Année 2024/2025

COMPTE RENDU TP1

NoSql

-

Rapport réalisé par

Wissal BENDIDI

Sup Galilée ingénieurs
L'ÉCOLE D'INGÉNIEURS DE L'UNIVERSITÉ SORBONNE PARIS NORD

Bases de données SQL vs NoSQL : Focus sur Redis

Introduction

Les bases de données SQL et NoSQL offrent des approches distinctes pour le stockage et la gestion des données. Le choix entre les deux dépend principalement des besoins spécifiques de l'application.

SQL (Structured Query Language)

- **Structure:** Données organisées dans des tables avec des relations bien définies.
- **Utilisation:** Idéal pour les données structurées et complexes, les transactions et les analyses.
- **Exemples:** MySQL, PostgreSQL, Oracle

NoSQL (Not Only SQL)

- **Structure:** Flexible, sans schéma prédéfini, permettant de stocker différents types de données (documents, clés-valeurs, graphes).
- **Utilisation:** Adapté aux grands volumes de données, aux applications à haute disponibilité et aux données non structurées.
- **Exemples:** MongoDB, Cassandra, Redis

Redis : Une base de données NoSQL en mémoire

- **Caractéristiques:** Extrêmement rapide, flexible et en mémoire.
- **Utilisations principales:**
    - Mise en cache
    - Sessions utilisateur
    - Compteurs…
- **Types de données supportés:**
    - Chaînes de caractères (strings)
    - Hashs (paires clé-valeur)
    - Listes
    - Ensembles
    - Streams…

Quand choisir Redis ?

- **Besoin de performance:** Redis est particulièrement rapide pour les opérations de lecture et d'écriture.
- **Données en mémoire:** Idéal pour stocker des données fréquemment utilisées.
- **Flexibilité:** Structure de données variée pour s'adapter à différents besoins.
- **Applications spécifiques:** Cache, sessions, compteurs, etc.

Conclusion

Le choix entre SQL et NoSQL dépend de nombreux facteurs, notamment la nature des données, les requêtes à effectuer et les contraintes de performance. Redis, en tant que base de données NoSQL en mémoire, offre une solution performante et flexible pour un large éventail d'applications.

En résumé:

- **SQL:** Structure rigide, idéal pour les données relationnelles.
- **NoSQL:** Structure flexible, adapté aux grands volumes de données et aux données non structurées.
- **Redis:** Un cas particulier de NoSQL, axé sur la performance et la flexibilité.

Installation et Lancement de Redis

1. *Mise à jour du système:*

```
wissal@wissal-P65:~$ sudo apt upgrade -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following package was automatically installed and is no longer required:
  mailcap
Use 'sudo apt autoremove' to remove it.
Get more security updates through Ubuntu Pro with 'esm-apps' enabled:
  emacs emacs-gtk docker.io libavcodec60 libavcodec60 libavutil58 libavutil58
  emacs-el libswresample4 libswresample4 gh emacs-common emacs-bin-common
Learn more about Ubuntu Pro at https://ubuntu.com/pro
#
# Patches available for rsync vulnerabilities, including a potential RCE,
# tracked by CVE-2024-12084 through CVE-2024-12088 and CVE-2024-12747.
# For more see: https://ubuntu.com/blog/rsync-remote-code-execution
#
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```


```
wissal@wissal-P65:~$ sudo apt upgrade -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following package was automatically installed and is no longer required:
  mailcap
Use 'sudo apt autoremove' to remove it.
Get more security updates through Ubuntu Pro with 'esm-apps' enabled:
  emacs emacs-gtk docker.io libavcodec60 libavcodec60 libavutil58 libavutil58
  emacs-el libswresample4 libswresample4 gh emacs-common emacs-bin-common
Learn more about Ubuntu Pro at https://ubuntu.com/pro
#
# Patches available for rsync vulnerabilities, including a potential RCE,
# tracked by CVE-2024-12084 through CVE-2024-12088 and CVE-2024-12747.
# For more see: https://ubuntu.com/blog/rsync-remote-code-execution
#
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```


2. Installation de Redis:

```
wissal@wissal-P65:~$ sudo apt install redis-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  mailcap
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libjemalloc2 liblzf1 redis-tools
Suggested packages:
  ruby-redis
The following NEW packages will be installed:
```

