
BIG DATA : INTRODUCTION A MONGODB

G. Mopolo-Moké

prof. PAST

Université Nice Sophia Antipolis

2021 / 2022

Diapositive 1

GMM1 Gabriel Mopolo Moke; 21/04/2019

Plan Général

- **1. Objectifs du cours**
- **2. Carte de visite de MongoDB**
- **3. Installation MongoDB Community Edition**
- **4. Les outils MongoDB**
- **5. Architecture d'une Instance MongoDB**
- **6. Gestion des Bases de données MongoDB**
- **7. Gestion des Collections MongoDB**
- **8. Gestion des documents MongoDB**
- **9. Indexation de documents MongoDB**
- **10. Recherche de documents MongoDB**
- **11. Les agrégats dans MongoDB**
- **12. API Java MongoDB**
- **13. Quiz**
- **14. Bilan et perspectives**
- **15. Webographie et Bibliographie**
- **16. Exercices**

1. Objectifs du cours

- Comprendre l'architecture d'une instance mongoDB
- Comprendre les principales caractéristiques de MongoDB et savoir positionner cette base de données par rapport aux autres BD NoSQL
- Pouvoir gérer des bases de données
- Pouvoir gérer des collections
- Pouvoir gérer des documents
- Comprendre le rôle du format JSON pour ce moteur
- Comprendre sa stratégie de réPLICATION
- Savoir poser des index secondaires
- Savoir accéder à un ou plusieurs aux objets (via la clé, via des prédictats, via un index)
- Pouvoir effectuer des jointures

2. Carte de visite de MongoDB

□ Plan

- Editeur
- Versions initial (Nr. De version et date)
- Versions actuelle (Nr. De version et date)
- Modèles de données supportés(Clé/Valeur, Orienté document, Orienté Colonnes, Oriénté Graphe)
- Gestion du schéma (sans schéma, dynamique, statique, mixte)
- Typage (none, static, dynamique)
- Support de SQL (DDL, DML)
- Support d'indexes secondaires
- Langage de développement du sgbd nosql
- Support / Pérénité (communauté , etc....)
- API Supportés
- Théorème CAP (CP, AP, AC)
- Méthode de partitionnement
- Méthode de réPLICATION
- Concept de consistance
- Concept de durabilité
- Clés étrangères
- Support de références (REF)
- Licences et prix

2. Carte de visite de MongoDB

□ Plan suite

- Différents types de versions (communautaire, entreprise, ...)
- Audience dans le marché
- Tables et tables filles
- Clé avec major et minor key
- Gestion des utilisateurs
- Gestion des droits
- Gestion des namespaces ou databases
- Systèmes d'exploitations supportés
- Disponible en mode DBaaS
- Applications communautaires l'utilisant
- Domaines d'applications
- Architecture du moteur NoSql
- Montée en charge
- Gestion de la disponibilité

2. Carte de visite de MongoDB

- Editeur : MongoDB inc.**
- Versions initial (Nr. De version et date) : 2009**
- Versions actuelle (Nr. De version et date) : 4.0.8, March 2019**
- Modèles de données supportés(Clé/Valeur, Orienté document, Orienté Colonnes, Oriénté Graphe): Orienté document**
- Gestion du schéma (sans schéma, dynamique, statique, mixte) :Sans schéma**
- Typage (none, static, dynamique): dynamique**

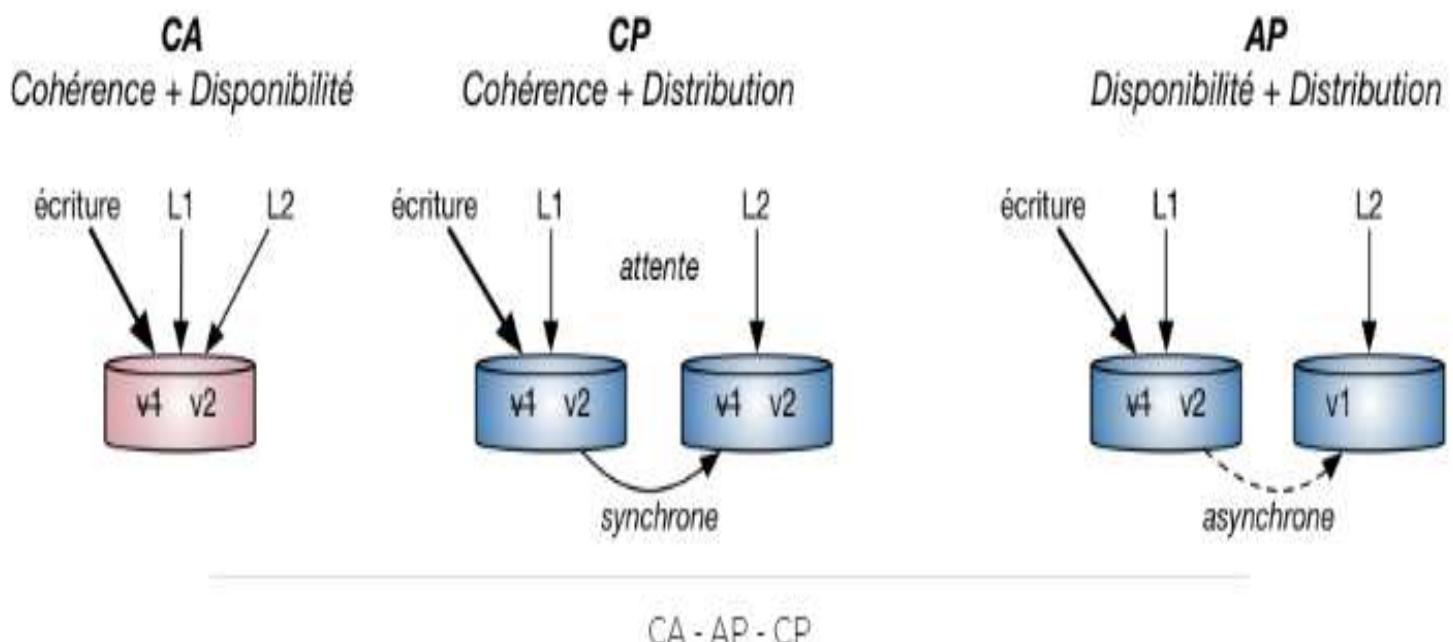
2. Carte de visite de MongoDB

- **Support de SQL (DDL, DML) : NON**, Read-only SQL queries via the MongoDB Connector for BI
- **Support d'indexes secondaires : OUI**
- **Langage de développement du sgbd nosql : C++**
- **Support / Pérénité (communauté , etc....) : MongoDB Inc., Communauté**
- **License : Open Source**
- **API Supportées :**
 - Actionscript, C, C#, C++, Clojure, ColdFusion, D,
 - Dart, Delphi, Erlang, Go, Groovy, Haskell,
 - Java, JavaScript, Lisp, Lua, MatLab, Perl, PHP,
 - PowerShell, Prolog, Python, R, Ruby, Scala, Smalltalk

2. Carte de visite de MongoDB

□ Théorème CAP (CP, AP, AC)

- MongoDB supporte : CP
- **CP** : Propose de distribuer les données sur plusieurs serveurs en garantissant la tolérance aux pannes (réPLICATION). Exemple : MongoDB, etc. Attention il y a des attentes. L2 doit attendre que V2 soit répliqué avant de lire. L1: Lecture 1, L2 : Lecture 2



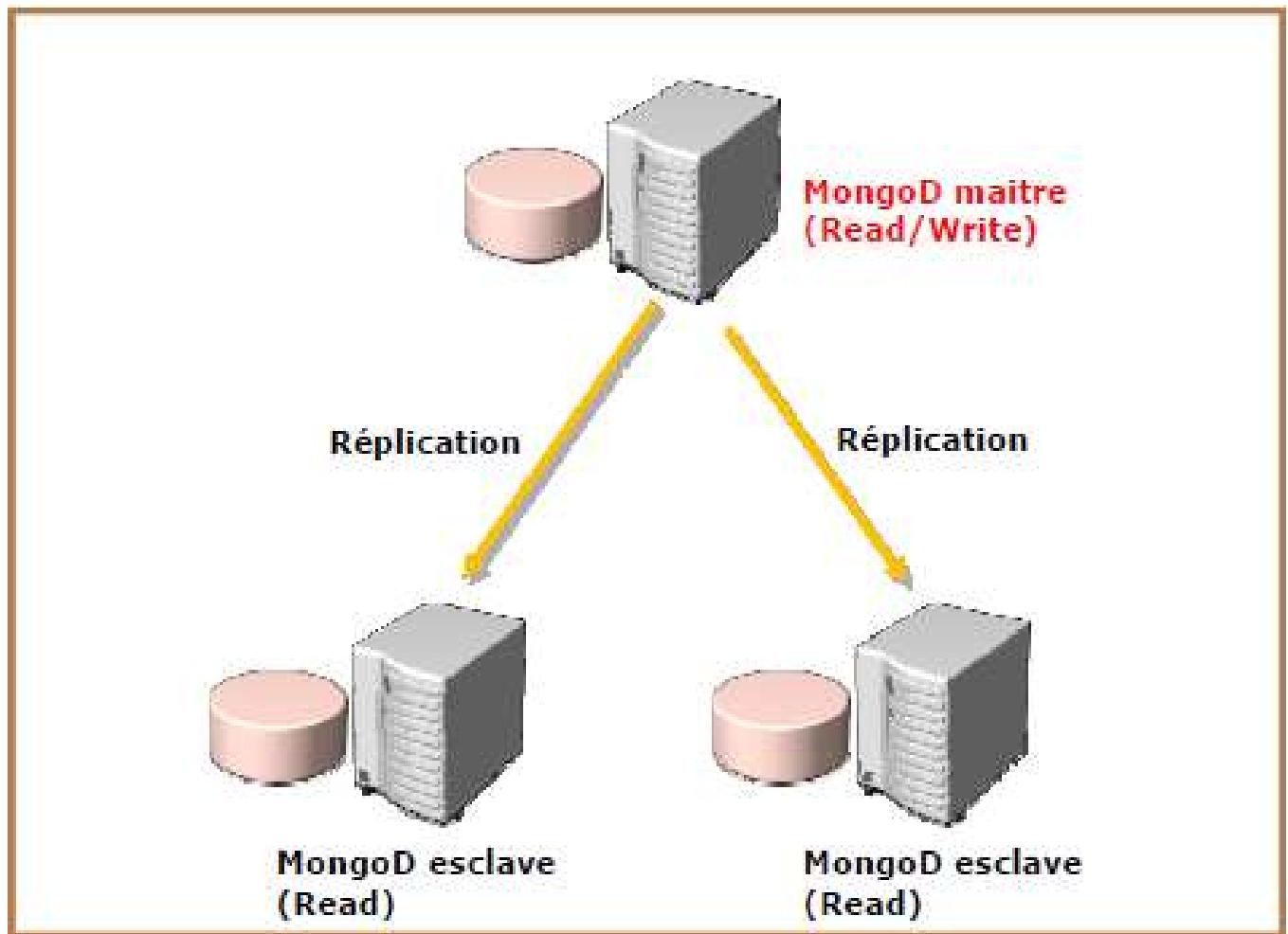
2. Carte de visite de MongoDB

□ Méthode de partitionnement : SHARDING

- Une clé de Sharding permet de repartir les données des collections sur plusieurs SHARD

2. Carte de visite de MongoDB

- Méthode de réPLICATION : Master-slave replication



2. Carte de visite de MongoDB

□ Concept de consistance

2. Carte de visite de MongoDB

□ Concept de durabilité

2. Carte de visite de MongoDB

- Clés étrangères : **NON mais gestion par l'utilisateur pour lier des documents**
- Support de références (REF) : **OUI**

2. Carte de visite de MongoDB

- Licences et prix : OpenSource, N/A
- Différents types de versions (communautaire, entreprise, ...): Les deux
- Audience dans le marché : Le plus populaires des moteur NoSQL

| Rank | | | DBMS | Database Model |
|----------|----------|----------|----------------------|----------------------------|
| Mar 2019 | Feb 2019 | Mar 2018 | | |
| 1. | 1. | 1. | Oracle | Relational, Multi-model |
| 2. | 2. | 2. | MySQL | Relational, Multi-model |
| 3. | 3. | 3. | Microsoft SQL Server | Relational, Multi-model |
| 4. | 4. | 4. | PostgreSQL | Relational, Multi-model |
| 5. | 5. | 5. | MongoDB | Document |
| 6. | 6. | 6. | IBM Db2 | Relational, Multi-model |
| 7. | ↑ 9. | 7. | Microsoft Access | Relational |
| 8. | ↓ 7. | 8. | Redis | Key-value, Multi-model |
| 9. | ↓ 8. | 9. | Elasticsearch | Search engine, Multi-model |
| 10. | 10. | ↑ 11. | SQLite | Relational |
| 11. | 11. | ↓ 10. | Cassandra | Wide column |
| 12. | 12. | ↑ 15. | MariaDB | Relational, Multi-model |
| 13. | 13. | 13. | Splunk | Search engine |
| 14. | 14. | ↓ 12. | Teradata | Relational |
| 15. | 15. | ↑ 18. | Hive | Relational |
| etc. | etc. | etc. | etc. | etc. |

2. Carte de visite de MongoDB

- Tables et tables filles : Pas de tables
- Clé avec major et minor key : Pas vu

2. Carte de visite de MongoDB

- **Gestion des utilisateurs : OUI, par défaut non, il est possible d'en créer**
- **Gestion des droits : OUI, uniquement si on a créé des utilisateurs :**
db.createUser(userDocument)
- **Gestion des namespaces ou databases : OUI, il est possible de créer des bases de données et des collections**

2. Carte de visite de MongoDB

- Systèmes d'exploitations supportés : Linux, Windows et MacOS
- Disponible en mode Dbaas : OUI, disponible en mode cloud computing

2. Carte de visite de MongoDB

- Applications communautaires l'utilisant**
- Domaines d'applications**

2. Carte de visite de MongoDB

□ Architecture du moteur NoSql :

- Organisé en SHARD
- Une SHARD contient un serveur maître et des serveurs esclave
- Une instance MongoDB comprend 1 ou plusieurs Bases de Données
- Une base de données comprends 0, 1 ou plusieurs collections
- Une collections contient 0,1 ou plusieurs documents

□ Montée en charge : Possibilité d'ajout de noeud

□ Gestion de la disponibilité : en cas de panne du maître, élection d'un nouveau maître parmi les nœud esclave

3. Installation MongoDB Community Edition

□ Plan

- Installation sous Windows
- Installation sous Linux Ubuntu / Debian
- Installation sous macOS

3. Installation MongoDB Community Edition

□ Installation sous Windows

- Etape 1 : Télécharger l'archive à partir de <https://www.mongodb.com/download-center/community>
- Etape 2 : Décompresser l'archive
- Etape 3 : Création de l'arborescence de stockage de l'instance par défaut
- Etape 4 : Configuration du path
- Etape 5 : Création du service d'arrêt/démarrage de Windows
- Etape 6 : Démarrage le serveur MongoDB
- Etape 7: Lancer le client shell mongo

3. Installation MongoDB Community Edition

□ Installation sous Linux Ubuntu / Debian

- Pour les deux distributions**

- 1) Importer la clé publique utilisée par le système de gestion des paquets.

```
$ sudo apt-key adv --keyserver  
hkp://keyserver.ubuntu.com:80 --recv  
9DA31620334BD75D9DCB49F368818C7  
2E52529D4
```

- 2) Créer un fichier "mongodb-org-4.0.list" pour MongoDB dans le dossier /etc/apt/sources.list.d/

```
$ cd /etc/apt/sources.list.d/  
$ touch mongodb-org-4.0.list
```

3. Installation MongoDB Community Edition

□ Installation sous Linux Ubuntu / Debian

- Pour UBUNTU**

- Pour Ubuntu 18.04 (Bionic) exécuter ce qui suit :

```
$ echo "deb [ arch=amd64 ]  
https://repo.mongodb.org/apt/ubuntu bionic/mongodb-  
org/4.0 multiverse" | sudo tee  
/etc/apt/sources.list.d/mongodb-org-4.0.list
```

- Pour Ubuntu 16.04 (Xenial) exécuter ce qui suit :

```
$ echo "deb [ arch=amd64,arm64 ]  
https://repo.mongodb.org/apt/ubuntu xenial/mongodb-  
org/4.0 multiverse" | sudo tee  
/etc/apt/sources.list.d/mongodb-org-4.0.list
```

- Pour Ubuntu 14.04 (Trusty) exécuter ce qui suit :

```
$ echo "deb [ arch=amd64 ]  
https://repo.mongodb.org/apt/ubuntu trusty/mongodb-  
org/4.0 multiverse" | sudo tee  
/etc/apt/sources.list.d/mongodb-org-4.0.list
```

3. Installation MongoDB Community Edition

□ Installation sous Linux Ubuntu / Debian

- Pour DEBIAN**

- Pour Debian 8 (Jessie) exécuter ce qui suit :

```
$ echo "deb http://repo.mongodb.org/apt/debian  
jessie/mongodb-org/4.0 main" | sudo tee  
/etc/apt/sources.list.d/mongodb-org-4.0.list
```

- Pour Debian 9 (Stretch) exécuter ce qui suit :

```
$ echo "deb http://repo.mongodb.org/apt/debian  
stretch/mongodb-org/4.0 main" | sudo tee  
/etc/apt/sources.list.d/mongodb-org-4.0.list
```

- Remarque: Actuellement, des paquets sont disponibles pour Debian 8 "Jessie" et Debian 9 "Stretch".

