# TP n°1 Partie Redis

**Partie 1 : L'objectif de ce cet exercice est de réaliser les manipulations décrites dans les trois vidéos ci-dessous, tout en les expliquant en détail. Il est important de bien clarifier chaque manipulation et d'utiliser éventuellement des figures et/ou des schémas pour appuyer vos propos. Ainsi, en vous lisant, le lecteur doit être en mesure de comprendre comment installer et utiliser REDIS et le situer parmi les quatre familles de systèmes de gestion de bases de données NoSQL, vues en cours.**

## REDIS 1 :

**Les systèmes NoSQL sont regroupés en quatre familles principales, chacune adaptée à un usage différent :**

**1. Clé–valeur**

- **Stockage sous forme de clé → valeur, comme un dictionnaire.**
- **Très rapide et simple.**
- **Exemples : Redis, DynamoDB.**

**2. Orientées colonnes**

- **Stockage par colonnes plutôt que par lignes.**
- **Optimisées pour le Big Data et les lectures massives.**
- **Exemples : Cassandra, HBase.**

**3. Orientées documents**

- **Stockage de documents JSON/BSON flexibles et semi-structurés.**
- **Très adaptées aux applications web.**

- **Exemples : MongoDB, CouchDB.**

**4. Orientées graphes**

- **Représentation des données sous forme de nœuds et relations.**
- **Idéales pour les réseaux, recommandations et analyses de liens.**
- **Exemples : Neo4j, OrientDB.**

**Situation de Redis parmi les quatre familles NoSQL**

**Redis** appartient clairement à la famille des **bases de données clé–valeur**. Il s'agit d'un système NoSQL en mémoire qui stocke les données sous forme de paires **clé** → **valeur**, ce qui le rend extrêmement rapide pour les opérations de lecture et d'écriture.

Contrairement aux bases orientées documents, colonnes ou graphes, Redis se concentre sur un modèle simple et performant, idéal pour le cache, les sessions et les traitements à haute vitesse.

**Installation de Redis sur Linux :**

commandes utilisés :

