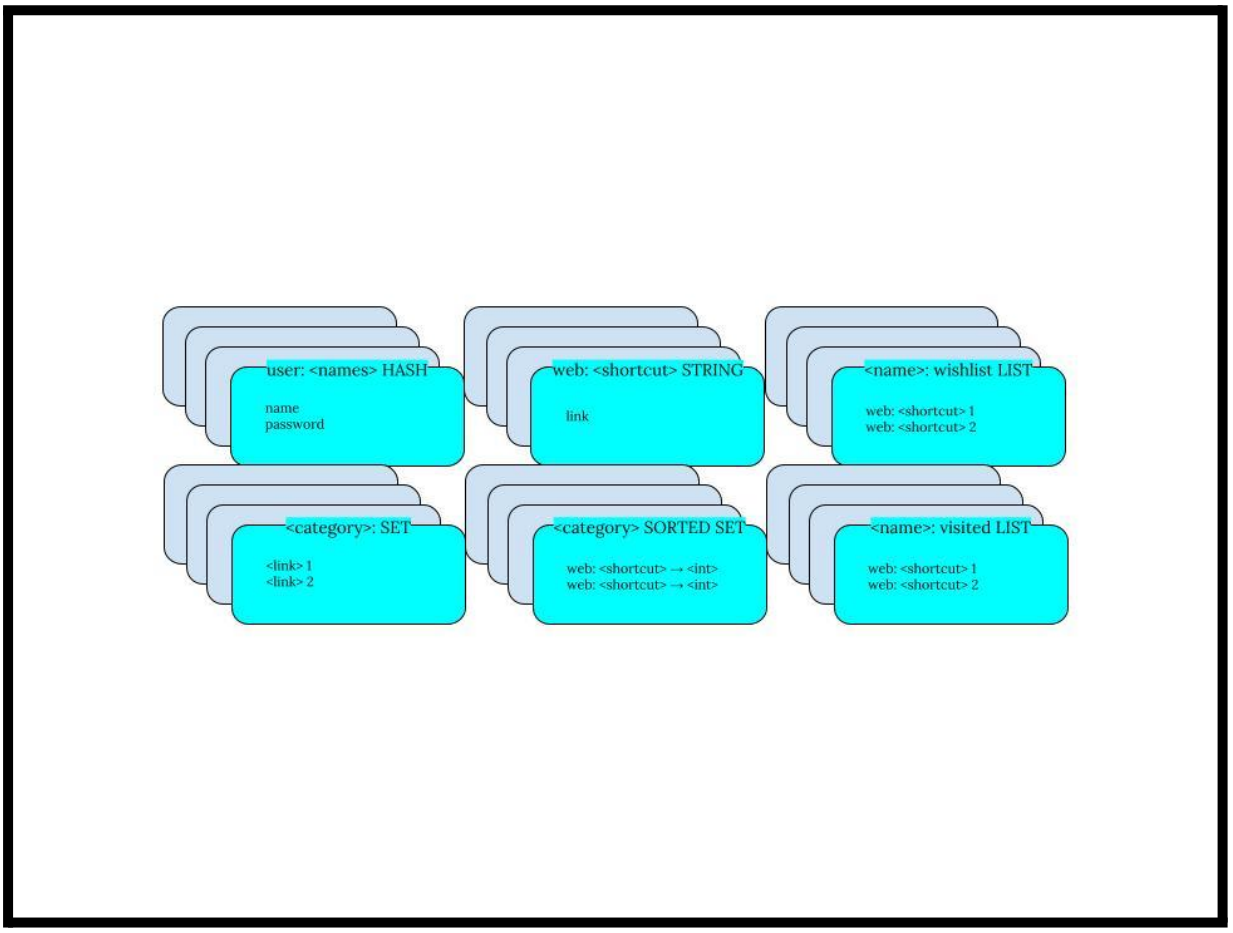
Questions Redis Lab

Pol Medina & Adrián García

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 a name and a data sample.**



