

Master 1 MIAAGE - BDA
Modèle de données Document
Cas d'étude : MongoDB

Marta Rukoz

Modèle Document

- Cas particulière du modèle clé-valeur où la valeur a une structure de données complexe connue en tant que document.
- Un document structure arborescente
- Peu de restrictions sur les valeurs tant qu'elle répond à l'exigence fondamentale d'être exprimable en tant que document.
- Principalement deux formats : **XML** et **JSON**.
- Deux représentations : **sérialisée et arborescente**
- Base de données : composée par des collections de documents
- Les documents sont contenus dans des collections, qui correspondent plus ou moins aux tables des SGBDR
- Les membres d'une collection ne partagent pas nécessairement la même structure

Modèle Document

Il existe deux représentations d'un document

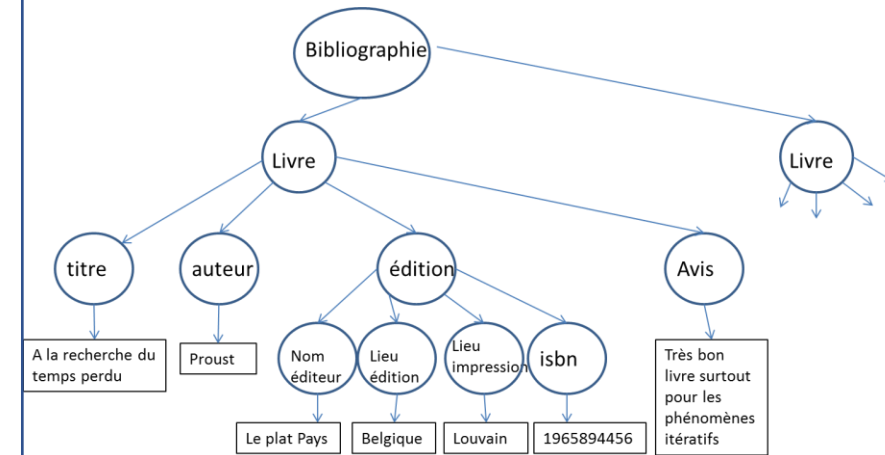
- **Forme sérialisée : c'est la forme courante,**
où le contenu est marqué par des accolades ouvrante/fermante.
- **Forme arborescente :**
structure du document.

Modèle Document : XML (*Extensible Markup Language*)

Représentation séquentielle

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bibliographie SYSTEM "EX1A-bibliographie.dtd">
<bibliographie>
  <livre>
    <titre>A la recherche du temps perdu</titre>
    <auteur>Proust</auteur>
    <edition>
      <nom_editeur>Le plat pays</nom_editeur>
      <lieu_edition>Belgique</lieu_edition>
      <lieu_impression>Louvain</lieu_impression>
      <isbn>1965894456</isbn>
    </edition>
    <avis>Tres bon livre surtout pour les phénomènes itératifs</avis>
  </livre>
  <livre>
    <titre>La La</titre>
    <auteur>Proust</auteur>
    <edition>
      <nom_editeur>Le plat pays</nom_editeur>
      <lieu_edition>Belgique</lieu_edition>
      <lieu_impression>Louvain</lieu_impression>
      <isbn>1965894456</isbn>
    </edition>
  </livre>
</bibliographie>
```

Représentation arborescente



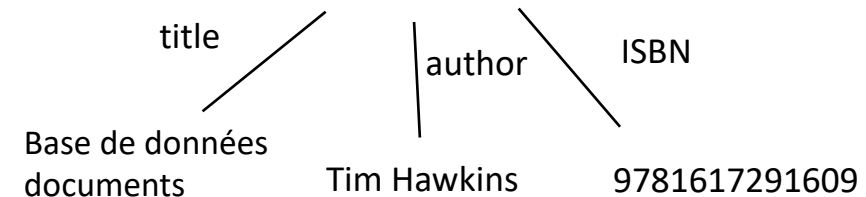
Le modèle document : JSON (*JavaScript Object Notation*)

sérialisée

