Requêtes MongoDB – Projet RH NoSQL

Contexte :
Projet de gestion des ressources humaines utilisant MongoDB et PyMongo dans le cadre du projet académique M1 Big data et IA.
Ce document présente une sélection de requêtes MongoDB avec des explications métier concrètes.

Projet présenté par BOUWEKA BIDJADA M Essodolom et DJOHSON Daniel

1. Connexion à la base de données

In [1]:
```python
from pymongo import MongoClient
import os
from dotenv import load_dotenv

# Chargement des variables d'environnement
load_dotenv()

# Connexion
client = MongoClient(os.getenv("MONGO_URI"))
db = client["rh_database"]

employees = db["employees"]
departments = db["departments"]
leave_requests = db["leave_requests"]
```

Import des données

In [5]:
```python
import json

# Charger et insérer les données depuis un fichier JSON (facultatif)
def import_json_collection(json_file_path, collection):
    with open(json_file_path, "r", encoding="utf-8") as file:
        data = json.load(file)
    if isinstance(data, list):
        collection.insert_many(data)
    else:
        collection.insert_one(data)

# Exemple :
# import_json_collection("data/employees.json", employees)
```

Liste des requetes

1. Tous les employés

In [6]:
```python
# But : Afficher tous les employés.
for emp in employees.find():
    print(emp)

# Résultat : Itération sur tous les documents.
```

2. Trouver un employé par nom

In [8]:
```python
# But : Trouver un employé avec le nom "Koffi".

employees.find_one({"last_name": "Koffi"})


# Résultat : Renvoie le premier document correspondant.
```

3. les employés d'un département

In [9]:
```python
#  But métier :
# Obtenir la liste de tous les employés appartenant au département D001.
# Cela peut être utile pour l’analyse RH d’une entité spécifique (ex. : département informatique).

employees.find({"department_id": "D001"})

#  Lecture du résultat :
# Retourne un curseur contenant tous les documents dont le champ "department_id" est égal à "D001".
# Chaque document correspond à un employé du département concerné.
```

Out[9]:
```
<pymongo.synchronous.cursor.Cursor at 0x1ead21760d0>
```

4. Nombre total d'employés

In [11]:
```python
# But : Obtenir le nombre total d'employés.

employees.count_documents({})


# Résultat : Affiche un entier.
```

Out[11]:
```
0
```

5. Les employés par date d'embauche décroissante

In [12]:
```python
#  But métier :
# Afficher tous les employés classés par date d'embauche décroissante.
# Cela permet d’identifier les derniers employés recrutés (plus récents en haut de la liste).

employees.find().sort("hire_date", -1)

#  Lecture du résultat :
# - `employees.find()` récupère tous les employés.
# - `.sort("hire_date", -1)` trie ces documents en ordre décroissant de la date d'embauche.
#    Le `-1` signifie "décroissant".
# Résultat : Le premier document correspond à l’employé embauché le plus récemment.
```

Out[12]:
```
<pymongo.synchronous.cursor.Cursor at 0x1ead2176990>
```

6. Mise a jour du salaire d'un employé

In [ ]:
```python
# But : Mettre à jour le salaire de l'employé

employees.update_one(
    {"_id": ObjectId("64b9f6e8ecdb16d93771e9f2")},
    {"$set": {"salary": 750000}}
)

# résultat : Le champ salary est modifié pour 64b9f6e8ecdb16d93771e9f2
```

7. Suppression d'un employé

In [ ]:
```python
# But : Supprimer l'employé 64b9f6e8ecdb16d93771e9f2.

employes.delete_one({"_id": ObjectId("64b9f6e8ecdb16d93771e9f2")})


# Résultat : Document 64b9f6e8ecdb16d93771e9f2 supprimé.
```

