

# Big Data - TP

MongoDB

**Mohamed Yassine Landolsi**

 [yassine-net@hotmail.fr](mailto:yassine-net@hotmail.fr)

Faculté des Sciences de Monastir



2021-2022

# Sommaire



**1 : « Introduction »**

---

**2 : « Préparation »**

---

**3 : « Architecture »**

---

**4 : « Requêtes »**

---

**5 : « Map & Reduce »**




---

**6 : « Indexation »**



# Guide

On a **3** styles de diapos :

- **Découvrir :** 
  - Introduction et définition des concepts.
- **Préparer :** 
  - Les commandes, outils et astuces à utiliser.
  - Il faut changer les mots qui sont **en bleu et en gras**.
  - Le texte en **vert** est un commentaire pour plus d'explication.
- **Tirer :** 
  - Les exercices.
  - On peut utiliser une commande ou un outil précédemment apparu.

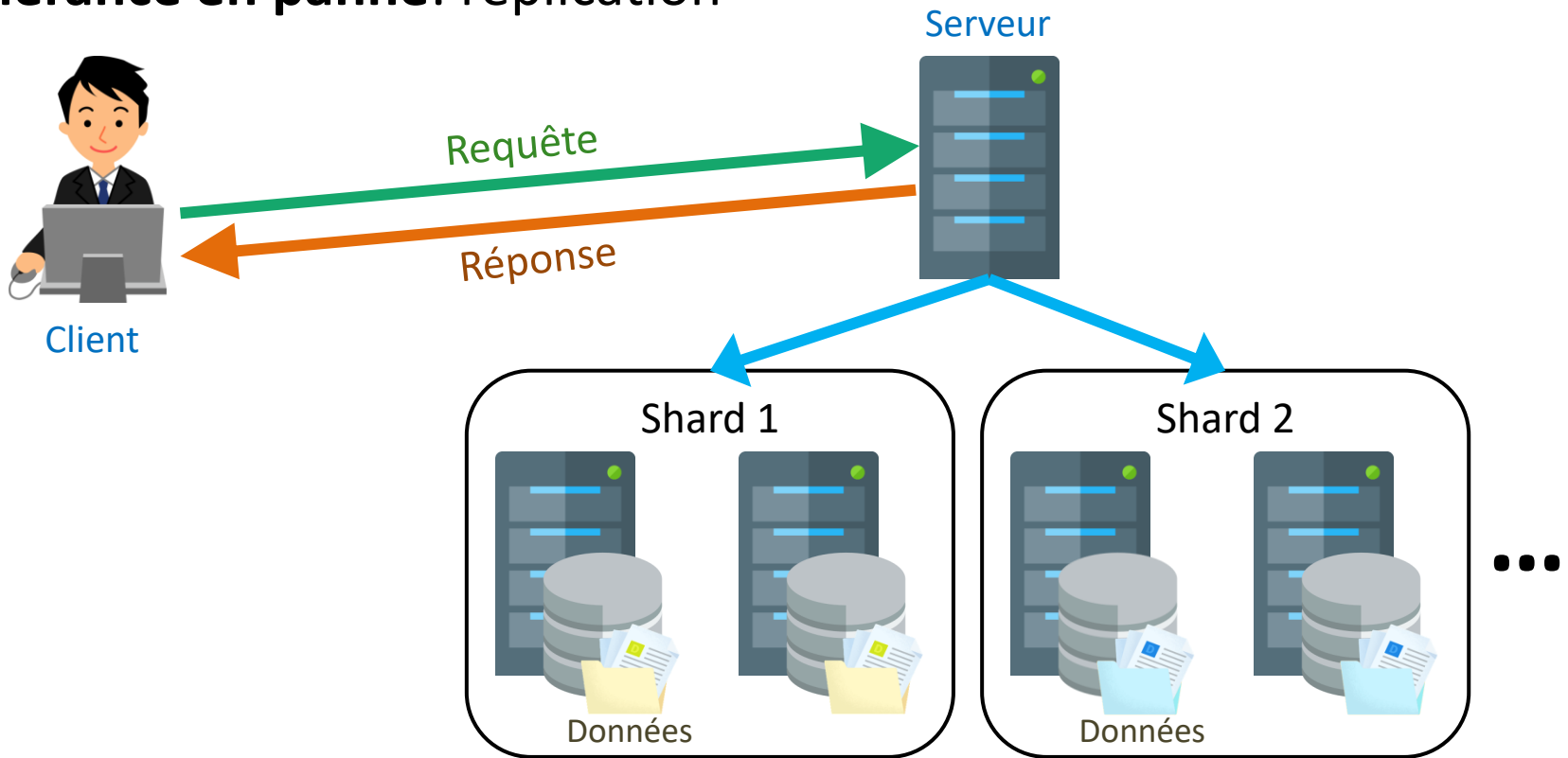
# Séance 1 >



# Introduction

## MongoDB?

- Système **NoSQL**
- **Scalable** et **distribuée**: partitionnement (Sharding)
- **Tolérance en panne**: réplication





# Introduction

## MongoDB?

- Données **semi-structurées** (JSON)
- **Flexibilité** (pas de schéma)
- **Langage d'interrogation**: originale et spécifique.

### Exemple :

#### Documents :

Collection  
«movies»

```
{  
  "_id": <ObjectId1>,  
  "title": "Pulp fiction",  
  "year": "1994",  
  "genre": "Action",  
  "director": {  
    "last_name": "Tarantino",  
    "first_name": "Quentin",  
    "birth_date": "1963"  
  }  
}
```

Document 1

```
{  
  "_id": <ObjectId2>,  
  "title": "Interstellar",  
  "year": "2014",  
  "genre": ["Adventure", "Drama", "Fiction"],  
  "director": {  
    "last_name": "Christopher",  
    "first_name": "Nolan"  
  }  
}
```

Document 2

Requête :      `db.movies.find( { "title": "Interstellar" } )`



# Introduction

## NoSQL vs SQL

NoSQL	SQL
Base	Base
Collection	Table
Champ (pas de schéma)	Colonne (schéma défini)
Document JSON (non structuré / structuré)	Ligne (structuré)
Autonomie	Jointure

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "cin": 12442154,
  "nom": "Ali",
  "date_nais": "15/02/1997",
  "groupes": ["groupeA", "groupeB"]
},
{
  "_id": ObjectId("507f191e810c19729de860ea"),
  "cin": 12354688,
  "nom": "Ahmed",
  "date_nais": "13/12/1998"
}
```

Structure complexe

Pas de schéma défini

id	cin	nom	date_nais
1	12442154	"Ali"	"15/02/1997"
2	12354688	"Ahmed"	"13/12/1998"



# Introduction

## Quand NoSQL?

- **Stocker et retrouver** de **larges** volumes de données.

## Données :

- Relations pas importantes.
- Changent au fil de temps.
- Non structurées.
- Augmentent en continu.



# BIG DATA





# Préparation

## Installation (mongod, mongo, mongos, mongoimport...)

1. Téléchargez : <https://www.mongodb.com/try/download/community>  
Choisissez l'onglet "**Community server**", package "**msi**", plateforme "**Windows**" et version "**3.2.22**".
2. Installez le fichier téléchargé.
3. Trouvez le dossier "**bin**" à partir du dossier "**MongoDB/Server**" dans "**Programmes files**".
4. Copiez le chemin du dossier "**bin**".  
Exemple : "**C:\Program Files\MongoDB\Server\3.2\bin**".
5. Ajoutez ce chemin au variable d'environnement "PATH".  
**Poste de travail → Propriétés → Paramètres système avancés → Variables d'environnement → Variables utilisateur.**



# Préparation

## Connexion au serveur

1. Ouvrez l'**invite de commande** du serveur.

2. Démarrage du serveur :

```
mongod --port numPorte --dbpath cheminData
```

Changez "**numPorte**" (exemple: "**27017**").

Changez "**cheminData**" (exemple: "**C:/data/db/n1**").

(assurez-vous que ce dossier existe & vide)

3. Ouvrez l'**invite de commande** du client.

4. Connexion au serveur (par client) :

```
mongo --port numPorte
```

Changez "**numPorte**" par la porte du serveur.

■ Pour connaître le nom de la machine par défaut :