```
{"title": "Base de données documents",  
"author": "Tim Hawkins", "ISBN":  
"9781617291609"}
```

Cette représentation permet le stockage et l'échange de documents

arborescente



Le arêtes sont étiquetées par les clés

Il est plus facile de raisonner sur la forme arborescente pour concevoir des traitements

Structure de base d'un Document JSON (Javascript Object Notation) (1)

- *Les paires (clé, valeur) : structure de base*
- *Valeur simple ou atomique*
 - *"title": "Base de données documents"*
- *Valeurs complexe ou objet* : La construction (clé, valeur) permet de créer des agrégats (ensemble de paires (clé, valeur), chaque clé étant unique). En JSON les agrégats sont codés avec des accolades ouvrante/fermante.
 - *{"title": "Base de données documents", "author": "Tim Hawkins", "ISBN": "9781617291609"}*

Structure de base d'un Document JSON

(Javascript Object Notation) (2)

- *Valeurs complexe ou objet*
 - On peut créer une paire clé-valeur dans laquelle la valeur est un agrégat, et imbriquer les agrégats les uns dans les autres.
 - `{"title": "Base de données documents", "author": {"firstname": "Tim", "lastname": "Hawkins"}, "ISBN": "9781617291609"}`
- *imbrication de listes : les tableaux*
 - `{"title": "Base de données documents", "authors": ["Tim Hawkins", "Kyle Banke"], "ISBN": "9781617291609"}`
- *Un document* : des agrégats d'agrégats et/ou des tableaux , sans limitation de profondeur.

Modèle relationnel Vs modèle document

Bases de données relationnelles

- Quelles données stocker sans nécessairement savoir comment les utiliser ;
- Comment stocker les données.
- Coût de la flexibilité des requêtes payé d'avance sur le stockage.

Bases de données document

- Comment seront utilisés les données sans nécessairement connaître toutes les données qui seront stockées.
- Des modifications fondamentales au "schéma" sont faites à la volée,
- Décisions de conception peuvent-être payées plus tard.

Modèle relationnel Vs modèle document

Agence

NAgence	Nom	Ville	Directeur
A1	Paris1	Paris	Keller
A2	Lyon1	Lyon	Martin

Compte

NCompte	Nagence	Solde
310	A1	1500
315	A1	20000
440	A2	600

Compte-Client

Ncompte	Nclient
310	C1
310	C2
315	C1
440	C2

Client

Nclient	Nom	Adresse
C1	Peters	X Av.
C2	Martin	Ad2

On veut traiter les comptes de chaque agence

Représentation des entités

Agence

NAgence	Nom	Ville	Directeur
A1	Paris1	Paris	Keller
A2	Lyon1	Lyon	Martin

```
" Agence": [ {" NAgence": " A1",  
              " Nom ": " Paris1 ",  
              " Ville ": " Paris ",  
              " Directeur ": " Keller"  
            },  
            {" NAgence": " A2",  
              " Nom ": " Lyon1 ",  
              " Ville ": " Lyon ",  
              " Directeur ": " Martin"  
            }  
          ]
```

Représentation des associations (1:N)

Compte

NCompte	Nagence	Solde
310	A1	1500
315	A1	20000
440	A2	600

```
" Agence": [  
    {" NAgence": " A1",  
      " Nom ": " Paris1 ",  
      " Ville ": " Paris ",  
      " Directeur ": " Keller « ,  
      « Comptes »: [ {« Ncompte »: « 310 », « Solde »: 1500},  
                      {« Ncompte »: « 315 », « Solde »: 20000},  
                      ]  
    },  
    {" NAgence": " A2",  
      " Nom ": " Lyon1 ",  
      " Ville ": " Lyon ",  
      " Directeur ": " Martin ",  
      « Comptes »: [ {« Ncompte »: « 440 », « solde »: 600 } ]  
    }  
  ]
```

Représentation des associations (N:N)

