Université Sorbonne Paris Nord

Institut sup guallilé

Compte Rendu TP Redis

Bases de données NoSQLs

Nom : Ben Saad
Prénom :Chayma.
Groupe : Ing3 info

Année universitaire : 2024/2025

Table des matières

1 Introduction aux Bases de Données **2**
1.1 Bases de Données Relationnelles 2
1.2 Bases de Données NoSQL 3
1.3 Les 4 Familles de Bases de Données NoSQL 4
1.3.1 1. Bases orientées Documents 4
1.3.2 2. Bases orientées Clés-Valeurs 4
1.3.3 3. Bases orientées Colonnes 4
1.3.4 4. Bases orientées Graphes 4

2 Introduction à Redis **5**
2.1 Qu'est-ce que Redis ? 5
2.2 Manipulation avec Redis sur Docker 6
2.2.1 Conteneur Redis et accès à `redis-cli` 6
2.2.2 Stockage des données et persistance 6
2.2.3 Création et manipulation de paires clé-valeur 6
2.2.4 Compteur de visiteurs 6
2.2.5 Expiration et durée de vie des clés 7
2.2.6 Utilisation d'autres structures de données 7
2.2.7 Sorted Sets (ZSET) - classement par score 9
2.2.8 Hashes (HSET, HGET, HMSET) - stockage d'objets utilisateur . . 9
2.2.9 Publication / Abonnement (Pub/Sub) et gestion de bases multiples 11
2.2.10 Gestion de plusieurs bases de données 11

Chapitre 1

Introduction aux Bases de Données

1.1 Bases de Données Relationnelles

Les bases de données relationnelles (SQL) sont organisées sous forme de tables contenant des lignes (tuples) et des colonnes (attributs). Elles utilisent un schéma strict, et les relations entre les tables sont définies par des clés primaires et étrangères. Elles sont basées sur le langage SQL (Structured Query Language) pour effectuer des opérations de requêtes, création, manipulation et contrôle des données.

Caractéristiques principales :

— Schéma fixe et structuré.
— Données organisées en tables.
— Support des relations complexes (join, contraintes).
— Cohérence et intégrité des données (ACID).

Exemples : MySQL, PostgreSQL, Oracle, SQL Server.

1.2 Bases de Données NoSQL

Les bases de données NoSQL (Not Only SQL) ont été créées pour répondre aux limitations des bases relationnelles lorsque les données sont volumineuses, non structurées ou lorsque les performances (scalabilité) sont prioritaires.

Caractéristiques principales :

— Pas de schéma fixe (flexibles).

— Haute performance et scalabilité horizontale.

— Adaptées aux données non structurées (JSON, graphiques, colonnes, etc.).

— Faible complexité pour certaines opérations.

Exemples : MongoDB, Redis, Cassandra, Neo4j.

1.3 Les 4 Familles de Bases de Données NoSQL

Les bases NoSQL se divisent en quatre grandes familles :

1.3.1 1. Bases orientées Documents

Stockent des données sous forme d'objets JSON ou BSON. **Exemples :** MongoDB, CouchDB.

1.3.2 2. Bases orientées Clés-Valeurs

Stockent des données sous forme de paires clé-valeur. Simples, ultra rapides. **Exemples :** Redis, Riak, DynamoDB.

1.3.3 3. Bases orientées Colonnes

Optimisées pour l'analyse de grandes quantités de données. **Exemples :** Cassandra, HBase.

1.3.4 4. Bases orientées Graphes

Modélisent les relations entre entités sous forme de nœuds et arêtes. **Exemples :** Neo4j, OrientDB.

Chapitre 2

Introduction à Redis

2.1 Qu'est-ce que Redis ?

Redis est une base de données NoSQL de type clé-valeur, stockée en mémoire (In-Memory Database), ce qui lui permet d'être extrêmement rapide. Son nom signifie **Remote Dictionary Server**.

Principales caractéristiques :

— Stockage en mémoire (In-Memory).

— Très haute performance.

— Support de structures avancées : listes, ensembles, hashes, sorted sets.

— Utilisée dans cache, sessions web, file d'attente.

2.2 Manipulation avec Redis sur Docker

2.2.1 Conteneur Redis et accès à `redis-cli`

Explication : Le prompt `127.0.0.1:6379>` indique que nous sommes maintenant connectés au serveur Redis et prêts à exécuter des commandes. Le serveur tourne en local sur l'adresse `127.0.0.1` et écoute sur le port `6379`.

```
docker run -d redis
docker exec -it redis-server redis-cli
127.0.0.1:6379>
```

2.2.2 Stockage des données et persistance

Explication : Généralement, les données sont persistées dans la RAM, mais on peut également les persister sur disque pour conserver les informations entre les redémarrages du serveur Redis.

