

## BD ORIENTÉE DOCUMENT (1)

- Elles stockent une collection de « **documents** »
- elles sont basées sur le modèle « clé-valeur » mais la valeur est un document en **format semi-structuré hiérarchique** de type **JSON** ou **XML** (possible aussi de stocker n'importe quel objet, via une sérialisation)
- les **documents** n'ont **pas de schéma**, mais une **structure arborescente** : ils contiennent une liste de champs, un champ a une valeur qui peut être une liste de champs, ...
- elles ont généralement une **interface d'accès HTTP REST** permettant d'effectuer des requêtes (plus complexe que l'interface CRUD des BD clés/valeurs)
- Implémentations les plus connues :
  - **CouchDB** (fondation Apache)
  - **RavenDB** (pour plateformes « .NET/Windows » - LINQ)
  - **MongoDB, Terrastore, ...**

26/09/2020

1

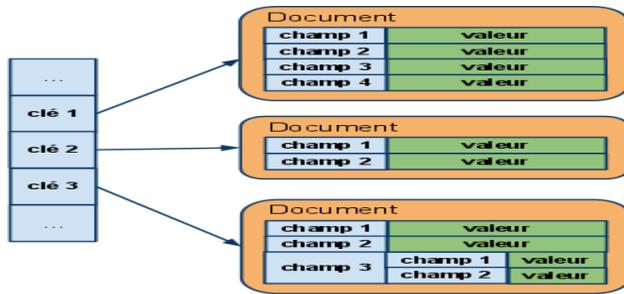
## BD ORIENTÉE DOCUMENT (2)

- Un **document** est composé de **champs** et **des valeurs associées**
- ces **valeurs** :
  - peuvent être **requêtées**
  - sont soit d'un **type simple** (entier, chaîne de caractère, date, ...)
  - soit elles mêmes **composées** de plusieurs couples clé/valeur.
- bien que les documents soient structurés, ces BD sont dites "**schemaless**" : il n'est **pas nécessaire de définir au préalable les champs** utilisés dans un document.
- les documents peuvent être très **hétérogènes** au sein de la BD
- permettent **d'effectuer des requêtes sur le contenu** des documents/objets : pas possible avec les BD clés/valeurs simples
- Elles sont principalement utilisées dans le **développement de CMS** (Content Management System - outils de gestion de contenus).

26/09/2020

2

## BD ORIENTÉE DOCUMENT: « EXEMPLE »



26/09/2020

3

## BD ORIENTÉE DOCUMENT: « FORCES ET FAIBLESSE »

- **Forces :**
  - modèle de données simple mais puissant (expression de structures imbriquées)
  - bonne mise à l'échelle (surtout si sharding pris en charge)
  - pas de maintenance de la BD requise pour ajouter/supprimer des « colonnes »
  - forte expressivité de requêtage (requêtes assez complexes sur des structures imbriquées)
- **Faiblesses :**
  - inadaptée pour les données interconnectées
  - modèle de requête limitée à des clés (et indexes)
  - peut alors être lent pour les grandes requêtes (avec MapReduce)

26/09/2020

4

## BD ORIENTÉE DOCUMENT: «UTILISATIONS PRINCIPALES »

- Les BD NoSQL de type « Document » principalement utilisées pour
  - Enregistrement d'événements
  - Systèmes de gestion de contenu
  - Web analytique ou analytique temps-réel
  - Catalogue de produits
  - Systèmes d'exploitation
  - ...

26/09/2020

5

## BD MONGODB

26/09/2020

6

## QU'EST-CE QUE MONGODB?

MongoDB, un SGBD "NoSQL", l'un des plus populaires

- fait partie des NoSQL dits "documentaires" (avec CouchDB)
- s'appuie sur un modèle de données semi-structuré (encodage JSON) ;
- pas de schéma (complète flexibilité) ;
- **un langage d'interrogation** original (et spécifique) ;
- pas (ou très peu) **de support transactionnel**.

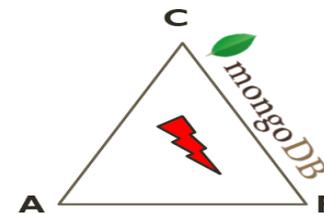
Construit dès l'origine comme un système **scalable** et **distribué**.

- distribution par partitionnement (sharding) ;
- technique adoptée : découpage par intervalles (type BigTable, Google) ;
- tolérance aux pannes par réplication.