```
" Agence": [
  {" NAgence": " A1", " Nom ": " Paris1 ", " Ville ": " Paris ",
    " Directeur ": " Keller « ,
    « Comptes »: [ {« Ncompte »: « 310 »,
      « Solde »: 1500,
      « Client »: [ {« Nclent »: C1,
        « Nom »: «Peters »,
        « Adresse »: 'xAv.' },
        {« Nclent »: C2,
        « Nom »: « Martin»,
        « Adresse »: 'Ad2 },
      ] },
    {« Ncompte »: « 315 », « Solde »: 20000,
    « Client »: [ {« Nclent »: C1,
      « Nom »: «Peters »,
      « Adresse »: 'xAv.' } ] } ] ]
},
```

Compte-Client

Ncompte	Nclient
310	C1
310	C2
315	C1
440	C2

Client

Nclient	Nom	Adresse
C1	Peters	X Av.
C2	Martin	Ad2

```
 {" NAgence": " A2",
  " Nom ": " Lyon1 ",
  " Ville ": " Lyon ",
  " Directeur ": " Martin ",
  « Comptes »: [ {« Ncompte »: « 440 »,
    « solde »: 600 ,
    « Client »: [{« Nclent »: C2,
      « Nom »: « Martin»,
      « Adresse »: 'Ad2 } ] } ]
}]
```

Le modèle Document

- Pour plus de référence sur la syntaxe JSON : <http://www.json.org/>. Pour valider si un document JSON est bien formé vous pouvez utiliser l'outil de validation en ligne <http://jsonlint.com/>
- Une proposition de schéma pour JSON : <http://json-schema.org/> Un langage de requêtes JSON : JAQL, <http://code.google.com/p/jaql/>

MongoDB

- SGBD orienté document : hiérarchie de paires clé-valeur, sans contrainte de présence ou de quantité.
- Open source développée par MongoDB Inc depuis 2007
- Nom provenant de l'anglais "humongous" ("énorme")
- Ecrit en C++
- Aucune contrainte entre les documents
- Stockage des documents au format BSON (Binary Serialized dOcument Notation ou Binary JSON) – mais structure visible par les utilisateurs en JSON
- Document BSON = unité de stockage (~ une ligne dans une BDR)
- Les documents sont contenus dans des *collections* (~table en relationnel)
 - Tout document appartient à une collection et a un champ appelé **_id** qui identifie le document dans la base de données

MongoDB

```
{ "_id" : ObjectId("5c44d53dac02b79807b1c1ac"),  
  "titre" : "A la recherche du temps perdu",  
  "auteur" : "Proust",  
  "edition" :  
    { "nom_editeur" : "Le plat pays",  
      "lieu_edition" : "Belgique",  
      "lieu_impression" : "Louvain",  
      "isbn" : "1965894456" },  
  "annee" : "1990",  
  "avis" : "Tres bon livre surtout pour les phenomenes iteratifs",  
  "prix" : "1990"  
}
```

Id est un nombre à 12 bytes qui assure l'unicité de chaque document

temps courant	Identifiant de la machine	identifiant du processus MongoDB	valeur d'un compteur incrémental
4 bytes	3 bytes	2 bytes	3 bytes

MongoDB : Modèle de données

- En général, regroupement de toutes les données relatives à un objet dans un même "document"
 - ⇒ nombre de jointures réduit et
 - ⇒ impact positif sur les performances (toutes les données sont récupérées en une seule lecture)
- Modèle des documents pouvant varier de structure dans une même collection
- Depuis la version 3.2, MongoDB offre la possibilité d'associer un schéma à une collection et de contrôler que les documents insérés sont conformes au schéma.
- Schéma dynamique : possibilité d'ajouter de nouveaux champs sans affecter les autres documents
- Schéma flexible mais conçu selon le type de requêtes

MongoDB : modèle de données

Relation implémentée
par une imbrication de
documents

```
{
  "_id": ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address": [
    {
      "building": "22 A, Indiana Apt",
      "pincode": 123456,
      "city": "Los Angeles",
      "state": "California"
    },
    {
      "building": "170 A, Acropolis Apt",
      "pincode": 456789,
      "city": "Chicago",
      "state": "Illinois"
    }
  ]
}
```

MongoDB : modèle de données

Relation implémentée
par l'utilisation de
référence