3. Installation MongoDB Community Edition

□ Installation sous Linux Ubuntu / Debian

- **Pour les deux distributions**

1) Recharger la base de données des paquets locaux

\$ sudo apt-get update

2) Installer les paquets MongoDB. Pour installer la dernière version stable, exécutez ce qui suit:

\$ sudo apt-get install -y mongodb-org

3. Installation MongoDB Community Edition

□ Installation sous Linux Ubuntu / Debian

- **Répertoire mongoDB et localisation de certains fichiers**
 - **Repertoire de mongoDb**
 - Si vous l'avez installé via le gestionnaire de paquets, le répertoire de données /var/lib/mongodb et le répertoire log /var/log/mongodb sont créés pendant l'installation.
 - **Fichier de configuration**
 - Le package officiel de MongoDB comprend un fichier de configuration (/etc/mongod.conf)
 - Ces paramètres (tels que les spécifications du répertoire de données et du répertoire de journaux) prennent effet au démarrage.

3. Installation MongoDB Community Edition

□ Installation sous Linux Ubuntu / Debian

- **Démarrage, Vérification, Arrêt de MongoDB**

1) Démarrer le serveur MongoDB

\$ sudo service mongod start

2) Vérifier que le serveur MongoDB a bien démarré avec succès

Vérifiez que le processus mongod a bien démarré en vérifiant le contenu du fichier journal dans /var/log/mongodb/mongodod.log pour une lecture de ligne:
[initandlisten] waiting for connections on port 27017

3) Arrêter le serveur MongoDB

\$ sudo service mongod stop

4) Redémarrer le serveur MongoDB

\$ sudo service mongod restart

3. Installation MongoDB Community Edition

□ Installation sous Linux Ubuntu / Debian

- **Lancer le shell mongoDB**

\$ mongo

3. Installation MongoDB Community Edition

□ Installation macOS

- MongoDB ne supporte que les versions MacOS 10.11 et ultérieures sur Intel x86-64.
- **Prérequis:** Si vous n'avez pas encore ajouté la source référentiel du dépôt de formules officielles de mongoDB, ajouter la liste des formules à partir d'un terminal, en tapant ce qui suit :

```
$ brew tap mongodb/brew
```

- **Installer ensuite MongoDB**

```
$ brew install mongodb-community@4.0
```

3. Installation MongoDB Community Edition

□ Installation macOS

- Démarrer le serveur MongoDB**

1) Démarrer MongoDB en mode manuel

```
$ mongod --config /usr/local/etc/mongod.conf
```

2) Démarrer MongoDB en mode automatique

```
$ brew services start mongodb-community@4.0
```

- Lancer un shell MongoDB**

```
$ mongo
```

4. Les outils MongoDB

□ Les Outils et utilitaires MongoDB

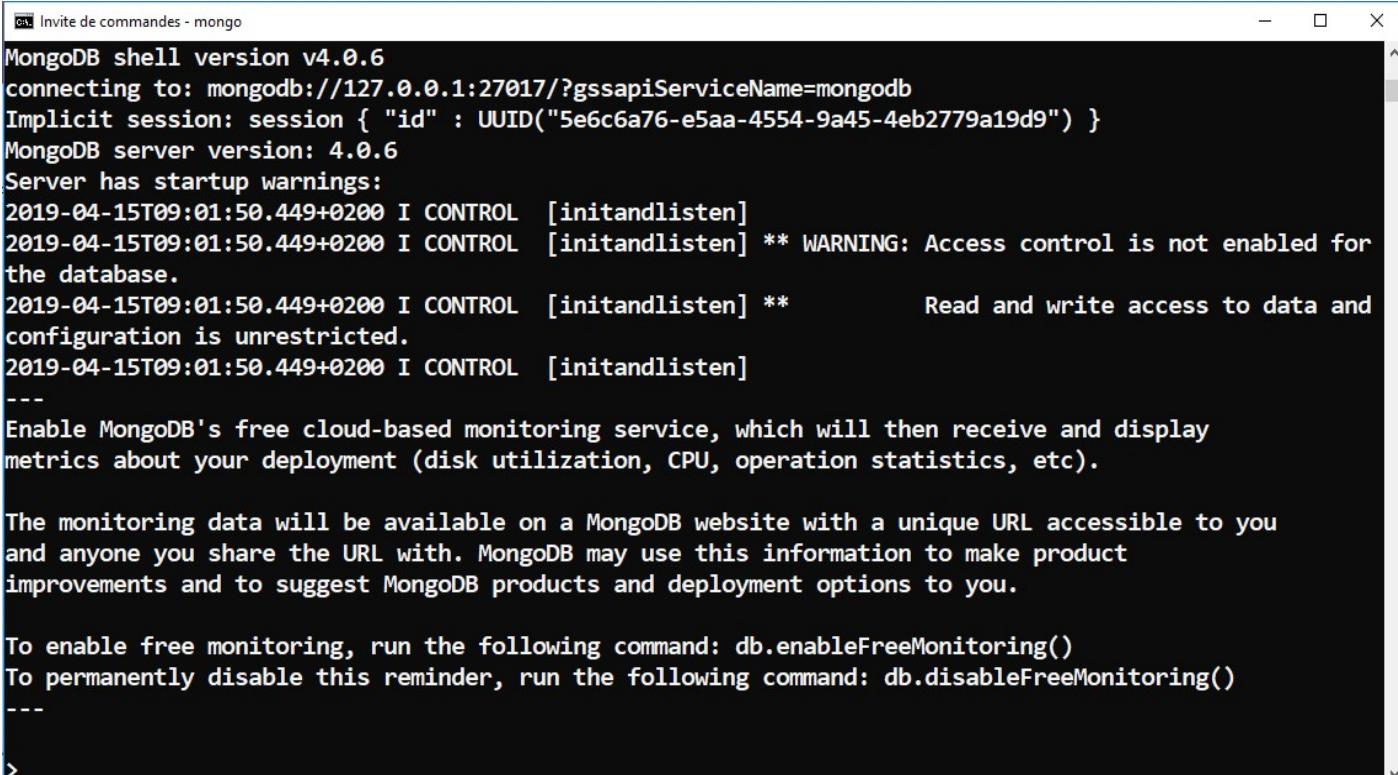
- mongod (le moteur de base)
- mongo (le shell javascript)
- mongos (le contrôleur de Sharding)
- Les outils d'import/export
 - mongoimport,
 - mongoexport,
 - mongodump,
 - mongorestore,
 - bsondump
- mongofiles (l'utilitaire GridFS)
- mongostat (visualisation des stats d'une instance mongoDB)
- mongosniff (le tcpdump de mongo)
- mongotop, mongoperf

4. Les outils MongoDB

□ Lancement du shell mongoDB sous windows

- 1. S'assurer que le server MongoDB a été lancé
- 2. Ouvrir un CMD
- Exécuter ensuite MONGO
- C:\>**mongo**

>



```
MongoDB shell version v4.0.6
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("5e6c6a76-e5aa-4554-9a45-4eb2779a19d9") }
MongoDB server version: 4.0.6
Server has startup warnings:
2019-04-15T09:01:50.449+0200 I CONTROL  [initandlisten]
2019-04-15T09:01:50.449+0200 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for
the database.
2019-04-15T09:01:50.449+0200 I CONTROL  [initandlisten] ** Read and write access to data and
configuration is unrestricted.
2019-04-15T09:01:50.449+0200 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

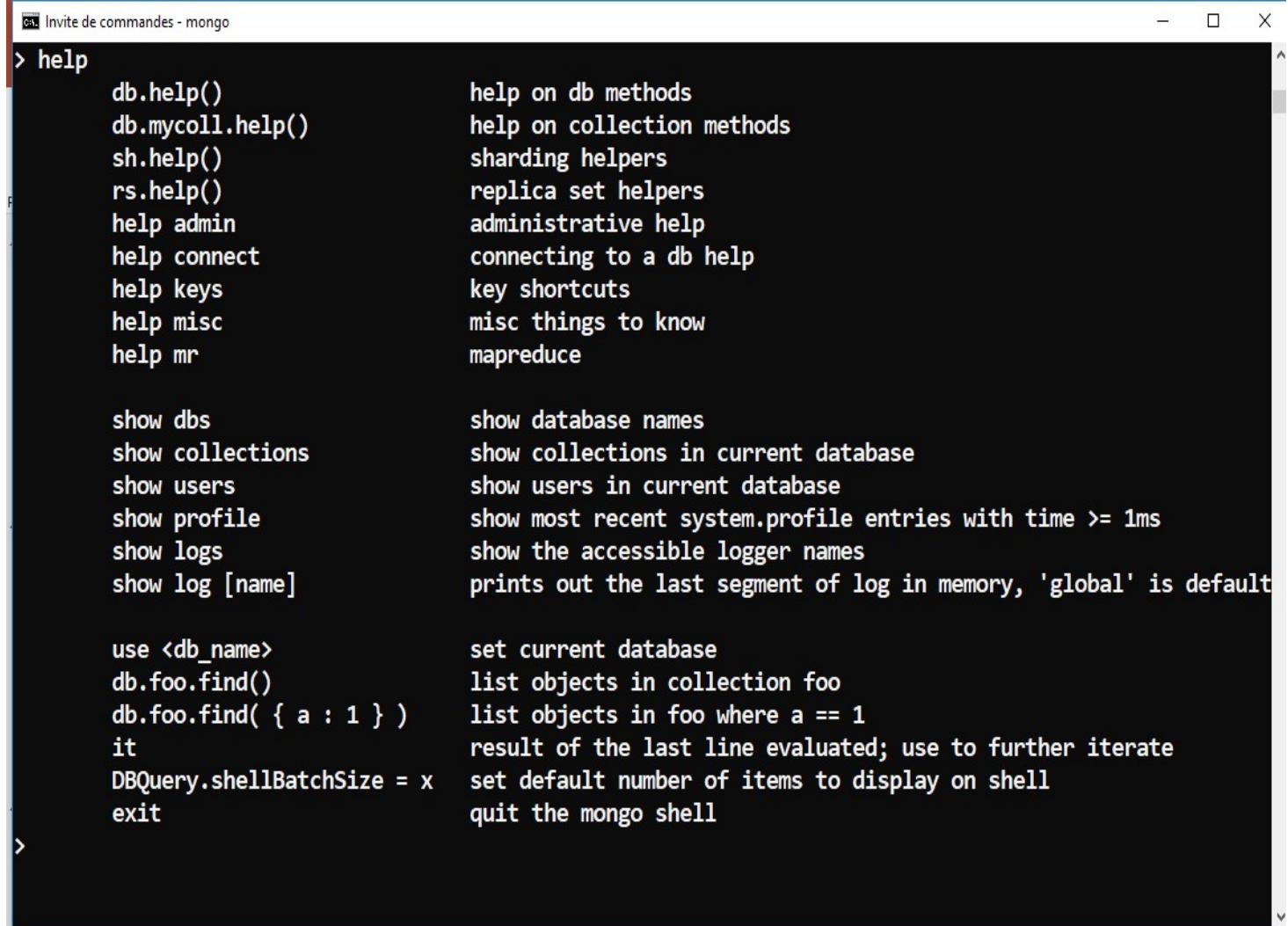
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
```

Note : Sous Linux ou MacOS on lance la même commande. Voir aussi le chapitre installation Linux et MacOS

4. Les outils MongoDB

□ Connaitre les commandes disponibles

□ > help



```
Invité de commandes - mongo
> help
  db.help()                      help on db methods
  db.mycoll.help()                help on collection methods
  sh.help()                       sharding helpers
  rs.help()                       replica set helpers
  help admin                      administrative help
  help connect                    connecting to a db help
  help keys                       key shortcuts
  help misc                       misc things to know
  help mr                         mapreduce

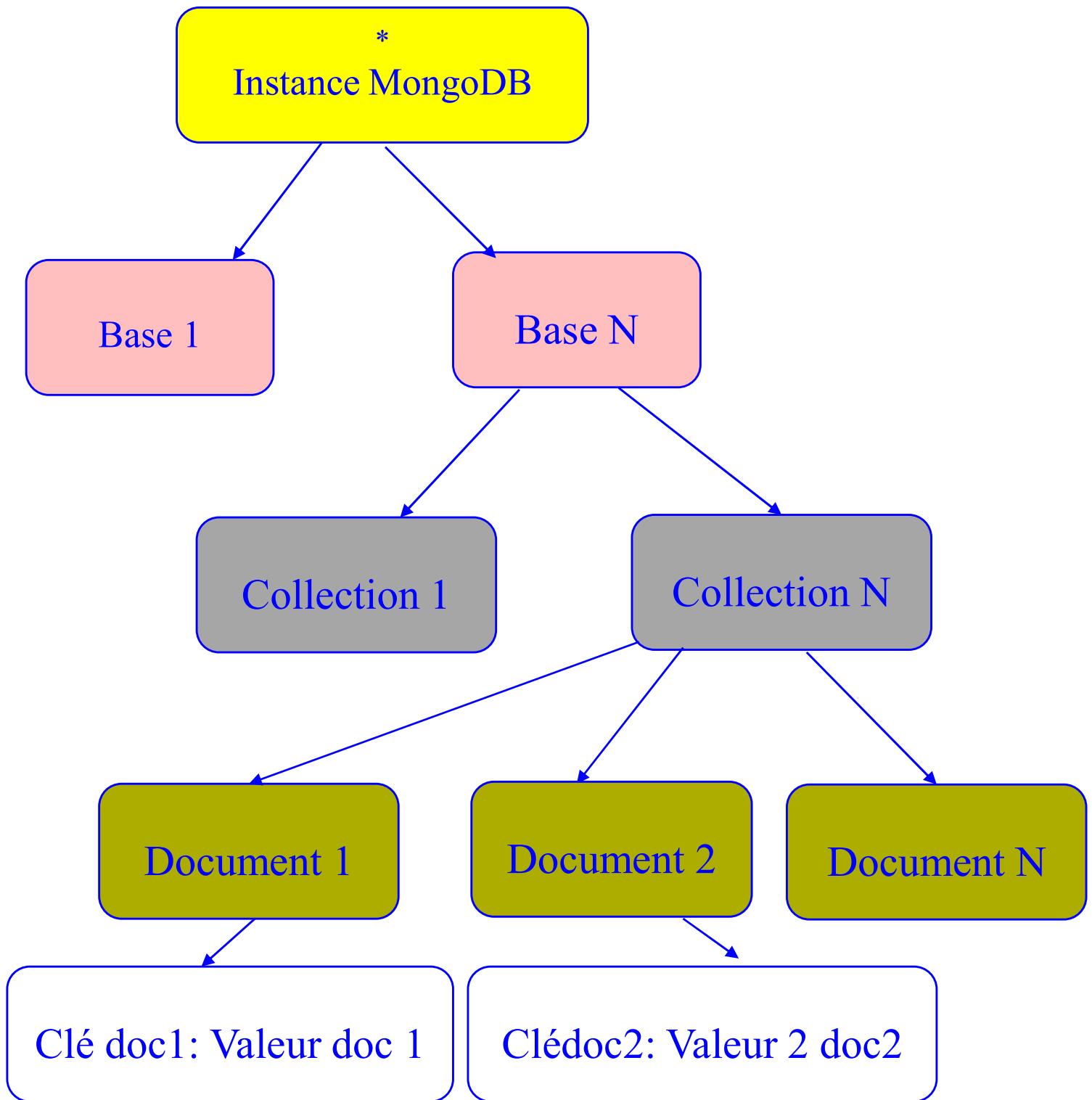
  show dbs                        show database names
  show collections                show collections in current database
  show users                      show users in current database
  show profile                    show most recent system.profile entries with time >= 1ms
  show logs                       show the accessible logger names
  show log [name]                 prints out the last segment of log in memory, 'global' is default

  use <db_name>                  set current database
  db.foo.find()                   list objects in collection foo
  db.foo.find( { a : 1 } )        list objects in foo where a == 1
  it                            result of the last line evaluated; use to further iterate
  DBQuery.shellBatchSize = x     set default number of items to display on shell
  exit                          quit the mongo shell
>
```

- A partir de Mongo 4.4 certains utilitaires sont fournis séparément. Pour les récupérer, suivre le lien :
https://www.mongodb.com/try/download/database-tools?tck=docs_databasetools

5. Architecture d'une Instance MongoDB

□ Architecture



5. Architecture d'une Instance MongoDB

□ Caractéristiques d'une instance

- Un port d'écoute (par défaut 27017)
- Un processus serveur
- Un répertoire racine de stockage
- Un fichier de log
- Un fichier de configuration: mongod.conf