```
getHostName()
```

Exemple de résultat: "**NomPC**".



**NomPC:27017**  
C:/data/db/n1



# Préparation

## Gestion des bases

- Placer dans la base voulue :

**use nomBD**

Changez "**nomBD**" par le nom de la base.

Sera créée après l'ajout de la 1<sup>ère</sup> collection.

- Liste des bases créées :

**show dbs**

Bases réservées au système : **admin, config, local**

- Suppression de la base courante :

**db.dropDatabase()**

Il faut utiliser la base à supprimer.





# Préparation

## Gestion des collections

- Création d'une collection :

**db.createCollection("nomCollection")**

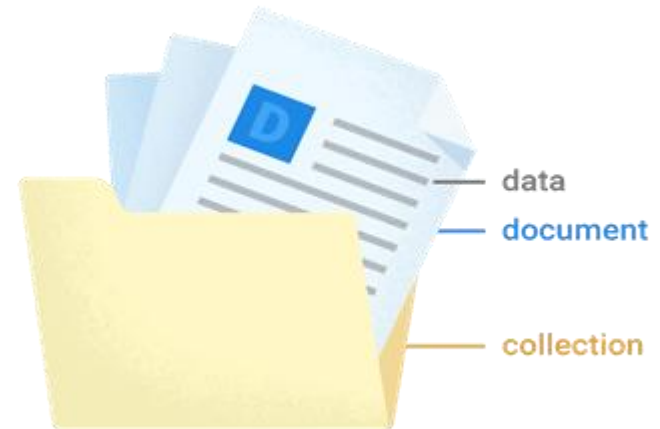
Changez "**nomCollection**" par le nom de la collection.

- Liste des collections :

**show collections**

- Suppression d'une collection :

**db.nomCollection.drop()**



- Importation d'une collection (JSON) :

**mongoimport -d nomBD -c nomCollection --port numPorte --file cheminFichier.json --jsonArray**



# Préparation

## Gestion des documents (quelques requêtes)

- Ajout d'un document:

**db.nomCollection.insert(doc)**

Changez "**doc**" par le document à insérer (exemple: {"**champ1**": val1, ...} ).

Changez "**nomCollection**" par le nom de la collection.

Si la collection n'existe pas, elle sera créée automatiquement.

- Modification d'un seul document :

**db.nomCollection.update(docReq, {\$set: docModif})**

Changez "**docReq**" par le filtre de sélection (format document).

Changez "**docModif**" par les modifications à appliquer (format document).

Changez "**docReq**" par {} pour sélectionner tous les documents.

- Récupération des documents :

**db.nomCollection.find(docReq)**

- Suppression des documents :

**db.nomCollection.remove(docReq)**





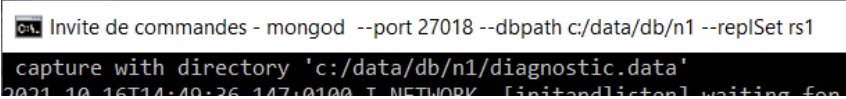
# Préparation

## Astuces

- Les commandes **mongo**, **mongod**, **mongos**, **mongoimport**... chacune doit être exécutée en dehors de l'autre.

**Exemple:** `C:\Users>mongo --port 27018`

- Le titre de l'**invite de commande** indique la commande en cours d'exécution.

**Exemple:** 

- Un chemin qui contient un espace doit être entre guillemets "...".

**Exemple:** `mongo --file "c:/mon doss/fich.json"`

- Le bouton « ↑ » récupère une copie de la dernière commande exécutée.
- Un éditeur de texte (comme Notepad++) peut être plus pratique pour préparer les commandes avant de les exécuter.



# Préparation

## Exercices TP1

1. Démarrez un serveur sur la porte **27017** en utilisant le chemin de données « **C:/data/db/n1** ».
2. Connectez-vous au serveur et utilisez la base « **TestDB** ».
3. Utilisez la base « **UserDB** ».
4. Créez la collection « **users** » dans la base « **UserDB** ».
5. Affichez les bases créées.
6. Créez la collection « **produits** » dans la même base.
7. Affichez les collections.
8. Supprimez la collection « **produits** ».
9. Supprimez la base « **TestDB** ».
10. Ajoutez **3** utilisateurs dans la collection « **users** », leur champ **nom** a "**Ahmed**", "**Amine**" et "**Youssef**" comme valeurs, avec **age 15, 20 et 15** respectivement.
11. Recherchez les utilisateurs avec l'âge **15**.
12. Modifiez l'âge de **Youssef** par **20**.
13. Supprimez les utilisateurs qui ont l'âge **20**.
14. Affichez tous les utilisateurs.

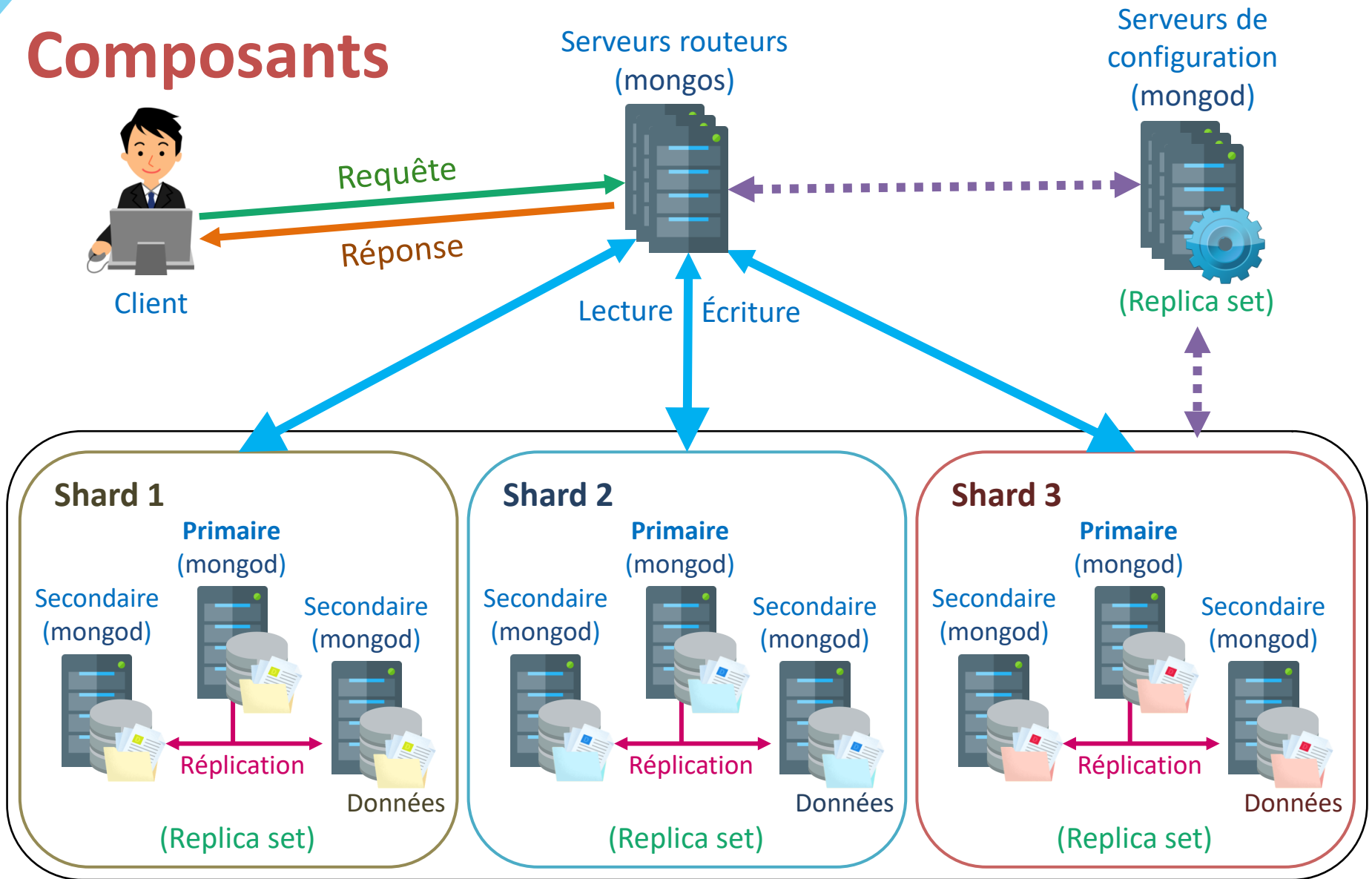
# Séance 2 >





# Architecture

## Composants





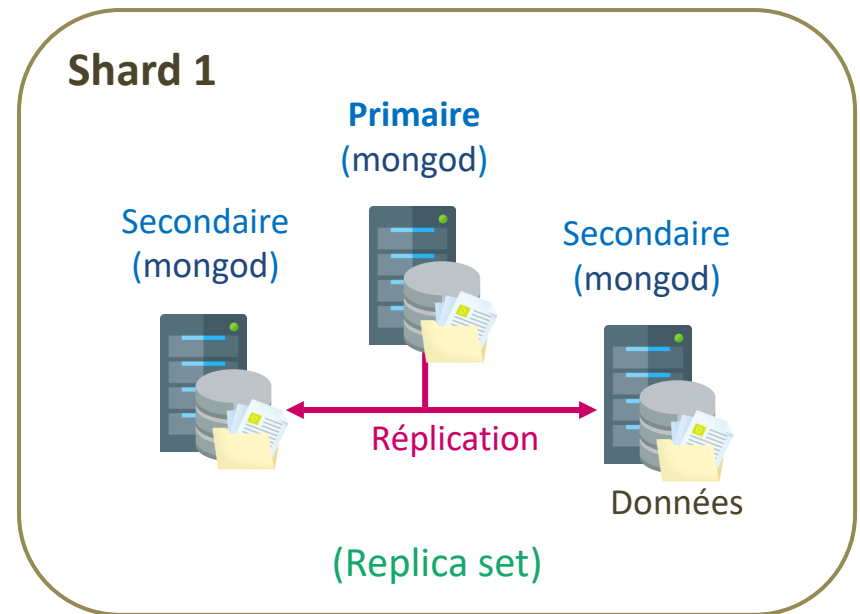
# Architecture

## Composants (Shard)

- Contient un **fragment de données**.
- Espace **insuffisant**? ajouter un **autre Shard**.

### Replica set :

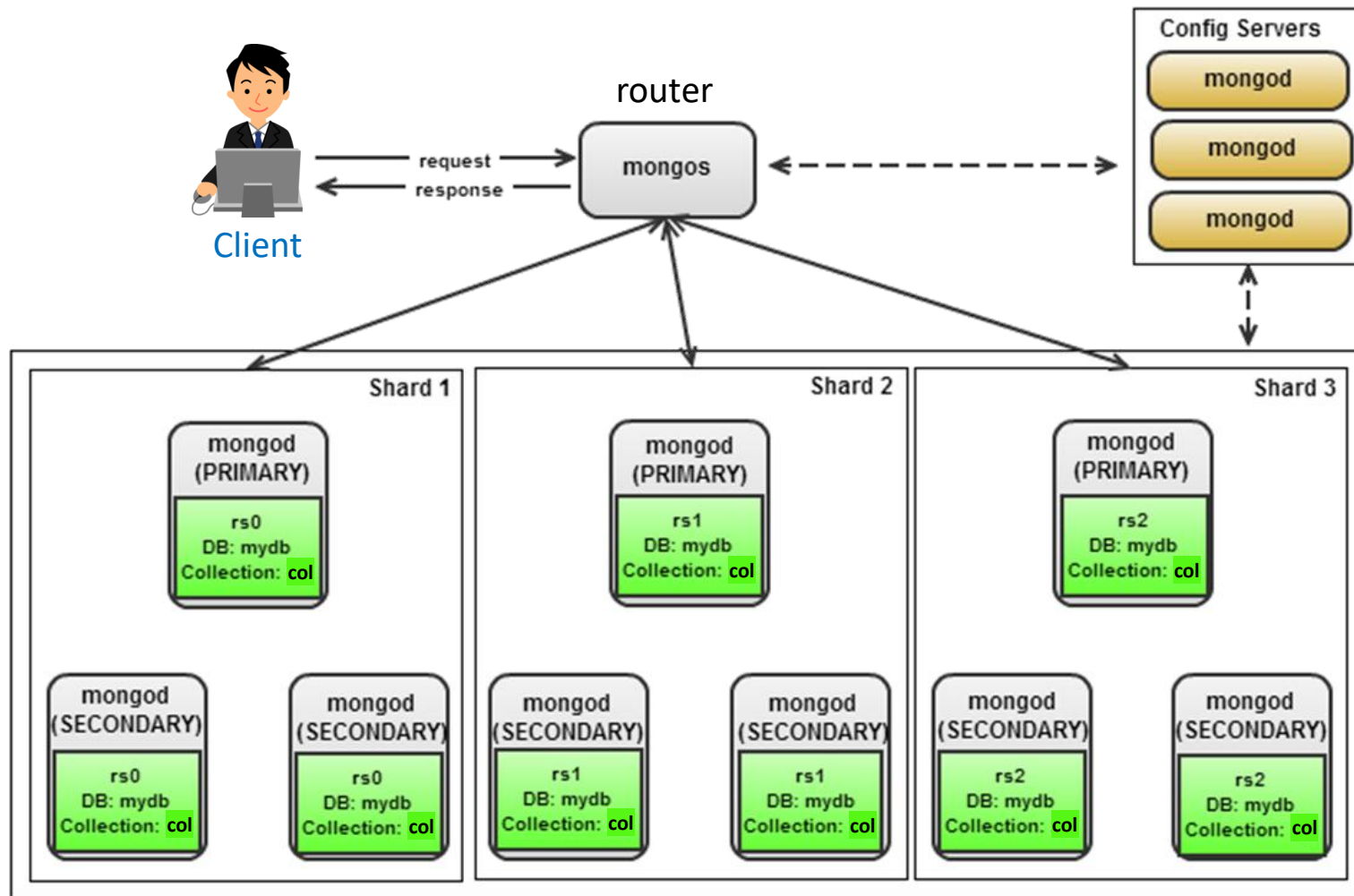
- **Chaque serveur** :
  - a une **copie** du fragment.
  - applique les **mise à jours**.
- Le **serveur primaire** :
  - reçoit **toutes les requêtes**.
- Les **serveurs secondaires** :
  - sont des **remplaçants**.
  - peuvent être **interrogeables**.





# Architecture

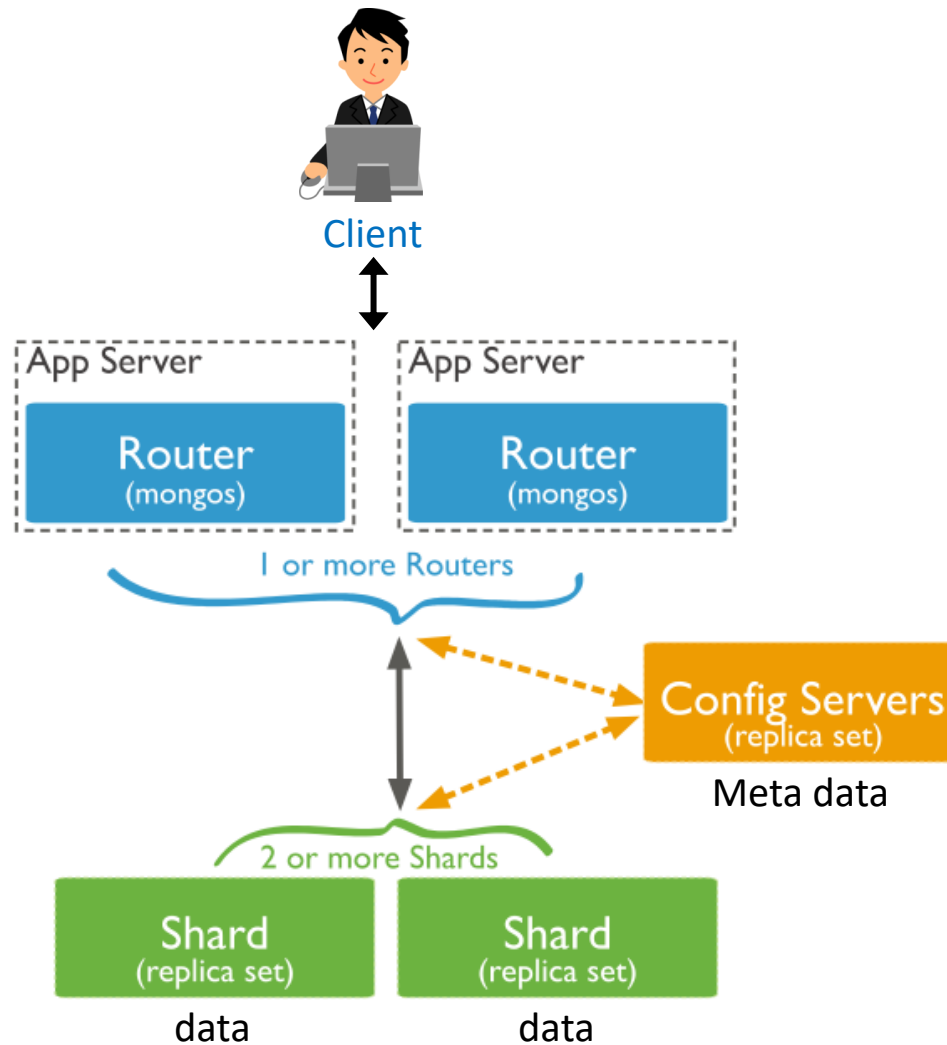
## Composants (autre style)





# Architecture

## Composants (abstrait)





# Architecture

## Configuration (Réplica set) Création

1. Démarrage d'un serveur dans un Replica set :

**mongod** --port **numPorte** --dbpath **cheminData** --replSet **nomRS**

Changez "**nomRS**" par le nom du Replica set.

Vous pouvez démarrer plusieurs serveurs.

2. Connexion au serveur qui doit être primaire.

**mongo** --port **numPorte**

3. Définir ce serveur en tant que primaire :

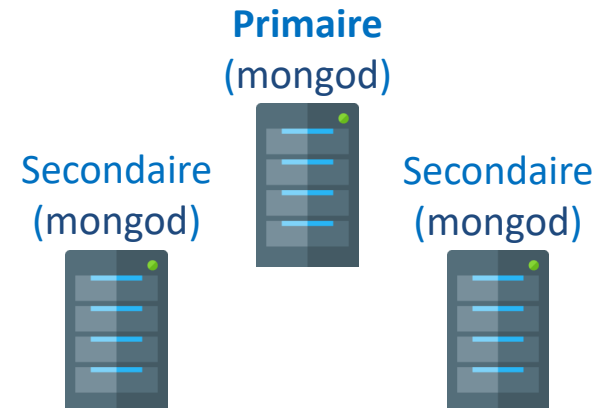
**rs.initiate()**

4. Définir un serveur secondaire :

**rs.add("nomMachine:numPorte")**

Changez "**nomMachine**" (exemple: "**NomPC**").

Vous pouvez ajouter plusieurs serveurs.





# Architecture

## Configuration (Réplica set)

- Autoriser l'interrogation d'un serveur secondaire :  
**rs.slaveOk()**
- Voir si le serveur est primaire :  
**db.isMaster()**
- Arrêt du serveur :  
**use admin**  
**db.shutdownServer()**
- Informations sur le Replica set :  
**rs.status()**



# Préparation

## Astuces

- La raccourcis « **ctrl + c** » pour reconnecter à un autre serveur en utilisant la même fenêtre d'invite de commande (du client).

Exemple:

```
Invite de commandes - mongo --port 27018
test:PRIMARY> ^C
bye

C:\Users\YassineLand>mongo --port 27018
MongoDB shell version: 3.2.22
connecting to: 127.0.0.1:27018/test
test:SECONDARY> _
```

- La sélection dans la fenêtre d'un serveur démarré peut causer un blocage.

Exemple:

```
Sélection invite de commandes - mongod --port 27018 --dbpath c:/data/db/hoed2 --replSet test
).
2021-10-23T12:25:20.875+0100 I REPL [Replicat
2021-10-23T12:25:20.882+0100 I REPL [Replicat
2021-10-23T12:25:21.046+0100 I REPL [Replicat
nc from
2021-10-23T12:25:36.174+0100 I NETWORK [i v nd]
1:49959 #3 (2 connections now open)
```

- La commande **getHostName()** retourne le nom de la machine après la connexion à un serveur.

Exemple:

```
test:PRIMARY> getHostName()
DESKTOP-249M265
```



# Architecture

## Exercices TP2 (Réplica set)

1. Créez le Replica set « **test** », en utilisant le nom de votre machine, avec les composants suivants :

Primaire  
(mongod)



**NomPC:27017**  
C:/data/db/noeud1

Secondaire  
(mongod)



**NomPC:27018**  
C:/data/db/noeud2





# Architecture

## Exercices TP2 (Réplica set)

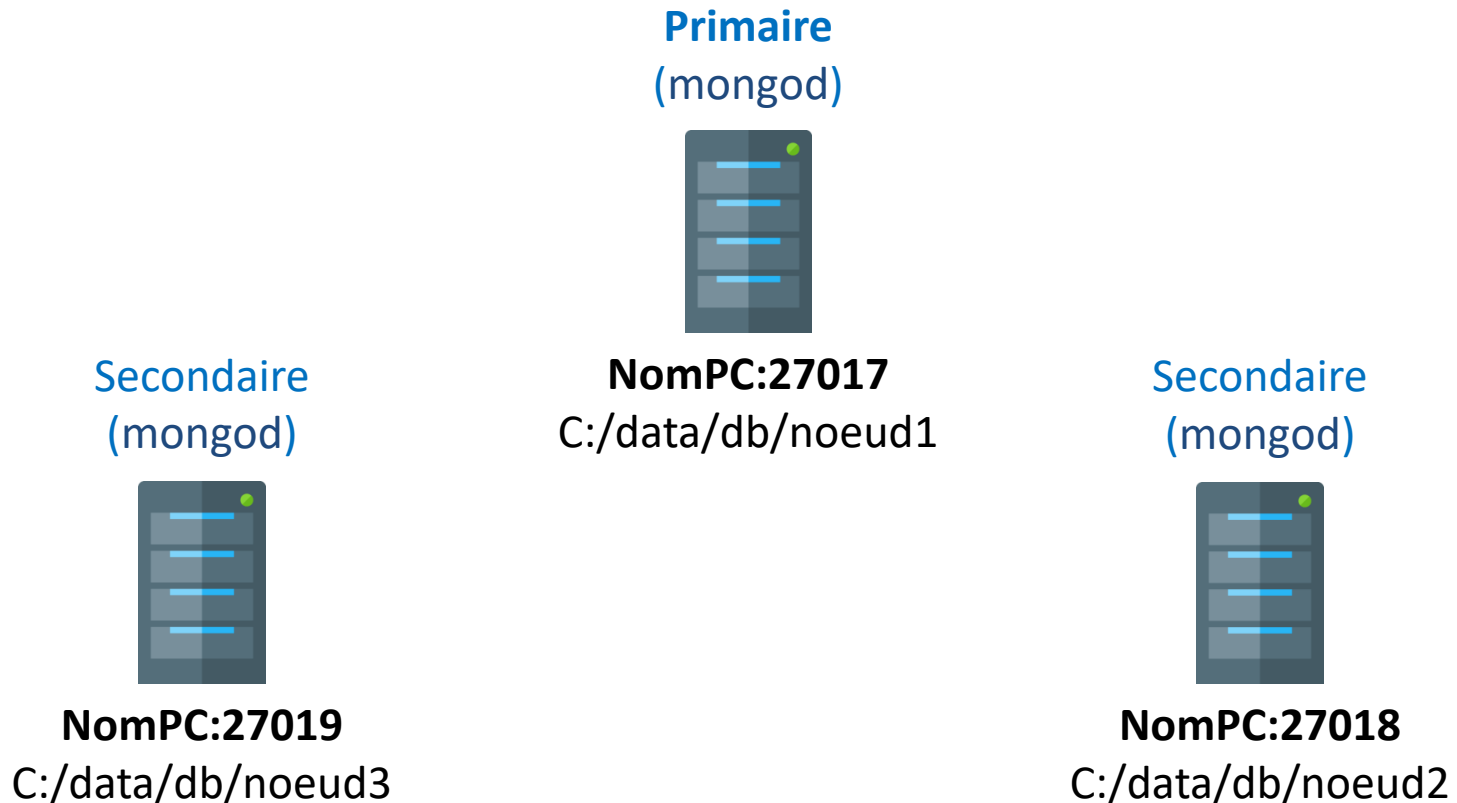
2. Voir si le serveur « **NomPC:27017** » est primaire.
3. Voir des informations complètes sur le Replica set.
4. Ajoutez un document à la collection « **test** » de la base « **dbtp** » sous le **serveur primaire**.
5. Essayez de retourner tous les documents de la base « **dbtp** » sur le **serveur secondaire**. Que se passe-t-il?
6. Refaire ça après avoir rendu ce serveur interrogeable.
7. Tuez le **serveur primaire**. Le second va-t-il s'élire un primaire?
8. Redémarrez le serveur « **NomPC:27017** ». Que se passe-t-il?



# Architecture

## Exercices TP2 (Réplica set)

9. Répéter l'expérience en ajoutant un serveur secondaire sur la porte **27019** :



**Séance ? >**



# Architecture

## Configuration (Sharding) Création

1. Création d'un serveur de configuration :

**mongod** --port **numPorte** --dbpath **cheminData** --configsrv

Changez **numPorte** par la porte de la configuration.

2. Lancement du routeur avec ce serveur.

**mongos** --port **numPorte** --configdb **ServeurDB** --chunkSize **CS**

Changez **CS** par la taille du fragment en Mo (exemple: **1**).

Changez **ServeurDB** par **nomMachine:numPorte** de la configuration.

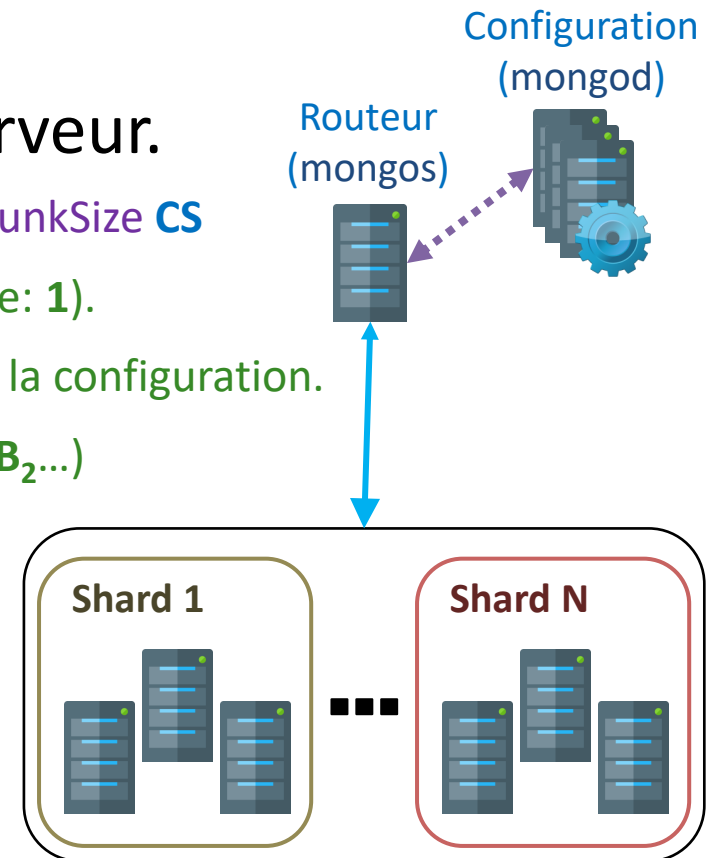
(Pour un Replica set : **nomRS/ServeurDB<sub>1</sub>,ServeurDB<sub>2</sub>...**)

Changez **numPorte** par la porte du routeur.

3. Connexion au routeur.

**mongo** --port **numPorte**

Changez **numPorte** par la porte du routeur.





# Architecture

## Configuration (Sharding) Création

4. Ajout d'un Shard :

