



RÉSUMÉ THÉORIQUE – FILIÈRE DÉVELOPPEMENT DIGITAL OPTION WEB FULL STACK

M111 – GÉRER LES DONNÉES

Elaboré par :

Widad JAKJOURD

Formatrice à ISTA TIC - SAFI



90 heures



Equipe de rédaction et de lecture

Equipe de rédaction :

Mme Jakjoud Widad : Formatrice en
développement digital

Equipe de lecture :

Mme Laouija Soukaina : Formatrice animatrice au
CDC Digital & IA



SOMMAIRE

1. Exploiter les fonctionnalités avancées d'un SGBD relationnel

Maitriser le langage de programmation procédurale sous MySQL
Optimiser une base de données MySQL
Protéger la base de données MySQL

2. Exploiter les fonctionnalités des bases de données NoSQL MongoDB

Découvrir les bases de données NoSQL
Mettre en place une base de données MongoDB
Modéliser les documents
Manipuler les données avec mongoDB
Effectuer des requêtes depuis des programmes Python
Sécuriser une base de données MongoDB

MODALITÉS PÉDAGOGIQUES



1

LE GUIDE DE SOUTIEN
Il contient le résumé théorique et le manuel des travaux pratiques



2

LA VERSION PDF
Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life



3

DES CONTENUS TÉLÉCHARGEABLES
Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life



4

DU CONTENU INTERACTIF
Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life



5

DES RESSOURCES EN LIGNES
Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage



PARTIE 2

Exploiter les fonctionnalités des bases de données NoSQL MongoDB

Dans ce module, vous allez :

- Découvrir les bases de données NoSQL
- Mettre en place une base de données MongoDB
- Modéliser les documents
- Manipuler les données avec mongoDB
- Effectuer des requêtes depuis des programmes Python
- Sécuriser une base de données MongoDB



50 heures



CHAPITRE 1

Découvrir les bases de données NoSQL

Ce que vous allez apprendre dans ce chapitre :

- Définir le concept de bases de données NoSQL,
- Comparer les bases de données traditionnelles et NoSQL,
- Recenser les caractéristiques des NoSQL
- Identifier les bases de données NoSQL,
- Recenser les types de bases de données NoSQL (document, clé / valeur, colonne, graphe)
- Comparer les différents types de bases de données NoSQL



05 heures

CHAPITRE 1

Découvrir les bases de données NoSQL

1. Définir le concept de bases de données NoSQL,
2. Comparer les bases de données traditionnelles et NoSQL,
3. Recenser les caractéristiques des NoSQL,
4. Recenser les types de bases de données NoSQL (document, clé / valeur, colonne, graphe),
5. Comparer les différents types de bases de données NoSQL



01 – Introduction aux Bases de données NoSQL

C'est quoi une BD NoSQL?

Des SGBD Relationnels au NoSQL

- Les défis majeurs des **SGBDs** étaient toujours le stockage des données et la recherche des données,
- Les **SGBDR** sont adaptés à gérer des données bien structurées de types simples (chaines de caractères, entier, ...) et représentables sous forme de tables (colonnes => propriétés et lignes => données),
- Ils reposent sur le modèle relationnelle d'Edgard Codd et ont prouvé leur **efficacité** pour des décennies grâce à:

Une séparation logique et physique

Une forte structuration des données et un fort typage

Une représentation tabulaire

Un langage déclaratif (SQL)

Un ensemble de contraintes permettant d'assurer l'intégrité des données

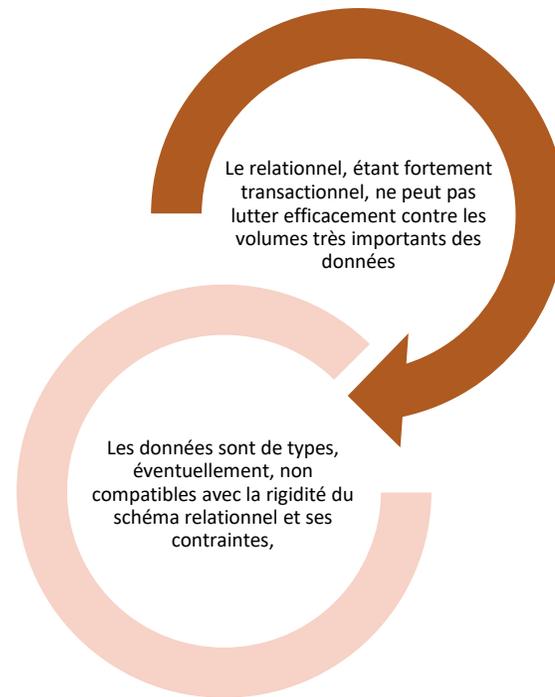
Et une forte cohérence transactionnelle

01 – Introduction aux Bases de données NoSQL

C'est quoi une BD NoSQL?

Des SGBD Relationnels au NoSQL

Mais , les **SGBDR** ont montré leur limite face **aux 3V (Volume, Velocity, Veracity)** que caractérisent l'ére actuelle des données **(Big Data)**:



01 – Introduction aux Bases de données NoSQL

C'est quoi une BD NoSQL?



Des SGBD Relationnels au NoSQL

- Le **NoSQL** (Not Only SQL) propose une nouvelle manière de gérer les données, sans respecter *forcement* le paradigme relationnel,
- Le **NoSQL** supporte de nouveaux types de données (xml, collections d'objets, triplets,...) ,
- Cette approche propose de relâcher certaines contraintes lourdes du relationnel (structure des données, langage d'interrogation ou la cohérence) pour favoriser la distribution,
- Le **NoSQL** ne remplace pas les bases **SQL**, il les complète en apportant des avantages en terme de stockage réparti par exemple.



01 – Introduction aux Bases de données NoSQL

C'est quoi une BD NoSQL?

Définition

- Le **NoSQL** est un ensemble de technologies de **BD** reposant sur un modèle différent du modèle relationnel,
- Les Bases **NoSQL** sont le fruit du mouvement **NoSQL** apparu au milieu des années 2000,
- Le mouvement a initialement piloté les besoins Big Data des principaux acteurs du web **GAFA** (Google, Amazone, Facebook, Apple,...):

Google avec sa base
Hbase

Apple avec sa base *„„„„„*

Facebook avec sa base
Cassandra

Amazone avec sa base
DynamoDB

- Les serveurs de données **NoSQL** se caractérisent par des architectures distribuées ce qui leur permettent de mieux répondre aux problématiques du big data.

01 – Introduction aux Bases de données NoSQL

C'est quoi une BD NoSQL?



Avantages du NoSQL

- Le format de la base **NoSQL** est basée essentiellement sur des paires clé-valeur beaucoup plus simple à mettre en œuvre,
- Il est possible de stocker directement des objets manipulés dans des langages de programmation comme des listes, des collections d'objets, des tableaux de valeurs,...
- Les bases de données **NoSQL** sont pour la plupart Open-source et ne possèdent pas de droits de licence,
- Il est très facile d'étendre une base de données **NoSQL** en rajoutant, tout simplement des serveurs,
- Les données sont regroupées par unités logiques et non dans des tables ce qui facilite la manipulation .
- **Par exemple**, pour avoir les informations d'un client qui a passé une commande donnée, on aura pas besoin de passer par des jointures entre les tables client et commande.

01 – Introduction aux Bases de données NoSQL

C'est quoi une BD NoSQL?



Inconvénients du NoSQL

- Absence du concept de clé étrangère, ce qui veut dire qu'il n'y a pas de mécanisme pour vérifier la cohérence des données (il faut le faire au niveau de la programmation),
- **NoSQL** n'est pas adaptable aux applications basées sur des transactions sécurisées et fiables (Gestion bancaire par exemple),
- Les requêtes **SQL** et **NoSQL** ne sont pas compatibles.

CHAPITRE 1

Découvrir les bases de données NoSQL

1. Définir le concept de bases de données NoSQL,
- 2. Comparer les bases de données traditionnelles et NoSQL,**
3. Recenser les caractéristiques des NoSQL
4. Recenser les types de bases de données NoSQL (document, clé / valeur, colonne, graphe)
5. Comparer les différents types de bases de données NoSQL



01 – Introduction aux Bases de données NoSQL

SQL ou NoSQL?



Comparaison entre les bases de données traditionnelles et NoSQL

Base de données SQL

Les données sont représentées sous forme de tables composées de n nombre de lignes de données, Elles respectent un schéma stricte et standard.

L'augmentation de la charge est gérée par l'augmentation du processeur, de la RAM, du SSD, etc. sur un seul serveur:
Scalabilité (mise à l'échelle) verticale.

L'augmentation de la charge n'est pas pris en compte nativement, elle risque de compromettre l'intégrité transactionnelle de la BD.

Base de données NoSQL

les données sont représentées sous forme de collections de paires clé-valeur, de documents, de graphes, etc. Elles ne possèdent pas de définitions de schéma standard.

L'augmentation de la charge est gérée plutôt par l'ajout de serveurs supplémentaires : Scalabilité (mise à l'échelle) horizontale.

L'augmentation de la charge est automatique, si un serveur tombe en panne, il se remplace automatiquement par un autre serveur sans interruption du service.

01 – Introduction aux Bases de données NoSQL SQL ou NoSQL?



Comparaison entre les bases de données traditionnelles et NoSQL

Base de données SQL

Assure l'intégrité des données en assurant la conformité ACID (Atomicité, Cohérence, Isolation et Durabilité)

Recommandée par de nombreuses entreprises en raison de sa structure et de ses schémas prédéfinis.

Mais, ne convient pas au stockage de données hiérarchiques

La plus appropriée pour les applications transactionnelles à usage intensif étant plus stable et assurant l'atomicité, l'intégrité et la cohérence des données.

Base de données NoSQL

Repose sur les propriétés BASE (Basically Available, Soft state, Eventually Consistent) (voir le slide 23)

Recommandée pour les données semi structurées ou même non structurées

Hautement préférée pour les ensembles de données volumineux et hiérarchiques

01 – Introduction aux Bases de données NoSQL

SQL ou NoSQL?



Résumons

NoSQL

Offre des meilleures performances que **SQL** vu qu'il ne gère aucune règle de cohérence

Optimisé pour gérer d'énormes volumes de données avec performance

SQL

Basé sur le langage de requête unifié (**SQL**) qui apporte une certaine uniformité entre les différentes bases **SQL**,

Offre une meilleure fiabilité et cohérence des données au détriment de la performance si les données deviennent volumineuses



Certes **NoSQL** et **SQL** permettent de stocker/rechercher de l'information, mais ils ne servent pas les mêmes objectifs ce qui rend toute comparaison subjective voir non justifiée.

CHAPITRE 1

Découvrir les bases de données NoSQL

1. Définir le concept de bases de données NoSQL,
2. Comparer les bases de données traditionnelles et NoSQL,
- 3. Recenser les caractéristiques des bases de données NoSQL**
4. Recenser les types de bases de données NoSQL (document, clé / valeur, colonne, graphe)
5. Comparer les différents types de bases de données NoSQL

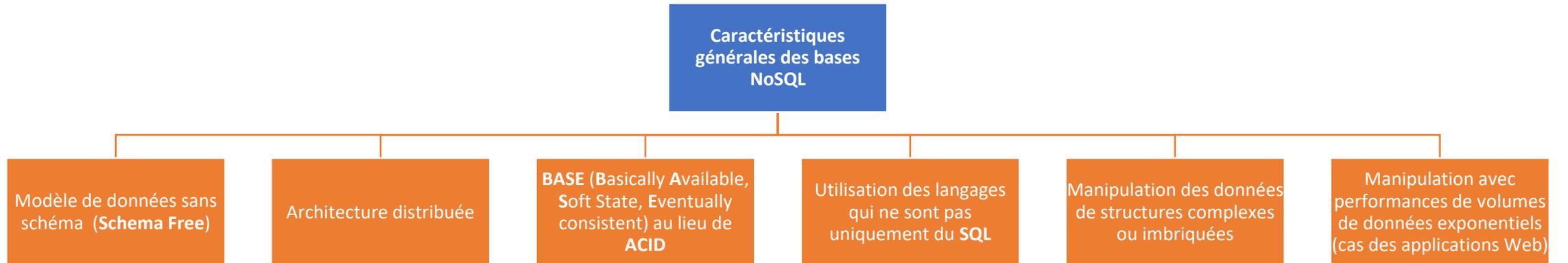


01 – Introduction aux Bases de données NoSQL

Caractéristiques



Caractéristiques générales des bases NoSQL



01 – Introduction aux Bases de données NoSQL

Caractéristiques



1- Modèle Sans schéma (Schema Free)

- Dans un contexte relationnel, la création d'une base de données commence par la modélisation des entités et associations puis d'en déduire un schéma de la base,
- Cette démarche crée une rigidité dans la phase d'implémentation, puisqu'elle implique d'avoir une vision assez claire des évolutions de l'application dès le départ et au fil du temps, ce qui n'est pas souvent le cas de nos jours !!
- Les bases de données **NoSQL** s'appuient sur des données dénormalisées, non modélisées par des relations, mais plutôt par des enregistrements (ou documents) intégrés, il est donc possible d'interagir sans utiliser de langages de requêtes complexe.

01 – Introduction aux Bases de données NoSQL

Caractéristiques

1- Modèle Sans schéma (Schema Free)

- Exemple

SQL

Posts(id,titre)
Commentaires(id, #idPosts,texte)

Posts

Id	titre
P1	Titre1
P2	Titre2

Commentaires

Id	idPosts	texte
C1	P1	comment1
C2	P2	comment2
C3	P1	comment3



NoSQL

Posts(id,titre,commentaires)

Posts

P1	Titre1	Comment1	Comment3
P2	Titre2	Comment2	

2- Architecture distribuée

- Le volume de données à stocker ainsi que les traitements demandés par les organismes modernes, ne peuvent plus être satisfaits sur une seule machine quelque soit sa performance, même en utilisant un réseau de machine l'interconnexion entre machines rendent les traitements très lents,



Solution : un patron d'architecture propose de distribuer les traitements (**le travail/la charge**) sur plusieurs machine puis regrouper les résultats de chaque machine et les agrège dans un résultat final → Apparition de MapReduce en 2003,

- **MAIS**, les bases de données traditionnelles ne permettent pas l'implantation d'un tel patron d'architecture,
- Les bases **NoSQL** sont conçues pour distribuer les données et traitements associés sur de multiple nœuds(serveurs) →partitionnement horizontal,



Problème : impossible d'avoir en même temps une disponibilité des données satisfaisante, une tolérance au partitionnement et une meilleure cohérence des données,

- Il faut toujours condamné un aspect en faveur des autres !