3. Lancement du serveur Redis:

```
wissal@wissal-P65:~$ sudo systemctl start redis-server
wissal@wissal-P65:~$ sudo systemctl enable redis-server
Synchronizing state of redis-server.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable redis-server
```

4. Vérification de l'état de l'installation

```
wissal@wissal-P65:~$ sudo systemctl status redis-server
● redis-server.service - Advanced key-value store
     Loaded: loaded (/usr/lib/systemd/system/redis-server.service; enabled; pre>
     Active: active (running) since Thu 2025-01-23 14:15:44 CET; 2min 53s ago
       Docs: http://redis.io/documentation,
             man:redis-server(1)
   Main PID: 7600 (redis-server)
     Status: "Ready to accept connections"
      Tasks: 5 (limit: 18957)
     Memory: 3.4M (peak: 4.3M)
```

4. Résolution de conflits:

Erreur "Address already in use":

Identification du processus utilisant le port 6379:

sudo netstat -tulnp | grep 6379

Si le processus est une autre application:

sudo kill <PID>

Relance du serveur Redis:

sudo systemctl restart redis-server

Vérification du fichier de configuration:

Vérifier si le port dans le fichier `/etc/redis/redis.conf` est correctement configuré.

```
wissal@wissal-P65:~$ sudo netstat -tulnp | grep 6379
tcp        0      0 127.0.0.1:6379          0.0.0.0:*               LISTEN      9366/redis-server 1
tcp6       0      0 ::1:6379                ::*                     LISTEN      9366/redis-server 1
wissal@wissal-P65:~$ sudo systemctl stop redis-server
wissal@wissal-P65:~$ sudo systemctl start redis-server
wissal@wissal-P65:~$ sudo nano /etc/redis/redis.conf
wissal@wissal-P65:~$ sudo nano /etc/redis/redis.conf
wissal@wissal-P65:~$ sudo systemctl restart redis-server
```

Et maintenant on lance Redis !

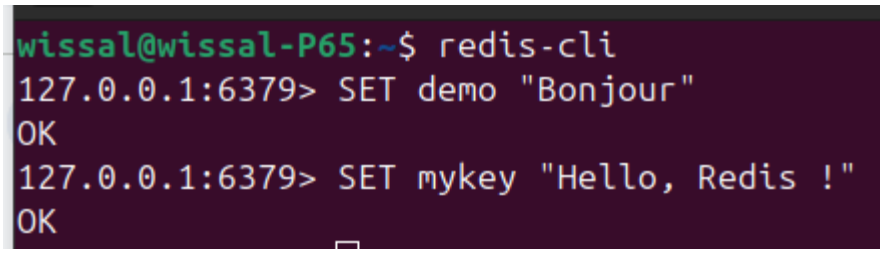
```
wissal@wissal-P65:~$ redis-server
9943:C 23 Jan 2025 14:33:23.426 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
9943:C 23 Jan 2025 14:33:23.426 # Redis version=7.0.15, bits=64, commit=00000000, modified=0, pid=9943, just started
9943:C 23 Jan 2025 14:33:23.426 # Warning: no config file specified, using the default config. In order to specify a
config file use redis-server /path/to/redis.conf
9943:M 23 Jan 2025 14:33:23.428 * Increased maximum number of open files to 10032 (it was originally set to 1024).
9943:M 23 Jan 2025 14:33:23.428 * monotonic clock: POSIX clock_gettime
                _.-``__ ''-._
           _.-``    `.  `_.  ''-._           Redis 7.0.15 (00000000/0) 64 bit
       .-`` .-```.  ```\/    _.,_ ''-._
      (    '      ,       .-`  | `,    )     Running in standalone mode
      |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
      |    `-._   `._    /     _.-'    |     PID: 9943
       `-._    `-._  `-./  _.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |           https://redis.io
       `-._    `-._`-.__.-'_.-'    _.-'
      |`-._`-._    `-.__.-'    _.-'_.-'|
      |    `-._`-._        _.-'_.-'    |
       `-._    `-._`-.__.-'_.-'    _.-'
           `-._    `-.__.-'    _.-'
               `-._        _.-'
                   `-.__.-'

9943:M 23 Jan 2025 14:33:23.430 # Server initialized
9943:M 23 Jan 2025 14:33:23.430 # WARNING Memory overcommit must be enabled! Without it, a background save or replica
tion may fail under low memory condition. Being disabled, it can can also cause failures without low memory condition
, see https://github.com/jemalloc/jemalloc/issues/1328. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysc
tl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
9943:M 23 Jan 2025 14:33:23.431 * Ready to accept connections
```

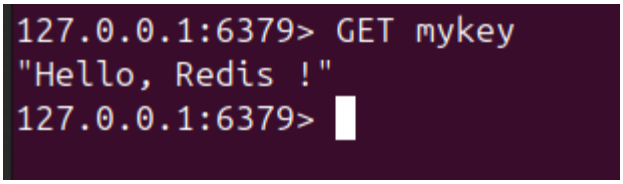
Objectif général : Comprendre les bases de Redis et réaliser quelques opérations simples.