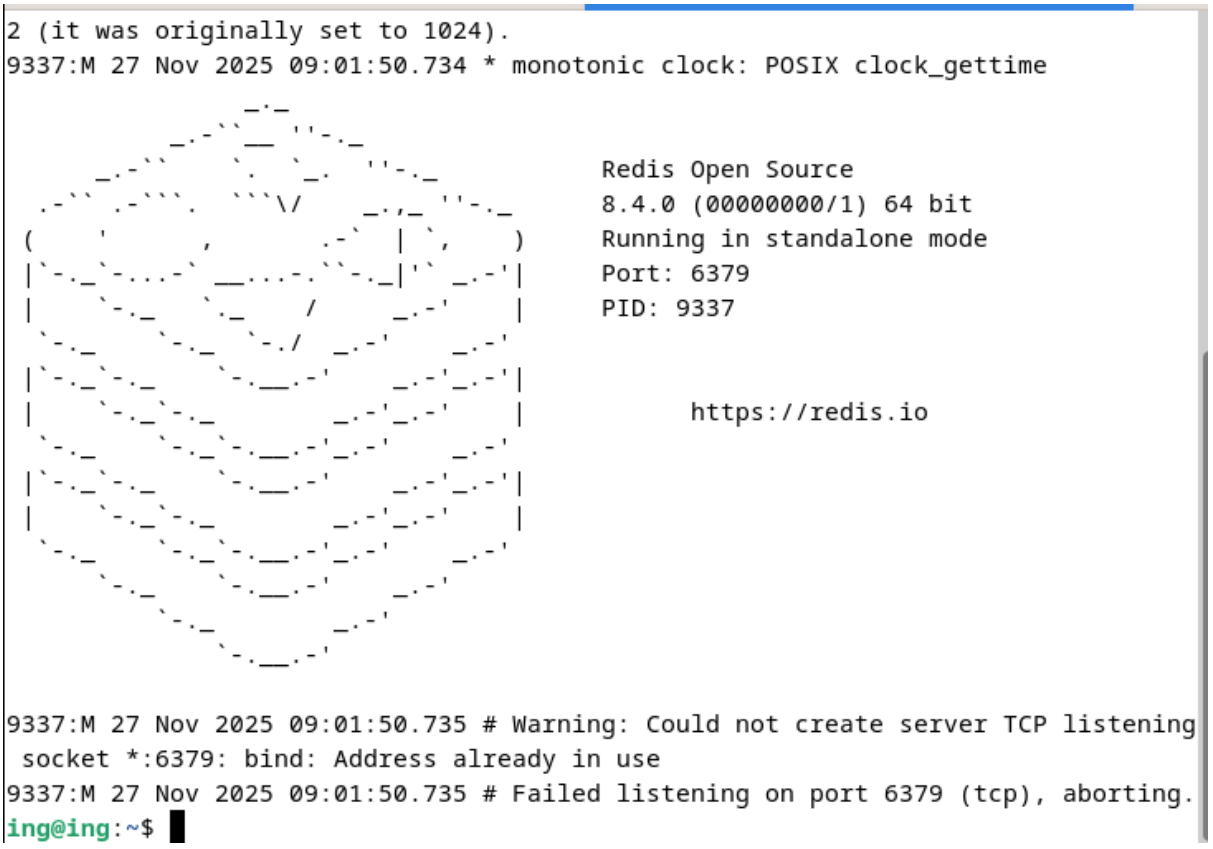
https://redis.io/docs/latest/operate/oss_and_stack/install/archive/install-redis/install-redis-on-linux/

```
sudo apt-get install lsb-release curl gpg
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/redis-archive-keyring.gpg
sudo chmod 644 /usr/share/keyrings/redis-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg]
https://packages.redis.io/deb $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/redis.list
sudo apt-get update
sudo apt-get install redis

sudo systemctl enable redis-server
sudo systemctl start redis-server
```

```
ing@ing:~$ sudo apt-get install lsb-release curl gpg
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyr
ings/redis-archive-keyring.gpg
sudo chmod 644 /usr/share/keyrings/redis-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://pack
ages.redis.io/deb $(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/re
dis.list
sudo apt-get update
sudo apt-get install redis
[sudo] Mot de passe de ing :
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
lsb-release est déjà la version la plus récente (12.0-1).
lsb-release passé en « installé manuellement ».
curl est déjà la version la plus récente (7.88.1-10+deb12u14).
gpg est déjà la version la plus récente (2.2.40-1.1+deb12u1).
gpg passé en « installé manuellement ».
Les paquets suivants ont été installés automatiquement et ne sont plus nécessair
es :
  linux-headers-6.1.0-37-amd64 linux-headers-6.1.0-37-common
  linux-headers-6.1.0-38-amd64 linux-headers-6.1.0-38-common
  linux-headers-6.1.0-39-amd64 linux-headers-6.1.0-39-common
  linux-image-6.1.0-37-amd64 linux-image-6.1.0-38-amd64
```

1) **Lancer le serveur Redis : redis-server**




2) **Afin de communiquer avec la base de données on lance le client via la commande Redis : redis-cli**


→ **Par défaut, Redis écoute sur le port 6379**

**Manipulations de base avec Redis**

Avant de commencer les manipulations, rappelons une remarque importante :

> **Remarque :** Redis est une base de données en mémoire (RAM), ce qui la rend extrêmement rapide. Cependant, elle peut également

assurer la persistance des données sur le disque grâce à ses mécanismes de sauvegarde (RDB et AOF).

Les opérations CRUD (Create, Read, Update, Delete) sont très simples car Redis fonctionne avec des paires **clé → valeur**.