5. Architecture d'une Instance MongoDB

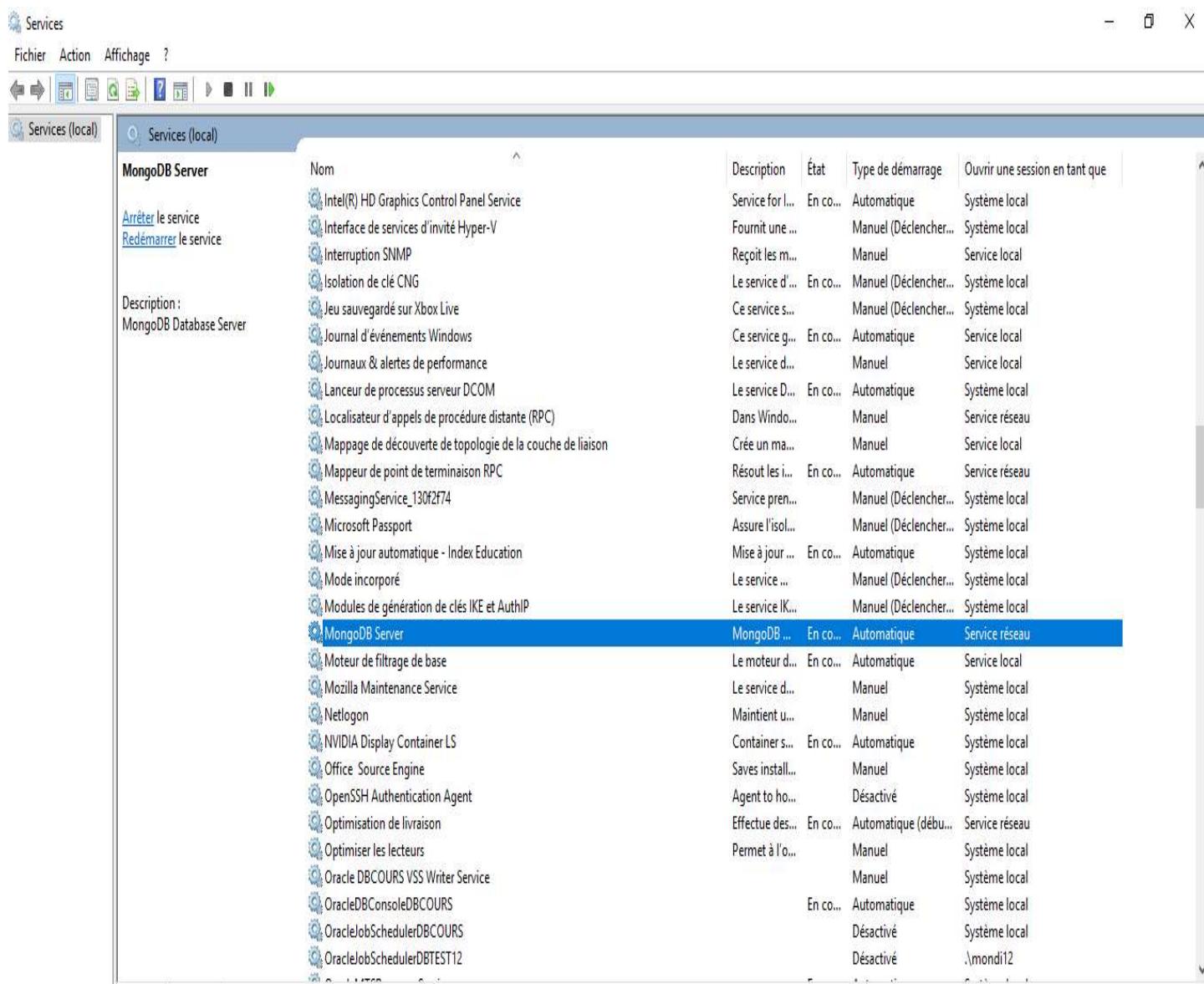
□ Arrêt / démarrage d'une instance sous Linux

- Démarrage
 - /etc/init.d/mongod start (démarrage de mongoDB via script)
 - mongod --dbpath <racine_instance> (démarrage de mongoDB)
 - mongod -f <fichier.conf>
- Arrêt
 - /etc/init.d/mongod stop
 - /etc/init.d/mongod restart (arrêt/relance)
 - Commande db.shutdownServer() à partir du client mongo et la base admin

5. Architecture d'une Instance MongoDB

□ Arrêt / démarrage d'une instance sous Windows. Via les services Windows ou en mode ligne de commandes

- Via les services Windows



5. Architecture d'une Instance MongoDB

□ Arrêt / démarrage d'une instance sous Windows. Via les services Windows ou en mode ligne de commandes

- Via le mode ligne de commandes

```
D:\>net start MongoDB
```

```
D:\>net stop MongoDB
```

```
C:\ Administateur : Invite de commandes
```

```
Microsoft Windows [version 10.0.17134.706]
(c) 2018 Microsoft Corporation. Tous droits réservés.
```

```
C:\Windows\system32>d:
```

```
D:\>net start MongoDB
Le service demandé a déjà été démarré.
```

```
Vous obtiendrez une aide supplémentaire en entrant NET HELPMSG 2182.
```

```
D:\>net stop MongoDB
Le service MongoDB Server s'arrête.
Le service MongoDB Server a été arrêté.
```

```
D:\>
D:\>net start MongoDB
Le service MongoDB Server démarre..
Le service MongoDB Server a démarré.
```

```
D:\>
```

6. Gestion des Bases de données MongoDB

□ Plan

- Lister les bases de données existantes
- Switcher sur une base de données existantes
- Identifier la base courante
- Créer une nouvelle bases de données
- Supprimer une base de données

6. Gestion des Bases de données MongoDB

□ Lister les bases de données existantes

➤ show dbs

```
>
> show dbs
2019-04-21T16:28:55.751+0200 I NETWORK [js] trying reconnect to 127.0.0.1:27017 failed
2019-04-21T16:28:55.752+0200 I NETWORK [js] reconnect 127.0.0.1:27017 ok
admin    0.000GB
airbase   0.000GB
config    0.000GB
local    0.000GB
test     0.000GB
>
```

6. Gestion des Bases de données MongoDB

□ Switcher sur une base de données existantes

- use <NomBase>
- use airbase

```
> show dbs
2019-04-21T16:28:55.751+0200 I NETWORK  [js] trying reconnect to 127.0.0.1:27017 failed
2019-04-21T16:28:55.752+0200 I NETWORK  [js] reconnect 127.0.0.1:27017 ok
admin    0.000GB
airbase   0.000GB
config    0.000GB
local     0.000GB
test      0.000GB
>
>
>
> use airbase
switched to db airbase
>
```

6. Gestion des Bases de données MongoDB

□ Identifier la base de données courante

- La commande à lancer est : db

> db

```
> show dbs
2019-04-21T16:28:55.751+0200 I NETWORK  [js] trying reconnect to 127.0.0.1:27017 failed
2019-04-21T16:28:55.752+0200 I NETWORK  [js] reconnect 127.0.0.1:27017 ok
admin      0.000GB
airbase    0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
>
>
>
> use airbase
switched to db airbase
>
>
> db
airbase
```

6. Gestion des Bases de données MongoDB

□ Créer une nouvelle bases de données

- Crédation **implicite** (il faut au moins y ajouter une collection afin de la créer définitivement)

>use airbase2

>show dbs

Ou >show databases

>db.createCollection('PILOTE');

```
Sélection Invité de commandes - mongo
> show dbs
admin      0.000GB
airbase    0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
>
> use airbase2
switched to db airbase2
>
> show dbs
admin      0.000GB
airbase    0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
>
> db.createCollection('PILOTE');
{ "ok" : 1 }
>
>
> show dbs
admin      0.000GB
airbase    0.000GB
airbase2   0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
>
```

6. Gestion des Bases de données MongoDB

□ Créer une nouvelle bases de données

- Crédation **implicite** lors du lancement du client ligne de commande **mongo**

C:\> **mongo airbase3**

-- il faut moins une collection pour qu'elle persiste

>dbs

>show dbs

>db.createCollection('PILOTE3');

>show dbs

```
> db
airbase3
> show dbs
admin      0.000GB
airbase    0.000GB
airbase2   0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
> db.createCollection('PILOTE3');
{ "ok" : 1 }
> show dbs
admin      0.000GB
airbase    0.000GB
airbase2   0.000GB
airbase3   0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
>
```

6. Gestion des Bases de données MongoDB

□ Supprimer une base de données

- **Approche 1 pour supprimer une base**

C:\> mongo

- use nomBase
- use airbase3
- db.dropDatabase()

```
> show dbs
admin      0.000GB
airbase    0.000GB
airbase2   0.000GB
airbase3   0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
>
>
> db.dropDatabase()
{ "dropped" : "airbase3", "ok" : 1 }
> db
airbase3
> show dbs
admin      0.000GB
airbase    0.000GB
airbase2   0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
```

6. Gestion des Bases de données MongoDB

□ Supprimer une base de données

- **Approche 2 pour supprimer une base**

C:\> mongo

- use nomBase
- use airbase2
- db.runCommand({ dropDatabase: 1 })

```
> show dbs
admin      0.000GB
airbase    0.000GB
airbase2   0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
> use airbase2
switched to db airbase2
> db.runCommand( { dropDatabase: 1 } )
{ "dropped" : "airbase2", "ok" : 1 }
> show dbs
admin      0.000GB
airbase    0.000GB
config     0.000GB
local      0.000GB
test       0.000GB
>
```

7. Gestion des Collections MongoDB

□ Plan

- Définition
- Créer une collection dans une base
- Visualiser des collections dans une base
- Supprimer une collection dans une base

7. Gestion des Collections MongoDB

□ Définition

- Une collection est un container de documents
- Une collection en mongoDB équivaut à peu près à une TABLE dans une BD relationnelle
- Il existe deux approches de création de collection en MongoDB : IMPLICITE ou EXPLICITE

7. Gestion des Collections MongoDB

□ Créer une collection dans une base

- **Création implicite** d'une collection lors de l'insertion d'un document

Syntaxe :

db.<collectionName>.insert(

C:\> mongo

>use airbase

>db.pilote.insert({plnum:2, plnom: "Milou" ,
adr:"Nice" });

-- Une collection appelé pilote est créée

```
> use airabse
switched to db airabse
> db.pilote.insert({plnum:2, plnom:"Milou" , adr:"Nice" });
WriteResult({ "nInserted" : 1 })
> db.getCollectionNames();
[ "pilote" ]
>
```

7. Gestion des Collections MongoDB

□ Créer une collection dans une base

- **Création explicite d'une collection**

Syntaxe :

```
Db.createCollection("avion");
```

```
C:\> mongo
```

```
>use airbase
```

```
>
```

-- Une collection appelé avion est créée

```
> use airbase
switched to db airbase
> db.pilote.insert({plnum:2, plnom:"Milou" , adr:"Nice" });
WriteResult({ "nInserted" : 1 })
> db.getCollectionNames();
[ "pilote" ]
> db.createCollection("avion");
{ "ok" : 1 }
> db.getCollectionNames();
[ "avion", "pilote" ]
>
```

7. Gestion des Collections MongoDB

□ Visualiser des collections dans une base

- A travers la commande : **show collections**
>show collections
- Ou l'appel de la fonction **db.getCollectionNames**
>db.getCollectionNames

```
>
> show collections
avion
pilote
>
>
> db.getCollectionNames();
[ "avion", "pilote" ]
>
>
>
```

7. Gestion des Collections MongoDB

□ Supprimer une collection dans une base

- La commande de suppression d'une collection est :

db.<collectionName>.drop()

```
C:\> mongo  
>use airbase  
>show collections  
>db.avion.drop();  
>show collections
```

```
>  
> show collections  
avion  
pilote  
> db.avion.drop()  
true  
> show collections  
pilote  
>
```

8. Gestion des documents

MongoDB

□ Plan

- Définition
- Les formats JSON/BJSON
- Approche de modélisation : Imbrication ou partage des objets
- Schéma exemple
- Validation du schéma (contraintes)
- Insertion de documents
- Modification de documents
- Suppression de documents
- Ajout d'un champ dans un document
- Suppression d'un champ dans un document
- Renommage d'un champ dans un document

8. Gestion des documents

MongoDB

□ Définition

- Les documents sont des objets au format Json
- Ils sont comparables aux enregistrements d'une table dans une base de données relationnelle.
- Tout document appartient à une collection et une seule
- Un champ appelé `_id` permet d'identifier de façon unique un document dans une base de données MongoDB
- MongoDB enregistre les documents sur le disque sous un format BSON (JSON binaire).

8. Gestion des documents

MongoDB

□ Les formats JSON/BJSON

- **JSON (JavaScript Object Notation)**
 - Permet de représenter de l'information structurée
 - Format de données textuelle
 - Utilise une notation JavaScript
 - Décrit par la RFC 4627
 - Un document JSON, ne comprend que deux éléments structurels :
 - Des ensembles de paires nom / valeur ;
 - Des listes ordonnées de valeurs.

8. Gestion des documents

MongoDB

□ Les formats JSON/BJSON

- **BSON est la représentation binaire des objets JSON**
 - **Ajoute des améliorations et des capacités supplémentaires**
 - expressions régulières
 - Bytes Arrays
 - Date et TimeStamps
 - Stockage de blocs de code ou de fonctions JavaScript
 - LE FORMAT JSON d'origine ne prend pas en charge les données binaires

8. Gestion des documents

MongoDB

□ Les formats JSON/BJSON

- **Avantage du format JSON**
 - Format abstrait pour une représentation simplifiée dans les différents langages
 - Indépendant du langage de programmation
 - Simple et complet pour la représentation des objets
 - Utilise des types de données connus et simples à décrire
 - Une bonne lisibilité de la syntaxe
 - Temps de traitement très proche de celui des fichiers XML
 - Moins volumineux en terme de stockage

8. Gestion des documents

MongoDB

□ Les formats JSON/BJSON

- **Types disponibles en JSON (voir aussi json.org)**
 - string
 - number
 - Object (un JSON)
 - array
 - true
 - false
 - Null

- **Types supplémentaires BJSON**
 - Bytes Arrays
 - Date
 - TimeStamps

8. Gestion des documents

MongoDB

□ Approche de modélisation : Imbrication ou partage des objets

- Imbrication
 - permet d'emboiter des documents.
Lorsqu'il n'y a pas de risque de redondance, il s'agit d'une approche très efficace qui permet de charger d'un seul document et ses sous-documents

```
{  
  _id: <ObjectId1>,  
  username: "123xyz",  
  contact: {  
    phone: "123-456-7890",  
    email: "xyz@example.com"  
  },  
  access: {  
    level: 5,  
    group: "dev"  
  }  
}
```

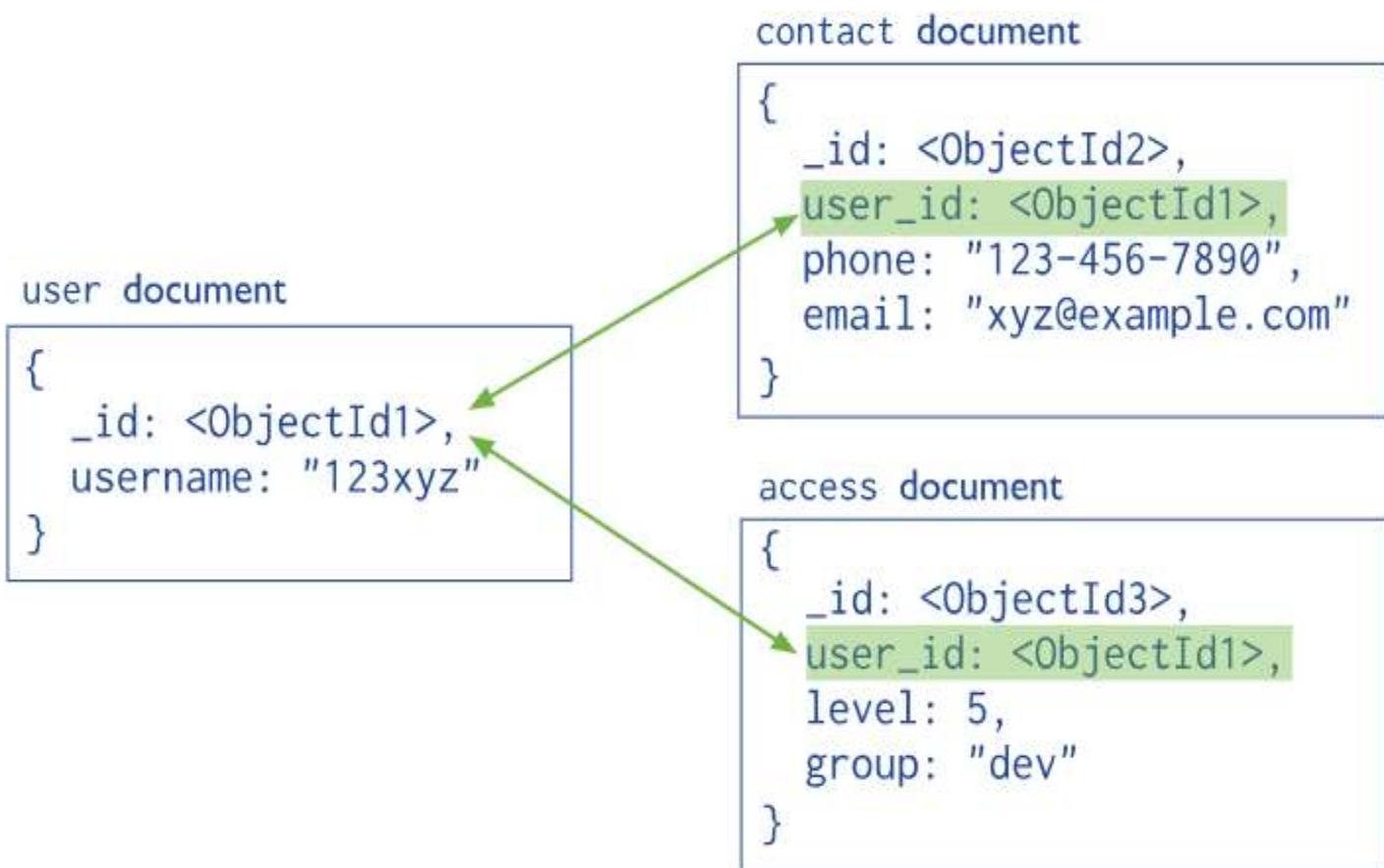
Embedded sub-document

Embedded sub-document

8. Gestion des documents MongoDB

□ Approche de modélisation : Imbrication ou partage des objets

- Partage des objets
 - Très utile s'il y a risque de redondance non souhaité. Plus couteux dans le traitements

