# **Introduction à MongoDB**

L’objectif de ce laboratoire est de s’initier à une base de données de type Document.

## L’application « GFP » (Gestion de Fiches de Personnes)

Ce programme permet de gérer des fiches de personnes. En utilisant le module de recherche, il est possible de voir les personnes d’un certain âge ainsi que ceux qui ont déjà travaillé dans un domaine précis. Ce programme a été fait en Python.



## Notes

* Pour réaliser ce projet, je vous recommande FORTEMENT de suivre les étapes en ordre.
* Si vous avez des questions, n’hésitez pas à me demander de l’aide.

## Exemple d’opérations pour MongoDB via Python

### Insérer un document

from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.blogDB # Accès à la BD « blogDB »

doc = {"author": "Fred", # Création d’un document (JSON)

"text": "My first blog post!",

"tags": ["mongodb", "python", "pymongo"]}

messages = db.messages # Accès à la collection messages

messages.insert\_one(doc) # Insertion dans la collection du document

### Supprimer un document

from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.blogDB # Accès à la BD « blogDB »

messages = db.messages

id = … # Idéalement, avoir le \_id du document à supp.

messages.delete\_one({"\_id" : id})

### Chercher, ordonnancer, limiter et retourner une collection

from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.blogDB # Accès à la BD « blogDB »

messages = db.messages

query = {} # Construction du WHERE

query["author"] = "Fred"

query["tags"] = "python"

cursor = messages.find(query).sort("date", 1).limit(5) # Recherche

results = []

for document in cursor: # Parcourir les résultats

results.append(document)

return results

### Modifier un document

from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.blogDB # Accès à la BD « blogDB »

messages = db.messages

id = … # Idéalement, avoir le \_id du document à mod.

doc = { # Nouvelle version du message

…

}

messages.replace\_one({"\_id" : id}, doc)

## Les tâches à réaliser

Ces tâches doivent être réalisées en ordre, autrement il sera difficile de tester que tout fonctionne.

### Tâche #1 : Afficher le nombre de fiches présentes

Lorsque l’on démarre le programme, il y a un petit texte qui affiche le nombre de fiches actuellement stockées dans la base de données. Il s’agit donc de terminer cette fonctionnalité. Pour se faire, vous devez compléter la méthode *PersonDAO*.*getSize()* qui se trouve dans le module dao.mongo.

Note importante : Pour cette tâche et toutes celle qui suivent, une connexion a déjà été créée pour MongoDB. Pour l’utiliser dans PersonDAO, faites comme ce qui suit. Vous n’avez donc pas besoin d’appeler verifyConnection() :

// Exemple où la BD est nommée ficheGFP et la collection personnes

Connection.client.ficheGFP.personnes.count\_documents({})

#### Notes/Énoncés :

def getSize():

nombre: int = Connection.client.clients.personnes.count\_documents({})

return nombre

On doit donner un nom de bd, ici clients.

### Tâche #2 : Ajouter une fiche personne

Il s’agit de terminer la méthode *PersonDAO.save()* afin qu’une fiche puisse être ajoutée dans la BD. Pour vérifier que cela fonctionne bien, testez-le dans une console mongo.

**IMPORTANT**: Si id est à « None », alors c’est une insertion. Sinon c’est une mise à jour. Par exemple :

if id is None :

// insert

else :

// update

**IMPORTANT** : Afin de bien fonctionner avec l’interface graphique, un document doit avoir une structure comme ceci :

doc = {

"firstName" : "…",

"lastName" : "…",

"age" : …

"jobs" : […, …]

}

#### Notes/Énoncés :

def savePerson(firstName, lastName, age, previousJobs, id = None):

doc = {

"firstName": firstName,

"lastName": lastName,

"age": age,

"jobs": previousJobs

}

if id is None:

Connection.client.clients.personnes.insert\_one(doc)

else:

Connection.client.clients.personnes.replace\_one({"\_id" : id}, doc)

### Tâche #3 : Réflexion

Selon vous, pourquoi est-il toujours préférable de conserver une date de naissance au lieu de l’âge dans une base de données?

Oui, parce que l’age ne va pas changer avec le temps. Avec la date on peut calculer l’age dynamiquement.

### Tâche #4 : Module de recherche

Il est possible de chercher selon l’âge. Il faudra donc terminer la méthode *PersonDAO.search()* en considérant que le paramètrs peut être laissé vide (len = 0).

Exemple d’utilisation de « len » :

if len(age) > 0 :

…

Les résultats doivent être triés en ordre croissant de prénom. Aussi, n’oubliez pas de limiter la taille de la collection résultante en fonction du paramètre *limit*.

#### Notes/Énoncés :

def search(age, limit):

# age est un string, vous devez donc le transformer en int comme suit : int(age)

results = []

query = {}

query["age"] = int(age)

if len(age) > 0:

cursor = Connection.client.clients.personnes.find(query).limit(limit)

for document in cursor:

results.append(document)

return results

### Tâche #5 : Supprimer une fiche

Après avoir fait une recherche, il est possible de cliquer sur le nom d’une personne afin d’accéder à sa fiche. En bas à gauche de la fiche, il est possible de la supprimer. Vous devez donc terminer la fonction *PersonDAO.* *deletePerson()* pour que la fiche soit effectivement supprimée.

#### Notes/Énoncés :

def deletePerson(id):

Connection.client.clients.personnes.delete\_one({"\_id" : id})

### Tâche #6 : Modification d’une fiche

Lorsque l’on accède à la fiche d’une personne existante, il est possible d’y apporter des modifications, puis de les sauvegarder. Vous devez maintenant modifier la méthode *PersonDAO.savePerson()* afin qu’elle puisse gérer les modifications autant que les insertions.

Lors d’une modification, le paramètre *id* n’est pas à None.

#### Notes/Énoncés :

if id is None:

Connection.client.clients.personnes.insert\_one(doc)

else:

Connection.client.clients.personnes.replace\_one({"\_id" : id}, doc)