

---

# Rapport : TP1 NoSQL

---

**BOITI Aymane**

**3<sup>ème</sup> année informatique – cycle ingénieur**

# Introduction aux bases de données NoSQL et à Redis

## Différences entre les bases de données relationnelles et NoSQL

Les bases de données relationnelles (SGBDR) ont longtemps été la norme pour le stockage et la gestion des données. Elles sont caractérisées par une structure rigide basée sur des tables, des relations et un langage de requête normalisé, SQL. Elles respectent les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité), garantissant la cohérence des transactions même en cas de pannes.

Cependant, avec l'augmentation du volume, de la vitesse et de la variété des données (les 3V du Big Data), les SGBDR ont montré leurs limites en termes de scalabilité et de performance. Les bases de données NoSQL ("Not Only SQL") sont apparues comme une alternative permettant plus de flexibilité dans la modélisation des données et une meilleure gestion de la distribution et de la réplication des données. Contrairement aux SGBDR, elles adoptent les propriétés BASE (Basically Available, Soft state, Eventually consistent), favorisant la disponibilité et la tolérance aux pannes au détriment de la cohérence stricte.

## Les familles de bases de données NoSQL

Les bases de données NoSQL se classent en plusieurs familles, selon leur modèle de stockage et leur mode d'interrogation :

1. **Bases Clé-Valeur** : Elles stockent les données sous forme de paires clé-valeur, ce qui permet un accès rapide. Exemple : Redis, DynamoDB.
2. **Bases Orientées Colonnes** : Elles organisent les données en colonnes plutôt qu'en lignes, optimisant ainsi les requêtes analytiques sur de gros volumes de données. Exemple : Apache Cassandra, HBase.
3. **Bases Orientées Documents** : Elles stockent les données sous forme de documents JSON ou BSON, offrant une grande flexibilité pour la structuration des données. Exemple : MongoDB, CouchDB.
4. **Bases Orientées Graphes** : Elles sont conçues pour stocker des relations complexes entre les données sous forme de graphes de nœuds et d'arêtes. Exemple : Neo4j, FlockDB.
5. **Bases Série Temporelle** : Conçues pour la gestion de données chronologiques, elles permettent l'optimisation des requêtes basées sur le temps. Exemple : InfluxDB, OpenTSDB.

## Introduction à Redis

Redis (Remote Dictionary Server) est une base de données NoSQL de type clé-valeur, entièrement en mémoire, offrant une très grande rapidité d'accès aux données. Il est souvent utilisé pour le caching, la gestion de sessions et les files d'attente distribuées. Redis supporte plusieurs structures de données comme les listes, les ensembles, les hachages et les flux. Sa capacité à assurer la persistance des données et sa compatibilité avec les architectures distribuées en font un choix populaire pour les applications modernes à haute performance.

## Tutoriel sur Redis : Manipulation des Clés et Structures de Données

### 1. Installation et Connexion

Avant de commencer, assurez-vous que Redis est installé sur votre machine. Vous pouvez le télécharger depuis [le site officiel de Redis](https://redis.io) ou l'installer via Docker.

```
docker run redis  
  
docker exec -it <container_id> redis-cli
```

Pour démarrer le serveur Redis :

**Commande : redis-server**

```
redis-server
```

```
vagrant@redis-server:~$ redis-server  
4045:C 23 Jan 2025 14:26:13.303 # 000000000000 Redis is starting 000000000000  
4045:C 23 Jan 2025 14:26:13.303 # Redis version=6.0.16, bits=64, commit=00000000, modified=0, pid=4045, just started  
4045:C 23 Jan 2025 14:26:13.303 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf  
  
Redis 6.0.16 (00000000/0) 64 bit  
  
Running in standalone mode  
Port: 6379  
PID: 4045  
  
http://redis.io  
  
4045:M 23 Jan 2025 14:26:13.304 # Server initialized  
4045:M 23 Jan 2025 14:26:13.304 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.  
4045:M 23 Jan 2025 14:26:13.304 * Ready to accept connections
```

Pour se connecter au serveur Redis via le client en ligne de commande (CLI) :

**Commande : redis-cli**

```
redis-cli
```

Par défaut, Redis fonctionne sur localhost (127.0.0.1) et écoute sur le port 6379.

```
vagrant@redis-server:~$ redis-cli  
127.0.0.1:6379>
```

---

## 2. Définir une clé

Commande :

SET <clé> <valeur>

Exemple : SET demo "Bonjour"

```
127.0.0.1:6379> SET demo "Bonjour"
OK
```

Explication :

- SET permet de créer une clé et lui attribuer une valeur.
- Redis renvoie OK si l'opération réussit.