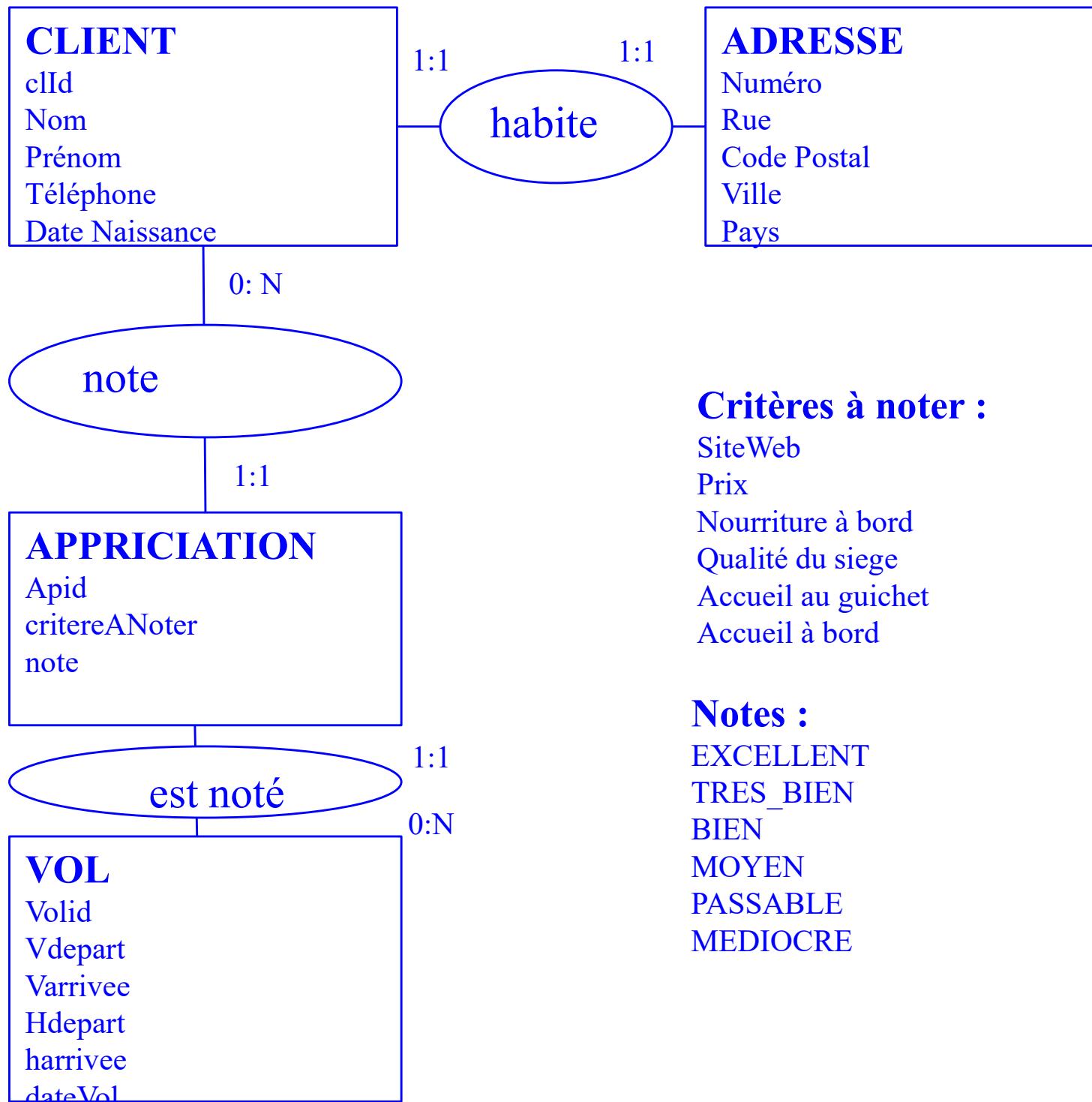


8. Gestion des documents

MongoDB

□ Schéma exemple

- Schéma Merise



8. Gestion des documents

MongoDB

□ Schéma exemple

- Schéma JSON/BJSON
 - Document Client
 - Client 1 : ERZULIE
 - Vue qu'il y a un lien 1:1 entre Adresse et Client, Adresse devient un sous-document

```
{  
    _id: 1,  
    nom: "ERZULIE",  
    prenom:["Maria", "Frida"],  
    telephone:"00509232472",  
    DateNaiss:"01/01/1880",  
    adresse: {  
        numero: 11,  
        rue:"Rue des miracles",  
        codePostal: "HT8110",  
        ville: "PORT-AU-PRINCE",  
        pays: "HAITI"  
    }  
}
```

8. Gestion des documents

MongoDB

□ Schéma exemple

- Schéma JSON/BJSON
 - Document Client
 - Client 2 : BARON
 - Vue qu'il y a un m lien 1:1 entre Adresse et Client, Adresse devient un sous-document

```
{  
    _id: 2,  
    nom: "BARON",  
    prenom:["Samedi"],  
    telephone:"00509232488",  
    DateNaiss:"01/01/1870",  
    adresse: {  
        numero: 1,  
        rue:"Rue des lwa",  
        codePostal: "HT8111",  
        ville: "PORT-AU-PRINCE",  
        pays: "HAITI"  
    }  
}
```

8. Gestion des documents

MongoDB

□ Schéma exemple

- Schéma JSON/BJSON
 - Document VOL (avec `_id` explicite)
 - Y inclure un Array contenant les **appréciations**
 - Les **appréciations** d'un vol sont attachées à lui et lui seul
 - Les **appréciations** sont regroupés par clients
 - A chaque client il y a l'Array de ces **appréciations**

8. Gestion des documents

MongoDB

□ Schéma exemple

- Schéma JSON/BJSON: Document VOL

```
{ _id: "100",
  villeDepart: "Nice",
  villeArrivee: "Paris",
  heureDepart: "10:10",
  heureArrivee: "11:30",
  dateVol: "12/12/2018"
  appreciations:
  [
    {idClient: 1,
      notes: [
        {apid: 11, critereANoter: "SiteWeb", note: "BIEN"},
        {apid: 12, critereANoter: "Prix", note: "BIEN"},
        {apid: 13, critereANoter: "Nourriture à bord", note: "BIEN"},
        {apid: 14, critereANoter: "Qualité siège", note: "MOYEN"},
        {apid: 15, critereANoter: "Accueil guichet", note: "TRES_BIEN"},
        {apid: 16, critereANoter: "Accueil à bord", note: "EXCELLENT"}
      ]
    },
    {idClient: 2,
      notes: [
        {apid: 21, critereANoter: "SiteWeb", note: "TRES_BIEN"},
        {apid: 22, critereANoter: "Prix", note: "MEDIOCRE"},
        {apid: 23, critereANoter: "Nourriture à bord", note: "BIEN"},
        {apid: 24, critereANoter: "Qualité siège", note: "MOYEN"},
        {apid: 25, critereANoter: "Accueil guichet", note: "TRES_BIEN"},
        {apid: 26, critereANoter: "Accueil à bord", note: "BIEN"}
      ]
    }
  }
}
```

8. Gestion des documents

MongoDB

□ Validation du schéma (contraintes)

- MongoDB est normalement un SGBD schema less
- On peut ajouter dans une collection les BJson qu'on veut
- Il est maintenant possible d'introduire des contraintes d'intégrités à faire respecter sur les schéma Bjson d'une collection
 - Donner la liste des champs obligatoires
 - Donner la valeur minimale et maximale
 - Vérifier qu'un champ est d'un type donné
 - Définir un énuméré (liste des valeurs possibles pour un champ)
 - Définir des expressions régulières
 - Interdire l'ajout de nouveaux champs dans un Record

8. Gestion des documents

MongoDB

□ Validation du schéma (contraintes)

- Exemple

```
db.createCollection("students", [
  validator: [
    $jsonSchema: {
      bsonType: "object",
      required: [ "name", "year", "major", "gpa", "address.city", "address.street" ]
      properties: {
        name: [
          bsonType: "string",
          description: "must be a string and is required"
        ],
        gender: [
          bsonType: "string",
          description: "must be a string and is not required"
        ],
        year: [
          bsonType: "int",
          minimum: 2017,
          maximum: 2017,
          exclusiveMaximum: false,
          description: "must be an integer in [ 2017, 2017 ] and is required"
        ],
        major: [
          enum: [ "Math", "English", "Computer Science", "History", null ],
          description: "can only be one of the enum values and is required"
        ],
        gpa: [
          bsonType: [ "double" ],
          minimum: 0,
          description: "must be a double and is required"
        ],
        "address.city" : [
          bsonType: "string",
          description: "must be a string and is required"
        ],
        "address.street" : [
          bsonType: "string",
          description: "must be a string and is required"
        ]
      }
    }
  ]
})
```

8. Gestion des documents

MongoDB

□ Validation du schéma (contraintes)

- Exemple, champs obligatoires, valeurs obligatoires

```
db.myCollection4.drop();
db.createCollection("myCollection4", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["a", "b", "c", "d", "e"],
      properties: {
        a: {
          bsonType: "number"
        },
        b: {
          bsonType: "string"
        },
        c: {
          enum: ["abc", "def", "ghi"],
          description:"Liste des valeurs autorisées"
        },
        d: {
          bsonType: "date",
          description: "La date doit être au format ISO : ISODate('1990-03-02T00:00:00Z')"
        },
        e: {
          bsonType: "number",
          minimum:1,
          maximum:1000,
          description: "La date doit être au format ISO : ISODate('1990-03-02T00:00:00Z')"
        }
      }
    }
  }
});
```

8. Gestion des documents

MongoDB

□ Validation du schéma (contraintes)

- Exemple, interdiction d'ajout de champs

```
-- interdiction d'ajout de champs
db.myCollection5.drop();
db.createCollection("myCollection5", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      additionalProperties: false,
      required: ["a", "b", "c", "d", "e"],
      properties: {
        a: {bsonType: "number"},
        b: {bsonType: "string"},
        c: {
          enum: ["abc", "def", "ghi"],
          description:"Liste des valeurs autorisées"
        },
        d: {
          bsonType: "date",
          description: "La date doit être au format ISO : ISODate('1990-03-02T00:00:00Z')"
        },
        e: {
          bsonType: "number",
          minimum:1,
          maximum:1000,
          description: "La date doit être au format ISO : ISODate('1990-03-02T00:00:00Z')"
        }
      }
    }
  }
});
-- Erreur : "errmsg" : "Document failed validation"
db.myCollection5.insert(
  {_id:1, a:1, b:"b", c:"abc", d:ISODate('1990-03-02T00:00:00Z'), e:1 }
);
```

8. Gestion des documents

MongoDB

□ Validation du schéma (contraintes)

- Exemple, interdiction d'ajout de champs

```
-- interdiction d'ajout de champs, _id doit être dans la liste des champs obligatoires
db.myCollection6.drop();
db.createCollection("myCollection6", {
    validator: {
        $jsonSchema: {
            bsonType: "object",
            additionalProperties: false,
            required: ["_id", "a", "b", "c", "d", "e"],
            properties: {
                _id: {bsonType: "number"},
                a: {bsonType: "number"},
                b: {bsonType: "string"},
                c: {enum: ["abc", "def", "ghi"],
                    description:"Liste des valeurs autorisées"
                },
                d: {bsonType: "date",
                    description: "La date doit être au format ISO : ISODate('1990-03-02T00:00:00Z')"
                },
                e: {
                    bsonType: "number",
                    minimum:1,
                    maximum:1000,
                    description: "La date doit être au format ISO : ISODate('1990-03-02T00:00:00Z')"
                } }}});}

-- Erreur : "errmsg" : "Document failed validation"
db.myCollection6.insert(
{ _id:1, a:1, b:"b", c:"abc", d:ISODate('1990-03-02T00:00:00Z'), e:1, f:1 })
);
```

8. Gestion des documents

MongoDB

□ Insertion de documents

- Trois méthodes permettent d'effectuer des insertions de documents
 - `db.collection.insertOne()`: Insère un document dans la collection
 - `db.collection.insertMany()`: Insère plusieurs documents dans la collection
 - `db.collection.insert()`: Insère indifféremment un ou plusieurs documents dans la collection

8. Gestion des documents

MongoDB

□ Insertion de documents

- D'autres méthodes peuvent servir à l'insertion de documents. Voir la document pour leur usage
 - `db.collection.update()` when used with the `upsert: true` option.
 - `db.collection.updateOne()` when used with the `upsert: true` option.
 - `db.collection.updateMany()` when used with the `upsert: true` option.
 - `db.collection.findAndModify()` when used with the `upsert: true` option.
 - `db.collection.findOneAndUpdate()` when used with the `upsert: true` option.
 - `db.collection.findOneAndReplace()` when used with the `upsert: true` option.
 - `db.collection.save()`.
 - `db.collection.bulkWrite()`.

8. Gestion des documents

MongoDB

□ Insertion de documents

• Insertion des clients du client 1

c:\>mongo

➤ use airbase

➤ db.createCollection("clients");

➤ db.createCollection("vols");

➤ document={

 _id: 1,

 nom: "ERZULIE",

 prenoms:["Maria", "Freda"],

 telephone:"00509232485",

 DateNaiss:"01/01/1881",

 adresse: {

 numero: 11,

 rue:"Rue des miracles",

 codePostal: "HT8110",

 ville: "PORT-AU-PRINCE",

 pays: "HAITI"

 }

}

>db.clients.insert(document);

WriteResult({ "nInserted" : 1 })

La collection clients est aussi créés par la même occasion.

8. Gestion des documents

MongoDB

□ Insertion de documents

- **Insertion des clients du client 2**

```
c:\>mongo
```

```
> use airbase
```

```
> db.clients.insertOne(
```

```
{
```

```
    _id: 2,
```

```
    nom: "BARON",
```

```
    prenoms:["Samedi"],
```

```
    telephone:"00509232488",
```

```
    DateNaiss:"01/01/1870",
```

```
    adresse: {
```

```
        numero: 1,
```

```
        rue:"Rue des lwa",
```

```
        codePostal: "HT8111",
```

```
        ville: "PORT-AU-PRINCE",
```

```
        pays: "HAITI"
```

```
}
```

```
});
```

```
{ "acknowledged" : true, "insertedId" : 2 }
```

8. Gestion des documents

MongoDB

□ Insertion de documents

• Insertion d'un VOL

```
c:\>mongo
> use airbase
> document =
{
    _id: "100",
    villeDepart: "Nice",
    villeArrivee: "Paris",
    heureDepart: "10:10",
    heureArrivee: "11:30",
    dateVol: "12/12/2018",
    appreciations:[
        {idClient:1,
         notes:[
             {apid: 11, critereANoter: "SiteWeb", note: "BIEN"},
             {apid: 12, critereANoter: "Prix", note: "BIEN"},
             {apid: 13, critereANoter: "Nourriture à bord", note: "BIEN"},
             {apid: 14, critereANoter: "Qualité siège", note: "MOYEN"},
             {apid: 15, critereANoter: "Accueil guichet", note: "TRES_BIEN"},
             {apid: 16, critereANoter: "Accueil à bord", note: "EXCELLENT"}]
        },
        ...
    ... Voir la suite page suivante
```

8. Gestion des documents

MongoDB

□ Insertion de documents

- Insertion d'un vol suite

```
{idClient:2,  
  notes:[  
    {apid: 21, critereANoter: "SiteWeb", note:  
      "TRES_BIEN"},  
    {apid: 22, critereANoter: "Prix", note: "MEDIocre"},  
    {apid: 23, critereANoter: "Nourriture à bord", note:  
      "BIEN"},  
    {apid: 24, critereANoter: "Qualité siège", note:  
      "MOYEN"},  
    {apid: 25, critereANoter: "Accueil guichet", note:  
      "TRES_BIEN"},  
    {apid: 26, critereANoter: "Accueil à bord", note: "BIEN" }  
  ]  
}  
]
```

```
>db.vols.insertOne(document);  
{ "acknowledged" : true, "insertedId" : 100 }
```

8. Gestion des documents

MongoDB

□ Modification de documents

- Les méthodes suivantes permettent d'effectuer des mises à jour
 - `db.collection.updateOne(<filter>, <update>, <options>)`
 - `db.collection.updateMany(<filter>, <update>, <options>)`
 - `db.collection.replaceOne(<filter>, <update>, <options>)`
- Filter : Expression de recherche
- Update : Modification à faire
- Options : Divers options (voir la documentation)
 - {
 - `upsert: <boolean>`,
 - `writeConcern: <document>`,
 - `collation: <document>`,
 - `arrayFilters: [<filterdocument1>, ...]`

