Prática CBD - 1

Initial Configurations

Starting the Redis Server

```
$ redis-server
```

Booting up the Redis Console

```
$ redis-cli
```

Redis Data Types & Commands

String

In Redis, strings are serialized characters in C. Integers are stored in Redis as a string.

SET key string optional nx|xx

Setting a key's string value. Note that, usually, Redis Key's follow the same nomenclature principle as Java Packages (in Java, classes are usually named using the template package.class_name, e.g com.CoolClass, in Redis we usually separate Key names by domain (or any other classification) using :, e.g Books:Action_Books")

`SET Book:1 "Infinite Jest"` **Key**: Book:1 ; **Value**: "Infinite Jest"

GET key

Getting the value of a key

`GET Book:1` **Key**: Book:1 ; **Return**: "Infinite Jest"

INCR key & INCRBY key integer

Increase a string "integer" value by 1 or by the value specified in integer. If the value isn't a "string integer" we get returned **(error) ERR value is not an integer or out of range**

```
 SET    Book:2 1
 INCR   Book:2
 INCRBY Book:2 5
 ```
Return: (integer) 2
Return: (integer) 7

DECR key & DECRBY key integer
Decrease a string "integer" value by 1 or by the value specified in integer. If the valu

`SET   Book:3 5
 DECR   Book:3
 DECRBY Book:3 2`
Return: (integer) 4
Return: (integer) 2

MSET key1 string key2 string (...)
Set multiple key-string value

`MSET Person:1 "DS" Person:2 "Adolfo Dias" Person:3 "Mafalda Meopito"`
Keys: Person:1 ,Person:2 ,Person:3 ; **Values**: "DS", "Adolfo Dias", "Mafalda Meopi

MGET key1 key2 (...)
Get multiple key's values

`MGET Person:1 Person:2 Person:3`
Return: 1)"DS" 2)"Adolfo Dias" 3)"Mafalda Meopito"

List

Lists in Redis are ordered collections of Redis strings that allows for duplicates value

LPUSH key value & RPUSH key value
The **LPUSH** command adds one or more elements to the front of a list, the **RPUSH** co

LPUSH Game:1 "Overwatch"
LPUSH Game:1 "CS:GO"
RPUSH Game:1 "TES: Skyrim" ``` **Key**: Game:1 ; **Value(s)**: "Overwatch", "CS:GO", "TES: Skyrim"

LRANGE key start end

Getting the values of a List from index start to index end

`LRANGE Game:1` **Key**: Game:1 ; **Return**: 1)"CS:GO" 2)"Overwatch" 3)"TES:Skyrim"

LINDEX key index

Getting the value of a list at given index

`LINDEX Game:1 0` **Return**: "CS:GO"

LPOP key & RPOP key

The **LPOP** command remove the first element from the list and returns it the calling client while the **RPOP** command removes and returns the last element of the list.

```
 LPOP    Game:1
RPOP    GAME:1
```

Return: "CS:GO" **Return**: "TES: Skyrim"

Other Commands

LTRIM key start end

Cuts down an existing list from start to end

BLOP key second delay & BROP key second delay

To add in implementing simple queues in Redis using the List data-type, the **BLPOP** and **BRPOP** commands are similar to LPOP and RPOP commands only they will block sending a return back to client if the list is empty. These blocking commands return two values, the key of the list and an element.

Hash

Hash data structures map one or more fields to corresponding value pairs. In Redis, all hash values must be Redis strings with unique field names. Basically a "dictionary" of fields and values

HSET key field value & HMSET key field1 value1 field 2 value2 (...)

Set the value of a single field or set multiple field-values using the **HMSET** command. We can think of the **KEY** as being the dictionary (as in the actual book), the **FIELD** as being the word in the dictionary, and the **VALUE** the word's meaning

```
 HSET Game:Overwatch Type "FPS"
 HSET Game:CS:GO     Type "FPS" Developer "Valve" Price "Free"
 ```
Keys: Game:Overwatch, Game:CS:GO ; **Field-Value**: Type-FPS, Type-FPS, Developer-Va

HGET key field & HMGET key field1 field2 (...) & HGETALL
Get the value of a specified field (or multiple with **HMGET**, or all with **HGETALL**)
```

HGET Game:Overwatch Type HMGET Game:CS:GO Type Developer HGETALL Game:CS:GO ``` **Return**: "FPS" ; "FPS", "Valve" ; "Type - FPS", "Developer - Valve", "Price - Free"

### Other Commands

#### HEXISTS key field

Returns 1 if given field exists or 0 if it doesn't

#### HLEN key

Returns the number of fields in given key

#### HKEYS key

Returns all fields in given key

#### HVALS key

Returns all values in given key

#### HDEL key field

Delete a given field inside the key

#### HINCRBY key field value

Increases the integer value of a hash field's value by a given number.

## Set

A set is a collection of **unordered** string values where **uniqueness of each member is guaranteed** (i.e we can't have two members (or values) of a set be the same) (e.g People:TumTumTum = 'DS', 'Alina', 'João', 'André', 'Tomás', ' Mariana', 'DS' would be a viable list but **NOT A VIABLE SET**). Redis sets also implement union, intersection, difference set semantics and some other dope abilities.

### SADD key member (member2 member3 ...)
```

