Compte Rendu

TP : Nouvelles Base de données

Karim Zaafrani

IUT Villetaneuse

UNIVERSITÉ SORBONNE PARIS NORD

Sommaire

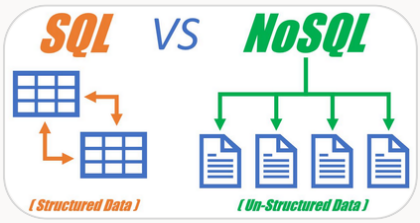
Introduction

Dans le cadre de ce projet, j'ai exploré les fonctionnalités de deux systèmes de bases de données NoSQL populaires : Redis et MongoDB. Ces deux technologies s'inscrivent dans la catégorie NoSQL, une approche moderne des bases de données conçue pour répondre aux besoins de flexibilité, de scalabilité et de performance que les systèmes relationnels (SQL) traditionnels peinent parfois à atteindre.

I. Contexte : NoSQL vs SQL

Les bases de données SQL (relationnelles) reposent sur un schéma strict de tables interconnectées, permettant des transactions complexes et garantissant l'intégrité des données. En revanche, les bases de données NoSQL sont conçues pour être plus flexibles, permettant de stocker des données semi-structurées ou non structurées. NoSQL regroupe plusieurs types de modèles, notamment :

- Clé-valeur : comme Redis, offrant une grande rapidité d'accès en mémoire pour des opérations simples.
- Document : comme MongoDB, facilitant la gestion de documents JSON-like qui sont idéals pour stocker des données semi-structurées.



Les bases SQL sont idéales pour les applications nécessitant une forte intégrité des données, comme les systèmes financiers, tandis que les bases NoSQL conviennent mieux aux applications nécessitant une grande scalabilité et des performances élevées, comme les réseaux sociaux ou les services de streaming.

II. Objectifs et Méthodologie :

Au cours de mes séances de travaux pratiques, j'ai exploré et comparé les concepts fondamentaux et les outils offerts par Redis et MongoDB pour comprendre leurs spécificités et avantages dans divers cas d'utilisation. Redis, en tant que base de données en mémoire, se distingue par sa vitesse d'accès, ce qui en fait un excellent choix pour la mise en cache. MongoDB, orientée document, est adaptée pour gérer des données plus complexes et semi-structurées avec des options avancées de persistance.

J'ai ainsi abordé des sujets clés, tels que la gestion des caches, la création d'index pour optimiser les performances, la persistance des données, ainsi que la scalabilité pour les applications de grande envergure. J'ai également expérimenté des fonctionnalités avancées, telles que les jointures simulées, la dénormalisation des données et le traitement de différents formats (CSV/JSON).

III. Présentation du Compte Rendu :

L'objectif de ce compte rendu est de documenter mon approche, les outils et méthodes employés, les résultats obtenus, ainsi que les défis rencontrés tout au long de ce projet. Cette exploration m'a permis de mieux comprendre les forces et limites de Redis et MongoDB, ainsi que leur potentiel pour des applications modernes, tout en mettant en lumière les cas d'utilisation spécifiques où chaque système excelle.

Matériels & Méthodes

Dans le cadre de ce projet, j'ai sélectionné différents outils et environnements pour assurer une compatibilité optimale avec les bases de données NoSQL, Redis et MongoDB. Ces choix m'ont permis de créer un environnement de développement complet et de faciliter la visualisation des données et l'analyse des performances. Mon objectif était de simplifier les interactions avec les bases de données et d'assurer un suivi en temps réel tout en rendant l'installation et la gestion de chaque technologie plus efficaces.

I. Système d'éxploitation :

J'ai utilisé Ubuntu, installé en local, comme système d'exploitation principal pour ce projet. Ubuntu offre un environnement robuste et adapté au développement avec des bases de données NoSQL. Il permet également une gestion efficace des installations de MongoDB et Redis, et offre un excellent support pour les outils et bibliothèques nécessaires à ce projet.

II. Environnement de développement:

J'ai utilisé Visual Studio Code comme environnement de développement intégré (IDE) pour ce projet. Visual Studio Code est léger, personnalisable et offre de nombreuses extensions utiles pour le développement avec des bases de données, telles que les extensions pour MongoDB et Redis. Il m'a permis d'écrire, tester et déboguer du code efficacement tout en facilitant la gestion de mes fichiers et configurations dans un seul espace de travail.

III. Bases de données NoSQL :

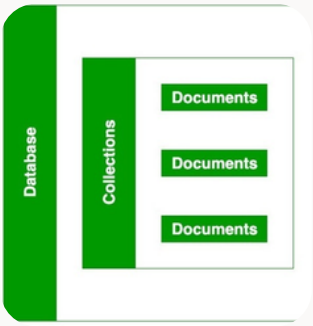
J'ai mis en pratique les outils proposés par MongoDB à travers une bases de donnes que j'ai personnellement crée

MongoDB est une base de données NoSQL orientée documents, conçue pour gérer des volumes massifs de données non structurées ou semi-structurées. Contrairement aux bases de données relationnelles qui stockent les données sous forme de tables avec des lignes et des colonnes, MongoDB utilise des collections et des documents. Les documents sont au format BSON (Binary JSON), ce qui permet de stocker des données complexes, comme des objets imbriqués, sous forme de paires clé-valeur.

- **Structure de MongoDB :**

MongoDB est une base de données orientée document qui utilise une structure flexible. Voici les éléments clés de sa structure :

- **a. Base de données :** Conteneur principal qui regroupe plusieurs collections.
- **b.Collection :** Ensemble de documents, similaire à une table dans une base de données relationnelle, sans schéma fixe.
- **c. Document :** Unité de données représentée en format BSON (Binary JSON), composée de paires clé-valeur.



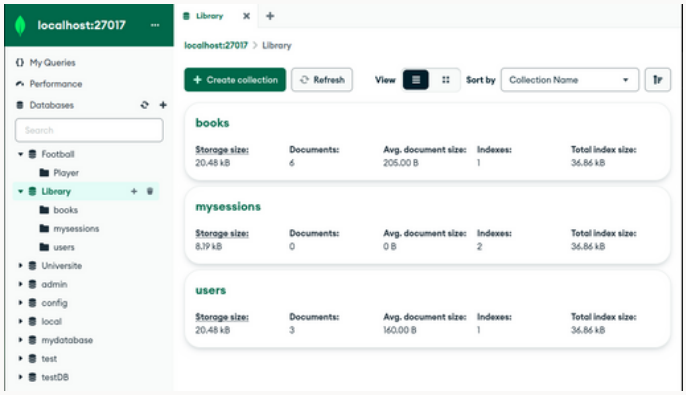
J'ai étudié les structures de données clé-valeur, les ensembles et les filtres de Bloom. Ces concepts m'ont permis de comprendre comment Redis gère les données de manière efficace et rapide.