Below the examples can be seen:

```
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> HMSET user:Pol name "Pol" password s3cret
OK
127.0.0.1:6379> HVALS user:Pol
1) "Pol"
2) "s3cret"
127.0.0.1:6379> SET web:uab http://www.uab.cat/enginyeria
OK
127.0.0.1:6379> GET uab
"http://www.uab.cat/enginyeria"
```

```
127.0.0.1:6379> SADD news nytimes.com theverge.com
(integer) 0
127.0.0.1:6379> SMEMBERS news
1) "theverge.com"
2) "nytimes.com"
127.0.0.1:6379> ZADD popular 500 gog 1 yah 99 uab
(integer) 3
127.0.0.1:6379> ZRANGE popular 0 2
1) "yah"
2) "uab"
3) "gog"
127.0.0.1:6379> LPUSH adrian:wishlist web:ig web:x web:drive web:linkedIn
(integer) 4
127.0.0.1:6379> LPUSH pol:wishlist web:insta web:x web:amazon web:tiktok
(integer) 4
127.0.0.1:6379> LPUSH adrian:visited_list web:uab web:skype web:tiktok web:intel
(integer) 4
127.0.0.1:6379> LPUSH pol:visited_list web:linkedIn web:drive web:hp
(integer) 3
```

2. **How would you build a top 10 list of most active users? Data structures? Operations?**

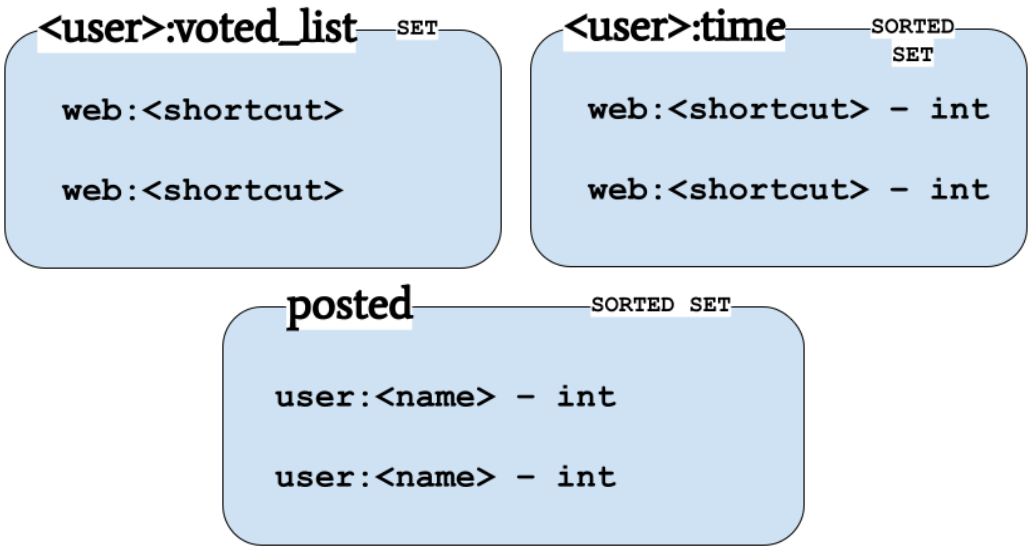
* *Note: The development of this exercise has been done by connecting the Redis client to the local host using Python.*

```python
import redis

# Connect redis and python
r = redis.StrictRedis(host="localhost", port=6379, db=0)
```

In order to build a top 10 list of most active users a data structure such as the sorted set will be used.

The main operation that will be taken into account is a weighted sum of 3 main activities: posts, user time, and the length of the voted list set.



- **What is going to be your definition of user activity? Posting a new URL, voting for a URL?**

As has been pointed out above three data structures will determine the definition of user activity.