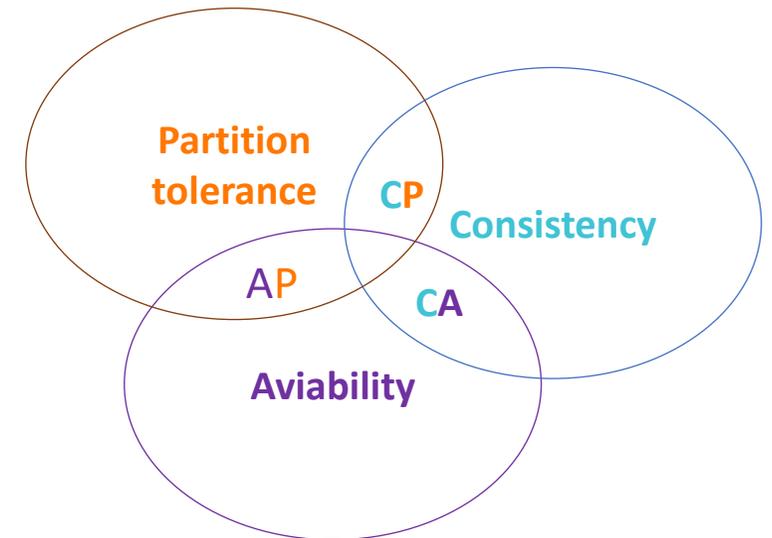
2- Architecture distribuée

Théorème de CAP (Consistency, Availability, Partition tolerance)

Dans toute base de données, on ne peut respecter au plus que deux propriétés parmi les trois propriétés suivantes: la cohérence, la disponibilité et la distribution

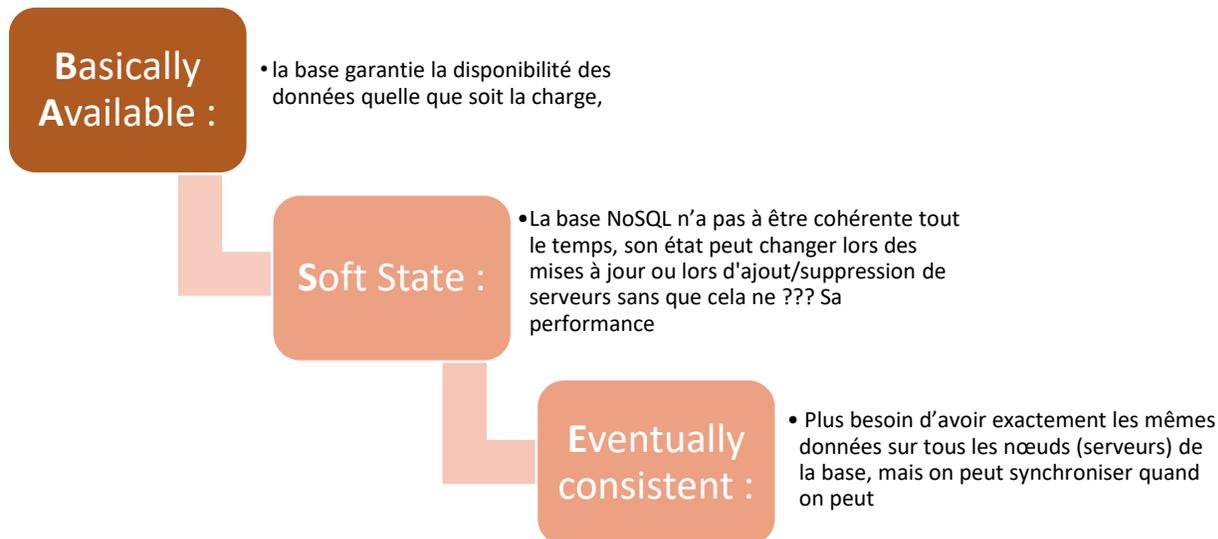
A savoir :

- **Consistency** (cohérence): tous les nœuds(serveurs) sont à jour sur les données au même moment,
- **Availability** (disponibilité): la perte d'un nœud(serveur) n'empêche pas le système de fonctionner et de servir l'intégralité des données,
- **Partition tolerance** (résistance au partitionnement): chaque nœud(serveur) doit pouvoir fonctionner de manière autonome,



3- BASE vs ACID

- Les propriétés **ACID** ne sont pas partiellement ou totalement applicables dans un contexte **NoSQL**,
- Les bases **NoSQL** reposent, par contre, sur les propriétés **BASE**:



Les bases **NoSQL** privilégient la disponibilité à la cohérence : **AP** (Availability + Partition tolerance) plutôt que **CP** (Consistency + Partition tolerance)

CHAPITRE 1

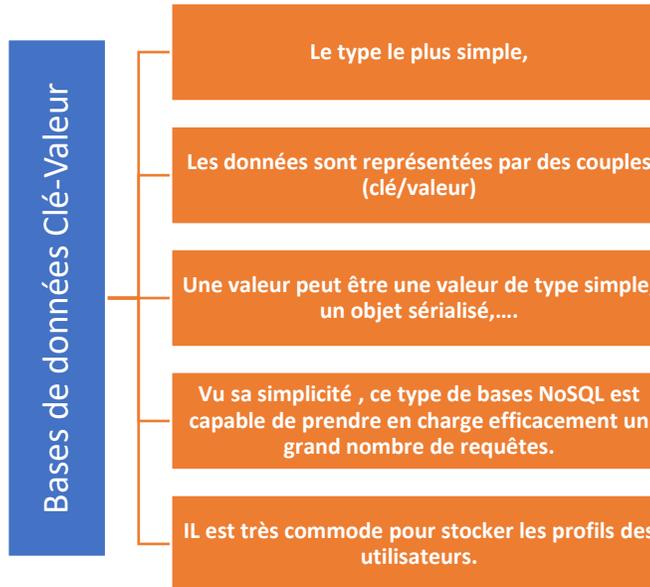
Découvrir les bases de données NoSQL

1. Définir le concept de bases de données NoSQL,
2. Comparer les bases de données traditionnelles et NoSQL,
3. Recenser les caractéristiques des bases de données NoSQL
- 4. Recenser les types de bases de données NoSQL (document, clé / valeur, colonne, graphe)**
5. Comparer les différents types de bases de données NoSQL



Les quatre types des bases NoSQL

1. Bases de données Clé-Valeur



Exemple :

Clé	Valeur
Ahmed	type: Formateur; spec: Dev digital; modules : M102, M104, M106, M203
Sanaa	type: Stagiaire; filière: Dev digital; groupe : DD203; niveau : 2A
Kamal	type: Stagiaire; filière: Infra digitale; niveau : 1A

01 – Introduction aux Bases de données NoSQL

Types des bases NoSQL

Les quatre types des bases NoSQL

1. Bases de données Clé-Valeur

Exemples de bases de données Clé/valeur

Dynamo DB
Amazon



Berkeley DB ou **BDB** solution d'oracle
GMAIL, RPM,SVN,...



Voldemort de LinkedIn
(et pas le sorcier de Harry Potter 😊)



Riak DB
Apache



Les quatre types des bases NoSQL

2. Bases de données orientées Document

Bases de données orientées document

Evolution du type clé/valeur

La valeur ici est un document dont la structure reste libre,

Les documents sont stockés sous forme de fichiers JSON ou XML,

Ce type possède l'avantage d'éviter les jointures pour reconstruire une information puisque tout est compris dans la structure des documents.

Exemples

Clé

Ahmed

Sanaa

Sanaa

Valeur

```
{ "type": "Formateur",  
  "spec": "Dev digital",  
  "modules": ["M102",  
              "M104", "M106",  
              "M203"] }
```

```
{ "type": "Stagiaire",  
  "filiere": "Dev digital",  
  "groupe": "DD203",  
  "niveau": "2A",  
  "option": "Mobile" }
```

```
{ "type": "Stagiaire",  
  "filiere": "Infra digital",  
  "niveau": "1A" }
```

01 – Introduction aux Bases de données NoSQL

Types des bases NoSQL

Les quatre types des bases NoSQL

2. Bases de données orientées Document

Exemples de bases de données orientées Document

Mongo DB de SourceForge
Adobe, Bosch, Cisco, eBay,...



CouchDB d'Apache
Disney, PayPal, Ryanair,....



RavenDB
Plateformes .Net/Windows



Cassandra de FaceBook
NY Times, eBay, Sky, Pearson Education



Les quatre types des bases NoSQL

3. Bases de données orientées Colonne

Bases de données orientées Colonne

Ce type change le paradigme traditionnel de la représentation des données en lignes,

Il rend possible de focaliser les requêtes sur les colonnes importantes sans avoir à traiter les données des autres colonnes (jugées alors inutile pour la requête),

Ce type est adapté aux systèmes avec de gros calculs analytiques (comptage, moyenne, somme,...)

Les quatre types des bases NoSQL

3. Bases de données orientées Colonne

Exemple

- Représentation traditionnelle (représentation en ligne)

Id	Type	Spécialité	Niveau	Filière	Groupe	Option	Module
Ahmed	Formateur	Dev Digital					M102, M104, M106, M202
Sanaa	Stagiaire		2A	Dev Digital	DD203	Mobile	
Kamal	Stagiaire		1A	Infra Digitale			
Laila	Formateur	Infra Digitale					M105,M107,M201

01 – Introduction aux Bases de données NoSQL

Types des bases NoSQL



Les quatre types des bases NoSQL

3. Bases de données orientées Colonne

Exemple

Id	Type	Spécialité	Niveau	Filière	Groupe	Option	Module
Ahmed	Formateur	Dev Digital					M102, M104, M106, M202
Sanaa	Stagiaire		2A	Dev Digital	DD203	Mobile	
Kamal	Stagiaire		1A	Infra Digitale			
Laila	Formateur	Infra Digitale					M105, M107, M201

- Exemples de représentations par colonnes

Id	Type
Ahmed	Formateur
Sanaa	Stagiaire
Kamal	Stagiaire
Laila	Formateur

Id	Filière
Sanaa	Stagiaire
Kamal	Stagiaire

Id	Module
Ahmed	M102
Ahmed	M104
Ahmed	M106
Ahmed	M202
Laila	M105
Laila	M107
Laila	M201

Id	Option
Sanaa	Mobile

01 – Introduction aux Bases de données NoSQL

Types des bases NoSQL

Les quatre types des bases NoSQL

3. Bases de données orientées Colonnes

Exemples de bases de données orientées Colonnes

BigTable DB de Google



HBase d'Apache



SparkSQL d'Apache



Elasticsearch db



Les quatre types des bases NoSQL

4. Bases de données orientées Graphe

Bases de données orientées Graphe

Il est basé sur la théorie des graphes et ses fondements mathématiques,

Ce type est conçu principalement pour les données fortement interconnectées (comme les données des réseaux sociaux) avec un nombre indéterminé de relations entre elles,

En d'autres termes, c'est le type le mieux approprié pour modéliser le monde réel

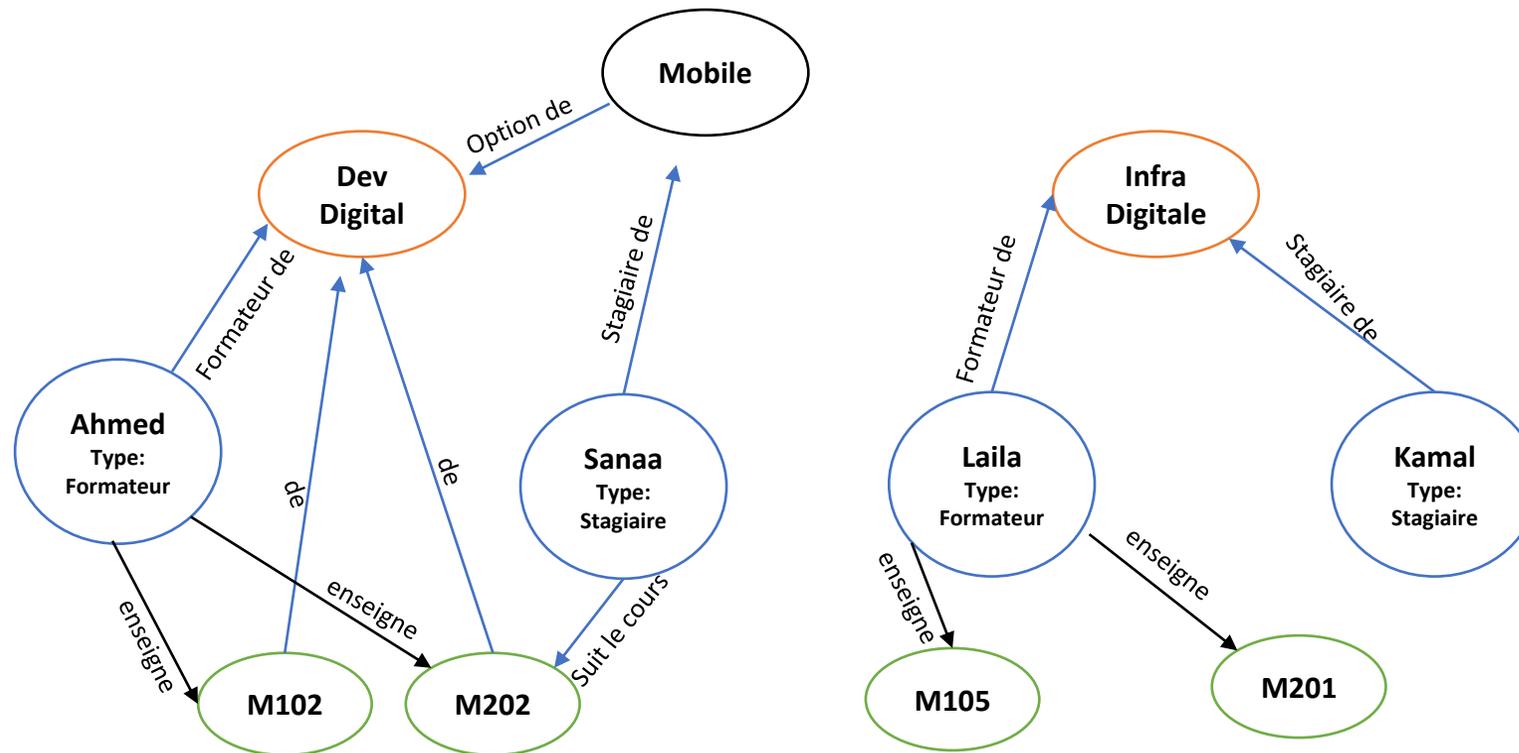
Exemple : la principale solution est Neo4j



Les quatre types des bases NoSQL

3. Bases de données orientées Graphe

Exemple



CHAPITRE 1

Découvrir les bases de données NoSQL

1. Définir le concept de bases de données NoSQL,
2. Comparer les bases de données traditionnelles et NoSQL,
3. Recenser les caractéristiques des bases de données NoSQL
4. Recenser les types de bases de données NoSQL (document, clé / valeur, colonne, graphe)
5. **Comparer les différents types de bases de données NoSQL**



01 – Introduction aux Bases de données NoSQL

Comparaison des types de bases NoSQL



	Riak (Clé/Valeur)	MongoDB (Document)	Cassandra (Document)	HBase (Colonne)
Cout	+	++	++	++
Cohérence	+	++	+	+
Disponibilité	++	+	++	++
Langages d'Interrogation	++	++	+	++
Fonctionnalités	Solution hautement disponible avec un langage de requêtes performant Approprié au stockage dans le cloud	La solution la plus populaire, structure souple et bonnes performances Favorise la cohérence à la disponibilité	solution mature, populaire, Excellente solution pour grands volumes de données besoins de bases distribuées Mais langage trop réduit	Destinée aux données volumineuses, Privilégie le langage et la disponibilité à la cohérence des données



Pourquoi choisir MongoDB

MongoDB vient du terme **Humongous DB** qui veut dire une base de données **gigantesque**

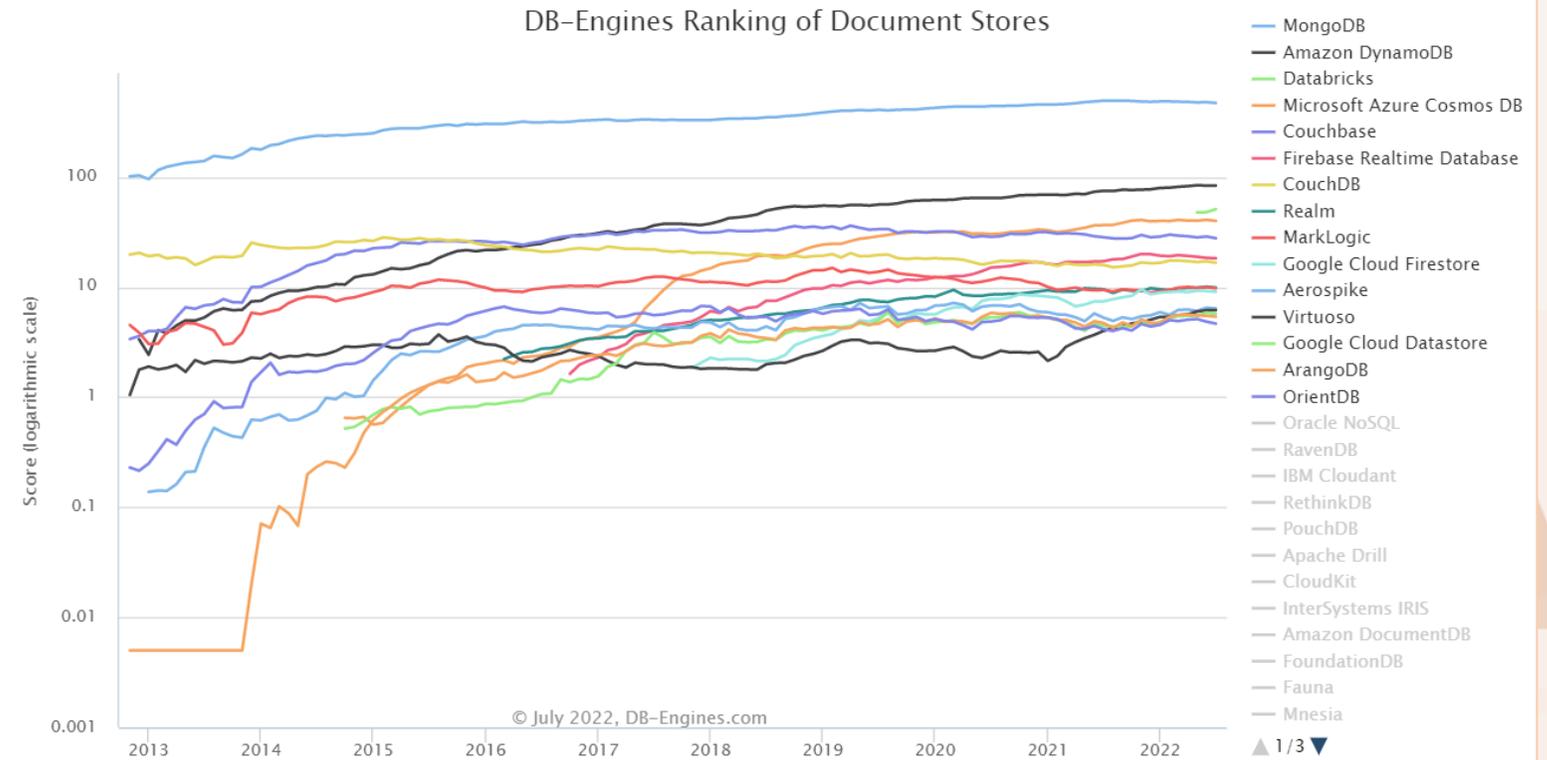
C'est la base **NoSQL** la plus populaire grâce aux points forts suivants :



Pourquoi choisir MongoDB

- La popularité de **MongoDB** vient du fait qu'elle est fortement utilisée par les développeurs partout dans le monde,
- Elle est appréciée également d'être facilement intégrable dans toute application gérant des documents/objets.