8. Gestion des documents

MongoDB

□ Modification de documents

- Exemple de modification de documents

```
> db.clients.updateOne({ nom: "ERZULIE"
}, { $set: {DateNaiss: "12/12/1900" } }, {});
```

```
{ "acknowledged" : true, "matchedCount" :
1, "modifiedCount" : 1 }
```

- Attention il faut respecter la casse sur le nom des champs et des valeurs

8. Gestion des documents

MongoDB

□ Suppression de documents

- Les méthodes suivantes permettent d'effectuer des suppressions
 - db.collection.deleteMany()
 - db.collection.deleteOne()

```
db.collection.deleteOne(  
    <filter>,  
    {  
        writeConcern: <document>,  
        collation: <document>  
    }  
)
```

8. Gestion des documents

MongoDB

□ Suppression de documents

- Exemple de suppression d'un document

- Supprimer le client ERZULIE

➤ db.clients.deleteOne({nom:"ERZULIE"},{});
➤ Vérification

```
> db.clients.deleteOne({nom:"ERZULIE"},{});  
{ "acknowledged" : true, "deletedCount" : 1 }  
> db.clients.find();  
{ "_id" : 2, "nom" : "BARON", "prenoms" : [ "Samedi" ], "telephone" : "00509232488", "DateNaiss" : "01/01/1870", "adresse"  
: { "numero" : 1, "rue" : "Rue des Iwa", "codePostal" : "HT8111", "ville" : "PORT-AU-PRINCE", "pays" : "HAITI" } }
```

8. Gestion des documents

MongoDB

□ Ajout d'un champ dans un document

- L'opérateur **\$set** permet d'ajouter un champ dans un ou plusieurs documents. Si le champ existe une mise à jour est effectuée
- { \$set: { <field1>: <value1>, ... } } à combiner avec update
- Exemple
 - Ajouter le champ `voyageurAssidu` dans chaque document de la collection `clients`
`> db.clients.update({}, { $set: { voyageurAssidu: "OUI" } });`
Cette action n'ajoute le champ que dans le premier document
 - Pour ajouter le champ dans tous les documents d'une collection. Deux possibilités
 - `> db.clients.updateMany({}, { $set: { voyageurAssidu: "OUI" } });`
 - `> db.clients.update({}, { $set: { voyageurAssidu: "OUI" } }, {multi:true});`

8. Gestion des documents

MongoDB

□ Suppression d'un champ dans un document

- L'opérateur **\$unset** permet de supprimer des champs dans un ou plusieurs documents.
 { \$set: { <field1>: <value1>, ... } } à combiner avec update
- Exemple
 - Supprimer le champ `voyageurAssidu` dans chaque document de la collection `clients`
`> db.clients.update({}, { $unset: { voyageurAssidu: 1 } });`
**Cette action ne modifie que le premier document
 - Pour supprimer le champ dans l'ensemble des documents. Deux possibilités :
`> db.clients.updateMany({}, { $unset: { voyageurAssidu: 1 } });`
`> db.clients.update({}, { $unset: { voyageurAssidu: 1 } }, {multi:true});`

8. Gestion des documents

MongoDB

□ Renommage d'un champ dans un document

- L'opérateur **\$rename** permet de renommer des champs dans un ou plusieurs documents.
 { \$rename: { <field1>: <newName1>, ... } } à combiner avec update
- Exemple
 - Renomme le champ DateNaiss en dateNaissance dans un document
`> db.clients.update({_id:1}, { $rename: {"DateNaiss": "dateNaissance"} });`
 - Renomme le champ DateNaiss en dateNaissance dans tous les documents
`> db.clients.updateMany({}, { $rename: {"DateNaiss": "dateNaissance"} });`
`> db.clients.update({}, { $rename : {{"dateNaiss": "dateNaissance"} }, {multi:true}});`

9. Indexation de documents

MongoDB

□ Plan

- Création d'index
- Affichage d'un plan d'exécution d'une requête
- Affichage des indexes créés sur une collection
- Suppression d'un index

9. Indexation de documents

MongoDB

□ Crédit d'index

- Les méthodes suivantes permettent de poser des index
 - `db.collection.createIndex()`
 - `db.collection.createIndex(<key and index type specification>, <options>)`
- Il existe différents types d'index
 - Des index mono-colonnes
 - Des index multicolonnes
 - Des index ascendants
 - Des index descendants
 - Des index uniques ou non unique
- Il faut avoir une bonne stratégie d'indexation. Elle doit dépendre des requêtes à lancer

9. Indexation de documents

MongoDB

□ Création d'index

- <key and index type specification>
 - Document contenant la liste des champs à indexer {champ1:1, champ2:1}
- 1: Index ascendant; -1: Index descendant**
- <options>
 - Ce document est optionnel. Il permet de spécifier entre autre le nom de l'index. Voici les champs possibles de ce document:
 - **background** : true ou false (si true construit l'index sans bloquer les activités)
 - **name** : nom index (nom explicite)
 - **unique** : true ou false (true si index unique)
 - Si aucun nom d'index est donné explicitement, mongoDB donne un nom implicite

9. Indexation de documents

MongoDB

□ Création d'index

- **Création d'un index sur la colonne NOM de la collection CLIENTS (nom index implicite)**

```
>db.clients.createIndex({nom:1});
```

```
> db.clients.createIndex({nom:1});
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 2,
    "numIndexesAfter" : 3,
    "ok" : 1
}
```

9. Indexation de documents

MongoDB

□ Création d'index

- **Création d'un index sur la colonne DateNaiss de la collection CLIENTS (nom index explicite)**

```
>db.clients.createIndex({DateNaiss :1},  
{background:false, unique:false,  
name:"idx_clients_DateNaiss" });
```

```
> use airbase  
switched to db airbase  
> show collections  
clients  
pilote  
vols  
> db.clients.createIndex({DateNaiss :1}, {background:false, unique  
:false, name:"idx_clients_DateNaiss" });  
{  
    "createdCollectionAutomatically" : false,  
    "numIndexesBefore" : 1,  
    "numIndexesAfter" : 2,  
    "ok" : 1  
}  
>
```

9. Indexation de documents

MongoDB

□ Affichage d'un plan d'exécution d'une requête

```
> db.clients.find({nom: "ERZULIE"}).explain();
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "airabse.clients",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "nom" : {
        "$eq" : "ERZULIE"
      }
    },
    "winningPlan" : {
      "stage" : "FETCH",
      "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
          "nom" : 1
        },
        "indexName" : "nom_1",
        "isMultiKey" : false,
        "multiKeyPaths" : {
          "nom" : [ ]
        }
      }
    }
  }
}
```

9. Indexation de documents

MongoDB

□ Affichage d'un plan d'exécution d'une requête (suite)

```
        },
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 2,
        "direction" : "forward",
        "indexBounds" : {
            "nom" : [
                "[\"ERZULIE\", \"ERZULIE\"]"
            ]
        }
    },
    "rejectedPlans" : [ ]
},
"serverInfo" : {
    "host" : "Gabriel",
    "port" : 27017,
    "version" : "4.0.6",
    "gitVersion" : "caa42a1f75a56c7643d0b68d3880444375ec42e3"
},
"ok" : 1
}
>
```

9. Indexation de documents

MongoDB

□ Affichage d'un plan d'exécution d'une requête (suite) Nom index explicite

```
> db.clients.find({DateNaiss:"01/01/1870"}).explain();
```

```
sion, got ':' @(shell):1:27
> db.clients.find({DateNaiss:"01/01/1870"}).explain();
{
  "queryPlanner" : {
    "plannerVersion" : 1,
    "namespace" : "airbase.clients",
    "indexFilterSet" : false,
    "parsedQuery" : {
      "DateNaiss" : {
        "$eq" : "01/01/1870"
      }
    },
    "winningPlan" : {
      "stage" : "FETCH",
      "inputStage" : {
        "stage" : "IXSCAN",
        "keyPattern" : {
          "DateNaiss" : 1
        },
        "indexName" : "idx_clients_DateNaiss",
        "isMultiKey" : false,
        "multiKeyPaths" : {
          "DateNaiss" : [ ]
        },
      }
    }
  }
}
```

9. Indexation de documents

MongoDB

□ Affichage d'un plan d'exécution d'une requête (suite) : Nom index explicite

```
> db.clients.find({DateNaiss:"01/01/1870"}).explain();
```

```
        "isUnique" : false,
        "isSparse" : false,
        "isPartial" : false,
        "indexVersion" : 2,
        "direction" : "forward",
        "indexBounds" : {
            "DateNaiss" : [
                "[\"01/01/1870\", \"01/01/1870\"]"
            ]
        }
    },
    "rejectedPlans" : [ ]
},
"serverInfo" : {
    "host" : "Gabriel",
    "port" : 27017,
    "version" : "4.0.6",
    "gitVersion" : "caa42a1f75a56c7643d0b68d3880444375ec42e3"
},
"ok" : 1
}
```

9. Indexation de documents

MongoDB

□ Affichage des indexes créés sur une collection

- La méthode db.collection.getIndexes() permet d'afficher tous les indexes créés sur une collection

– Exemple

```
> use airbase  
>db.clients.getIndexes();
```

```
> db.clients.getIndexes();  
[  
  {  
    "v" : 2,  
    "key" : {  
      "_id" : 1  
    },  
    "name" : "_id_",  
    "ns" : "airbase.clients"  
  },  
  {  
    "v" : 2,  
    "key" : {  
      "DateNaiss" : 1  
    },  
    "name" : "idx_clients_DateNaiss",  
    "ns" : "airbase.clients",  
    "background" : false  
  },  
  {  
    "v" : 2,  
    "key" : {  
      "nom" : 1  
    },  
    "name" : "nom_1",  
    "ns" : "airbase.clients"  
  }]
```

9. Indexation de documents

MongoDB

□ Suppression d'un index

- Pour supprimer un index il faut appeler la méthode db.collection.dropIndex(index)
- Exemple :
 - Suppression de l'index nom_1
> db.clients.dropIndex("nom_1");

```
> db.clients.dropIndex("nom_1");
{ "nIndexesWas" : 3, "ok" : 1 }
> db.clients.getIndexes();
[
  {
    "v" : 2,
    "key" : {
      "_id" : 1
    },
    "name" : "_id_",
    "ns" : "airbase.clients"
  },
  {
    "v" : 2,
    "key" : {
      "DateNaiss" : 1
    },
    "name" : "idx_clients_DateNaiss",
    "ns" : "airbase.clients",
    "background" : false
  }
]
```

10. Recherche de documents

MongoDB

□ Plan

- Les méthodes pour effectuer les recherches
- Recherche de tous les documents d'une collection
- Recherche de documents connaissant la valeur d'une propriété
- Recherche de documents connaissant la valeur plusieurs propriétés
- Recherche de documents avec l'opérateur `>`, `>=`, `<`, ...
- Recherches de documents avec le connecteur logique OR
- Recherche de documents via une colonne indexée
- Jointure côté serveur

10. Recherche de documents

MongoDB

□ Les méthodes pour effectuer les recherches

- Les méthodes suivantes permettent de rechercher des documents :
 - `db.collection.find()` : renvoie tous les documents d'une collections
 - `db.collection.find(query, projection)` : renvoie les documents qui matchent avec “query” et affiche les colonnes indiquées dans “projection”
- Query contient des tests et évaluation:
 - Opérateur de comparaison : `$eq`, `$gt`, `$in`, `$lt`, `$lte`, `$ne`, `$nin`
 - Opérateur logique : `$and`, `$not`, `$nor`, `$or`
 - Vérification d'éléments : `$exists`, `$type`
 - Evaluation d'expressions: `$expr`, `$jsonSchema`, `$mo`, `$regex`, `$text`, `$where`
- Projection contient des expressions :
 - { `field1: <value>, field2: <value> ... }`
 - Si `value=1` champ retourné, si `value =0` non

10. Recherche de documents

MongoDB

□ Recherche de tous les documents d'une collection

- Rechercher tous les documents de la collection CLIENTS
 - db.clients.find();

```
> db.clients.find();
{ "_id" : 1, "nom" : "ERZULIE", "prenoms" : [ "Maria", "Frida" ], "telephone" : "00509232472", "DateNaiss" : "01/01/1880", "adresse" : { "numero" : 11, "rue" : "Rue des miracles", "codePostal" : "HT8110", "ville" : "PORT-AU-PRINCE", "pays" : "HAITI" } }
{ "_id" : 2, "nom" : "BARON", "prenoms" : [ "Samedi" ], "telephone" : "00509232488", "DateNaiss" : "01/01/1870", "adresse" : { "numero" : 1, "rue" : "Rue des lwa", "codePostal" : "HT8111", "ville" : "PORT-AU-PRINCE", "pays" : "HAITI" } }
>
```

10. Recherche de documents MongoDB

□ Recherche de tous les documents d'une collection

- -- Pour un meilleurs affichage
- Db.clients.find().pretty();

```
> db.clients.find().pretty();
{
    "_id" : 1,
    "nom" : "ERZULIE",
    "prenoms" : [
        "Maria",
        "Frida"
    ],
    "telephone" : "00509232472",
    "DateNaiss" : "01/01/1880",
    "adresse" : {
        "numero" : 11,
        "rue" : "Rue des miracles",
        "codePostal" : "HT8110",
        "ville" : "PORT-AU-PRINCE",
        "pays" : "HAITI"
    }
}
{
    "_id" : 2,
    "nom" : "BARON",
    "prenoms" : [
        "Samedi"
    ],
    "telephone" : "00509232488",
    "DateNaiss" : "01/01/1870",
    "adresse" : {
        "numero" : 1,
        "rue" : "Rue des lwa",
        "codePostal" : "HT8111",
        "ville" : "PORT-AU-PRINCE",
        "pays" : "HAITI"
    }
}
```

10. Recherche de documents MongoDB

□ Recherche de documents connaissant la valeur d'une propriété

- Recherche de clients de nom ERZULIE

```
> db.clients.find({nom:"ERZULIE"}).pretty();
```

```
> db.clients.find({nom:"ERZULIE"}).pretty();
{
    "_id" : 1,
    "nom" : "ERZULIE",
    "prenoms" : [
        "Maria",
        "Frida"
    ],
    "telephone" : "00509232472",
    "DateNaiss" : "01/01/1880",
    "adresse" : {
        "numero" : 11,
        "rue" : "Rue des miracles",
        "codePostal" : "HT8110",
        "ville" : "PORT-AU-PRINCE",
        "pays" : "HAITI"
    }
}
>
```

10. Recherche de documents MongoDB

□ Recherche de documents connaissant la valeur de plusieurs propriétés

- Recherche de clients de nom BARON et ayant le telephone suivant : :"00509232488"

```
> db.clients.find({nom:"BARON",
    telephone:"00509232488"}).pretty();
```

```
> db.clients.find({nom:"BARON", telephone:"00509232488"}).pretty();
{
    "_id" : 2,
    "nom" : "BARON",
    "prenoms" : [
        "Samedi"
    ],
    "telephone" : "00509232488",
    "DateNaiss" : "01/01/1870",
    "adresse" : {
        "numero" : 1,
        "rue" : "Rue des lwa",
        "codePostal" : "HT8111",
        "ville" : "PORT-AU-PRINCE",
        "pays" : "HAITI"
    }
}
>
```

10. Recherche de documents MongoDB

□ Recherche de documents avec l'opérateur >, >=, <, ...