```
sh.addShard("ServeurDB")
```

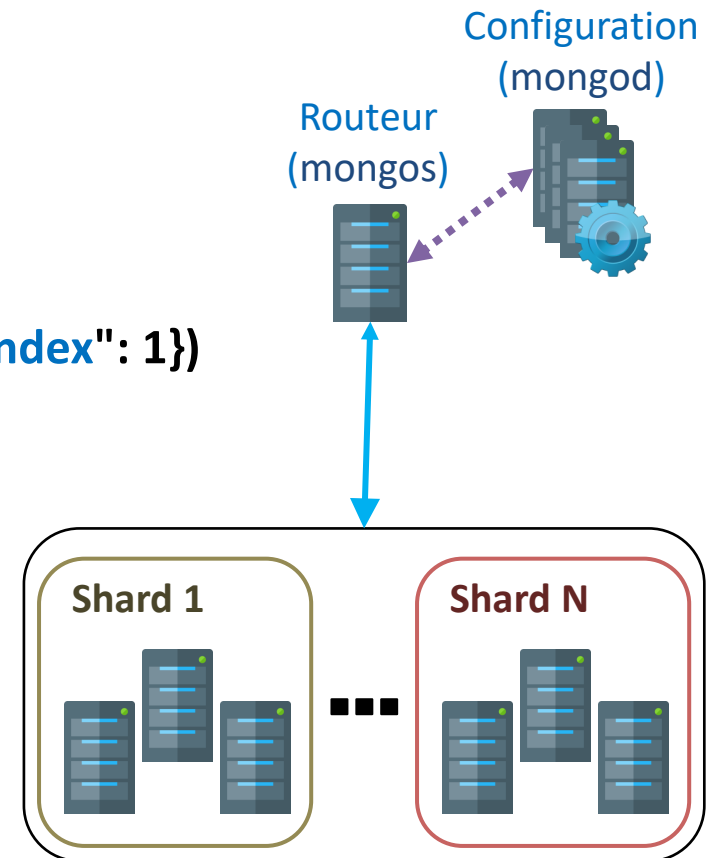
5. Choisir la base :

```
sh.enableSharding("nomBD")
```

6. Choisir la collection et son index :

```
sh.shardCollection("nomBD.nomCollection", {"index": 1})
```

Changez **index** par un champ (exemple: **\_id**).





# Architecture

## Configuration (Sharding)

- Informations sur le Sharding :  
`sh.status()`



# Architecture

## Exercices TP? (Sharding)

1. Démarrez le serveur simple suivant (pas dans un Replica set) :

**Serveur**  
(mongod)



**NomPC:27019**  
C:/data/db/n1



# Architecture

## Exercices TP? (Sharding)

2. Importez la collection « **employees.json** » sous le nom « **employees** » dans la base « **dbtp** » dans le serveur de la porte **27019**.  
→ `mongoimport -d dbtp -c employees --port 27019 --file c:/employees.json --jsonArray`
3. Démarrez le serveur simple suivant (pas dans un Replica set) :

Serveur  
(mongod)



NomPC:27020  
C:/data/db/n2

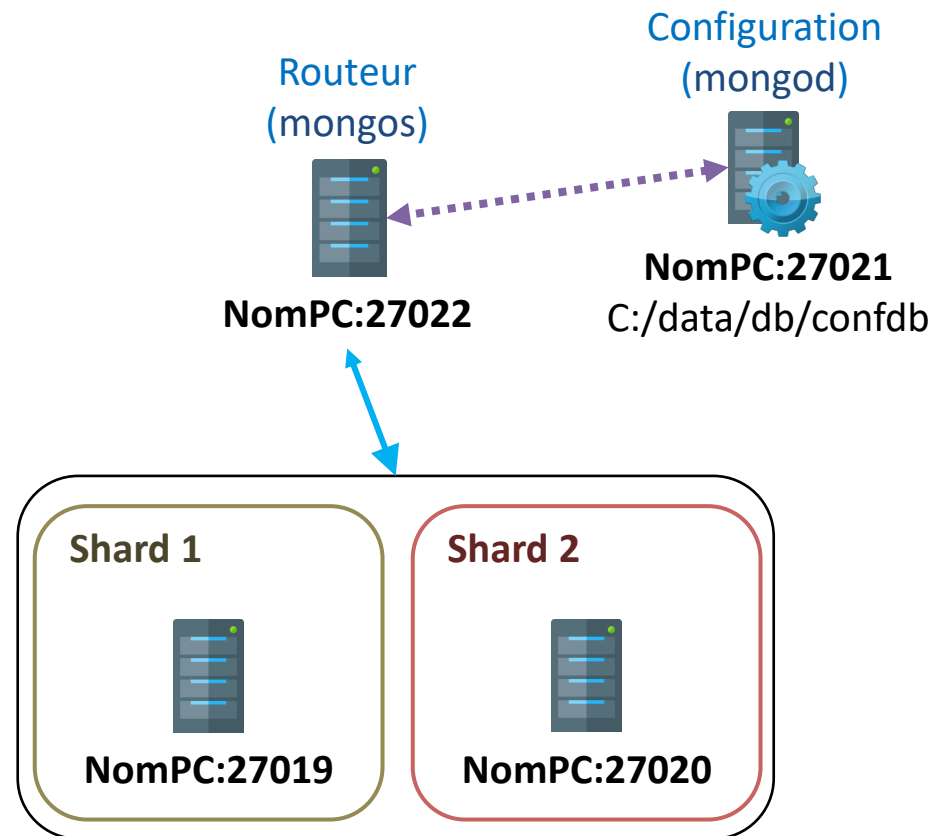




# Architecture

## Exercices TP? (Sharding)

5. Créez ce Sharding sur la collection « **employees** » de la base « **dbtp** » en utilisant « **\_id** » comme index et avec taille de fragment **1 Mo** :





# Architecture

## Exercices TP? (Sharding)

6. Connectez-vous au routeur et affichez des informations sur le Sharding.
7. Retrouvez les employés qui ont le **nom "Pasqua"**.

# Séance 3 >



# Requêtes

## Définition





# Requêtes

## Installation

1. Téléchargez et installez le logiciel à travers le lien suivant :  
<https://robomongo.org>  
Cliquez sur « **Download Robo 3T** ».
  2. Ouvrez le logiciel.
  3. Cliquez sur **Create / Edit** pour créer ou modifier une connexion.  
Vous pouvez changez la porte selon le serveur démarré.
  4. Cliquez sur « **Save** ».
  5. Cliquez sur « **Connecter** ».
  6. Cliquez droit sur le nom de la base (à gauche).
  7. Choisissez « **Open Shell** ».
- Téléchargement des collections en format JSON :  
<https://drive.google.com/drive/folders/10cWgnL7sZqilsrOW1ne0JKpF3N95XkV7>



# Requêtes

## Astuces

- On peut commenter les requêtes déjà exécutées pour les conserver et éviter la double exécution.

### Exemple :

The screenshot shows a MongoDB interface with a sidebar on the left displaying the database structure: Localhost (2) > System > mydb. The main window has a tab for the query `//db.employees.find({}) ...`. Below the tab, the query is shown in a dark editor with a red bracket on the left indicating a comment block. The query is:

```
//db.employees.find({})  
//db.employees.insert({"email": "fsm@gmail.com", "naissance":{"jour": 25, "mois": 5, "annee":2015}})  
db.employees.find({"naissance.mois": 5})
```

Below the query, the execution status is shown: `employees` 0.001 sec. The result is displayed in a table with two columns: Key and Value.

Key	Value
(1) ObjectId("617e972fac15a38bbba0cdd")	{ 3 fields }
_id	ObjectId("617e972fac15a38bbba0cdd")
email	fsm@gmail.com
naissance	{ 3 fields }
jour	25.0
mois	5.0
annee	2015.0



# Requêtes

## Requêtes d'interrogation

- Sélectionner tous les documents qui répondent à une requête :  
`db.nomCollection.find(docReq)`
- Sélectionner un seul document :  
`db.nomCollection.findOne(docReq)`
- Sélectionner un nombre limité de documents :  
`db.nomCollection.find(docReq).limit(nbrMax)`  
Changez **nbrMax** par le nombre limite de documents à retourner.
- Sauter un nombre de documents et sélectionner le reste :  
`db.nomCollection.find(docReq).skip(nbrSauts)`  
Changez **nbrSauts** par le nombre de sauts.
- Retourner le nombre des documents sélectionnés :  
`db.nomCollection.find(docReq).count()`



# Requêtes

## Requêtes d'interrogation

- Trier les documents sélectionnés selon un champ :  
`db.nomCollection.find(docReq).sort({"champTri": OrdreTri})`  
Changez **champTri** par le champ à trier.  
Changez **OrdreTri** par **1** (croissant) ou **-1** (décroissant).
- Choisir les champs à afficher/masquer :  
`db.nomCollection.find(docReq, {"champAff": modeAff})`  
Changez **champAff** par le champ à afficher/masquer.  
Changez **modeAff** par **1** pour l'afficher ou **0** pour le masquer.  
(Les champs avec **1** vont masquer les autres sauf **\_id**)





# Requêtes

## Accès aux champs

- Accès direct à un champ :

"**champDest**": **val**

Changez **champDest** par le champ auquel on veut accéder.

champ

- Accès à un champ imbriqué :

"**champSrc.champDest**": **val**

Changez **champSrc** par le champ qui contient **champDest**.

On peut avoir un chemin source : **champSrc<sub>1</sub>.champSrc<sub>2</sub>**. ...

- Accès à une indice spécifique d'une liste :

"**champSrc.indice**": **val**

Changez **indice** par l'indice de la valeur à laquelle on veut accéder (0, 1, 2, ...).



# Requêtes

## Types des valeurs

- Type simple :

**val**

Exemple: entier (**54**), réel (**36.5**) ou chaîne ("**Ahmed**").

- Liste :

**[val<sub>1</sub>, val<sub>2</sub>, ...]**

Exemple: [**"Ahmed"**, **"Ali"**, **"Saleh"**]

- Document (ou objet) :

**{"champ<sub>1</sub>": val<sub>1</sub>, "champ<sub>2</sub>": val<sub>2</sub>, ...}**

Exemple: {"nom": **"Ahmed"**, "poids": **67.4**, "amis": [**"Ahmed"**, **"Ali"**]}

**val**



# Requêtes

## Filtre de sélection (opérateurs de comparaison)

- Comparer un champ par une valeur :

"**champComp**": {**\$opComp**: **valComp**}

Changez **champComp** par le champ à comparer.

Changez **valComp** par la valeur à comparer.

Changez **opComp** par l'opérateur de comparaison :

- **gt** : greater than / supérieur à
- **gte** : greater than or equal / supérieur ou égale
- **lt** : less than / inférieur à
- **lte** : greater than / inférieur ou égale
- **eq** : equal / égale à (par défaut)
- **ne** : not equal / différent de
- **in** : dans
- **nin** : not in / pas dans
- **exists** : existe (**valComp** doit être **true** pour existe, sinon **false**).

Pour **in** et **nin** : **valComp** doit être une liste : [**valComp**<sub>1</sub>, **valComp**<sub>2</sub>, ... ]

docReq



# Requêtes

## Filtre de sélection (opérateurs logiques)

- Utiliser des opérateurs logiques :

**{\$opLogic: docReq}**

**docReq**

Changez **opLogic** par l'opérateur logique :

- **or** : ou
- **and** : et (par défaut)
- **nor** : ou exclusif
- **not** : non

Pour **or**, **and** et **nor** : **docReq** doit être une liste : [**docReq<sub>1</sub>**, **docReq<sub>2</sub>**, ... ]



# Requêtes

## Astuces

- L'opérateur de comparaison par défaut est « égale à ».

**Exemple:** Ces deux requêtes sont identiques :

```
db.users.find({"age": 25})
```

```
db.users.find({"age": {$eq: 25}})
```

- L'opérateur logique par défaut est « et ».

**Exemple:** Ces deux requêtes sont identiques :

```
db.users.find({"nom": "Ali", "age": 25})
```

```
db.users.find({$and: [{"nom": "Ali"}, {"age": 25}]})
```

- On peut avoir multiples opérateurs dans le même filtre de sélection.

**Exemple:** Un user entre 20 et 30 ans, ou son nom est différent à Ali.