1. posted - SORTED SET → Here each of the users in our application will be stored with an integer that represents the number of times this user has posted since the account was created.

```
# Create a sorted set in which each user has the times he/she has posted
r.zadd('posted', {
    'user:Pol': 12,
    'user:Adrian': 53,
    'user:Antonio': 70,
    'user:Alex': 10,
    'user:Jack': 45,
    'user:Roger': 26,
    'user:Joan': 19,
    'user:Xavier': 11,
    'user:Marc': 14,
    'user:Paula': 36,
    'user:Lucia': 3,
    'user:Patricia': 52,
})
```

2. user:time - SORTED SET → Here each of the websites a particular user has visited will be stored. Different websites will have a particular integer that will represent the minutes this user has spent on the website.

```
# Create sorted sets for <user>:time_spent
r.zadd('Pol:time_spent', {'yt': 120, 'google': 85, 'yah': 45, 'tv': 15})
r.zadd('Adrian:time_spent', {'google': 60, 'snap': 110, 'yt': 30})
r.zadd('Antonio:time_spent', {'acer': 50, 'hp': 70, 'uab': 25})
r.zadd('Alex:time_spent', {'J&J': 95, 'google': 105, 'apple': 40})
r.zadd('Jack:time_spent', {'insta': 295})
r.zadd('Roger:time_spent', {'pais': 90, 'microsoft': 55, 'nike': 35})
r.zadd('Joan:time_spent', {'yt': 100, 'hp': 95, 'apple': 60, 'levis': 100})
r.zadd('Xavier:time_spent', {'uab': 85, 'eastpack': 70, 'sony': 45})
r.zadd('Marc:time_spent', {'insta': 120, 'meta': 65, 'J&J': 40})
r.zadd('Paula:time_spent', {'yt': 90, 'nike': 80, 'apple': 110, 'gmail': 5, 'pytorch': 30
r.zadd('Lucia:time_spent', {'acer': 50, 'apple': 105, 'linux': 65})
r.zadd('Patricia:time_spent', {'mit': 75, 'vanguardia': 65, 'leetcode': 55})
```

3. user:voted_list - SET → In here each of the websites a particular user has voted for will be stored.

