Rapport de Manipulation Redis

Découverte d'une base de données NoSQL clé-valeur

Votre Nom

27 novembre 2025

Table des matières

1 Introduction

Ce rapport présente les manipulations effectuées sur Redis, un système de gestion de base de données NoSQL de type clé-valeur. Redis stocke les données en mémoire RAM, ce qui le rend extrêmement rapide. Il est généralement utilisé comme système de cache en complément d'une base de données relationnelle classique.

Les manipulations ont été réalisées à partir de trois vidéos tutoriels qui couvrent les commandes de base, les structures de données avancées et le système de publication/souscription.

1.1 Les familles NoSQL

Il existe quatre grandes familles de bases NoSQL :
— **Clé-Valeur** : Redis, Memcached (pour le cache)
— **Document** : MongoDB (pour des données JSON)
— **Colonnes** : Cassandra (pour le Big Data)
— **Graphe** : Neo4j (pour les réseaux)
Redis appartient à la famille clé-valeur et se distingue par ses performances en mémoire.

2 Vidéo 1 - Commandes de base

2.1 Installation et connexion

Pour démarrer, on lance d'abord le serveur Redis :

```
redis-server
```

Le serveur écoute par défaut sur le port 6379 à l'adresse localhost (127.0.0.1). Ensuite, on se connecte avec le client :

```
redis-cli
```

2.2 Opérations CRUD simples

Les opérations de base permettent de créer, lire et supprimer des clés.

```
SET demo "bonjour"
GET demo
DEL demo
```

La commande SET crée ou met à jour une clé. GET récupère la valeur. DEL supprime la clé et retourne 1 si elle existait, 0 sinon.

On peut aussi utiliser des clés structurées avec le séparateur " :" :

```
SET user:1234 "Samir"
GET user:1234
```

2.3 Compteurs atomiques

Redis gère très bien les compteurs, par exemple pour compter des visiteurs sur un site. Les opérations INCR et DECR sont atomiques, ce qui évite les problèmes de concurrence quand plusieurs utilisateurs se connectent en même temps.

```
SET 1er-mars 0
INCR 1er-mars    # retourne 1
INCR 1er-mars    # retourne 2
DECR 1er-mars    # retourne 1
```

2.4 Gestion de l'expiration

On peut définir une durée de vie pour les clés avec EXPIRE. C'est utile pour gérer automatiquement le cache.

```
SET ma_cle "ma_valeur"
TTL ma_cle              # retourne -1 (duree illimitee)
EXPIRE ma_cle 20        # expire dans 20 secondes
TTL ma_cle              # retourne le temps restant
```

Après expiration, la clé est automatiquement supprimée.

2.5 Les listes

Les listes permettent de stocker plusieurs éléments dans un ordre précis. On peut insérer à gauche (LPUSH) ou à droite (RPUSH).

```
RPUSH mes_cours "BDA"
RPUSH mes_cours "Services Web"
```

Pour afficher les éléments, on utilise LRANGE avec l'index de début et de fin :

```
LRANGE mes_cours 0 -1    # affiche tout
LRANGE mes_cours 0 0     # premier element
```

Attention : GET ne fonctionne pas sur les listes, ça renvoie une erreur.

On peut aussi insérer au début :

```
LPUSH mes_cours "Machine Learning"
```

Pour supprimer, LPOP retire à gauche et RPOP à droite.

2.6 Les sets (ensembles)

Les sets sont comme des listes mais sans ordre et sans doublons. Tous les éléments doivent être uniques.

```
SADD utilisateurs "Augustin"
SADD utilisateurs "Ines"
SADD utilisateurs "Samir"
SADD utilisateurs "Marc"
```

Si on essaie d'ajouter "Marc" une deuxième fois, ça retourne 0.

Pour afficher :

```
SMEMBERS utilisateurs
```

Pour supprimer, on utilise SREM avec la valeur (pas d'index car pas d'ordre) :

```
SREM utilisateurs "Marc"
```

On peut faire l'union de deux sets :

```
SADD autres_utilisateurs "Mondji"
SADD autres_utilisateurs "Philippe"
SUNION utilisateurs autres_utilisateurs
```

3 Vidéo 2 - Structures avancées

3.1 Sorted Sets (sets ordonnés)

Les sorted sets permettent de classer des éléments avec un score. C'est très pratique pour faire des classements ou des systèmes de recommandation.

```
ZADD score_4 19 "Augustin"
ZADD score_4 18 "Ines"
ZADD score_4 15 "Samir"
ZADD score_4 8 "Philippe"
```

Pour afficher par ordre croissant :

```
ZRANGE score_4 0 -1
# retourne: Philippe, Samir, Ines, Augustin
```

Pour l'ordre décroissant (du meilleur au moins bon) :

```
ZREVRANGE score_4 0 -1
# retourne: Augustin, Ines, Samir, Philippe
```

Pour connaître la position d'un élément :

```
ZRANK score_4 "Augustin"    # retourne 3
```

L'indexation commence à 0, donc Augustin est en position 3.