- Recherche de clients nés après 1870

```
>db.clients.find({DateNaiss:{'$gt':'01/01/1871'}}).pretty()  
>
```

```
> db.clients.find({DateNaiss:{'$gt':'01/01/1871'}}).pretty();  
{  
    "_id" : 1,  
    "nom" : "ERZULIE",  
    "prenoms" : [  
        "Maria",  
        "Frida"  
    ],  
    "telephone" : "00509232472",  
    "DateNaiss" : "01/01/1880",  
    "adresse" : {  
        "numero" : 11,  
        "rue" : "Rue des miracles",  
        "codePostal" : "HT8110",  
        "ville" : "PORT-AU-PRINCE",  
        "pays" : "HAITI"  
    }  
}
```

10. Recherche de documents MongoDB

□ Recherche de documents avec le connecteur logique OR

- Recherche des clients nées après 1870 ou qui habitent au numéro 11.

```
> db.clients.find({"$or":
```

```
[{"DateNaiss":{"$gt":"01/01/1871"}}, {"adresse.numero":11}]).pretty();
```

```
> db.clients.find({"$or": [{"DateNaiss":{"$gt":"01/01/1871"}}, {"adresse.numero":11}]}).pretty();
{
  "_id" : 1,
  "nom" : "ERZULIE",
  "prenoms" : [
    "Maria",
    "Frida"
  ],
  "telephone" : "00509232472",
  "DateNaiss" : "01/01/1880",
  "adresse" : {
    "numero" : 11,
    "rue" : "Rue des miracles",
    "codePostal" : "HT8110",
    "ville" : "PORT-AU-PRINCE",
    "pays" : "HAITI"
  }
}
```

10. Recherche de documents

MongoDB

□ Recherche de documents avec projection

- Recherche tous les clients et afficher uniquement le champs nom

```
>db.clients.find({}, {nom:1}).pretty();
```

```
> db.clients.find({}, {nom:1}).pretty();
{ "_id" : 1, "nom" : "ERZULIE" }
{ "_id" : 2, "nom" : "BARON" }
>
```

10. Recherche de documents

MongoDB

□ Recherche de documents avec projection

- Recherche tous les clients de nom ERZULIE et afficher tous les champs **sauf dateNaiss et telephone**

> db.clients.find({nom:"ERZULIE"}, {DateNaiss:0, telephone:0}).pretty();

```
> db.clients.find({nom:"ERZULIE"}, {DateNaiss:0, telephone:0}).pretty();
{
    "_id" : 1,
    "nom" : "ERZULIE",
    "prenoms" : [
        "Maria",
        "Frida"
    ],
    "adresse" : {
        "numero" : 11,
        "rue" : "Rue des miracles",
        "codePostal" : "HT8110",
        "ville" : "PORT-AU-PRINCE",
        "pays" : "HAITI"
    }
}
```

10. Recherche de documents MongoDB

□ Recherche de documents via une colonne indexée

- Recherche tous les clients via l'index posé sur le champ NOM

```
> db.clients.find({nom:"ERZULIE"}, {DateNaiss:0, telephone:0}).pretty();
```

```
> db.clients.find({nom:"ERZULIE"}, {DateNaiss:0, telephone:0}).pretty();
{
  "_id" : 1,
  "nom" : "ERZULIE",
  "prenoms" : [
    "Maria",
    "Frida"
  ],
  "adresse" : {
    "numero" : 11,
    "rue" : "Rue des miracles",
    "codePostal" : "HT8110",
    "ville" : "PORT-AU-PRINCE",
    "pays" : "HAITI"
  }
}
```

11. Les agrégats dans MongoDB

□ Plan

- Les différentes étapes d'agrégation
- Pipe des étapes d'agrégation
- Etape d'agrégation \$addFields
- Etape d'agrégation \$bucket
- Etape d'agrégation \$bucketAuto
- Etape d'agrégation \$collStats
- Etape d'agrégation \$count
- Etape d'agrégation \$facet
- Etape d'agrégation \$geoNear
- Etape d'agrégation \$graphLookup
- Etape d'agrégation \$group (Group By)
- Etape d'agrégation \$indexStats
- Etape d'agrégation \$limit
- Etape d'agrégation \$listSessions
- Etape d'agrégation \$lookup (jointure)
- Etape d'agrégation \$match
- Etape d'agrégation \$out
- Etape d'agrégation \$project
- Etape d'agrégation \$redact
- Etape d'agrégation \$replaceRoot
- Etape d'agrégation \$sample
- Etape d'agrégation \$skip
- Etape d'agrégation \$sort
- Etape d'agrégation \$sortByCount
- Etape d'agrégation \$unwind (unnest)

11. Les agrégats dans MongoDB

□ Plan

- Les différentes étapes d'agrégation
- Pipe des étapes d'agrégation
- Etape d'agrégation \$addFields
- Etape d'agrégation \$bucket
- Etape d'agrégation \$bucketAuto
- Etape d'agrégation \$collStats
- Etape d'agrégation \$count
- Etape d'agrégation \$facet
- Etape d'agrégation \$geoNear
- Etape d'agrégation \$graphLookup
- Etape d'agrégation \$group (Group By)
- Etape d'agrégation \$indexStats
- Etape d'agrégation \$limit
- Etape d'agrégation \$listSessions
- Etape d'agrégation \$lookup (jointure)
- Etape d'agrégation \$match
- Etape d'agrégation \$out
- Etape d'agrégation \$project
- Etape d'agrégation \$redact
- Etape d'agrégation \$replaceRoot
- Etape d'agrégation \$sample
- Etape d'agrégation \$skip
- Etape d'agrégation \$sort
- Etape d'agrégation \$sortByCount
- Etape d'agrégation \$unwind (unnest)

11. Les agrégats dans MongoDB

□ Les différentes étapes d'agrégation

- MongoDB propose **plusieurs étapes d'agrégations** :
\$addFields, \$bucket, \$bucketAuto, \$collStats, \$count,
\$facet, \$geoNear, \$graphLookup, **\$group**,
\$indexStats, \$limit, \$listSessions, **\$lookup**, \$match,
\$out, **\$project**, \$redact, \$replaceRoot, \$sample, \$skip,
\$sort, \$sortByCount, **\$unwind**
- Ces étapes sont des paramètres de la fonction
db.collection.aggregate([{etape 1}, ..., {etape N}])
- Une étape peut être appelée zéro, une ou plusieurs fois
- Ces étapes reçoivent en entrée des documents et
renvoient après leur action des documents
- Chaque étape sera défini par la suite.
- Nous nous limitions à certaines d'entres elles

11. Les agrégats dans MongoDB

□ Pipeline des étapes d'agrégation

- La fonction db.collection.**aggregate**([{ <étape> }, ...]) permet d'appliquer les opérateurs sur une ou plusieurs collections
- Lors de l'appel de db.collection.**aggregate()**, un **tableau des étapes** est passé en argument
 - db.collection.**aggregate**([{ <étape1> }, ..., { <étapeN> }])
- Le tableau contient une ou plusieurs étapes
- Les étapes sont appliquées dans leur ordre d'apparition dans le tableau.
- Une étape reçoit en entrée des documents et fournit en sortie des documents à l'étape suivante. ON APPELLE CELA AUSSI PIPELINE DES ETAPES
- Les pipeline des étapes ressemblent au PIPE Unix/Linux

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$addFields

- Ajoute de nouveaux champs aux documents.
\$addFields produit des documents qui contiennent tous les champs existants des documents de saisie et les champs nouvellement ajoutés.
- Syntaxe
`{ $addFields: { <newField>: <expression>, ... } }`

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$addFields

- Exemple

```
➤ Use airbase  
➤ db.createCollection("etudiant");  
➤ db.etudiant.insert({  
    _id: 1,  
    student: "Maya",  
    homework: [ 10, 5, 10 ],  
    quiz: [ 10, 8 ],  
    extraCredit: 0  
});
```

```
> db.etudiant.insert({  
    _id: 2,  
    student: "Ryan",  
    homework: [ 5, 6, 5 ],  
    quiz: [ 8, 8 ],  
    extraCredit: 8  
});
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$addFields

- Exemple

```
> db.etudiant.aggregate( [  
    {  
        $addFields: {  
            totalHomework: { $sum: "$homework" } ,  
            totalQuiz: { $sum: "$quiz" }  
        }  
    },  
    {  
        $addFields: { totalScore:  
            { $add: [ "$totalHomework", "$totalQuiz",  
                    "$extraCredit" ] } }  
    }  
)
```

```
{ "_id" : 1, "student" : "Maya", "homework" : [ 10, 5, 10 ], "quiz" : [ 10, 8 ], "extraCredit" : 0, "totalHomework" : 25, "totalQuiz" : 18, "totalScore" : 43 }  
{ "_id" : 2, "student" : "Ryan", "homework" : [ 5, 6, 5 ], "quiz" : [ 8, 8 ], "extraCredit" : 8, "totalHomework" : 16, "totalQuiz" : 16, "totalScore" : 40 }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucket

- Classement des documents entrants en groupes, appelés « buckets », sur la base d'une expression et de limites de buckets spécifiées, et production d'un document pour chaque bucket. Chaque document de sortie contient un champ `_id` dont la valeur spécifie la limite inférieure inclusive du groupe. L'option de sortie spécifie les champs inclus dans chaque document de sortie.
- \$bucket ne produit que les documents de sortie pour les compartiments qui contiennent au moins un document d'entrée.

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucket

- Syntaxe

```
{
```

```
$bucket: {
```

```
    groupBy: <expression>,
```

```
    boundaries: [ <lowerbound1>, <lowerbound2>, ... ],
```

```
    default: <literal>,
```

```
    output: {
```

```
        <output1>: { <$accumulator expression> },
```

```
        ...
```

```
        <outputN>: { <$accumulator expression> }
```

```
}
```

```
}
```

```
}
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucket

- Exemple

```
db.artists.insertMany([
  { "_id" : 1, "last_name" : "Bernard", "first_name" : "Emil",
    "year_born" : 1868, "year_died" : 1941, "nationality" : "France" },
  { "_id" : 2, "last_name" : "Rippl-Ronai", "first_name" : "Joszef",
    "year_born" : 1861, "year_died" : 1927, "nationality" : "Hungary" },
  { "_id" : 3, "last_name" : "Ostroumova", "first_name" : "Anna",
    "year_born" : 1871, "year_died" : 1955, "nationality" : "Russia" },
  { "_id" : 4, "last_name" : "Van Gogh", "first_name" : "Vincent",
    "year_born" : 1853, "year_died" : 1890, "nationality" : "Holland" },
  { "_id" : 5, "last_name" : "Maurer", "first_name" : "Alfred",
    "year_born" : 1868, "year_died" : 1932, "nationality" : "USA" },
  { "_id" : 6, "last_name" : "Munch", "first_name" : "Edvard",
    "year_born" : 1863, "year_died" : 1944, "nationality" : "Norway" },
  { "_id" : 7, "last_name" : "Redon", "first_name" : "Odilon",
    "year_born" : 1840, "year_died" : 1916, "nationality" : "France" },
  { "_id" : 8, "last_name" : "Diriks", "first_name" : "Edvard",
    "year_born" : 1855, "year_died" : 1930, "nationality" : "Norway" }
])
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucket

- Exemple

```
db.artists.aggregate( [  
    // First Stage  
    {  
        $bucket: {  
            groupBy: "$year_born",           // Field to group by  
            boundaries: [ 1840, 1850, 1860, 1870, 1880 ], // Boundaries for  
            the buckets  
            default: "Other",             // Bucket id for documents  
            which do not fall into a bucket  
            output: {                   // Output for each bucket  
                "count": { $sum: 1 },  
                "artists" :  
                    {  
                        $push: {  
                            "name": { $concat: [ "$first_name", " ", "$last_name"] },  
                            "year_born": "$year_born"  
                        }  
                    }  
            }  
        },  
        // Second Stage  
        {  
            $match: { count: {$gt: 0} }  
        }  
    }  
)
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucket

- Exemple

```
{ "_id" : 1840, "count" : 1, "artists" : [ { "name" : "Odilon Redon", "year_born" : 1840 } ] }
{ "_id" : 1850, "count" : 2, "artists" : [ { "name" : "Vincent Van Gogh", "year_born" : 1853 }, { "name" : "Edvard Diriks", "year_born" : 1855 } ] }
{ "_id" : 1860, "count" : 4, "artists" : [ { "name" : "Emil Bernard", "year_born" : 1868 }, { "name" : "Joszef Rippl-Ronai", "year_born" : 1861 }, { "name" : "Alfred Maurer", "year_born" : 1868 }, { "name" : "Edvard Munch", "year_born" : 1863 } ] }
{ "_id" : 1870, "count" : 1, "artists" : [ { "name" : "Anna Ostroumova", "year_born" : 1871 } ] }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucketAuto

- Catégorise les documents entrants en un nombre spécifique de groupes, appelés bucket, sur la base d'une expression spécifiée. Les limites de ces groupes sont automatiquement déterminées afin de répartir les documents de manière uniforme dans le nombre de groupes spécifié.
- Chaque groupe est représenté comme un document dans la sortie. Le document de chaque groupe contient un champ `_id`, dont la valeur spécifie la limite inférieure inclusive et la limite supérieure exclusive du groupe, et un champ de comptage qui contient le nombre de documents dans le groupe. Le champ "count" est inclus par défaut lorsque la sortie n'est pas spécifiée.

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucketAuto

- groupBy expression : Une expression pour regrouper les documents. Pour spécifier un chemin d'accès à un champ, préfixez le nom du champ par le signe \$ et mettez-le entre guillemets.
- buckets integer : Un entier positif de 32 bits qui spécifie le nombre de plages dans lesquelles les documents d'entrée sont groupés.
- document de sortie : Facultatif. Un document qui spécifie les champs à inclure dans les documents de sortie en plus du champ _id. Pour spécifier le champ à inclure, vous devez utiliser des expressions d'accumulation (de groupe)
- Granularity : Le \$bucketAuto accepte un paramètre de granularité optionnel qui garantit que les limites de tous les godets adhèrent à une série de nombres préférentiels spécifiés

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucketAuto

- Syntaxe

{

```
$bucketAuto: {  
    groupBy: <expression>,  
    buckets: <number>,  
    output: {  
        <output1>: { <$accumulator expression> },  
        ...  
    }  
    granularity: <string>  
}  
}
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$bucketAuto

- Exemple

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$collStats

- Renvoie des statistiques concernant une collection ou une vue.
- Syntaxe

```
{
```

```
$collStats:
```

```
{
```

```
    latencyStats: { histograms: <boolean> },
```

```
    storageStats: { scale: <number> },
```

```
    count: {}
```

```
}
```

```
}
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$collStats

- Exemple

```
db.orders.aggregate( [ { $collStats: { latencyStats: { histograms: true } } } ] )
```

```
{
  "ns" : "airbase5.employees",
  "host" : "Gabriel:27017",
  "localTime" : ISODate("2020-02-20T23:22:06.073Z"),
  "latencyStats" : {
    "reads" : {
      "histogram" : [
        {
          "micros" : NumberLong(32),
          "count" : NumberLong(1)
        },
        {
          "micros" : NumberLong(128),
          "count" : NumberLong(1)
        },
        {
          "micros" : NumberLong(3072),
          "count" : NumberLong(1)
        },
        {
          "micros" : NumberLong(4096),
          "count" : NumberLong(2)
        }
      ]
    }
  }
}
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$collStats

- Exemple

```
        },
        {
            "micros" : NumberLong(6144),
            "count" : NumberLong(3)
        }
    ],
    "latency" : NumberLong(33231),
    "ops" : NumberLong(8)
},
"writes" : {
    "histogram" : [
        {
            "micros" : NumberLong(262144),
            "count" : NumberLong(1)
        }
    ],
    "latency" : NumberLong(298154),
    "ops" : NumberLong(1)
},
"commands" : {
    "histogram" : [ ],
    "latency" : NumberLong(0),
    "ops" : NumberLong(0)
},
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$collStats

- Exemple

```
,  
  "transactions" : {  
    "histogram" : [ ],  
    "latency" : NumberLong(0),  
    "ops" : NumberLong(0)  
  }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$count

- Passe à l'étape suivante un document qui contient une décompte du nombre de documents entrés à l'étape précédente.
- Syntaxe

```
{ $count: <string> }
```

- Exemple

```
db.scores.insertMany([  
  { "_id" : 1, "subject" : "History", "score" : 88 },  
  { "_id" : 2, "subject" : "History", "score" : 92 },  
  { "_id" : 3, "subject" : "History", "score" : 97 },  
  { "_id" : 4, "subject" : "History", "score" : 71 },  
  { "_id" : 5, "subject" : "History", "score" : 79 },  
  { "_id" : 6, "subject" : "History", "score" : 83 }]);
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$count

- Exemple

Compte les documents dont score est > 80

```
db.scores.aggregate(
```

```
[  
  {  
    $match: {  
      score: {  
        $gt: 80  
      }  
    }  
  },  
  {  
    $count: "passing_scores"  
  }  
]
```

```
{ "passing_scores": 4 }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$facet

- Traite plusieurs pipelines d'agrégation en une seule étape sur le même ensemble de documents d'entrée. Chaque sous-pipeline a son propre champ dans le document de sortie où ses résultats sont stockés sous forme de tableau de documents.
- L'étape \$facet permet de créer des agrégations à facettes multiples qui caractérisent les données dans plusieurs dimensions, ou facettes, au cours d'une seule étape d'agrégation. Les agrégations à facettes multiples fournissent des filtres et des catégorisations multiples pour guider la navigation et l'analyse des données. Les détaillants utilisent couramment les facettes pour restreindre les résultats de recherche en créant des filtres sur le prix des produits, le fabricant, la taille, etc.
- Les documents d'entrée ne passent qu'une seule fois à l'étape des facettes \$. \$facet permet de faire plusieurs agrégations sur un même ensemble de documents d'entrée, sans avoir besoin de récupérer les documents d'entrée plusieurs fois. Traduit avec www.DeepL.com/Translator (version gratuite)

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$facet

- Syntaxe

```
{ $facet:  
  {  
    <outputField1>: [ <stage1>, <stage2>, ... ],  
    <outputField2>: [ <stage1>, <stage2>, ... ],  
    ...  
  }  
}
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$facet

- Exemple

```
db.inventory.insertMany([
  { "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926,
    "price" : NumberDecimal("199.99"),
    "tags" : [ "painting", "satire", "Expressionism", "caricature" ] },
  { "_id" : 2, "title" : "Melancholy III", "artist" : "Munch", "year" : 1902,
    "price" : NumberDecimal("280.00"),
    "tags" : [ "woodcut", "Expressionism" ] },
  { "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925,
    "price" : NumberDecimal("76.04"),
    "tags" : [ "oil", "Surrealism", "painting" ] },
  { "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" : "Hokusai",
    "price" : NumberDecimal("167.30"),
    "tags" : [ "woodblock", "ukiyo-e" ] },
  { "_id" : 5, "title" : "The Persistence of Memory", "artist" : "Dali", "year" : 1931,
    "price" : NumberDecimal("483.00"),
    "tags" : [ "Surrealism", "painting", "oil" ] },
  { "_id" : 6, "title" : "Composition VII", "artist" : "Kandinsky", "year" : 1913,
    "price" : NumberDecimal("385.00"),
    "tags" : [ "oil", "painting", "abstract" ] },
  { "_id" : 7, "title" : "The Scream", "artist" : "Munch", "year" : 1893,
    "tags" : [ "Expressionism", "painting", "oil" ] },
  { "_id" : 8, "title" : "Blue Flower", "artist" : "O'Keefe", "year" : 1918,
    "price" : NumberDecimal("118.42"),
    "tags" : [ "abstract", "painting" ] }
]);
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$facet

- Exemple

