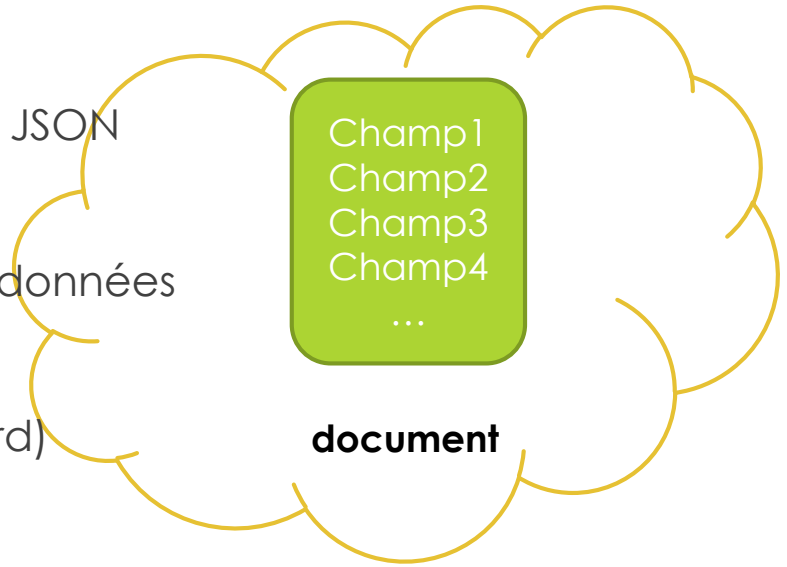
Orientée Document

MONGODB

Boutaina. hdioud@um5.ac.ma

Principes de MongoBD

- Base orientée document, OPEN SOURCE
- Représentation des informations sous forme de document, au format JSON
- Pas de définition de schéma de base obligatoire pour enregistrer les données
- Pas de relation entre les objets de la base (pas de jointure en standard)
- Il existe de nombreuses fonctions avancées comme : aggragation-framework, map-reduce,indexation plain-text, clusturing, …



Comparaison BDR/ MongoDB

BDR	MongoDB
Base de données et/ou schéma	Base de données
Table,Vue	collection
Enregistrement ou Tuple	Document (JSON,BSON)
Attribut (atomique)	propriété (chaîne, entier, tableau, structure)
Jointure	Document imbriqués
Clé étrangère	Référence
index	index

Principes: représentation des données

- **Définition : JSON**

JSON est un format de représentation logique de données, hérité de la syntaxe de création d'objets en JavaScript. C'est un format réputé léger (il ne contient pas trop de caractères de structuration), assez facilement lisible par les humains, facilement parsable par les machines, et indépendant des langages qui l'utilisent (sa seule fonction est de décrire des données, qui sont ensuite utilisées différemment pour chaque cas suivant le contexte).

Principes: représentation des données

- **Stockage d'un objet complet sous forme d'un document JSON**
  - **Exemple :**
    - SQL

id	nom	prénom	age
1	SALAHI	Mohammed	59
2	YOUSRI	Jamal	32

    - JSON

```
[{_id : 1,nom: 'SALAHI',prenom:'Mohammed',age:59},
{_id : 2,nom: 'YOUSRI',prenom:'Jamal',age:32}]
```

Relation N/N

id	marque	modele	annee
1	FORD	Mustang	1967
2	CHEVROLET	Camaro	2015
3	RENAULT	Twingo	1998