```
db.users.find({$or: [  
    {"age": {$gt: 20, $lt: 30}},  
    {$not: {"nom": "Ali"}}  
]})
```



# Requêtes

## Requêtes de modification

- Modifier un seul document :  
`db.nomCollection.update(docReq, docModif)`
- Modifier plusieurs documents :  
`db.nomCollection.updateMany(docReq, docModif)`



# Requêtes

## Modification (opérateurs de modification)

- Modifier les champs des documents :

**{ $\$$ opModif: docModif}**

docModif

Changez **opModif** par l'opération souhaitée :

- **set** : modifier les valeurs des champs (en donnant les nouvelles valeurs).  
(exemple: **{ $\$$ set: {"classe": "3<sup>ls</sup>i", "diplome": "licence"} }** )
- **unset** : supprimer des champs (en donnant la valeur **1**).  
(exemple: **{ $\$$ unset: {"email": 1, "tel": 1, "age": 1} }** )
- **inc** : incrémenter la valeur des champs (en donnant le nombre à ajouter).  
(exemple: **{ $\$$ inc: {"prix": 50, "quantite": -3} }** )
- **mul** : multiplier la valeur des champs (en donnant le multiplicateur).  
(exemple: **{ $\$$ mul: {"vitesse": 2} }** )



# Requêtes

## Modification (opérateurs de modification)

- Modifier des listes dans les documents :

**{*\$opModif*: docModif}**

**docModif**

Changez **opModif** par l'opération souhaitée :

- **push** : ajouter une valeur à la liste.  
(exemple: ajouter un ami **{*\$push*: {"amis": {"nom": "Amine", "age": 7} }}** )
- **pop** : supprimer la première (-1) ou la dernière (1) valeur d'une liste.  
(exemple: supprimer le dernier ami : **{*\$pop*: {"amis": 1} }}**)
- **pull** : supprimer un élément d'une liste selon une requête.  
(exemple: supprimer les amis de l'âge >50 : **{*\$pull*: {"amis": {"age": {*\$gt*: 50} }}** )

```
{
  "_id": <ObjectId1>,
  "nom": "Youssef",
  "amis": [ {"nom": "Taha", "age": 5}, {"nom": "Ali", "age": 57}, {"nom": "Amine", "age": 7} ]
}
```





# Requêtes

## Astuces

- Sans l'opérateur de modification, les documents sélectionnés seront remplacés par le document **docModif** (sauf **\_id**).

### Exemple :

Requête :

```
db.movies.update({"titre": "Inception"}, {"titre": "Matrix", "score": 8.7})
```

Document sélectionné :

```
{"_id": "movie:1", "titre": "Inception", "annee": 2010}
```

Résultat :

```
{"_id": "movie:1", "titre": "Matrix", "score": 8.7}
```

- Les filtres de sélection (**docReq**) sont utilisés aussi dans les requêtes de modification et de suppression.

### Exemple :

