

PARADIGMES BASE DE DONNÉES NOSQL **(REDIS-MONGODB)**

Réalisé par:

Mghirbi Alae

Supervisé par:

Morad M'lik

10 / 2025

TABLE DES MATIÈRES

- 01** Introduction générale:
- 02** Matériels et Méthodes
- 03** Résultats
 - Résultats de connexion à la base de données
 - Résultats de création de la base de données
 - Le Client side caching
 - La persistance des données
 - Les Pipelines
 - Transformation de csv en JSON :
 - Jointure de deux JSON
 - optimisation des recherches
- 04** Discussion
 - Rdis
 - MongoDB
- 05** Conclusion
- 06** Ressources

INTRODUCTION

Après avoir attentivement lu et assimilé l'intégralité du contenu du cours, j'ai pu élaborer cette introduction:

https://lipn.univ-paris13.fr/~cerin/BD_AVANCEES/SQL_NoSQL_NewSQL.pdf

Contexte des bases de données NoSQL et NewSQL

Avec l'augmentation des volumes de données et des besoins des applications modernes, la gestion des bases de données a évolué vers des approches plus flexibles et scalables. Les bases de données relationnelles traditionnelles (SQL) ont montré des limitations en matière de flexibilité et de scalabilité pour des applications distribuées et massivement parallèles. En réponse, les bases de données NoSQL (Not Only SQL) ont émergé, offrant des modèles plus souples (documents, paires clé-valeur, graphes) pour traiter des données non structurées ou semi-structurées, tout en assurant des performances élevées et une scalabilité horizontale. Les bases de données NewSQL ont également été développées, cherchant à combiner la scalabilité des NoSQL avec les propriétés ACID des bases relationnelles.



Différences entre SQL, NoSQL et NewSQL

1. SQL (Bases de données relationnelles traditionnelles)

Les bases de données SQL se caractérisent par une structure de données rigide, avec des tables définies selon un schéma pré-déterminé. Elles garantissent une consistance forte grâce aux transactions ACID, mais leur scalabilité est principalement verticale, nécessitant l'ajout de puissance à un serveur unique, ce qui peut poser problème à grande échelle [1].

2. NoSQL

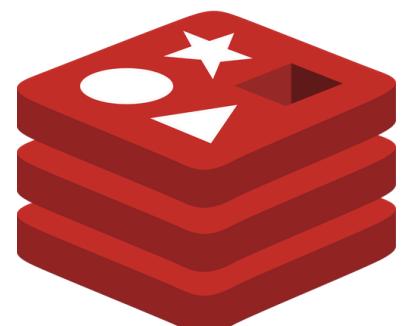
Les bases NoSQL, comme MongoDB et Redis, se distinguent par leur flexibilité et leur capacité à évoluer horizontalement. MongoDB est une base orientée documents qui permet de stocker des données sous format JSON-like, tandis que Redis est un magasin clé-valeur en mémoire utilisé pour des performances ultra-rapides dans des systèmes de cache, files d'attente ou gestion de sessions . NoSQL permet de mieux s'adapter aux environnements nécessitant des structures de données variables et une distribution géographique des données[1].

3. NewSQL

NewSQL, quant à lui, combine la scalabilité horizontale du NoSQL avec les transactions ACID du SQL. CockroachDB en est un exemple emblématique : elle répartit les données de manière efficace sur plusieurs nœuds, tout en garantissant des transactions cohérentes et une tolérance aux pannes[1] .

Parmi ces technologies, deux bases de données NoSQL majeures se démarquent :

- Redis : Utilisé principalement comme un magasin de données en mémoire, Redis est reconnu pour sa rapidité exceptionnelle et son efficacité dans la gestion des caches, des files d'attente, et des sessions. Il est largement adopté pour les systèmes nécessitant une haute performance et une latence ultra-faible.





- **MongoDB** : Célèbre pour sa flexibilité, MongoDB est une base de données orientée documents qui permet de stocker des données sous un format JSON-like. Elle est souvent utilisée dans des applications nécessitant une grande variabilité dans les structures de données et une évolutivité horizontale.

Ces bases de données jouent un rôle essentiel dans les architectures modernes, où la gestion des données doit être à la fois flexible, scalable, et capable de répondre aux exigences des applications en temps réel. Redis et MongoDB, en particulier, ont su s'imposer comme des choix incontournables dans différents secteurs, que ce soit pour les applications web, les jeux en ligne, ou encore les systèmes IoT, et seront examinés en détail dans les sections suivantes de ce rapport.

MATÉRIELS ET MÉTHODES

Matériel

Pour mener cette expérience, j'ai utilisé ma machine personnelle, équipée d'une machine virtuelle pour exécuter Redis et des services cloud pour gérer les bases de données Redis et MongoDB. L'environnement de travail a été configuré pour permettre à la fois l'installation locale et l'utilisation des plateformes de gestion en ligne.



Méthodes

1. Redis

Pour la base de données Redis, j'ai utilisé deux approches afin d'explorer différentes configurations et options de gestion.

-Redis en Local :

J'ai d'abord installé Redis sur une **machine virtuelle** en utilisant une image **Debian**. Cette configuration locale m'a permis de tester le fonctionnement de Redis en mode hors ligne, sans dépendance des services cloud.



-Redis Cloud avec Azure :

Pour explorer l'intégration de Redis dans un environnement cloud, j'ai utilisé **Redis Cloud** via Azure. Cela m'a permis de tester la scalabilité et la gestion des bases de données Redis dans un environnement en ligne.

-Outils de Visualisation et de Connexion :

Afin de faciliter la manipulation et la visualisation des données dans Redis, j'ai utilisé **Redis Insight**, un outil graphique qui permet de voir les données stockées en temps réel. En outre, j'ai employé le client **Python** pour établir une connexion avec Redis et exécuter des opérations de manipulation de données.



2. MongoDB

Pour la base de données MongoDB, j'ai choisi une approche basée sur les services cloud afin d'exploiter les fonctionnalités avancées de MongoDB en ligne.

-MongoDB Atlas :

J'ai utilisé **MongoDB Atlas** pour créer et héberger ma base de données MongoDB dans le cloud. **MongoDB Atlas** offre une interface de gestion intuitive et de nombreuses options de configuration, ce qui m'a permis de déployer une base de données sans avoir besoin de gérer l'infrastructure sous-jacente.



-Outils de Visualisation et de Connexion :

Pour visualiser et gérer ma base de données MongoDB, j'ai utilisé **MongoDB Compass**, qui fournit une interface graphique pour naviguer dans les collections et exécuter des requêtes. Pour la manipulation et l'interaction programmée avec la base, j'ai utilisé le client **Python**, qui m'a permis de créer des scripts pour insérer, mettre à jour et lire les données de la base MongoDB de manière efficace.



RESULTATS

Dans cette section, j'expose les résultats obtenus en manipulant et en comparant les bases de données MongoDB et Redis. Pour chaque base de données, j'ai créé une structure de données similaire, effectué des opérations courantes (insertion, mise à jour, lecture, et suppression) et analysé les performances et la facilité de manipulation.

Résultat de connexion et visualisation: Redis:

Après avoir installé Redis, nous avons pu nous connecter à la base de données, à la fois en local et en ligne. Voici les résultats de la connexion pour les deux cas :

Version en ligne:

```
C:\> Users\alaem> OneDrive\Bureau\3BUT\BDD> try.py > ...
1 import redis # type: ignore
2
3 r = redis.Redis(
4     host='redis-14277.c56.east-us.azure.redns.redis-cloud.com',
5     port=14277,
6     password='5f4BL5tx413OrZkZgPg5HyaNi796nGS3')
7
8 response = r.ping()
9
10 # Afficher la réponse
11 if response:
12     print("PONG") # Si la réponse est True
13 else:
14     print("Pas de réponse")
15
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[Running] python -u "c:\Users\alaem\OneDrive\Bureau\3BUT\BDD\try.py"
PONG

[Done] exited with code=0 in 1.094 seconds
```

Version locale:

```
[ etudiant@debian12-docker (30) 0 jobs 14:07:06
~/Bureau $ redis-cli
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> ]
```

Après avoir créé une base de données avec le cloud et installé Redis Insight, j'ai réussi à me connecter à cette base pour visualiser les résultats. Voici les informations reçues lors de la connexion et de l'accès à la base

The screenshot shows the Redis Insight web interface. At the top, there are tabs for 'Redis Databases' and 'Redis Data Integration'. On the left, there's a sidebar with icons for adding a database, a refresh button, and a 'Try Redis Cloud' button. The main area displays a table with one row:

Database Alias	Host:Port	Connection Type	Modules	Last connection
alaе-free-db	redis-14277.c56.east-us.azure.redns.redis... redis-14277.0.ec2.us-east-1.amazonaws.com:6379	Standalone	Redis	2 days ago

MongoDB:

Suite à l'installation MongoDB, je me suis connecté à la base de données en utilisant le client Python. Voici les résultats obtenus :

```
mongoDB > 🛡 connection.py > ...
1  from pymongo.mongo_client import MongoClient
2  from pymongo.server_api import ServerApi
3  uri = "mongodb+srv://alaemghirbi2003#alaе.8wk9s.mongodb.net/?retryWrites=true&w=majority&appName=alaе"
4  # Create a new client and connect to the server
5  client = MongoClient(uri, server_api=ServerApi('1'))
6  # Send a ping to confirm a successful connection
7  try:
8      client.admin.command('ping')
9      print("Pinged your deployment. You successfully connected to MongoDB!")
10 except Exception as e:
11     print(e)
```

PROBLEMS 22 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Données importées avec succès dans Redis.