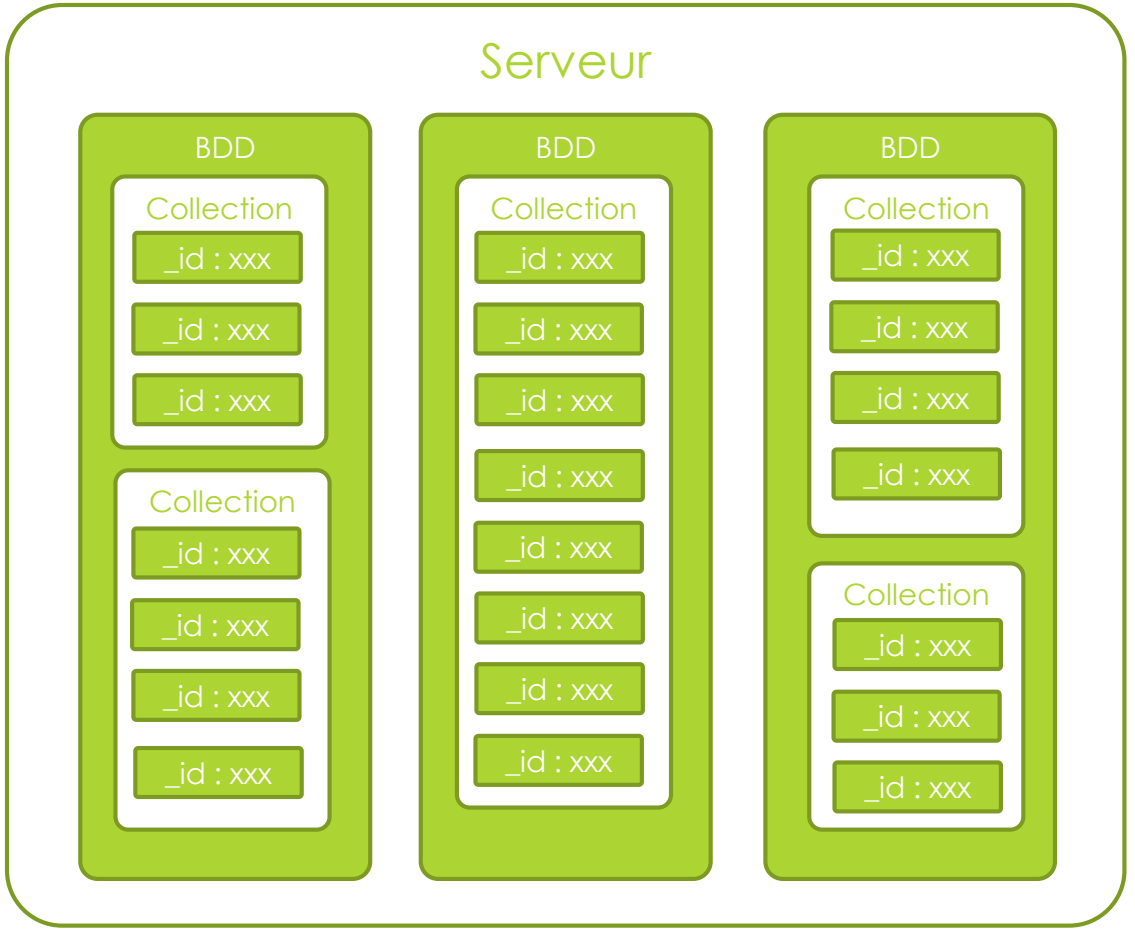
id	nom	prénom	age
1	SALAHI	Mohammed	59
2	YOUSRI	Jamal	32

id_prop	id_car
1	3
2	1
2	3

```
[
   {
      "_id":1,
      "nom":"SALAHI",
      "prenom":"'Mohammed'",
      "age":59,
      "voitures":[
         {
            "marque":"'RENAULT'",
            "modele":"'TWINGO'",
            "annee":1998
         }
      ]
   },
   {
      "_id":2,
      "nom":"YOUSRI",
      "prenom":"'Jamal'",
      "age":32,
      "voitures":[
         {
            "marque":"'FORD'",
            "modele":"'Mustang'",
            "annee":1967
         },
         {
            "marque":"'CHEVROLET'",
            "modele":"'Camaro'",
            "annee":2015
         }
      ]
   }
]
```

Organisation de MongoDB

- **Serveur** ➔ permet d'héberger plusieurs bases de données
- **Base de données** ➔ contient un ensemble de collections. Chaque base de données est sécurisée et étanche par rapport aux autres
- **Collections** ➔ contient un ensemble de documents de même nature
- **Documents** ➔ représente une ensemble de données structurées
- **ObjectId** ➔identifiant unique généré automatiquement pour identifier chaque document



Importation et exportation de documents

- **Importation de documents au format JSON**

On utilise l'outil mongoimport : un exécutable à part entière, pas une commande du mongo shell

```
mongoimport --db dbName --collection collectionName
    --mode importMode --file fileName.json --jsonArray
```

- **Exportation de documents au format JSON**

On utilise l'outil mongoexport : un exécutable à part entière, pas une commande du mongo shell

```
mongoexport --db dbName --collection collectionName
    --out fileName.json --query theQuery
```

Les commandes de base

- **db** : affiche les informations de la base courante
- use <db> : utiliser la base db
- **db**.<collection>.**find**(<critère>,<projection>) : retourne les 10 premiers enregistrements correspondant aux critères de recherche de la collection
  - **Exemple** : db.personne.find()
- **db**.<collection>.**findOne**(<critère>,<projection>) : retourne le premier enregistrement correspondant aux critères de recherche de la collection
- **db**.<collection>.**insertOne/insertMany**(<documents>) : insert le ou les documents dans la collection
- **db**.<collection>.**updateOne/updateMany**(<filter>,<action>) : met à jour le ou les documents dans la collection
- **db**.<collection>.**replaceOne**(<filter>,<replacement>) : remplace le document matchant les filtre dans la collection
- **db**.<collection>.**removeOne/removeMany**(<filter>) : supprime le ou les documents correspondant à la recherche dans la collection

Requêtes: Opérateurs

Permet de filtrer les enregistrements retournés lors de l'exécution de la requête

- Liste des divers types d'opérateurs

Comparaisons :

$eq, $ne	==, !=
$gt, $gte, $lt,$lte	>, >=, <, <=
$in, $nin	∈, ∉

Logiques :

$and, $or	AND, OR
$not, $nor	NOT, NOR

Op de test sur tableau :

$all	Test sur le contenu d'un tableau
$elemMatch	
$size	Taille du tableau

Op de test sur élément :

$exists	Existence d'un champ ?
$type	Teste le type d'un champ

Op d'évaluation :