```
{
  "_id": ObjectId("52ffc4a5d85242602e000000"),
  "building": "22 A, Indiana Apt",
  "pincode": 123456,
  "city": "Los Angeles",
  "state": "California"
}
```

```
{
  "_id": ObjectId("52ffc33cd85242f436000001"),
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin",
  "address_ids": [
    ObjectId("52ffc4a5d85242602e000000"),
    ObjectId("52ffc4a5d85242602e000001")
  ]
}
```

MongoDB : Schémas flexibles

- Documents dans une collection peuvent avoir différents schémas.
- Dans un document on peut :
 - ajouter de nouveaux champs,
 - supprimer des champs existants ou changer les valeurs des champs pour un nouveau type,
- il suffit de mettre à jour les documents en fonction de la nouvelle structure.
- Chaque document peut correspondre aux champs de données de l'entité représentée, même si le document présente des variations substantielles par rapport aux autres documents de la collection

MongoDB : Schémas flexibles

- En pratique, les documents d'une collection partagent une structure similaire.
- On peut appliquer des **règles de validation** des documents d'une collection lors des opérations de mise à jour et d'insertion.

Créer une collection

(Collections de taille fixe)

capped collection

Une collection de taille fixe qui écrase automatiquement ses entrées les plus anciennes lorsqu'elle atteint sa taille maximale.

```
db.createCollection(name, options)
```

```
db.createCollection("students", { capped : true, size : 5242880, max : 5000 } )
```

Si **size** est plus petit ou égal à 4096, alors la collection aura un plafond de 4096 octets. Dans le cas contraire, MongoDB augmentera la taille fournie pour en faire un multiple entier de 256.

max indique le nombre maximal de documents pour la collection. Si une collection atteint la limite de taille maximale avant d'atteindre le nombre maximal de documents MongoDB supprime les documents les plus anciens.

MongoDB : opérations **C**RUD

```
db.users.insertOne(  ← collection
{
  name: "sue",        ← field: value
  age: 26,            ← field: value
  status: "pending"   ← field: value
}
)                    } document
```

MongoDB : opérations CRUD

- Insertion d'un document avec `_id` (ObjectId)généré par MongoDB

```
db.products.insert( { item: "card", qty: 15 } )
```

```
{ "_id" : ObjectId("5063114bd386d8fadbd6b004"), "item" : "card", "qty" : 15 }
```

- Insertion avec identificateur à l'insertion

```
db.products.insert( { _id: 10, item: "box", qty: 20 } )
```

```
{ "_id" : 10, "item" : "box", "qty" : 20 }
```

MongoDB : opérations CRUD

- Possibilité de mixer la gestion des `_id`

```
db.products.insert(  
  [  
    { _id: 11, item: "pencil", qty: 50, type: "no.2" },  
    { item: "pen", qty: 20 },  
    { item: "eraser", qty: 25 }  
  ]  
)
```

```
{ "_id" : 11, "item" : "pencil", "qty" : 50, "type" : "no.2" }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a0"), "item" : "pen", "qty" : 20 }  
{ "_id" : ObjectId("51e0373c6f35bd826f47e9a1"), "item" : "eraser", "qty" : 25 }
```


MongoDB : opérations CRUD

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

MongoDB : opérations CRUD

```
db.inventory.find( {} )
```

```
SELECT * FROM inventory
```

```
db.inventory.find( { status: "D" } )
```

```
SELECT * FROM inventory WHERE status = "D"
```

```
db.inventory.find( { status: { $in: [ "A", "D" ] } } )
```

```
SELECT * FROM inventory WHERE status in ("A", "D")
```

```
db.inventory.find( { status: "A", qty: { $lt: 30 } } )
```

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

MongoDB : opérations CRUD

```
db.inventory.find( { status: "A" }, { item: 1, status: 1 } )
```

```
SELECT _id, item, status from inventory WHERE status = "A"
```

```
db.inventory.find( { status: "A" }, { item: 1, status: 1, _id: 0 } )
```

```
SELECT item, status from inventory WHERE status = "A"
```

Attention : Pas d'erreur en cas de faute de frappe sur des noms d'attributs


MongoDB : opérations CRUD