DB-Engines Ranking - Trend of Document Stores Popularity



https://db-engines.com/en/ranking_trend/document+store



CHAPITRE 2

Mettre en place une base de données MongoDB

Ce que vous allez apprendre dans ce chapitre :

- Installer un serveur MongoDB ,
- Administrer le serveur,
- Créer une base de données
- Créer une collection de documents,



07,5 heures

CHAPITRE

Mettre en place une base de données MongoDB

1. **Installer un serveur MongoDB ,**
2. Administrer le serveur,
3. Créer une base de données
4. Créer une collection de documents,



Installation et configuration du serveur

Etapes d'installation



Installation du serveur

- Sous Windows:
 - Télécharger le serveur depuis l'adresse suivante :
<https://www.mongodb.com/try?jmp=nav#community>
 - Installer le serveur en appuyant sur suivant
 - Une fois installé créer un dossier pour stocker les données (documents)
 - Lancer le serveur avec le service ***mongod.exe***

Disque local (C:) > Programmes > MongoDB > Server > 6.0 > bin

Nom	Modifié le	Type	Taille
InstallCompass.ps1	05/07/2022 18:40	Script Windows Po...	2 Ko
mongod.cfg	21/07/2022 01:16	Fichier source Con...	1 Ko
mongod.exe	05/07/2022 20:15	Application	54 578 Ko
mongod.pdb	05/07/2022 20:15	Program Debug D...	812 588 Ko
mongos.exe	05/07/2022 20:22	Application	33 764 Ko
mongos.pdb	05/07/2022 20:22	Program Debug D...	453 508 Ko

Available Downloads

Version

6.0.0 (current)

Platform

Windows

Package

msi



Download

Copy Link

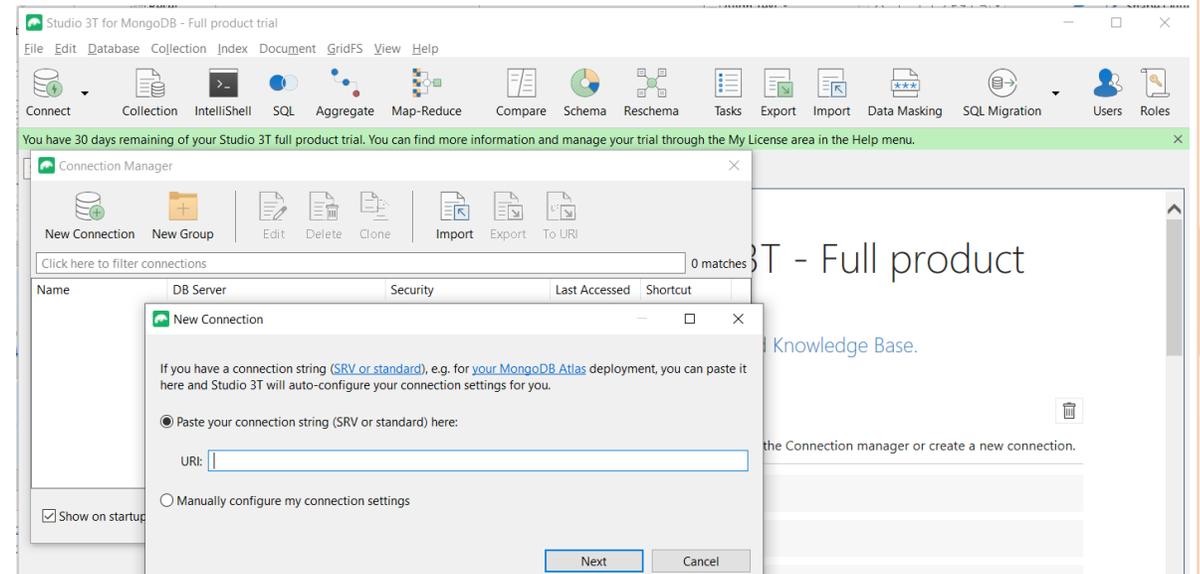
Installation et configuration du serveur

Etapes d'installation



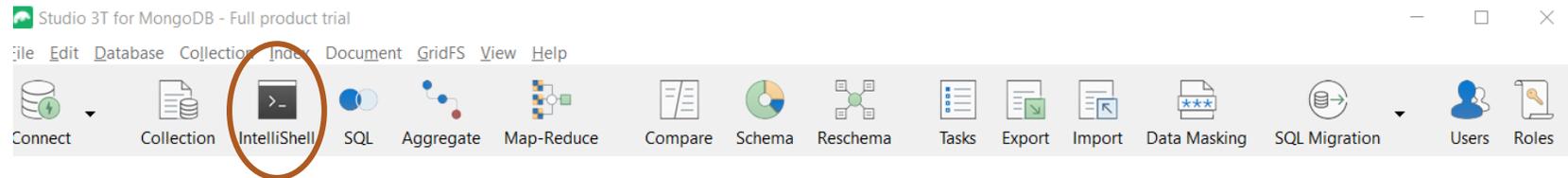
Installation de l'interface utilisateur

- Sous Windows:
 - Télécharger l'interface utilisateur **Studio 3T** et installer le en suivant les instructions,
 - Enregistrer le produit pour l'activer
 - Tester bien si le service MongoDB est en cours d'exécution sur le port 27017
 - Créer une première connexion



Les commandes possibles sur les bases de données

- Après avoir créer la connexion, lancer l'ILE **IntelliShell** de Studio 3T

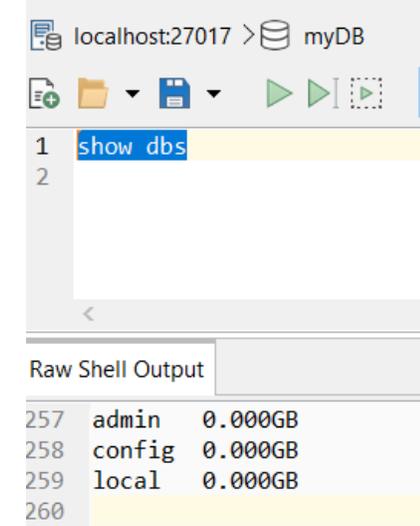


- Exécuter la commande : **db.help()**

```
Quickstart x IntelliShell: connexion x
localhost27017 > mabase
1 db.help()
2
Raw Shell Output
203 DB methods:
204 db.adminCommand(nameOrCommand) - switches to 'admin' db, and runs command [just calls db.runCommand(...)]
205 db.aggregate([pipeline], {options}) - performs a collectionless aggregation on this database; returns a cursor
206 db.auth(username, password)
207 db.cloneDatabase(fromhost) - will only function with MongoDB 4.0 and below
208 db.commandHelp(name) returns the help for the command
209 db.copyDatabase(fromdb, todb, fromhost) - will only function with MongoDB 4.0 and below
210 db.createCollection(name, {size: ..., capped: ..., max: ...})
211 db.createUser(userDocument)
212 db.createView(name, viewOn, [{operator: {...}}, ...], {viewOptions})
213 db.currentOp() displays currently executing operations in the db
214 db.dropDatabase(writeConcern)
215 db.dropUser(username)
216 db.eval() - deprecated
217 db.fsyncLock() flush data to disk and lock server for backups
218 db.fsyncUnlock() unlocks server following a db.fsyncLock()
219 db.getCollection(cname) same as db['cname'] or db.cname
220 db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections
221 db.getCollectionNames()
222 db.getLastErrorMessage() - just returns the err msg string
223 db.getLastMongoDb() - return full status object
224 db.getLogComponents()
225 db.getMongo() get the server connection object
226 db.getMongo().setSecondaryOk() allow queries on a replication secondary server
227 db.getName()
228 db.getProfilingLevel() - deprecated
```

Les bases de données existantes sur un cluster

- Exécuter la commande : **show dbs**
 - Admin
 - Config
 - et Local
- Ces trois bases de données font partie de tout serveur MongoDB



```
localhost:27017 > myDB
1 show dbs
2
Raw Shell Output
257 admin 0.000GB
258 config 0.000GB
259 local 0.000GB
260
```

Création d'une base de données

- Il faut noter qu'il n'y a pas de commande « create » dans le shell MongoDB,
- Pour créer une base de données , on utilise la commande **use** suivie du nom de la base à créer
- La commande use, en fait, permet d'utiliser une base de données existante ou de la créer si elle n'existe pas déjà dans le serveur:

use myDB

- En d'autres termes, la base de données **myDB** est créée si on essaie de l'utiliser,
- Même si la commande de création de la base **myDB** s'est bien déroulée, mais elle n'est pas entièrement créée tant qu'elle est vide.

show dbs

```
switched to db myDB
admin 0.000GB
config 0.000GB
local 0.000GB
```

Consultation

- Une collection pour une base de données MongoDB est l'équivalent d'une table pour une base de données relationnelle, sauf que la collection n'a pas de schéma (voir après)
- Une base de données MongoDB, gère une collection de documents et non des tables,
- Pour lister les collections d'une base de données, on exécute la commande

```
show collections
```

Exemple:

<pre>use admin show collections</pre>	<pre>switched to db admin system.version</pre>
<pre>use local show collections</pre>	<pre>switched to db local startup_log</pre>
<pre>use config show collections</pre>	<pre>switched to db config system.sessions</pre>

admin contient la collection ***system.version***
local contient la collection ***startup_log***
Et **config** contient la collection ***system.sessions***

Création des bases de données

Suppression d'une collection ou d'une base de données



Suppression d'une collection ou d'une base de données

- Pour supprimer une collection d'une base de données, on exécute la commande

```
db.nomcollection.drop()
```

- Pour supprimer la base de données courante, on exécute la commande

```
db.dropDatabase()
```

Document

- Les documents sont les unités de base dans une base MongoDB,
- Ils sont l'équivalent des enregistrements des tables dans une base de données relationnelle,
- Ils sont représentables (dans la majorité des cas) sous forme d'objets JSON
- Tout document appartient à une collection et a un champ appelé `_id` qui l'identifie dans la base de données,
- Exemple:

```
{  
  "nom" : "Sanaa",  
  "filier" : "Dev Digital",  
  "niveau" : "2A",  
  "option": "Mobile"  
}
```

Collection

- Une collection est un ensemble de documents,
- On vient de voir que c'est l'équivalent d'une table en relationnel,
- Contrairement aux bases de données relationnelles, les collections n'ont pas de schéma spécifique que les documents doivent respecter,
- Les champs des documents d'une collection sont libres et peuvent être différents d'un document à un autre. Le seul champ commun est obligatoire est le "_id".
- Néanmoins pour que la base soit maintenable, il est préférable d'avoir dans une collection des documents de même type

Collection

Exemples **(ces exemples à revoir)**

Collection avec des documents de même schéma

```
{
  name: "al",
  age: 18,
  status: "D",
  groups: [ "politics", "news" ]
}
```

Collection avec des documents de schémas différents

```
{
  "_id" : ObjectId("54ce14c9dc4d5f155a2332ff"),
  "nom" : "Training Day",
  "acteurs" : [
    {
      "nom" : "Washington",
      "prenom" : "Denzel"
    }
  ]
}

{
  "_id" : ObjectId("54ce14c9dc4d5f155a233300"),
  "nom" : "Iron Man",
  "acteurs" : [
    {
      "nom" : "Downey"
    }
  ]
}
```

Exemple de création d'une collection dans une base de données

- Soit la base de données **myDB** qu'on a déjà créer,
- Pour créer une collection **myCollection** dans la base **myDB**, il suffit d'ajouter un (ou plusieurs) documents à la collection dans la base de données:

- **Utiliser la base de données**

```
use myDB
```

- **Ajouter un document (simple objet JSON dans ce cas) à la collection**

```
db.myCollection.insert(  
  { "id" : "Sanaa",  
    "filierre" : "Dev Digital",  
    "niveau" : "2A",  
    "option":"Mobile"  
  } )
```

Exemple de création d'une collection dans une base de données

- Afficher les bases de données du serveur

Show dbs

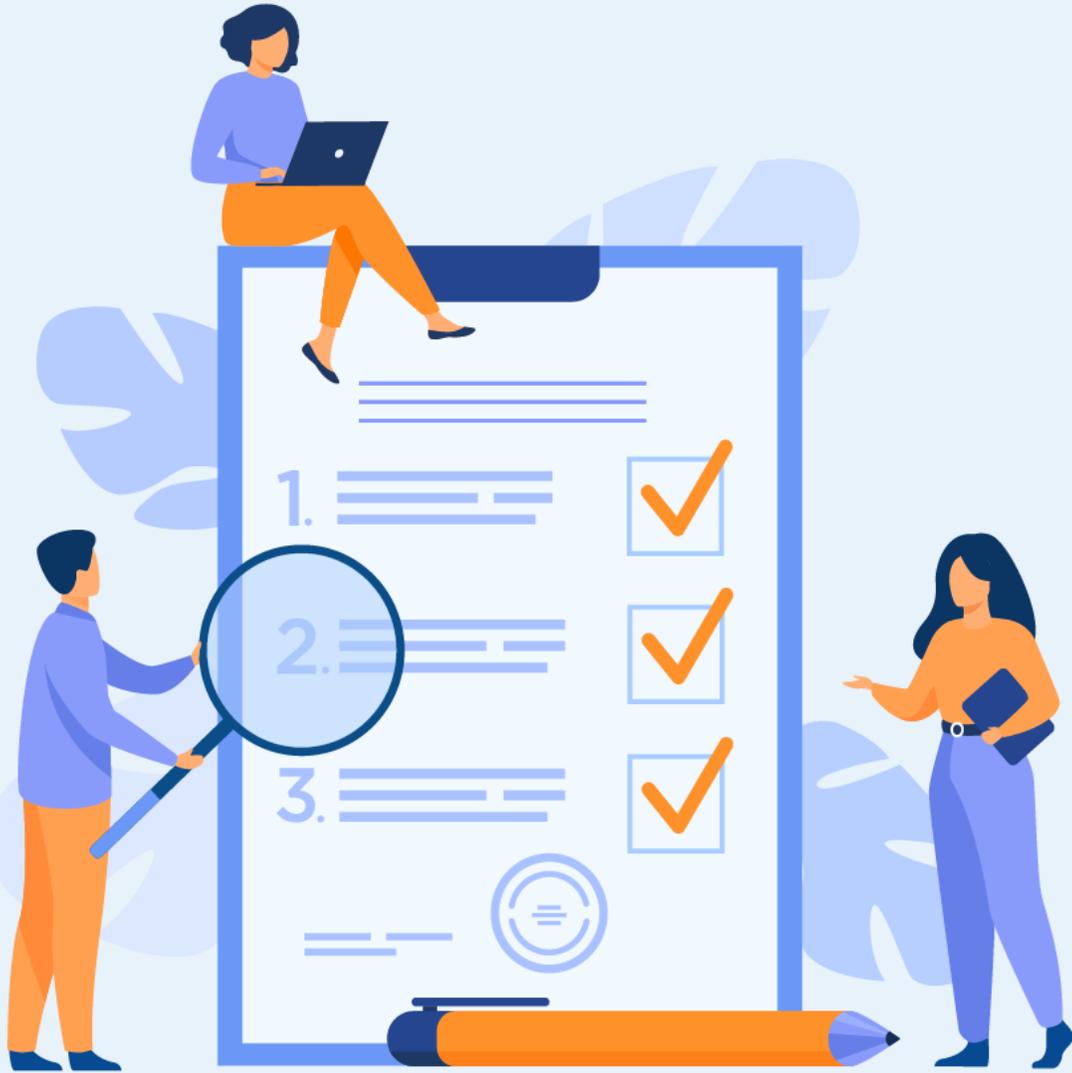
- On doit avoir ce résultat

```
Bulk/this.insert@src/mongo/shell/bulk_api.js:650:20
DBCollection.prototype.insert@src/mongo/shell/collection.js:313:13
@(shell):1:1

admin 0.000GB
config 0.000GB
local 0.000GB
switched to db myDB
WriteResult({ "nInserted" : 1 })
admin 0.000GB
config 0.000GB
local 0.000GB
myDB 0.000GB
```