2.2.3 Création et manipulation de paires clé-valeur

Explication : Créer une paire clé-valeur où `"user:1234"` est la clé et `"SAMIR"` est la valeur. Pour récupérer la valeur, on utilise `GET`. Pour supprimer une clé, on utilise `DEL`, le retour `(integer) 1` indique que la clé existait et a été supprimée.

```
127.0.0.1:6379> SET demo "Bonjour"
OK
127.0.0.1:6379> SET user:1234 "SAMIR"
OK
127.0.0.1:6379> GET user:1234
"SAMIR"
127.0.0.1:6379> DEL user:1234
(integer) 1
```

2.2.4 Compteur de visiteurs

Explication : Parmi les utilisations classiques de Redis, on peut compter le nombre de visiteurs. On initialise le compteur à 0 avec `SET 27novembre 0` et on l'incrémente avec `INCR 27novembre`. Redis gère la concurrence si plusieurs utilisateurs se connectent en même temps. Pour décrémenter, on utilise `DECR`, le retour montre le nombre actuel de visiteurs.

```
127.0.0.1:6379> SET 27novembre 0
OK
127.0.0.1:6379> INCR 27novembre
(integer) 1
127.0.0.1:6379> INCR 27novembre
(integer) 2
127.0.0.1:6379> INCR 27novembre
(integer) 3
127.0.0.1:6379> INCR 27novembre
(integer) 4
127.0.0.1:6379> INCR 27novembre
(integer) 5
```

2.2.5 Expiration et durée de vie des clés

Explication : On peut définir une durée de vie pour une clé. `TTL macle` retourne le temps restant avant expiration. Si le retour est `-1`, la clé n'a pas de limite de vie et restera en RAM jusqu'à saturation, moment où Redis supprimera certaines clés automatiquement. Pour définir un délai d'expiration, on utilise `EXPIRE macle 150` (en secondes).

```
127.0.0.1:6379> SET macle "chayma"
OK
127.0.0.1:6379> TTL macle
(integer) -1
127.0.0.1:6379> EXPIRE macle 150
(integer) 1
127.0.0.1:6379> TTL macle
(integer) 143
```

2.2.6 Utilisation d'autres structures de données

Listes

Explication : Les listes permettent de stocker des valeurs ordonnées. `RPUSH` ajoute un élément à droite, `LRANGE` affiche les éléments (préciser l'indice de début et de fin, `-1` pour tout afficher), `LPOP` supprime à gauche et `RPOP` à droite.

```
127.0.0.1:6379> RPUSH mesCours "BDA"
(integer) 1
127.0.0.1:6379> RPUSH mesCours "Services web"
(integer) 2
```

```
127.0.0.1:6379> LRANGE mesCours 0 -1
1) "BDA"
2) "Services web"
127.0.0.1:6379> LPOP mesCours
"BDA"
```

Sets

Explication : Les sets stockent des valeurs uniques, sans ordre. `SADD` ajoute un élément, `SMEMBERS` liste tous les éléments, `SUNION` fait l'union de deux sets.

```
127.0.0.1:6379> SADD utilisateurs "Chayma"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "ines"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "jhon"
(integer) 1
127.0.0.1:6379> SADD utilisateurs "joseph"
(integer) 1
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Chayma"
2) "ines"
3) "jhon"
4) "joseph"
127.0.0.1:6379> SREM utilisateurs "jhon"
(integer) 1
127.0.0.1:6379> SMEMBERS utilisateurs
1) "Chayma"
2) "ines"
3) "joseph"
127.0.0.1:6379> SADD utilisateurs2 "karim"
(integer) 1
127.0.0.1:6379> SUNION utilisateurs utilisateurs2
1) "karim"
2) "ines"
3) "joseph"
4) "Chayma"
```

2.2.7 Sorted Sets (ZSET) - classement par score

Explication : Les *Sorted Sets* sont des ensembles ordonnés où chaque élément possède un score. Cela permet, par exemple, de classer des utilisateurs selon leurs points ou notes. On utilise `ZADD` pour ajouter un élément avec son score. `ZRANGE` permet de récupérer les éléments dans l'ordre croissant des scores, et `ZREVRANGE` dans l'ordre décroissant. Pour connaître la position d'un élément, on utilise `ZRANK`.

```
127.0.0.1:6379> ZADD notes 19 "Chayma"
(integer) 1
127.0.0.1:6379> ZADD notes 20 "Karima"
(integer) 1
127.0.0.1:6379> ZADD notes 18 "Salima"
(integer) 1
127.0.0.1:6379> ZADD notes 20 "Sihem"
(integer) 1
127.0.0.1:6379> ZRANGE notes 0 -1
1) "Salima"
2) "Chayma"
3) "Karima"
4) "Sihem"
127.0.0.1:6379> ZRANGE notes 0 1
1) "Salima"
2) "Chayma"
127.0.0.1:6379> ZRANK notes "Salima"
(integer) 0
127.0.0.1:6379> ZRANK notes "CHAYMA"
(nil)
127.0.0.1:6379> ZRANK notes "Chayma"
(integer) 1
```

Note pédagogique : Les scores permettent de classer les éléments, et l'indice commence à 0. Le `nil` indique que l'élément n'existe pas. Cela peut être utilisé dans des systèmes de recommandation, de leaderboard ou pour trier des éléments par importance.