[Done] exited with code=0 in 1.242 seconds

[Running] python -u "c:\Users\alaem\OneDrive\Bureau\3BUT\BDD\mongoDB\connection.py"
Pinged your deployment. You successfully connected to MongoDB!

[Done] exited with code=0 in 11.251 seconds

Pour visualiser ma base de données, j'ai utilisé MongoDB Compass, qui m'a permis d'accéder à une interface intuitive après m'être connecté. Cette interface m'a facilité l'exploration des collections, la visualisation des documents et l'exécution de requêtes pour analyser les données.

The screenshot shows the MongoDB Compass interface connected to the database `alae.8wk9s.mongodb.net`. The left sidebar lists connections, and the main area displays the `admin` database. Other databases listed are `config`, `local`, and `sample_mflix`. Each database entry shows storage size, collection count, and index count.

Database	Storage size	Collections	Indexes
<code>admin</code>	0 B	0	0
<code>config</code>	-	-	-
<code>local</code>	-	-	-
<code>sample_mflix</code>	97.50 MB	6	10

création de bases de données : Redis:

Pour Redis, j'ai créé une base de données destinée à la gestion des employés. Dans cette structure, chaque nom d'employé est utilisé comme clé, et ses informations sont stockées en tant que valeurs associées. voila la base que j'ai réussi a créer

The screenshot shows the Redis UI interface. On the left, a sidebar provides navigation and monitoring tools. The main area displays a list of keys under the database `alae-free-db`. A detailed view of the `Emna` key is shown on the right, displaying its fields and values.

Type	Name	Length	Value
HASH	Emna	No limit	128 B
HASH	Karim	No limit	128 B
HASH	Yassine	No limit	136 B
HASH	David	No limit	240 B
HASH	Alice	No limit	240 B
HASH	Rania	No limit	144 B
HASH	Alae	No limit	144 B
STRING	compteur	No limit	56 B
HASH	Hanen	No limit	144 B
JSON	resultatJointure	No limit	1 KB
HASH	Eve	No limit	240 B
HASH	Charlie	No limit	248 B

Key Details for Emna:

Field	Value
age	25
departement	Marketing
performance	88
date_embauche	2022-03-12

MongoDB:

Il est possible de créer une base de données manuellement, mais dans mon cas, j'ai choisi de télécharger un jeu de données déjà disponible sur Atlas pour travailler avec. Il s'agit d'une base de données existante.

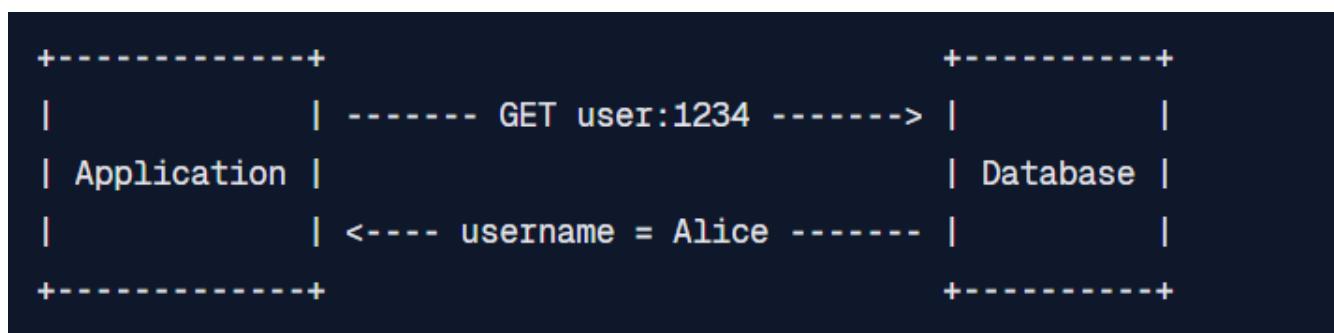
Voilà ce que j'ai eu comme résultat:

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar titled 'My Queries' with a 'CONNECTIONS' section listing 'alae.8wk9s.mongodb.net'. Below it is a tree view of databases and collections: 'sample_mflix' is expanded, showing 'comments', 'embedded_movies', 'movies', 'sessions', 'theaters' (which is selected), and 'users'. Other databases like 'sample_airbnb', 'sample_geospatial', and 'sample_weatherdata' are also listed. The main area is titled 'alae.8wk9s.mongodb.net > sample_mflix > theaters'. It shows 'Documents 1.6K' and 'Indexes 2'. A search bar at the top says 'Type a query: { field: 'value' } or Generate query'. Below it are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. A table lists five documents, each with an '_id' (ObjectId), 'theaterId' (1000, 1003, 1008, 1004, 1002), and a 'location' object. Navigation buttons at the bottom right indicate 1-25 of 1564 documents.

Le Client side caching

Le caching côté client consiste à stocker des données localement sur le client (application ou navigateur) pour éviter des requêtes répétées au serveur. Cela accélère les opérations en réduisant la latence et la charge serveur.

Lorsqu'une application accède fréquemment aux mêmes données, celles-ci sont mises en cache, permettant une récupération rapide lors de futures demandes sans contacter la base de données.[4]



[<https://redis.io/docs/latest/develop/reference/client-side-caching/>]

Lorsqu'un cache côté client est utilisé, l'application stocke les réponses des requêtes populaires directement dans la mémoire de l'application, afin de pouvoir réutiliser ces réponses ultérieurement, sans avoir à contacter à nouveau la base de données.

Application	(No chat needed)	Database
Local cache		
user:1234 =		
username		
Alice		

Redis:

J'ai implémenté ce code pour faire fonctionner le cache côté client, et nous remarquons que lors du premier appel, les données sont lues depuis le serveur. En revanche, lors du deuxième appel, les données sont récupérées à partir du cache. Bien que nous ne puissions pas vérifier cela physiquement, l'activation du cache à l'aide de la configuration du cache nous garantit ce comportement.

```

26 cache = r.get_cache()
27 cache.delete_by_redis_keys(["alae"])
28
29 r.hget("alae", "name") # Read from the server
30 r.hget("alae", "name") # Read from the cache
31
32

```

MongoDB:

Ce code met en place un système de mise en cache pour les résultats d'une requête MongoDB à l'aide de diskcache. Lorsqu'une donnée est demandée, le cache vérifie si elle est déjà stockée. Si oui, elle est retornée immédiatement. Sinon, la fonction interroge la base de données, stocke le résultat dans le cache pour les futures demandes, et le retourne. Cela réduit le nombre de requêtes à la base de données et améliore les performances de l'application.

```

from diskcache import Cache
import connexion
client=connexion.connexion()
db=client['ma_base_de_donnees']
collection = db['ma_collection']
cache = Cache('cache_dir') # Répertoire pour stocker le cache sur le disque
@cache.memoize()
def get_donnees():
    return db.collection.find_one()

# Utilisation de la fonction
result = get_donnees()

```

Persistante des données

Redis persistante

J'ai réussi à assurer la persistance de mes données dans Redis, ce qui me permet de récupérer les informations en cas de panne grâce à trois méthodes différentes. Tout d'abord, j'ai mis en place la persistance **RDB (Redis Database)**, qui prend des instantanés ponctuels de l'ensemble de mes données à des intervalles spécifiés. Ces snapshots capturent l'état complet de la base de données et facilitent ainsi la restauration en cas de besoin. Avec **RDB**, j'utilise les commandes **SAVE** et **BGSAVE**. La commande **SAVE** exécute une sauvegarde synchrone, bloquant le serveur jusqu'à la fin de la sauvegarde, tandis que **BGSAVE** effectue cette opération en arrière-plan, permettant au serveur de rester opérationnel. Grâce à ces options, mes données sont régulièrement enregistrées de manière durable et peuvent être restaurées à tout moment si nécessaire. voila le fichier dump.rdb qui me permettra de recuperer mes donnés en cas de panne

```
atac@atac:~/Desktop$ redis stable  
GNU nano 6.2  
REDIS0012*      redis-ver^E7.4.0*  
redis-bits@*^Ectime*0*f*^Hused-mem* @^R@*^Haof-base* @*^E@^Ntutorialspos
```

MongoDB

Sauvegarde dans MongoDB Atlas

MongoDB Atlas offre des sauvegardes automatiques, ce qui signifie que vous n'avez généralement pas besoin de faire des sauvegardes manuelles. Voici quelques points à considérer :

Sauvegardes Automatiques : MongoDB Atlas crée des sauvegardes automatiques tous les jours. Ces sauvegardes sont stockées dans le cloud et peuvent être restaurées facilement.

