Gloire BAYOUNDOULA
INFOA3

Matière : **Bases de données NoSQL**
Partie : **Redis**
Enseignant : **Monsieur Samir Youcef**

Note: J'ai réalisé ce TP sur Windows en utilisant Docker desktop. Une fois, Docker desktop installé, il comporte un terminal linux dans lequel on peut utiliser toutes les commandes linux requises pour le TP.
Mais vous disposez d'un environnement linux, ce sera plus pratique.

Rendu TP1 - Partie 1

Les bases de données NoSQL

Les bases de données NoSQL (pour Not Only SQL) sont des systèmes de gestion de données qui se distinguent des bases de données relationnelles traditionnelles (SQL) par leur modèle de stockage plus flexible et plus facile à faire évoluer et par leur capacité à gérer de très grands volumes de données, souvent distribuées. Elles ne remplacent pas les bases SQL, mais les complètent selon les besoins.

Une base NoSQL n'utilise pas obligatoirement le modèle tabulaire (tables, lignes, colonnes).
Elle peut stocker les données sous plusieurs formes :

- modèle clé-valeur : Redis
  Base de données de type {clé, valeur} en mémoire
- modèle documentaire : MongoDB
  Les données sont stockées sous forme de documents (souvent JSON)
- modèle orienté colonne : Cassandra
  Les données sont stockées par colonnes plutôt que par lignes.

Redis 1

La commande "redis-cli" permet de lancer le client, après avoir lancé le serveur avec "redis-server".

On commence le test en définissant une clé et en lui attribuant une valeur avec :

- SET gloire "Bonjour".

La clé est gloire et la valeur Bonjour.
La réponse "OK" montre que la clé a été bien définie.

```
PS C:\Users\cloud> docker exec -it mon-redis redis-cli
127.0.0.1:6379> SET gloire "Bonjour"
OK
127.0.0.1:6379>
```

Les opérations de Redis permettent des actions CRUD : Create, Read, Update, Delete sur les éléments ou clés créés.

Pour récupérer la valeur de la clé, on utilise la commande get :
I

```
127.0.0.1:6379> get gloire
"Bonjour"
127.0.0.1:6379>
```

Pour supprimer un utilisateur, on utilise la commande suivante : del nom-utilisateur
On obtient (integer)1 si l'utilisateur existe et (integer) 0 sinon.

```
127.0.0.1:6379> del user:1234
(integer) 1
127.0.0.1:6379> del user:12
(integer) 0
```

Pour compter le nombre de visiteurs d'un site web qu'on sauvegarde dans une base de données clé-valeur, on utilise la commande set.
Dans notre exemple, on a initialisé la clé à 0. Ensuite,

```
127.0.0.1:6379> set 23jan 0
OK
```

Pour incrémenter la valeur de la clé, on utilise la commande incr :

```
127.0.0.1:6379> incr 23jan
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379> incr 23jan
(integer) 2
127.0.0.1:6379> incr 23jan
(integer) 3
127.0.0.1:6379> incr 23jan
```

Pour décrémenter la valeur de la clé, on utilise la commande decr :
On remarque qu'on a incrémenté jusqu'à la valeur 5. Et on décrémenté jusqu'à la valeur 3 :

```
127.0.0.1:6379> incr 23jan
(integer) 4
127.0.0.1:6379> incr 23jan
(integer) 5
127.0.0.1:6379> decr 23jan
(integer) 4
127.0.0.1:6379>
127.0.0.1:6379> decr 23jan
(integer) 3
```

Les variables sur Redis sont dans la RAM et non dans le disque dur. Donc en cas de problème comme une panne électrique, on ne peut pas retrouver les données que l'on manipule.
C'est pourquoi il est possible de définir la durée de vie d'une clé.

La durée de vie d'une clé est par défaut infinie. On l'observe avec la commande suivante "ttl macle" qui retourne "(integer) -1":

```
127.0.0.1:6379> set macle mavaleur
OK
127.0.0.1:6379> ttl macle
(integer) -1
```

Mais on peut définir le temps de vie de la clé avec la commande :
expire macle temps_d'expiration_en_secondes
Dans cet exemple, j'ai mis 150 secondes. Et avec la commande <<ttl macle>> qui est exécutée à plusieurs reprises, on peut voir que le temps de vie de la clé diminue au fur et à mesure :

```
127.0.0.1:6379> expire macle 150
(integer) 1
127.0.0.1:6379> ttl macle
(integer) 139
127.0.0.1:6379> ttl macle
(integer) 129
127.0.0.1:6379> ttl macle
(integer) 120
127.0.0.1:6379> ttl macle
(integer) 114
```