8. Ajout d'un nouveau département

In [ ]:
```python
#  But métier :
# Ajouter un nouveau département nommé "Innovation" dans la base RH,
# avec pour responsable (manager) la personne nommée "ADJOVI".

departments.insert_one({
    "department_id": "D005",
    "name": "Innovation",
    "manager": "ADJOVI"
})

#  Lecture du résultat :
# - Insère un nouveau document dans la collection "departments".
# - Le document contient un identifiant unique "D005", un nom de département et un manager associé.
# - Si l'insertion réussit, PyMongo retourne un objet avec l’`_id` généré automatiquement (ou défini si précisé).
```

9. Liste des départements

In [ ]:
```python
#  But métier :
# Obtenir la liste complète de tous les départements enregistrés dans la base RH.
# Cela permet, par exemple, d'afficher tous les départements disponibles dans une interface ou de les analyser.

list(departments.find())

#  Lecture du résultat :
# - `departments.find()` renvoie un curseur contenant tous les documents de la collection "departments".
# - La fonction `list()` convertit ce curseur en liste Python pour faciliter l'affichage ou l'itération.
# Résultat : une liste de dictionnaires (un par département), avec leurs champs (ex: department_id, name, manager, etc.).
```

Filtres et requetes avancées

1. Employés avec salaire > 500000 et dans le département D001

In [ ]:
```python
# But : Trouver employés avec salaire > 500000 ET department_id = D001.
cursor = employees.find({
    "$and": [
        {"salary": {"$gt": 500000}},
        {"department_id": "D001"}
    ]
})
for e in cursor:
    print(e)
# Résultat : Documents filtrés selon deux conditions.
```

2. Employés avec prénom commençant par "A" (regex)

In [ ]:
```python
# But : Lister employés dont le prénom commence par "A".
cursor = employees.find({"first_name": {"$regex": "^A", "$options": "i"}})
for e in cursor:
    print(e)
# Résultat : Recherche insensible à la casse.
```

3. Employés dans une liste de départements

In [ ]:
```python
# But : Chercher employés dans les départements D001, D002 ou D003.
cursor = employees.find({"department_id": {"$in": ["D001", "D002", "D003"]}})
for e in cursor:
    print(e)
# Résultat : Documents avec department_id dans la liste.
```

4. Employés avec champ "phone" existant

In [ ]:
```python
# But : Trouver employés avec un numéro de téléphone renseigné.
cursor = employees.find({"phone": {"$exists": True}})
for e in cursor:
    print(e)
# Résultat : Documents avec champ phone.
```

5. Employés avec salaire entre 400000 et 600000

In [ ]:
```python
# But : Filtrer salariés dans une fourchette salariale.
cursor = employees.find({
    "salary": {"$gte": 400000, "$lte": 600000}
})
for e in cursor:
    print(e)
# Résultat : Documents filtrés par salaire.
```

6. Projection : afficher nom, prénom et salaire triés par salaire décroissant

In [ ]:
```python
# But : Afficher seulement last_name, first_name, salary, triés par salary décroissant.
cursor = employees.find(
    {},
    {"_id": 0, "last_name": 1, "first_name":1, "salary": 1}
).sort("salary", -1)
for e in cursor:
    print(e)
# Résultat : Liste avec les champs projetés et triée.
```

Agrégations

1. Moyenne de salaire par département

In [ ]:
```python
# But : Calculer la moyenne des salaires par département.
pipeline = [
    {"$group": {
        "_id": "$department_id",
        "avg_salary": {"$avg": "$salary"}
    }}
]
result = list(employees.aggregate(pipeline))
print(result)
# Résultat : Liste avec moyenne par département.
```

2. Top 3 employés les mieux payés

In [ ]:
```python
# But : Lister les 3 employés avec les plus hauts salaires.
pipeline = [
    {"$sort": {"salary": -1}},
    {"$limit": 3}
]
result = list(employees.aggregate(pipeline))
print(result)
# Résultat : Les 3 documents avec les salaires les plus élevés.
```