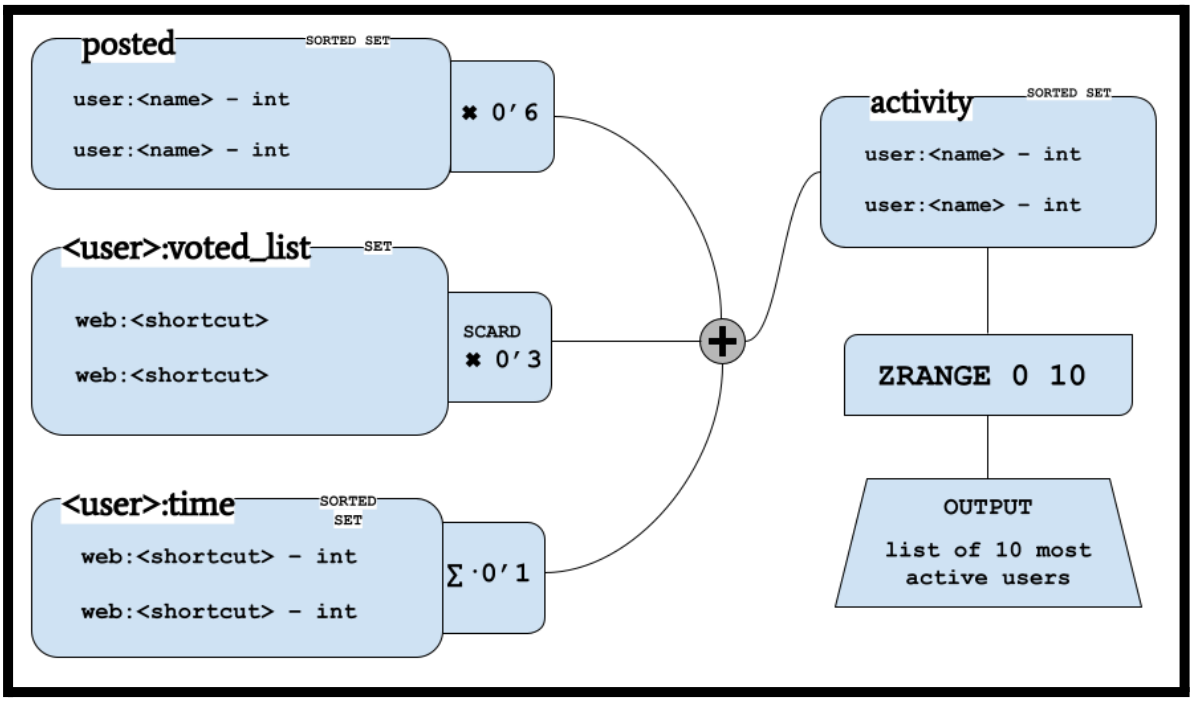
```python
# Create sets for <user>:voted_list
r.sadd('Pol:voted_list', 'yt', 'google', 'yah', 'tv')
r.sadd('Adrian:voted_list', 'google', 'snap', 'yt')
r.sadd('Antonio:voted_list', 'acer', 'hp', 'uab')
r.sadd('Alex:voted_list', 'J&J', 'google', 'apple')
r.sadd('Jack:voted_list', 'insta')
r.sadd('Roger:voted_list', 'pais', 'microsoft', 'nike')
r.sadd('Joan:voted_list', 'yt', 'hp', 'apple', 'levis')
r.sadd('Xavier:voted_list', 'uab', 'eastpack', 'sony')
r.sadd('Marc:voted_list', 'insta', 'meta', 'J&J')
r.sadd('Paula:voted_list', 'yt', 'nike', 'apple', 'gmail', 'pytorch')
r.sadd('Lucia:voted_list', 'acer', 'apple')
r.sadd('Patricia:voted_list', 'mit', 'vanguardia', 'leetcode')
```

- **How are you going to measure and store details about this user activity?**

The measurements and details about the activity of a particular user will the determined by a weighted sum.

user activity score = (*nº of posts of a user* x **0.6**) + (∑ *of minutes spent at each website by a user* x **0.1**) + (*length of the voted_list SET of a user* x **0.3**)

Once the activity score is calculated this information is introduced into the SORTED SET - activity. In this SORTED SET the score will be stored as well as the user:<name> it corresponds to.



```python
# List of all users
users = ['Pol', 'Adrian', 'Antonio', 'Alex', 'Jack', 'Roger', 'Joan', 'Xavier', 'Marc', 'Paula', 'Lucia', 'Patricia']

# Loop through each user to compute the activity score
for user in users:

    # 1. Get the score from the "posted" sorted set
    posted_score = r.zscore('posted', 'user:' + user) or 0

    # 2. Get the length of the corresponding "<user>:voted_list"
    voted_length = r.scard(user + ':voted_list')

    # 3. Get the total score from the corresponding "<user>:time_spent" sorted set
    time_spent_scores = [score for _, score in r.zrangebyscore(user + ':time_spent', '-inf', 'inf', withscores=True)]
    time_spent_score = sum(time_spent_scores)

    # Compute the activity score using the given weighted sum formula
    activity_score = (posted_score * 0.6) + (voted_length * 0.3) + (time_spent_score * 0.1)

    # Store the computed activity score in the "activity" sorted set
    r.zadd('activity', {'user:' + user: activity_score})
```

Result of the example:

```
Top 10 users based on activity:

1. user:Paula with an activity score of 77
2. user:Antonio with an activity score of 57
3. user:Jack with an activity score of 56
4. user:Adrian with an activity score of 52
5. user:Patricia with an activity score of 51
6. user:Joan with an activity score of 48
7. user:Pol with an activity score of 34
8. user:Roger with an activity score of 34
9. user:Marc with an activity score of 31
10. user:Alex with an activity score of 30
```

- **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 given the case user123 votes (or posts) for www.google.com this action would affect the top 10 list; this is given due to the fact that this increment in the count of a metric which is used to determine a particular user's activity affects in fact this user's activity score. In order to keep always updated the top list, this will be calculated every 24 hours giving a different list each day based on the changes in the database those 24 hours.