Instructions détaillées :

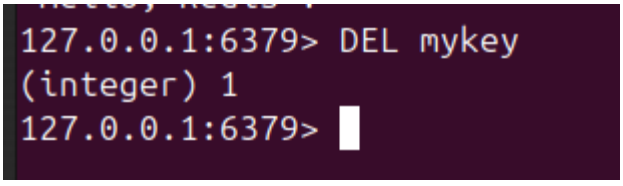
1. **Démarrer le serveur Redis :**
    - Ouvrir un terminal.
    - Exécuter la commande pour démarrer le service Redis (Explication dans la page d'avant)
2. **Se connecter à Redis :**
    - Utiliser le client Redis (redis-cli) pour se connecter au serveur local.
3. **Créer une clé et lui attribuer une valeur :**
    - Utiliser la commande `SET` pour créer une nouvelle clé et lui associer une valeur.
    - **Exemple :** `SET mykey "Hello, Redis!"`



4. **Récupérer la valeur d'une clé :**
    - Utiliser la commande `GET` pour récupérer la valeur associée à une clé.
    - **Exemple :** `GET mykey`



5. **Supprimer une clé :**
    - Utiliser la commande `DEL` pour supprimer une clé.
    - **Exemple :** `DEL mykey`



1. Opérations avancées et types de données

1. **Incrémenter et décrémenter une valeur numérique :**

    - Utiliser les commandes `INCR` et `DECR` pour augmenter ou diminuer la valeur numérique associée à une clé.
    - **Exemple :** `INCR compteur`

```
127.0.0.1:6379> INCR compteur
(integer) 1
```

2. **Définir une durée de vie à une clé :**

    - Utiliser la commande `EXPIRE` pour définir une durée de vie en secondes pour une clé.
    - **Exemple :** `EXPIRE mykey 3600` (expire dans une heure)

```
127.0.0.1:6379> EXPIRE mykey 3600
(integer) 1
```

3. **Utiliser des listes :**

    - Créer une liste : `LPUSH maListe élément1`

```
127.0.0.1:6379> LPUSH maListe element1
(integer) 1
```

    - Ajouter un élément à une liste : `RPUSH maListe élément2`

```
127.0.0.1:6379> RPUSH maListe élément2
(integer) 2
```

    - Récupérer une liste : `LRANGE maListe 0 -1`

```
127.0.0.1:6379> LRANGE maListe 0 1
1) "\xc3\xa9l\xc3\xa9ment2"
2) "\xc3\xa9l\xc3\xa9ment1"
```

    - Supprimer un élément d'une liste : `LPOP maListe` ou `RPOP maListe`

```
127.0.0.1:6379> LPOP maListe
"element1"
```

4. **Utiliser des ensembles :**
    - Créer un ensemble : `SADD monEnsemble élément1 élément2`

```
127.0.0.1:6379> SADD monEnsemble élément1 élément2
(integer) 2
```

    - Ajouter un élément à un ensemble : `SADD monEnsemble élément3`

```
127.0.0.1:6379> SADD monEnsemble élément3
(integer) 1
```

    - Récupérer les éléments d'un ensemble : `SMEMBERS monEnsemble`

```
127.0.0.1:6379> SMEMBERS monEnsemble
1) "\xc3\xa9l\xc3\xa9ment1"
2) "\xc3\xa9l\xc3\xa9ment3"
3) "\xc3\xa9l\xc3\xa9ment2"
```

    - Supprimer un élément d'un ensemble : `SREM monEnsemble élément1`

```
127.0.0.1:6379> SREM monEnsemble élément1
(integer) 1
```

    - Faire l'union de deux ensembles : `SUNION ensemble1 ensemble2`

Concepts clés à retenir

- **Clé-valeur :** Redis est basé sur le concept de clés et de valeurs. Chaque donnée est associée à une clé unique.
- **Types de données :** Redis supporte différents types de données (strings, hashes, lists, sets, sorted sets, etc.).
- **Persistence :** Redis stocke les données en mémoire par défaut, mais peut les sauvegarder sur disque.
- **Expiration:** Vous pouvez définir une durée de vie pour les clés.
- **Atomicité des opérations :** Les opérations Redis sont atomiques, ce qui garantit leur intégrité.

