TD : Introduction à Redis

1. Mise en place de l'environnement Redis via Docker

1.1 Pré-requis

- Windows 11
- Docker Desktop installé et lancé
- VS Code installé
- Terminal : PowerShell

1.2 Récupération de l'image Redis

Dans PowerShell :

```
docker pull redis:latest
```

1.3 Lancer un conteneur Redis

```
docker run -d --name redisTD -p 6379:6379 redis
```

- Redis écoute maintenant sur le port **6379**.

1.4 Connexion au CLI Redis

```
docker exec -it redisTD redis-cli
```

Vous êtes maintenant dans le shell Redis.

1.5 Test rapide de fonctionnement

```
SET test "Hello Redis"
GET test
```

Résultat attendu : `"Hello Redis"`

---

2. Redis : rappels théoriques essentiels (du cours)

> Les éléments ci-dessous sont tirés directement du contenu du cours □filecite□turn0file0□.

2.1 Pourquoi Redis ?

- Base **clé/valeur** ultra rapide (**O(1)**)
- Architecture **in-memory** : < 1 ms
- Support de **structures de données natives** : Strings, Lists, Sets, Hashes, Sorted Sets
- Protocole **RESP** simple et lisible
- Grande force : opérations atomiques, pipelining, TTL, Lua scripting

---

3. Les commandes de base : manipuler les 5 structures essentielles

---

3.1 Strings (chaînes)

Équivalent de variables simples : texte, nombres, JSON, binaire.

Commandes à tester : Strings

```
SET username "alice"
GET username
INCR compteur
DECR compteur
SETEX token 30 abc123
TTL token
```

3.2 Lists (listes chaînées)

Idéales pour files, logs, activités récentes.

Commandes à tester : Lists

```
LPUSH tasks "task1"
RPUSH tasks "task2"
LRANGE tasks 0 -1
LPOP tasks
RPOP tasks
```

3.3 Sets (ensembles uniques)

Idéal pour unicité, tags, intersections.

Commandes à tester : Sets

```
SADD tags "redis"
SADD tags "database"
SADD tags "redis"   # ne s'ajoute pas une 2e fois
SMEMBERS tags
SISMEMBER tags "redis"
```

Exercice : intersection

```
SADD user1:likes "sport" "cinema"
SADD user2:likes "cinema" "voyage"
SINTER user1:likes user2:likes
```

3.4 Hashes (objets structurés)

Idéal pour représenter un utilisateur, un produit, un profil.

Commandes à tester : Hashes

```
HSET user:1000 name "Alice" age 30 city "Paris"
HGET user:1000 name
HGETALL user:1000
```

Intérêt rappelé dans le cours

- Mise à jour partielle efficace
- Économie mémoire

3.5 Sorted Sets (ZSets)

Idéals pour classements, séries temporelles, files prioritaires.

Commandes à tester : ZSets

```
ZADD leaderboard 150 "Alice"
ZADD leaderboard 200 "Bob"
ZADD leaderboard 180 "Charlie"
ZRANGE leaderboard 0 -1 WITHSCORES
ZREVRANGE leaderboard 0 -1 WITHSCORES
```

---

4. TTL, transactions, pipelining et scripting Lua

4.1 Gestion du TTL

```
SET session:user1 "active"
EXPIRE session:user1 60
TTL session:user1
```

4.2 Transactions

```
MULTI
SET a 10
INCR a
EXEC
```

4.3 Pipelining (via redis-cli)

```
(echo "SET p1 1"; echo "SET p2 2"; echo "SET p3 3") | redis-cli --pipe
```

4.4 Script Lua simple

Décrémenter un solde seulement si suffisant.

```
EVAL "local b = redis.call('GET', KEYS[1]); if tonumber(b) >= tonumber(ARGV[1])
then return redis.call('DECRBY', KEYS[1], ARGV[1]); end; return -1;" 1 balance 20
```

---

5. Mini-projet Redis — Version enrichie + Jeu de données complet** : Système de likes & classement en temps réel

Ce mini-projet est **guidé pas à pas**. Tu disposes :

- d'un **jeu de données complet fourni** (articles + utilisateurs),
- de **toutes les commandes Redis à copier / coller**,
- d'exercices d'analyse à faire à la fin.