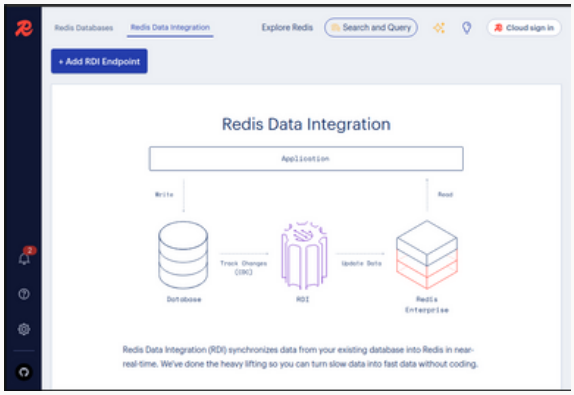
Redis est un système de gestion de bases de données en mémoire, open source, qui utilise une structure de données clé-valeur. Connu pour sa rapidité, Redis offre des temps de réponse très faibles, ce qui le rend idéal pour le caching et les applications nécessitant des performances élevées. Il prend en charge divers types de données comme des chaînes, des listes, des ensembles et des hachages. Redis permet également la persistance des données, la réplication et la haute disponibilité, en faisant un choix populaire pour les développeurs qui cherchent à optimiser les performances de leurs applications.

IV. Outils de gestion et visualisation des bases de données :

1. MongoDB Compass :

J'ai utilisé MongoDB Compass pour explorer visuellement les collections et effectuer des requêtes avancées. Cette interface graphique m'a permis de visualiser plus facilement les données sans passer par la ligne de command

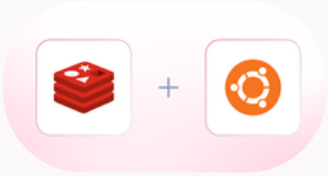




2. Redis Inisight :

Redis Insight m'a aidé à observer les structures de données en mémoire, ce qui m'a facilité le suivi des opérations et l'optimisation des requêtes.

V. Installation des Bases de données NoSQL :

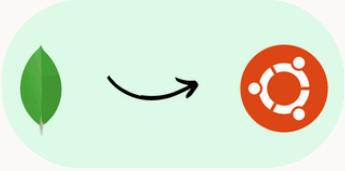


Pour instsaller mongodb en local sur mon Système Linux , j'ai suivi ces étapes :

1. Mettre à jour les dépôts de votre système: "***sudo apt update***"
2. Installer Redis: "***sudo apt install redis-server***"
3. Vérifier l'installation de Redis: "***redis-server***"

Pour instsaller mongodb en local sur mon Système Linux , j'ai suivi ces étapes :

1. Mettre à jour les dépôts de votre système: "***sudo apt update***"
2. Installer MongoDB: "sudo apt install -y mongodb-org"
3. Vérifier l'installation de MongoDB: "***sudo systemctl start mongodb***"



Installatoin de l'environement virtuel pour PYTHON :

1. Créer un environnement virtuel : "***python -m venv venv"***
2. Activez l'environnement virtuel : "***venv/bin/activate***"
3. Mettre à jour pip: "***python -m pip install --upgrade pip***"
4. Installez les dépendances nécessaires : "**pip install pymongo pandas redis pybloom_live"**

Réstultats

I. Création de la base de données :

a.création de la base de donnée Redis (Cloud) :

Dans le cadre de notre TP, j'ai choisi une solution cloud en utilisant Redis pour créer une base de données distante nommée “Karim-free-db”. Grâce à l'essai gratuit offert par Redis, j'ai pu mettre en place cette base sans frais et manipuler les données à distance de manière efficace et sécurisée. Cette configuration m'a permis de tirer parti des avantages du cloud, tels que l'accessibilité et la haute disponibilité, tout en optimisant la performance de mes opérations sur les données.


```python
import redis

r = redis.Redis(
  host='redis-10937.c82.us-east-1-2.ec2.redns.redis-cloud.com',
  port=10937,
  password='YkwYGJidRXUR3urKokCGsjNDlezkTyWK')

print ( "Vous êtes connecté à Redis !" )
```

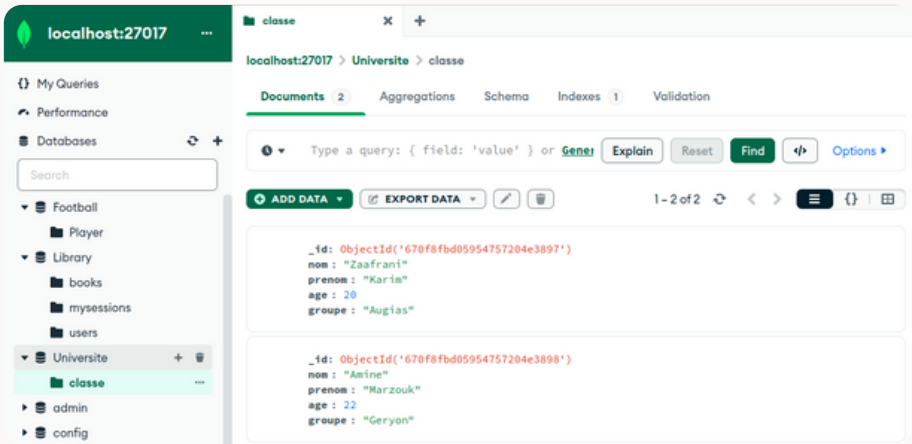
b. création de la base de donée DB

Pour MongoDB, j'ai choisi une solution locale pour créer et gérer ma base de données. La création d'une base dans MongoDB est simple et se fait par la commande use suivie du nom souhaité. MongoDB crée automatiquement la base dès qu'une collection ou un document y est ajouté. Par exemple, en entrant use maBaseDeDonnees, je prépare l'environnement pour y stocker mes données de manière structurée. Cette approche me permet de tester et manipuler les données localement, offrant flexibilité et contrôle direct sur les opérations sans dépendre d'une connexion Internet.

```python
# 1. Collection "classe"
classe = db['classe']

# Insertion de 2 étudiants

classe.insert_many([{
    "nom": "Zaafrani",
    "prenom": "Karim",
    "age": 20,
    "groupe": "Augias",
},{
    "nom": "Amine",
    "prenom": "Marzouk",
    "age": 22,
    "groupe": "Geryon",
}])
print("etudiant inséré avec succès")
```



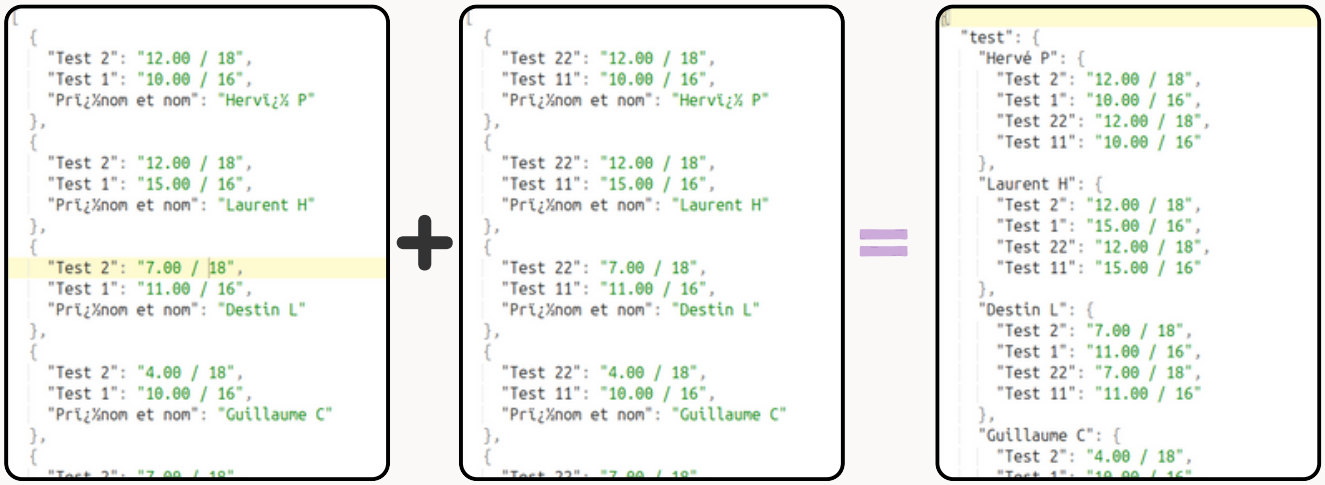
II. Jointures :

a. Défintion :