### 1.1. Create (création)

```
ing@ing:~$ redis-cli
127.0.0.1:6379>  SET user:1234 "Samir"
OK
127.0.0.1:6379>
```

*Cette commande crée une clé `user:1234` avec la valeur `"Samir"`.*

### 1.2. Read (lecture)

```
127.0.0.1:6379> GET user:1234
"Samir"
127.0.0.1:6379>
```

*Redis retourne la valeur associée à la clé `user:1234`.*

### *1.3. Update (mise à jour)*

*Il suffit de refaire un `SET` :*

```
127.0.0.1:6379> SET user:1234 "Ahmed"
OK
127.0.0.1:6379> GET user:1234
"Ahmed"
127.0.0.1:6379>
```

*La valeur précédente est remplacée : mise à jour immédiate.*

### *1.4. Delete (suppression)*

```
127.0.0.1:6379> DEL user:1234
(integer) 1
127.0.0.1:6379>
```

*La clé est supprimée définitivement.*

### *2. Compter le nombre de visiteurs d'un site web*

*Redis est souvent utilisé comme compteur grâce à ses opérations atomiques.*
***Initialiser un compteur :***

```
127.0.0.1:6379> SET 27nov 0
OK
127.0.0.1:6379>
```

***Incrémenter :***

```
127.0.0.1:6379> INCR 27nov
(integer) 1
127.0.0.1:6379> INCR 27nov
(integer) 2
127.0.0.1:6379> INCR 27nov
(integer) 3
127.0.0.1:6379> INCR 27nov
(integer) 4
127.0.0.1:6379>
```

***Décrémenter :***

```
127.0.0.1:6379> DECR 27nov
(integer) 3
127.0.0.1:6379>
```

***Interprétation :***

- `INCR` *et* `DECR` *modifient la valeur sans risque de conflit.*

- *Redis **gère très bien** **la concurrence**, même si plusieurs utilisateurs incrémentent en même temps.*

- *C'est pourquoi Redis est utilisé dans les systèmes de comptage en temps réel.*

### 3. Gestion de la durée de vie des clés

Par défaut, une clé a une durée de vie **illimitée**.

**Exemple :**

```
127.0.0.1:6379> SET macle "mavaleur"
OK
127.0.0.1:6379> TTL macle
(integer) -1
127.0.0.1:6379>
```

`Interprétation : -1 signifie que la clé n'a` **`pas de durée d'expiration.`**

#### *Définir une durée de vie (expiration)*

```
127.0.0.1:6379> EXPIRE macle 120
(integer) 1
127.0.0.1:6379> TTL macle
(integer) 116
127.0.0.1:6379>
```

*Le temps diminue à mesure que les secondes s'écoulent.*

### *4. Autres structures de données dans Redis*

*Redis ne gère pas que des chaînes : il propose aussi d'autres types, comme les **listes**.*

#### *Manipulations sur les listes*

### *4.1. Ajouter des éléments dans une liste*

***Ajouter à droite :***

```
127.0.0.1:6379> RPUSH mescours "service web"
(integer) 1
127.0.0.1:6379>
```

***Ajouter à gauche :***

```
127.0.0.1:6379> LPUSH mescours "cloud computing"
(integer) 2
127.0.0.1:6379>
```

*Les listes sont ordonnées : on peut ajouter à droite ou à gauche.*

### *4.2. Afficher les éléments d'une liste*

*On n'utilise pas GET, mais LRANGE.*

***Afficher toute la liste :***

```
127.0.0.1:6379> LRANGE mescours 0 -1
1) "cloud computing"
2) "service web"
127.0.0.1:6379>
```

***Afficher seulement le premier élément :***

```
127.0.0.1:6379> LRANGE mescours 0 0
1) "cloud computing"
127.0.0.1:6379>
```

### 4.3. Supprimer un élément de la liste

**Supprimer à gauche ou à droite:**

```
127.0.0.1:6379> LPOP mescours
"cloud computing"
127.0.0.1:6379> RPOP mescours
"service web"
127.0.0.1:6379>
```

**Remarque :** Dans Redis, les listes acceptent les doublons. Cela signifie qu'un même élément peut apparaître plusieurs fois dans la même liste. Redis ne vérifie pas l'unicité des valeurs : il garde tous les éléments dans l'ordre d'insertion.

### Manipulations de base sur les Sets dans Redis

Les *sets* (ensembles) sont un type de données Redis qui stockent des valeurs **uniques**, sans ordre particulier.
Ils sont très utiles pour manipuler des groupes d'éléments sans doublons : tags, utilisateurs connectés, paniers, etc.

