationnelles :
  1. Modèles de données
  2. Introduction à Mongodb

Introduction

Le modèle de données relationnel et la normalisation des données ont montré leurs limites. En effet, avec l'augmentation du volume de données manipulé (centaines-milliers Teraoctets), le modèle de données non relationnel est apparu et s'est imposé en proposant d'abandonner quelques contraintes du relationnel et de se focaliser sur la manière de stocker et d'interroger ces données.

Plusieurs modèles de données

Le NOSQL (Not Only SQL) est une base de données non relationnelle. Les systèmes de gestion de base de données correspondants permettent le stockage et l'analyse du Big Data (ou données massives). Plusieurs modèles de données sont apparus pour gérer les données dans les bases de données NOSQL :

1. Clés/Valeurs
2. Colonnes
3. Documents
4. Graphes

Le modèle de données qui sera choisi dépendra des besoins de l'application

Modèle de données orienté *Clé/Valeur*

1- La *clé* identifie de façon unique la donnée et *valeur* contient n'importe quel type de données. Cette façon de décrire les données peut être vue comme une table de hachage (distribuée) facilitant ainsi la recherche des données sur le réseau. Ce modèle permet la prise en charge de larges volumes de données

Systèmes : Rebis, SimpleDB
Applications : e-commerce, messagerie en ligne.

Modèle de données orienté *Colonne*

2- Ce type de stockage s'intéresse aux colonnes (au lieu des lignes dans les systèmes classiques) en les distribuant sur le réseau. Dans ce modèle chaque colonne est traitée séparément et les valeurs sont stockées de façon contigüe. Ce modèle offre de hautes performances pour effectuer de gros traitements sur des colonnes (agrégats)

Systèmes : BigTable, Hbase etc.
Applications : compteur, moyenne, etc.

Modèle de données orienté *Document*

3- Ce modèle repose sur le modèle Clé/Valeur avec une flexibilité accrue car il permet de stocker des valeurs complexes sous forme de documents JSON. A la différence du modèle Clé/Valeur, une valeur dans le modèle orienté document peut être examinée par une requête.

Systèmes : MongoDB, Cassandra etc.
Applications : diverses

Modèle de données orienté *Graphe*

3- Dans ce modèle les données sont représentées par des noeuds et les liens entre les noeuds représentent des liens sémantiques entre les données dans les noeuds.

Systèmes : Neo4J, FlockDB etc.
Applications : réseaux (sociaux, électrique etc.)

Caractéristiques des bases de données NOSQL

1. Différents modèles de stockage de données
2. Absence de jointure : les bases de données NOSQL sont capables d'extraire les données sans effectuer de jointure
3. Abasence de schéma : Un utilisateur n'a pas besoin de définir le schéma de la base de données NOSQL qui continue de changer dans le temps
4. Plusieurs serveurs : la plupart des systèmes NOSQL offrent la possibilité de stocker une base de données sur plusieurs serveurs tout en gardant de bonnes performances

Propriétés des bases de données NOSQL

Les propriétés ACID des SGBDR (vu au semestre 3) ne sont pas applicables dans un contexte distribué tel que NOSQL (provoquent des latences dans les transactions). Les propriétés suivantes appelées **BASE** ont été proposées dans ce cadre dans le but est de relâcher la pression sur les transactions :

- BA *Basically Available :* le système doit garantir un taux de disponibilité des données
- S *Soft-state :* la base de données n'a pas à être cohérente à tout instant
- E *Eventually consistent :* A terme, la base atteint un état cohérent

Théorème de Brewer et les trois propriétés : CAP

Eric A. Brewer a proposé trois propriétés pour caracériser les bases de données :

1. Consistency (C) : la base de données doit être dans un état cohérent : une donnée doit avoir une seule valeur quelque soit le nombre des réplicas
2. Availability (A) : la base de données doit rester disponible
3. Partition (P) : Le système doit continuer à fonctionner même si la communication entre des serveurs est temporairement interrompu.