Backup

Database Name	Last Snapshot (UTC)	Next Snapshot (UTC)	Oldest Snapshot (UTC)	Snapshot Region	PIT
---------------	---------------------	---------------------	-----------------------	-----------------	-----

>>>Les sauvegardes s'affichent ici, mais je dispose uniquement de la version gratuite de MongoDB Atlas, qui ne prend pas en charge les sauvegardes.

Les Pipelines:

Redis :

Les pipelines dans Redis sont utilisés pour exécuter plusieurs commandes de manière consécutive sans attendre de réponse immédiate pour chacune. Cette technique permet d'envoyer un lot de commandes en une seule requête réseau, réduisant les allers-retours entre le client et le serveur et améliorant ainsi les performances, surtout pour des opérations répétitives.

```
pipeline.py > ...
4     host='redis-14277.c56.east-us.azure.redns.redis-cloud.com',
5     port=14277,
6     password='5f4BL5tx413OrZkZgPg5HyaNi796nGS3')
7     pipe = r.pipeline()
8     pipe.set('pip1', 5)
9     pipe.set('pip2', 18.5)
10    pipe.set('pip3', "hello world!")
11
12    print(pipe.execute())

[Running] python -u "c:\Users\alaem\OneDrive\Bureau\3BUT\BDD\pipeline.py"
[True, True, True]
```

MongoDB :

Les pipelines dans MongoDB font référence à un ensemble d'étapes de traitement de données en série, particulièrement utilisés dans le **framework d'agrégation**. Chaque étape du pipeline applique une opération spécifique sur les données, filtrant, transformant ou résumant les documents de manière séquentielle pour obtenir un résultat final. Ce processus est efficace pour les analyses et les rapports sur des ensembles de données complexes.

Exemple:

Prenons un exemple dans lequel nous souhaitons calculer la somme des "likes" pour chaque catégorie, mais uniquement pour les documents ayant plus d'un "like". Ce pipeline utilisera l'étape \$match pour filtrer les documents et l'étape \$group pour regrouper les données et effectuer le calcul.

```
Atlas atlas-8etut9-shard-0 [primary] exercicecli> db.posts.aggregate([
...   // Stage 1: Only find documents that have more than 1 like
...   {
...     $match: { likes: { $gt: 1 } }
...   },
...   // Stage 2: Group documents by category and sum each categories likes
...   {
...     $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
...   }
... ])
[ { _id: 'news', totalLikes: 2 }, { _id: 'Event', totalLikes: 5 } ]
Atlas atlas-8etut9-shard-0 [primary] exercicecli>
```

Transformation de csv en JSON :

Redis :



En m'inspirant du code de l'exercice 2.5, j'ai recréé mon propre fichier CSV, ainsi qu'un code pour le convertir en JSON. Le résultat final est le suivant :

Fichier CSV initial :

A screenshot of a Microsoft Excel spreadsheet titled "employes_data.xlsx". The spreadsheet has a header row with columns labeled A through F. The data rows below contain information for six employees: Alice, Bob, Charlie, David, and Eve. The columns represent Nom, Performance, Engagement, Âge, Date de début, and Actif respectively. The data is as follows:

	A	B	C	D	E	F
1	Nom	Performance	Engagement	Âge	Date de début	Actif
2	Alice	85	90	30	2020-01-15 00:00:00	VRAI
3	Bob	78	88	25	2019-06-12 00:00:00	FAUX
4	Charlie	92	95	35	2021-03-08 00:00:00	VRAI
5	David	67	75	40	2018-11-23 00:00:00	VRAI
6	Eve	90	85	28	2022-05-19 00:00:00	FAUX
7						
8						
9						

résultat obtenu après transformation en JSON:

A screenshot of a terminal window displaying the generated JSON output. The JSON structure is as follows:

```
1  {
2      "employees": [
3          "Alice": {
4              "Performance": "85",
5              "Engagement": "90",
6              "Age": "30",
7              "Date de début": "2020-01-15 00:00:00",
8              "Actif": "VRAI"
9          },
10         "Bob": {
11             "Performance": "78",
12             "Engagement": "88",
13             "Age": "25",
14             "Date de début": "2019-06-12 00:00:00",
15             "Actif": "FAUX"
16         },
17     ]
18 }
```

```

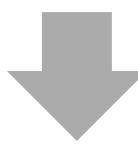
17         "Charlie": {
18             "Performance": "92",
19             "Engagement": "95",
20             "Age": "35",
21             "Date de début": "2021-03-08 00:00:00",
22             "Actif": "VRAI"
23         },
24         "David": {
25             "Performance": "67",
26             "Engagement": "75",
27             "Age": "40",
28             "Date de début": "2018-11-23 00:00:00",
29             "Actif": "VRAI"
30         },
31         "Eve": {
32             "Performance": "90",
33             "Engagement": "85",
34             "Age": "28",
35             "Date de début": "2022-05-19 00:00:00",
36             "Actif": "FAUX"
37         }
38     }
39 }
```

MongoDB :

Pour MongoDB, voici le fichier CSV de départ que j'utilise pour l'importation et le travail sur les données :

```

mongoDB >  csv1.csv >  data
1   id,name,age,details,scores
2   1,John,25,"{""city"":""New York""}, ""position"":""Developer""}","[85,90,78]"
3   2,Alice,30,"{""city"":""San Francisco""}, ""position"":""Manager""}","[88,91,92]"
4   3,Bob,22,"{""city"":""Chicago""}, ""position"":""Intern""}","[77,89,85]"
5
```



résultat obtenu après transformation en JSON:

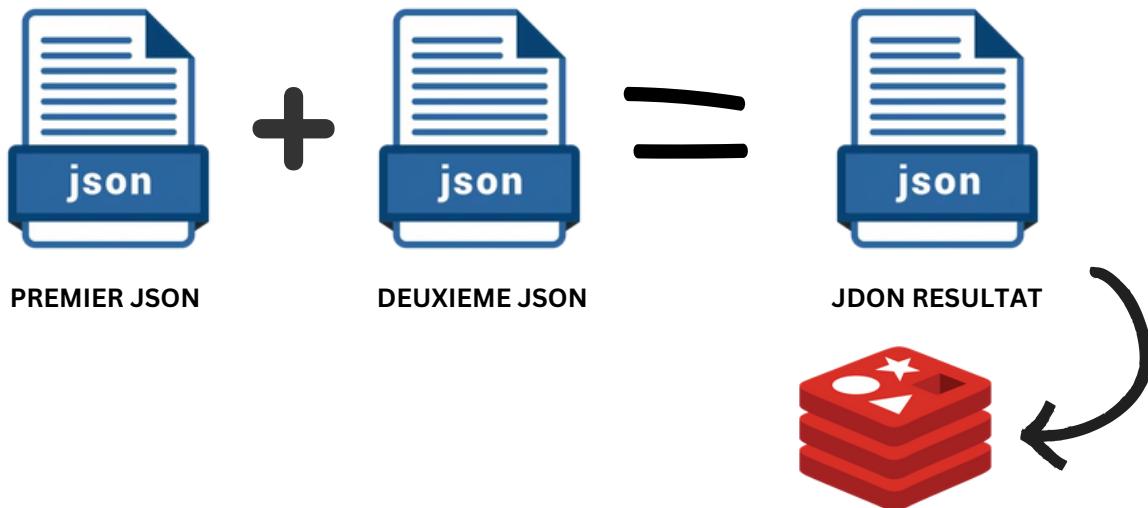
```

1   {
2     "25": {
3       "id": "1",
4       "name": "John"
5     },
6     "30": {
7       "id": "2",
8       "name": "Alice"
9     },
10    "22": {
11      "id": "3",
12      "name": "Bob"
13    }
14  }
```

```

6     "30": {
7       "id": "2",
8       "name": "Alice"
9     },
10    "22": {
11      "id": "3",
12      "name": "Bob"
13    }
14  }
```

Jointure de deux JSON : Redis :



À partir des deux fichiers que j'ai préalablement convertis en JSON, j'ai pu réaliser une jointure et obtenir les résultats suivants :

```
departement.csv > data
1 Nom;Departement;Position;Manager
2 Alice;Informatique;Développeur;M. Dupont
3 Bob;RH;Recruteur;Mme. Martin
4 Charlie;Marketing;Analyste;Mme. Pierre
5 David;Ventes;Commercial;M. Jean
6 Eve;Informatique;Chef de projet;Mme. Durand
7
```

```
employes_data.csv > data
1 Nom;Performance;Engagement;age;Date de debut;Actif
2 Alice;85;90;30;2020-01-15 00:00:00;VRAI
3 Bob;78;88;25;2019-06-12 00:00:00;FAUX
4 Charlie;92;95;35;2021-03-08 00:00:00;VRAI
5 David;67;75;40;2018-11-23 00:00:00;VRAI
6 Eve;90;85;28;2022-05-19 00:00:00;FAUX
7
```

Avec ces deux fichiers CSV de départ, la jointure s'effectuera sur l'attribut commun nom de l'employé. Voici une description de chaque fichier et de l'opération de jointure :

1. Fichier CSV Employés :

- Contient les informations de base des employés.
- Attributs : nom, departement, position, et manager.