---

## 3. Lire une valeur

Commande :

GET <clé>

Exemple : GET demo

```
127.0.0.1:6379> GET demo
"Bonjour"
```

Explication :

- Permet de récupérer la valeur associée à une clé.

---

## 4. Supprimer une clé

Commande :

DEL <clé>

Exemple : DEL demo

```
127.0.0.1:6379> DEL demo
(integer) 1
```

**Explication :**

- Renvoie 1 si la clé a été supprimée, 0 si elle n'existe pas.
- 

## 5. Incrémenter une valeur

**Commande :**

INCR <clé>

**Exemple :** INCR visite\_count

```
127.0.0.1:6379> INCR visite_count
(integer) 1
127.0.0.1:6379> INCR visite_count
(integer) 2
127.0.0.1:6379> INCR visite_count
(integer) 3
127.0.0.1:6379> INCR visite_count
(integer) 4
```

**Explication :**

- Augmente la valeur d'une clé numérique de 1.
  - Très utile pour compter les visiteurs d'un site.
- 

## 6. Décrémenter une valeur

**Commande :**

DECR <clé>

**Exemple :** DECR visite\_count

```
127.0.0.1:6379> INCR visite_count
(integer) 4
127.0.0.1:6379> DECR visite_count
(integer) 3
127.0.0.1:6379> DECR visite_count
(integer) 2
```

**Explication :**

- Diminue la valeur d'une clé numérique de 1.

---

## 7. Définir une durée de vie pour une clé

Commande :

```
EXPIRE <clé> <secondes>
```

Exemple : EXPIRE demo 60

```
127.0.0.1:6379> set demo "Bonjour"
OK
```

Explication :

- Définit une durée de vie en secondes avant la suppression automatique.

Remarque :

- TTL <clé> permet de vérifier le temps restant.

```
127.0.0.1:6379> TTL demo
(integer) 24
```

---

## 8. Manipulation de listes

Ajouter un élément à une liste

Commande :

```
LPUSH <liste> <valeur>
```

```
RPUSH <liste> <valeur>
```

Exemple :

- LPUSH cours "BDA"

```
127.0.0.1:6379> LPUSH cours "BDA"
(integer) 1
```

- RPUSH cours "Services Web"

```
127.0.0.1:6379> RPUSH cours "Services Web"
(integer) 2
```

Explication :

- LPUSH ajoute à gauche, RPUSH ajoute à droite.

### Afficher les éléments d'une liste

Commande :

```
LRANGE <liste> <start> <stop>
```

Exemple : LRANGE cours 0 -1

```
127.0.0.1:6379> LRANGE cours 0 -1  
1) "BDA"  
2) "Services Web"
```

Explication :

- 0 -1 affiche toute la liste.

### Supprimer un élément d'une liste

Commande :

```
LPOP <liste>
```

```
RPOP <liste>
```

Exemple : LPOP cours

```
127.0.0.1:6379> LPOP cours  
"BDA"
```

Explication :

- LPOP supprime à gauche, RPOP à droite.

---

## 9. Manipulation d'ensembles (Sets)

### Ajouter un élément

Commande :

```
SADD <set> <valeur>
```

Exemple : SADD utilisateurs "Augustin"

```
127.0.0.1:6379> SADD utilisateurs "Augustin"  
(integer) 1
```

**Explication :**

- Un ensemble ne peut pas contenir de doublons.

**Afficher les éléments d'un ensemble**

**Commande :**

```
SMEMBERS <set>
```

**Exemple : SMEMBERS utilisateurs**

```
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Marc"
2) "Aymane"
3) "Augustin"
```

**Explication :**

- Affiche tous les éléments d'un ensemble.

**Supprimer un élément d'un ensemble**

**Commande :**

```
SREM <set> <valeur>
```

**Exemple : SREM utilisateurs "Augustin"**

```
127.0.0.1:6379> SREM utilisateurs "Augustin"
(integer) 1
```

**Explication :**

- Supprime un élément du set s'il est présent.
- Renvoie 1 si la suppression a eu lieu, 0 si l'élément n'était pas présent.

