Grau Enginyeria de Dades
Curs 2023-2024
Gestió de Dades
Redis lab – Linux Ubuntu version
Based in Redis – Seven Databases in Seven Weeks - chapter 8

1. Work environment preparation

To use Redis, we first need to install the server and run it.

```
sudo apt update
sudo apt install -y redis
redis-server -v
>Redis server v=5.0.7 sha=000000:0 malloc=jemalloc-5.2.1 bits=64
build=636cde3b5c7a3923
```

If everything is fine, you should see the redis server version message and we can proceed to run the command-line interpreter or redis-cli

```
redis-cli
127.0.0.1>ping
PONG
127.0.0.1>
```

Now, we can start executing commands and using Redis functionalities.

2. Redis interactive session: String type

First thing to now is how to exit from Redis and how to find help. You must type quit to exit and help and the command name to get information on the command.

Exit from Redis:

```
redis-cli
127.0.0.1>quit
```

Getting help from Redis:

```
redis-cli
127.0.0.1>help SET
SET key value
summary: Set the string value of a key
since: 1.0.0
group: string
```

This tutorial will create a service **to shorten URL links**. That is, given a long URL like *http://thisisaveryverylongaddress.com* we will generate a short version *http://bit.ly/VLD* .

Using the last short URL redirects the user to the first longer version. We can use this functionality to type less and to get some statistics on URL usage.

We start by using the **simple STRING Redis type** to store short and long addresses.

Then, we will use a SET to key a shortcode like **uab** to a value like **http://www.uab.cat/enginyeria**. String SET needs two parameters:

- a string that will be our key: uab
- a value http://www.uab.cat/enginyeria

To retrieve the value, we just need the operation GET and the key name:

```
127.0.0.1> SET uab http://www.uab.cat/enginyeria
OK
127.0.0.1> GET uab
"http://www.uab.cat/enginyeria"
```

If we have a long list of values to store, we can use MSET with any number of key-value pairs separated by blank spaces. Similarly, we use MGET to get the multiple values from a list of keys. For example:

```
127.0.0.1> MSET gog http://www.google.es yah http://www.yahoo.com
OK
127.0.0.1> MGET gog yah
1)"http://www.google.com"
2)"http://www.yahoo.com"
```

We can add a counter to our database to keep a running total of how many short keys are in our dataset. We create a count and increment it with INCR.

```
127.0.0.1> SET count 2
OK
127.0.0.1> INCR count
(integer) 3
127.0.0.1> GET count
"3"
```

Notice that GET returns the count value as a string while INCR is recognizing it as an integer and add one to its value. If you try to increment a non-integer value, you will get an error "not an integer" message.

Exercise 1: add three of your favorite websites to the list with SET and with MSET.

Other useful commands are INCRBY or decrement (DECR, DECRBY).

2.1. Transactions

With Redis MULTI command we can wrap some operations in a single block that will complete either successfully or not at all, never end up with a partial execution of the operations. In the following example, we create a transaction with SET and INCR.

```
127.0.0.1> MULTI
OK
127.0.0.1> SET gnews http://news.google.com
QUEUED
127.0.0.1> INCR count
QUEUED
127.0.0.1> EXEC
1) OK
2) (integer) 4
```

When using MULTI, the commands are not executed when the transaction is defined. Instead, they are queued and then the instructions are executed in sequence. For all transactions in the queue, they can be removed with the use of DISCARD that will clear the queue.

2.2. Complex Datatypes to manage users and URLs

HASHES

We are going to use a hash to keep track of users who sign up for the URL-shortening service. For that we can define a simple STRING syntax: *username:<user name>* and *userpwd:<user name>* as keys to insert values in our hash

```
127.0.0.1> MSET username:luke "Luke" userpwd:luke "s3cret"
OK
127.0.0.1> MGET username:luke userpwd:luke
1) "Luke"
2) "s3cret"
```

Another possibility is to use a HASH with its own key-value pairs, then we just need to use a single Redis key to get all values of the hash. With HKEYS we can also get the hash keys.

```
127.0.0.1> HMSET user:luke name "Luke" password s3cret
OK
127.0.0.1> HVALS user:luke
1) "Luke"
2) "s3cret"
127.0.0.1> HKEYS user:luke
1) "name"
2) "password"
```

Other useful commands are HDEL, HINCRBY, HGETALL or HSETNX (set a value only if the key does not exist yet).

Exercise 2: add yourself to the hash set with HMSET using your username and a new password

LISTS

We want to allow users to keep a list of shortened URLs for later review. We will use the LIST data type to keep values as a queue (first in, first out) or as a stack (last value in, last value out).