- **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 duplicate values?**

In order to deal with the problem of a particular user voting for the same website twice, the data structure that is used to store a particular user's voted_list is the SET; this is done due to the fact that a SET cannot have repeated 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.**

To not add complexity to the recommender we would use the already created ***wishlist*** and ***visited*** data structures. For the target user and for all the other users we would merge both lists into a single one. By doing this we would concentrate on a single list of all the topics that each user finds interesting.

With this merged list for each user, we would now compare them to the target user by finding the common webs and adding them to a set.

```
127.0.0.1:6379> LPUSH adrian:wishlist web:ig web:x web:drive web:linkedin
(integer) 8
127.0.0.1:6379> LPUSH pol:wishlist web:ig web:x web:amazon web:tiktok
(integer) 8
127.0.0.1:6379> LPUSH adrian:visited_list web:uab web:skype web:tiktok web:intel

(integer) 4
127.0.0.1:6379> LPUSH pol:visited_list web:linkedin web:drive web:hp
(integer) 3
127.0.0.1:6379> LPUSH jorge:visited_list web:linkedin web:hp web:intel web:amazon
(integer) 4
127.0.0.1:6379> LPUSH jorge:wishlist web:skype web:google web:ig
(integer) 3
```

```
# Create sets for <user>:wishlist
r.lpush('Pol:wishlist', 'web:yt', 'web:google', 'web:yah', 'web:tv')
r.lpush('Adrian:wishlist', 'web:google', 'web:snap', 'web:yt')
r.lpush('Antonio:wishlist', 'web:acer', 'web:hp', 'web:uab')
r.lpush('Alex:wishlist', 'web:J&J', 'web:google', 'web:apple')
r.lpush('Jack:wishlist', 'web:insta')
r.lpush('Roger:wishlist', 'web:pais', 'web:microsoft', 'web:nike')
r.lpush('Joan:wishlist', 'web:yt', 'web:hp', 'web:apple', 'web:levis')
r.lpush('Xavier:wishlist', 'web:uab', 'web:eastpack', 'web:sony')
r.lpush('Marc:wishlist', 'web:insta', 'web:meta', 'web:J&J')
r.lpush('Paula:wishlist', 'web:yt', 'web:nike', 'web:apple', 'web:gmail', 'web:pytorch')
r.lpush('Lucia:wishlist', 'web:acer', 'web:apple')
r.lpush('Patricia:wishlist', 'web:mit', 'web:vanguardia', 'web:leetcode')
```

```
# Create sorted sets for <user>:visited_list
r.lpush('Pol:visited_list', 'web:insta', 'web:amazon', 'web:hp', 'web:tiktok', 'web:twitter')
r.lpush('Adrian:visited_list', 'web:uab', 'web:insta', 'web:hp' )
r.lpush('Antonio:visited_list', 'web:cv', 'web:linkedin', 'web:gmaps')
r.lpush('Alex:visited_list', 'web:bershka', 'web:zara', 'web:insta')
r.lpush('Jack:visited_list', 'web:samsung', 'web:apple')
r.lpush('Roger:visited_list', 'web:amazon', 'web:elmundo')
r.lpush('Joan:visited_list', 'web:recetas', 'web:stack', 'web:boss')
r.lpush('Xavier:visited_list', 'web:acer', 'web:google', 'web:h&m', 'web:twitter')
r.lpush('Marc:visited_list', 'web:apple', 'web:yt', 'web:playstation')
r.lpush('Paula:visited_list', 'web:acer', 'web:google', 'web:keras', 'web:wikipedia')
r.lpush('Lucia:visited_list', 'web:gmaps', 'web:amazon', 'web:vanguardia')
r.lpush('Patricia:visited_list', 'web:elmundo', 'web:pais', 'web:eastpack', 'web:nike')
```