3. Nombre d’employés par fonction

In [ ]:
```python
# But : Compter combien d'employés par job_title.
pipeline = [
    {"$group": {
        "_id": "$job_title",
        "count": {"$sum": 1}
    }}
]
result = list(employees.aggregate(pipeline))
print(result)
# Résultat : Documents avec job_title et nombre d'employés.
```

4. employés avec infos département

In [ ]:
```python
# But : Afficher employés avec leur département (join departments).
pipeline = [
    {
        "$lookup": {
            "from": "departments",
            "localField": "department_id",
            "foreignField": "department_id",
            "as": "department_info"
        }
    }
]
result = list(employees.aggregate(pipeline))
print(result)
# Résultat : Chaque employé a un tableau department_info attaché.
```

5. Décomposer tableau de congés (unwind) puis compter par employé

In [ ]:
```python
#  But métier :
# Compter le nombre de jours de congé pris par employé (si on a un tableau `leave_days`).

pipeline = [
    {"$unwind": "$leave_days"},  # Décompose chaque jour du tableau
    {"$group": {
        "_id": "$employee_id",
        "total_leave_days": {"$sum": 1}
    }}
]

# 📌 Résultat :
# Liste avec chaque employé et le nombre de jours de congé utilisés.
# ⚠️ À adapter si ta structure ne contient pas de tableau `leave_days`.
```

6. Filtrer congés approuvés puis compter par employé

In [ ]:
```python
#But métier :
# Identifier les employés ayant eu le plus de congés approuvés.

pipeline = [
    {"$match": {"status": "Approved"}},
    {"$group": {
        "_id": "$employee_id",
        "approved_leaves": {"$sum": 1}
    }}
]

result = list(leave_requests.aggregate(pipeline))
print(result)

#  Résultat :
# Chaque ligne indique le nombre de demandes de congé approuvées par employé.
```

7. Projeter uniquement nom et département, trier par nom

In [ ]:
```python
#  But métier :
# Créer une liste alphabétique des employés avec leur département pour affichage ou export.

pipeline = [
    {"$project": {
        "first_name": 1,
        "last_name": 1,
        "department_id": 1,
        "_id": 0
    }},
    {"$sort": {"last_name": 1}}
]

result = list(employees.aggregate(pipeline))
print(result)

#  Résultat :
# Liste des employés avec noms triés (sans _id), utile pour un tableau ou une interface.
```

8. Agrégation imbriquée : Moyenne des salaires des employés par manager

In [ ]:
```python
#  But métier :
# Calculer la moyenne des salaires par manager (en se basant sur les départements qu'ils dirigent).

pipeline = [
    {
        "$lookup": {
            "from": "departments",
            "localField": "department_id",
            "foreignField": "department_id",
            "as": "dept_info"
        }
    },
    {"$unwind": "$dept_info"},
    {"$group": {
        "_id": "$dept_info.manager",
        "avg_salary": {"$avg": "$salary"}
    }},
    {"$sort": {"avg_salary": -1}}
]

result = list(employees.aggregate(pipeline))
print(result)

#  Résultat :
# Liste avec chaque manager et la moyenne des salaires des employés de son département.
```

In [ ]:

9. Nombre de congés par employé

In [ ]:
```python
#  But métier :
# Calculer le **nombre total de demandes de congés** effectuées par chaque employé.
# Cette analyse permet à la RH de visualiser la fréquence des demandes de congé par employé.

pipeline = [
    {"$group": {
        "_id": "$employee_id",                  # Grouper les documents par identifiant d'employé
        "total_leaves": {"$sum": 1}             # Compter le nombre de demandes (chaque document = 1)
    }}
]

list(leave_requests.aggregate(pipeline))

#  Lecture du résultat :
# - Chaque document retourné contient :
#     - `_id`: l'identifiant de l'employé (ex: "EMP001")
#     - `total_leaves`: le nombre total de demandes de congés faites par cet employé
# Résultat attendu :
# [
#   {"_id": "EMP001", "total_leaves": 3},
#   {"_id": "EMP002", "total_leaves": 1},
#   ...
# ]
```