2. Fichier CSV Performances :

- Contient des informations de performance et de carrière des employés.
- Attributs : nom, performance, age, et date_d_embauche.

3. Résultat de la jointure :

- En joignant ces deux fichiers sur l'attribut nom, on obtient un fichier enrichi combinant les informations d'identité, de poste, et de performance pour chaque employé.

voila le résultat de la jointure visualiser sur redis insight :

The screenshot shows the Redis Insight interface. On the left, a sidebar lists keys: David, Alice, resultat_jointure, Charlie, Eve, user-session:t23, and Bob. The resultat_jointure key is highlighted and shows a value of 248 B. On the right, a detailed view of the resultat_jointure key is shown in a table:

Field	Value
Nom	Charlie
Performance	92
Engagement	95
age	35
Date de debut	2021-03-08 00:00:00
Actif	VRAI
Departement	Marketing
Position	Analyste

optimisation des recherches :

Résultats de l'utilisation des Filtres de Bloom (Redis)

Après avoir déployé des filtres de Bloom dans Redis, on observe :

- Vitesse de recherche : Amélioration significative avec vérification rapide de l'appartenance.
- Réduction des requêtes : Diminution des requêtes non nécessaires vers des bases secondaires.
- Mémoire optimisée : Faible utilisation de mémoire pour des ensembles de données volumineux

Après le déploiement du filtre de Bloom, ma base de données répond de manière aussi rapide et me signale les faux positifs pour les éléments qui n'existent pas, optimisant ainsi son utilisation. Voici le résultat après avoir interroger le bloom filter deux éléments 1 et 3 :

```
● (.venv) alae@alae-linux:~/Desktop/redis-py$ /home/alae/Desktop/redis-py/.venv/bin/python /home/alae/Desktop/redis-py/connection.py
element1 exists: 1
element3 exists: 0
○ (.venv) alae@alae-linux:~/Desktop/redis-py$ 
```

il est inutile de reinterroger la base sur l'élément 3 puisqu'on est positif que celui-là n'existe pas

Résultats de l'utilisation des Indexes (MongoDB)

Après avoir déployé les indexées sur Redis, on observe :

- Accélération des requêtes complexes : Réduction notable du temps de réponse pour les recherches avancées.
- Précision des résultats : Résultats exacts sans erreurs d'appartenance.
- Consommation de ressources : Augmentation de la mémoire et du stockage requis pour maintenir les index.

```
Atlas atlas-8etut9-shard-0 [primary] sample_mflix> db.movies.getIndexes()
[
  {
    v: 2,
    key: { _id: 1 },
    name: '_id_'
  },
  {
    v: 2,
    key: { _fts: 'text', _ftsx: 1 },
    name: 'title_text',
    weights: { title: 1 },
    default_language: 'english',
    language_override: 'language',
    textIndexVersion: 3
  }
]
```

Cette capture d'écran présente l'index que j'ai conçu, qui améliore l'efficacité de la recherche de titre de texte en facilitant l'accès rapide aux enregistrements pertinents, réduisant ainsi les délais de réponse.

DISCUSSION

Dans cette section, nous allons examiner les résultats obtenus et la méthode utilisée pour les obtenir. Pour cela, nous commencerons par les commandes de base.

Redis : Connexion à la base Redis:

Pour se connecter à Redis avec le client Python, il est nécessaire de créer une instance de connexion en utilisant la bibliothèque redis-py. Dans cette instance, on spécifie les paramètres de connexion tels que l'hôte, le port de connexion, et le mot de passe. Une fois cette configuration définie, la commande PING peut être utilisée pour vérifier la connexion. Si le serveur Redis est correctement configuré et accessible, PING renverra "PONG", confirmant que la connexion est établie et fonctionnelle.

```
C:\> Users> alaem> OneDrive> Bureau> 3BUT> BDD> try.py > ...
1 import redis # type: ignore
2
3 r = redis.Redis(
4     host='redis-14277.c56.east-us.azure.redns.redis-cloud.com',
5     port=14277,
6     password='5f4BL5tx413OrZkZgPg5HyaNi796nGS3')
7
8 response = r.ping()
9
10 # Afficher la réponse
11 if response:
12     print("PONG") # Si la réponse est True
13 else:
14     print("Pas de réponse")
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "c:\Users\alaem\OneDrive\Bureau\3BUT\BDD\try.py"
PONG

[Done] exited with code=0 in 1.094 seconds
```

>>> Nous avons reçu la réponse "PONG", ce qui signifie que la connexion à la base Redis a bien été établie avec succès. Cette réponse confirme que le serveur Redis est accessible et prêt à traiter les requêtes.

Définir une clé et obtenir sa valeur:

```
 1 import redis
 2
 3 r = redis.Redis(
 4     host='localhost',
 5     port=6379)
 6
 7 r.set("nom", "Morad")
 8
 9 print(r.get("nom"))
```

```
[ etudiant@debian12-docker ~ ]$ /bin/python3 /home/etudiant/Documents/try.py
b'Morad'

[ etudiant@debian12-docker ~ ]$
```

La méthode set nous permet de créer une variable dans Redis, où la clé est définie comme "nom" et la valeur comme "Morad", puis de l'enregistrer dans la base de données Redis. Ensuite, la méthode get est utilisée pour récupérer la valeur associée à la clé "nom", nous renvoyant ici la valeur "Morad".

Et apres voila comment j'ai crée ma base

```
r = redis.Redis(
    host='redis-14277.c56.east-us.azure.redns.redis-cloud.com',
    port=14277,
    password='5f4BL5tx413OrZkZgPg5HyaNi796nGS3')

# Dictionnaire de données pour chaque employé
employes = {
    "Alae": {"age": 21, "departement": "Informatique", "performance": 85, "date_embauche": "2023-02-15"},
    "Rania": {"age": 28, "departement": "Ressources Humaines", "performance": 92, "date_embauche": "2021-06-20"},
    "Karim": {"age": 32, "departement": "Finance", "performance": 78, "date_embauche": "2019-10-01"},
    "Emna": {"age": 25, "departement": "Marketing", "performance": 88, "date_embauche": "2022-03-12"},
    "Yassine": {"age": 30, "departement": "Ventes", "performance": 80, "date_embauche": "2020-07-05"},
    "Hanen": {"age": 26, "departement": "Logistique", "performance": 76, "date_embauche": "2018-11-23"},
}

# Ajout des informations de chaque employé dans Redis
for nom, info in employes.items():
    # Ajout des informations de l'employé sous forme de clé hachée (hash) Redis
    r.hset(nom, mapping=info)
```

Persistance des données:

Pour sauvegarder mes données, comme vu dans les résultats, j'ai utilisé les options de persistance de Redis. Mais comment Redis écrit-il les données sur le disque ?

La persistance fait référence à l'enregistrement de données sur un support de stockage durable, comme un disque SSD. Redis propose plusieurs options de persistance :

-RDB (Redis Database): La persistance RDB prend des instantanés ponctuels de votre ensemble de données à des intervalles spécifiés.