In this case, we have the ***wishlist and the visited list*** for Pol, Adrián, and Jorge (on redis), and of all the other examples on Python. We will use Jorge as the target user.

```
alumno@ubuntu-vb:~$ redis-cli RPUSH jorge_merge $(redis-cli LRANGE jorge:wishlist 0 -1)(integer) 3
alumno@ubuntu-vb:~$ redis-cli RPUSH jorge_merge $(redis-cli LRANGE jorge:visited_list 0 -1)
(integer) 7
```

```
alumno@ubuntu-vb:~$ redis-cli lrange jorge_merge 0 -1
1) "web:ig"
2) "web:google"
3) "web:skype"
4) "web:amazon"
5) "web:intel"
6) "web:hp"
7) "web:linkedin"
```

```python
# Create sorted sets for target user ⟶ Jorge
r.lpush('Jorge:wishlist', 'web:eastpack', 'web:insta', 'web:yah', 'web:amazon', 'web:apple')
r.lpush('Jorge:visited_list', 'web:acer', 'web:hp', 'web:gmaps', 'web:twitter', 'web:wikipedia')

# Create merged list for Jorge
wishlist = r.lrange('Jorge:wishlist', 0, -1)
visited_list = r.lrange('Jorge:visited_list', 0, -1)

merge_list = [];
merge_list.extend(wishlist); merge_list.extend(visited_list)

for item in merge_list:
    r.rpush('Jorge:merge', item)
```

Now we have the merged list. We will do the same for Pol and Adrián.

```
alumno@ubuntu-vb:~$ redis-cli LRANGE pol_merge 0 -1
1) "web:hp"
2) "web:drive"
3) "web:linkedin"
4) "web:uab"
5) "web:tiktok"
6) "web:amazon"
7) "web:x"
```

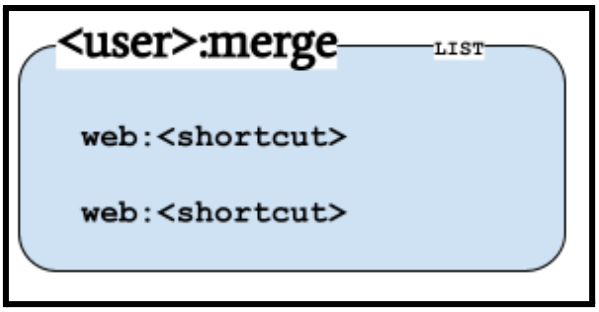
```
alumno@ubuntu-vb:~$ redis-cli LRANGE adrian_merge 0 -1
1) "web:linkedin"
2) "web:drive"
3) "web:x"
4) "web:ig"
5) "web:intel"
6) "web:tiktok"
7) "web:skype"
8) "web:uab"
```

```python
# Merge all the users wishlist with visited_list and compare with jorge
users = ['Pol', 'Adrian', 'Antonio', 'Alex', 'Jack',
         'Roger', 'Joan', 'Xavier', 'Marc', 'Paula',
         'Lucia', 'Patricia']

for user in users:
    merge_list = []
    wishlist = r.lrange(user+':wishlist', 0, -1)
    visited_list = r.lrange(user+':visited_list', 0, -1)

    merge_list.extend(wishlist); merge_list.extend(visited_list)

    for item in merge_list:
        r.rpush(user+':merge', item)
```