Pendant les séances de TP, nous avons adopté une démarche de dénormalisation afin de simplifier et combiner les données provenant de plusieurs tables. Après cette étape de dénormalisation, il est essentiel de réaliser des jointures entre les tables pour maintenir les liens et interactions au sein de la base de données. Ces jointures permettent de consolider les informations dispersées tout en assurant la cohérence et la continuité des relations entre les différentes entités de notre base de données.

b. Cas d'Utilisation :

Dans cet exemple, extrait d'un exercice du cours, nous avons deux fichiers CSV : "test1.csv" et "test2.csv". La principale différence entre eux réside dans les attributs nommés "test1" et "test2" dans "test1.csv", qui deviennent respectivement "test11" et "test22" dans "test2.csv". La fusion de ces deux fichiers se fera en utilisant la clé commune "Prénom et nom" pour associer correctement les informations des deux fichiers.



test1.csv test2.csv jointure.csv

c. Résultats :

À l'aide du script "jointure.py", j'ai réalisé une jointure entre les deux fichiers CSV, en utilisant la clé commune "Nom et Prénom". Le fichier résultant présente pour chaque "Prénom et nom" un objet consolidé, qui regroupe l'ensemble des attributs correspondants des deux fichiers, "test1.csv" et "test2.csv". Ainsi, chaque enregistrement consolidé contient toutes les informations des deux fichiers source, facilitant l'accès aux données combinées.

```
La jointure a été réalisée et le résultat est enregistré dans 'jointure.json'
        Test 2      Test 1        Prénom et nom     Test 22     Test 11
0   12.00 / 18  10.00 / 16             Hervé P  12.00 / 18  10.00 / 16
1   12.00 / 18  15.00 / 16           Laurent H  12.00 / 18  15.00 / 16
2    7.00 / 18  11.00 / 16           Destin L   7.00 / 18  11.00 / 16
3    4.00 / 18  10.00 / 16         Guillaume C   4.00 / 18  10.00 / 16
4    7.00 / 18  12.00 / 16            Haytem D   7.00 / 18  12.00 / 16
5    9.00 / 18   6.00 / 16          Cam Chau N   9.00 / 18   6.00 / 16
6    6.00 / 18  11.00 / 16             Sarra Z   6.00 / 18  11.00 / 16
7   11.00 / 18  11.00 / 16           Romulus L  11.00 / 18  11.00 / 16
8    8.00 / 18   5.00 / 16           Thierno D   8.00 / 18   5.00 / 16
9   13.00 / 18  13.00 / 16  Rosenthal Preston R  13.00 / 18  13.00 / 16
```

III. Conversion d’un fichier CSV en JSON :

a. Introduction:

La conversion des données de format CSV en JSON est une étape essentielle dans le processus d'intégration des données dans Redis. En effet, Redis, en tant que base de données NoSQL de type clé-valeur, utilise un format de stockage optimisé pour l’accès rapide et la manipulation de données structurées. Le format JSON, en raison de sa flexibilité et de sa compatibilité avec les structures de données hiérarchiques, s'adapte particulièrement bien à cet usage, contrairement au CSV, qui est plus rigide et orienté vers des lignes et colonnes sans structure imbriquée.

b. Cas d’utilisation :

Dans cet exemple, extrait d'un exercice du cours et réalisé avec le script Python "csv_to_json.py", l’illustration présente la conversion d’un fichier CSV en fichier JSON. À gauche, un extrait du fichier CSV d'origine est affiché, structuré en lignes et colonnes, où chaque ligne correspond à un enregistrement et chaque colonne représente un champ de données. À droite, une capture d'écran montre le fichier JSON obtenu après la conversion. Cette conversion assure une structure de données compatible avec le modèle clé-valeur de Redis, facilitant ainsi leur exploitation dans ce système.

IV. Bloom Filter avec Redis :

Dans le cadre de mon exploration de Redis, j’ai utilisé le BloomFilter pour optimiser le traitement de certaines requêtes. Les BloomFilters sont une structure de données probabiliste qui permet de vérifier si un élément est présent dans un ensemble de manière très efficace en mémoire. Bien que les Bloom filters ne garantissent pas des réponses exactes, ils sont précieux pour éviter des appels répétitifs et inutiles à une base de données plus lourde, tout en minimisant l'utilisation de mémoire.

a. Fonctionnement des Bloom Filter avec Redis :

Un Bloom filter utilise une série de hachages pour enregistrer la présence possible d'un élément dans un ensemble. Lorsqu'un élément est ajouté, plusieurs fonctions de hachage calculent différentes positions dans une structure de bits. Pour vérifier la présence d'un élément, le Bloom Filter vérifie toutes les positions calculées par les hachages pour cet élément. Si toutes les positions sont définies sur 1, il est probable que l'élément est présent. Sinon, l'élément est certainement absent. Il faut noter que cette méthode peut générer des faux positifs, indiquant qu'un élément pourrait être présent même s'il ne l'est pas, mais jamais de faux négatifs.

b. Avantages des Bloom Filter avec Redis :

- Économie de Mémoire : Les Bloom Filters consomment peu de mémoire, même pour gérer de très grands ensembles d'éléments, ce qui les rend parfaits pour des vérifications rapides sans nécessiter beaucoup de ressources.
- Amélioration des Performances : En réduisant les vérifications inutiles d'une base de données plus lourde, ils augmentent l'efficacité de l'application en filtrant les éléments de manière préliminaire.
- Simplicité et Rapidité : La nature probabiliste du Bloom Filter permet des vérifications rapides, ce qui est essentiel pour des applications à haute performance.

c. Limites et Considérations :

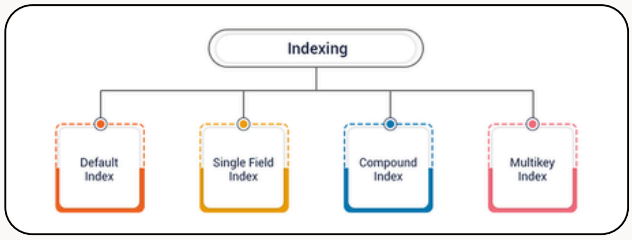
Bien que les Bloom Filters soient efficaces, ils ne sont pas parfaits. Ils peuvent indiquer de manière incorrecte qu'un élément est présent (faux positifs), mais ne provoqueront jamais de faux négatifs. Cette limitation est acceptable dans des situations où de légers risques de faux positifs sont tolérables pour l'optimisation de la performance.

