Gloire BAYOUNDOULA
INFOA3

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

Rendu TP1

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 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 :

```
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
```

Ensuite, on incrémente la valeur de la clé, avec 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
```

La durée de vie d'une clé est par défaut infinie comme on peut l'observer 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. 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 pouvoir 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"
```