Ces snapshots permettent de capturer l'état de la base de données à un moment donné. Avec cette option, on peut utiliser les commandes `SAVE` et `BGSAVE` pour gérer ces snapshots :

- **SAVE** : Exécute une sauvegarde synchrone de la base de données, bloquant le serveur Redis jusqu'à la fin de la sauvegarde.

```
connection.py 1      redis_save.py X
redis-py > redis_save > redis_save.py > ...
1 import redis
2 rb = redis.Redis(host='localhost', port=6379)
3 rb.save()
4 print("données sauvgardées")
```

>> Résultat: alae@alae-linux:~/Desktop\$ /bin/python3 /home/alae/Desktop/redis-py/redis_save/redis_save.py
données sauvgardées
alae@alae-linux:~/Desktop\$ Go to Line/Column

- **BGSAVE** : Crée une sauvegarde asynchrone en arrière-plan, permettant à Redis de continuer à traiter des requêtes pendant l'opération.

```
import redis
rb = redis.Redis(host='localhost', port=6379)
rb.bgsave()
print("données sauvgardées")
```

>> Résultat: alae@alae-linux:~/Desktop\$ /bin/python3 /home/alae/Desktop/redis-py/redis_save/redis_save.py
données sauvgardées

--**AOF (Append Only File)** : La persistance AOF enregistre chaque opération d'écriture reçue par le serveur. Ces opérations peuvent ensuite être rejouées au démarrage du serveur, recréant ainsi l'ensemble de données original. Les commandes sont enregistrées en utilisant le même format que le protocole Redis lui-même.

```
import redis
from redis.cache import CacheConfig
r = redis.Redis(host='localhost', port=6379)

# Activer le mode AOF
r.config_set('appendonly', 'yes')

# Définir la stratégie de synchronisation AOF
r.config_set('appendfsync', 'everysec') # Vous pouvez changer en 'always' ou 'no' selon vos besoins

# Vérifier les paramètres
append_only = r.config_get('appendonly')
append_sync = r.config_get('appendfsync')

print(f"AOF activé : {append_only['appendonly']}")
```

```
print(f"Stratégie de synchronisation AOF : {append_sync['appendfsync']}")
```

>>Résultat:

```
● alae@alae-linux:~/Desktop$ /bin/python3 /home/alae/Desktop/redis-py/redis_save/redis_save.py
AOF activé : yes
Stratégie de synchronisation AOF : everysec
● alae@alae-linux:~/Desktop$
```

In 17. Col 74. Spaces: 4. UFT-8. LF. () Python 3.10.12.6

transformation de CSV en JSON

Comme montré dans les résultats, la conversion du CSV en JSON permet de structurer les données de manière optimale pour Redis, qui exige un format clé-valeur. En transformant le fichier CSV, nous facilitons son insertion dans Redis en structurant les données selon les besoins du système. Je me suis appuyé sur un exercice vu en cours pour explorer cette approche, en adaptant le code pour rendre la conversion plus généralisée et applicable à différents types de fichiers CSV. Cette méthode de conversion est donc flexible et adaptable, permettant de transformer n'importe quel fichier CSV en JSON pour une utilisation dans Redis. Voici le code que j'ai utilisé pour cette opération :

```
● ● ●

1 import csv
2 from json import dump
3
4 def csv_to_json(csv_file, json_file):
5     # Dictionnaire pour stocker les données
6     data_dict = {}
7
8     # Lire le fichier CSV
9     with open(csv_file, encoding='utf-8') as csvfile:
10         my_reader = csv.DictReader(csvfile) # Lire les colonnes automatiquement à partir du CSV
11
12         # Parcourir chaque ligne du fichier CSV
13         for row in my_reader:
14             # Stocker chaque employé sous la clé correspondant à leur nom
15             # Ajouter les détails des autres colonnes
16             data_dict[row['Nom']] = {
17                 'Performance': row['Performance'],
18                 'Engagement': row['Engagement'],
19                 'Position': row['Position'],
20                 'Salary': row['Salary'],
21                 'Hire Date': row['Hire Date'],
22                 'Remote': row['Remote']
23             }
24
25         # Créer un dictionnaire avec la clé "employeurs"
26         result_dict = {'employeurs': data_dict}
27
28         # Enregistrer le résultat dans un fichier JSON
29         with open(json_file, 'w', encoding='utf-8') as jsonfile:
30             dump(result_dict, jsonfile, ensure_ascii=False, indent=4)
31
32         print(f"Data successfully written to {json_file}")
33
34 # Appeler la fonction avec ton fichier CSV et le fichier de sortie JSON
35 csv_to_json("emploies_data.csv", "employeurs_data.json")
36
```

jointure entre deux JSON

Il est clair qu'une entreprise disposant d'une base de données ne se limite pas à un seul fichier CSV, mais utilise plusieurs fichiers, certains ayant des relations entre eux. Ainsi, il est souvent nécessaire de réaliser des jointures sur deux fichiers avant de les insérer dans la base de données, comme illustré dans les résultats obtenus.

Pour réaliser cette opération, il suffit d'identifier un attribut commun entre les fichiers CSV et d'utiliser cet attribut comme clé pour lier les informations. Cette approche permet de créer un nouveau fichier JSON qui combine les données des deux fichiers initiaux de manière structurée et cohérente. Voici le code que j'ai utilisé pour effectuer cette jointure et obtenir le fichier JSON final adapté pour l'insertion dans notre base de données :

```
import json

def jointure(json1, json2):
    # Pas besoin de transformer en dictionnaires car json1 et json2 sont déjà des dictionnaires

    # Utiliser les noms des clés du premier JSON
    d1_name = list(json1.keys())[0] # On prend le nom de la première clé dans json1
    d2_name = list(json2.keys())[0] # On prend le nom de la première clé dans json2

    d1 = json1[d1_name]
    d2 = json2[d2_name]

    # Dictionnaire pour stocker le résultat de la jointure
    d_res = {}

    # Deux boucles imbriquées pour faire la jointure
    for key1, val1 in d1.items():
        for key2, val2 in d2.items():
            if key1 == key2:
                d = {}
                d.update(val1) # Mettre à jour avec les valeurs de val1
                d.update(val2) # Mettre à jour avec les valeurs de val2
                d_res[key1] = d # Ajouter le résultat dans d_res

    my_my_dict = {}
    my_my_dict['resultat'] = d_res # Renommer la clé comme "resultat"
    z = json.dumps(my_my_dict) # Convertir le dictionnaire final en JSON

    return z

# Fonction pour ouvrir et lire un fichier JSON
def read_json_file(json_file):
    with open(json_file, 'r', encoding='utf-8') as file:
        data = json.load(file) # Charger le contenu JSON dans une variable Python
    return data

# Main program
if __name__ == "__main__":
    d1 = read_json_file('employes_data.json')
    d2 = read_json_file('departement.json')
    d = jointure(d1, d2) # Passer les dictionnaires directement
    print(d)
```

J'ai stocké le résultat final dans ma base Redis sous forme de couple clé-valeur. Ce format est parfaitement adapté à Redis, cela permet un accès rapide et efficace aux données stockées

The screenshot shows the Redis Copilot interface. On the left, there's a sidebar with various icons. The main area displays a table of key-value pairs from a database named 'alae-free-db'. The table has columns for Key Type, Key Name, Value Type, and Size. The keys listed are pip2, foo, alae, pip1, résultat_jointure, pip3, user-session:123, and nom. The 'résultat_jointure' key is currently selected. On the right side, there's a 'Redis Copilot' panel with a welcome message, links to Google, GitHub, and SSO, and a checkbox for agreeing to terms.

Total: 8	Last refresh: 1 min	Filter by Key Name or Pattern	
STRING	pip2	No limit	56 B
STRING	foo	No limit	56 B
STRING	alae	No limit	64 B
STRING	pip1	No limit	48 B
JSON	résultat_jointure	No limit	1 KB
STRING	pip3	No limit	64 B
HASH	user-session:123	No limit	128 B
STRING	nom	No limit	64 B

BloomFilter

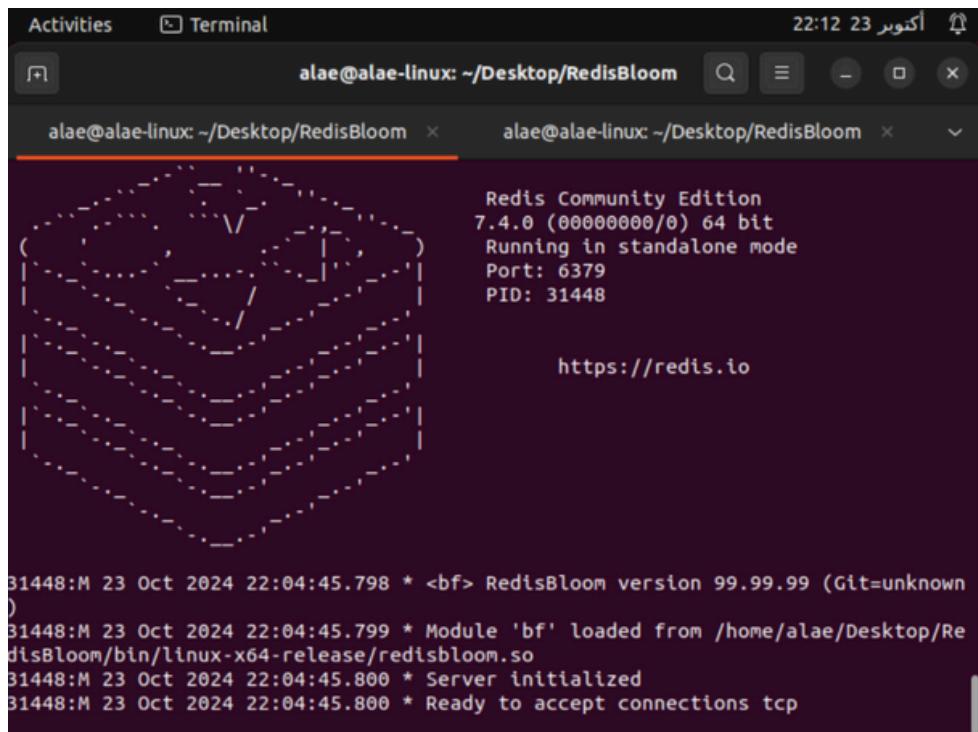
Pour améliorer mes recherches, j'ai exploré le filtre de Bloom. Redis propose une extension appelée RedisBloom qui permet d'utiliser des structures de données probabilistes, y compris les filtres de Bloom. Cela simplifie l'implémentation d'un filtre de Bloom sans nécessiter de codage manuel. Voici comment l'utiliser :

