

# Bases de datos NoSQL

## Práctica 3: Redis

Daniel Pérez Efremova

# PARTE 1

## 1. Redis Strings

En el primer caso se escribe el valor. En el segundo, ya existe una clave con el valor “uno”, por tanto, al pasar el parametro nx se devuelve un error.

```
daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22$ redis-cli
127.0.0.1:6379> set uno "soy un valor" nx
OK
127.0.0.1:6379> set uno "soy un valor" nx
(nil)
```

En este caso, al pasar el parámetro xx, se busca si existe la clave “dos”, como no existe, no se crean ni actualiza el valor. En la segunda línea, se escribe el valor. En la tercera, al existir la clave “dos” se actualiza el valor.

```
daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22
127.0.0.1:6379> set dos "soy otro valor :3" xx
(nil)
127.0.0.1:6379> set dos "soy otro valor :3"
OK
127.0.0.1:6379> set dos "soy otro valor distinto :(" xx
OK
127.0.0.1:6379> get dos
"soy otro valor distinto :("
127.0.0.1:6379>
```

## 2. Redis Hashes

En Redis, un hash es un tipo de registro que se estructura como una colección de pares clave valor identificados de manera única. Para crear un hash se utiliza la instrucción HSET. Para obtener todos los valores y campos de uno de los registros se utiliza HGETALL.

```
daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22
127.0.0.1:6379> HSET alumno:1 nombre daniel apellido perez
(integer) 0
127.0.0.1:6379> HSET alumno:2 nombre pedro apellido garcia
(integer) 0
127.0.0.1:6379> HGETALL alumno:1
1) "nombre"
2) "daniel"
3) "apellido"
4) "perez"
```

## 3. Redis Sets

Se crea un SET en el hash anterior para comprobar las funcionalidades pedidas. Se añade el conjunto libros con SADD. Para comprobar si existe un elemento en el conjunto se utiliza la instrucción SISMEMBER.

```
daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22
(error) ERR wrong number of arguments for 'sadd' command
127.0.0.1:6379> SADD alumno:1:libros 123
(integer) 1
127.0.0.1:6379> SADD alumno:1:libros 234
(integer) 1
127.0.0.1:6379> SISMEMBER alumno libros 234
(error) ERR wrong number of arguments for 'sismember' command
127.0.0.1:6379> SISMEMBER alumno:1:libros 1
(integer) 0
127.0.0.1:6379> SISMEMBER alumno:1:libros 123
(integer) 1
127.0.0.1:6379> SISMEMBER alumno:1:libros 234
(integer) 1
```

La instrucción SPOP elimina y muestra uno o más elementos de un SET. Para evitar que se eliminen elementos pero que sí se escoga un elemento aleatoriamente se debe usar la instrucción SRANDMEMBER. Se devuelve 1 si es elemento.

## 4. Redis Sorted Sets

Siguiendo el ejercicio propuesto

```
zrange <setname> 0 -1 withscores
```

Con los parámetros 0 -1 se listan todos los elementos (del primero al último).

## PARTE 2

### 1. Implementar un script en LUA llamado hello.lua que:

- Reciba como KEY una clave redis cuyo valor en base de datos sea un String con un saludo (por ejemplo, 'Hello')
- Reciba como ARGV un String con un nombre
- Extraiga el valor de la KEY y lo concatene con el ARGV que se le pase, dejando un espacio en blanco entre ambas
- Devuelva como resultado la concatenación obtenida

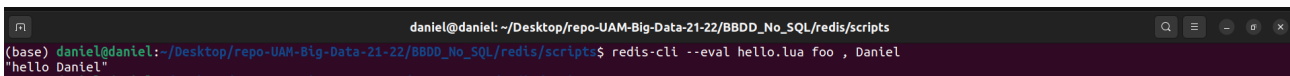
El código propuesto es el siguiente.

*Código 1: hello.lua*

```
return redis.call("get", KEYS[1]).." " ..ARGV[1]
```

Se implementa una llamada al servicio redis mediante redis.call, en concreto, a la instrucción get pasándole como argumento la primera clave que se pase como argumento al script. Se concatena el resultado de la llamada con el primer elemento del argumento vector (ARGV) que se pase al script. El operador de concatenación es '..'.

En la siguiente captura se observa la prueba de funcionamiento.



Se crea previamente la terna clave-valor (foo, Hello) mediante consola de comandos en redis.