We set the key to *username:wishlist* and push any number of values to the right end of the list. We use RPUSH to add elements to the LIST. For example, let's add three values into luke wishlist:

```
127.0.0.1> RPUSH luke:wishlist gog yah gnews
(integer) 3
127.0.0.1> LLEN luke:wishlist
3
127.0.0.1> LRANGE luke:wishlist 0 -1
1) "gog"
2) "yah"
3) "gnews"
```

We can get the LIST length at any time with LLEN and retrieve any subsection of the LIST with LRANGE by specifying the first and the last positions. Remember that first is 0 and a negative position means give all elements from the beginning to the end of the LIST.

To get a value from the LIST in the order we added them, that is, to use the list as a queue, we use the LPOP command that extracts the first element in the left of the queue

```
127.0.0.1> LPOP luke:wishlist
"gog"
```

We use RPUSH/RPOP to implement a stack accessing the end of the LIST and LPUSH and RPOP to implement a queue.

If we want to extract the URLs of the wishlist and add them to a "visited" list we must use a special single command RPOPLPUSH (right pop, left push)

```
127.0.0.1> RPOPLPUSH luke:wishlist luke:visited
"yah"
```

SETS

We are using SETS in this tutorial to build a common group of URLs for all users. SETS are a good choice as they store no duplicate values and give support for set operations like unions or intersections.

In our example we are going to categorize URLs using labels as common keys with SADD

```
127.0.0.1:6379> SADD news nytimes.com theverge.com
(integer) 2
```

We have two new values at news SET. If we need the contents of the SET, we can use SMEMBERS command:

```
127.0.0.1:6379> SMEMBERS news
1)"theverge.com"
2)"nytimes.com"
```

Now we add a new SET for technology-related news

```
127.0.0.1:6379> SADD tech theverge.com cnet.com
(integer) 2
```

Now we can apply several SET operations to obtain the intersection of news and tech, diff, union, and store union in a new set:

```
127.0.0.1:6379> SINTER news tech
1)"theverge.com"

127.0.0.1:6379> SDIFF news tech
1)"nytimes.com"

127.0.0.1:6379> SDIFF tech news
1)"cnet.com"

127.0.0.1:6379> SUNION news tech
1)"theverge.com"
2)"cnet.com"
3)"nytimes.com"

127.0.0.1:6379> SUNIONSTORE websites news tech
127.0.0.1:6379> SMEMBERS websites
1)"theverge.com"
2)"cnet.com"
3)"nytimes.com"

127.0.0.1:6379> SCARD websites
(integer) 3
```

Exercise 3: add a new type of URL called video and add there some examples: youtube.com, vimeo.com, netflix.com. Join your video datasets with websites from the previous example.

SORTED SETS

Now we want to have a ranked list of popular URLs. Each time a user visits a specific URL, its score will be increased. For that we will use sorted SETS where we have one element per each key and an integer value to describe the score of the element. We must be careful as each new insertion means a new search to keep data sorted. In our case, for each element in the sorted set we need to provide the key and the ranking

```
127.0.0.1:6379> ZADD popular 500 gog 1 yah 99 uab
(integer) 3
```

We can update the score or increment/decrement it by some number

```
127.0.0.1:6379> ZINCRBY popular 1 uab
"100"
```

Exercise 4: add two new short URLs to popular sorted list with their initial ranking values

SET RANGE QUERIES

To get some values from our popular webpages set we can use a range command that returns the values of the set by their position. This is ordered by score from lowest to highest. The top three scoring popular sites are:

```
127.0.0.1:6379> ZRANGE popular 0 2
1)"yah"
2)"upf"
3)"gog"

127.0.0.1:6379> ZREVRANGE popular 0 2
1)"gog"
2)"uab"
3)"yah"

127.0.0.1:6379> ZRANGE popular 0 1 WITHSCORES
1)"yah"
2)"1"
3)"uab"
4)"100"
```

Ranges can also be queried using scores with ZRANGEBYSCORE operations. As low and high range numbers are inclusive by default, we can make a score number exclusive by using an opening braket: "("

```
127.0.0.1:6379> ZRANGEBYSCORE popular 100 500
1)"uab"
2)"gog"

127.0.0.1:6379> ZRANGEBYSCORE popular (100 500
1)"gog"

127.0.0.1:6379> ZRANGEBYSCORE popular -inf inf
1)"yah"
2)"gog"
3)"uab"

127.0.0.1:6379> ZREVRANGEBYSCORE popular inf -inf
1)"uab"
2)"gog"
3)"yah"
```

Exercise 5: add a new item to popular ZSET with the value 9999. Check that the new item is inserted in popular by obtaining the top 5 list of items with largest decreasing score.