Triangle CAP

1. Couple CA : représente le fait que des opérations concurrentes sur un même granule retournent une nouvelle valeur cohérente du granule sans délai d'attente (système SQL)
2. Couple CP : permet de répartir les données sur des serveurs tout en garantissant la cohérence des données. Cette cohérence est assurée par un protocole de synchronisation des données dupliquées en introduisant un temps de latence dans les réponses aux requêtes (système NOSQL MongoDB)
3. Couple AP : permet de répartir les données et garantir la disponibilité des données (réponses rapides aux requêtes) mais il ne garantit pas la cohérence des données grace à l'utilisation d'un protocole asynchrone des replicas (système NOSQL Cassandra)

Avantages et inconvénients des systèmes relationnels

1. Les propriétés ACID facilite le développement
2. Les contraintes d'intégrités doivent être respectées
3. Les vues assurent la confidentialité des données
4. Les vues simplifient l'écriture des requêtes
5. Les utilisateurs sont familiarisés avec le modèle relationnel
6. Comprendre et définir le schéma de la base de données. La modification du schéma peut être couteuse
7. Les jointures affectent les performances du système
8. Les performances sont limitées pour un très grand volume de données (des Teraoctets)
9. L' évolutivité se fait de manière verticale (cad. la mise à l'échelle se fait en ajoutant davantage de puissance mémoires ou CPU)

Avantages et inconvénients du système non relationnel

1. Distribution des données sur les serveurs
2. l' évolutivité se fait de manière horizontale en ajoutant plus de serveurs
3. Absence de jointures
4. Absence de schéma
5. le paradigme NOSQL est une technologie récente pour beaucoup d'utilisateurs
6. Absence de standard au niveau des langages de requêtes
7. Des modèles de données où chacun répond à un besoin précis

Dans le cadre d'un SGBDR la première étape consiste à concevoir le schéma de la base de données avant de déterminer les requêtes ou les applications qui utiliseront cette base
Dans le contexte NOSQL il est important de connaitre le type d'application qui utilise le futur base avant de commencer l'étape de modélisation.

Pour la modélisation de données nous distinguons trois types d'association entre les entités (repris du modèle Entité-Association):

1. 1-1
2. 1-N
3. N-N

Les associations entre les entités dans les documents peuvent être exprimés sous les deux formes suivantes:

1. 1-1 : par integration
2. 1-N : par integration
3. N-N : par référence

Lien de type 1-1 :
Nous considérons le cas où une personne peut être mariée (ou pacsée) avec une autre personne.
Pour modéliser cette situation, nous utilisons la première forme par intégration pour représenter cette association. Dans ce cas le lien entre les deux personnes se trouve intégré dans un même document en ajoutant les propriétés respectives des personnes : nom, prénom, age etc.

Lien de type 1-N :
Nous considérons le cas où une personne possède plusieurs voitures.

Pour modéliser cette situation, nous utilisons la forme d'intégration pour représenter cette relation. Dans ce cas, une personne est décrite avec toutes ses voitures sous forme d'un tableau dans le même document avec leurs propriétés respectives.

Lien de type N-N :

Nous considérons le cas où dans un Bon de Commande (BC) il y a plusieurs produits qui sont référencés et qu'un produit peut être référencé dans plusieurs BC.

Pour modéliser cette situation, nous utilisons la deuxième forme de référence pour représenter les liens entre les deux entités BC et Produit: les BC et les produits sont stockés dans des documents différents et chacun fait référence à l'autre (une commande fait référence à ses produit et inversement).

Principaux composants de Mongodb

Mongodb est un système de bases de données non-relationnel orienté document. On distingue principalement les trois composants suivants :

1- Base de données
2- Collection
3- Document

Principaux composants

1- `Base de données` : Il constitue le composant principal de MongoDB. Une base de données Mongo est un conteneur des structures de données appelées collections. Une base de données Mongo gère ses propres fichiers physiques. Un serveur MongoDB héberge plusieurs bases de données

2- `Collection` : Une collection est un ensemble de documents. Par analogie, une collection représente une table dans une base de données relationnelle. Une collection est unique mais plusieurs collections peuvent être présentes en même temps dans une base de données Mongo.
Une collection n'est pas soumise au respect d'un schéma conceptuel de données comme dans le cadre d'une base de données relationnelle.