V. Les indexes dans MongoDB:

Les indexes en MongoDB sont des structures essentielles qui facilitent l'optimisation des performances des bases de données, notamment en accélérant les opérations de recherche et de tri. MongoDB étant conçu pour gérer d'importants volumes de données non structurées, les indexes permettent de retrouver rapidement des documents spécifiques sans devoir parcourir l'intégralité d'une collection. Cela devient particulièrement crucial pour des bases de données de grande taille, où la rapidité d'accès aux informations est primordiale pour l'efficacité des applications.

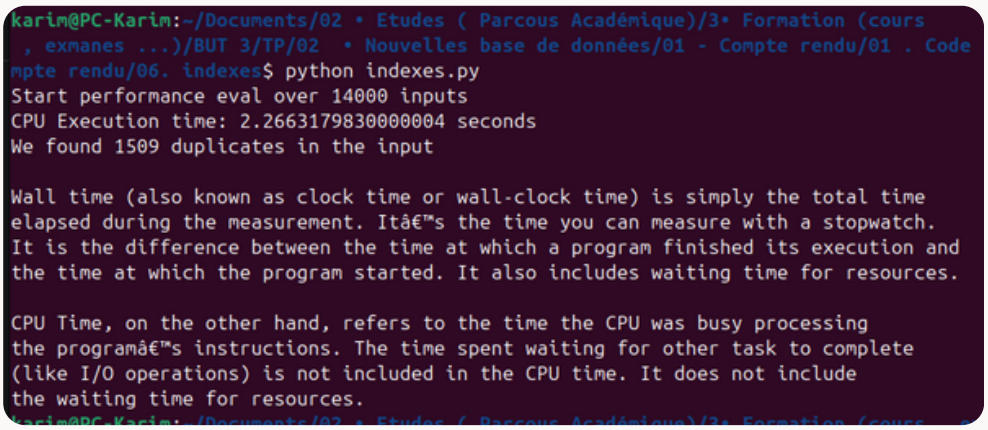
a. fonctionnement :

Les indexes fonctionnent en créant une copie structurée de certains champs d'une collection. Cette structure organise les données sous forme d'arbre B (B-tree), ce qui permet d'effectuer des recherches binaires rapides. Lorsque MongoDB doit rechercher un document, il utilise d'abord les indexes pour restreindre le nombre de documents à examiner, réduisant ainsi le temps de réponse. Les indexes peuvent être appliqués à des champs uniques ou combinés pour des recherches multicritères, et il existe plusieurs types d'indexes en MongoDB, tels que les indexes simples, composés, ou encore les indexes géospatiaux pour les données de localisation.



b. Avantages des indexes :

- Accélération des recherches : Les indexes améliorent considérablement la vitesse des recherches. En accédant directement aux données ciblées, MongoDB réduit le nombre d'opérations de lecture, rendant les recherches plus rapides.
- Amélioration du tri : Lorsqu'un index est créé sur un champ, MongoDB peut trier les documents par ce champ beaucoup plus rapidement, ce qui est particulièrement utile pour des requêtes de tri complexe.
- Performances sur les requêtes complexes : Pour des recherches qui nécessitent de filtrer ou de croiser plusieurs critères, les indexes réduisent le temps de traitement. Les indexes composés, par exemple, permettent de combiner plusieurs champs dans un même index.



c. Inconvénients des indexes:

- Impact sur les performances d'insertion : La création d'un index implique que chaque nouvelle insertion ou mise à jour doit également mettre à jour la structure de l'index. Cela peut ralentir les opérations d'écriture dans une collection, car MongoDB doit non seulement stocker le document mais aussi mettre à jour les indexes associés.
- Consommation de mémoire : Les indexes consomment de l'espace disque supplémentaire et peuvent augmenter l'utilisation de la mémoire. Pour des collections de grande taille, cela peut devenir un défi si l'espace de stockage est limité.

VI. Caching avec Redis :

Redis est particulièrement prisé pour le caching grâce à sa capacité à stocker des données en mémoire. En tant que base de données NoSQL en mémoire, Redis permet des temps de réponse extrêmement rapides, ce qui le rend idéal pour des opérations de mise en cache. Dans ce projet, j'ai exploré et mis en œuvre le caching avec Redis pour optimiser les performances de l'application en évitant des appels répétés à une base de données sous-jacente ou à une API externe.

a. Fonctionnement du Caching avec Redis :

Redis stocke les données en mémoire (RAM), ce qui signifie que les lectures et les écritures sont beaucoup plus rapides que dans des bases de données traditionnelles stockées sur disque. Le cache Redis agit comme une couche intermédiaire entre l'application et une source de données plus lente, permettant de stocker des données qui sont souvent demandées pour un accès rapide. Cela réduit la charge sur la base de données principale et améliore les performances globales de l'application.

b. Avantages du Caching avec Redis :

- Temps de réponse réduit : En stockant en cache les données fréquemment demandées, Redis permet un accès quasi-instantané aux informations, ce qui est essentiel pour les applications nécessitant de faibles latences.
- Réduction de la charge : Le caching réduit la fréquence des accès aux bases de données plus lourdes, ce qui peut améliorer la scalabilité et éviter les surcharges.
- Persistance temporaire : Grâce à la gestion avancée de l'expiration des clés, Redis permet de gérer efficacement la mémoire en expirant automatiquement les entrées du cache lorsque leur pertinence diminue.

c. Mise en œuvre et Stratégies de Caching

Pour ce projet, j'ai utilisé Redis pour mettre en cache certaines données temporaires, en utilisant des stratégies d'expiration comme TTL (Time-To-Live) pour s'assurer que le cache reste cohérent avec les données sources. Redis propose également différentes stratégies de gestion du cache (par exemple, LRU - Least Recently Used) permettant d'optimiser l'utilisation de la mémoire.

VII. Persistance et Sauvegarde dans Redis :

Pour assurer la persistance des données dans Redis, plusieurs options de sauvegarde sont disponibles. Redis, étant une base de données en mémoire, propose des mécanismes pour éviter la perte de données en cas de panne. Les deux principales méthodes sont le AOF (Append-Only File) et le BGSAVE, qui diffèrent en termes de fréquence de sauvegarde, de charge sur le serveur, et de fiabilité.

1. AOF (Append-Only File)

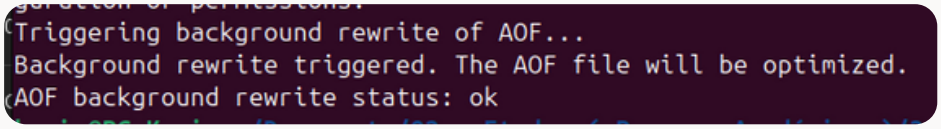
L'AOF est une méthode de persistance où chaque opération qui modifie les données (comme SET, INCR, etc.) est enregistrée dans un fichier journal. Ce fichier journal conserve les opérations sous forme de commandes Redis, qui peuvent être rejouées pour restaurer l'état de la base de données en cas de panne.

a. fonctionnement :

- Chaque commande de modification est ajoutée en fin de fichier, garantissant ainsi une chronologie complète des opérations.
- Redis peut être configuré pour synchroniser l'AOF de manière continue, toutes les secondes, ou à chaque modification.

b. Avantages de l'AOF :

- Fiabilité accrue : Les pertes de données sont réduites, surtout si le fichier est configuré pour se synchroniser à chaque seconde ou modification.
- Récupération facile : Le fichier AOF permet une reprise précise de l'état de la base en rejouant les commandes, ce qui en fait un choix sécurisé pour des environnements critiques.



c. Inconvénients de l'AOF :

- Performance : L'écriture fréquente dans le fichier journal peut affecter les performances, surtout si elle est configurée pour se synchroniser à chaque opération.
- Taille du fichier : À long terme, le fichier AOF peut devenir volumineux. Redis permet cependant de compacter ce fichier, en supprimant les opérations redondantes.