Pour supprimer une clé, on utilise la commande "del".

```
127.0.0.1:6379> del macle
(integer) 1
```

Pour définir une liste avec redis :

```
127.0.0.1:6379> RPUSH mesCours "NoSQL"
(integer) 1
127.0.0.1:6379> RPUSH mesCours "Algo"
(integer) 2
```

Pour afficher les éléments de la liste, on n'utilise pas la commande "get" comme pour avoir les valeurs des clés, cela crée une erreur.
Cependant on utilise la commande :
LRANGE leNomdeLaListe indice_de_départ indice_de_fin

Lorsque l'indice de départ est 0, cela correspond à l'indice du premier élément de la liste, et lorsque l'indice de fin est -1, cela équivaut à la fin de la liste, en d'autres termes on affiche toute la liste du début à la fin dans notre exemple :

```
127.0.0.1:6379> get mesCours
(error) WRONGTYPE Operation against a key holding the wrong kind of value
127.0.0.1:6379> LRANGE mesCours 0 -1
1) "NoSQL"
2) "Algo"
```

En faisant varier les indices, on récupère l'élément à un indice précis dans la liste :
Dans l'exemple, quand les deux indices sont égaux à 0, on récupère l'élément de la liste d'indice 0. Et quand les indices sont égaux à 1, cela correspond à l'élément de la liste d'indice 1.

```
127.0.0.1:6379> LRANGE mesCours 0 0
1) "NoSQL"
127.0.0.1:6379> LRANGE mesCours 1 1
1) "Algo"
```

Pour supprimer les éléments de la liste, on utilise les commandes LPOP/RPOP.
LPOP = Left-Pop correspond à la suppression de l'élément le plus à gauche
RPOP = Right-Pop correspond à la suppression de l'élément le plus à droite.
"LPOP mesCours" supprime l'élement de gauche qui est "NoSQL", le premier élément de la liste :

```
127.0.0.1:6379> LPOP mesCours
"NoSQL"
127.0.0.1:6379> LRANGE mesCours 0 -1
1) "Algo"
```

"RPOP mesCours" supprime l'élement de droite qui est "Algo", le deuxième et dernier élément de la liste :

```
127.0.0.1:6379> RPOP mesCours
"Algo"
127.0.0.1:6379> LRANGE mesCours 0 -1
(empty array)
```

Dans une liste, on peut ajouter le même élément plusieurs fois car les valeurs ne sont pas obligatoirement distinctes .

NB: A ce stade, j'ai effectué le RPOP précédent après le résultat suivant. Donc, la première occurrence de la valeur "Algo" est celle obtenue en amont du document à l'étape de la définition des listes.

```
127.0.0.1:6379> RPUSH mesCours "Algo"
(integer) 2
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> RPUSH mesCours "Algo"
(integer) 3
127.0.0.1:6379> RPUSH mesCours "Algo"
(integer) 4
127.0.0.1:6379> RPUSH mesCours "Algo"
(integer) 5
```

Cependant, ce n'est pas le cas avec les sets ("ensembles") qui doivent contenir des valeurs distinctes. On peut remarquer que lorsqu'on veut ajouter la même valeur une deuxième fois, ça ne marche pas. La valeur 0 est retournée ce qui signifie que la deuxième occurrence du mot "Dévouement" n'a pas pu être insérée dans le set :

```
127.0.0.1:6379> SADD utilisateurs "Aimour"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Bonte"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Courage"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Devouement"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "Devouement"
(integer) 0
```

Pour afficher les éléments d'un set :

```
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Aimour"
2) "Bonte"
3) "Courage"
4) "Devouement"
```

Lorsqu'on veut supprimer un élément du set, on doit préciser la valeur à supprimer et non l'indice de l'élément comme avec les listes :
La première exécution supprime bien l'élément, et la deuxième montre que l'élément recherché n'existe plus dans le set, donc qu'il a bien été supprimé. Enfin, on remarque aussi qu'en affichant de nouveau le set, l'élément "Aimour" n'est plus présent.

```
127.0.0.1:6379> SREM utilisateurs "Aimour"
(integer) 1
127.0.0.1:6379> SREM utilisateurs "Aimour"
(integer) 0
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Bonte"
2) "Courage"
3) "Devouement"
```

Pour l'union de 2 ensembles :
Création d'un nouvel ensemble<< autresUtilisateurs>> :

```
127.0.0.1:6379> SADD autresUtilisateurs "Herve"
(integer) 1
127.0.0.1:6379> SADD autresUtilisateurs "Greg"
(integer) 1
127.0.0.1:6379> SADD autresUtilisateurs "Hector"
(integer) 1
```

Union des ensembles <<utilisateurs>> et <<autresUtilisateurs>> :

```
127.0.0.1:6379> SUNION utilisateurs autresUtilisateurs
1) "Bonte"
2) "Courage"
3) "Devouement"
4) "Herve"
5) "Greg"
6) "Hector"
```

Redis 2

A présent, nous allons voir d'autres commandes de base pour classer les scores des utilisateurs. C'est l'une des utilisations de Redis dans le cadre du système de recommandation.