Principaux composants

3- `Document` : Un document constitue l'unité de base dans une base de données Mongo. Un document est composé d'un ensemble de clés/valeurs. A l'inverse d'une base de données classique les documents dans une collection peuvent contenir des champs différents

Exemple d'un document dans MongoDb

Un document : {"_id":1, "nom": "Dupond",
"age" :25,
"tél" : 06-45-78-09-12"}
Un autre document : {"_id":2,"nom": "Marion",
"Age" :35,
"tél" : 07-45-78-23-12",
"adresse" : {
"n°" : "15",
"nom" : "rue du Lila",
"CP" : 75000,
"Ville" : "Paris" }

Chaque document a un identifiant. MongoDB est sensible au type et à la casse (majuscule, minuscule)

Stockage de documents : JSON

MongoDB stocke les documents dans le langage **JSON** (JavaScript Object Notation). JSON est un langage standard utilisé dans des applications pour échanger des données de la même façon que le langage XML.
Au niveau interne MongoDB représente les documents JSON en utilisant un format d'encodage binaire appelé **BSON** (Binary JSON). BSON ajoute à JSON la prise en charge de quelques types de données tels que Date et Binaire qui ne sont pas pris en charge dans JSON.

Types de données

1. String
2. Integer
3. Boolean
4. Double
5. Arrays
6. Object
7. Date
8. Object ID (pour stocker les ID des documents)
9. Binary
10. JavaScript code

Lancement et choix de la base de données

mongo Cette commande permet de lancer Mongo (équivalent à sqlplus dans Oracle). Le client essaie de se connecter au serveur Mongo. Au cas où le nom de la base de données n'est pas spécifié au lancement du client, ce dernier choisit une base de données par défaut appelée **Test**

use ExBd Cette commande pointe, si elle existe, sur la base de données **ExBd**. Dans le cas contraire mongo crée la base de données. *Dans MongoDB pas de création explicite des bases de données. Une collection est créée lors de l'insertion d'un document*

show dbs Cette commande permet d'afficher la liste des bases de données existantes dans Mongodb. `Les bases de données listées sont celles qui contiennent au moins une collection`

Insertion de documents dans une collection

Les documents Mongodb sont spécifiés dans le format JSON.

insert la commande suivante permet d'insérer dans la collection *personnes* un document dans la base de données :
db.personnes.insert({"nom":"Martin","age":30})

Consultation de tous les documents d'une collection

find la commande suivante permet de consulter l'ensemble des documents dans **personnes** : db.personnes.find().

La réponse de Mongodb est retournée comme suit :
{"_id":ObjectId("550744..."),"nom":"Martin","age":30}
`_id :  représente l'identifiant du document`

Consultation d'un document spécifique dans une collection

find la commande suivante permet de consulter le(s) document(s) contenant le nom *Martin* dans la base de données : db.personnes.find({"name":"Martin"}).

La réponse de Mongodb est retournée comme suit :
{"_id":ObjectId("550744..."),"nom":"Martin","age":30}
`_id :  représete l'identifiant du document`

Consultation d'un document spécifique dans une collection

find la commande suivante permet de consulter le(s) document(s) contenant le nom *Martin* et l'age 30 dans la base de données : db.personnes.find({"name":"Martin","age":30}).

La réponse de Mongodb est retournée comme suit :
{"_id":ObjectId("550744..."),"nom":"Martin","age":30}
`_id : représente l'identifiant du document`

Consultation suivi d'une projection d'un document spécifique dans personnes

find la commande suivante permet de consulter le(s) document(s) dont le nom est *Martin* et afficher seulement leur age :
db.users.find({"name":"Martin"},"age":1).

La réponse de Mongodb est retournée comme suit :
{"_id":ObjectId("550744..."),"age":30}
`_id :  représente l'identifiant du document`

Consultation utilisant un interval: $gt,$gte,$lt,$lte

find la commande suivante permet de consulter le(s) personnes dont l'age est supérieur à 20 et afficher leur nom:
db.users.find({"age":{$gt : 20}},"nom":1).
La réponse de Mongodb est retournée comme suit :
{"_id":ObjectId("550744..."),"nom":"Martin"}
_id : représente l'identifiant du document

Insertion d'un deuxième document dans personnes

insert On insère un autre document dans la collection *personnes* :

```
db.personnes.insert({"nom":"Dupont","age":40})
```