26/09/2020

7

## MONGODB : THÉORÈME DE CAP?



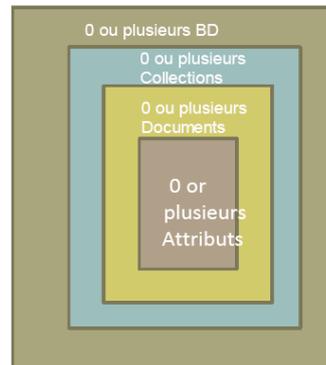
MongoDB met l'accent sur la consistance et la tolérance aux pannes

26/09/2020

8

## MONGODB: HIÉRARCHIE DES OBJETS

- Une instance MongoDB peut avoir zéro ou plusieurs « bases de données »
- Une base de données peut contenir zéro ou plusieurs «collections».
- Une collection peut contenir zéro ou plusieurs «documents».
- Un document peut avoir un ou plusieurs «champs».
- Les index de MongoDB fonctionnent comme leurs homologues SGBDR



26/09/2020

9

## COMPARAISON BASES RELATIONNELLES ET MONGODB

BDR		MongoDB
Base de donnée	➔	Base de données
Table, Vue	➔	Collection
Tuple	➔	Document (JSON, BSON)
Colonne	➔	Champs
Index	➔	Index
Jointure	➔	Document imbriqués
Clé étrangère	➔	Référence
Partition	➔	Shard

26/09/2020

## MONGODB: « PROCESSUS ET CONFIGURATION »

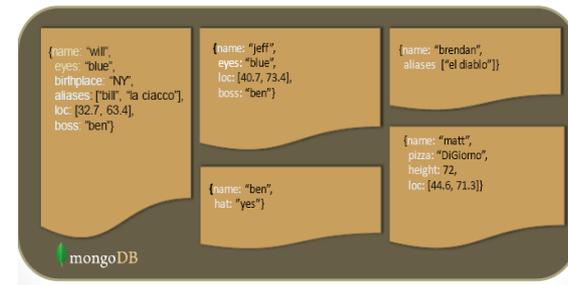
- Le processus Mongod: instance de la base de données
- Le processus Mongos: Sharding processus
  - Analogue à un routeur de base de données.
  - Traite toutes les requêtes.
  - Décide le nombre de processus mongos qui doivent recevoir la requête.
  - Mongos rassemble les résultats et les renvoie au client.
- Le processus Mongo: un shell interactif (le client)
  - Un environnement JavaScript pour l'utiliser avec MongoDB.
- Vous pouvez avoir un processus mongos pour tout le système, peu importe le nombre d'instance « mongod».
- Ou vous pouvez avoir un processus mongos local pour chaque client si vous voulez minimiser la latence du réseau.

26/09/2020

11

## MONGODB: « PAS DE SCHÉMA »

- MongoDB n'a pas besoin de schéma de données prédéfini
- Chaque document d'une collection peut avoir des données différentes



26/09/2020

12

## LE FORMAT JSON

- Les données sont dans des paires nom / valeur
- Une paire nom / valeur consiste en un nom de champ suivi de deux points, suivi d'une valeur:
  - Exemple: nom: "MERAD"
- Les données sont séparées par des virgules:
  - Exemple: nom: "MERAD", prénom: "Manel"
- Les accolades contiennent des objets
  - Exemple: {nom: "MERAD", prénom: "Manel", pays : "Algérie"}
- Un tableau est stocké entre crochets [ ]
  - Exemple [{nom: "MERAD", prénom: "Manel", , pays : " Algérie"}, {nom: "LAMI", prénom: "Yanis", pays : " Algérie"}]

26/09/2020

13

## LES OPÉRATIONS CRUD

- **Create**
  - db.collection.insert( <document> )
  - db.collection.save( <document> )
  - db.collection.update( <query>, <update>, { upsert: true } )
- **Read**
  - db.collection.find( <query>, <projection> )
  - db.collection.findOne( <query>, <projection> )
- **Update**
  - db.collection.update( <query>, <update>, <options> )
- **Delete**
  - db.collection.remove( <query>, <justOne> )