2. BGSAVE

La commande BGSAVE crée un instantané (ou "snapshot") de la base de données en mémoire et enregistre cet état dans un fichier de sauvegarde, appelé dump.rdb. Ce fichier est généré en arrière-plan, ce qui permet au serveur Redis de continuer à répondre aux requêtes.

a. fonctionnement :

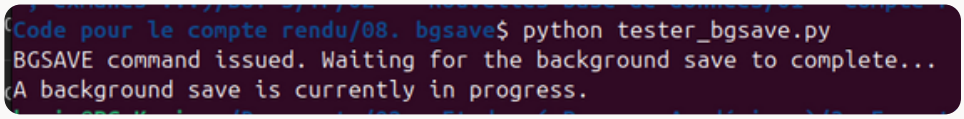
- Redis lance un processus secondaire (background) pour générer l'instantané, ce qui limite l'impact sur les performances du serveur principal.
- BGSAVE peut être déclenché manuellement ou configuré pour s'exécuter automatiquement à intervalles réguliers.

b. Avantages de BGSAVE:

- Moins de charge en écriture : Puisque les sauvegardes sont réalisées à intervalles définis, BGSAVE génère moins d'écritures fréquentes, ce qui améliore les performances générales.
- Économie d'espace : Comparé à AOF, les fichiers RDB générés par BGSAVE sont généralement plus petits car ils ne sauvegardent que l'état complet de la base à des intervalles réguliers, et non chaque opération.

c. Inconvénients de BGSAVE :

- Risque de perte de données : En cas de panne entre deux sauvegardes, toutes les modifications réalisées depuis la dernière sauvegarde sont perdues.
- Consommation de mémoire : La création de l'instantané en arrière-plan peut temporairement augmenter l'utilisation de la mémoire, ce qui peut poser problème dans les environnements à ressources limitées.



3. Comparaison AOF vs BGSAVE :

Dans ce projet, le choix entre AOF et BGSAVE dépend de la nature des données et des exigences de l'application :

- AOF est plus adapté pour des environnements nécessitant une faible tolérance à la perte de données et une récupération rapide. Cependant, son impact sur les performances doit être pris en compte.
- BGSAVE convient mieux lorsque la base de données peut tolérer une perte minimale de données et qu'il est crucial de limiter la charge d'écriture.

En pratique, Redis permet de combiner les deux méthodes pour obtenir un équilibre entre performance et sécurité.

Discussion

I. Manipulation de REDIS:

Dans cette partie, j'ai appris à manipuler Redis en utilisant des scripts Python et l'invite de commande. J'ai découvert comment interagir avec Redis depuis Python, en utilisant des commandes pour stocker, récupérer et gérer des données dans la base. Cela m'a permis de mieux comprendre la structure clé-valeur de Redis et d'explorer ses différentes fonctionnalités, comme la gestion des listes, des ensembles et des hashs.

1. importation du module Redis :

Pour chaque script python incluant redis , il faut commencer par importer la bibliothèque redis

```python
import redis

r = redis.Redis(
  host='redis-10937.c82.us-east-1-2.ec2.redns.redis-cloud.com',
  port=10937,
  password='YkwYGJidRXUR3urKokCGsjNDlezkTyWK')

print ( "Vous êtes connecté à Redis !" )
```

```python
r.set("MyName" , "Karim")

name = r.get ("name")

print ("votre Nom est : " , name)

#output : votre Nom est :  b'karim'
```

2. Les variables en Redis : (clé - valeur)

En Redis, les variables sont stockés sous forme de paires clé-valeur, ce qui permet de stocker et de retrouver facilement des informations.

3. Les dictionnaires en Redis :

Avec Redis, les dictionnaires permettent de stocker des ensembles de données organisés en plusieurs paires clé-valeur au sein d'une même clé, ce qui est très pratique pour structurer les informations.

```python
r.hset('user-session:1234', mapping={
    'name': 'Karim Zaafrani',
    "University": 'IUT Villetaneuse',
    "age": 20
})

myMap = r.hgetall('user-session:1234')
print ("Votre map est : ", myMap)

'''
Output :
Votre map est :
    {b'name': b'Karim Zaafrani', b'surname': b'Smith',
    b'company': b'Redis', b'age': b'20',
    b'University': b'IUT Villetaneuse'}
```

```python
r.set ("index" , 0)
print ("Votre index avant incrémentation ", r.get("index"))
r.incr("index")
print ("votre index après incrémentation : ", r.get("index"))

# Votre index avant incrémentation  b'0'
# votre index après incrémentation :  b'1'
```

4. Opération sur les entier en Redis :

En Redis, il existe des fonctions prédéfinies pour les entiers, comme incrémenter ou décrémenter des valeurs, ce qui est utile pour gérer des compteurs ou suivre des données numériques facilement.

5. Gestion de temps d’éxpiration :

En Redis, j'ai découvert comment définir un temps d’expiration pour les clés, ce qui permet de gérer leur durée de vie automatiquement. Cela est très pratique pour des données temporaires, comme des sessions utilisateur ou des caches, qui n’ont pas besoin de rester en mémoire indéfiniment.

```
(integer) 19
127.0.0.1:6379> ttl key
(integer) -1
127.0.0.1:6379> expire key 60
(integer) 1
127.0.0.1:6379> ttl key
(integer) 58
127.0.0.1:6379> set key 2000
OK
127.0.0.1:6379> ttl key
(integer) -1
```

6. Listes :

Avec Redis, j'ai appris à utiliser les listes, qui permettent de stocker des suites de valeurs dans un ordre précis. Cela est très utile pour gérer des files d'attente ou des historiques, car on peut facilement ajouter des éléments au début ou à la fin de la liste.

```
127.0.0.1:6379> LPUSH tableau "Rania Laffet"
(integer) 1
127.0.0.1:6379> RPUSH tableau "Emna Grami"
(integer) 2
127.0.0.1:6379> LPUSH tableau "Karim Zaafrani"
(integer) 3
127.0.0.1:6379> LRANGE tableau 0 -1
1) "Karim Zaafrani"
2) "Rania Laffet"
3) "Emna Grami"
127.0.0.1:6379> LRANGE tableau 0 0
1) "Karim Zaafrani"
127.0.0.1:6379> RPOP tableau
"Emna Grami"
127.0.0.1:6379> LRANGE tableau 0 -1
1) "Karim Zaafrani"
2) "Rania Laffet"
```