2.2.8 Hashes (HSET, HGET, HMSET) - stockage d'objets utilisateur

Explication : Les hashes permettent de stocker des objets complexes sous forme de champs et valeurs. C'est utile pour représenter un utilisateur avec plusieurs attributs (nom, âge, email...).

```
127.0.0.1:6379> HSET user:11 username "Chayma"
(integer) 1
127.0.0.1:6379> HSET user:11 age "25"
(integer) 1
127.0.0.1:6379> HSET user:11 email "chayma.bensaad@gmail.com"
(integer) 1
127.0.0.1:6379> HMSET user:12 username "karima" age "40" email "kariima@gmail.com"
OK
127.0.0.1:6379> HGETALL user:11
1) "username"
2) "Chayma"
3) "age"
4) "25"
5) "email"
6) "chayma.bensaad@gmail.com"
127.0.0.1:6379> HGETALL user:12
1) "username"
2) "karima"
3) "age"
4) "40"
5) "email"
6) "kariima@gmail.com"
127.0.0.1:6379> HINCRBY user:12 age 4
(integer) 44
```

Explications complémentaires :

— `HSET` ajoute un champ et sa valeur à une clé hash.

— `HMSET` permet d'ajouter plusieurs champs en une seule commande.

— `HGETALL` retourne tous les champs et valeurs d'un hash.

— `HINCRBY` incrémente un champ numérique directement dans le hash.

Avantages pédagogiques :

— Pas de schéma fixe, donc chaque hash peut avoir des champs différents.

— Permet une répartition sur plusieurs noeuds pour le passage à l'échelle.

— Réduit les contraintes comparé à une table relationnelle stricte.

Résumé : Redis fournit différentes structures de données adaptées à divers besoins : paires clé-valeur simples, listes pour séquences, sets pour valeurs uniques, sorted sets pour classement, et hashes pour objets complexes. Ces structures permettent d'optimiser la vitesse d'accès et le stockage en mémoire tout en conservant la possibilité de persistance et de passage à l'échelle.

2.2.9 Publication / Abonnement (Pub/Sub) et gestion de bases multiples

Explication : Redis fournit un mécanisme de communication en temps réel entre clients via le système *Publish/Subscribe*. Un client peut s'abonner à un ou plusieurs canaux et rester à l'écoute de messages. Un autre client peut publier des messages sur ces canaux, et tous les abonnés les recevront immédiatement.

— `SUBSCRIBE <canal>` : s'abonner à un canal.

— `PUBLISH <canal> <message>` : envoyer un message à tous les abonnés.

— `PSUBSCRIBE <pattern>` : s'abonner à tous les canaux correspondant à un motif.

Exemple pratique :

```
127.0.0.1:6379> SUBSCRIBE mesCours
Reading messages... (press Ctrl-C to quit)
# client 1 reste à l'écoute

127.0.0.1:6379> PUBLISH mesCours "Nouveau cours sur MongoDB"
(integer) 1
# message reçu immédiatement par tous les abonnés du canal mesCours

127.0.0.1:6379> PSUBSCRIBE mes*
# client abonné à tous les canaux commençant par "mes"

127.0.0.1:6379> PUBLISH mesNotes "Une nouvelle note est arrivée"
(integer) 1
# reçu par le client abonné au pattern mes*
```

Remarques pédagogiques :

— Seuls les clients abonnés au canal ou correspondant au pattern reçoivent les messages.

— Pub/Sub est idéal pour les applications temps réel comme les notifications, chat ou échanges de messages.

2.2.10 Gestion de plusieurs bases de données

Explication : Redis propose 16 bases par défaut (indexées de 0 à 15). La base utilisée par défaut est 0. Pour changer de base, on utilise `SELECT <numéro>`. Cela permet de séparer logiquement les données ou d'isoler différents environnements.

```
127.0.0.1:6379> SELECT 1
OK
```

```
127.0.0.1:6379> KEYS *
(empty list)
# aucune clé définie dans la base 1

127.0.0.1:6379> SELECT 0
OK
127.0.0.1:6379> KEYS *
1) "user:11"
2) "notes"
```

Remarques pédagogiques :

— Chaque base est indépendante, donc les clés d'une base ne sont pas visibles dans une autre.

— Important pour la reprise après panne : Redis ne persiste pas automatiquement toutes les données sur disque, donc il faut configurer la persistance pour récupérer les dernières données en cas de crash.