26/09/2020

14

## L'OPÉRATION CREATE

**DB.collection** spécifie la collection ou seront enregistrés les documents.

- `db.nom_collection.insert (<document>)`
  - Omettre le champ `_id` pour que MongoDB génère automatiquement la clé
  - Exemple `db.parts.insert ({type: "tournevis", quantité: 15})`
  - `db.parts.insert ({_id: 10, type: "marteau", quantité: 1})`
- `db.nom_collection.update (<requête>, <mise à jour>, {upsert: true})`
  - Mettra à jour un ou plusieurs enregistrements dans une collection satisfaisant la requête
- `db.nom_collection.save (<document>)`
  - Met à jour un enregistrement existant ou crée un nouvel enregistrement

26/09/2020

15

## L'OPÉRATION READ

- `db.collection.find( <query>, <projection> ).cursor modified`
  - Fournit des fonctionnalités similaires à la commande **SELECT**
  - `<query>` where condition , `<projection>` attributs dans le résultat.
  - Exemple: `varPartsCursor= db.parts.find({parts:"marteau"}).limit(5)`
  - la requête à un curseur pour limiter le nombre de lignes affichés par page
  - on peut modifier la requêtes en ajoutant `limits, skips, and sort orders(Tri)`.
- Pour avoir le première ligne de résultat on utilise:
  - `db.collection.findOne( <query>, <projection> )`

26/09/2020

16

## L'OPÉRATION REMOVE

- `db.nom_collection.remove(<query>,<justone>)`
  - Supprimer tous les enregistrements d'une collection qui correspondent au critère
  - `<justone>` -spécifie de supprimer seulement 1 enregistrement correspondant au critère
  - Exemple: `db.parts.remove (type: / ^ h /)` - supprime toutes les pièces commençant par h
  - `Db.parts.remove ()` -dsupprime tous les documents dans la collections de part

26/09/2020

17

## EXEMPLES- CRUD

```
> db.user.insert({_id:51
  nom: "LAIDI",
  prénom : "Ahmed",
  age: 39
})
```

```
> db.user.find ()
{ "_id" : ObjectId("51"),
  "nom" : "LAIDI",
  "prénom" : "Ahmed",
  "age" : 39
}
```

```
> db.user.update(
  {"_id" : ObjectId("51")},
  {
    $set: {
      age: 40,
      Salaire: 7000}
  }
)
```

```
> db.user.remove({
  "nom": /^L/
})
```

26/09/2020

18

## CAS PRATIQUE

Un document correspond à un film, qui aura un nom, un réalisateur et une liste d'acteurs qui ont joué dans ce film. Un acteur a un nom et un prénom

- Commencez par créer une base de données s'appelant Cinema
  - `>use Cinema`
- Insérer le film "The godfather" réalisé par "Francis Ford Coppola" dans lequel a joué "Marlon Brando", "Al Pacino" et "Robert Duvall"
  - `db.Cinema.insert({nom:"The godfather",realisateur:"Francis Ford Coppola",acteurs:[{nom:"Pacino",prenom:"Al"},{nom:"Brando",prenom:"Marlon"},{nom:"Duvall",prenom:"Robert"}]})`
- Marvel prévoit bientôt de faire un film d'avengers avec de jeunes personnages de BD dont Kamala Khan.
- Le nom du réalisateur et des acteurs qui vont jouer dans le film ne sont pas encore connus. Il nous est toutefois demandé d'ajouter cette information dans notre collection sachant que le film s'appellera "Young Avengers". On donnera l'\_id 1 au film.
  - `db.Cinema.insert({_id:"1",nom:"Young Avengers"})`

26/09/2020

19

## CAS PRATIQUE

- Nous venons d'apprendre que le film sera finalement réalisé par "Joe Johnston".
- Ajoutez le nom du réalisateur aux informations
  - `db.Cinema.update({_id:"1"},{$set:{realisateur:"Joe Johnston"}})`
- Après les premiers castings, il a été convenu que le personnage de Kamala Khan sera interprété par l'actrice pakistanaise Sanam Jhung.
- Mettez à jour les informations dans la base.
  - Nous utiliserons l'opérateur \$push qui permet d'ajouter une nouvelle valeur dans un tableau.
  - `$push:{tab:element}` pour ajouter element à la liste des éléments dans tab
  - tab est créé si il n'existe pas
  - `db.Cinema.update({_id:"1"},{$push:{acteurs:{nom:"Jhung",prenom:"Sanam"}}})`
- Ayant du mal à trouver les acteurs adaptés aux personnages, Marvel décide d'abandonner le projet.
- Nous allons donc retirer ce dernier de notre base de données
  - `db.Cinema.remove({_id:"1"})`