### 2. Implementar un script en LUA llamado multiply.lua que:

- Reciba como KEY una clave redis cuyo valor en base de datos sea un Integer con un número (por ejemplo, 3)
- Reciba como ARGV un Integer con un número (por ejemplo, 5)
- Extraiga el valor de la KEY y lo multiplique con el ARGV que se le pase
- Devuelva como resultado de la multiplicación

El código propuesto es el siguiente:

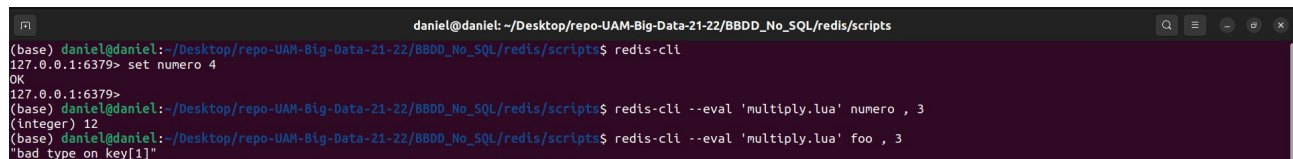
*Código 2: multiply.lua*

```
local first_key = redis.call('get',KEYS[1])
if not tonumber(first_key) then return "bad type on key[1]" end
local second_argv = ARGV[1]
if not tonumber(second_argv) then return "bad type on ARGV[1]" end
return tonumber(first_key) * tonumber(second_argv)
```

Se implementa una llamada al servicio redis mediante redis.call, en concreto, a la instrucción get pasándole como argumentos una clave y un argumento vector.

Se comprueba que cada argumento del script sea un número entero haciendo un cast del tipo de dato a entero. De esta forma el script soporta enteros almacenados como string. En caso de pasar como parámetro un string no numérico el programa finaliza con un mensaje de error.

En la siguiente captura se observa la prueba de funcionamiento.



```
daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ redis-cli
127.0.0.1:6379> set numero 4
OK
127.0.0.1:6379>
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ redis-cli --eval 'multiply.lua' numero , 3
(integer) 12
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ redis-cli --eval 'multiply.lua' foo , 3
"bad type on key[1]"
```

En la prueba se crea la clave numérica (numero1 y numero2) y se llama al script pasando como argumento la clave y un entero. En caso de pasar la clave foo del ejercicio anterior junto al entero, se devuelve el mensaje de error.

## PARTE 3

### 1. Implementar un script en Python llamado hello.py que:

- Reciba como primer parámetro una clave redis cuyo valor en base de datos sea un String con un saludo (por ejemplo, 'Hello')
- Reciba como segundo parámetro un String con un nombre
- Cree una conexión a Redis
- Utilice la conexión a redis para extraer el valor de la clave que se le pasó como primer parámetro
- Concatene el valor obtenido con el segundo parámetro que se le pasó, dejando un espacio en blanco entre ambas
- Imprima como resultado la concatenación obtenida

El código propuesto es el siguiente:

Código 3: hello.py

```
#!/usr/bin/python

import redis
import sys

redis_conn = redis.Redis(
    host='127.0.0.1',
    port=6379)

def say_hello_from_key(key:str, arg: str):
    """
    Descripcion: Saluda al usuario con una clave a una BD Redis
    que contenga un string de saludo y lo concatena con el nombre
    del segundo argumento.
    """
    assert redis_conn.exists(key), 'key[1] not in db'
    key_value = redis_conn.get(key).decode("utf-8")
    result = f'{key_value} {arg}'

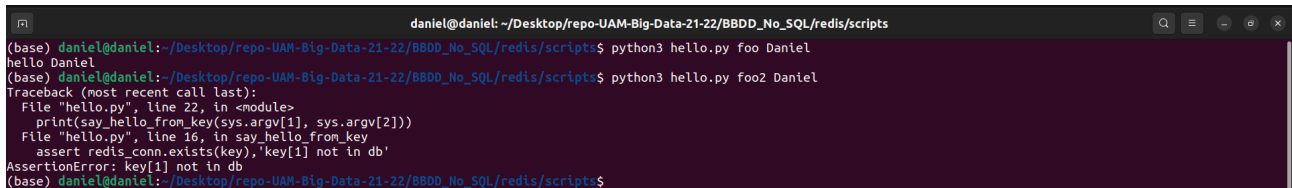
    return result
```

```
print(say_hello_from_key(sys.argv[1], sys.argv[2]))
```

La estrategia es conectar al servicio Redis y mediante la instrucción `get` leer el valor de la clave `foo` (o cualquier otra) que contiene el saludo “hello”. Se lee el segundo argumento que contenga el nombre. Se concatenan con un string en Python.

Si no se encuentra la clave de Redis pasada como primer argumento salta una excepción.

Se adjunta prueba de funcionamiento.



```
daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ python3 hello.py foo Daniel
hello Daniel
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ python3 hello.py foo2 Daniel
Traceback (most recent call last):
  File "hello.py", line 22, in <module>
    print(say_hello_from_key(sys.argv[1], sys.argv[2]))
  File "hello.py", line 16, in say_hello_from_key
    assert redis_conn.exists(key), 'key[1] not in db'
AssertionError: key[1] not in db
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$
```

## 2. Implementar un script en Python llamado `helloLUA.py` que:

- Reciba como primer parámetro una clave redis cuyo valor en base de datos sea un String con un saludo (por ejemplo, ‘Hello’)
- Reciba como segundo parámetro un String con un nombre
- Cree una conexión a Redis
- Abra el fichero `hello.lua` implementado con anterioridad
- Suba el script de LUA a la caché de redis (`register_script`)
- Llame al script cargado utilizando como KEY la clave de Redis recibida como primer parámetro del script de Python y como ARGV el String recibido como segundo parámetro del script de Python.
- Imprima el resultado del script