Tout d'abord, je dois cloner le dépôt Git de RedisBloom et ensuite le compiler

```
alae@alae-linux:~/Desktop/RedisBloom$ make clean
make
Building /home/alae/Desktop/RedisBloom/bin/linux-x64-release/t-digest-c/src/libt
digest_static.a ...

Building /home/alae/Desktop/RedisBloom/bin/linux-x64-release/t-digest-c/libtdige
st_static.a ...
[ 50%] Building C object src/CMakeFiles/tdigest_static.dir/tdigest.c.o
[100%] Linking C static library libtdigest_static.a
[100%] Built target tdigest_static
Compiling deps/bloom/bloom.c...
Compiling deps/murmur2/MurmurHash2.c...
Compiling deps/rmutil/util.c...
Compiling src/rebloom.c...
Compiling src/sb.c...
Compiling src/cf.c...
Compiling src/rm_topk.c...
Compiling src/rm_tdigest.c...
Compiling src/topk.c...
Compiling src/rm_cms.c...
Compiling src/cms.c...
Linking /home/alae/Desktop/RedisBloom/bin/linux-x64-release/redisbloom.so...
```

J'ai lancer le serveur RedisBloom

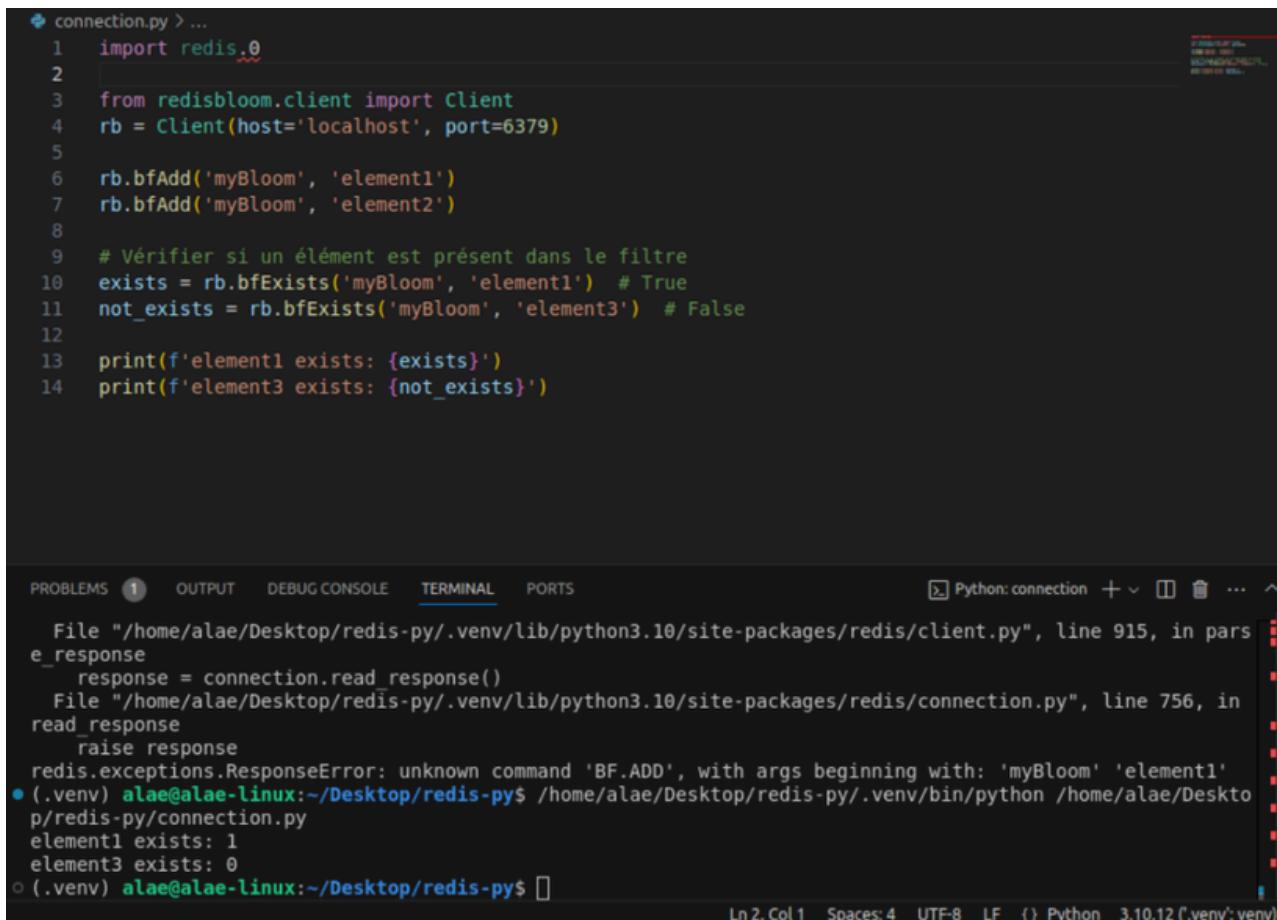


```
Redis Community Edition
7.4.0 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 31448

https://redis.io

31448:M 23 Oct 2024 22:04:45.798 * <bf> RedisBloom version 99.99.99 (Git=unknown)
)
31448:M 23 Oct 2024 22:04:45.799 * Module 'bf' loaded from /home/alae/Desktop/RedisBloom/bin/linux-x64-release/redisbloom.so
31448:M 23 Oct 2024 22:04:45.800 * Server initialized
31448:M 23 Oct 2024 22:04:45.800 * Ready to accept connections tcp
```

Maintenant, je peux utiliser RedisBloom avec mon client Python. J'ai exécuté les commandes nécessaires pour faire fonctionner le filtre et obtenir les résultats observés dans la section 'Résultats'



```
connection.py > ...
1  import redis
2
3  from redisbloom.client import Client
4  rb = Client(host='localhost', port=6379)
5
6  rb.bfAdd('myBloom', 'element1')
7  rb.bfAdd('myBloom', 'element2')
8
9  # Vérifier si un élément est présent dans le filtre
10 exists = rb.bfExists('myBloom', 'element1') # True
11 not_exists = rb.bfExists('myBloom', 'element3') # False
12
13 print(f'element1 exists: {exists}')
14 print(f'element3 exists: {not_exists}')


PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Python:connection + ⌂ ⌂ ... File "/home/alae/Desktop/redis-py/.venv/lib/python3.10/site-packages/redis/client.py", line 915, in parse_response
e_response
    response = connection.read_response()
    File "/home/alae/Desktop/redis-py/.venv/lib/python3.10/site-packages/redis/connection.py", line 756, in read_response
    raise response
redis.exceptions.ResponseError: unknown command 'BF.ADD', with args beginning with: 'myBloom' 'element1'
● (.venv) alae@alae-linux:~/Desktop/redis-py$ /home/alae/Desktop/redis-py/.venv/bin/python /home/alae/Desktop/redis-py/connection.py
element1 exists: 1
element3 exists: 0
○ (.venv) alae@alae-linux:~/Desktop/redis-py$ 
```

MongoDB :

Pour obtenir les résultats observés dans la section des résultats, j'ai exploré MongoDB, en utilisant ses commandes de base et en manipulant les données. Dans cette section, nous allons discuter de la manière dont j'ai réalisé ces résultats.