**Fusionner deux ensembles**

**Commande :**

```
SUNION <set1> <set2>
```



**Exemple :** SUNION utilisateurs autres\_utilisateurs

```
127.0.0.1:6379> SUNION utilisateurs autres_utilisateurs
1) "Aymane"
2) "Marc"
3) "Autre1"
4) "Autre2"
```

**Explication :**

- Fusionne deux ensembles et affiche le résultat contenant tous les éléments uniques des deux sets.
- 

## 2ème Partie : Explications avancées et optimisation des accès aux données

### 1. Optimisation des accès aux données en mémoire vs disque

Les bases de données classiques stockent les informations sur disque, ce qui engendre une latence importante lors des accès aux données. L'accès à une donnée en mémoire est environ un million de fois plus rapide qu'un accès sur disque.

Redis permet de contourner cette latence en stockant toutes les données en mémoire RAM. Cependant, il est essentiel de comprendre que si la RAM est saturée, Redis supprime les anciennes valeurs en fonction de l'algorithme d'éviction configuré.

### 2. Ensembles ordonnés (Sorted Sets)

Les ensembles ordonnés sont utilisés pour stocker des données classées selon un score, utile notamment dans les systèmes de recommandation.

**Ajouter un élément avec un score**

**Commande :**

```
ZADD <ensemble> <score> <valeur>
```

**Exemple :** ZADD scores 19 "Augustin"

ZADD scores 18 "Ines"

```
127.0.0.1:6379> ZADD scores 19 "Augustin"
(integer) 1
127.0.0.1:6379> ZADD scores 18 "Ines"
(integer) 1
```

## Récupérer les éléments triés

Commande :

```
ZRANGE <ensemble> <start> <stop>
```

Exemple : ZRANGE scores 0 -1

```
127.0.0.1:6379> ZRANGE scores 0 -1  
1) "Ines"  
2) "Augustin"
```

Explication :

- Affiche les éléments par ordre croissant de score.

## Récupérer les éléments triés en ordre décroissant

Commande :

```
ZREVRANGE <ensemble> <start> <stop>
```

Exemple : ZREVRANGE scores 0 -1

```
127.0.0.1:6379> ZREVRANGE scores 0 -1  
1) "Augustin"  
2) "Ines"
```

---

## 3. Hachages (Hashes) et structuration des données

Les hachages permettent de stocker des objets structurés sous forme de paires clé-valeur multiples.

### Ajouter un champ dans un hachage

Commande :

```
HSET <hash> <champ> <valeur>
```

Exemple : HSET user:10 username " Mario"

HSET user:10 age 15

```
127.0.0.1:6379> HSET user:10 username "Mario"  
(integer) 1  
127.0.0.1:6379> HSET user:10 age 15  
(integer) 1
```

### Ajouter plusieurs valeurs dans un hachage

Commande :

```
HMSET <hash> <champ1> <valeur1> <champ2> <valeur2> ...
```

Exemple : HMSET user:11 username "Aymane" age 21 email [aymane@uspn.fr](mailto:aymane@uspn.fr)

```
127.0.0.1:6379> HMSET user:11 username "Aymane" age 21 email "aymane@uspn.fr"
OK
```

### Incrémenter un champ d'un hachage

Commande :

```
HINCRBY <hash> <champ> <valeur>
```

Exemple : HINCRBY user:11 age 4

```
127.0.0.1:6379> HINCRBY user:11 age 4
(integer) 25
```

### Récupérer toutes les valeurs d'un hachage

Commande :

```
HGETALL <hash>
```

Exemple : HGETALL user:11

```
127.0.0.1:6379> HGETALL user:11
1) "username"
2) "Aymane"
3) "age"
4) "25"
5) "email"
6) "aymane@uspn.fr"
```

---

#### 4. Impact de l'accès aux données et importance du caching

Les systèmes de gestion de bases de données classiques doivent toujours récupérer les données sur disque avant de les charger en mémoire. Cela génère un délai important, surtout lorsque de nombreuses requêtes sont effectuées.

Redis joue un rôle de cache en stockant les données fréquemment consultées directement en RAM, évitant ainsi de solliciter le disque trop souvent. Dans les architectures modernes, Redis est souvent couplé à une base relationnelle classique pour optimiser les performances.

---

### 3ème Partie : Pub/Sub et gestion des bases de données

#### 1. Système de messagerie avec Redis (Pub/Sub)

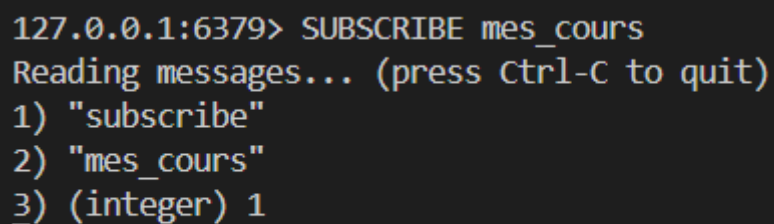
Redis propose un mécanisme de **publication/souscription (Pub/Sub)** permettant d'envoyer des messages à plusieurs clients abonnés à un canal spécifique. Ce système est particulièrement utilisé pour les applications en temps réel, telles que les systèmes de messagerie instantanée et les notifications push.