On va utiliser des sets ordonnés :

- **Ajoute "Augustin" avec un score de 19 à l'ensemble ordonné score4**
  ZADD score4 19 Augustin

- **Ajoute "Inès" avec un score de 18 à score4**
  ZADD score4 18 Inès

- **Ajoute "Philippe" avec un score de 8 à score4**
  ZADD score4 8 Philippe

- **Récupère tous les éléments de score4 dans l'ordre croissant**
  ZRANGE score4 0 -1

- **Récupère les deux premiers éléments de score4 (ordre croissant)**
  ZRANGE score4 0 1

- **Récupère tous les éléments de score4 dans l'ordre décroissant**
  ZREVRANGE score4 0 -1

- **Renvoie le rang (indice) de "Augustin" dans score4 (ordre croissant)**
  ZRANK score4 Augustin

- **Définit le champ username de users2.11 à "Youssef"**
  HSET users2.11 username Youssef

- **Définit l'âge à 31 pour users2.11**
  HSET users2.11 age 31

- **Définit l'e-mail pour users2.11**
  HSET users2.11 email "youssef@monsite.fr"

- **Récupère toutes les valeurs associées à users2.11**

  HGETALL users2.11

- **Ajouter plusieurs champs en une seule commande pour users2.4**

  HMSET users2.4 username Augustin age 5 email "augustin@gmail.fr"

- **Récupère toutes les valeurs associées à users2.4**

  HGETALL users2.4

- **Incrémente l'âge de users2.4 de 4**

  HINCRBY users2.4 age 4

- **Récupère l'âge actuel de users2.4**

  HGET users2.4 age

- **Récupère toutes les valeurs des champs stockés dans users2.4**

  HVALS users2.4

Redis 3

Voici les commandes Redis pour utiliser les pub/subs utilisés pour les applications en temps réel, autrement dit pour les échanges de messages, les envois de notifications et les inscriptions sur différents canaux de notifications :

1. Souscrire à un canal avec SUBSCRIBE

On utilise la commande SUBSCRIBE pour écouter un canal spécifique, ici mescours

SUBSCRIBE mescours

Tous les messages envoyés sur ce canal seront reçus en temps réel par tous les utilisateurs abonnés.

2. Publier un message sur un canal avec PUBLISH

Depuis le deuxième client, on publie un message sur le canal mescours :

PUBLISH mescours "Un nouveau cours sur MongoDB"

Tous les utilisateurs abonnés au canal mescours recevront immédiatement ce message.

3. Envoyer un message ciblé à un utilisateur spécifique

Pour envoyer un message à un utilisateur spécifique, on peut publier un message sur un canal précis :

PUBLISH users2.1 "Bonjour user1"

Seuls les abonnés au canal users2.1 recevront ce message.

4. Souscrire à plusieurs canaux avec un pattern PSUBSCRIBE

Si on veut écouter plusieurs canaux dont le nom commence par mes, on utilise PSUBSCRIBE :

PSUBSCRIBE mes*

Tous les messages publiés sur des canaux comme mescours, mesnotes, etc., seront reçus.

5. Publier un message sur un canal correspondant au pattern

Depuis le deuxième client, on envoie un message sur mesnotes :

PUBLISH mesnotes "Une nouvelle note est arrivée"

L'utilisateur abonné avec PSUBSCRIBE mes* recevra ce message.

6. Lister toutes les clés enregistrées dans Redis avec KEYS

Pour afficher toutes les clés enregistrées dans Redis :

KEYS *

7. Changer de base de données avec SELECT

Par défaut, Redis a 16 bases de données (numérotées de 0 à 15). Pour passer à une autre base :

SELECT 1

Cela permet de basculer sur la base de données numéro 1.

8. Revenir à la base de données par défaut (0)

SELECT 0

Cela permet de retrouver les clés définies dans la base par défaut.

9. Attention à la persistance des données

Redis n'écrit pas immédiatement les données sur disque. Il faut configurer la persistance pour éviter les pertes en cas de panne. Cela peut être fait avec :

SAVE

ou

BGSAVE

Ces deux commandes permettent de sauvegarder l'état actuel de Redis.