```
db.inventory.aggregate( [  
  {  
    $facet: {  
      "categorizedByTags": [  
        { $unwind: "$tags" },  
        { $sortByCount: "$tags" }  
      ],  
      "categorizedByPrice": [  
        // Filter out documents without a price e.g., _id: 7  
        { $match: { price: { $exists: 1 } } },  
        {  
          $bucket: {  
            groupBy: "$price",  
            boundaries: [ 0, 150, 200, 300, 400 ],  
            default: "Other",  
            output: {  
              "count": { $sum: 1 },  
              "titles": { $push: "$title" }  
            }  
          }  
        }  
      ],  
      "categorizedByYears(Auto)": [  
        {  
          $bucketAuto: {  
            groupBy: "$year",  
            buckets: 4  
          }  
        }  
      ]  
    }  
  ]  
]
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$facet

- Exemple

```
{ "categorizedByTags" : [ { "_id" : "painting", "count" : 6 }, { "_id" : "oil", "count" : 4 }, { "_id" : "Expressionism", "count" : 3 }, { "_id" : "abstract", "count" : 2 }, { "_id" : "Surrealism", "count" : 2 }, { "_id" : "ukiyo-e", "count" : 1 }, { "_id" : "woodblock", "count" : 1 }, { "_id" : "woodcut", "count" : 1 }, { "_id" : "satire", "count" : 1 }, { "_id" : "caricature", "count" : 1 } ], "categorizedByPrice" : [ { "_id" : 0, "count" : 2, "titles" : [ "Dancer", "Blue Flower" ] }, { "_id" : 150, "count" : 2, "titles" : [ "The Pillars of Society", "The Great Wave off Kanagawa" ] }, { "_id" : 200, "count" : 1, "titles" : [ "Melancholy III" ] }, { "_id" : 300, "count" : 1, "titles" : [ "Composition VII" ] }, { "_id" : "Other", "count" : 1, "titles" : [ "The Persistence of Memory" ] } ], "categorizedByYears(Auto)" : [ { "_id" : { "min" : null, "max" : 1902 }, "count" : 2 }, { "_id" : { "min" : 1902, "max" : 1918 }, "count" : 2 }, { "_id" : { "min" : 1918, "max" : 1926 }, "count" : 2 }, { "_id" : { "min" : 1926, "max" : 1931 }, "count" : 2 } ] }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$geoNear

- Produit des documents dans l'ordre du plus proche au plus éloigné d'un point donné.
- Syntaxe

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$graphLookup

- Effectue une recherche récursive sur une collection, avec des options permettant de restreindre la recherche par profondeur de récursion et filtre de requête.
- Syntaxe

{

```
$graphLookup: {  
    from: <collection>,  
    startWith: <expression>,  
    connectFromField: <string>,  
    connectToField: <string>,  
    as: <string>,  
    maxDepth: <number>,  
    depthField: <string>,  
    restrictSearchWithMatch: <document>  
}  
}
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$graphLookup

- Exemple

```
db.employes.aggregate( [  
    {  
        $graphLookup: {  
            from: "employes",  
            startWith: "$reportsTo",  
            connectFromField: "reportsTo",  
            connectToField: "name",  
            as: "reportingHierarchy"  
        }  
    }  
])
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$graphLookup

- Exemple

```
{ "_id" : 1, "name" : "Dev", "reportingHierarchy" : [ ] }
{ "_id" : 2, "name" : "Eliot", "reportsTo" : "Dev", "reportingHierarchy" : [ { "_id" : 1, "name" : "De
v" } ] }
{ "_id" : 3, "name" : "Ron", "reportsTo" : "Eliot", "reportingHierarchy" : [ { "_id" : 1, "name" : "De
v" }, { "_id" : 2, "name" : "Eliot", "reportsTo" : "Dev" } ] }
{ "_id" : 4, "name" : "Andrew", "reportsTo" : "Eliot", "reportingHierarchy" : [ { "_id" : 1, "name" :
"Dev" }, { "_id" : 2, "name" : "Eliot", "reportsTo" : "Dev" } ] }
{ "_id" : 5, "name" : "Asya", "reportsTo" : "Ron", "reportingHierarchy" : [ { "_id" : 1, "name" : "Dev
" }, { "_id" : 2, "name" : "Eliot", "reportsTo" : "Dev" }, { "_id" : 3, "name" : "Ron", "reportsTo" :
"Eliot" } ] }
{ "_id" : 6, "name" : "Dan", "reportsTo" : "Andrew", "reportingHierarchy" : [ { "_id" : 2, "name" : "E
liot", "reportsTo" : "Dev" }, { "_id" : 1, "name" : "Dev" }, { "_id" : 4, "name" : "Andrew", "reportsT
o" : "Eliot" } ] }
>
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$group (Group By)

- Regroupe les documents en entrée par l'expression `_id` spécifiée et, pour chaque regroupement distinct, produit un document.
- Le champ `_id` de chaque document de sortie contient le groupe unique par valeur.
- Les documents de sortie peuvent également contenir des champs calculés qui contiennent les valeurs d'une expression de groupe
- Syntaxe

{

`$group:`

{

`_id: <expression>, // Group By Expression`

`<field1>: { <accumulator1> : <expression1> },`

...

}

}

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$group (Group By)

- Exemple

```
db.books.insertMany([
  { "_id" : 8751, "title" : "The Banquet", "author" :
    "Dante", "copies" : 2 },
  { "_id" : 8752, "title" : "Divine Comedy", "author" :
    "Dante", "copies" : 1 },
  { "_id" : 8645, "title" : "Eclogues", "author" : "Dante",
    "copies" : 2 },
  { "_id" : 7000, "title" : "The Odyssey", "author" :
    "Homer", "copies" : 10 },
  { "_id" : 7020, "title" : "Iliad", "author" : "Homer",
    "copies" : 10 }
])
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$group (Group By)

- Exemple

```
db.books.aggregate([
  { $group : { _id : "$author", books: { $push: "$title" } } }
])
D
```

```
> db.books.aggregate([
...   { $group : { _id : "$author", books: { $push: "$title" } } }
... ])
{ "_id" : "Homer", "books" : [ "The Odyssey", "Iliad" ] }
{ "_id" : "Dante", "books" : [ "The Banquet", "Divine Comedy", "Eclogues" ] }
>
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$indexStats

- Renvoie des statistiques concernant l'utilisation de chaque indice pour la collecte. S'il fonctionne avec un contrôle d'accès, l'utilisateur doit avoir des privilèges qui incluent l'action indexStats.
- Syntaxe

```
{ $indexStats: { } }
```

- Exemple

```
db.orders.insertMany([
  { "_id" : 1, "item" : "abc", "price" : 12, "quantity" : 2,
    "type": "apparel" },
  { "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1,
    "type": "electronics" },
  { "_id" : 3, "item" : "abc", "price" : 10, "quantity" : 5,
    "type": "apparel" }
]);
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$indexStats

- Exemple : création d'indexes

```
db.orders.createIndex( { item: 1, quantity: 1 } )
```

```
db.orders.createIndex( { type: 1, item: 1 } )
```

- Exemple : quelques actions

```
db.orders.find( { type: "apparel"} )
```

```
{ "_id" : 1, "item" : "abc", "price" : 12, "quantity" : 2, "type" : "apparel" }
{ "_id" : 3, "item" : "abc", "price" : 10, "quantity" : 5, "type" : "apparel" }
```

```
db.orders.find( { item: "abc" } ).sort( { quantity: 1 } )
```

```
{ "_id" : 1, "item" : "abc", "price" : 12, "quantity" : 2, "type" : "apparel" }
{ "_id" : 3, "item" : "abc", "price" : 10, "quantity" : 5, "type" : "apparel" }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$indexStats

- Exemple : index stats

```
db.orders.aggregate( [ { $indexStats: { } } ] )
```

```
{ "name" : "item_1_quantity_1", "key" : { "item" : 1, "quantity" : 1 }, "host" : "Gabriel:27017", "accesses" : { "ops" : NumberLong(1), "since" : ISODate("2020-02-20T22:25:16.120Z") } }
{ "name" : "_id_", "key" : { "_id" : 1 }, "host" : "Gabriel:27017", "accesses" : { "ops" : NumberLong(0), "since" : ISODate("2020-02-20T22:23:49.313Z") } }
{ "name" : "type_1_item_1", "key" : { "type" : 1, "item" : 1 }, "host" : "Gabriel:27017", "accesses" : { "ops" : NumberLong(1), "since" : ISODate("2020-02-20T22:25:20.261Z") } }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$limit

- Limite le nombre de documents qui passent à l'étape suivante. Renvoie les N premiers documents

- Syntaxe

```
{ $limit: <positive integer> }
```

- Exemple

```
db.orders.aggregate(  
  { $limit : 2 }  
)
```

```
{ "_id" : 1, "item" : "abc", "price" : 12, "quantity" : 2, "type" : "apparel" }  
{ "_id" : 2, "item" : "jkl", "price" : 20, "quantity" : 1, "type" : "electronics" }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$listSessions

- Liste toutes les sessions stockées dans la collection system.sessions dans la base de données config. Ces sessions sont visibles par tous les membres du déploiement MongoDB.

- Syntaxe

```
{ $listSessions: <document> }
```

- Exemple

```
use config
```

```
db.system.sessions.aggregate( [ { $listSessions: { } } ] )
```

```
> use config
switched to db config
>
> db.system.sessions.aggregate( [ { $listSessions: { } } ] )
{ "_id" : { "id" : UUID("838427bd-633b-4936-85b1-dfda13919918"), "uid" : BinData(0, "47DEQpj8HBSa+/TImW+5JCeRkm5NMpJWZG3hSuFU="), "lastUse" : ISODate("2020-02-20T22:38:54.346Z") }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$lookup (jointure)

- Effectue une jointure extérieure gauche à une collection non durcie dans la même base de données pour filtrer les documents de la collection "jointe" en vue de leur traitement.
- À chaque document saisi, l'étape \$lookup ajoute un nouveau champ de tableau dont les éléments sont les documents correspondants de la collection "jointe".
- L'étape \$lookup fait passer ces documents remodelés à l'étape suivante.
- Nous ne traitons ici que l'équijointure dans un premier temps.

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$lookup (jointure)

- Equijointure
 - Pour effectuer une correspondance d'égalité entre un champ des documents d'entrée et un champ des documents de la collection "jointe", l'étape \$lookup a la syntaxe suivante :
:

```
db.localCollection.aggregate([ {  
    $lookup:  
        {  
            from: <collection to join>,  
            localField: <field from the input  
            documents from localCollection>,  
            foreignField: <field from the  
            documents of the "from" collection>,  
            as: <output array field>  
        }  
    }])
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$lookup (jointure)

- Equijointure
 - Equivaut à

```
SELECT *, <output array field>
FROM collection
WHERE <output array field>
IN (SELECT *
FROM <collection to join>
WHERE <foreignField>=
<collection.localField>);
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$lookup (jointure)

- Equijointure : Exemple

```
db.clients.aggregate([
  {
    $lookup:
      {
        from: "vols",
        localField: "_id",
        foreignField: "appreciations.idClient",
        as: "clients_vols"
      }
  }
]).pretty();
```

Renvoie tous les clients et leurs vols

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$lookup (jointure)

- Equijointure : Exemple

```
db.clients.aggregate([
  {
    $match: { "_id": 1 }
  },
  {
    $lookup:
      {
        from: "vols",
        localField: "_id",
        foreignField:"appreciations.idClient",
        as: "clients_vols"
      }
  }
]).pretty();
```

Renvoie uniquement le client 1 et ses vols
\$match permet de restreindre.

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$match

- Filtre les documents pour ne transmettre que les documents qui correspondent à la (aux) condition(s) spécifiée(s) à l'étape suivante du pipeline
- Syntaxe

```
{ $match: { <query> } }
```

- Exemple

```
db.clients.aggregate([
  {
    $match: { "_id": 1 }
  }
]).pretty();
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$match

- Exemple

```
> use airbase
switched to db airbase
> db.clients.aggregate([
...  {
...    $match: { "_id": 1 }
...  }]).pretty();
{
  "_id" : 1,
  "nom" : "ERZULIE",
  "prenoms" : [
    "Maria",
    "Frida"
  ],
  "telephone" : "00509232472",
  "adresse" : {
    "numero" : 11,
    "rue" : "Rue des miracles",
    "codePostal" : "HT8110",
    "ville" : "PORT-AU-PRINCE",
    "pays" : "HAITI"
  },
  "dateNaissance" : "01/01/1880"
}
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$out

- Prend les documents retournés par le pipeline d'agrégation et les écrit dans une collection spécifique. L'opérateur \$out doit être la dernière étape du pipeline. L'opérateur \$out permet au cadre d'agrégation de renvoyer des ensembles de résultats de n'importe quelle taille.

- Syntaxe

```
{ $out: "<output-collection>" }
```

- Exemple

```
db.clients.aggregate([
  {
    $match: { "_id": 1 },
  },
  {$out:"quelquesClients" }
]).pretty();
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$out

- Exemple : résultat

```
> show collections
artists
clients
devoirs
eleves
etudiant
pilote
quelquesClients
scores
vols
>
```

```
> db.quelquesClients.find().pretty();
{
    "_id" : 1,
    "nom" : "ERZULIE",
    "prenoms" : [
        "Maria",
        "Frida"
    ],
    "telephone" : "00509232472",
    "adresse" : {
        "numero" : 11,
        "rue" : "Rue des miracles",
        "codePostal" : "HT8110",
        "ville" : "PORT-AU-PRINCE",
        "pays" : "HAITTI"
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$project

- Transmet les documents avec les champs demandés à l'étape suivante du processus. Les champs spécifiés peuvent être des champs existants dans les documents de saisie ou des champs nouvellement calculés.
- Syntaxe

```
{ $project: { <specification(s)> } }
```

- Exemple

```
db.clients.aggregate([
  {"$project": {
    "nom": 1
  }
}).pretty();
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$project

- Exemple : résultat

```
switched to db airbase
> db.clients.aggregate([
... {"$project": {
...   "nom":1
... }
... }
... ]).pretty();
{ "_id" : 2, "nom" : "BARON" }
{ "_id" : 1, "nom" : "ERZULIE" }
{ "_id" : 3, "nom" : "AGWE" }
```

- Exemple 2

```
db.clients.aggregate([
{"$project": {
  "nom":1,
  "adresse":1,
  "_id":0
}
]);
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$project

- Exemple 2: résultat

```
> db.clients.aggregate([ {"$project": { "nom":1, "adresse":1, "_id":0 } } ]);  
{ "nom" : "BARON", "adresse" : { "numero" : 1, "rue" : "Rue des lwa", "codePostal" : "HT8111", "ville" : "PORT-AU-PRINCE", "pays" : "HAITI" } }  
{ "nom" : "ERZULIE", "adresse" : { "numero" : 11, "rue" : "Rue des miracles", "codePostal" : "HT8110", "ville" : "PORT-AU-PRINCE", "pays" : "HAITI" } }  
{ "nom" : "AGWE", "adresse" : { "numero" : 1, "rue" : "Rue des lwa", "codePostal" : "HT8111", "ville" : "PORT-AU-PRINCE", "pays" : "HAITI" } }  
>
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$redact

- Restreint le contenu des documents sur la base des informations stockées dans les documents eux-mêmes
- Syntaxe

{ \$redact: <expression> }

- Exemple.