Document inséré dans la collection →

La base de données est effectivement créée →



CHAPITRE 3

Modéliser les documents

Ce que vous allez apprendre dans ce chapitre :

- Structurer un document JSON,
- Recenser les différences entre modéliser pour MongoDB versus une base de données relationnelles,
- Modéliser les liens,
- Utiliser des espaces de noms, des collections et des documents,

 07,5 heures

CHAPITRE 3

Modéliser les documents

1. **Structurer un document JSON,**
2. Identifier la différences entre la modélisation pour MongoDB versus une base de données relationnelles,
3. Modéliser les liens,
4. Utiliser des espaces de noms, des collections et des documents



01 – Structure d'un document JSON

Définition



JSON

JSON (JavaScript Object Notation) est un format standard de représentation logique de données, hérité de la syntaxe de création d'objets en JavaScript

Utilisé pour structurer et transmettre des données sur des sites web (par exemple, envoyer des données depuis un serveur vers un client ou vice versa)

C'est un format réputé texte léger (pas trop de caractères de structuration), lisible par les humains avec l'extension **.json**

Bien que JSON puise sa syntaxe du JavaScript, il est indépendant de tout langage de programmation. Il peut ainsi être interprété par tout langage à l'aide d'un parser

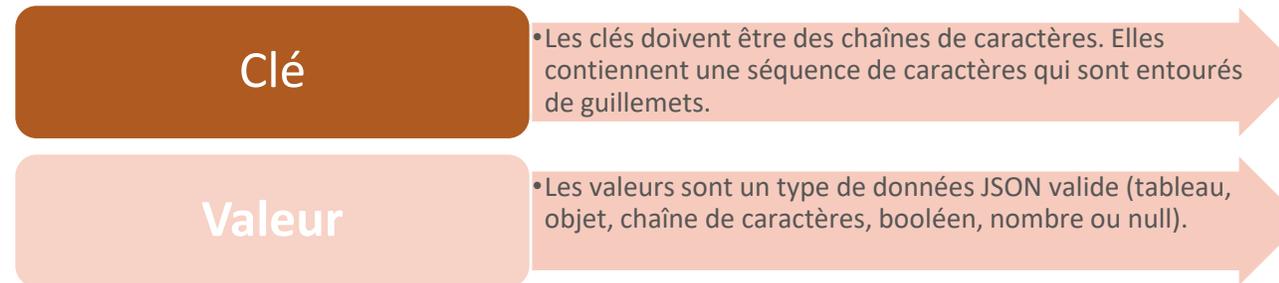
01 – Structure d'un document JSON

Syntaxe de base



Objet JSON

- Un objet **JSON** se base sur deux éléments essentiels : **Les clés** et **les valeurs**.



- Un objet **JSON** commence et se termine par des accolades {}.
- Il peut contenir plusieurs paires **clé/valeur**, séparées par une virgule., La clé est suivie de « : » pour la distinguer de la valeur.

01 – Structure d'un document JSON

Syntaxe de base



Types de valeurs JSON

- **JSON** supporte en principe trois types de valeurs
 - **Primitif** : nombre, booléen, chaîne de caractères, null,
 - **Objet** : liste de paires "clé": valeur entrés entre accolades, séparés par des virgules.

Exemple :

```
"stagiaire" :{"prenom":"Amina", "filiere":"Dev Digital ", "niveau":"1A" }
```

- **Tableau (Array)**: ensemble ordonné de valeurs, entouré de crochets [] ces valeurs sont séparées par une virgule,

Exemple:

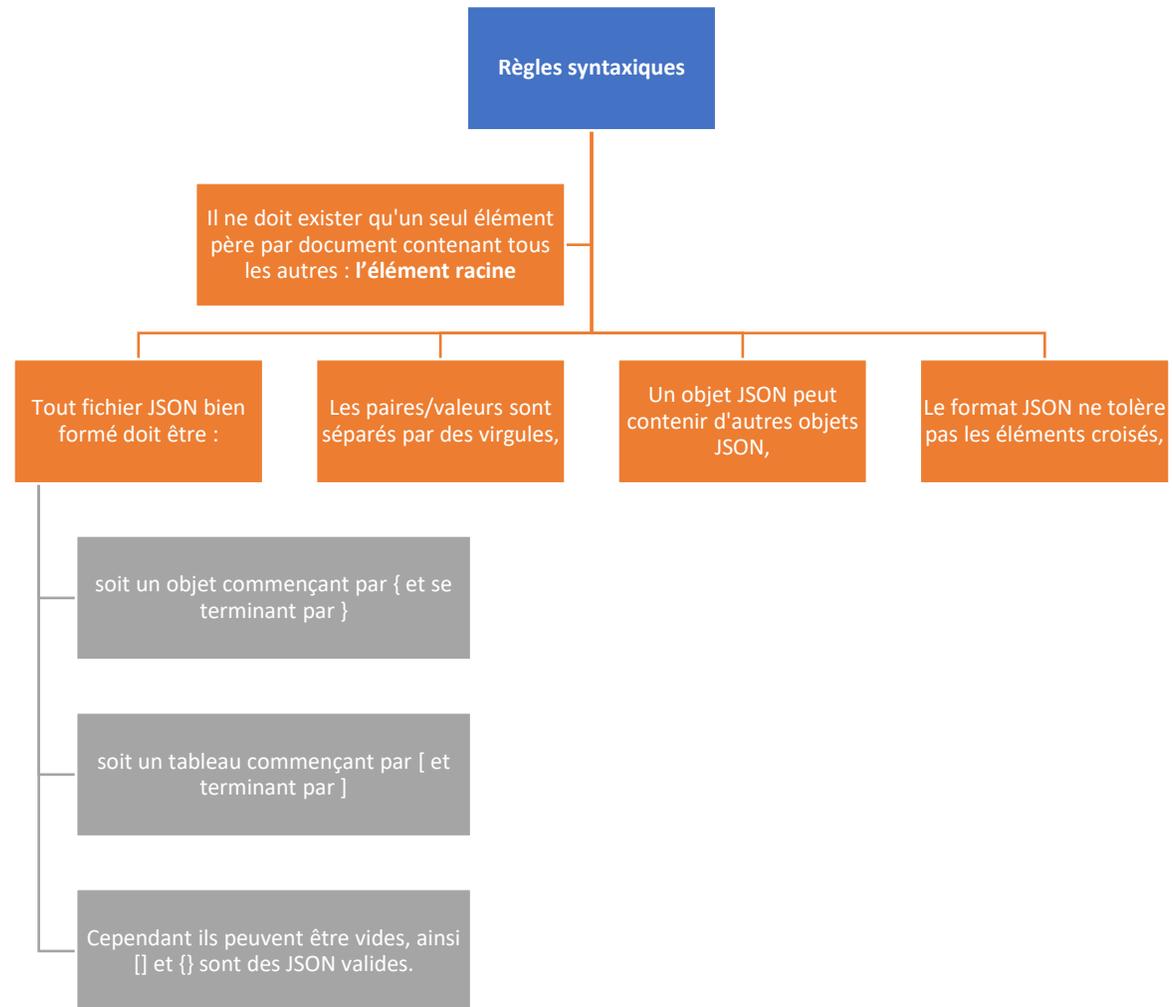
```
"stagiaires" :[  
  {"prenom":"Amina", "filiere":"Dev Digital ", "niveau":"1A" },  
  {"prenom":"Kamal", "filiere":"Infra Digitale ", "niveau":"1A" },  
  {"prenom":"Sanaa", "filiere":"Infra Digitale ", "niveau":"1A" }  
]
```

01 – Structure d'un document JSON

Syntaxe de base



Règles syntaxiques



01 – Structure d'un document JSON

Exemple



Exemple de tableau JSON

```
{
  [
    { _id: 12,
      "prenom": "Kamal",
      "type": "Stagiaire",
      "filier": "Dev Digital",
      "groupe": "DD203",
      "niveau": "2A",
      "option": "Mobile"},
    {
      _id: 17,
      "prenom": "Amina",
      "filier": "Infra Digitale ",
      "niveau": "1A",
    }
  ]
}
```

01 – Structure d'un document JSON

Usages de JSON



Chargements Asynchrones

- Avec la montée en flèche des chargements asynchrones tels que l'**AJAX** le format JSON s'est montré adapté que XML,

APIs

- Des sociétés telles que Twitter, Facebook ou LinkedIn, offrent essentiellement des services basés sur l'échange, d'informations, et font preuve d'un intérêt grandissant envers les moyens possibles pour distribuer ces données à des tiers,
- JSON domine le domaine des APIs au détriment du format XML qui avait été pionnier,

Bases de données

- Très utilisé dans le domaine des bases de données NoSQL (MongoDB, CouchDB, Riak...),
- Il est également, possible de soumettre des requêtes à des SGBDR et de récupérer une réponse en JSON

CHAPITRE 3

Modéliser les documents

1. Structurer un document JSON,
2. **Identifier la différences entre la modélisation pour MongoDB versus une base de données relationnelles,**
3. Modéliser les liens,
4. Utiliser des espaces de noms, des collections et des documents



02 – Modélisation MongoDB vs BDR

Normalisation vs Dénormalisation



Dans le cas des bases de données relationnelles, la modélisation des données repose sur la normalisation des structures de données afin d'éviter toute duplication de l'information,

Une fois les structures des données normalisées, on procède à la création des requêtes se basant essentiellement sur les jointures engendrées par la normalisation,

Les bases de données **NoSQL**, se caractérisent par l'abandon des jointures et transactions au profit d'un temps de réponse court et des performances optimales,



Comment, alors, concevoir un schéma de données approprié à ce contexte afin de mieux interroger les données?



Réponse : La dénormalisation

02 – Modélisation MongoDB vs BDR

Normalisation vs Dénormalisation



Définition et objectif

La **dénormalisation** consiste à regrouper plusieurs tables liées par des références, en une seule table, en éliminant les jointures, elle favorise la redondance des données

Son objectif est d'améliorer les performances de la base de données en recherche sur les tables considérées

Elle vise également à améliorer les performances de la base de données en recherche sur les tables considérées

Dénormaliser consiste à dupliquer les données (ou une partie de données)d'une structure de données dans une autre

02 – Modélisation MongoDB vs BDR

MongoDB et la dénormalisation



- Les bases de données relationnelles s'appuient sur le modèle relationnel,
- **La normalisation** est une contrainte obligatoire pour la validation du modèle des données,
- **La normalisation** au niveau conceptuel impose la structure des données de la base (Modèle conceptuel des données)
- Le passage du Modèle Conceptuel des Données au Modèle Logique des Données détermine la nature des relations entre les données (liaison par clé primaire au niveau de la table et par clé étrangère entre les tables),
- Les requêtes de la manipulation des données (Recherche opérations CRUD) doivent respecter le modèle logique préétabli,
- Le modèle des données impose la manière d'écriture des requêtes sur les données

02 – Modélisation MongoDB vs BDR

MongoDB et la dénormalisation



- La différence majeure au niveau de la modélisation des données entre les bases de données relationnelles et celles orientées **Document** est la dénormalisation des données,
- En effet, les données ne sont pas soumises aux contraintes de normalisation:
 - **Attributs non atomiques** : première forme normale non respectée,
 - **Données redondantes** : deuxième forme normale non respectée,
 - ...
- Dans l'absence totale ou partielle d'un modèle des données, c'est la nature des relations qui exige plutôt le type de requêtes qu'on désire élaborer sur les données,
- On détermine le schéma des données suivant l'utilisation des données par l'application c.-à-d. les requêtes

02 – Modélisation MongoDB vs BDR

Récapitulation

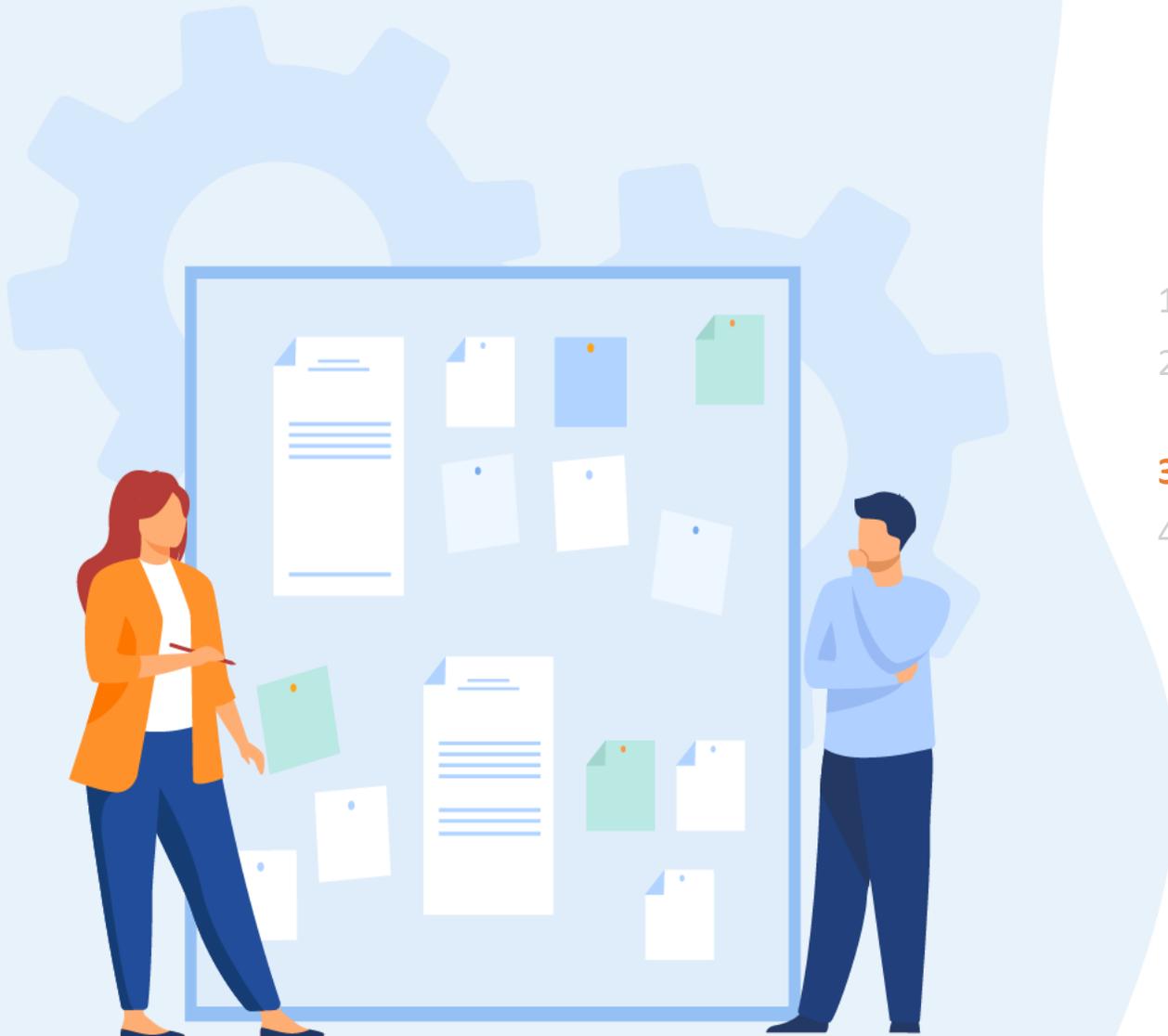


Base de données relationnelle	MongoDB (Base de données orientée documents)
Base de données	Base de données
Table	Collection
Enregistrement	Document
Schéma de données fixe	Schéma de données flexible
Les enregistrements de la table doivent avoir le même ensemble de champs	Les documents d'une même collection n'ont pas besoin d'avoir le même ensemble de champs
Le type de données d'un champ est fixe pour tous les enregistrements de la table	Le type de données d'un champ peut différer d'un document à l'autre d'une collection.
Les requêtes sur les données doivent respecter un modèle des données logique fixe	Le type d'utilisation des données (les requêtes) détermine le schéma des données

CHAPITRE 3

Modéliser les documents

1. Structurer un document JSON,
2. Identifier la différences entre la modélisation pour MongoDB versus une base de données relationnelles,
- 3. Modéliser les liens,**
4. Utiliser des espaces de noms, des collections et des documents



Les types de relations entre les données sous MongoDB

- MongoDB détermine deux types de relations entre les données :
 - **Les relations d'enchâssement (embedding) :**
 - L'imbrication d'un (ou d'une partie) d'un document dans un autre, on parle de document autonome,
 - L'imbrication permet d'éviter de faire des jointures: inutile de faire des jointures pour restituer l'information qui n'est pas dispersée sur plusieurs entités (tables en relationnel),
 - On utilise l'imbrication des documents (**embedding**) quand les documents sont très petits et n'ont pas tendance à grandir dans le futur. La taille des documents ne doit pas dépasser 16Mb,
 - Adéquates pour les contextes qui privilégient la recherche à la mise à jour,

- MongoDB détermine deux types de relations entre les données :
 - **Les relations de liaisons (Linking) :**
 - La duplication de l'identifiant d'un document dans un autre document,
 - Reprend, en quelque sorte, le concept de jointure entre les tables relationnelles,
 - N'est privilégiée que dans le contexte de relations plusieurs-plusieurs:

Exemple : Commande \leftrightarrow Produit

Un produit peut être commandé plusieurs fois et une commande peut contenir plusieurs produits,

Une imbrication des Produits dans la commande aura de gros impacts sur les mises à jour (tous les produits à mettre à jour !)