```
db.users.updateMany(  
  { age: { $lt: 18 } },  
  { $set: { status: "reject" } }  
)
```

← collection
← update filter
← update action

MongoDB : opérations CRUD

```
db.users.deleteMany(  
  { status: "reject" }  
)
```



collection

delete filter

MongoDB : atomicité

- Ecriture d'un document est une opération atomique
- Ecriture de plusieurs documents est atomique pour chacun des documents, mais pas pour l'ensemble (`db.collection.updateMany()`)
- **Version 4.0** : Transactions multi-documents sur les ensembles répliques (réplica sets) dans, MongoDB
- **Version 4.2** : Transactions distribuées avec transactions multi-documents sur les clusters *sharded* et transactions multi-documents sur le réplica *sets*

MongoDB : Isolation

- Read Uncommitted
- Read Uncommitted and Single Document Atomicity : concurrent read operations may still see an updated document before the changes are made durable.
- Read Uncommitted And Multiple Document Write :

MongoDB en pratique

Pou installer MogoDB sur :

C:\ProgramFiles\MongoDB\Server\3.4\

Pour installer MongoDB <https://docs.mongodb.com/v3.4/tutorial/install-mongodb-on-windows/>

Téléchargez-le et décompressez l'archive récupérée

Vous devez ajoutez un répertoire nommé mongo, il est recommandé de mettre sa localisation dans Environment Variables :

MONGO_HOME = c:\Program Files\MongoDB\server\3.6

Mongodb = %MONGO_HOME%\bin

MongoDB travail dans une approche classique Client/Serveur

MongoDB serveur

Pour utiliser MongoDB vous devez lancer le serveur, dans un terminal exécutez :

```
>mongod
```

Le serveur démarre et on obtient entre autres choses :
:

Répertoire où les données seront stockées

2019-01-15T20:04:16.675+0100 I CONTROL [initandlisten]

2019-01-15T20:04:17.908+0100 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'C:/data/db/diagnostic.data'

2019-01-15T20:04:17.913+0100 I NETWORK [initandlisten] waiting for connections on port 27017

Les connections seront sur le port 27017

```
>mongod --port 28008 --dbpath /data/db % Démarre MongoDB dans le port 28008
```

MongoDB Client

Pour l'application client, vous avez deux options : l'interpréteur de commande mongo ou une application graphique, RockMongo, RoboMongo, etc.

Interpréteur de commande

Dans un autre terminal de commandes, En ligne de commande, on démarre le client : mongo

Cet outil est un interpréteur javascript et on peut donc lui soumettre des instructions en Javascript, ainsi que des commandes propres à MongoDB.

```
> mongo
```

```
MongoDB shell version v3.6.0
```

```
connecting to: mongodb://127.0.0.1:27017
```

```
MongoDB server version: 3.6.0
```

Avant d'initier MongoDB shell, il faut d'abord démarrer mongod

Références

- Site officiel : <https://www.mongodb.com/fr>
- Documentation : : <https://docs.mongodb.com/manual/introduction/>
- Console en ligne :
https://www.tutorialspoint.com/mongodb_terminal_online.php
- Tutoriels :
 - <https://docs.mongodb.com/manual/tutorial/>
 - <http://b3d.bdpedia.fr/mongodb.html>
 - <https://stph.scenari-community.org/contribs/nos/Mongo1/co/presentation.html>
 - https://www.tutorialspoint.com/mongodb/mongodb_overview.htm

Références

- Brad Dayley, Brendan Dayley and Caleb Dayley. Node.js, MongoDB and Angular Web Development. Second Edition, Addison-Wesley, 2018.
- Kyle Banker, Peter Bakkum, Shaun Verch, Douglas Garrett and Tim Hawkins. MongoDB is action. Manning Publications, 2016.
- Luc Perkins, Eric Redmon and Jim R. Wilson. Seven Databases in Seven Weeks. Second Edition. A guide to modern databases and the NoSQL movement. The pragmatic Bookshelf. Edited by Jacquelyn Carter, 2018.
- Rudi Bruchez. Les bases de données NoSQL: Comprendre et mettre en oeuvre. Eyrolles éditions, 2013.