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$replaceRoot

- Remplace le document en entrée par le document spécifié.
- L'opération remplace tous les champs existants dans le document de saisie, y compris le champ `_id`.
- Vous pouvez promouvoir un document intégré existant au niveau supérieur, ou créer un nouveau document à des fins de promotion (voir exemple).
- Syntaxe

```
{ $replaceRoot: { newRoot: <replacementDocument> } }
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$replaceRoot

- Exemple

```
db.personnes.insertMany([
    { "_id": 1, "name" : { "first" : "John",
    "last" : "Backus" } },
    { "_id": 2, "name" : { "first" : "John",
    "last" : "McCarthy" } },
    { "_id": 3, "name": { "first" : "Grace",
    "last" : "Hopper" } }
]);
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$replaceRoot

- Exemple

```
db.personnes.aggregate([
  { $replaceRoot: { newRoot: "$name" } }
])
```

```
> db.personnes.aggregate([
...   { $replaceRoot: { newRoot: "$name" } }
... ])
{ "first" : "John", "last" : "Backus" }
{ "first" : "John", "last" : "McCarthy" }
{ "first" : "Grace", "last" : "Hopper" }
>
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$replaceRoot

- Exemple 2 : créer et sauver la nouvelle collection

```
db.personnes.aggregate([
  { $replaceRoot: { newRoot: "$name" } },
  {$out:"newpers" }
])
```

```
> show collections
clients
newpers
personnes
vols
> db.newpers.find();
{ "_id" : ObjectId("5e4edc51fe514debe23a1aad"), "first" : "John", "last" : "Backus" }
{ "_id" : ObjectId("5e4edc51fe514debe23a1aae"), "first" : "John", "last" : "McCarthy" }
{ "_id" : ObjectId("5e4edc51fe514debe23a1aaf"), "first" : "Grace", "last" : "Hopper" }
>
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$sample

- Sélectionne au hasard (aléatoirement) le nombre de documents spécifié dans l'expression

- Syntaxe

```
{$sample: { size: <positive integer> } }
```

- Exemple

```
db.sampleSource.insertMany([
  { "_id" : 1, "name" : "david", "q1" : true, "q2" : true },
  { "_id" : 2, "name" : "Zorro", "q1" : false, "q2" : false },
  { "_id" : 3, "name" : "Annie", "q1" : true, "q2" : true },
  { "_id" : 4, "name" : "Allan", "q1" : true, "q2" : false },
  { "_id" : 5, "name" : "Zembla", "q1" : false, "q2" : true },
  { "_id" : 6, "name" : "Zoé", "q1" : true, "q2" : true },
  { "_id" : 7, "name" : "Sonia", "q1" : false, "q2" : true }
]);
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$sample

- Exemple

```
db.sampleSource.aggregate(  
  [ { $sample: { size: 3 } } ]  
)
```

- Résultat

```
> db.sampleSource.aggregate(  
...   [ { $sample: { size: 3 } } ]  
... )  
{ "_id" : 4, "name" : "Allan", "q1" : true, "q2" : false }  
{ "_id" : 5, "name" : "Zembla", "q1" : false, "q2" : true }  
{ "_id" : 2, "name" : "Zorro", "q1" : false, "q2" : false }  
> db.sampleSource.aggregate( [ { $sample: { size: 3 } } ] )  
{ "_id" : 6, "name" : "Zoé", "q1" : true, "q2" : true }  
{ "_id" : 7, "name" : "Sonia", "q1" : false, "q2" : true }  
{ "_id" : 4, "name" : "Allan", "q1" : true, "q2" : false }  
>
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$skip

- Sauter le nombre spécifié de documents qui passent à l'étape et faire passer les documents restants à l'étape suivante.

- Permet de sauter N document

- Syntaxe

```
{ $skip: <positive integer> }
```

- Exemple : sauter les 5 premiers documents

```
db.sampleSource.aggregate({ $skip : 5 });
```

```
> db.sampleSource.aggregate(  
... { $skip : 5 }  
... );  
{ "_id" : 6, "name" : "Zoé", "q1" : true, "q2" : true }  
{ "_id" : 7, "name" : "Sonia", "q1" : false, "q2" : true }  
>
```

11. Les agrégats dans MongoDB

□ Etape d'agrégation \$sort

- trie tous les documents en entrée et les renvoie au pipeline dans l'ordre spécifié
- Syntaxe

```
{ $sort: { <field1>: <sort order>, <field2>: <sort order>
... } }
```

- Exemple
- ```
db.sampleSource.aggregate(
 { $sort : { "q1": 1 , name : 1 } }
);
```

# 11. Les agrégats dans MongoDB

## □ Etape d'agrégation \$sort

- Exemple

```
> db.sampleSource.aggregate(
... { $sort : { "q1": 1 , name : 1 } }
...);
{ "_id" : 7, "name" : "Sonia", "q1" : false, "q2" : true }
{ "_id" : 5, "name" : "Zembla", "q1" : false, "q2" : true }
{ "_id" : 2, "name" : "Zorro", "q1" : false, "q2" : false }
{ "_id" : 4, "name" : "Allan", "q1" : true, "q2" : false }
{ "_id" : 3, "name" : "Annie", "q1" : true, "q2" : true }
{ "_id" : 6, "name" : "Zoé", "q1" : true, "q2" : true }
{ "_id" : 1, "name" : "david", "q1" : true, "q2" : true }
>
```

# 11. Les agrégats dans MongoDB

---

## □ Etape d'agrégation \$sortByCount

- Regroupe les documents entrants en fonction de la valeur d'une expression donnée, puis calcule le nombre de documents dans chaque groupe distinct.
- Chaque document en sortie contient deux champs : un champ `_id` contenant la valeur du groupe distinct, et un champ de comptage contenant le nombre de documents appartenant à ce groupe ou à cette catégorie.
- Les documents sont triés par nombre dans l'ordre décroissant.
- Syntaxe  
`{ $sortByCount: <expression> }`

# 11. Les agrégats dans MongoDB

## □ Etape d'agrégation \$sortByCount

- Exemple

```
db.sampleSource.aggregate(
 { $sortByCount : "$q1" }
)
```

```
> db.sampleSource.aggregate(
... { $sortByCount : "$q1" }
...);
{ "_id" : true, "count" : 4 }
{ "_id" : false, "count" : 3 }
```

# 11. Les agrégats dans MongoDB

---

## □ Etape d'agrégation \$unwind (unnest)

- Déconstruit un champ de type tableau à partir des documents d'entrée pour produire un document pour chaque élément. Chaque document de sortie est le document d'entrée, la valeur du champ de tableau étant remplacée par l'élément.
- Syntaxe  
{ \$unwind: <field path> }

# 11. Les agrégats dans MongoDB

## □ Etape d'agrégation \$unwind (unnest)

- Exemple

```
db.oeuvres.insertMany([
 { "_id" : 1, "title" : "The Pillars of Society", "artist" :
 "Grosz", "year" : 1926, "tags" : ["painting", "satire",
 "Expressionism", "caricature"] },
 { "_id" : 2, "title" : "Melancholy III", "artist" : "Munch",
 "year" : 1902, "tags" : ["woodcut", "Expressionism"] },
 { "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" :
 1925, "tags" : ["oil", "Surrealism", "painting"] },
 { "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" :
 "Hokusai", "tags" : ["woodblock", "ukiyo-e"] },
 { "_id" : 5, "title" : "The Persistence of Memory", "artist" :
 "Dali", "year" : 1931, "tags" : ["Surrealism", "painting",
 "oil"] },
 { "_id" : 6, "title" : "Composition VII", "artist" :
 "Kandinsky", "year" : 1913, "tags" : ["oil", "painting",
 "abstract"] },
 { "_id" : 7, "title" : "The Scream", "artist" : "Munch", "year" :
 1893, "tags" : ["Expressionism", "painting", "oil"] },
 { "_id" : 8, "title" : "Blue Flower", "artist" : "O'Keefe",
 "year" : 1918, "tags" : ["abstract", "painting"] }
]);
```

# 11. Les agrégats dans MongoDB

## □ Etape d'agrégation \$unwind (unnest)

- Exemple

```
db.oeuvres.aggregate(
 [{ $unwind : "$tags" }]
);
```

```
> db.oeuvres.aggregate(
... [{ $unwind : "$tags" }]
...);
{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926, "tags" : "painting"
}
{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926, "tags" : "satire"
}
{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926, "tags" : "Expressionism"
}
{ "_id" : 1, "title" : "The Pillars of Society", "artist" : "Grosz", "year" : 1926, "tags" : "caricature"
}
{ "_id" : 2, "title" : "Melancholy III", "artist" : "Munch", "year" : 1902, "tags" : "woodcut" }
{ "_id" : 2, "title" : "Melancholy III", "artist" : "Munch", "year" : 1902, "tags" : "Expressionism" }

{ "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925, "tags" : "oil" }
{ "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925, "tags" : "Surrealism" }
{ "_id" : 3, "title" : "Dancer", "artist" : "Miro", "year" : 1925, "tags" : "painting" }
{ "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" : "Hokusai", "tags" : "woodblock" }
{ "_id" : 4, "title" : "The Great Wave off Kanagawa", "artist" : "Hokusai", "tags" : "ukiyo-e" }
{ "_id" : 5, "title" : "The Persistence of Memory", "artist" : "Dali", "year" : 1931, "tags" : "Surrealism" }
{ "_id" : 5, "title" : "The Persistence of Memory", "artist" : "Dali", "year" : 1931, "tags" : "painting" }
```

# 12. API Java MongoDB

---

## □ Plan

- Installation du driver MongoDB JDBC
- Connexion à la base
- Création d'une collection
- Chargement d'une collection
- Insertion de documents dans une collection
- Recherche de document dans une collection
- Modification de documents
- Suppression de documents

## 12. API Java MongoDB

---

### □ Installation du driver MongoDB JDBC

- Nous supposons que vous avez déjà installé java sur votre machine
- Créer un dossier **tpmongodb** sur un disque où vous avez de la place. Dans ce dossier créer ensuite les sous-dossiers suivants :
  - **mongojar**
  - **tp**
- Télécharger mongo-java-driver-3.12.7.jar depuis ici :  
<http://central.maven.org/maven2/org/mongodb/mongo-java-driver/3.12.7/>. Placez ce jar dans le dossier **tpmongodb\mongojar**
- Placez tous vos programmes dans **tp**
- Dans chacun de vos programmes, il faut définir **tp** comme package

# 12. API Java MongoDB

---

## □ Installation du driver MongoDB JDBC

- Assurez que la variable d'environnement PATH contient le path java : ..\java\jdk-x.x.x\bin
- Définir une variable d'environnement comme suit :  
Set MYPATH=D:\tpmongodb

- **Ligne de compilation**

```
javac -g -cp %MYPATH%\mongojar\mongo-java-driver-3.12.7.jar;%MYPATH%
```

```
%MYPATH%\tp\nomFichier.java
```

- **Ligne d'exécution**

```
java -Xmx256m -Xms256m -cp
%MYPATH%\mongojar\mongo-java-driver-3.12.7.jar;%MYPATH% tp.nomFichier
```

# 12. API Java MongoDB

## □ Connexion à la base

```
package tp;
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class ConnectToDB {

 public static void main(String args[]) {

 // Creating a Mongo client
 MongoClient mongo = new MongoClient("localhost" , 27017);

 // Creating Credentials
 MongoCredential credential;
 credential = MongoCredential.createCredential("sampleUser", "myDb",
 "password".toCharArray());
 System.out.println("Connected to the database successfully");

 // Accessing the database
 MongoDatabase database = mongo.getDatabase("myDb");
 System.out.println("Credentials ::"+ credential);

 }
}
```

# 12. API Java MongoDB

## □ Crédation d'une collection

```
package tp;
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class CreatingCollection {

 public static void main(String args[]) {

 // Creating a Mongo client
 MongoClient mongo = new MongoClient("localhost" , 27017);

 // Creating Credentials
 MongoCredential credential;
 credential = MongoCredential.createCredential("sampleUser", "myDb",
 "password".toCharArray());
 System.out.println("Connected to the database successfully");

 //Accessing the database
 MongoDatabase database = mongo.getDatabase("myDb");

 //Creating a collection
 database.createCollection("sampleCollection");
 System.out.println("Collection created successfully");
 }
}
```

# 12. API Java MongoDB

## □ Chargement d'une collection

```
package tp;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class SelectingCollection {

 public static void main(String args[]) {

 // Creating a Mongo client
 MongoClient mongo = new MongoClient("localhost" , 27017);

 // Creating Credentials
 MongoCredential credential;
 credential = MongoCredential.createCredential("sampleUser", "myDb",
 "password".toCharArray());
 System.out.println("Connected to the database successfully");

 // Accessing the database
 MongoDatabase database = mongo.getDatabase("myDb");

 // Retieving a collection
 MongoCollection<Document> collection =
 database.getCollection("myCollection");
 System.out.println("Collection myCollection selected successfully");
 }
}
```

# 12. API Java MongoDB

## □ Insertion de documents dans une collection

```
package tp;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class InsertingDocument {
 public static void main(String args[]) {
 // Creating a Mongo client
 MongoClient mongo = new MongoClient("localhost" , 27017);
 // Creating Credentials
 MongoCredential credential;
 credential = MongoCredential.createCredential("sampleUser", "myDb",
 "password".toCharArray());
 System.out.println("Connected to the database successfully");
 // Accessing the database
 MongoDatabase database = mongo.getDatabase("myDb");
 // Retrieving a collection
 MongoCollection<Document> collection =
 database.getCollection("sampleCollection");
 System.out.println("Collection sampleCollection selected successfully");
 Document document = new Document("title", "MongoDB")
 .append("id", 1)
 .append("description", "database")
 .append("likes", 100)
 .append("url", "http://www.tutorialspoint.com/mongodb/")
 .append("by", "tutorialspoint");
 collection.insertOne(document);
 System.out.println("Document inserted successfully");
 }
}
```

# 12. API Java MongoDB

## □ Rechercher tous les documents

```
package tp;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import java.util.Iterator;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
public class RetrievingAllDocuments {
 public static void main(String args[]) {
 // Creating a Mongo client
 MongoClient mongo = new MongoClient("localhost" , 27017);
 // Creating Credentials
 MongoCredential credential;
 credential = MongoCredential.createCredential("sampleUser", "myDb",
 "password".toCharArray());
 System.out.println("Connected to the database successfully");
 // Accessing the database
 MongoDatabase database = mongo.getDatabase("myDb");
 // Retrieving a collection
 MongoCollection<Document> collection =
database.getCollection("sampleCollection");
 System.out.println("Collection sampleCollection selected successfully");
 // Getting the iterable object
FindIterable<Document> iterDoc = collection.find();
 int i = 1;
 // Getting the iterator
 Iterator it = iterDoc.iterator();
 while (it.hasNext()) {
 System.out.println(it.next());
 i++;
 }
 }
}
```

# 12. API Java MongoDB

## □ Modification de documents

```
package tp;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import java.util.Iterator;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
public class UpdatingDocuments {
 public static void main(String args[]) {
 // Creating a Mongo client
 MongoClient mongo = new MongoClient("localhost" , 27017);
 // Creating Credentials
 MongoCredential credential;
 credential = MongoCredential.createCredential("sampleUser", "myDb",
 "password".toCharArray());
 System.out.println("Connected to the database successfully");
 // Accessing the database
 MongoDatabase database = mongo.getDatabase("myDb");
 // Retrieving a collection
 MongoCollection<Document> collection =
database.getCollection("sampleCollection");
 System.out.println("Collection myCollection selected successfully");
collection.updateOne(Filters.eq("id", 1), Updates.set("likes", 150));
 System.out.println("Document update successfully...");
 // Retrieving the documents after updation
 // Getting the iterable object
 FindIterable<Document> iterDoc = collection.find();
 int i = 1;
```

# 12. API Java MongoDB

---

## □ Modification de documents

```
// Getting the iterator
Iterator it = iterDoc.iterator();
while (it.hasNext()) {
 System.out.println(it.next());
 i++;
}
}
```

# 12. API Java MongoDB

## □ Suppression du document

```
package tp;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;

import java.util.Iterator;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;

public class DeletingDocuments {

 public static void main(String args[]) {

 // Creating a Mongo client
 MongoClient mongo = new MongoClient("localhost" , 27017);

 // Creating Credentials
 MongoCredential credential;
 credential = MongoCredential.createCredential("sampleUser", "myDb",
 "password".toCharArray());
 System.out.println("Connected to the database successfully");

 // Accessing the database
 MongoDatabase database = mongo.getDatabase("myDb");

 // Retrieving a collection
 MongoCollection<Document> collection =
database.getCollection("sampleCollection");
 System.out.println("Collection sampleCollection selected successfully");
 }
}
```

# 12. API Java MongoDB

---

## □ Suppression du document

```
// Deleting the documents
collection.deleteOne(Filters.eq("id", 1));
System.out.println("Document deleted successfully...");

// Retrieving the documents after updation
// Getting the iterable object
FindIterable<Document> iterDoc = collection.find();
int i = 1;

// Getting the iterator
Iterator it = iterDoc.iterator();

while (it.hasNext()) {
 System.out.println("Inserted Document: "+i);
 System.out.println(it.next());
 i++;
}
}
```

# 13.QuiZ

---

## □ 13. Quiz

# 14. Bilan et perspectives

---

## □ 14. Bilan et perspectives

- Ce cours est une introduction à MongoDB
- A l'issue de ce cours vous devez être à même de
  - Positionner le moteur NoSql MongoDB
  - Comprendre l'architecture d'une instance MongoDB
  - De gérer des bases de données
  - De gérer des collections
  - De Gérer des Documents
  - De poser des index
  - De consulter des documents
  - De prendre en main l'API Java
- En Perspective pour aller plus loin
  - Il faut aller vers plus d'administration
  - Gérer des jointures
  - Approfondir la manipulation des données
  - Gérer des données GRAPHE, etc.

# 15. Webographie et Bibliographie

---

## □ 15. Webographie et Bibliographie

- [1] Lien vers la documentation MongoDB  
<https://docs.mongodb.com/manual/>
- [2] MongoDB in Action, de Kyle Banker , Peter Bakkum, et al. | 7 avril 2016
- [3] API Java MongoDB  
<https://mongodb.github.io/mongo-java-driver/>
- [4] Tutorial programmation Java avec MongoDB  
[https://www.tutorialspoint.com/mongodb/mongodb\\_java.htm](https://www.tutorialspoint.com/mongodb/mongodb_java.htm)
- [5] Lien vers le download des utilitaire MongoDB  
[https://www.mongodb.com/try/download/database-tools?tck=docs\\_databasetools](https://www.mongodb.com/try/download/database-tools?tck=docs_databasetools)
- [6] API du Java MongoDB Driver 3.12  
<https://mongodb.github.io/mongo-java-driver/3.12/javadoc/index.html>
- [7] Migration de MongoDB vers PostGres  
<https://eventuallycoding.com/2019/07/03/mongodb-vers-postgresql/>

# 16. Exercices

---

## □ Plan

- Voir le fichier dans la dropbox dossier exercices