L'enchassement (embedding)

- Pour une relation entre deux documents A et B, cela consiste à imbriquer partiellement ou totalement le document B dans le document A.
- Ce modèle de relations dénormalisés permet aux applications de récupérer et de manipuler des données associées en une seule opération de base de données,
- **Exemple:**

```
{  
  "_id": "1234",  
  "prenom": "Amina",  
  "filier":  
    { "_id": "DD",  
      "intitule": "Développement digital" }  
}
```

} Document imbriqué

La liaison (Linking)

- Les relations de liaison permettent d'inclure les liens ou des références d'un document dans un autre,
- Ce modèle de relations récupère les données en deux étapes:
 - une première requête pour récupérer l'identifiant,
 - une deuxième requête pour récupérer les données de l'autre côté de la relation.
- **Exemple:**

```
{"_id": "DD", "intitule": "développement digital"}  
{  
  "_id": "1234",  
  "prenom": "Amina",  
  "filiere": "DD"  
}
```

Enchâssement vs liaison

- On utilise l'imbrication des documents (**embedding**) quand les documents sont très petits et n'ont pas tendance à grandir dans le futur. La taille des documents ne doit pas dépasser **16Mb**,
- Si la taille de la collection ou les documents va augmenter dans le futur, il vaut mieux opter pour la liaison des documents.

CHAPITRE 3

Modéliser les documents

1. Structurer un document JSON,
2. Identifier la différences entre la modélisation pour MongoDB versus une base de données relationnelles,
3. Modéliser les liens,
4. **Utiliser des espaces de noms, des collections et des documents**



Définition

- Les bases de données sont des groupes de collections stockées sur le disque à l'aide d'un seul ensemble de fichiers de données,
- Un espace de nom *namespace* est la concaténation du nom de la base de données et des noms de collection, séparés par un point,

Exemple:

`myDB.Stagiaires` → le nom de la base de données(myDB) suivi du nom de la collection

- Les collections sont des conteneurs pour les documents

Définition

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:

- **String** : chaîne de caractères, les strings BSON sont en UTF-8,

Exemple :

```
{"_id":"1234", "prenom":"Amina"}
```

- **Integer**: entier qui peut être stocker le type de données entier sous deux formes : entier signé 32 bits et entier signé 64 bits.

Exemple :

```
{"_id":"1234", "prenom":"Amina", "age": 19}
```

Types de données

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:

- **Double** : utilisé pour stocker les valeurs à virgule flottante. ,

Exemple :

```
{"_id":"1234","prenom":"Amina","moyBaccalaureat":14.25}
```

- **Boolean**: utilisé pour stocker vrai ou faux

Exemple:

```
{"_id":"1234","prenom":"Amina","moyBaccalaureat":14.25,"admis":true}
```



Types de données

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:
 - **Date** :
 - stocke la date sous forme de millisecondes (entier 64bits),
 - Le type de données BSON prend généralement en charge la date et l'heure UTC et il est signé, les valeurs négatives représentent les dates antérieures à 1970,
 - La date peut être exprimée sous forme de string : **Date()** ou d'objet date **new Date()**

Types de données

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:
 - **Date :**

Exemple

```
{
  "prenom": "Ahmed",
  "niveau": "1A",
  "filier": "Dev digital",
  "DateInscription_1": Date(),
  "DateInscription_2": new Date()
}
```

Vue JSON

```
{
  "_id" : ObjectId("62dd1dfff1a19f3d7ecc66252"),
  "prenom" : "Ahmed",
  "niveau" : "1A",
  "filier" : "Dev digital",
  "DateInscription_1" : "Sun Jul 24 2022 11:25:03 GMT+0100",
  "DateInscription_2" : ISODate("2022-07-24T10:25:03.613+0000")
}
```

Types de données

- Dans **MongoDB**, les documents sont stockés dans BSON, le format codé binaire de JSON, elle prend en charge divers types de données à savoir:
 - **Null** : utilisé pour stocker la valeur null,

Exemple :

```
{  
  "_id": "1234",  
  "prenom": "Amina",  
  "telephone": null  
}
```

Types de données

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:
 - **Données binaires** : utilisé pour stocker les données binaires,

Exemple :

```
{  
  "_id": "1234",  
  "prenom": "Amina",  
  "telephone": null,  
  "BinaryValues": "10010001",  
}
```

Types de données

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:
 - **Array**: Ensemble des valeurs pouvant être de types de données identiques ou différentes. Dans **MongoDB**, le array est créé à l'aide de crochets ([]).

Exemple:

```
{
  "_id": "1234",
  "prenom": "Kamal",
  "niveau": "2A",
  "option": "Mobile",
  "skills": [
    "python",
    "javascript",
    "php" ]
}
```

Types de données

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:
 - **ObjectId (Id d'objet)**: pour chaque nouveau document crée dans une collection, MongoDB crée automatiquement un identifiant d'objet unique `_id` s'il n'est pas crée explicitement,

Exemple: Soient les deux documents suivants:

```
{"prenom":"Ahmed","niveau":"1A","filiere":"Dev digital"}  
{"_id":1234,"prenom":"Alaa","niveau":"2A","filiere":"Dev digital"}
```

Vue table

Stagiaire > filiere			
_id	prenom	niveau	filiere
1234.0	Alaa	2A	Dev digital
62dd137d1a19f3d7ecc66250	Ahmed	1A	Dev digital

Types de données

- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:
 - **ObjectId (Id d'objet)**: pour chaque nouveau document crée dans une collection, MongoDB crée automatiquement un identifiant d'objet unique `_id` s'il n'est pas crée explicitement,

Exemple: Soient les deux documents suivants:

```
{"prenom":"Ahmed","niveau":"1A","filier":"Dev digital"}  
{"_id":1234,"prenom":"Alaa","niveau":"2A","filier":"Dev digital"}
```

Vue JSON

```
{  
  "_id" :  
  ObjectId("62dd128a1a19f3d7ecc6624f"),  
  "prenom" : "Ahmed",  
  "niveau" : "1A",  
  "filier" : "Dev digital"  
}  
  
{  
  "_id" : 1234.0,  
  "prenom" : "Alaa",  
  "niveau" : "2A",  
  "filier" : "Dev digital"  
}
```

Types de données

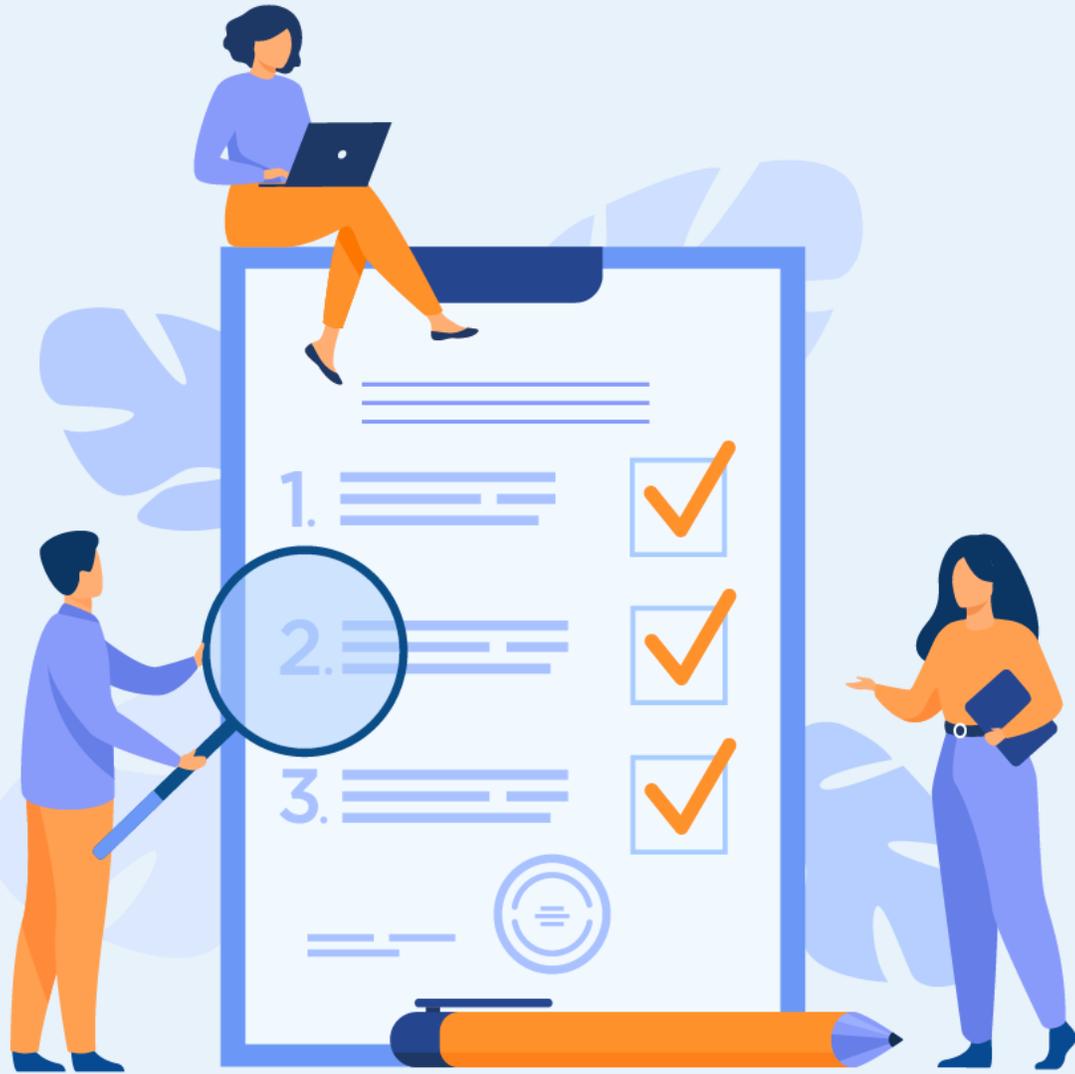
- Dans **MongoDB**, les documents sont stockés dans **BSON**, le format codé binaire de **JSON**, elle prend en charge divers types de données à savoir:
 - **ObjectId (Id d'objet)**: pour chaque nouveau document crée dans une collection, MongoDB crée automatiquement un identifiant d'objet unique `_id` s'il n'est pas crée explicitement,

- **Exemple:**

```
{
  "_id" :
  ObjectId("62dd128a1a19f3d7ecc6624f"),
  "prenom" : "Ahmed",
  "niveau" : "1A",
  "filier" : "Dev digital"
}
{
  "_id" : 1234.0,
  "prenom" : "Alaa",
  "niveau" : "2A",
  "filier" : "Dev digital"
}
```

← Identifiant généré automatiquement par MongoDB

← Identifiant crée explicitement par MongoDB



CHAPITRE 4

Manipuler les données avec MongoDB

Ce que vous allez apprendre dans ce chapitre :

- Manipuler les requêtes de base CRUD
- Rechercher l'information,
- Utiliser les opérateurs,
- Trier les documents.

 07,5 heures

CHAPITRE 4

Manipuler les données avec MongoDB

1. Manipuler les requêtes de base CRUD
2. Rechercher l'information,
3. Utiliser les opérateurs,
4. Trier les documents.



01 – MongoDB et requêtes CRUD

Insérer des documents



Insérer des documents

- L'insertion sous MongoDB se fait en utilisant la méthode **insert** en fournissant en paramètres le ou les document (s) à insérer:

```
db.le_nom_de_la_collection.insert({"cle1" : "valeur1", "cle2" : "valeur2"})
```

```
db.Stagiaires.insert ({"prenom": "Amina", "filiere": "Dev Digital ", "niveau": "1A" })
```

- On peut utiliser la méthode **insertOne**:

```
db.le_nom_de_la_collection.insertOne ({"cle1" : "valeur1", "cle2" : "valeur2"})
```

```
db.Stagiaires.insertOne ({"prenom": "Amina", "filiere": "Dev Digital ", "niveau": "1A" })
```

01 – MongoDB et requêtes CRUD

Supprimer des documents



Supprimer des documents

- Pour supprimer une collection, on utilise la fonction drop().

```
db.le_nom_de_la_collection.drop()
```

- Pour supprimer un document de la collection:

```
db.le_nom_de_la_collection.deleteOne ({"cle1" : "valeur1"})
```

```
db.Filieres.deleteOne ({"_id": "DD" })
```

- Pour supprimer plusieurs documents répondants à une condition de la collection:

```
db.le_nom_de_la_collection.deleteMany ({"cle1" : "valeur1"})
```

```
db.Filieres.deleteMany ({"intitule": "Développement Digital" })
```

01 – MongoDB et requêtes CRUD

Remplacer un document



Remplacer un document

- MongoDB met à disposition la fonction update avec différents opérateurs en fonction du type de mise à jour souhaité.
- La fonction update prend deux arguments obligatoires :
 - un document représentant la condition de recherche des documents de la collection
 - un document représentant la mise à jour souhaitée
- Si plusieurs documents vérifient la condition de recherche, seul le premier document sera modifié.

```
db.le_nom_de_la_collection.update()
```

```
db.Filieres.update(  
{"_id":"ID", "intitule":"Infrastructure digitale"},  
{"_id":"ID", "intitule":"Infrastructure digitale", "nbOptions":3} )
```

01 – MongoDB et requêtes CRUD

Remplacer un document



Remplacer un document

- Pour modifier plusieurs documents à la fois, il est nécessaire d'ajouter **{multi: true}** en fin de requête.

Exemple

```
db.Filieres.update(  
  {"_id":"ID", "intitule":"Infrastructure digitale"},  
  {"_id":"ID", "intitule":"Infrastructure digitale", "nbOptions":3} ,  
  {multi: true}  
)
```

01 – MongoDB et requêtes CRUD

Mettre à jour des documents



Mettre à jour des documents

- MongoDB met à disposition la fonction update avec différents opérateurs en fonction du type de mise à jour souhaité.
- Si l'on souhaite conserver les autres champs, il suffit d'inclure la seconde ligne dans un **\$set**.