2. Analyse et simplification des instructions Redis à partir de la transcription

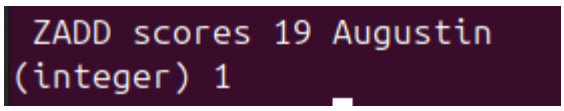
Compréhension de la tâche

L'objectif est d'extraire les commandes Redis utilisées dans la transcription vidéo concernant les ensembles ordonnés (Sorted Sets) et les hachages (Hashes), puis de les simplifier pour une meilleure compréhension.

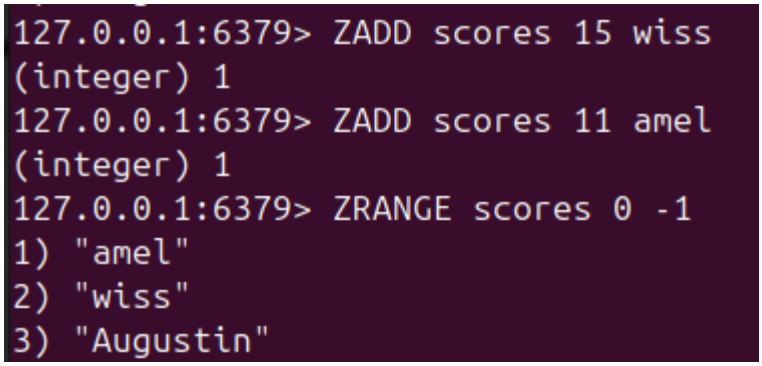
Les commandes Redis identifiées et leurs fonctions :

1. Ensembles ordonnés (Sorted Sets)

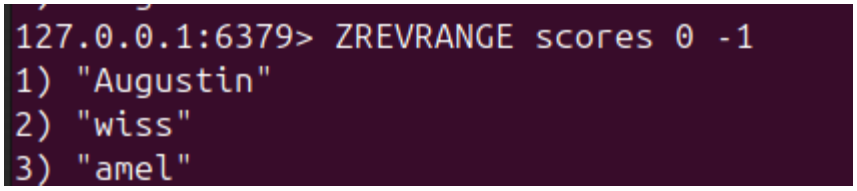
- **ZADD clé score membre:** Ajoute un membre à un ensemble ordonné avec un score associé.
    - Exemple : `ZADD scores 19 Augustin`



- **ZRANGE clé début fin:** Récupère une plage de membres d'un ensemble ordonné par ordre croissant de score.
    - Exemple : `ZRANGE scores 0 -1`



- **ZREVRANGE clé début fin:** Récupère une plage de membres d'un ensemble ordonné par ordre décroissant de score.
    - Exemple : `ZREVRANGE scores 0 -1`



- **ZRANK clé membre:** Retourne le rang d'un membre dans un ensemble ordonné (en commençant par 0).
    - Exemple : `ZRANK scores Augustin`

```
127.0.0.1:6379> ZRANK scores Augustin
(integer) 2
```

2. Hashs

- **HSET clé champ valeur:** Ajoute un champ et sa valeur à un hash.
    - Exemple : `HSET utilisateur:1 nom "Jean Dupont"`

```
127.0.0.1:6379> HSET utilisateur:1 nom "Jean Dupont"
(integer) 1
```

- **HGET clé champ:** Récupère la valeur d'un champ dans un hash.
    - Exemple : `HGET utilisateur:1 nom`

```
127.0.0.1:6379> HGET utilisateur:1 nom
"Jean Dupont"
```

- **HGETALL clé:** Récupère toutes les paires champ-valeur d'un hash.
    - Exemple : `HGETALL utilisateur:1`

```
127.0.0.1:6379>  HGETALL utilisateur:1
1) "nom"
2) "Jean Dupont"
```

- **HINCRBY clé champ incrément:** Incrémente la valeur numérique d'un champ dans un hash.
    - Exemple : `HINCRBY utilisateur:1 age 1`