5.1 Objectifs pédagogiques

À la fin du mini-projet, tu sauras :

- modéliser un problème simple en **clé/valeur** en combinant plusieurs types Redis,
- instancier un jeu de données complet **dans Redis**,
- manipuler :

  - **Strings** (compteurs de likes),
  - **Sets** (utilisateurs uniques par article),
  - **Lists** (journal des événements),
  - **Sorted Sets** (classement des articles),

- exécuter des requêtes simples d'**analyse en temps réel**.

5.2 Préparation de l'environnement (rappel)

Étape 1 – Vérifier que le conteneur Redis tourne

Dans **PowerShell** :

```
docker ps
```

Tu dois voir un conteneur nommé `redisTD`. Si ce n'est pas le cas :

```
docker run -d --name redisTD -p 6379:6379 redis
```

Étape 2 – Ouvrir un shell Redis

```
docker exec -it redisTD redis-cli
```

Le prompt doit ressembler à :

```
127.0.0.1:6379>
```

Étape 3 – (Optionnel) Réinitialiser la base pour le mini-projet

⚠ Cela supprime toutes les données existantes dans Redis.

```
FLUSHALL
```

5.3 Jeu de données fourni

5.3.1 Articles

Nous allons gérer 5 articles de blog :

ID	Clé Redis	Titre	Catégorie
101	`article:101`	Introduction à Redis	backend
102	`article:102`	Caching HTTP pour vos APIs	performance
103	`article:103`	Files de messages avec Redis Lists	architecture
104	`article:104`	Rate limiting avec Redis	sécurité
105	`article:105`	Classements temps réel avec Sorted Sets	temps-réel

Chaque article sera stocké dans un **Hash** :

```
HSET article:{id} title "..." category "..."
```

5.3.2 Utilisateurs

Nous simulerons 6 utilisateurs :

ID utilisateur	Identifiant logique
u1	user:001
u2	user:002
u3	user:003
u4	user:004
u5	user:005
u6	user:006

On ne va pas détailler les profils utilisateurs (ce n'est pas le but), on utilisera **seulement leurs IDs** dans les événements.

5.4 Instanciation du jeu de données dans Redis

5.4.1 Création des Hashes d'articles

Dans le shell `redis-cli` (après `docker exec -it redisTD redis-cli`), **copie/colle** le bloc suivant :

```
HSET article:101 title "Introduction à Redis" category "backend"
HSET article:102 title "Caching HTTP pour vos APIs" category "performance"
HSET article:103 title "Files de messages avec Redis Lists" category
"architecture"
HSET article:104 title "Rate limiting avec Redis" category "sécurité"
HSET article:105 title "Classements temps réel avec Sorted Sets" category "temps-
réel"
```

5.4.2 Vérification

Teste au moins un article :

```
HGETALL article:101
```

Tu dois voir les champs `title` et `category`.

5.5 Modélisation des likes & événements

Nous utilisons les clés suivantes :

- Compteur de likes par article (**String**) :
  - `article:{id}:likes`
- Ensemble des utilisateurs ayant liké un article (**Set**) :
  - `article:{id}:users`
- Classement global des articles (**Sorted Set**) :
  - `leaderboard:articles`
- Journal texte des événements de likes (**List**) :
  - `events:likes`

Idée générale : quand un utilisateur like un article :

1. vérifier qu'il ne l'a pas déjà liké (Set),
2. si nouveau like :
   - incrémenter le compteur (String),
   - mettre à jour le classement (ZSet),
   - pousser un message dans le journal (List).

5.6 Définir la fonction "like" (séquence de commandes)

On va simuler la fonction applicative "*like*" manuellement avec des commandes Redis.

Pour un utilisateur `user:001` qui like l'article `101` :

```
SADD article:101:users user:001
```

- Si la commande retourne `1` → c'est un **nouveau like**, on exécute alors :

```
INCR article:101:likes
ZINCRBY leaderboard:articles 1 101
LPUSH events:likes "user:001 liked article 101"
```

- Si la commande retourne `0` → l'utilisateur avait **déjà liké** cet article : **ne rien faire** (ou juste écrire un message dans le journal si tu veux tracer).