#### 1. Ajouter des éléments dans un set

```
127.0.0.1:6379> SADD meset "Paris"
(integer) 1
127.0.0.1:6379> SADD meset "Tunis"
(integer) 1
127.0.0.1:6379> SADD meset "Paris"
(integer) 0
127.0.0.1:6379>
```

- `Paris` ne sera ajouté qu'une seule fois.
- Redis **ignore automatiquement les doublons**.

#### 2. Afficher les éléments du set

```
127.0.0.1:6379> SMEMBERS meset
1) "Paris"
2) "Tunis"
127.0.0.1:6379>
```

**Interprétation :**

- Affiche tous les éléments de l'ensemble.
- L'ordre n'est pas garanti (set = non ordonné).

### 3. Vérifier si un élément appartient au set

```
127.0.0.1:6379> SISMEMBER meset "Paris"
(integer) 1
127.0.0.1:6379>
```

*→ 1 signifie que l'élément est présent.*

*→ 0 signifie qu'il n'est pas présent.*

### *4. Supprimer un élément du set*

```
127.0.0.1:6379> SREM meset "Tunis"
(integer) 1
127.0.0.1:6379>
```

***Interprétation :***

- `Tunis` *est retiré définitivement du set.*

*Opérations ensemblistes (très utiles !)*

*Exemple complet :*

```
127.0.0.1:6379> SADD etudiants "Samir"
(integer) 1
127.0.0.1:6379> SADD etudiants "Farah"
(integer) 1
127.0.0.1:6379> SADD etudiants "Samir"
(integer) 0
127.0.0.1:6379>
127.0.0.1:6379> SADD presence "Farah"
(integer) 1
127.0.0.1:6379> SADD presence "Amine"
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379> SMEMBERS etudiants
1) "Samir"
2) "Farah"
127.0.0.1:6379> SMEMBERS presence
1) "Farah"
2) "Amine"
127.0.0.1:6379>
127.0.0.1:6379> SINTER etudiants presence   # étudiants présents
(empty array)
127.0.0.1:6379> SDIFF etudiants presence    # étudiants absents
1) "Samir"
127.0.0.1:6379>
```

***Interprétation :***

*Le set **etudiants** contient deux étudiants : Samir et Farah.*
*Redis ignore automatiquement le doublon Samir" grâce à l'unicité des sets.*

*Le set **presence** contient Farah et Amine.*

***Intersection (SINTER)***
*→ Élément commun entre les deux sets = "Farah"*
*→ Farah est une étudiante présente.*

***Différence (SDIFF)***
***→ Élément dans etudiants mais absent dans presence = "Samir"***
*→ Samir est absent.*

## REDIS 2 : Manipulations des Sets Ordonnés (Sorted Sets) dans Redis

### But des sets ordonnés

Les **sorted sets** (ZSET) sont un type de données Redis qui permet d'associer :

- **une valeur** (ex : un nom d'étudiant)
- **un score numérique** (ex : un classement, une note, un âge…)

Chaque élément est **unique**, mais son score permet de le classer automatiquement.

### 1. Ajouter des éléments avec un score : ZADD

```
127.0.0.1:6379> ZADD score4 19 "Augustin"
(integer) 1
127.0.0.1:6379> ZADD score4 18 "Ines"
(integer) 1
127.0.0.1:6379> ZADD score4 10 "Samir"
(integer) 1
127.0.0.1:6379> ZADD score4  8 "Philippe"
(integer) 1
127.0.0.1:6379>
127.0.0.1:6379>
```

**Interprétation :**

- `score4` = nom du sorted set
- chaque élément a un score → Redis peut automatiquement les trier
- si un score change, l'ordre change automatiquement

### 2. Afficher tous les éléments triés : ZRANGE

```
127.0.0.1:6379> ZRANGE score4 0 -1
1) "Philippe"
2) "Samir"
3) "Ines"
4) "Augustin"
127.0.0.1:6379>
```

→ Redis affiche les éléments **du plus petit score au plus grand score**

**3. Afficher un intervalle précis :**

```
127.0.0.1:6379> ZRANGE score4 0 1
1) "Philippe"
2) "Samir"
127.0.0.1:6379>
```