The last step will be to compare Jorge with Pol and Jorge with Adrián and obtain a list of the common websites. To do so we need SET and use SINTER. We will also create a copy of the lists into a SET format.

```
alumno@ubuntu-vb:~$ redis-cli SADD jorge_set $(redis-cli LRANGE jorge_merge 0 -1)
(integer) 7
alumno@ubuntu-vb:~$ redis-cli SADD pol_set $(redis-cli LRANGE pol_merge 0 -1)
(integer) 7
alumno@ubuntu-vb:~$ redis-cli SADD adrian_set $(redis-cli LRANGE adrian_merge 0 -1)
(integer) 8
alumno@ubuntu-vb:~$
```

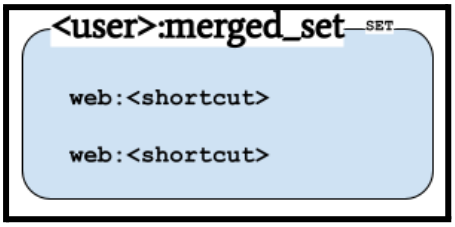
```
alumno@ubuntu-vb:~$ redis-cli SADD jorge_pol $(redis-cli SINTER jorge_set pol_set)
(integer) 3
alumno@ubuntu-vb:~$ redis-cli SADD jorge_adrian $(redis-cli SINTER jorge_set adrian_set)
(integer) 4
```

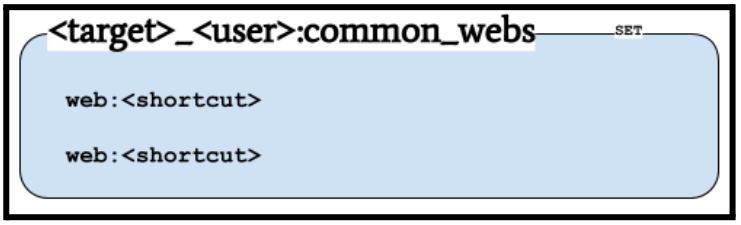
```
alumno@ubuntu-vb:~$ redis-cli SMEMBERS jorge_pol
1) "web:amazon"
2) "web:hp"
3) "web:linkedin"
alumno@ubuntu-vb:~$ redis-cli SMEMBERS jorge_adrian
1) "web:skype"
2) "web:intel"
3) "web:linkedin"
4) "web:ig"
```

```
for item in merge_list:
    r.rpush(user+':merge', item)
    r.sadd(user+':merged_set', item)


matches = r.sinter('Jorge:merged_set', user+':merged_set')

for element in list(matches):
    r.sadd('Jorge_'+user+':common_webs', element)
```





- **How would you find the most similar users from your database if you have a target user?**

To find the most similar user from the database we would look for the highest value on the number of matches of the output sorted set. It is computed with SINTER. Once this is done, we have a sorted list of most similar users. So the person with the highest coincidence on the wishlist and on the already visited list will be more prone to have the same likings.

We are going to find out if Jorge has more similar likes than Pol or Adrián. To do so, we have to retrieve the number of items in the shared sets.

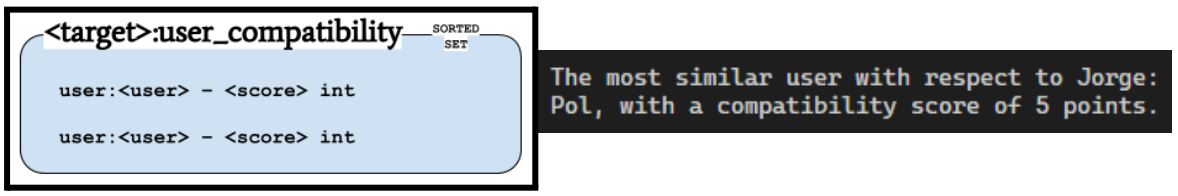
```
alumno@ubuntu-vb:~$ redis-cli SCARD jorge_pol
(integer) 3
alumno@ubuntu-vb:~$ redis-cli SCARD jorge_adrian
(integer) 4
```