El código propuesto es el siguiente:

Código 4: `helloLUA.py`

```
#!/usr/bin/python

import redis
import sys

redis_conn = redis.Redis(
    host='127.0.0.1',
    port=6379)

key, arg = sys.argv[1], sys.argv[2]
assert redis_conn.exists(key), 'key[1] not in db'
```

```

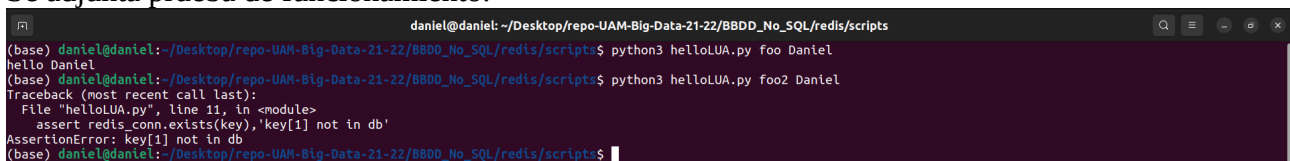
with open ('hello.lua') as f:
    lua_file = f.read()
    say_hello = redis_conn.register_script(lua_file)

result = say_hello(keys=[key], args=[arg], client=redis_conn)
print(result.decode("utf-8"))

```

La estrategia es cargar el artefacto script de Lua mediante `register_script` e interactuar con el desde Python. Se pasan las claves y argumentos al script desde python. Si la clave pasada como argumento al programa Python no existe en el servicio Redis salta una excepción.

Se adjunta prueba de funcionamiento.



```

daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ python3 helloUA.py foo Daniel
hello Daniel
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ python3 helloUA.py foo2 Daniel
Traceback (most recent call last):
  File "helloUA.py", line 11, in <module>
    assert redis_conn.exists(key), 'key[1] not in db'
AssertionError: key[1] not in db
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$

```

### 3. Implementar un script en Python llamado `wordCount.py` que:

- Reciba como primer parámetro el path a un fichero de texto
- Extraiga las palabras del fichero
- Pasa todas las palabras a minúsculas
- Borra todos los símbolos de puntuación que puedas (`\n`, `\r`, `'.'`, etc)
- (Opcional): Para un resultado más limpio borra todas las stopwords de tu texto
- Cree una conexión a Redis
- Cree un sorted set en Redis
- Inserte las palabras en el sorted set creado, de forma que si una palabra ya existe aumente en 1 su score
- Genere el ranking de las 10 palabras con más apariciones en el texto
- Imprima el ranking por pantalla

El código propuesto es:

Código 5: `wordCount.py`

```

#!/usr/bin/python

import redis
import sys
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
nltk.download('stopwords')

```



```

redis_conn = redis.Redis(
    host='127.0.0.1',
    port=6379)

file_path = sys.argv[1]

with open (file_path, 'r') as f:
    text_file = f.read()

tokenizer = RegexpTokenizer(r'\w+')
text_tokens = tokenizer.tokenize(text_file)
text_tokens_without_sw = [word for word in text_tokens if not word in
stopwords.words()]

for token in text_tokens_without_sw:
    redis_conn.zincrby("word_count_sample", 1, token)

sorted_set_ranked = redis_conn.zrevrangebyscore('word_count_sample',
min='-inf', max='+inf', withscores=True)
for token in sorted_set_ranked:
    print(f'{token[0].decode("utf-8")} {int(token[1])}')

redis_conn.delete("word_count_sample")

```

La estrategia consiste en ingestar el fichero de texto mediante Python, usar la libreria nltk para eliminar stopwords y puntuación, y finalmente crear un sorted set en Redis para realizar el recuento. Este recuento es eficiente ya que el sorted set se almacena en Redis ya ordenado. Se elimina el sorted set al final del programa para facilitar el testing.

Se ha empleado el texto de prueba siguiente:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Se adjunta prueba de funcionamiento

```
daniel@daniel: ~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts
(base) daniel@daniel:~/Desktop/repo-UAM-Big-Data-21-22/BBDD_No_SQL/redis/scripts$ python3 wordCount.py sample.txt
[nltk_data] Downloading package stopwords to /home/daniel/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
dolor 2
dolor 2
voluptate 1
veniam 1
velit 1
ullamco 1
tempor 1
sed 1
reprehenderit 1
quis 1
proident 1
pariatur 1
officia 1
occaecat 1
nulla 1
nostrud 1
mollit 1
minim 1
magna 1
laborum 1
laboris 1
labore 1
irure 1
ipsum 1
incididunt 1
id 1
fugiat 1
exercitation 1
elit 1
etusmod 1
deserunt 1
cupidatat 1
culpa 1
consequat 1
consectetur 1
commodo 1
cillum 1
aute 1
anim 1
```