Simplification et explications

- **ZADD, ZRANGE, ZREVRANGE, ZRANK:** Ces commandes permettent de gérer des ensembles ordonnés, c'est-à-dire des collections d'éléments associés à des scores. Elles sont particulièrement utiles pour des classements, des systèmes de recommandation, etc.
- **HSET, HGET, HGETALL, HINCRBY:** Ces commandes permettent de manipuler des hachages, qui sont des structures de données similaires à des objets JSON. Les hachages sont idéaux pour stocker des informations liées à un élément particulier (par exemple, les informations d'un utilisateur).
-

Cas d'utilisation

- **Sorted Sets:** Classement de scores, implémentation de systèmes de recommandation, gestion de files de priorité.
- **Hashes:** Stockage d'informations structurées (par exemple, des profils utilisateurs, des produits), mise en cache d'objets complexes.

Points clés

- **Scores:** Les Sorted Sets associent un score à chaque élément, permettant ainsi de les trier.
- **Champs:** Les Hashs stockent des paires clé-valeur, similaires aux objets dans de nombreux langages de programmation.
- **Flexibilité:** Redis offre une grande flexibilité pour structurer vos données grâce à ces types de données.

En résumé Les Sorted Sets et les Hashs sont deux structures de données très puissantes dans Redis. Les Sorted Sets sont parfaites pour les données qui doivent être triées en fonction d'un score, tandis que les Hashs sont idéaux pour stocker des informations structurées. En combinant ces deux types de données, vous pouvez créer des applications complexes et performantes.

3. Analyse des commandes Redis utilisées dans la vidéo : Pub/Sub

Compréhension de la tâche

L'objectif est d'extraire les commandes Redis utilisées dans la transcription vidéo concernant le système de publication/souscription (Pub/Sub) et de les simplifier pour une meilleure compréhension.

Les commandes Redis identifiées et leurs fonctions :

- **SUBSCRIBE canal1 canal2 ...:** Souscrit à un ou plusieurs canaux.
  - Exemple : `SUBSCRIBE monCanal`

- **PUBLISH canal message:** Publie un message sur un canal.
  - Exemple : `PUBLISH monCanal "Hello, world!"`

- **PSUBSCRIBE pattern:** Souscrit à tous les canaux correspondant à un motif.
  - Exemple : `PSUBSCRIBE my*` (souscrit à tous les canaux commençant par "my")

- **UNSUBSCRIBE canal1 canal2 ...:** Se désabonne d'un ou plusieurs canaux.
  - Exemple : `UNSUBSCRIBE monCanal`

- **PUNSUBSCRIBE pattern:** Se désabonne de tous les canaux correspondant à un motif.
  - Exemple : `PUNSUBSCRIBE my*`

- **SELECT index:** Sélectionne une base de données.
  - Exemple : `SELECT 1`

Simplification et explications

- **SUBSCRIBE/PUBLISH:** Ces commandes forment la base du système Pub/Sub. Un client se souscrit à un canal, et lorsqu'un message est publié sur ce canal, tous les clients abonnés le reçoivent.
- **PSUBSCRIBE/PUNSUBSCRIBE:** Ces commandes permettent de souscrire ou de se désabonner de multiples canaux en utilisant des motifs, ce qui est pratique pour gérer des groupes de canaux.
- **SELECT:** Cette commande permet de basculer entre différentes bases de données dans Redis.

Cas d'utilisation

- **Messagerie en temps réel:** Chat en direct, notifications push, systèmes de messagerie instantanée.
- **Systèmes d'événements:** Détection d'événements, déclenchement d'actions en fonction d'événements.
- **Systèmes distribués:** Coordination entre différents services.

Points clés

- **Canaux:** Les canaux sont des noms auxquels les clients peuvent se souscrire.
- **Messages:** Les messages sont publiés sur des canaux spécifiques.
- **Patterns:** Les patterns permettent de souscrire à plusieurs canaux en utilisant des expressions régulières simples.
- **Asynchronisme:** Le système Pub/Sub est asynchrone, ce qui signifie que les émetteurs et les récepteurs ne sont pas bloqués pendant l'envoi ou la réception de messages.

En résumé

Le système Pub/Sub de Redis est un outil puissant pour construire des applications en temps réel et des systèmes distribués. Il permet à différents clients de communiquer de manière efficace et asynchrone en s'abonnant à des canaux et en publiant des messages.