Exemple

```
db.Filieres.update(  
  {"intitule" : "Developpement digital"},  
  {$set: {"organisme": "OFPPT"}},  
  {multi: true}  
)
```

Sélection

- MongoDB propose deux types de requêtes simples *find* et *findOne*

```
db.le_nom_de_la_collection.findOne()
```

Sélectionner le premier document d'une collection

```
db.le_nom_de_la_collection.find()
```

Sélectionner tous les documents d'une collection

Sélection

Soit la collection Stagiaires avec les documents suivants:

```
db.Stagiaires.insert({
  "nom": "Alami",
  "prenom": "Amina",
  "filier": {"_id": "DD",
            "intitule": "Developpement digital"},
  "moy1A": 14.5,
  "niveau": "2A",
  "option": "Mobile"})
```

```
db.Stagiaires.insert({
  "nom": "Ennaim",
  "prenom": "Nidal",
  "filier": {"_id": "ID",
            "intitule": "Infrastructure digitale"},
  "moy1A": 17.25,
  "niveau": "2A",
  "option": "Cyber Security"})
```

```
db.Stagiaires.insert({
  "nom": "Alami",
  "prenom": "Salim",
  "filier": {"_id": "DD",
            "intitule": "Developpement digital"},
  "moy1A": 12.75,
  "niveau": "2A",
  "option": "Full Stack"})
```

```
db.Stagiaires.insert({
  "nom": "Dalil",
  "prenom": "Karima",
  "filier": {"_id": "DDesign",
            "intitule": "Digital Design"},
  "niveau": "1A" })
```

Sélection

```
db.Stagiaires.findOne()
```

```
{ "_id" : ObjectId("62e01de8dc15f0d3b0bc282c"),  
  "nom" : "Alami",  
  "prénom" : "Amina",  
  "filière" : {  
    "_id" : "DD",  
    "intitule" : "Développement digital"  
  },  
  "moy1A":14.50,  
  "niveau" : "2A",  
  "option" : "Mobile"}
```

```
db.Stagiaires.find()
```

```
{ "_id" : ObjectId("62e01e5fdc15f0d3b0bc282f"),  
  "nom" : "Alami",  
  "prénom" : "Amina",  
  "filière" : {  
    "_id" : "DD"  
  },  
  { "_id" : ObjectId("62e01e5fdc15f0d3b0bc2830"),  
    "nom" : "Ennaïm",  
    "prénom" : "Nidal",  
    "filière" : {  
      "_id" : "TD"  
    }  
  },  
  { "_id" : ObjectId("62e01e5fdc15f0d3b0bc2831"),  
    "nom" : "Alami",  
    "prénom" : "Salim",  
    "filière" : {  
      "_id" : "DD",  
      "intitule" : "Développement digital"  
    }  
  },  
  "moy1A":12.75,  
  "niveau" : "2A",  
  "option" : "FullStack"}
```

Restriction

- La restriction permet de sélectionner les documents qui vérifient une condition. La condition est un fichier JSON avec les critères de sélection des documents.
- Ce document JSON doit être passé en paramètre à la requête **find** ou **findOne**

```
db.le_nom_de_la_collection.find(  
  {"champ1":"val1","champ2":"val2"}  
)
```



Sélectionne les documents dont les valeurs des champs (champ1, champ2) sont respectivement val1 et val2

```
db.le_nom_de_la_collection.findOne(  
  {"champ1":"val1","champ2":"val2"}  
)
```



Sélectionne le premier document dont les valeurs des champs (champ1, champ2) sont respectivement val1 et val2

Restriction

Sélectionner le premier stagiaire de la collection Stagiaires dont le nom est "Alami"

```
db.Stagiaires.findOne({"nom":"Alami"})
```



```
{
  "_id" : ObjectId("62e0249adc15f0d3b0bc2838"),
  "nom" : "Alami",
  "prenom" : "Amina",
  "filier" : {
    "_id" : "DD",
    "intitule" : "Développement digital"
  },
  "moy1A" : 14.50,
  "niveau" : "2A",
  "option" : "Mobile"
}
```

02 – Recherche de l'information

Restriction



Restriction

Sélectionner le premier stagiaire de la collection Stagiaires dont le nom est "Alami"

```
db.Stagiaires.find({"nom":"Alami"})
```

```
{
  "_id" : ObjectId("62e0249adc15f0d3b0bc2838"),
  "nom" : "Alami",
  "prenom" : "Amina",
  "filiere" : {
    "_id" : "DD",
    "intitule" : "Développement digital"
  },
  "moy1A":14.50",
  "niveau" : "2A",
  "option" : {
    "_id" : ObjectId("62e0249adc15f0d3b0bc283a"),
    "nom" : "Alami",
    "prenom" : "Salim",
    "filiere" : {
      "_id" : "DD",
      "intitule" : "Développement digital"
    },
    "moy1A":12.75",
    "niveau" : "2A",
    "option" : "Mobile"
  }
}
```

Projection

- La projection permet de limiter les informations retournées en précisant les champs souhaités dans un document JSON passé en paramètre à la requête **find** ou **findOne**

```
db.nom_de_la_collection.find(  
  {}, {"champ1":1, "champ2":1})
```



Sélectionne les champs (champ1, champ2) de tous les documents d'une collection

```
db.nom_de_la_collection.findOne(  
  {}, {"champ1":1, "champ2":1})
```



Sélectionne les champs (champ1, champ2) du premier document d'une collection

Projection

Sélectionner les noms et prénoms du premier stagiaires de la collection Stagiaires

```
db.Stagiaires.findOne({}, {"nom":1, "prenom":1})
```



```
{  
  "_id" : ObjectId("62e01de8dc15f0d3b0bc282c"),  
  "nom" : "Alami",  
  "prénom" : "Amina"  
}
```

Remarques.

- Pour ajouter un champ à la sélection, il suffit de le préciser et lui affecter la valeur 1,
- Le champ `_id` est renvoyé systématiquement, pour l'exclure, il faut le préciser et lui affecter la valeur 0.

```
db.Stagiaires.findOne({},  
 {"nom":1, "prenom":1, " _id":0})
```



```
{  
  "nom" : "Alami",  
  "prénom" : "Amina"  
}
```

Projection

Sélectionner les noms et prénoms des stagiaires

```
db.Stagiaires.find({}, {"nom":1, "prenom":1, "_id":0})
```



```
{  
  "nom" : "Alami",  
  "prenom" : "Amina"  
}
```

```
{  
  "nom" : "Ennaim",  
  "prenom" : "Nidal"  
}
```

```
{  
  "nom" : "Alami",  
  "prenom" : "Salim"  
}
```

Projection

Sélectionner les prénoms et les options des stagiaires dont le nom est « Alami »

```
db.Stagiaires.find({"nom": "Alami"}, {"prenom":1,"option":1,"_id":0})
```



```
{  
  "prenom" : "Amina",  
  "option" : "Mobile"  
}
```

```
{  
  "prenom" : "Salim",  
  "option" : "FullStack"  
}
```

Projection

On peut créer les deux fichiers JSON de la sélection et la projection en dehors de la requête

```
projection = {"prenom":1,"option":1,"_id":0}  
selection = {"nom":"Alami"}  
db.Stagiaires.find(selection,projection)
```



```
{  
  "prenom" : "Amina",  
  "option" : "Mobile"  
}
```

```
{  
  "prenom" : "Salim",  
  "option" : "FullStack"  
}
```

Projection

Sélectionner les noms et prénoms des stagiaires de l'option « Mobile »

```
projection = {"nom":1,"prenom":1,"_id":0}  
sélection = {"option": "Mobile"}  
db.Stagiaires.find(sélection , projection)
```



```
{  
  "nom" : "Alami",  
  "prénom" : "Amina"  
}
```

Projection

- Pour spécifier un sous champ d'un champ, il est nécessaire d'utiliser le formalisme **champ.souschamp**

```
projection = {"nom":1,"prenom":1,"_id":0}
sélection = {"filiere.intitule": "Developpement digital"}
db.Stagiaires.find(sélection , projection)
```



```
{
  "nom" : "Alami",
  "prénom" : "Amina"
}
```

```
{
  "nom" : "Alami",
  "prénom" : "Salim"
}
```

02 – Recherche de l'information

Fonction distinct()



Fonction distinct()

- Une fonction très utile pour lister les valeurs prises par le champ de la collection passé en paramètre,
- **distinct()** retourne les valeurs prises par le champ sous forme de tableau

```
db.Stagiaires.distinct("option")
```



```
[  
  "Cyber Security",  
  "FullStack",  
  "Mobile"  
]
```

- On utilise toujours le même formalisme **champ.souschamp** pour lister les valeurs d'un sous champ

```
db.Stagiaires.distinct("filieres.intitule")
```



```
[  
  "Développement digital",  
  "Infrastructure digitale"  
]
```

02 – Recherche de l'information

Fonction count



Fonction count

- Une fonction très utile pour faire des dénombrements suite à des sélections et connaître la taille du résultat ,
- **count()** s'ajoute à la suite d'une fonction **find** et retourne le nombre de documents renvoyés

```
db.Stagiaires.find().count()
```



4 // nombre total de Stagiaires dans La collection

```
db.Stagiaires.find({"niveau":"2A"}).count()
```



3 // nombre de stagiaires en deuxième années

Fonction limit

- Cette fonction permet de limiter le nombre de documents renvoyés par la fonction **find**,
- **limit(n)** s'ajoute à la suite d'une fonction **find** et retourne les n premiers documents resultats,
- **limit()**, sans spécification du nombre n, retourne tous les documents renvoyés par **find**

```
db.Stagiaires.find({}, {"nom":1,"prenom":1,"_id":0}).limit(2)
```



```
{  
  "nom" : "Alami",  
  "prenom" : "Amina"  
}  
  
{  
  "nom" : "Ennaim",  
  "prenom" : "Nidal"  
}
```

03 – Les opérateurs

Les opérateurs de comparaison



Les opérateurs de comparaison

Opérateur	Description	Exemple
\$eq	<i>Equal</i> : =	Sélectionner les stagiaires dont la moyenne de la première année est supérieure ou égale à 14.00:
\$gt	<i>Greater Than</i> : >	
\$gte	<i>Greater Than or Equal</i> : >=	selection = {"moy1A":{"\$gte":14.00}}
\$lt	<i>Less Than</i> : <	db.Stagiaires.find(selection,{"nom":1,"prenom":1,"_id":0})
\$lte	<i>Less Than or Equal</i> : <=	
\$ne	<i>Not Equal</i> : ≠	

```
{
  "nom" : "Alami",
  "prenom" : "Amina"
}
{
  "nom" : "Ennaim",
  "prenom" : "Nidal"
}
```

03 – Les opérateurs

Les opérateurs de comparaison de listes



Les opérateurs de comparaison de listes

Opérateur	Description	Exemple
\$in	Appartient à la liste	Sélectionner les stagiaires dont l'option est FullStack ou Cyber Security : <pre>selection = {"option":{"\$in":["FullStack","Cyber Security"]}} db.Stagiaires.find(selection,{"nom":1,"prenom":1,"_id":0})</pre>
\$nin	N'appartient pas à la liste	<pre>{ "nom" : "Ennaim", "prenom" : "Nidal" }</pre> <pre>{ "nom" : "Alami", "prenom" : "Salim" }</pre>

03 – Les opérateurs

Les opérateurs logiques



Les opérateurs logiques

Opérateur	Description	Exemple
\$and	ET Logique	Sélectionner les stagiaires qui ont une moyenne de la première année supérieure ou égale à 15.00 ou bien qui ont choisi l'option est Mobile : selection = { "\$or": [{ "moy1A" : { "\$gte" : 15 } }, { "option": "Mobile" }] } db.Stagiaires.find(selection,{"nom":1,"prenom":1,"_id":0})
\$or	OU Logique	
\$nor	Not (OU Logique)	<pre>{ "nom" : "Alami", "prenom" : "Amina" } { "nom" : "Ennaim", "prenom" : "Nidal" }</pre>

Les opérateurs logiques

Opérateur	Description	Exemple
\$and	ET Logique	Sélectionner les stagiaires qui ne vérifient pas la condition suivante : « stagiaires ayant une moyenne de la première année supérieure ou égale à 15.00 ou bien qui ont choisi l'option est Mobile »:
\$or	OU Logique	selection = { "\$nor": [{ "moy1A" : { "\$gte" : 15 } }, { "option": "Mobile" }] }
\$nor	Not (OU Logique)	db.Stagiaires.find(selection, {"nom":1, "prenom":1, "_id":0}) <pre>{ "nom" : "Alami", "prenom" : "Salim" } { "nom" : "Dalil", "prenom" : "Karima" }</pre>

L'opérateur Exists

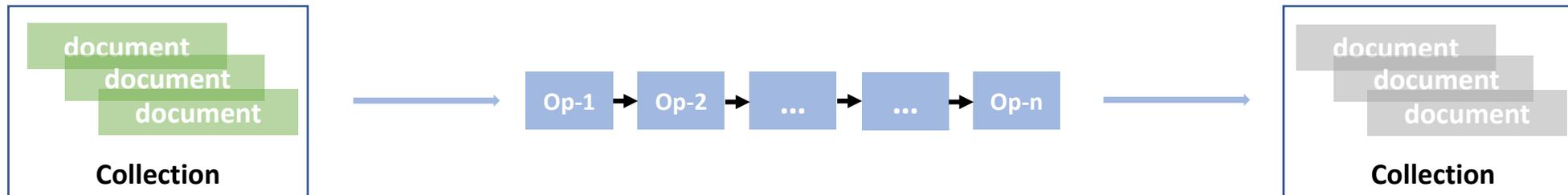
Opérateur	Description	Exemple
\$exists	Teste la présence ou non d'un champ	<p>Sélectionner les stagiaires qui n'ont pas d'option</p> <pre>selection = { "option": { \$exists : false } }</pre> <pre>db.Stagiaires.find(selection, {"nom":1,"prenom":1,"_id":0})</pre> <pre>{ "nom" : "Dalil", "prenom" : "Karima" }</pre>

L'opérateur Exists

Opérateur	Description	Exemple
\$exists	Teste la présence ou non d'un champ	<p>Sélectionner les stagiaires qui ont une option</p> <pre>selection = { "option": {\$exists : true } }</pre> <pre>db.Stagiaires.find(selection,{"nom":1,"prenom":1,"_id":0})</pre> <pre>{ "nom" : "Alami", "prenom" : "Amina" }</pre> <pre>{ "nom" : "Ennaim", "prenom" : "Nidal" }</pre> <pre>{ "nom" : "Alami", "prenom" : "Salim" }</pre>

L'agrégation

- En plus de la recherche classique, le calcul d'agrégat est très utilisé, pour l'analyse, la modélisation ou la visualisation de données,
- Ce calcul s'effectue avec la fonction `aggregate()` qui prend en paramètre un tableau d'opérations (**pipeline**),
- L'output de chaque opération du pipeline sera l'input de l'opération suivante, pour produire une nouvelle collection
- Chaque étape du pipeline est définie par un opérateur passé à la commande d'agrégation suivante:



```
db.nom_de_la_collection.aggregate([opérateur1, opérateur2, ...])
```

03 – Les opérateurs

Les opérateurs d'agregation : \$match



Les opérateurs d'agregation : \$match

- Permet de réduire la quantité de documents qui sera fourni en entrée à l'étape qui va suivre,
- On pourrait le mettre n'importe où dans la requête mais il est particulièrement intéressant en début ou en fin de requête,

```
db.nom_de_la_collection.aggregate([{$match: {selection}}])
```



Filterer les documents de la collection suivant la sélection
fournie à l'opérateur **\$match**

03 – Les opérateurs

Les opérateurs d'agregation : \$match



Les opérateurs d'agregation : \$match

- Retourner les stagiaires de la filiere développement digital

```
db.Stagiaires.aggregate([
  {$match: {"filiere.intitule" : "Developpement digital" }}
])
```



```
{ "_id" : ObjectId("62e0f02cd55ee92da665d213"),
  "nom" : "Alami",
  "prenom" : "Amina",
  "filiere" : {
    "_id" : "DD",
    "intitule" : "Developpement digital"
  },
  "moy1A" : 14.5,
  "niveau" : "2A",
  "option" : "Mobile" }
```

```
{
  "_id" : ObjectId("62e0f02cd55ee92da665d215"),
  "nom" : "Alami",
  "prenom" : "Salim",
  "filiere" : {
    "_id" : "DD",
    "intitule" : "Developpement digital"
  },
  "moy1A" : 12.75,
  "niveau" : "2A",
  "option" : "FullStack" }
```

03 – Les opérateurs

Les opérateurs d'agregation : \$group



Les opérateurs d'agregation : \$group

- Permet d'effectuer des regroupements et faire des opérations d'accumulation sur les documents (**GROUP BY** en SQL),
- Le champ de regroupement est indiquée par id,
- On peut faire tous les calculs d'agrégats classique en utilisant les fonctions d'agregation suivantes:

\$sum (somme), **\$avg** (moyenne), **\$min** (minimum), **\$max** (maximum)

```
db.nom_de_la_collection.aggregate([
  {$group: { _id: $champ_regroupement,
  variable: { fonctionAggregation : $champ },
  ...}
}]
```

03 – Les opérateurs

Les opérateurs d'agregation : \$group

Les opérateurs d'agregation : \$group

- Calculer la somme des moyennes de la première année par filière:

```
db.Stagiaires.aggregate([  
  {$group: {_id: "$filiere.intitule", total : {$sum:"$moy1A"}}}  
])
```



```
{  
  "_id" : "Digital Design",  
  "total" : NumberInt(0)  
}
```

```
{  
  "_id" : "Developpement digital",  
  "total" : 27.25  
}
```

```
{  
  "_id" : "Infrastructure digitale",  
  "total" : 17.25  
}
```

03 – Les opérateurs

Les opérateurs d'agregation : \$group



Les opérateurs d'agregation : \$group

Renvoyer la moyenne maximale de la première année par filière:

```
db.Stagiaires.aggregate([
  {$group: { _id: "$filiere.intitule", noteMax : {$max:"$moy1A"}}}
])
```



```
{
  "_id" : "Digital Design",
  "noteMax" : null
}
```

```
{
  "_id" : "Developpement digital",
  "noteMax" : 14.5
}
```

```
{
  "_id" : "Infrastructure digitale",
  "noteMax" : 17.25
}
```