$mod	Calcule et teste le résultat d'un modulo
$regex	Recherche d'expression régulière
$text	Analyse d'un texte
$where	Test de sélectionne des enregistrements

- + d'autres types d'opérateurs …

Requêtes : les projections

- Permet de sélectionner les champs qui seront retournées après l'exécution de la requête

- Exemple :
  - `db.personne.find({},{})`
    - `{ _id:XXX, nom:"BRAMI", prenom: "JAMAL"}`
  - `db.personne.find({},{nom:1})`
    - `{ _id:XXX, nom:"BRAMI"}`
  - `db.personne.find({},{_id:-1,nom:1, prenom:1})`
    - `{ nom:"BRAMI", prenom: "JAMAL"}`

Application de méthodes sur une collection

- **Application de méthodes aux collections**

db.*laCollection*.find(...).methode(...)

Parfois aussi :

db.*laCollection*.methode(...)

- **sort(...)**
- **forEach(...)**
- **...**

- **count(...)**
- **...**

Framework d'agrégation

- **Principe :**

  MongoDB propose une autre façon de coder des traitements : en formant un pipeline d'opérations

  db.collectionName.aggregate({op1}, {op2}, {op3}…)

  - La sortie d'une opération est l'entrée de la suivante, ou bien la sortie finale de l'agrégation
  - On peut pipeliner successivement autant d'opérations que l'on veut

Framework d'agrégation

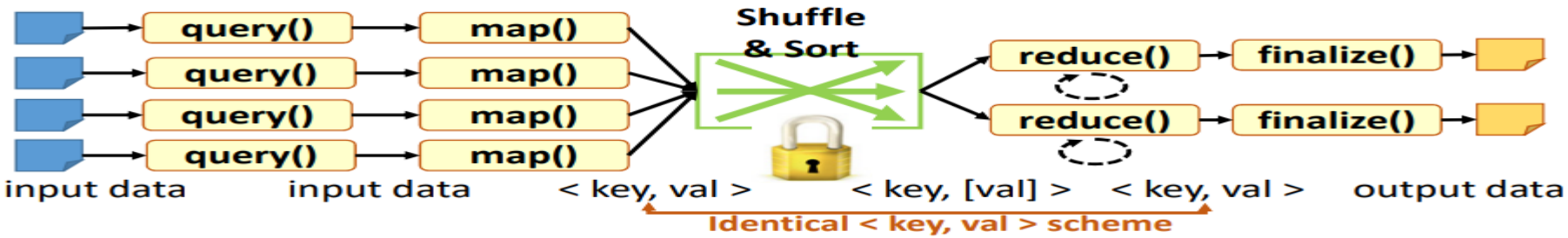
- **Opération pour l'agrégation**
  - **$match :** Permet de sélectionner/filtrer les enregistrements d'entrée
  - **$project :** Permet de projeter des attributs : seulement ceux voulus, et au besoin en les renommant ou en créant de nouveaux attributs
  - **$group:**Permet de regrouper les enregistrements retenus selon l'_id (que l'on peut redéfinir au passage), et d'appliquer des fonctions de groupe au autres attributs projetés.
  - **$unwind:** Permet de remplacer un enregistrement contenant un tableau par une suite d'enregistrements contenant chacun un seul élément du tableau
  - **$sort:** Permet d'ordonner les enregistrements selon un des attributs, par ordre croissant (+1) ou décroissant (-1).
  - **$limit:** Permet de sélectionner/filtrer les enregistrements d'entrée
  - **$lookup :** La solution pour réaliser des « JOIN » sur plusieurs collections…

mapReduce de MongoDB

- **MongoDB possède son propre « Map-Reduce » :**
  - Ses propres règles de fonctionnement (un peu différentes d'Hadoop)
  - Ses propres règles de déploiement des tâches
  - Et son propre middleware sous-jacent (il n'est pas bâti au dessus d'Hadoop)
  - N'exploite pas les index
  - Fonctionne sur des bases distribuées (sharded)
- **Principes du mapReduce de MongoDB :**
  - Une query pour pré-filtrer la collection traitée
  - Une fonction **map()**, en Java Script et qui accède à la base
  - Une fonction **reduce()**, en Java Script et qui ne doit PAS accéder à la base qui doit être commutative, associative et idempotente (!!)
  - Une fonction **finalize()**, en Java Script et qui ne doit pas accéder à la base
  - La possibilité de définir un ensemble de variables globales aux 3 fonctions map(), reduce() et finalize()

mapReduce de MongoDB

- **Principes du mapReduce de MongoDB :**



- **map()** fonctionne comme en Hadoop, mais sur une seule collection
- pour éviter d'implanter un filtrage en JS, une query exprimée en Mongo shell est applicable en amont (plus pratique et plus rapide)
- **reduce()** est plus contraint qu'en Hadoop car MongoDB applique reduce() sur des de sortie de map(), et sur des sorties précédentes de reduce() :
  - le format de sortie de reduce() doit être celui de sortie de map().
  - La fonction reduce() doit être commutative, associative et indempotente.
- **finalize()** permet de modifier la sortie finale de reduce() sans contrainte

Utilisation de MongoDB