5.7 Scénario complet de likes (jeu de données fourni)

Nous allons maintenant **rejouer un scénario réaliste** avec nos 6 utilisateurs et 5 articles.

5.7.1 Initialisation (remise à zéro des structures de likes)

Dans `redis-cli`, exécute :

```
DEL leaderboard:articles
DEL events:likes
DEL article:101:likes article:102:likes article:103:likes article:104:likes
article:105:likes
DEL article:101:users article:102:users article:103:users article:104:users
article:105:users
```

5.7.2 Scénario fourni

Nous allons simuler les actions suivantes :

1. `user:001` like les articles 101 et 102
2. `user:002` like les articles 101, 103, 105
3. `user:003` like les articles 102 et 105
4. `user:004` like les articles 101 et 104
5. `user:005` like les articles 103 et 105
6. `user:006` like les articles 101, 102, 103

Pour chaque like, applique **le schéma standard** :

> 1. `SADD` sur `article:{id}:users`
> 2. si retour = 1 → `INCR`, `ZINCRBY`, `LPUSH`

Bloc de commandes détaillé à exécuter (TU PEUX LE COPIER/COLLER)

> 💡 Tu peux exécuter ces commandes **une par une** en observant l'évolution, ou par blocs.

Likes de `user:001`

```

```

```
SADD article:101:users user:001
INCR article:101:likes
ZINCRBY leaderboard:articles 1 101
LPUSH events:likes "user:001 liked article 101"

SADD article:102:users user:001
INCR article:102:likes
ZINCRBY leaderboard:articles 1 102
LPUSH events:likes "user:001 liked article 102"
```

Likes de `user:002`

```
SADD article:101:users user:002
INCR article:101:likes
ZINCRBY leaderboard:articles 1 101
LPUSH events:likes "user:002 liked article 101"

SADD article:103:users user:002
INCR article:103:likes
ZINCRBY leaderboard:articles 1 103
LPUSH events:likes "user:002 liked article 103"

SADD article:105:users user:002
INCR article:105:likes
ZINCRBY leaderboard:articles 1 105
LPUSH events:likes "user:002 liked article 105"
```

Likes de `user:003`

```
SADD article:102:users user:003
INCR article:102:likes
ZINCRBY leaderboard:articles 1 102
LPUSH events:likes "user:003 liked article 102"

SADD article:105:users user:003
INCR article:105:likes
ZINCRBY leaderboard:articles 1 105
LPUSH events:likes "user:003 liked article 105"
```

Likes de `user:004`

```
SADD article:101:users user:004
INCR article:101:likes
ZINCRBY leaderboard:articles 1 101
LPUSH events:likes "user:004 liked article 101"
```

```
SADD article:104:users user:004
INCR article:104:likes
ZINCRBY leaderboard:articles 1 104
LPUSH events:likes "user:004 liked article 104"
```

Likes de `user:005`

```
SADD article:103:users user:005
INCR article:103:likes
ZINCRBY leaderboard:articles 1 103
LPUSH events:likes "user:005 liked article 103"

SADD article:105:users user:005
INCR article:105:likes
ZINCRBY leaderboard:articles 1 105
LPUSH events:likes "user:005 liked article 105"
```

Likes de `user:006`

```
SADD article:101:users user:006
INCR article:101:likes
ZINCRBY leaderboard:articles 1 101
LPUSH events:likes "user:006 liked article 101"

SADD article:102:users user:006
INCR article:102:likes
ZINCRBY leaderboard:articles 1 102
LPUSH events:likes "user:006 liked article 102"

SADD article:103:users user:006
INCR article:103:likes
ZINCRBY leaderboard:articles 1 103
LPUSH events:likes "user:006 liked article 103"
```

5.8 Vérifier et analyser les données

5.8.1 Compteurs de likes par article (Strings)

```
GET article:101:likes
GET article:102:likes
GET article:103:likes
GET article:104:likes
GET article:105:likes
```

> **Exercice 1 :** noter le nombre de likes pour chaque article et vérifier qu'il correspond bien au scénario.