##### S'abonner à un canal

##### Commande :

SUBSCRIBE <canal>

**Exemple :** SUBSCRIBE mes\_cours



```
127.0.0.1:6379> SUBSCRIBE mes_cours
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "mes_cours"
3) (integer) 1
```

##### Explication :

- L'utilisateur reste en écoute sur le canal spécifié.
- Tous les messages publiés sur ce canal seront reçus en temps réel.

##### Publier un message sur un canal

##### Commande :

PUBLISH <canal> <message>

**Exemple :** PUBLISH mes\_cours "Un nouveau cours sur MongoDB."

```
127.0.0.1:6379> PUBLISH mes_cours "Un nouveau cours sur MongoDB."
(integer) 1
```

```
127.0.0.1:6379> SUBSCRIBE mes_cours
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "mes_cours"
3) (integer) 1
1) "message"
2) "mes_cours"
3) "Un nouveau cours sur MongoDB."
```

**Explication :**

- Envoie un message à tous les utilisateurs abonnés au canal mes\_cours.
- Les abonnés reçoivent immédiatement la notification.

**S'abonner à plusieurs canaux avec un pattern**

**Commande :**

```
PSUBSCRIBE <pattern>
```

**Exemple :** PSUBSCRIBE mes\*

```
127.0.0.1:6379> PUBLISH mes_cours "Un nouveau cours sur MongoDB."
(integer) 1
```

```
127.0.0.1:6379> PSUBSCRIBE mes*
Reading messages... (press Ctrl-C to quit)
1) "psubscribe"
2) "mes*"
3) (integer) 1
1) "pmessage"
2) "mes*"
3) "mes_cours"
4) "Un nouveau cours sur MongoDB."
```

**Explication :**

- L'utilisateur s'abonne à tous les canaux commençant par mes (ex. mes\_cours, mes\_notes).
  - Tous les messages envoyés sur ces canaux seront reçus.
-

## 2. Gestion des bases de données dans Redis

Redis permet d'utiliser plusieurs bases de données sur un même serveur. Par défaut, Redis démarre avec **16 bases numérotées de 0 à 15**.

### Afficher toutes les clés d'une base

Commande :

```
KEYS *
```

Exemple : KEYS \*

```
127.0.0.1:6379> KEYS *
1) "visite_count"
2) "user:11"
3) "cours"
4) "user:10"
5) "scores"
6) "autres_utilisateurs"
7) "utilisateurs"
```

Explication :

- Renvoie la liste des clés stockées dans la base de données actuelle.

### Changer de base de données

Commande :

```
SELECT <numéro_base>
```

Exemple : SELECT 1

```
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> █
```

Explication :

- Change la base de données active à la base numéro 1.
- Toutes les opérations suivantes s'appliqueront à cette base.

### Vérifier les clés d'une base spécifique

Commande :

```
SELECT <numéro_base>
KEYS *
```

**Exemple :** SELECT 1

KEYS \*

```
127.0.0.1:6379> SELECT 1
OK
127.0.0.1:6379[1]> KEYS *
(empty array)
```

**Explication :**

- Passe sur la base 1 et affiche les clés disponibles.

---

### 3. Persistance des données et reprise après panne

Contrairement aux bases relationnelles classiques, Redis ne persiste pas les données en temps réel. Par défaut, les informations sont stockées en mémoire et peuvent être perdues en cas d'arrêt brutal du serveur. Pour éviter cela, Redis propose plusieurs options de persistance :

1. **RDB (Redis Database File)** : Effectue des sauvegardes périodiques des données sur disque.
2. **AOF (Append-Only File)** : Enregistre chaque commande exécutée pour une récupération complète en cas de panne.
3. **Combinaison des deux** : Redis permet d'utiliser simultanément les modes RDB et AOF pour garantir un bon équilibre entre performance et sécurité des données.

**Important :**

- La configuration de la persistance doit être définie manuellement dans le fichier de configuration de Redis (redis.conf).
- En cas de panne, seules les données sauvegardées seront restaurées.