Connexion à la base MongoDB:

J'ai commencé par installer mongosh, l'outil de ligne de commande pour interagir avec MongoDB. Ensuite, j'ai utilisé les commandes de connexion pour accéder à ma base de données en ligne.

```
PS C:\Users\alaem> mongosh "mongodb+srv://alae.8wk9s.mongodb.net/" --apiVersion 1 --username alaemghirbi
Enter password: *****
Current Mongosh Log ID: 6707efe579b63376ea2da671
Connecting to: mongodb+srv://<credentials>@alae.8wk9s.mongodb.net/?appName=mongosh+2.0.0
Using MongoDB: 7.0.12 (API Version 1)
Using Mongosh: 2.0.0
mongosh 2.3.2 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

|
```

Une fois connecté à MongoDB, vous pouvez commencer à explorer les commandes pour comprendre son fonctionnement. MongoDB utilise des documents stockés dans des collections, qui fonctionnent de manière similaire aux tables dans les bases de données relationnelles. Contrairement aux bases relationnelles, les documents d'une même collection peuvent avoir des champs différents, mais ils partagent tous le champ "_id" comme clé primaire. Les bases de données et collections sont créées automatiquement dès qu'un document y est inséré. Ce sont les lignes de code nécessaire pour créer un document dans une collection

```
# Sélectionner une base de données
db = client['ma_base_de_donnees']

# Sélectionner ou créer une collection
collection = db['ma_collection']
```



Ensuite, j'ai inséré plusieurs documents et les ai lus, comme illustré dans le code suivant.

```
documents=[  
    {"nom": "Alae", "age": 21, "ville": "Nabeul"},  
    {"nom": "Rania", "age": 21, "ville": "Tunis"},  
    {"nom": "Emna", "age": 21, "ville": "BenArous"},  
    {"nom": "Karim", "age": 21, "ville": "HammamLif"}]  
  
result= collection.insert_many(documents)  
  
print(" inséré avec succès avec les IDs : ",result.inserted_ids)  
  
for doc in collection.find():  
    print(doc)
```

On peut également appliquer un filtre lors de la lecture des documents. Dans cet exemple, je choisis d'afficher uniquement les documents dont l'âge est supérieur à 25, comme suit :

```
import connexion  
client=connexion.connexion()  
db=client['ma_base_de_donnees']  
collection=db['ma_collection']  
for doc in collection.find({'age':{"$gt": 25}}):  
    print(doc)
```

Je peux également effectuer des opérations de mise à jour. Par exemple, je vais remplacer tous les âges égales à 21 par 28, comme suit :

```
1 #maj plusieurs documents  
2 collection.update_many({'age':21},{'$set':{'age':28}})  
3 for doc in collection.find():  
4     print(doc)  
5
```

Projection:

La notion de projection dans MongoDB permet de sélectionner uniquement certains champs à afficher dans les résultats d'une requête, ce qui optimise la lecture des données en ne retournant que les informations nécessaires. Par exemple, en utilisant une projection, on peut récupérer uniquement les noms et âges des documents sans inclure d'autres champs.

Pour sélectionner les colonnes à afficher, nous utilisons 1 pour inclure un champ et 0 pour l'exclure. Dans cet exemple, seules les colonnes titre et date seront affichées

```
Atlas atlas-8etut9-shard-0 [primary] exercicecli> db.posts.find({}, {title: 1, date: 1})
[
  {
    _id: ObjectId("670f8f42701390217d40513b"),
    title: 'FirstPostTitle',
    date: 'Wed Oct 16 2024 12:07:00 GMT+0200 (heure d'été d'Europe centrale)'
  },
  {
    _id: ObjectId("670f9044701390217d40513c"),
    title: 'Post Title 2',
    date: 'Wed Oct 16 2024 12:12:22 GMT+0200 (heure d'été d'Europe centrale)'
  },
  {
    _id: ObjectId("670f9186701390217d40513d"),
    title: 'Post Title 3',
    date: 'Wed Oct 16 2024 12:12:22 GMT+0200 (heure d'été d'Europe centrale)'
  },
  {
    _id: ObjectId("670f9186701390217d40513e"),
    title: 'Post Title 4',
    date: 'Wed Oct 16 2024 12:12:22 GMT+0200 (heure d'été d'Europe centrale)'
  }
]
Atlas atlas-8etut9-shard-0 [primary] exercicecli>
```

```
Atlas atlas-8etut9-shard-0 [primary] exercicecli> db.posts.find({}, {_id: 0, title: 1, date: 1})
[
  {},
  {
    title: 'FirstPostTitle',
    date: 'Wed Oct 16 2024 12:07:00 GMT+0200 (heure d'été d'Europe centrale)'
  },
  {
    title: 'Post Title 2',
    date: 'Wed Oct 16 2024 12:12:22 GMT+0200 (heure d'été d'Europe centrale)'
  },
  {
    title: 'Post Title 3',
    date: 'Wed Oct 16 2024 12:12:22 GMT+0200 (heure d'été d'Europe centrale)'
  },
  {
    title: 'Post Title 4',
    date: 'Wed Oct 16 2024 12:12:22 GMT+0200 (heure d'été d'Europe centrale)'
  }
]
```

Il est important de noter qu'on ne peut pas utiliser les attributs 0 et 1 dans la même commande de projection, sauf pour le champ _id (qui s'affiche par défaut). Si vous incluez un champ avec 1, tous les autres champs seront exclus par défaut, sauf si _id est spécifiquement désactivé.

Les Indexes:

Les index dans MongoDB fonctionnent comme ceux d'un livre, permettant de retrouver rapidement une information sans parcourir toute la collection. Sans index, MongoDB effectue un scan complet, ce qui ralentit les requêtes sur de grandes collections. Un index utilise une structure (comme un B-Tree) pour stocker de manière ordonnée les valeurs d'un ou plusieurs champs,

permettant de localiser les documents plus efficacement.

Types d'index :

- **Index unique** : Assure l'unicité des valeurs dans un champ.
- **Index composé** : Indexe plusieurs champs pour optimiser les requêtes complexes.
- **Index de texte** : Pour les recherches en texte intégral.
- **Index géospatial** : Optimise les requêtes de données géographiques

Les index dans MongoDB offrent des avantages significatifs, notamment en améliorant les performances des requêtes et en optimisant les requêtes complexes. Cependant, ils présentent également des limites, telles qu'une utilisation accrue de l'espace de stockage et un ralentissement potentiel des opérations d'écriture. En résumé, bien que les index soient essentiels pour garantir des performances efficaces, surtout sur de grandes collections, leur gestion doit être équilibrée en tenant compte des ressources disponibles.

On peut créer un index simple avec la ligne de commande. Dans cet exemple, je vais créer un index de texte sur le champ titre comme suit :

```
Atlas atlas-8etut9-shard-0 [primary] sample_mflix> db.movies.createIndex({title:"text"});  
title_text
```

-->Cela permettra d'optimiser les recherches basées sur le contenu du champ titre.

Schéma de Validation:

Un schéma de validation dans MongoDB est un mécanisme qui établit des règles sur la structure des documents d'une collection, garantissant que les documents insérés ou mis à jour respectent un format prédéfini. Utilisant JSON Schema, il permet de définir les types de données, les champs obligatoires et des contraintes comme la longueur minimale des chaînes ou les plages de valeurs numériques. Cela aide à maintenir l'intégrité des données en veillant à ce que les documents correspondent aux attentes définies.

```
Atlas atlas-8etut9-shard-0 [primary] sample_mflix> db.createCollection("NouvelleBD", {validator:{$jsonSchema:{bsonType:"object", required:["Redis", "MongoDB"], properties:{"Redis":{bsonType:"string"}, "MongoDB":{bsonType:"string"}, "Croackcoach":{bsonType:"string"}}}}})  
{ ok: 1 }  
Atlas atlas-8etut9-shard-0 [primary] sample_mflix> |
```

transformation de CSV en JSON

Nous allons reprendre l'exercice de transformation de fichiers CSV en documents JSON, mais cette fois-ci en utilisant MongoDB. L'objectif est d'extraire des données d'un fichier CSV et de les insérer directement dans notre collection MongoDB sous forme de documents.

```
import csv
import json
from json import dumps, loads
from ast import literal_eval

def csv_to_json_complex(csv_file):
    """
    Convertit un fichier CSV en un dictionnaire JSON-like, tout en gérant les types de données complexes (JSON, listes)
    """
    data_dict = {}

    with open(csv_file, encoding='utf-8') as csvfile:
        my_reader = csv.DictReader(csvfile)
        for my_row in my_reader:
            my_dict = {}

            for key, value in my_row.items():
                # Vérifier et convertir les valeurs complexes (JSON ou liste)
                try:
                    if str(value).startswith('{') and str(value).endswith('}'):
                        my_dict[key] = literal_eval(value) # Transformer en dictionnaire
                    elif str(value).startswith('[') and str(value).endswith(']'):
                        my_dict[key] = literal_eval(value) # Transformer en liste
                    else:
                        my_dict[key] = int(value) if value.isdigit() else value # Entier ou chaîne de caractères
                except (ValueError, SyntaxError):
                    my_dict[key] = value # Par défaut, considérer comme une chaîne de caractères

            # Ajouter la ligne transformée au dictionnaire final
            data_dict[my_row['id']] = my_dict

    # Retourner le dictionnaire sous forme de chaîne JSON
    return dumps({"test": data_dict})
```