**Interprétation :**

- **Affiche les deux premiers du classement**
- **ici : *Philippe* et *Samir***

***4. Afficher dans l'ordre décroissant :***

```
127.0.0.1:6379> ZREVRANGE score4 0 -1
1) "Augustin"
2) "Ines"
3) "Samir"
4) "Philippe"
127.0.0.1:6379>
```

***Interprétation :***

- *Même principe que `ZRANGE`, mais **ordre inverse***

- → *du plus grand score au plus petit*

***5. Obtenir le rang d'un élément :***

```
127.0.0.1:6379> ZRANK score4 "Augustin"
(integer) 3
127.0.0.1:6379>
```

**Interprétation :**

- Redis compte à partir de 0
- position 3 = 4e place dans l'ordre croissant
- classements :
  0 → Philippe
  1 → Samir
  2 → Ines
  3 → Augustin

**Manipulations sur les Hashes Redis**

**But des Hashes dans Redis**

**Les hashes permettent de stocker plusieurs champs et valeurs sous une seule clé.**
**C'est comme une ligne d'un tableau ou un petit objet JSON.**

**Exemple :**

```
user:1001  →  { name: "Samir", age: 24, city: "Paris" }
```

**Pourquoi c'est utile ?**

- **Cela regroupe toutes les informations d'un objet *dans une seule clé*.**

- **C’est plus efficace et plus compact qu’avoir plusieurs clés séparées.**

- **Redis peut répartir les hashes facilement entre les nœuds d’un cluster → meilleure distribution des données.**

**1. Création d’un hash : HSET**

```
127.0.0.1:6379> HSET user:1001 name "Samir" age "24" city "Paris"
(integer) 3
127.0.0.1:6379>
```

**Interprétation :**

**On crée un utilisateur avec trois champs :**

- **`name = Samir`**

- **`age = 24`**

- **`city = Paris`**

**Tous regroupés sous la clé user:1001.**

---

**2. Lire un champ spécifique : HGET**

**Redis répo**

```
127.0.0.1:6379> HGET user:1001 name
"Samir"
127.0.0.1:6379>
```

**Interprétation :**

**On récupère uniquement le champ demandé → efficace et rapide.**

---

### 3. Lire tous les champs : HGETALL

```
127.0.0.1:6379> HGETALL user:1001
1) "name"
2) "Samir"
3) "age"
4) "24"
5) "city"
6) "Paris"
127.0.0.1:6379>
```

**Interprétation :**

**Redis renvoie tous les couples champ / valeur du hash → c'est comme afficher un objet complet.**

---

---

### 7. Compter le nombre de champs : HLEN

```
127.0.0.1:6379> HLEN user:1001
(integer) 3
127.0.0.1:6379>
```

**Retourne le nombre de champs dans le hash.**

**Conclusion : Redis ne fait pas de jointures comme les bases relationnelles, car chaque clé est indépendante et n'a pas de schéma commun avec les autres. Cette simplicité permet à Redis d'être très rapide et surtout de pouvoir se distribuer facilement sur plusieurs serveurs, ce qui facilite le passage à l'échelle.**

**REDIS 3 :**

client 1 : abonné (lecture) va s'inscrire à un cours

```
ing@ing:~$ redis-cli
127.0.0.1:6379> subscribe mescours user:1
1) "subscribe"
2) "mescours"
3) (integer) 1
1) "subscribe"
2) "user:1"
3) (integer) 2
Reading messages... (press Ctrl-C to quit or any key to type command)
```

Le client 2 : éditeur (envoi) publie un message :

```
127.0.0.1:6379> publish mescours "un nouveau cours sur mongoDB"
(integer) 1
127.0.0.1:6379>
```

Donc le client 1 va recevoir le message :

```
ing@ing:~$ redis-cli
127.0.0.1:6379> subscribe mescours user:1
1) "subscribe"
2) "mescours"
3) (integer) 1
1) "subscribe"
2) "user:1"
3) (integer) 2
1) "message"
2) "mescours"
3) "un nouveau cours sur mongoDB"
Reading messages... (press Ctrl-C to quit or any key to type command)
```

et il peut répondre :

```
127.0.0.1:6379> PUBLISH user:1 "Bonjour user 1 !"
(integer) 0
127.0.0.1:6379>
```