26/09/2020

20



## CAS 2 : CAN 2015

- Fichier can.txt : 9 équipes
- Afficher toutes les équipes
  - `db.equipes.find()`

```
BulkWriteResult({
  "writeErrors": [],
  "writeConcernErrors": [],
  "nInserted": 9,
  "nUpserted": 0,
  "nMatched": 0,
  "nModified": 0,
  "nRemoved": 0,
  "upserted": []
})
```

```
> db.equipes.find()
{ "_id": ObjectId("54c2370b5ec3bd6b8f4d3f20"), "nom": "Guinee Eq",
  "toriale", "joueurs": [ { "nom": "Ovono", "poste": "gardien",
  "best_player": false, "age": 21 }, { "nom": "Randy", "poste":
  "defenseur", "best_player": false, "age": 28 }, { "nom": "Mangu",
  "poste": "defenseur", "age": 18, "best_player": false }, {
  "nom": "Gomes", "poste": "defenseur", "best_player": false, "age
  ": 30 }, { "nom": "Bohale", "age": 27, "poste": "defenseur",
  "best_player": false }, { "nom": "Boula", "poste": "milieu", "be
  st_player": false, "age": 22 }, { "nom": "Boullia", "poste": "b
```

26/09/2020

23

- affichons le nom et l'âge de tous les joueurs de l'équipe du Sénégal
  - `db.equipes.find({nom:"Senegal"},{joueurs.nom:1,joueurs.age:1,_id:0})`

```
{ "joueurs": [ { "nom": "Bouna", "age": 32 }, { "nom": "Djilobodji", "age":
  26 }, { "nom": "Kara", "age": 25 }, { "nom": "Sane", "age": 27 }, { "nom":
  "Souare", "age": 24 }, { "nom": "Badji", "age": 24 }, { "nom": "Diop", "age":
  28 }, { "nom": "Gueye", "age": 25 }, { "nom": "Kouyate", "age": 25 }, {
  "nom": "Diouf", "age": 27 }, { "nom": "Dame", "age": 29 } ] }
```

- Afficher les équipes triées selon le classement
  - `db.equipes.find().sort({classement:1})`

26/09/2020

24

- Trier les équipes selon le classement FIFA ainsi que le nombre de bons joueurs (1/classement+nb\_Bon\_joueurs)
- `db.equipes.aggregate({{$project: {_id:0,nom:1,coeff:{$add:{{$divide:[1,"$classement"]},"$bestPlayers"]}}}, {$sort : { coeff:-1 } })`

```
{ "nom" : "Cote d'ivoire", "coeff" : 3.0357142857142856 }
{ "nom" : "Cameroun", "coeff" : 3.018867924528302 }
{ "nom" : "Tunisie", "coeff" : 2.0454545454545454 }
{ "nom" : "Senegal", "coeff" : 2.0285714285714285 }
{ "nom" : "Mali", "coeff" : 1.0204081632653061 }
{ "nom" : "Guinee", "coeff" : 0.02564102564102564 }
{ "nom" : "Cap Vert", "coeff" : 0.025 }
{ "nom" : "Congo", "coeff" : 0.01639344262295082 }
{ "nom" : "Guinee Equatoriale", "coeff" : null }
```

26/09/2020

25

## INDEX

- On peut utiliser des index pour optimiser les requêtes les plus fréquentes. L'accès aux documents indexés est ainsi bien plus rapide. Un index peut être créé sur n'importe quel attribut de documents.
  - Par défaut dans MongoDB un index est positionné sur le champ `_id`.
  - La syntaxe de création d'index est la suivante : `db.collection.ensureIndex({champs}, {options})`
  - Parmi les options possibles : `unique` : pas de doublon ; `sparse` : le champ doit être présent ;
- TTL : le document indexé a une durée de vie ; `name` : le nom de l'index ; `background` : l'indexation s'effectue en tâche de fond.
- Pour les champs, la valeur à positionner est `1` ou `-1` selon l'ordre voulu (ascendant ou descendant) des résultats.
  - Exemple: `db.parts.ensureIndex({"type": 1}, {"unique": true})`

26/09/2020

26

## RÉFÉRENCES

- Bernard ESPINASSE - Introduction aux systèmes NoSQL
- Kathleen Durant-Introduction to NoSQL and MongoDB