7. SET :

En Redis, les ensembles (SET), qui permettent de stocker des collections de valeurs uniques sans ordre particulier. Ils sont très utiles pour gérer des groupes d'éléments sans duplication, comme des listes de tags ou des identifiants uniques.

```
127.0.0.1:6379> SADD users "ALI"
(integer) 1
127.0.0.1:6379> SADD users "Karim"
(integer) 1
127.0.0.1:6379> SADD users "ALI"
(integer) 0
127.0.0.1:6379> SADD users "Karim"
(integer) 0
127.0.0.1:6379> SADD users "Karim"
(integer) 0
127.0.0.1:6379> SMEMBERS users
1) "Karim"
2) "ALI"
127.0.0.1:6379> SREM users "ALI"
(integer) 1
127.0.0.1:6379> SMEMBERS users
1) "Karim"
```

```
127.0.0.1:6379> SISMEMBER users "Karim"
(integer) 1
127.0.0.1:6379> SISMEMBER users "ALI"
(integer) 0
127.0.0.1:6379> SADD users2 "Rania"
(integer) 1
127.0.0.1:6379> SADD users2 "Alae"
(integer) 1
127.0.0.1:6379> SUNION users users2
1) "Alae"
2) "Karim"
3) "Rania"
```

8. SET Ordonné :

En redis, Les ensembles ordonnés (Sorted SET), stockent des collections de valeurs uniques tout en leur associant un score pour définir leur ordre. Cela est particulièrement utile pour créer des classements ou des listes triées, comme des scores de joueurs dans un jeu.

```
127.0.0.1:6379> ZADD score 10 "Antoine"
(integer) 1
127.0.0.1:6379> ZADD score 15 "Karim"
(integer) 1
127.0.0.1:6379> ZADD score 5 "EYA"
(integer) 1
127.0.0.1:6379> ZRANGE score 0 -1
1) "EYA"
2) "Antoine"
3) "Karim"
127.0.0.1:6379> ZREVRANGE score 0 -1
1) "Karim"
2) "Antoine"
3) "EYA"
127.0.0.1:6379> ZRANK score "Antoine"
(integer) 1
127.0.0.1:6379> ZREM score Antoine
(integer) 1
127.0.0.1:6379> ZRANGE score 0 -1
1) "EYA"
2) "Karim"
```

```
127.0.0.1:6379> HSET user:13 prenom "Karim"
(integer) 1
127.0.0.1:6379> HSET user:13 nom "Zaafrani"
(integer) 1
127.0.0.1:6379> HSET user:13 email "kzaafrani@gmail.com"
(integer) 1
127.0.0.1:6379> HGETALL user:13
1) "prenom"
2) "Karim"
3) "nom"
4) "Zaafrani"
5) "email"
6) "kzaafrani@gmail.com"
```

ou bien

```
127.0.0.1:6379> HMSET user:15 prenom Kylian nom mbappe age 24
OK
127.0.0.1:6379> HGETALL user:15
1) "prenom"
2) "Kylian"
3) "nom"
4) "mbappe"
5) "age"
6) "24"
```

9. HASH:

En Redis, les hachages (HASH), permettent de stocker des objets structurés sous forme de paires clé-valeur au sein d'une même clé. Ils sont idéaux pour organiser des données liées, comme les informations d'un utilisateur, tout en accédant facilement à chaque champ individuellement.

```
127.0.0.1:6379> HGET user:15 nom
"mbappe"
127.0.0.1:6379> HINCRBY user:15 age 1
(integer) 25
127.0.0.1:6379> HGET user:15 age
"25"
127.0.0.1:6379> HVALS user:15
1) "Kylian"
2) "mbappe"
3) "25"
127.0.0.1:6379> HKEYS user:15
1) "prenom"
2) "nom"
3) "age"
```

10. SYSTèME de canaux SUBSCRIBE|PUBLISH:

En Redis, j'ai appris à utiliser le système de canaux PUBLISH/SUBSCRIBE, qui permet d'envoyer des messages en temps réel entre différents clients. Cela est très pratique pour des notifications instantanées ou pour créer un système de communication entre applications, où un client peut publier un message et d'autres peuvent le recevoir immédiatement en s'abonnant au canal.

```
127.0.0.1:6379> SUBSCRIBE but3 but2
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "but3"
3) (integer) 1
1) "subscribe"
2) "but2"
3) (integer) 2
1) "message"
2) "but3"
3) "Bonjour BUT3"
1) "message"
2) "but3"
3) "Bonjour BUT2"
```

```
127.0.0.1:6379> PUBLISH but3 "Bonjour BUT3"
(integer) 1
127.0.0.1:6379> PUBLISH but3 "Bonjour BUT2"
(integer) 1
127.0.0.1:6379>
```

```
karim@PC-Karim:~$ redis-cli
127.0.0.1:6379> keys *
 1) "key"
 2) "i"
 3) "test"
 4) "score"
 5) "Rania"
 6) "tableau"
 7) "user:13"
 8) "users2"
 9) "user:11"
10) "myname"
11) "user:1"
12) "users"
13) "tab"
14) "mon_tableau"
15) "user:15"
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> keys *
(empty array)
127.0.0.1:6379[1]> SELECT 45
(error) ERR DB index is out of range
127.0.0.1:6379[1]>
```

11. les base de données redis :

En Redis, j'ai appris qu'il existe 16 bases de données que l'on peut utiliser. Par défaut, lorsque je lance "redis-cli" dans le terminal, je me trouve dans la base de données numéro 0, ce qui me permet de stocker et de gérer mes données de manière organisée au sein de cette instance.

II. Manipulation MongoDB:

1. se connecter à mongoDB:

Pour chaque script python incluant MongoDB , il faut commencer par importer la bibliothèque MongoDB

```python
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')  # Remplacer par votre URL MongoDB

# Base de données : "Université"
db = client['Universite']
```

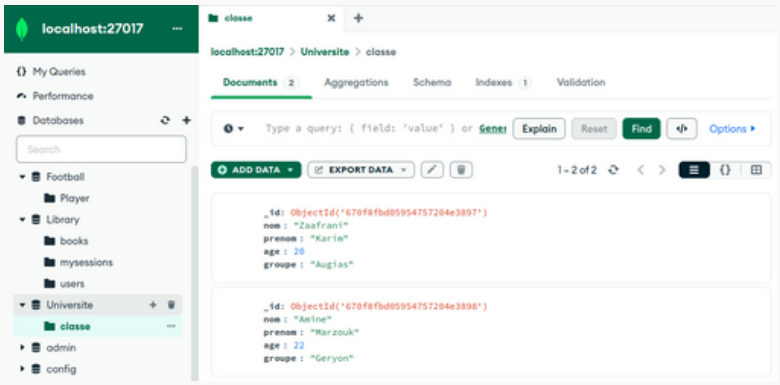
2. création de la base de donée

En MongoDB, la création d'une base de données se fait en utilisant la commande use, suivie du nom de la base de données souhaitée. Si la base de données n'existe pas déjà, MongoDB la créera automatiquement lors de l'ajout de collections ou de documents. Par exemple, en tapant use maBaseDeDonnees, on se prépare à travailler avec cette nouvelle base, prête à stocker des données.

```python
# 1. Collection "classe"
classe = db['classe']

# Insertion de 2 étudiants

classe.insert_many([{
    "nom": "Zaafrani",
    "prenom": "Karim",
    "age": 20,
    "groupe": "Augias",
},{
    "nom": "Amine",
    "prenom": "Marzouk",
    "age": 22,
    "groupe": "Geryon",
}])
print("etudiant inséré avec succès")
```



3. Les opérations CRUD avec MongoDB :

Les opérations CRUD en MongoDB incluent :

- Création : Ajouter un document avec insertOne() ou insertMany().
- Lecture : Récupérer des documents avec find() ou findOne().
- Mise à jour : Modifier des documents avec updateOne() ou updateMany().
- Suppression : Supprimer des documents avec deleteOne() ou deleteMany().