3.2 Performances RAM vs Disque

Un point intéressant abordé dans la vidéo concerne les performances. En résumé :

— RAM : temps d'accès de 10^{-8} secondes, débit de 4 Go/s

— Disque : temps d'accès de 10^{-2} secondes (beaucoup plus lent)

— Un accès disque est environ 1 million de fois plus coûteux qu'un accès RAM

Quand on charge une donnée du disque, on ramène en mémoire un bloc entier de 4096 octets minimum, même si on a besoin que de 8 octets. C'est pour ça que Redis est si rapide : tout est en RAM.

Dans la pratique, on utilise Redis comme cache pour déporter les données fréquemment consultées d'une base relationnelle.

3.3 Les hashes

Les hashes permettent de stocker des objets avec plusieurs champs. C'est pratique pour les profils utilisateurs par exemple.

Méthode 1 - champ par champ :

```
HSET user:11 username "Youssef"
HSET user:11 age 31
HSET user:11 email "samira@polytech.fr"
```

Méthode 2 - tout en une ligne :

```
HMSET user:4 username "Augustin" age 5 email "augustin@gmail.fr"
```

Pour récupérer toutes les infos :

```
HGETALL user:4
```

On peut incrémenter un champ numérique :

```
HINCRBY user:4 age 4    # passe de 5 a 9
```

Pour récupérer juste un champ :

```
HGET user:4 age
```

Pour avoir toutes les valeurs sans les clés :

```
HVALS user:4
```

Point important : avec les hashes, pas besoin d'avoir le même schéma pour tous les utilisateurs. On peut avoir des champs complètement différents, ce qui facilite la répartition sur plusieurs serveurs. Par contre, impossible de faire des jointures comme en SQL.

4 Vidéo 3 - Pub/Sub et gestion avancée

4.1 Système Pub/Sub

Le Pub/Sub permet d'envoyer des messages en temps réel. C'est utile pour les notifications, les chats, etc.

Pour tester, il faut 3 terminaux :

— Terminal 1 : le serveur redis-server
— Terminal 2 : un client qui s'abonne (subscriber)
— Terminal 3 : un client qui publie (publisher)

Dans le terminal 2 :

```
SUBSCRIBE mes_cours
```

Le client reste en écoute. Dans le terminal 3 :

```
PUBLISH mes_cours "Nouveau cours sur MongoDB"
```

Le message arrive instantanément dans le terminal 2.

On peut aussi envoyer des messages directs à un utilisateur :

```
PUBLISH user:1 "Bonjour user 1"
```

4.2 Pattern matching

On peut s'abonner à plusieurs canaux en utilisant un pattern. Par exemple, tous les canaux qui commencent par "m" :

```
PSUBSCRIBE m*
```

Maintenant si on publie sur "mes_cours" ou "mes_notes", le message sera reçu. Mais si on publie sur "notes", ça ne sera pas reçu car ça ne commence pas par "m".

4.3 Gestion des bases de données

Redis propose 16 bases de données numérotées de 0 à 15. Par défaut, on est connecté sur la base 0.

Pour voir toutes les clés de la base courante :

```
KEYS *
```

Pour changer de base :

```
SELECT 1
```

Les bases sont complètement isolées. Les clés de la base 0 ne sont pas visibles depuis la base 1.

4.4 Persistance - Point important

Un point crucial soulevé : attention, Redis n'écrit pas automatiquement sur disque. En cas de coupure électrique, on perd toutes les données qui n'ont pas été sauvegardées. Ce n'est pas comme PostgreSQL ou Oracle qui garantissent la persistance.

Il faut configurer manuellement la persistance dans le fichier redis.conf. Sans cette configuration, on peut perdre les dernières données en cas de panne.

5 Conclusion

Redis est une base NoSQL clé-valeur très rapide grâce au stockage en RAM. On a vu qu'elle propose plusieurs structures de données : strings simples, listes, sets, sorted sets et hashes. Le système Pub/Sub permet la messagerie en temps réel.

Les points à retenir :

— Redis est utilisé comme cache, pas comme base principale
— Les performances sont exceptionnelles (1 million de fois plus rapide que le disque)
— Pas de jointures SQL possibles
— Il faut configurer la persistance pour éviter les pertes de données
— 16 bases de données disponibles pour organiser les clés

Redis se positionne comme un complément aux bases relationnelles : on garde PostgreSQL ou MySQL pour les données critiques et la cohérence, et on utilise Redis pour le cache et les performances.