Ainsi, nous obtiendrons le fichier JSON présenté dans la section des résultats.
jointure entre deux JSON

Pour gérer les bases de données dans MongoDB, il est essentiel de procéder à des opérations telles que les jointures entre fichiers. Dans notre exemple, je vais présenter le code que j'ai implémenté pour réaliser une jointure entre deux fichiers JSON.

```
def jointure(mc,id1,id2):

    print(type(mc),id1,id2)
    doc1 = mc.find({ '_id':id1})
    doc2 = mc.find({ '_id':id2})
```

```

d_res = {}
for d1 in doc1:
    d11 = list(d1.keys())
    res1 = d1
    #print('==',d11,'==')
for d2 in doc2:
    d22 = list(d2.keys())
    res2 = d2
    #print('==',d22,'==')
for d_111 in d11:
    for d_222 in d22:
        if d_111 != '_id' and d_222 != '_id':
            if d_111 == d_222:
                d = {}
                d.update(res1[d_111])
                d.update(res2[d_222])
                #print(d)
                d_res[d_111] = d
                #print("**",d_111,d_222,"**")

my_my_dict = {}
my_my_dict['ids'] = d_res
z = dumps(my_my_dict)

# Save the join in the collection
mc.insert_one(my_my_dict)

return z

```

Ce code principal me permet de sélectionner les documents sur lesquels je souhaite effectuer la jointure, tout en incluant les modifications nécessaires pour adapter les champs et les collections selon mes besoins

```

if __name__ == "__main__":
    from pymongo import MongoClient
    from pymongo.server_api import ServerApi
    # Connexion à MongoDB
    uri = "mongodb+srv://alaemghirbi:alaemghirbi@alae.8wk9s.mongodb.net/?retryWrites=true&w=majority&appName=alae"
    client = MongoClient(uri, server_api=ServerApi('1'))
    db = client['ma_base_de_donnees_complexe']
    mycol = db['ma_collection_complexe']

```

```

# Convertir les CSV complexes en JSON
json_one = csv_to_json_complex("C:/Users/alaem/OneDrive/Bureau/3BUT/BDD/mongoDB/csv1.csv")
json_two = csv_to_json_complex("C:/Users/alaem/OneDrive/Bureau/3BUT/BDD/mongoDB/csv2.csv")
# Convertir les JSON en dictionnaires Python
d1_name = list.loads(json_one)[0]
d2_name = list.loads(json_two)[0]
d1 = loads(json_one)[d1_name]
d2 = loads(json_two)[d2_name]
# Insérer dans MongoDB
post_id_one = mycol.insert_one(d1).inserted_id
post_id_two = mycol.insert_one(d2).inserted_id
# Effectuer la jointure entre les deux documents
jointure_result = jointure(mycol, post_id_one, post_id_two)
# Afficher les documents fusionnés après la jointure
from pprint import pprint
cursor = mycol.find({})
for document in cursor:
    pprint(document)

```

Conclusion:

Dans cette section de discussion, j'ai présenté les instructions et les grandes lignes que j'ai suivies pour explorer Redis et MongoDB. J'ai commencé par comprendre les concepts fondamentaux de chaque base de données, en me familiarisant avec les commandes de base et les structures de données qu'elles offrent. Ensuite, j'ai appliqué ces connaissances à des cas pratiques, notamment en utilisant RedisBloom pour optimiser mes recherches avec des Bloom Filters, ainsi qu'en manipulant des documents JSON dans MongoDB. Cette approche m'a permis de comparer les fonctionnalités de chaque système et de tirer parti de leurs avantages respectifs pour gérer efficacement les données.

CONCLUSION

- En conclusion, **MongoDB** et **Redis** sont des bases de données **NoSQL** offrant chacune des **avantages spécifiques** en fonction des **besoins d'application**. **MongoDB**, base de données **orientée document**, est bien adaptée pour stocker des données **complexes et hiérarchiques** de façon **durable** grâce à une structure en **BSON**, proche du **JSON**. Ce type de base de données permet des **requêtes approfondies** et des **transactions multi-documents**, ce qui en fait un excellent choix pour les **applications** qui nécessitent une **gestion persistante de données structurées**, comme les **applications web**, les **systèmes de gestion de contenu** et les **environnements d'entreprise**. En termes de **scalabilité**, **MongoDB** supporte le **partitionnement horizontal** (sharding), permettant de gérer des **volumes importants** de données en les distribuant sur plusieurs **serveurs**, rendant la base de données particulièrement adaptée aux **applications en ligne** nécessitant de grandes quantités de **stockage**.
- De son côté, **Redis**, conçu comme une base de données **clé-valeur en mémoire**, se distingue par sa **rapidité exceptionnelle**, particulièrement utile pour des applications nécessitant des **accès ultra-rapides**. Il est très efficace pour gérer des **données temporaires** comme les **caches**, les **sessions utilisateur**, les **classements**, et les **analyses en temps réel**, faisant de lui un excellent choix pour les cas d'utilisation nécessitant des **réponses rapides** et fréquentes, par exemple pour des **systèmes de messagerie en temps réel** et des **files d'attente**. En revanche, **Redis** est **limité par la quantité de RAM disponible**, ce qui le rend plus approprié pour les **petites charges en mémoire**, souvent sur des systèmes **locaux** ou en ligne avec un besoin de **haute performance**. Ainsi, le **choix entre MongoDB et Redis** dépend largement du **contexte d'utilisation**, de la **complexité des données** à gérer et de la nécessité de **rapidité ou de persistance**.
- Pour les **déploiements locaux**, **Redis** est souvent privilégié pour des **opérations de calculs rapides** et de **tests**, alors que **MongoDB** est plus pertinent pour des **stockages locaux d'envergure** nécessitant un modèle de données **structuré**. En ligne, **MongoDB** est conseillé pour des **applications web** avec des bases de données larges et structurées, tandis que **Redis** est optimal pour des **services en temps réel** ayant besoin de **vitesses de réponse élevées**.

RESSOURCES:

[1] : <https://www.xenonstack.com/blog/sql-vs-nosql-vs-newsql>

Redis:



Redis Python library guide

Connect your Python application to a Redis database

redis.io



Redis persistence

How Redis writes data to disk

redis.io



Client-side caching reference

Server-assisted, client-side caching in Redis

redis.io



Qu'est-ce que Redis expliqué ?

Redis (REmote DIctionary Server) est un magasin de données NoSQL open source, en mémoire, utilisé principalement comme cache d'application ou bas...

ibm.com / May 3, 2023



Utiliser Redis avec Python

Manipuler la base de donnée Redis avec Python

[Loopbin](https://loopbin.com)



Tutoriel Redis : premiers pas avec cette technologie de base de données

Redis est une base de données in memory hautement performante. Dans ce tutoriel Redis, vous découvrirez comment l'installer et la configurer et comment saisir des données.

10/10 | jul. 27, 2020

redis/redis



Redis is an in-memory database that persists on disk. The data model is key-value, but many different kind of values...

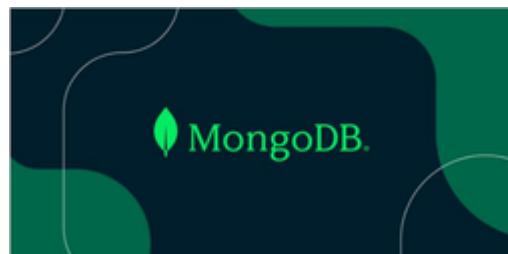
729 Contributors 19 Used by 202 Discussions 67k Stars 24k Forks

[redis/redis](https://github.com/redis/redis): Redis is an in-memory database that persists on disk. The data model is key-value, but many different...

Redis is an in-memory database that persists on disk. The data model is key-value, but many different kind of values are supported: Strings, Lists, Sets, Sorted Sets, Hashes, Streams, HyperLogLogs,...

[GitHub](https://github.com/redis/redis)

MongoDB



Introduction to MongoDB

Learn about the advantages of MongoDB document databases.

[mongodbs](https://mongodbs.com)



[W3Schools.com](https://w3schools.com)

W3Schools offers free online tutorials, references and exercises in all the major languages of the web. Covering popular subjects like HTML, CSS, JavaScript, Python, SQL, Java, and many, many more.

w3schools.com



MongoDB : Le guide complet

MongoDB est la techno de base de donnée NoSQL la plus populaire dans la tech. Elle promet flexibilité et scalabilité.

[WeLoveDevs.com](https://welovedevs.com) / Jul 28, 2020