03 – Les opérateurs

Les opérateurs d'agregation : \$sort



Les opérateurs d'agregation : \$sort

- Renvoyer la moyenne maximale de la première année par filière triée par ordre croissant:

```
db.Stagiaires.aggregate([
  {$group: {$_id: "$filiere.intitule", noteMax : {$max:"$moy1A"}}},
  {$sort: {noteMax:1}} //tri suivant l'ordre croissant des noteMax par filiere
                        //(voir slide 37)
])
```



```
{
  "_id" : "Digital Design",
  "noteMax" : null
}
```

```
{
  "_id" : "Developpement digital",
  "noteMax" : 14.5
}
```

```
{
  "_id" : "Infrastructure digitale",
  "noteMax" : 17.25
}
```

Fonction sort

- Le tri des documents renvoyés par une fonction **find** est réalisable avec la fonction **sort()**,
- On doit indiquer les champs de tri et leur attribuer une valeur de 1 pour un tri ascendant et une valeur de -1 pour un tri descendant,
- Sinon, les documents seront renvoyés sans aucun tri

```
db.Stagiaires.find({}, {"nom":1,"prenom":1,"_id":0}).sort({"nom":-1})
```



```
{  
  "nom" : "Ennaim",  
  "prenom" : "Nidal"  
}
```

```
{  
  "nom" : "Dalil",  
  "prenom" : "Karima"  
}
```

```
{  
  "nom" : "Alami",  
  "prenom" : "Amina"  
}
```

```
{  
  "nom" : "Alami",  
  "prenom" : "Salim"  
}
```

Fonction sort

- Le tri des documents renvoyés par une fonction **find** est réalisable avec la fonction **sort()**,
- On doit indiquer les champs de tri et leur attribuer une valeur de 1 pour un tri ascendant et une valeur de -1 pour un tri descendant,
- Sinon, les documents seront renvoyés sans aucun tri

```
db.Stagiaires.find({}, {"nom":1,"prenom":1,"_id":0}).sort({"nom":-1,"prenom":-1})
```



```
{  
  "nom" : "Ennaim",  
  "prenom" : "Nidal"  
}
```

```
{  
  "nom" : "Dalil",  
  "prenom" : "Karima"  
}
```

```
{  
  "nom" : "Alami",  
  "prenom" : "Salim"  
}
```

```
{  
  "nom" : "Alami",  
  "prenom" : "Amina"  
}
```

Fonction sort

- Le tri des documents renvoyés par une fonction **find** est réalisable avec la fonction **sort()**,
- On doit indiquer les champs de tri et leur attribuer une valeur de 1 pour un tri ascendant et une valeur de -1 pour un tri descendant,
- Sinon, les documents seront renvoyés sans aucun tri

```
db.Stagiaires.find({}, {"nom":1,"prenom":1,"_id":0}).sort()
```

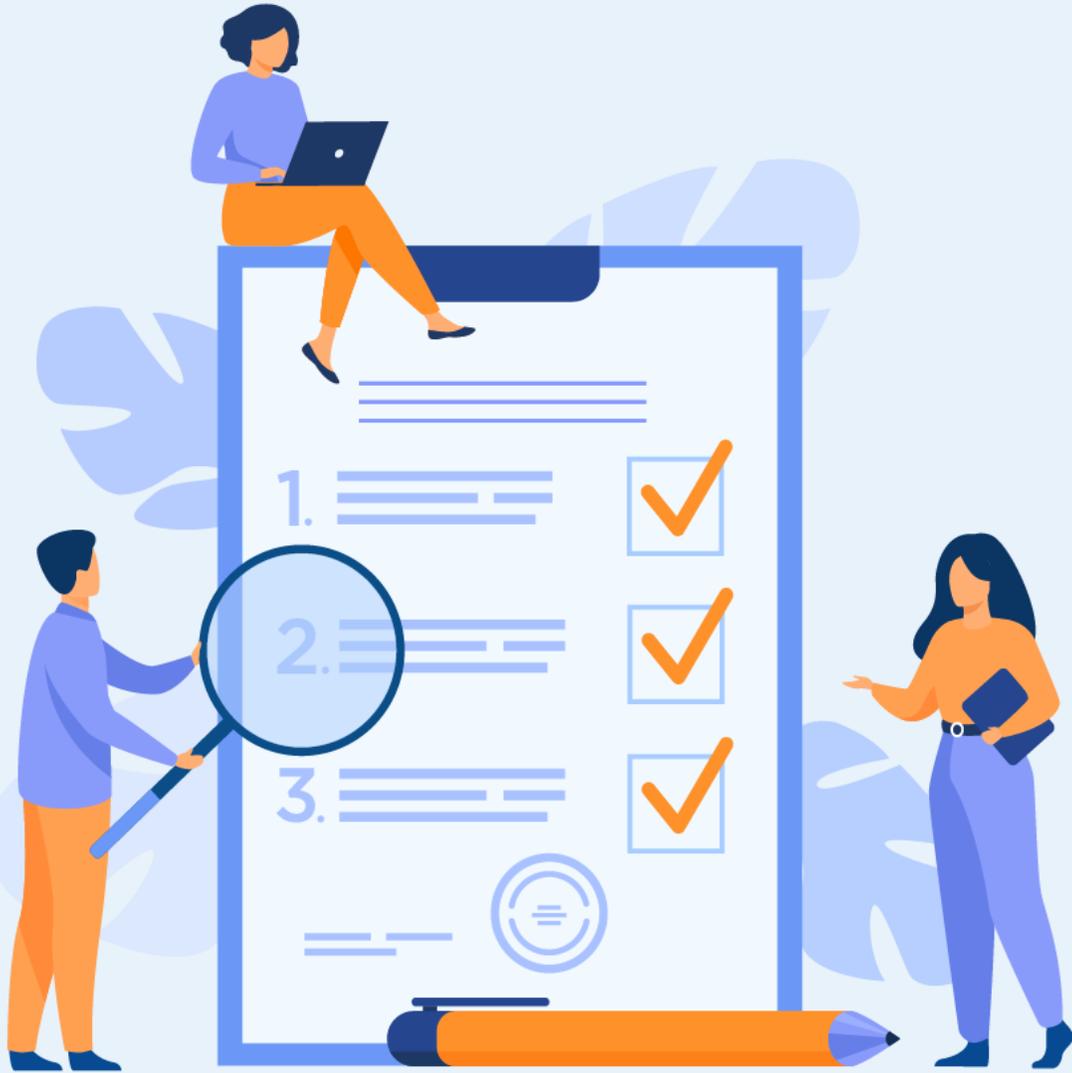


```
{  
  "nom" : "Alami",  
  "prenom" : "Amina"  
}
```

```
{  
  "nom" : "Ennaim",  
  "prenom" : "Nidal"  
}
```

```
{  
  "nom" : "Alami",  
  "prenom" : "Salim"  
}
```

```
{  
  "nom" : "Dalil",  
  "prenom" : "Karima"  
}
```



CHAPITRE 5

Effectuer des requêtes depuis des programmes Python

Ce que vous allez apprendre dans ce chapitre :

- Présentation et installation de pymongo,
- Connexion des bases de données avec le serveur MongoDB,
- Création des requêtes :
 - Requêtes simples,
 - Création des indexs,
 - Requêtes d'agrégation,
 - Requêtes de modifications.



07,5 heures

CHAPITRE 5

Effectuer des requêtes depuis des programmes Python

1. **Présentation et installation de pymongo,**
2. Connexion des bases de données avec le serveur MongoDB,
3. Création des requêtes :
 - Requêtes simples,
 - Création des indexs,
 - Requêtes d'agrégation,
 - Requêtes de modifications.

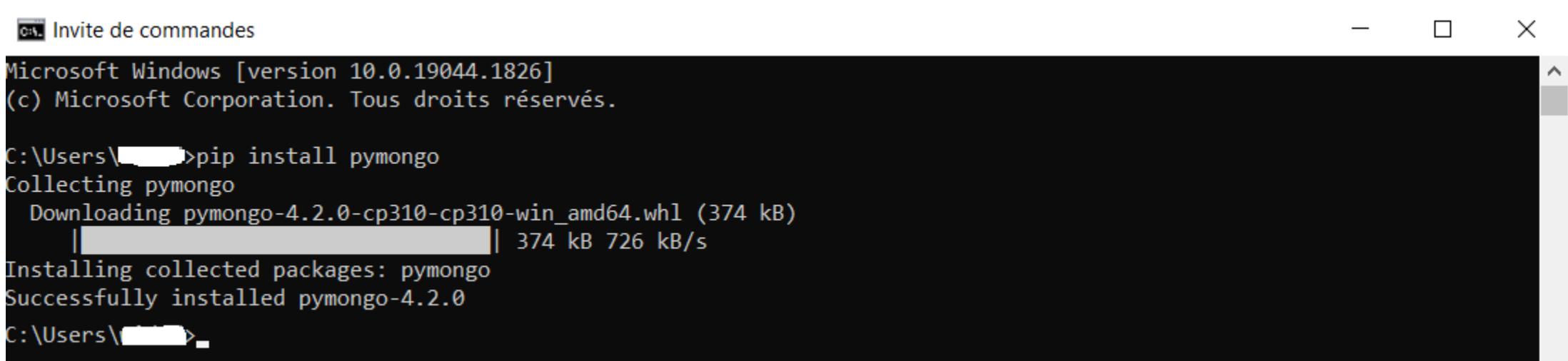


05 – Effectuer des requêtes depuis des programmes Python

Présentation et installation de pymongo,

Installation de pymongo

- **PyMongo** est une librairie Python native contenant des outils pour travailler avec MongoDB,
- **PyMongo** est maintenue par les développeurs de MongoDB officiel ce qui en fait la référence dans Python¹,
- Pour installer la librairie, il faut saisir la commande suivante dans un terminal (invité de commande par exemple)



```
Invite de commandes
Microsoft Windows [version 10.0.19044.1826]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\>pip install pymongo
Collecting pymongo
  Downloading pymongo-4.2.0-cp310-cp310-win_amd64.whl (374 kB)
    | 374 kB 726 kB/s
Installing collected packages: pymongo
Successfully installed pymongo-4.2.0
C:\Users\>
```

1: <https://pymongo.readthedocs.io/en/stable/>

CHAPITRE 5

Effectuer des requêtes depuis des programmes Python

1. Présentation et installation de pymongo,
2. **Connexion des bases de données avec le serveur MongoDB,**
3. Création des requêtes :
 - Requêtes simples,
 - Création des indexs,
 - Requêtes d'agrégation,
 - Requêtes de modifications.



05 – Effectuer des requêtes depuis des programmes Python

Connexion au serveur MongoDB



Connexion au serveur MongoDB

- La première étape consiste à créer une connexion avec le serveur **MongoDB**,
- Pour effectuer cette connexion on utilise **MongoClient**, qui se connecte, par défaut, à l'instance **MongoDB** s'exécutant sur localhost:**27017** si aucune chaîne de connexion n'est spécifiée,
- On commence par importer la classe **MongoDB** du module **pymongo**,
- Puis on affiche l'objet :

```
C:\Users\>python
Python 3.10.2 (tags/v3.10.2:a58ebcc, Jan 17 2022, 14:12:15) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from pymongo import MongoClient
>>> client = MongoClient()
>>> print(client)
MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True)
>>>
```

05 – Effectuer des requêtes depuis des programmes Python

Connexion au serveur MongoDB



Connexion à une base de données

- On peut spécifier une chaîne de connexion au serveur:

```
>>> client = MongoClient(host="localhost", port=27017)
>>> print(client)
MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True)
>>> _
```

- L'objet **client** est une instance de la classe **pymongo.mongo_client.MongoClient** où on retrouve les informations de la connexion comme le host le port, etc...
- Pour se connecter à la base de données *Exemples* du serveur objet de la connexion:

```
>>> db = client["Exemples"]
>>> print(db)
Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'Exemples')
>>> _
```

- Ainsi, l'objet **client** retourne un objet de classe **pymongo.database.Database**.

05 – Effectuer des requêtes depuis des programmes Python

Connexion au serveur MongoDB



Recapitulons

- La librairie **pymongo** permet d'utiliser l'IDE python pour manipuler les objets suivants:

L'interface cliente (la connexion au serveur),

Les bases de données du serveur,

et les collections d'une base de données spécifique.

CHAPITRE 5

Effectuer des requêtes depuis des programmes Python

1. Présentation et installation de pymongo,
2. Connexion des bases de données avec le serveur MongoDB,
3. **Création des requêtes :**
 - **Requêtes simples,**
 - **Création des indexs,**
 - **Requêtes d'agrégation,**
 - **Requêtes de modifications.**



05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes simples:

- **Pymongo** utilise la même syntaxe que l'interface **MongoDB**:

```
Client.baseDonnees.nomDeLaCollection.requete()
```

Requête simple	Explications
db.Stagiaires.find()	<p><pymongo.cursor.Cursor object at 0x000001D9D93BACE0></p> <ul style="list-style-type: none"> • Retourne un curseur (type Cursor) sur les données • Pymongo ne retourne pas les résultats sous forme de liste par souci de mémoire

```
>>> db.Stagiaires.find()
<pymongo.cursor.Cursor object at 0x000001D9D93BACE0>
```

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes simples:

- **Pymongo** utilise la même syntaxe que l'interface MongoDB:

Client.baseDonnees.nomDeLaCollection.requete()

Requête simple	Explications
<pre>resultat = db.Stagiaires.find() for r in resultat[:2]: print(r)</pre>	<ul style="list-style-type: none"> • On récupère le curseur dans la variable resultat (objet itérable en Python), • Pour accéder au contenu de la requête il faut parcourir l'objet renvoyé

```
>>> resultat = db.Stagiaires.find()
>>> for r in resultat[:2]:
...     print(r)
...
{'_id': ObjectId('62e0f02cd55ee92da665d213'), 'nom': 'Alami', 'prenom': 'Amina', 'filier': {'_id': 'DD', 'intitule': 'D
eveloppement digital'}, 'moy1A': 14.5, 'niveau': '2A', 'option': 'Mobile'}
{'_id': ObjectId('62e0f02cd55ee92da665d214'), 'nom': 'Ennaïm', 'prenom': 'Nidal', 'filier': {'_id': 'ID', 'intitule': '
Infrastructure digitale'}, 'moy1A': 17.25, 'niveau': '2A', 'option': 'Cyber Security'}
>>>
```

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes simples:

- Afin de rendre les requêtes plus lisibles, il est possible de créer des variables Python qui correspondent aux conditions de selection ou/et de projection quand utilise dans la requête

Requête simple	Explications
<pre>selection={} selection["option"]="Mobile" resultat = db.Stagiaires.find(selection) for i in resultat[:]: print(i)</pre>	<ul style="list-style-type: none"> On crée une variable selection de type dictionnaire , On y ajoute la clé et la valeur correspondant à la sélection voulue, On attribue la variable à la requête On récupère le curseur dans la variable resultat (objet itérable en Python), Pour accéder au contenu de la requête il faut parcourir l’objet renvoyé

```
>>> selection={}
>>> selection["option"]="Mobile"
>>> resultat = db.Stagiaires.find(selection)
>>> for i in resultat[:]:
...     print(i)
...
{'_id': ObjectId('62e0f02cd55ee92da665d213'), 'nom': 'Alami', 'prenom': 'Amina', 'filiere': {'_id': 'DD', 'intitule': 'Developpement digital'}, 'moy1A': 14.5, 'niveau': '2A', 'option': 'Mobile'}
>>>
```

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes simples:

- Afin de rendre les requêtes plus lisibles, il est possible de créer des variables **Python** qui correspondent aux conditions de sélection ou/et de projection quand utilise dans la requête.