Consultation utilisant des opérateurs logiques: $or,$and,$not

find la commande suivante permet de consulter le(s) document(s) dont l'age est supérieur à 20 et inférieur à 50. On affiche le nom:
db.personnes.find({$and:["age":{$gt : 20},{"age":{$lte : 50}}]},"nom":1).
La réponse de Mongodb est retournée comme suit :
{"_id":ObjectId("550745..."),"nom":"Dupont"}
_id : représete l'identifiant du document

Consultation utilisant des opérateurs logiques: $or,$and,$not

find la commande suivante permet de consulter le(s) document(s) des personnes dont les noms ne commencent pas D. On affiche leur age :
db.personnes.find({"nom":{$not:/D/},"age":1).
La réponse de Mongodb est retournée comme suit :
{"_id":ObjectId("550744..."),"age":30}
`_id :  représente l'identifiant du document`

Mise à jour de documents

update la commande suivante permet de mettre à jour de(s) document(s) dans une collection.
L'exemple suivant modifie l'age de Martin à 20 (option set).

```
db.personnes.update({"nom":"Martin"},{$set:{"age":20}})
```

Pour vérifier le changement il suffit de lancer la commande find.

Mise à jour de documents

update L'exemple suivant supprime le champs age de Martin dans la base de données (option unset).

```
db.personnes.update({"nom":"Martin"},{unset : {age:1}})
```

Mise à jour de documents

update L'exemple suivant ajoute le champs age à Martin dans la base de données (option push).

```
db.personnes.update({"nom":"Martin"},{push : {age:1}})
```

Suppression de documents

remove la commande suivante permet de supprimer tous les document(s) dans la collection personnes.

```
db.personnes.remove()
```

Suppression de documents

remove la commande suivante permet de supprimer seulement les personnes dont l'age est supérieur à 30.

```
db.personnes.remove({"age":{$gt:20}})
```

Manipulation des types complexes : Arrays

insert la commande suivante permet d'insérer un document contenant un champ de type Array dans la collection *resto*.

```
db.resto.insert({"menu":"tomate","fromage","beurre"})
```

Nous insérons un deuxième document dans *resto* :

```
db.resto.insert({"menu":"pizza","frites","beurre"})
```

Recherche dans un Array

insert la commande suivante permet de rechercher un menu contenant du fromage.

```
db.resto.find({"menu":"fromage"})
```

la commande suivante permet de rechercher un menu contenant les deux ingrédients fromage et beurre.

```
db.resto.find({"menu":{$all:["fromage","beurre"]}})
```

Document imbriqué

- Un document imbriqué est utilisé pour organiser les données dans une structure de données non plate.

insert la commande suivante permet d'insérer un document imbriqué dans la collection *auteur*. Dans cet exemple *Durand* a deux livres.

```
db.auteur.insert({ "ida": "a0", "nom": "Durand", "prenom":
"Robert", "livres": [{ "idl": "l1", "thème": "policier" }, {
"idl": "l2","thème": "SF" }]}
```

Recherche dans un document imbriqué

insert la commande suivante permet de rechercher les auteurs des livres du thème Policier.

```
db.auteur.find({"livres.thème":"Policier"},{"nom":1})
```

Insertion d'un document imbriqué

insert la commande suivante permet d'insérer un document imbriqué dans la collection *auteur*. Dans cet exemple on assigne le document imbriqué a une variable Mongodb. Ceci est possible puisque Mongodb supporte JavaScript.

```
x= { "ida": a1, "nom": "Martin", "prenom": "Sophie",
"livres": [{ "idl": "l1", "thème": "policier" }, { "idl":
"l3","thème": "Histoire" }]}

db.auteur.insert(x)
```

Utilisation de fonctions spécifiques

pretty Cette fonction appliquée à une requête de consultation find permet de mieux présenter le résultat de recherche.

```
db.auteur.find({"livres.thème":"Policier"},{"nom":1}).pretty()
```

Utilisation de fonctions spécifiques

limit Cette fonction appliquée à une requête de consultation find permet de limiter le nombre de résultats retournés.

```
db.auteur.find({"livres.thème":"Policier"},{"nom":1}).limit(2)
```