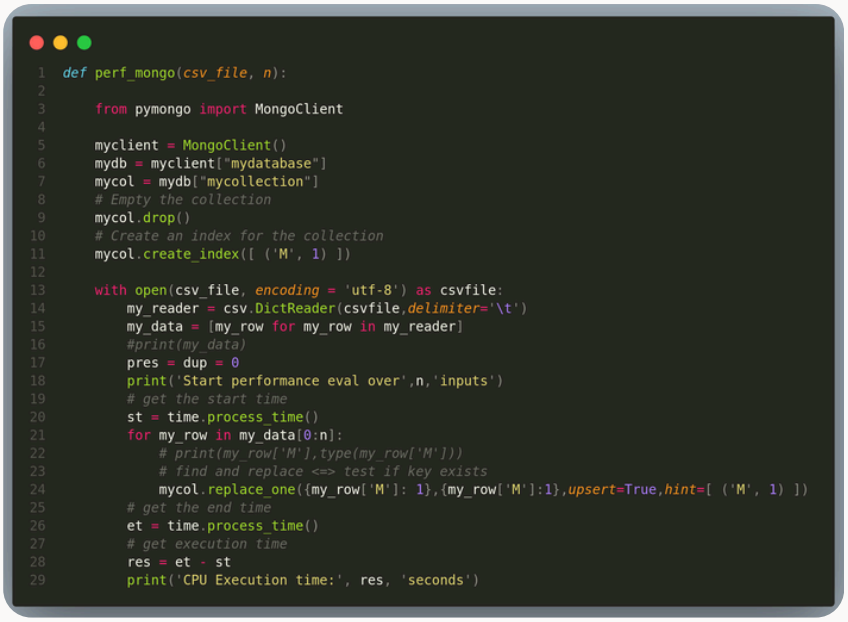
Ces opérations permettent de gérer les données dans la base de données.

```python
#afficher toute la classe
for etudiant in classe.find({}) :
    print  (etudiant)

#afficher karim
print ( classe.find_one({"nom":"Zaafrani"}))
```

4. Gestion des Index dans MongoDB

J'ai appris que les index dans MongoDB permettent d'accélérer l'accès aux documents en optimisant les requêtes. En indexant certains champs, je peux rendre les recherches bien plus rapides. Cependant, j'ai aussi compris que les index occupent de l'espace et ralentissent légèrement les écritures, car ils doivent être mis à jour à chaque modification de données.



```python
def perf_mongo(csv_file, n):

    from pymongo import MongoClient

    myclient = MongoClient()
    mydb = myclient["mydatabase"]
    mycol = mydb["mycollection"]
    # Empty the collection
    mycol.drop()
    # Create an index for the collection
    mycol.create_index([ ('M', 1) ])

    with open(csv_file, encoding = 'utf-8') as csvfile:
        my_reader = csv.DictReader(csvfile,delimiter='\t')
        my_data = [my_row for my_row in my_reader]
        #print(my_data)
        pres = dup = 0
        print('Start performance eval over',n,'inputs')
        # get the start time
        st = time.process_time()
        for my_row in my_data[0:n]:
            # print(my_row['M'],type(my_row['M']))
            # find and replace <=> test if key exists
            mycol.replace_one({my_row['M']: 1},{my_row['M']:1},upsert=True,hint=[ ('M', 1) ])
        # get the end time
        et = time.process_time()
        # get execution time
        res = et - st
        print('CPU Execution time:', res, 'seconds')
```

L'évaluation de la performance de MongoDB avec l'indexation a permis de traiter 14 000 entrées en seulement 1,97106399399 secondes de temps CPU. Ce test a également révélé la présence de 1 509 doublons dans l'ensemble des données.
Le faible temps d'exécution témoigne de l'efficacité de l'indexation dans MongoDB, particulièrement dans des cas d'insertion ou de mise à jour d'éléments lorsque les clés existent déjà. En réduisant le temps de recherche de chaque entrée grâce à l'index sur le champ M, MongoDB est capable d'améliorer la rapidité des opérations de base de données, minimisant ainsi le temps de traitement requis.
En outre, cette analyse permet de distinguer entre le temps d'horloge (wall time) et le temps CPU :

- Temps d'horloge : temps total écoulé, incluant les périodes d'attente de ressources.
- Temps CPU : temps que le processeur a effectivement passé à exécuter les instructions, sans inclure les attentes.

Ces résultats illustrent l'importance de l'indexation pour des bases de données de grande taille, démontrant que l'ajout d'index peut rendre les opérations de recherche et de gestion des doublons plus rapides et plus efficaces.

```
.venvkarim@PC-Karim:~/Documents/02 • Etudes ( Parcous Académique)/3• Formation (cours , exmanes ...)/BUT 3/
TP/02  • Nouvelles base de données/01 - Compte rendu/01 . Code pour le compte rendu/06. indexes$ python3 indexes.py
Start performance eval over 14000 inputs
CPU Execution time: 1.9710639939999999 seconds
We found 1509 duplicates in the input

Wall time (also known as clock time or wall-clock time) is simply the total time
elapsed during the measurement. Itâ€™s the time you can measure with a stopwatch.
It is the difference between the time at which a program finished its execution and
the time at which the program started. It also includes waiting time for resources.

CPU Time, on the other hand, refers to the time the CPU was busy processing
the programâ€™s instructions. The time spent waiting for other task to complete
(like I/O operations) is not included in the CPU time. It does not include
the waiting time for resources.
```

III. Conversion d'un Fichier CSV en JSON :

Dans cette section, je présente le processus de conversion d'un fichier CSV en JSON, en mettant en œuvre deux approches différentes pour atteindre cet objectif. L'un des intérêts de cette étape réside dans le besoin d'adapter les données au format JSON, plus flexible et facilement intégrable dans des systèmes NoSQL comme MongoDB et Redis.

J'ai d'abord opté pour une méthode de conversion simple, en utilisant Python pour lire les données CSV, les structurer, puis les écrire manuellement dans un fichier JSON. Cette méthode permet un contrôle détaillé de chaque étape, mais elle nécessite plus de lignes de code.