```
db.produits.update({"type": "souris", "qty": {$gt: 10}}, {$set: {"prix": 5}})
```

**docReq****docModif**



# Requêtes

## Requêtes d'insertion

- Ajouter un seul document :

**db.nomCollection.insert(doc)**

Par défaut, le champ **\_id** prend une valeur unique automatiquement.  
(On peut définir une valeur unique manuellement dans **doc**)

- Ajouter plusieurs documents :

**db.nomCollection.insert([doc<sub>1</sub>, doc<sub>2</sub>, ...])**



# Requêtes

## Requêtes de suppression

- Supprimer plusieurs documents :

**db.nomCollection.remove(docReq)**

Sélectionner les documents par le filtre **docReq** et les supprimer.



# Requêtes

## Exercices TP3 (Requêtes)

1. Créez un Replica set « **myrs** » (étapes dans diapo 21) avec les composants suivants :

Primaire  
(mongod)



**NomPC:27018**  
C:/data/db/node1

Secondaire  
(mongod)



**NomPC:27020**  
C:/data/db/node3

Secondaire  
(mongod)



**NomPC:27019**  
C:/data/db/node2



# Requêtes

## Exercices TP3 (Requêtes)

2. Dans une invite de commande, importez les documents de **employees.json** sur le **serveur primaire**, sous la base **mydb** et la collection **employees**.  
(commande d'importation dans diapo 12)
3. Connectez-vous au **serveur primaire** par l'outil **Robo 3T**.
4. Affichez **5** employés de la collection **employees**.
5. Comptez le nombre des employés qui ont un champ **prime**.
6. Récupérez les employés qui ont un **numero d'adresse** **< 10**.
7. Ajoutez un employé qui a le **nom "Mohamed"**, **prenom "Salah"** et un champ **amis** qui contient une liste vide **[]**.
8. Sélectionnez cet employé par son **nom** et **prenom** et ajoutez **3** objets à sa liste **amis**, chaque objet contient un champ **nom** et un champ **score**, où les noms sont **"Samir"**, **"Ali"** et **"Amir"** avec les scores **4**, **7** et **5** respectivement.
9. Supprimez les **amis** de cet employé qui ont un **score** **≤ 5** de cette liste.
10. Affichez cet employé en utilisant son **nom** et **prenom**.
11. Supprimez les employés qui ont un **prenom "David"** ou un **nom "David"** par une seule requête.

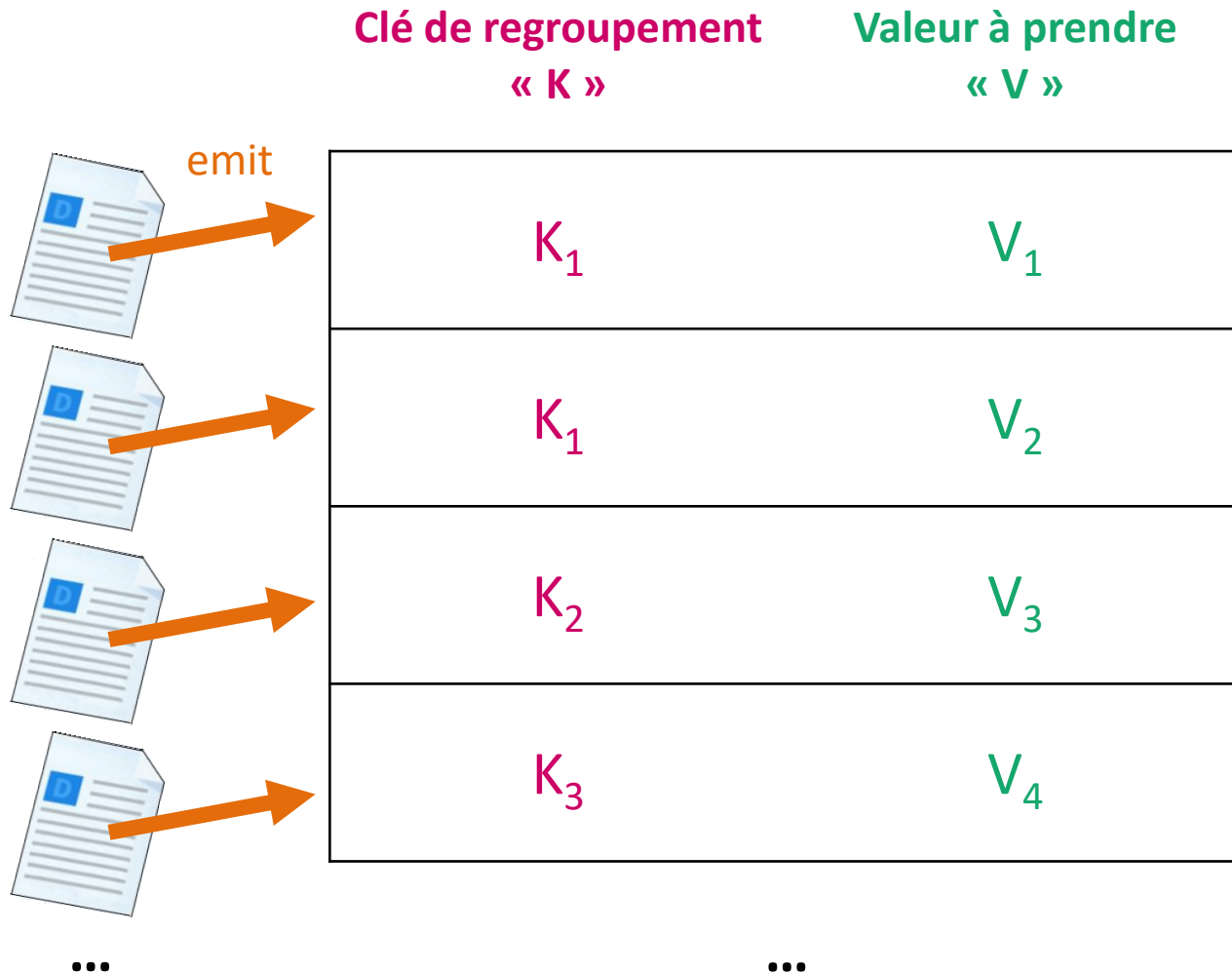
# Séance 4 >



# Map & Reduce

## Introduction

### Map

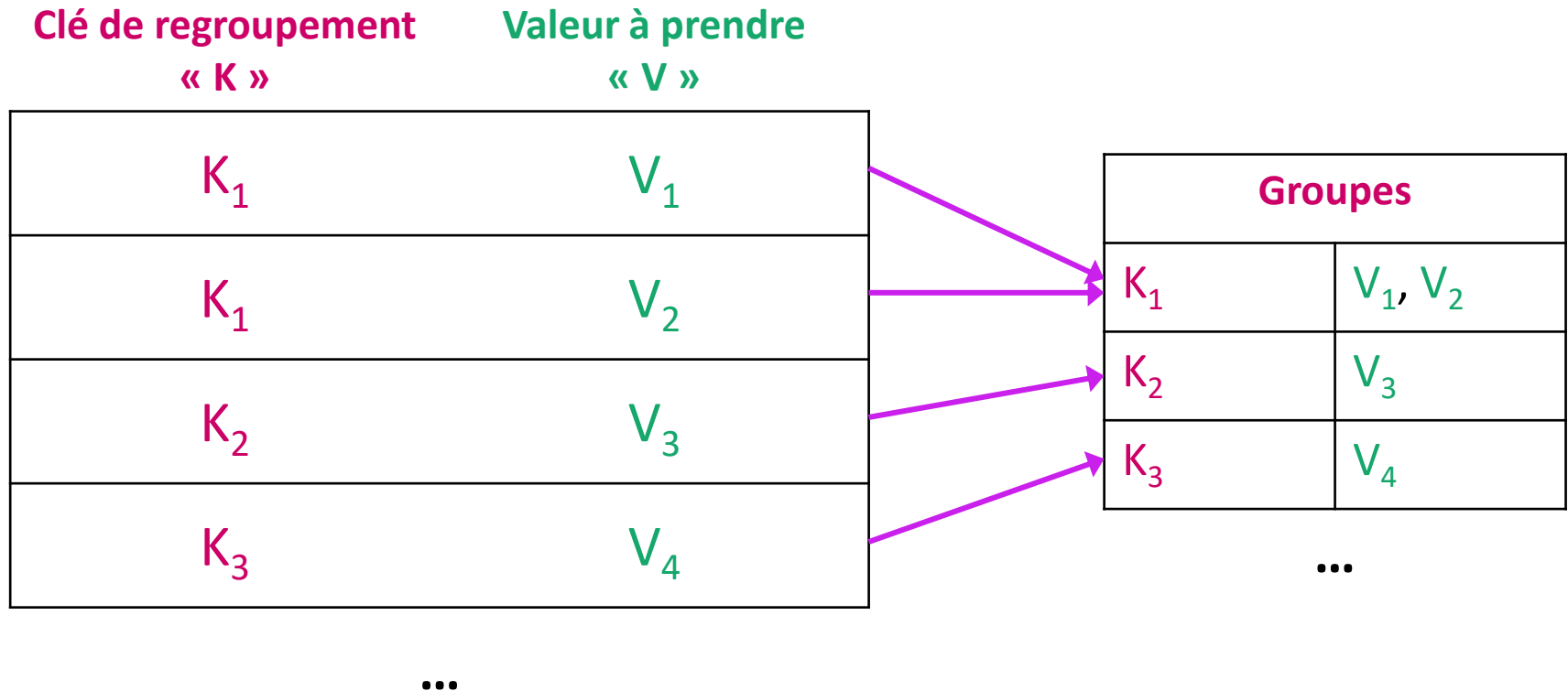




# Map & Reduce

## Introduction

### Shuffle (derrière les coulisses)







# Map & Reduce

## Introduction

### Reduce

Groupes	
$K_1$	$V_1, V_2$
$K_2$	$V_3$
$K_3$	$V_4$

...

Fonction qui peut prendre :

- la clé «  $K_i$  »
- sa liste de valeurs «  $V_1, V_2, \dots$  »



# Map & Reduce

## Exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`

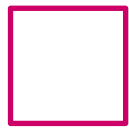
### 2) Shuffle :



2, 4



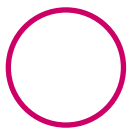
3, 7, 5



2, 1, 2

### 3) Reduce : chaque (clé, values)

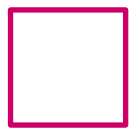
`sum(values)`



6

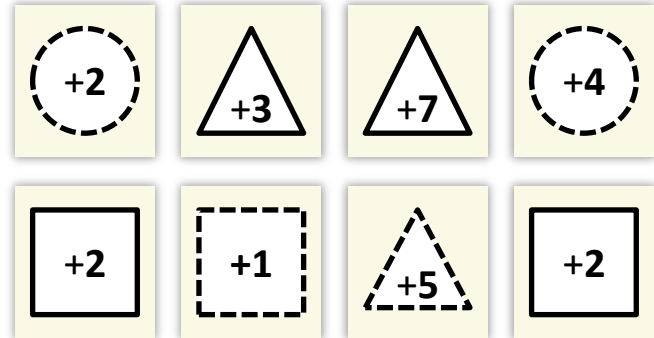


15



5

Documents :



Score total pour chaque forme.

**this** : fait référence au document  
**Map & Reduce** : sont 2 fonctions

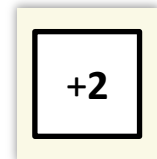
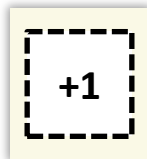
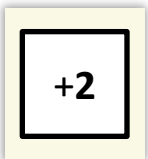
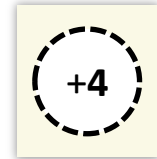
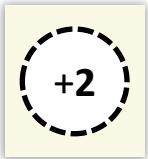


# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



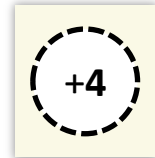
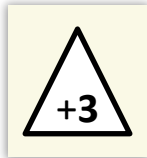
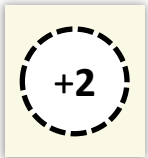


# Map & Reduce

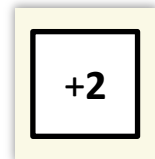
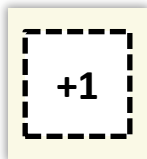
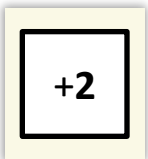
## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



`emit("cercle", 2)`



"cercle" : 2

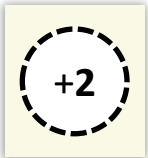


# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

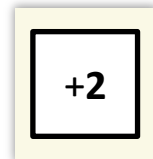
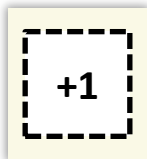
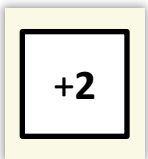
`emit(this.forme, this.score)`



`emit("cercle", 2)`



`emit("triangle", 3)`



"cercle" : 2

"triangle" : 3

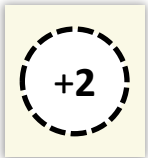


# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



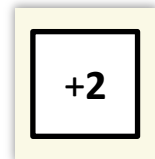
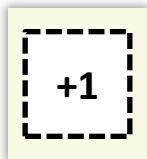
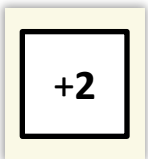
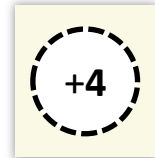
`emit("cercle", 2)`



`emit("triangle", 3)`



`emit("triangle", 7)`



"cercle" : 2

"triangle" : 3 , 7

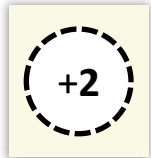


# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



`emit("cercle", 2)`



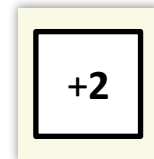
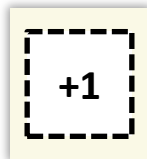
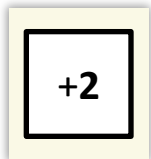
`emit("triangle", 3)`



`emit("triangle", 7)`



`emit("cercle", 4)`



"cercle" : 2 , 4

"triangle" : 3 , 7



# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



`emit("cercle", 2)`



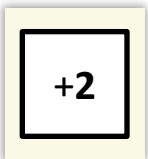
`emit("triangle", 3)`



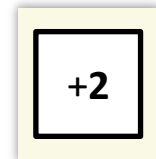
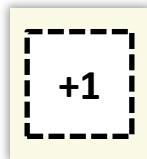
`emit("triangle", 7)`



`emit("cercle", 4)`



`emit("carre", 2)`



"cercle" : 2 , 4

"triangle" : 3 , 7

"carre" : 2



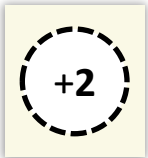


# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



`emit("cercle", 2)`



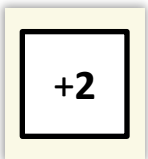
`emit("triangle", 3)`



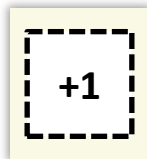
`emit("triangle", 7)`



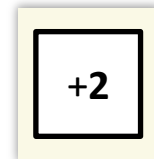
`emit("cercle", 4)`



`emit("carre", 2)`



`emit("carre", 1)`



"cercle" : 2 , 4

"triangle" : 3 , 7

"carre" : 2 , 1

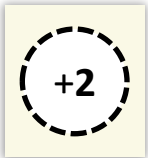


# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



`emit("cercle", 2)`



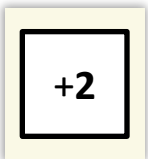
`emit("triangle", 3)`



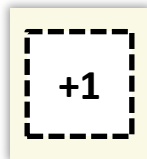
`emit("triangle", 7)`



`emit("cercle", 4)`



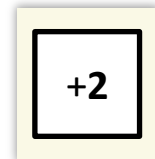
`emit("carre", 2)`



`emit("carre", 1)`



`emit("triangle", 5)`



"cercle" : 2 , 4

"triangle" : 3 , 7 , 5

"carre" : 2 , 1

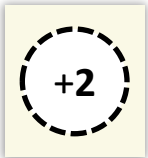


# Map & Reduce

## Illustration exemple 1 (abstrait)

### 1) Map :

`emit(this.forme, this.score)`



`emit("cercle", 2)`



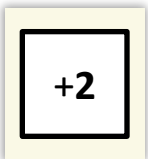
`emit("triangle", 3)`



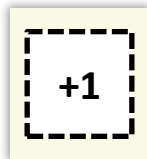
`emit("triangle", 7)`



`emit("cercle", 4)`



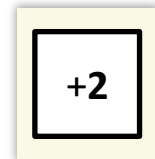
`emit("carre", 2)`



`emit("carre", 1)`



`emit("triangle", 5)`



`emit("carre", 2)`

"cercle" : 2 , 4

"triangle" : 3 , 7 , 5

"carre" : 2 , 1 , 2



# Map & Reduce

## Illustration exemple 1 (abstrait)

2) Reduce : chaque (clé, values)

sum(values)

"cercle" : 2 , 4

"triangle" : 3 , 7 , 5

"carre" : 2 , 1 , 2



# Map & Reduce

## Illustration exemple 1 (abstrait)

2) Reduce : chaque (clé, values)

sum(values)

"cercle" : 2 , 4

sum([2, 4])

"triangle" : 3 , 7 , 5

sum([3, 7, 5])

"carre" : 2 , 1 , 2

sum([2, 1, 2])



# Map & Reduce

## Illustration exemple 1 (abstrait)

2) Reduce : chaque (clé, values)

sum(values)

"cercle" : 2 , 4

sum([2, 4])



6



6

"triangle" : 3 , 7 , 5

sum([3, 7, 5])



15



15

"carre" : 2 , 1 , 2

sum([2, 1, 2])



5



5



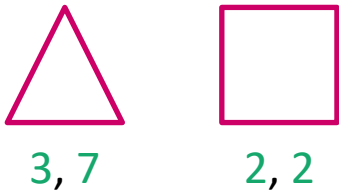
# Map & Reduce

## Exemple 2 (abstrait)

### 1) Map :

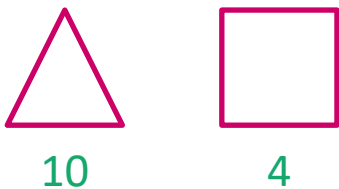
```
if (this.type_trait == "continu")  
    emit(this.forme, this.score)
```

### 2) Shuffle :

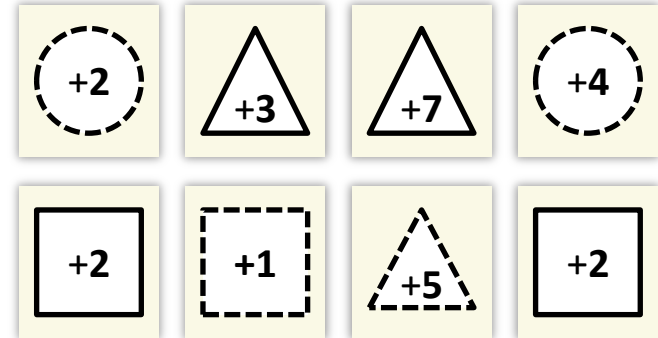


### 3) Reduce : chaque (clé, values)