Utilisation de fonctions spécifiques

sort Cette fonction appliquée à une requête de consultation find permet de trier le résultat selon les valeurs d'un champs (ou plusieurs).

```
db.auteur.find({"livres.thème":"Policier"},
{"nom":1}).sort({"nom":1})
```

Utilisation de fonctions spécifiques

count Cette fonction appliquée à une requête de consultation find permet de compter le nombre de documents retournés dans le résultat.

```
db.auteur.find({"livres.thème":"Policier"},{"nom":1}).count()
```

Utilisation des Agrégats

L'agrégation dans Mongodb permet de traiter successivement un ensemble d'opérations : le résultat retourné par la première opération est récupéré par l'opération suivante qui effectue un traitement et renvoie le résultat à l'opération suivante formant ainsi un *pipeline* : la sortie d'une opération est l'entrée de l'opération suivante. La dernière opération retourne le résultat final de l'agrégation.

Utilisation des Agrégats

Plusieurs opérations sont utilisées dans une agrégation. Nous distinguons les opérations suivantes :

1. $match
2. $project
3. $group
4. $unwind
5. $sort
6. $limit
7. $lookup

Utilisation des Agrégats

Opération $match:

Cette opération permet de filtrer les documents d'une collection selon un ou plusieurs critères

Exemple : dans la collection auteur je m'intéresse aux livres de thème Policier.

db.auteur.agregate({"$match":{"livres.theme:"Policier"} })

Utilisation des Agrégats

Opération $project:
Cette opération permet de projeter les documents sélectionnés dans une collection sur un ou plusieurs champs

Exemple : dans la collection auteur je m'intéresse aux noms des auteurs des livres de thème Policier.
db.auteur.agregate({"$match":{"livres.theme:"Policier"} },
{"$project":"nom":1 })

Utilisation des Agrégats

Opération $group:

Cette opération permet de regrouper les documents d'une collection selon les valeurs d'un ou plusieurs champs et d'appliquer ensuite une fonction d'agrégat sur les groupes

Exemple : dans la collection auteur on cherche à connaitre le nombre de livres par thème concernant l'auteur *Martin*.

```
db.auteur.agregate({"$match":{"nom:"Martin"} },
{"$project":"livres.theme":1 },
{"$group":{"_id":"$livres.theme","total":{sum:1} } })
```

Utilisation des Agrégats

Opération $sort:

Cette opération permet de trier le résultat en ordre croissant ou décroissant selon un ou des champs du résultat

Exemple : On souhaite trier le résultat en ordre croissant selon le nombre total de livres par thème.

```
db.auteur.agregate({"$match":{"nom:"Martin"} },
{"$project":"livres.theme":1 },
{"$group":{"_id":"$livres.theme","total":{sum:1}},{"$sort":{"total":1
} } })
```

Utilisation des Agrégats

Opération $limit:

Cette opération permet de limiter le nombre de réponse dans le résultat

Exemple : On souhaite obtenir seulement les deux thèmes les plus important parmi les livres publiés par Martin..

db.auteur.agregate({"$match":{"nom:"Martin"} },
{"$project":"livres.theme":1 },
{"$group":{"_id":"$livres.theme","total":{sum:1}},{"$sort":{"total":1
}},{"$limit":2 } })

Utilisation des Agrégats

Opération $unwind:

Cette opération permet de remplacer un document contenant un champs de type **Array** par une suite de documents contenant chacun un seul élément du tableau

Exemple : On souhaite obtenir seulement les deux thèmes les plus important parmi les livres publiés par Martin..

db.auteur.agregate([{"$unwind":"$livres.theme"}]) ???

Utilisation des Agrégats

Opération $lookup:
Cette commande permet de faire la jointure entre deux collections. Pour cela il faut utiliser cette commande dans une agrégation comme suit :

Exemple : On souhaite faire la jointure entre deux collections auteur (idlivre) et livre (numlivre)

```
db.auteur.aggregate([ {"$lookup" : { "localField" : "idlivre",
"from" : "livre", "foreignField" : "numlivre", "as" : "titre" } },
{"$out" : "resultat"}] )
```

Le résultat de cette requête se trouve dans la collection résultat contenant tous les auteurs avec les titres de leurs livres.

D'autres commandes Voir la documentation disponible sur le site MongoDB à l'adresse suivante :

https://www.mongodb.com/docs/