Requête simple	Explications
<pre>projection={"nom":1,"prenom":1,"_id":0} selection={} selection["filiere.intitule"]="Developpement digital" resultat = db.Stagiaires.find(selection,projection) for i in resultat[:]: print(i)</pre>	<ul style="list-style-type: none"> On ajoute une variable projection de type dictionnaire avec les elements correspondants aux champs de projection de la requête On attribue les deux variables à la requête On récupère le curseur dans la variable resultat puis on parcourt l'objet renvoyé

```
>>> projection={"nom":1,"prenom":1,"_id":0}
>>> selection={}
>>> selection["filiere.intitule"]="Developpement digital"
>>> resultat = db.Stagiaires.find(selection,projection)
>>> for i in resultat[:]:
...     print(i)
...
...
{'nom': 'Alami', 'prenom': 'Amina'}
{'nom': 'Alami', 'prenom': 'Salim'}
>>>
```

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes simples:

- Pour utiliser certaines méthodes sur la requête, c'est sensiblement toujours la même syntaxe de MongoDB

Client.baseDonnees.nomDeLaCollection.requete().methode()

Requête simple	Explications
<pre>r = db.Stagiaires.find().sort("nom",-1) for i in r[:]: print(i)</pre>	<ul style="list-style-type: none"> • On récupère le curseur dans la variable resultat (objet itérable en Python), • Pour accéder au contenu de la requête il faut parcourir l'objet renvoyé

```
>>> r = db.Stagiaires.find().sort("nom",-1)
>>> for i in r[:]:
...     print(i)
...
{'_id': ObjectId('62e0f02cd55ee92da665d214'), 'nom': 'Ennaim', 'prenom': 'Nidal', 'filiere': {'_id': 'ID', 'intitule': 'Infrastructure digitale'}, 'moy1A': 17.25, 'niveau': '2A', 'option': 'Cyber Security'}
{'_id': ObjectId('62e0fe5fd55ee92da665d216'), 'nom': 'Dalil', 'prenom': 'Karima', 'filiere': {'_id': 'DDesign', 'intitule': 'Digital Design'}, 'niveau': '1A'}
{'_id': ObjectId('62e0f02cd55ee92da665d213'), 'nom': 'Alami', 'prenom': 'Amina', 'filiere': {'_id': 'DD', 'intitule': 'Developpement digital'}, 'moy1A': 14.5, 'niveau': '2A', 'option': 'Mobile'}
{'_id': ObjectId('62e0f02cd55ee92da665d215'), 'nom': 'Alami', 'prenom': 'Salim', 'filiere': {'_id': 'DD', 'intitule': 'Developpement digital'}, 'moy1A': 12.75, 'niveau': '2A', 'option': 'FullStack'}
>>>
```

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Utilisation des index:

- **Définition:**

- Des structures de données spéciales qui stockent une petite partie de l'ensemble de données de la collection sous une forme facile à parcourir,
- L'index stocke la valeur d'un champ spécifique ou d'un ensemble de champs, triés par la valeur du champ.

- **Syntaxe:**

```
Client.BaseDeDonnee.Collection.requete()
```

- **Avec :**

Requête	Explication	Exemples
index_information()	Retourne la liste des index de la collection	for k,v in db.Stagiaires.index_information().items(): print("index:" ,k, "valeur :",v,"\n")
create_index()	Création d'un index	db.Stagiaires.create_index("ind1")
drop_index()	Suppression d'un index	db.Stagiaires.drop_index("ind1")

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes d'agrégation

- Les requêtes d'agrégation ont pour but de faire des calculs simples (agrégats) sur toute la collection ou seulement sur certains groupes,
- La syntaxe reste la même que l'interface **MongoDB**

```
Client.BaseDeDonnee.Collection.aggregate()
```

Exemple	Explication
<pre>resultat = db.Stagiaires.aggregate([{"\$group":{"_id":"\$filiere.intitule", "moyenne":{"\$avg":"\$moy1A"}} }) for i in resultat: print(i["moyenne"]," moyenne de la filiere ",i["_id"])</pre>	<ul style="list-style-type: none"> • On récupère le curseur dans la variable resultat, • Pour accéder au contenu de la requête il faut parcourir l'objet renvoyé:

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes d'agrégation

- Les requêtes d'agrégation ont pour but de faire des calculs simples (agrégats) sur toute la collection ou seulement sur certains groupes,
- La syntaxe reste la même que l'interface MongoDB

Client.BaseDeDonnee.Collection.aggregate()

Exemple	Explication
<pre>resultat = db.Stagiaires.aggregate([{"\$group":{"_id":"\$filiere.intitule", "moyenne":{"\$avg":"\$moy1A"}} }]) for i in resultat: print(i["moyenne"]," moyenne de la filiere ",i["_id"])</pre>	<ul style="list-style-type: none"> • On récupère le curseur dans la variable resultat, • Pour accéder au contenu de la requête il faut parcourir l'objet renvoyé:

```
>>> resultat = db.Stagiaires.aggregate([
... {"$group":{"_id":"$filiere.intitule","moyenne":{"$avg":"$moy1A"}}}]])
>>> for i in resultat:
...     print(i["moyenne"]," moyenne de la filiere ",i["_id"])
...
None moyenne de la filiere Digital Design
17.25 moyenne de la filiere Infrastructure digitale
13.625 moyenne de la filiere Developpement digital
>>>
```

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes de modification

- La syntaxe reste la même que l'interface **MongoDB**

```
Client.BaseDeDonnee.Collection.requete()
```

Requête pymongo	Fonctionnement
insert_one()	Insertion d'un seul document
insert_many()	Insertion d'une liste de documents
delete_one()	Suppression d'un document
delete_many()	Suppression d'une liste de documents
update_one()	Modification d'un document
update_many()	Modification d'une liste de documents

05 – Effectuer des requêtes depuis des programmes Python

Création des requêtes avec pymongo

Requêtes de modification

- Exemple d'insertion :

```
Invite de commandes - python
>>> db.Stagiaires.insert_one({'_id':"12345678","nom":"Jobs","prenom":"Steve","filiere":{"_id":"ID","intitule":"Infrastructure digitale"},"moy1A":14.25,"niveau":"2A","option":"Cloud"})
<pymongo.results.InsertOneResult object at 0x000001D9D93BB4F0>
>>> resultat = db.Stagiaires.find()
>>> for i in resultat[:]:
...     print(i)
...
{'_id': ObjectId('62e0f02cd55ee92da665d213'), 'nom': 'Alami', 'prenom': 'Amina', 'filiere': {'_id': 'DD', 'intitule': 'Developpement digital'}, 'moy1A': 14.5, 'niveau': '2A', 'option': 'Mobile'}
{'_id': ObjectId('62e0f02cd55ee92da665d214'), 'nom': 'Ennaïm', 'prenom': 'Nidal', 'filiere': {'_id': 'ID', 'intitule': 'Infrastructure digitale'}, 'moy1A': 17.25, 'niveau': '2A', 'option': 'Cyber Security'}
{'_id': ObjectId('62e0f02cd55ee92da665d215'), 'nom': 'Alami', 'prenom': 'Salim', 'filiere': {'_id': 'DD', 'intitule': 'Developpement digital'}, 'moy1A': 12.75, 'niveau': '2A', 'option': 'FullStack'}
{'_id': ObjectId('62e0fe5fd55ee92da665d216'), 'nom': 'Dalil', 'prenom': 'Karima', 'filiere': {'_id': 'DDesign', 'intitule': 'Digital Design'}, 'niveau': '1A'}
{'_id': '12345678', 'nom': 'Jobs', 'prenom': 'Steve', 'filiere': {'_id': 'ID', 'intitule': 'Infrastructure digitale'}, 'moy1A': 14.25, 'niveau': '2A', 'option': 'Cloud'}
>>>
```



CHAPITRE 6

Sécuriser une base de données MongoDB

Ce que vous allez apprendre dans ce chapitre :

- Importer / exporter les données
- Sécuriser les accès (authentification)



07,5 heures

CHAPITRE 6

Sécuriser une base de données MongoDB

1. Import / export des données avec MongoDB
2. Sécurité des accès (authentification)



Vue générale

- MongoDB propose deux manières d'import/export de la base de données:
 - mongoexport/mongoimport
 - mongodump/mongorestore
- On s'intéresse plutôt à mongoexport/mongoimport:
 - mongoexport: utilisé pour export (exporter) des données dans une Collection à un fichier (json, csv,..)
 - mongoimport: utilisé pour import (importer) des données dans une Collection à un fichier (json, csv,..)
- Mongoexport et mongoimport sont deux exécutables à part entière, pas deux commandes du mongo shell,
- Pour utiliser mongoexport ou mongoimport, le démon mongod doit être lancé, et l'exécutable s'y connectera

Utilisation de mongoexport

- On peut exporter les données d'une collection à un fichier JSON ou un fichier CSV
- La syntaxe est assez simple :

Exporter vers JSON

```
mongoexport -d nomBaseDonnees  
-c nomDeLaCollection  
-o fichier.json
```

Exporter la collection *nomDeLaCollection* de la base *nomBaseDonnees* dans le fichier *fichier.json*.

Si le fichier préexistant, il sera écrasé,

Exemple :

```
mongoexport -d Exemples -c Stagiaires -o C:/data/stagiaires.json
```

Stagiaires.json - Bloc-notes

Fichier Edition Format Affichage Aide

```
{ "_id" : ObjectId("62e0f02cd55ee92da665d213"),  
  "nom" : "Alami",  
  "prenom" : "Amina",  
  "filiere" : {"_id" : "DD","intitule" : "Developpement digital"},  
  "moy1A" : 14.5,  
  "niveau" : "2A",  
  "option" : "Mobile"}  
{ "_id" : ObjectId("62e0f02cd55ee92da665d214"),  
  "nom" : "Ennaim",  
  "prenom" : "Nidal",  
  "filiere" : {"_id" : "ID","intitule" : "Infrastructure digitale"},  
  "moy1A" : 17.25,  
  "niveau" : "2A",  
  "option" : "Cyber Security"}  
{ "_id" : ObjectId("62e0f02cd55ee92da665d215"),  
  "nom" : "Alami",  
  "prenom" : "Salim",  
  "filiere" : {"_id" : "DD","intitule" : "Developpement digital"},  
  "moy1A" : 12.75,  
  "niveau" : "2A",  
  "option" : "FullStack"}  
{ "_id" : ObjectId("62e0fe5fd55ee92da665d216"),  
  "nom" : "Dalil",  
  "prenom" : "Karima",  
  "filiere" : {"_id" : "DDesign","intitule" : "Digital Design"},  
  "niveau" : "1A"}  
{  
  "_id" : "12345678",  
  "nom" : "Jobs",  
  "prenom" : "Steve",  
  "filiere" : {"_id" : "ID","intitule" : "Infrastructure digitale"},  
  "niveau" : "1A"}
```

Utilisation de mongoexport

- On peut exporter les données d'une collection à un fichier JSON ou un fichier CSV
- La syntaxe est assez simple :

Exporter vers CSV

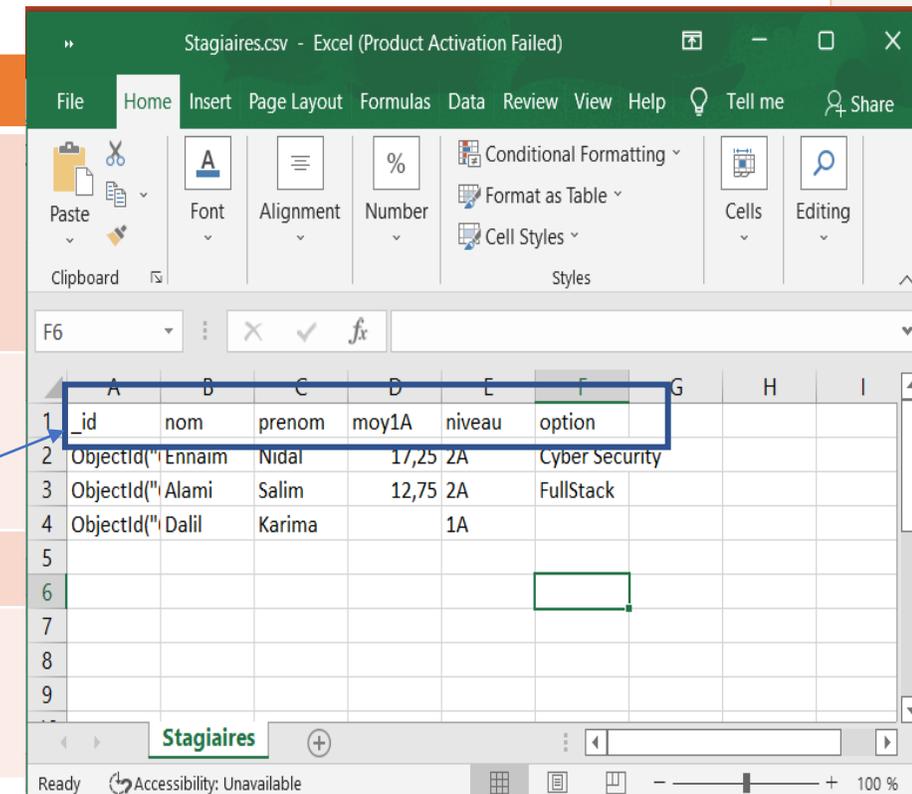
```
mongoexport -d nomBaseDonnees  
            -c nomDeLaCollection  
            -f champ_1,champ_2,champ_3  
            --csv -o fichier.csv
```

Exporter la collection *nomDeLaCollection* de la base *nomBaseDonnees* dans le fichier *fichier.csv*.

Si le fichier est préexistant, il sera écrasé

Exemple:

```
mongoexport -d Exemples -c Stagiaires -f _id,nom,prenom,moy1A,niveau,option --  
csv -o C:/data/Stagiaires.csv
```



	A	B	C	D	E	F	G	H	I
1	_id	nom	prenom	moy1A	niveau	option			
2	ObjectId("	Ennaïm	Nidal	17,25	2A	Cyber Security			
3	ObjectId("	Alami	Salim	12,75	2A	FullStack			
4	ObjectId("	Dalil	Karima		1A				
5									
6									
7									
8									
9									

Utilisation de mongoimport

- On peut importer les données à partir d'un fichier JSON ou un fichier CSV
- La syntaxe est assez simple :

Importer depuis JSON

```
mongoimport -d nomBaseDonnees  
            -c nomDeLaCollection  
            fichier.json
```

Importer vers la collection *nomDeLaCollection* de la base *nomBaseDonnees*, les données du fichier *fichier.json*.

Si le fichier n'existe pas , une exception sera déclenchée,

Exemple :

```
mongoimport -d Exemples -c Stagiaires -o C:/data/stagiaires.json
```

Utilisation de mongoimport

- On peut importer les données d'un fichier CSV vers une collection
- La syntaxe est assez simple :

Importer depuis CSV

```
mongoimport -d nomBaseDonnees  
            -c nomDeLaCollection  
            --type csv,  
            --file fichier.csv
```

Exporter la collection *nomDeLaCollection* de la base *nomBaseDonnees* dans le fichier *fichier.csv*.

```
mongoimport -d database_name -c collection_name --type csv --file locations.csv --headerline
```

Si le fichier n'existe pas , une exception sera déclenchée,

Exemple:

```
mongoimport -d Exemples -c Stagiaires -type csv --file C:/data/Stagiaires.csv
```

CHAPITRE 6

Sécuriser une base de données MongoDB

1. Import / export des données avec MongoDB
2. Sécurité des accès (authentification)



Création d'un utilisateur

- Les fonctionnalités de contrôle d'accès permettent un accès par authentification aux utilisateurs existants,
- Pour créer un utilisateur sur une base de données, il faut spécifier :
 - Le login
 - Le mot de passe
 - Et le (les) rôles à lui attribuer

```
use Examples
db.createUser(
  {user: "UserAdmin",
    pwd: "password" ,
    "roles" : [
      {
        "role" : "userAdmin",
        "db" : "Examples"
      }
    ]
  }
)
```

Les rôles d'un utilisateur

Role	Explication
dbAdmin	Administrateur de la base de données
dbOwner	Combine les éléments suivants : readWrite, dbAdmin userAdmin
read	Les rôles disponibles au niveau de la base de données:
readWrite	<ul style="list-style-type: none">• read : Lire les données sur toutes les collections non-système• readWrite : en plus des privilèges du rôle "lecture« , on a le droit d'écriture
userAdmin	créer et de modifier des users et des rôles sur la base de données actuelle

Affichage des détails d'un utilisateur

- Pour afficher les détails d'un user:

```
db.getUser(user)
```

```
{
  "_id" : "Examples.UserAdmin",
  "userId" : UUID("8d6ea3c4-d054-447e-bde7-
cd5dc9de5140"),
  "user" : "UserAdmin",
  "db" : "Examples",
  "roles" : [
    {
      "role" : "userAdmin",
      "db" : " Examples "
    }
  ],
  "mechanisms" : [
    "SCRAM-SHA-1",
    "SCRAM-SHA-256"
  ]
}
```

Supprimer un utilisateur

- Pour supprimer un user:

```
db.dropUser(user)
```

```
db.dropUser("UserAdmin")  
db.getUser("UserAdmin")
```



```
switched to db Examples  
true  
nul
```

Authentification à la base de données

- Pour s'authentifier à la base avec le compte d'un utilisateur:

```
db.auth(<login>,<motDePasse>)
```

```
db.auth("UserAdmin","password")
```



```
switched to db Examples  
1
```

- On peut tester les droits d'accès déjà attribués à l'utilisateur