```
sum(values)
```



### Documents :



Score total pour chaque forme  
quand le trait est continu.

**this** : fait référence au document  
**Map & Reduce** : sont 2 fonctions



# Map & Reduce

## Exemple 3 (abstrait)

### 1) Map :

if (this.score > 2)

emit(this.forme, 1)

### 2) Shuffle :



1



1, 1, 1

### 3) Reduce : chaque (clé, values)

values.length

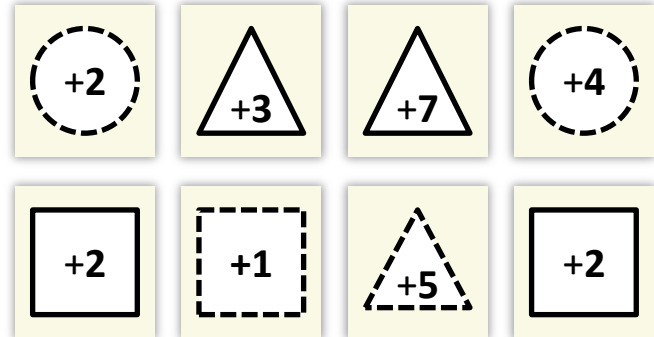


1



3

Documents :



Nombre de documents pour  
chaque forme quand le score > 2.

this : fait référence au document  
Map & Reduce : sont 2 fonctions





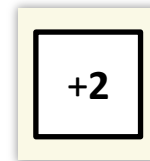
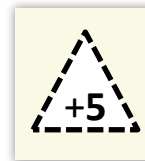
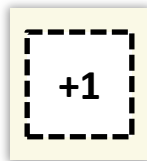
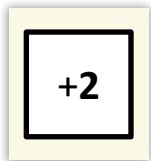
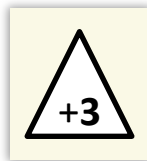
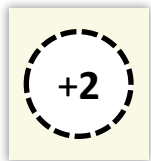
# Map & Reduce

## Illustration exemple 3 (abstrait)

### 1) Map :

if (this.score > 2)

emit(this.forme, 1)





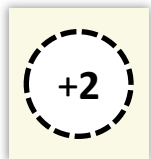
# Map & Reduce

## Illustration exemple 3 (abstrait)

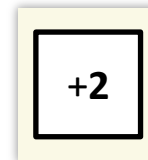
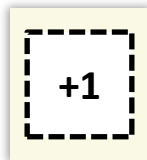
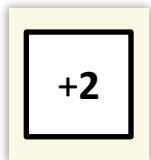
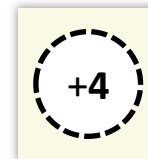
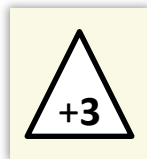
### 1) Map :

if (this.score > 2)

emit(this.forme, 1)



$\leq 2$





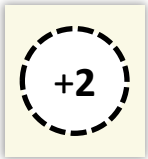
# Map & Reduce

## Illustration exemple 3 (abstrait)

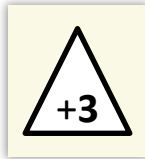
### 1) Map :

if (this.score > 2)

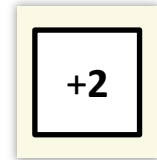
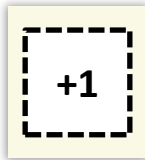
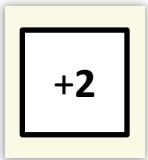
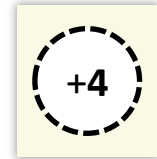
emit(this.forme, 1)



$\leq 2$



emit("triangle", 1)



"triangle" : 1



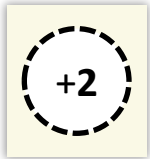
# Map & Reduce

## Illustration exemple 3 (abstrait)

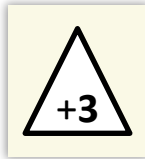
### 1) Map :

if (this.score > 2)

emit(this.forme, 1)



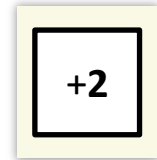
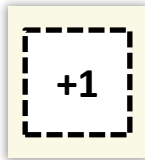
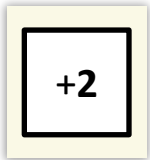
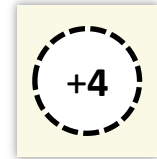
$\leq 2$



emit("triangle", 1)



emit("triangle", 1)



"triangle" : 1 , 1



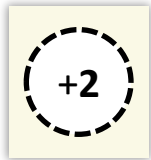
# Map & Reduce

## Illustration exemple 3 (abstrait)

### 1) Map :

if (this.score > 2)

emit(this.forme, 1)



$\leq 2$



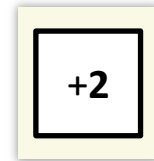
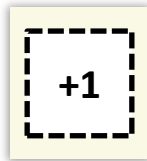
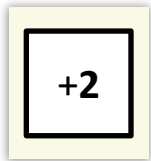
emit("triangle", 1)



emit("triangle", 1)



emit("cercle", 1)



"triangle" : 1 , 1

"cercle" : 1



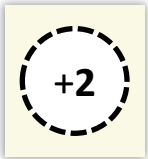
# Map & Reduce

## Illustration exemple 3 (abstrait)

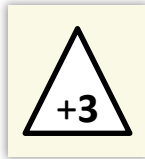
### 1) Map :

if (this.score > 2)

emit(this.forme, 1)



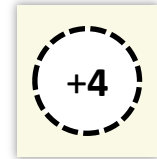
$\leq 2$



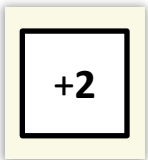
emit("triangle", 1)



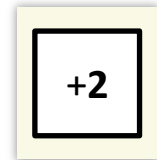
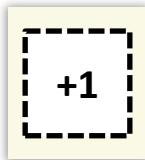
emit("triangle", 1)



emit("cercle", 1)



$\leq 2$



"triangle" : 1 , 1

"cercle" : 1



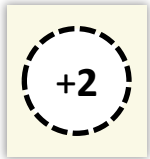
# Map & Reduce

## Illustration exemple 3 (abstrait)

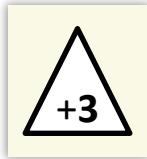
### 1) Map :

if (this.score > 2)

emit(this.forme, 1)



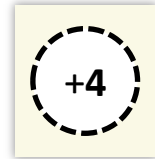
$\leq 2$



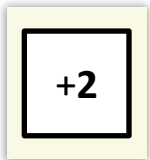
emit("triangle", 1)



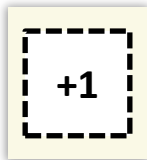
emit("triangle", 1)



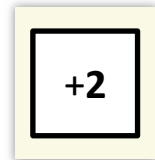
emit("cercle", 1)



$\leq 2$



$\leq 2$



"triangle" : 1 , 1

"cercle" : 1



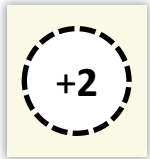
# Map & Reduce

## Illustration exemple 3 (abstrait)

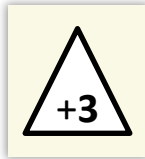
### 1) Map :

if (this.score > 2)

emit(this.forme, 1)



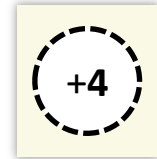
$\leq 2$



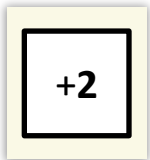
emit("triangle", 1)



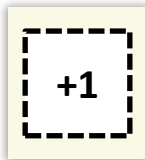
emit("triangle", 1)



emit("cercle", 1)



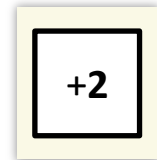
$\leq 2$



$\leq 2$



emit("triangle", 1)



"triangle" : 1 , 1 , 1

"cercle" : 1





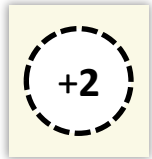
# Map & Reduce

## Illustration exemple 3 (abstrait)

### 1) Map :

if (this.score > 2)

emit(this.forme, 1)



≤ 2



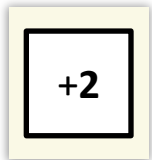
emit("triangle", 1)



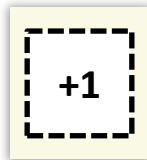
emit("triangle", 1)



emit("cercle", 1)



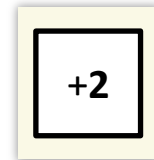
≤ 2



≤ 2



emit("triangle", 1)



≤ 2

"triangle" : 1 , 1 , 1

"cercle" : 1



# Map & Reduce

## Illustration exemple 3 (abstrait)

2) Reduce : chaque (clé, values)

values.length

"triangle" : 1 , 1 , 1

"cercle" : 1



# Map & Reduce

## Illustration exemple 3 (abstrait)

2) Reduce : chaque (clé, values)

values.length

"triangle" : 1 , 1 , 1

[1, 1, 1].length

"cercle" : 1

[1].length



# Map & Reduce

## Illustration exemple 3 (abstrait)

2) Reduce : chaque (clé, values)

values.length

"triangle" : 1 , 1 , 1

[1, 1, 1].length



3



3

"cercle" : 1

[1].length



1



1



# Map & Reduce

## Exemple 4 (abstrait)

### 1) Map :

`emit((this.score > 3), 1)`

### 2) Shuffle :

**true**                  **false**

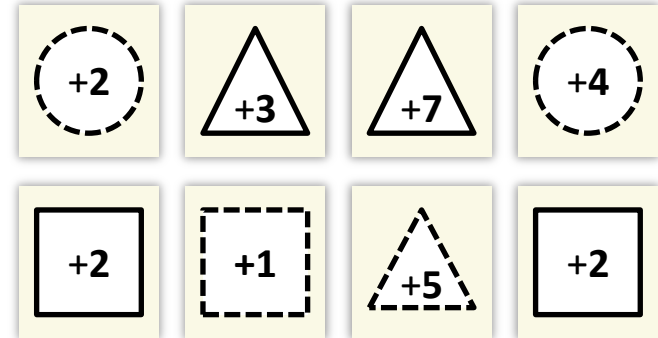
1, 1, 1                  1, 1, 1, 1, 1

### 3) Reduce : chaque (clé, values) values.length

**true**                  **false**

3                          5

Documents :



Nombre des documents qui ont un score > 3 et le nombre des autres documents.

**this** : fait référence au document  
**Map & Reduce** : sont 2 fonctions



# Map & Reduce

## Exemple 4 (abstrait)

### 1) Map :

`emit(this.type_trait, this.score)`

### 2) Shuffle :

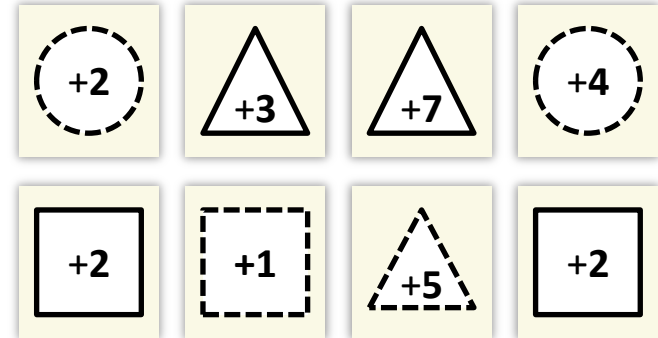
   
3, 7, 2, 2      2, 4, 1, 5

### 3) Reduce : chaque (clé, values)

`sum(values)`

   
14      12

Documents :



Score total pour chaque type de trait.

**this** : fait référence au document  
**Map & Reduce** : sont 2 fonctions



# Map & Reduce

## Map & Reduce (Syntaxe)

1) **var mapF = function()** {codeMap};

Changez **codeMap** par un code JS qui contient **emit(key, value)**.  
On peut changer **key** et **value** par n'importe quelle valeur.  
On peut utiliser la variable **this** pour accéder aux champs du document.

2) **var reduceF = function(key, values)** {codeReduce};

Changez **codeReduce** par un code qui contient **return value**.  
**value** peut être de type objet ou type simple (pas de type liste).  
On peut utiliser le clé du groupe **key** et sa liste de valeurs **values**.

3) **db.nomCollection.mapReduce(mapF, reduceF, {out: {"inline":1}} );**

Changez **nomCollection** par le nom de la collection.

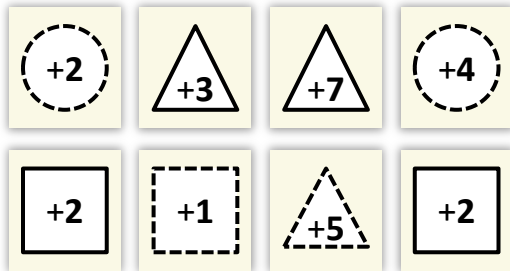


# Map & Reduce

## Exemple 1 (complet)

### Documents

(Collection « **cartes** »)