In this case, Adrian has one more common web than Pol. This means Adrián is more related to Jorge than Pol is. In Python, the code looks like follows:

```
nMatches = r.scard('Jorge_'+user+':common_webs')
r.zadd('jorge:user_compatibility', {user:nMatches})

top_user = r.zrevrange('jorge:user_compatibility', 0, 0, withscores=True)
```

We will create a data structure that stores the ranking of compatible users so that we don't have to constantly compute this metric. Which outputs:



- **When you have found a similar user, how would you find a new relevant URL to recommend?**

We will again work on the <user>:merged_set. We will compare both the user's and the target's data structures with each other and we will retrieve the URLs that the target hasn't either visited nor added to the wishlist. This way, we will recommend an URL that our target doesn't already know. So we will retrieve it at random from the most similar user. If the target has visited all websites, then we will look for the second compatible user in the list and so on.

```python
# Look for non shared webs on all users w.r.t Jorge
for user in r.zrevrange('jorge:user_compatibility', 0, -1, withscores=True):
    name = user[0].decode('utf-8')
    recommended_webs = list(r.sdiff(user[0].decode('utf-8')+':merged_set', 'Jorge:merged_set'))
    recommended_webs = [web.decode('utf-8') for web in recommended_webs]
    if len(recommended_webs) ≥ 1:
        break
```

```
import random

website = random.choice(recommended_webs).replace('web:', '')

print(f"Website recommended for user is {website}." )
```

```
The most similar user with respect to Jorge:
Pol, with a compatibility score of 5 points.

Website recommended for user is google.
```

We could implement different measures such as recommending by priority the most relevant topics for the target just by storing all the topics of all the websites, for example, but for the sake of simplicity we will leave it this way. Our method is not precise at all, since it chooses a web at random and that could be about a topic our target is not interested in. Therefore, the method based on topics is much better in comparison.

Note: all our Python code is hard-coded. It wouldn't be implemented like this, since there is an abysmal lack of automatization. However, it lets us make an idea about how the code should work. Moreover, a bigger dataset with much more and more realistic examples would help for the demonstration of our ideas.

4. **How would you obtain the most recent/last visited 10 websites for each user? The top 10 must be sorted by time of visit. The last visit must be in the first position on 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?**

In order to keep a list of URLs by time we will implement a sorted set with key-value pairs. This data structure will have the following format.

It will contain the website ID as the key and the seconds elapsed since 2000. These values will be updated when the user enters the URL and when it leaves it.

```
alumno@ubuntu-vb:~$ redis-cli ZADD pol:registry 121743493 web:linkedin 127183489 web:drive 31278457 w
eb:hp
(integer) 3
alumno@ubuntu-vb:~$ redis-cli ZADD adrian:registry 12354565 web:uab 2386595986 web:skype 2389545 web:
tiktok 48942395608 web:intel
(integer) 4
```

- **How will you find the 10 most recent websites?**

To find the most recent websites we will have to sort the set in descending order. This way, the website with the highest time mark will be the last that the user has visited. And the other counters will follow. To retrieve the 10 first, we will use the following functions: `ZREVRANGEBYSCORE <mysortedset> 20 10.`

```
alumno@ubuntu-vb:~$ redis-cli ZREVRANGEBYSCORE pol:registry inf -inf
1) "web:drive"
2) "web:linkedin"
3) "web:hp"
```

```
alumno@ubuntu-vb:~$ redis-cli ZREVRANGEBYSCORE adrian:registry inf -inf
1) "web:intel"
2) "web:skype"
3) "web:uab"
4) "web:tiktok"
```

Right now we have the most recent webs sorted in descending order. So the most recent is at the top. If I only want to show the first 10 websites I would use the following Redis command:

```
ZREVRANGEBYSCORE adrian_registry 0 9 WITHSCORES
```

- **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?**

The script would create the new user's data structures and would add to the registry the time that the user has been on that URL. We would obtain the time when the user entered the website and we would also obtain and update the time once the user has exited the website.