UNIONS

We are interested in building a destination key that contains the union or intersection of one or more keys. This is done using the union operation. We will need to worry about which keys are going to be joined and how to merge different key scores.

In our case, we want to calculate the importance of a sorted set of shortened URLs. First, we create a set with a key that scores websites by votes. Each visitor to a site can vote if they like a website and each vote adds one point.

```
127.0.0.1:6379> ZADD votes 2 yah 0 gog 1234 uab
(integer) 3
```

Now we want to get the most important websites of our system considering both votes and visits. We want to use both scores together to compute a new importance score where votes will receive the double of the weight of visits.

Exercise 6: apply the following operation

```
127.0.0.1 > ZUNIONSTORE importance 2 popular votes WEIGHTS 1 2
AGGREGATE SUM
(integer) 3
```

This is the union operation:

The syntax: ZUNIONSTORE destination numkeys key1 key2

In our example: ZUNIONSTORE importance 2 popular votes

- numkeys is 2,
- importance is the key to store the union into,
- popular and votes are the keys to apply the join operation.

ZUNIONSTORE importance 2 popular votes WEIGHTS 1 2 AGGREGATE SUM

Weight is the optional formula to calculate the resulting score for the aggregation. In this case, we use popular + (votes * 2) but it could be the minimum or maximum of the scores.

```
127.0.0.1:6379> ZRANGEBYSCORE importance -inf inf WITHSCORES
1)"yah"
2)"5"
3) "gog"
4) "500"
5) "uab"
6) "2568"
```

Summary of data models built in Redis tutorial:

- String SET for storing short names and web addresses: `SET gog www.google.com`
- String GET for retrieving long URLs from short names: `GET gog`
- String SET count for counting the number of addresses we store: `SET count 3`
- String INCR count to add 1 to current count: `INCR count`
- Hash MSET/MGET to store/get usernames and passwords: `MSET password:luke "s3cr3t" / MGET password:luke`
- Hash HMSET to use/add multiple key/values to a Hash element: `HMSET user:luke name "Luke" password s3cret`
- List RPUSH to store personal short URL lists for users. `RPUSH luke:wishlist gog`
- Set operations SADD/SMEMBERS to create sets of related URLs: `SADD news nytimes.com theverge.com`
- Set operations to look for intersection, union, ... `SINTER news tech`
- ZSET operations to keep ranked list of URLs: `ZADD popular 500 gog`
- ZSET get top lists of items: `ZRANGE popular 0 2`
- join two ZSET sorted sets to create a new aggregated ranked list with ZUNIONSTORE

Things we can do:

- store and look for shortened URLs
- Know how many URLs we have stored
- Manage users and passwords of our system

- Store personal lists of URLs
- Use set operations to manage classes of URL
- Use sorted ZSET to keep rankings
- Join ZSET to create more complex aggregations

Questions: **DUE TO October 17th**

1- Draw the **complete data model of the lab**. Draw and label all data entities, attributes and Redis data structures used and give an example of each structure with name and a data sample.

2- How would you build a **top 10 list of most active users**? Data structures? Operations?
   - What is going to be your definition of user activity? Posting a new URL, voting for a URL?
   - How are you going to measure and store details about this user activity?
   - Describe how your Redis design reacts to a new user activity action. For example, user123 votes (or posts) for www.google.com. Could this action affect your top list? How can you keep an always updated top list?
   - If user123 wants to vote again for google, you should not count this operation as a vote. How to use the property of Redis SET structures that do not store duplicated values?

3- How would you design a **simple recommender system for users**? If you have stored some user URLs of preference, how would you recommend new URLs to this user? Provide a list of Redis structures and operations needed.
   - If you have a target user, how would you find the most similar users from your database?
   - When you have found a similar user, how would you find a new relevant URL to recommend?

4- How would you obtain **the most recent/last visited** 10 web sites for each user? Top 10 must be sorted by time of visit. Last visit must be in the first position of the list. Provide a list of Redis structures and operations needed.
   - How will you manage user visits to keep a list of URLs by time?
   - How will you find 10 most recent websites?
   - How will your design react to a new (user123, "www.google.com") event? How would you update the time of the last visit to www.google.com?

Please show the use of your answers with some sample data as an example.