La seconde méthode fait appel à la bibliothèque pandas, qui permet une conversion bien plus rapide et avec un code plus concis, exploitant les fonctionnalités natives de la bibliothèque pour charger un fichier CSV et le convertir en JSON en quelques lignes.

```python
import pandas as pd

def csv_to_json_with_panda(csv_file):
    # étape 1: Lire le fichier CSV en utilisant pandas
    df = pd.read_csv(csv_file, encoding='latin1')

    # étape 2: Convertir le CSV en JSON
    json_data = df.to_json(orient='records', force_ascii=False)

    # étape 3: Enregistrer le JSON résultant
    file = open("test.json", "w")
    file.write(json_data)

    return json_data

csv_to_json_with_panda("test.csv")
```

Pour mettre en place le script de conversion CSV en JSON, voici les étapes pratiques que j'ai suivies :

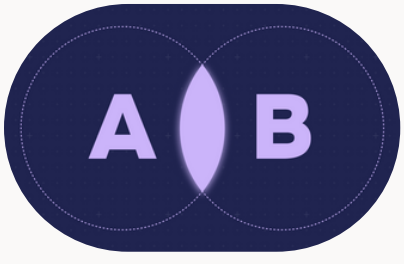
1. Préparer le script : mettre le script Python qui permet de convertir le fichier CSV en JSON dans un dossier.
2. Placer le fichier CSV : S'assurer que le fichier CSV à convertir est dans le même dossier que le script Python.
3. Exécuter le script : Lancer le script Python depuis le terminal ou l'IDE utilisé.

Une fois le script exécuté, il va automatiquement générer le fichier JSON dans le même dossier, prêt pour l'utilisation ou l'intégration dans des bases de données comme MongoDB ou Redis.
Ces étapes facilitent la conversion en un format facilement exploitable pour les systèmes NoSQL.

IV. Jointures :

Dans cette section, je présente le processus de réalisation de jointures entre des tables, en explorant deux approches distinctes pour atteindre cet objectif. Cette étape est essentielle pour combiner des données issues de plusieurs tables en une seule vue cohérente, ce qui est crucial pour des analyses complexes et pour structurer les informations de manière plus exploitable.

J'ai d'abord opté pour une méthode de jointure simple, en utilisant des requêtes SQL manuelles pour relier les tables. Cette approche offre un contrôle détaillé sur les types de jointures (INNER JOIN, LEFT JOIN, etc.) et les conditions appliquées, mais elle peut s'avérer plus longue et exige une attention particulière pour éviter des erreurs comme les doublons.



```python
import pandas as pd

# Charger les données depuis les fichiers CSV
test1 = pd.read_csv("test.csv", encoding='latin1')
test2 = pd.read_csv("test1.csv", encoding='latin1')

print(test1)
# Faire la jointure entre test1 et test2 sur la colonne "Prénom et nom"
merged_data = pd.merge(test1, test2, on="Prénom et nom")

# Enregistrer le résultat dans un fichier JSON
merged_data.to_json("jointure.json", orient="records", lines=True)

print("La jointure a été réalisée et le résultat est enregistré dans 'jointure.json'.")
print(merged_data)
```

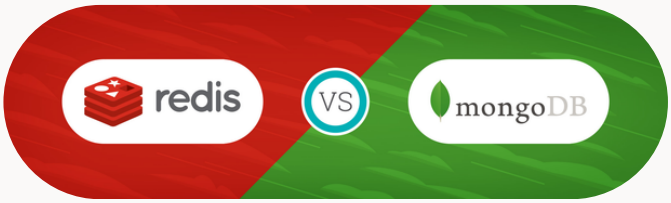
La seconde méthode fait appel à la bibliothèque pandas, qui permet d'effectuer des jointures de manière beaucoup plus rapide et concise en utilisant les fonctionnalités de fusion natives. En quelques lignes de code, il est possible de charger plusieurs tables et de les fusionner, rendant cette méthode très efficace pour manipuler des données dans un environnement Python.

Pour mettre en place le de jointure entre les tables, voici les étapes pratiques que j'ai suivies :

1. Préparer le script : mettre le script Python qui permet de convertir le fichier CSV en JSON dans un dossier.
2. Placer le 2 fichiers CSV : S'assurer que le fichier CSV à convertir est dans le même dossier que le script Python.
3. Exécuter le script : Lancer le script Python depuis le terminal ou l'IDE utilisé.

Une fois le script exécuté, il va automatiquement générer le fichier JSON dans le même dossier, prêt pour l'utilisation ou l'intégration dans des bases de données comme MongoDB ou Redis.
Ces étapes facilitent la conversion en un format facilement exploitable pour les systèmes NoSQL.

V. Redis vs MongoDB:



Redis:

- **Type :** Base de données clé-valeur en mémoire.
- **Structure :** Stocke principalement des données sous forme de paires clé-valeur, mais prend en charge divers types de données tels que les chaînes, les listes, les ensembles, les hachages, etc.
- **Cas d'utilisation :** Souvent utilisé pour la mise en cache, le stockage de sessions, l'analyse en temps réel et la messagerie pub/sub.

MongoDB :

- **Type :** Base de données NoSQL orientée document.
- **Structure :** Stocke les données dans des documents BSON (Binary JSON), qui permettent des structures de données plus complexes**.**
- **Cas d'utilisation :** Idéal pour des applications nécessitant une grande flexibilité dans le schéma, comme les systèmes de gestion de contenu, les applications web et les analyses de données.

En résumé, Redis est idéal pour des applications nécessitant une vitesse extrême et un stockage temporaire, tandis que MongoDB convient mieux pour des applications nécessitant une structure de données flexible et une persistance à long terme.

Conclusion

Ce compte rendu a permis de couvrir l'ensemble du processus de transition de données d'un modèle relationnel vers une base de données NoSQL, en passant par les étapes de conversion et d'intégration dans des formats adaptés pour des systèmes comme MongoDB et Redis. L’exercice a mis en lumière les avantages de ces technologies pour gérer des volumes importants de données de manière flexible, rapide et optimisée, tout en abordant des aspects clés tels que les formats de données (CSV et JSON) et leur compatibilité avec différents modèles de stockage.

Dans le cadre de l'utilisation des indexes dans MongoDB, j'ai pu observer directement les effets des optimisations sur la rapidité de récupération d'informations, mais aussi les contreparties en termes de temps d’insertion et de ressources de stockage. Ces expérimentations ont souligné la nécessité d’une stratégie d'indexation réfléchie pour atteindre un équilibre entre performances de lecture et d'écriture.

Enfin, cette étude m’a apporté une compréhension approfondie des mécanismes de conversion et de stockage de données dans des bases NoSQL. La comparaison entre les structures relationnelles et non relationnelles a révélé des points d’optimisation et des défis spécifiques. Ces connaissances seront précieuses pour concevoir des systèmes de gestion de données efficaces, où les choix technologiques peuvent répondre de manière ciblée aux exigences de performance et de scalabilité des applications modernes.

Références

- https://www.geeksforgeeks.org/introduction-to-redis-server/
- https://redis.io/docs/latest/develop/connect/clients/python/
- https://github.com/RedisBloom/RedisBloom
- https://stackoverflow.com/questions/2173082/what-exactly-is-nosql
- https://www.mongodb.com/fr-fr
- https://rtavenar.github.io/mongo_book/content/04_index.html#:~:text=MongoDB%20d%C3%A9finit%20les%20index%20au,dans%20une%20base%20de%20donn%C3%A9es.
- https://www.editions-eni.fr/livre/mongodb-comprendre-et-optimiser-l-exploitation-de-vos-donnees-avec-exercices-et-corriges-2e-edition-9782409046407/l-indexation-avec-mongodb
- https://lipn.univ-paris13.fr/~cerin/BD_AVANCEES/
- https://github.com/will4j/redis-cli-py
- https://www.youtube.com/watch?v=KUmXD_ubJU0https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/redis-tutoriel/