```
{"_id": "c1", "forme": "cercle", "score": 2, "type_trait": "pointille"}
{"_id": "c2", "forme": "tiangle", "score": 3, "type_trait": "continu"}
{"_id": "c3", "forme": "tiangle", "score": 7, "type_trait": "continu"}
{"_id": "c4", "forme": "cercle", "score": 4, "type_trait": "pointille"}
{"_id": "c5", "forme": "carre", "score": 2, "type_trait": "continu"}
{"_id": "c6", "forme": "carre", "score": 1, "type_trait": "pointille"}
{"_id": "c7", "forme": "tiangle", "score": 5, "type_trait": "pointille"}
{"_id": "c8", "forme": "carre", "score": 2, "type_trait": "continu"}
```

```
var mapF = function() {
  emit(this.forme, this.score);
};

var reduceF = function(key, values) {
  return Array.sum(values);
};
```

```
db.cartes.mapReduce(mapF, reduceF, {out: {"inline":1}} );
```

```
{
  "results": [
    {"_id": "cercle", "value": 6},
    {"_id": "triangle", "value": 15},
    {"_id": "carre", "value": 5}
  ],
  ....
}
```





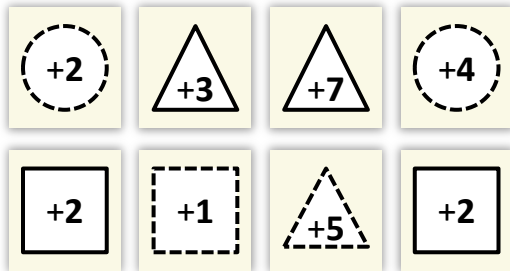


# Map & Reduce

## Exemple 2 (complet)

### Documents

(Collection « **cartes** »)



```
{ "_id": "c1", "forme": "cercle", "score": 2, "type_trait": "pointille" }
{ "_id": "c2", "forme": "tiangle", "score": 3, "type_trait": "continu" }
{ "_id": "c3", "forme": "tiangle", "score": 7, "type_trait": "continu" }
{ "_id": "c4", "forme": "cercle", "score": 4, "type_trait": "pointille" }
{ "_id": "c5", "forme": "carre", "score": 2, "type_trait": "continu" }
{ "_id": "c6", "forme": "carre", "score": 1, "type_trait": "pointille" }
{ "_id": "c7", "forme": "tiangle", "score": 5, "type_trait": "pointille" }
{ "_id": "c8", "forme": "carre", "score": 2, "type_trait": "continu" }
```

```
var mapF = function() {
  emit(this.forme, 1);
};

var reduceF = function(key, values) {
  return values.length;
};
```

```
db.cartes.mapReduce(mapF, reduceF, {out: {"inline":1}} );
```

```
{
  "results": [
    { "_id": "cercle", "value": 2 },
    { "_id": "triangle", "value": 3 },
    { "_id": "carre", "value": 3 }
  ],
  ....
}
```





# Map & Reduce

## Code JS

- Créer une variable de type liste (Array) :

```
var mylist = [];
```

```
mylist.push(val1);
```

```
mylist.push(val2);
```

```
// Changez vali par une valeur de n'importe quelle type.
```

- Accéder à la valeur d'un élément d'une liste :

```
mylist[ind]
```

```
// Changez ind par l'indice (position) de l'élément dans la liste (entier : 0, 1, 2, ...).
```

- Parcourir les éléments d'une liste :

```
for (var i=0; i<mylist.length; i++) {
```

```
    // Changez mylist par quelque chose qui retourne une valeur de type liste.
```

```
    // i représente l'indice de l'élément actuel (entier : 0, 1, 2, ...).
```

```
    // On peut accéder à chaque élément par mylist[i].
```

```
}
```



# Map & Reduce

## Code JS

- Taille d'une liste :  
`mylist.length();`
- Somme des éléments (nombres) d'une liste :  
`Array.sum(mylist)`
- Créer une variable de type objet (object/document) :  
`var myobj = {};`  
`myobj.champ1 = val1;`  
`myobj.champ2 = val2;`  
*// Changez champ<sub>i</sub> par le nom d'un champ.*
- Créer une variable de type objet (document) dans une seule ligne :  
`var myobj = {"champ1": val1, "champ2": val2};`
- Accéder à la valeur d'un champ :  
`myobj.champ`



# Map & Reduce

## Code JS

- Exécuter un code selon une condition :  

```
if (condition) {  
    // code sous condition  
}
```
- Vérifier l'existence d'un champ :  

```
if (myobj.champ1) {  
    // code sous condition  
}
```
- Répéter tant que la condition est vraie (boucle) :  

```
while (condition) {  
    // code sous condition  
}
```



# Map & Reduce

## Exercices TP4 (Map & Reduce)

1. Dans une invite de commande, démarrez un serveur simple.

→ **mongod** --port **27017** --dbpath **c:/data/db**

Serveur  
(mongod)



NomPC:27017  
C:/data/db

2. Dans une autre invite de commande, importez les documents du fichier **dblp.json** dans la collection **dblp** dans **ce serveur** sous la base **mydb**.  
→ **mongoimport** -d **mydb** -c **dblp** --port **27017** --file **c:/dblp.json** --jsonArray
3. Connectez-vous à **ce serveur** (porte **27017**) par l'outil **Robo 3T**.



# Map & Reduce

## Exercices TP4 (Map & Reduce)

4. Calculez le nombre des documents pour chaque **type** dans la collection **dblp**.
5. Calculez le nombre total d'**authors** dans les documents de chaque **series** quand **year** est égale à **2015**.
6. Calculez le nombre de documents de "**Oliver Rose**" (quand il est trouvé dans la liste d'**authors**) pour chaque **year**.

# Séance 5 >



# Exercices

## Exercices TP5

1. Dans une invite de commande, démarrez un serveur simple.

→ **mongod** --port **27017** --dbpath **c:/data/db**

Serveur  
(mongod)



NomPC:27017  
C:/data/db

2. Dans une autre invite de commande, importez les documents du fichier **movies.json** dans la collection **movies** à **ce serveur** sous la base **mydb**.  
→ **mongoimport** -d **mydb** -c **movies** --port **27017** --file **c:/movies.json** --jsonArray
3. Connectez-vous à **ce serveur** (porte **27017**) par l'outil **Robo 3T**.





# Exercices

## Exercices TP5

### Requêtes simples :

4. Trouvez les films quand le **last\_name** du **director** est "**Cameron**", ou quand **year** du film  $\geq$  **2005**.
5. Ajoutez un acteur de votre choix à la liste **actors** (avec un champ **role**, **last\_name** et **first\_name**) du film qui a **\_id** "**movie:19**".

### Map & Reduce :

6. Calculez le nombre de films pour chaque **genre**.
7. Calculez le nombre moyen des **actors** pour chaque **country** quand le **genre** est "**drama**".



# Exercices

## Exercices TP5

### Architecture :

8. Quittez les fenêtres d'invite de commande ouvertes et créez le Replica set « **rstp** », en utilisant le nom de votre machine, avec les composants suivants :

Primaire  
(mongod)



**NomPC:27019**  
C:/data/db/na

Secondaire  
(mongod)



**NomPC:27020**  
C:/data/db/nb



# Exercices

## Exercices TP5

9. Affichez des informations sur le Replica set créé.

# Séance 6 >



# Indexation

## Définition

Imaginez si les maisons étaient sans adresses...



Je dois frapper à toutes les portes jusqu'à ce que je trouve mon ami « Ali » !



# Indexation

## Définition

Mais quand les maisons sont indexées par **adresse**,



J'ai un indice!  
Je peux aller directement chez lui.

**Index** : permet un accès rapide.



# Indexation

## Indexation

- Index par défaut est « **\_id** ».
- Indexer les données par un champ :  
`db.nomCollection.createIndex({"index": 1})`  
Changez **index** par le champ d'indexation.
- Afficher des statistiques sur l'exécution :  
`requete(...).explain("executionStats")`  
changez **requete(...)** par la requête à analyser.





# Indexation

## Astuces

- Pour connaître le nombre de documents examinés par la requête après l'utilisation de **.explain("executionStats")** :

Key	Value
▼ (1)	{ 4 fields }
> queryPlanner	{ 6 fields }
▼ executionStats	{ 6 fields }
executionSuccess	true
nReturned	9
executionTimeMillis	0
totalKeysExamined	9
totalDocsExamined	9
> executionStages	{ 14 fields }
> serverInfo	{ 4 fields }
ok	1.0





# Indexation

## Exercices TP6

1. Dans une invite de commande, démarrez un serveur simple.

→ **mongod** --port **27017** --dbpath **c:/data/db**

Serveur  
(mongod)



NomPC:27017  
C:/data/db

2. Dans une autre invite de commande, à **ce serveur** sous la base **mydb**, importez les documents du fichier **moviesRef.json** dans la collection **moviesRef** et les documents du fichier **artists.json** dans la collection **artists**.  
→ **mongoimport** -d **mydb** -c **moviesRef** --port **27017** --file **c:/moviesRef.json** --jsonArray  
→ **mongoimport** -d **mydb** -c **artists** --port **27017** --file **c:/artists.json** --jsonArray
3. Connectez-vous à **ce serveur** (porte **27017**) par l'outil **Robo 3T**.



# Indexation

## Exercices TP6

### Indexation :

4. Par une requête simple, trouvez les **artists** qui ont **birth\_date** égale à **1964** en affichant les statistiques d'exécution (voir diapos 102). Combien de documents examinés par cette requête? (voir diapo 103)
5. Indexez les **artists** par le champ **birth\_date** (voir diapo 102).
6. Répétez la question (4).

### Map & Reduce :

7. Dans la collection **moviesRef**, calculez le nombre de films pour chaque acteur. (pour chaque **\_id** de leurs **actors**)
8. Au lieu du nombre de films, retournez les titres des films pour chaque acteur :
  - a) Modifiez la valeur à collecter dans Map pour collecter les **title** des films.
  - b) Modifiez Reduce pour retourner un objet qui contient un champ « **films** » qui prend comme valeur la liste des titres collectés.



# Indexation

## Exercices TP6

### Requêtes simples :

9. Trouvez les films du directeur Quentin Tarantino :

- **NB :** Dans **moviesRef**, on référence les informations des **artists** par **\_id**.

```
"director": {  
  "_id": "artist:3"  
},  
"actors": [  
  {  
    "_id": "artist:15",  
    "role": "John Ferguson"  
  },  
]
```

- Trouvez l'artiste qui a **first\_name** égale à **"Quentin"** et **last\_name** égale à **"Tarantino"** par la requête **findOne** dans la collection **artists**.
- Assignez cette requête à une variable. Donc, à travers de cette variable on peut accéder aux champs de l'artiste trouvé.
- Trouvez tous les films quand cet artiste est directeur. C'est-à-dire, dans la collection **moviesRef**, quand **\_id** du **director** du film est égale à **\_id** de l'artiste trouvé.