5.8.2 Utilisateurs uniques par article (Sets)

```
SMEMBERS article:101:users
SCARD article:101:users
```

> **Exercice 2 :** vérifier qu'il n'y a pas de doublon utilisateur par article (le cardinal doit être égal au nombre de likes).

Teste aussi pour un autre article :

```
SMEMBERS article:105:users
SCARD article:105:users
```

5.8.3 Classement global (Sorted Set)

Afficher le classement du plus liké au moins liké :

```
ZREVRANGE leaderboard:articles 0 -1 WITHSCORES
```

> **Exercice 3 :**
>
> 1. Identifier le **top 3** des articles.
> 2. Comparer les scores affichés avec les `GET article:{id}:likes`.

5.8.4 Journal des événements (List)

Afficher les 10 derniers événements :

```
LRANGE events:likes 0 9
```

> **Remarque :** la List se remplit côté **gauche** via `LPUSH`, donc l'élément le plus récent est à l'index 0.
> **Exercice 4 :** vérifier que l'ordre des événements correspond bien à l'ordre dans lequel tu as exécuté les commandes.

5.9 Extensions possibles (pour aller plus loin)

Si tu as du temps, tu peux :

- ajouter un **TTL** sur les likes d'un article (ex : un like ne compte que pendant 7 jours),
- ajouter une seconde List par article : `article:{id}:events`,
- stocker des informations basiques sur les utilisateurs dans des **Hashes** (`user:001`, `user:002`, ...),
- écrire un **script Lua** qui encapsule la logique du like (SADD + INCR + ZINCRBY + LPUSH) en une seule commande.

---

6. Exercice final d'entraînement (avec son propre jeu de données)

L'objectif est d'appliquer **toutes** les notions du TD : Strings, Sets, Lists, Hashes, Sorted Sets, TTL.

6.1 Contexte : système de paniers e-commerce

Tu dois modéliser le fonctionnement simplifié d'un site de vente.

Jeu de données fourni : Produits

ID	Nom	Prix
p1	Clavier mécanique	79
p2	Souris gamer	59
p3	Écran 27 pouces	229
p4	Casque audio	129

À instancier dans Redis :

```
HSET product:p1 name "Clavier mécanique" price 79
HSET product:p2 name "Souris gamer" price 59
HSET product:p3 name "Écran 27 pouces" price 229
HSET product:p4 name "Casque audio" price 129
```

6.2 Règles du système à implémenter

Chaque utilisateur possède :

- un **panier** (Set) → `cart:{user}`
- un **montant total** (String) → `cart:{user}:total`
- un **historique des actions** (List) → `cart:{user}:events`
- un **classement des utilisateurs qui dépensent le plus** (Sorted Set) → `leaderboard:spenders`

6.3 Scénario à exécuter

Reproduire les actions suivantes :

1. `user:001` ajoute p1 et p2
2. `user:002` ajoute p3, p4, p1
3. `user:003` ajoute p2, p2 (2e ajout ignoré), p4

Pour chaque ajout **si nouveau produit** :

- ajouter au panier : `SADD cart:{user} {productId}`
- ajouter au total : `INCRBY cart:{user}:total {prix}`
- journaliser : `LPUSH cart:{user}:events "{user} added {productId}"`
- mettre à jour le leaderboard : `ZINCRBY leaderboard:spenders {prix} {user}`

6.4 Travail demandé

Question 1 — Totaux par utilisateur

Afficher :

```
GET cart:user:001:total
GET cart:user:002:total
GET cart:user:003:total
```

Question 2 — Vérifier les paniers

```
SMEMBERS cart:user:001
SCARD cart:user:001
```

Question 3 — Afficher le classement des meilleurs clients

```
ZREVRANGE leaderboard:spenders 0 -1 WITHSCORES
```

Question 4 — Afficher l'historique d'un utilisateur

```
LRANGE cart:user:002:events 0 10
```