Adds one or more members to a set.

```
 SADD People:TumTumTum:Members "DS" "Alina" "André" "João" "Tomás"
`
**Key**: People:TumTumTum:Members ; **Set-Members**: "DS", "Alina", "André", "João", "To

### SREM key member (member2 member3 ...)
Removes one or members members to a set.
```

SREM People:TumTumTum:Members "DS" ``` **Key**: People:TumTumTum:Members ;**Set-Members**: "Alina", "André", "João", "Tomás"

SMEMBERS key

Get all members of a given set

```
 SADD     People:TumTumTum:Members "DS"
SMEMBERS People:TumTumTum:Members
SADD     People:TumTumTum:Members  "Mariana"
SMEMBERS People:TumTumTum:Members
SADD     People:TumTumTum:Members  "DS"
SMEMBERS People:TumTumTum:Members
`
**Return**: <br>"DS", "Alina", "Andre", "Joao", "Tomas" <br>"DS", "Alina", "Andre", "Joa

### SUNION key1 key2 (...) & SINTER key1 key2 (...) & SDIFF key1 key2 (...)
Commands used in Set Operations. Accordingly, these produce a set that results from the
```

SADD People:Agoiros:Members "DS" "Mendes" "Batista" "Dias" SUNION People:TumTumTum:Members People:Agoiros:Members SINTER People:TumTumTum:Members People:Agoiros:Members SDIFF People:TumTumTum:Members People:Agoiros:Members SDIFF People:Agoiros:Members People:TumTumTum:Members ``` **Return**:
"DS", "Alina", "Andre", "Joao", "Tomas", "Mariana", "Mendes", "Batista" "Dias"
"DS"
"Alina", "Andre", "Joao", "Tomas", "Mariana"
"Mendes", "Batista", "Dias"

Other Commands

SISMEMBER key member

Returns 1 if given member belongs to given set or 0 if it doesn't

SCARD key

Returns the number of members that belong to given set

Ordered Set

Basically the combination of a Redis List and a Redis Set. Like a List, this data structure is **ordered**, and like a Set, members in this structure are guaranteed to be **unique**. Each member is given a score and the set is sorted according to these values. If two members have identical scores, the members are then sorted lexicographically by value (i.e alphabetically).

ZADD key score member (score2 member2 score3 member3 ...)

Adds one or members to an ordered set. The way these members are then ordered depends on the score they're given (normally, the lower the score the lower it's index)

```
 ZADD Games:Metacritic:Scores 99 "Ocarina of Time" 84 "FTL" 59 "Anthem"
 ```
Key: Games:Metacritic:Scores ; **Set-Members**: "Ocarina of Time", "FTL", "Anthem"

ZREM key member (member2 member3 ...)
Removes one or more members from the Set
```

ZREM Games:Metacritic:Scores 99 "Anthem" ``` **Key**: Games:Metacritic:Scores ;**Set-Members**: "Ocarina of Time", "FTL"

### ZRANGE key start stop [WITHSCORES] & ZREVRANGE key start stop [WITHSCORES]

Get all members of a given set ordered by their index (**ZRANGE**) or from higher to lower score (**ZREVRANGE**). Using the keyword **WITHSCORES** will also display each member's score.
```

```
 ZRANGE    Games:Metacritic:Scores 0 -1
ZRANGE    Games:Metacritic:Scores 0 -1 WITHSCORES
ZREVRANGE Games:Metacritic:Scores 0 -1
```
Return:
"Anthem", "FTL", "Ocarina of Time"
"Anthem", 59, "FTL", 84, "Ocarina

Other Commands
ZRANK key member
Returns a given member's **index** (or NIL if it doesn't belong to the set)
ZSCORE key member
Returns a given member's **score** (or NIL if it doesn't belong to the set)
ZCARD key
Returns the total number of members in the sorted set
ZCOUNT key lowscore highscore
Returns the total number of members in the sorted set that are between the lowscore and
ZRANGEBYSCORE key lowscore highscore
Returns the members that have a score between the given lowscore and highscore. It also

Bit Array or Bit Map
In a bitstring, 8 bits are stored per byte, with the first bit at position 0 being the s

SETBIT key offset value & GETBIT key offset
SETBIT and **GETBIT** commands operate on a single bit offset in a Redis string. Bit

SETBIT Jantar:Pagamentos 0 1 SETBIT Jantar:Pagamentos 2 1 GETBIT Jantar:Pagamentos 0 GETBIT Jantar:Pagamentos 1 GETBIT Jantar:Pagamentos 2 ``` **Key**: Jantar:Pagamentos ;**Returns do GETBIT**: "1", "0", "1"

BITCOUNT key

Returns the amount of bits set to 1

```
 BITCOUNT Jantar:Pagamentos
```
Return: "2"

BITPOS key value
Returns the position (offset) of the first bit set to the given value (0 or 1). If we ha

BITPOS Jantar:Pagamentos 0 BITPOS Jantar:Pagamentos 1 ``` **Return**:
"1"
"0"

BITPOP operation destination_key key1 (key2, key3, ...)

Stores in **destination_key** the result of the Bitwise operation specified applied over the designated keys. Viable operations are: **AND OR XOR NOT**

HyperLogLogs (lol nice name)

They're a special probabilistic data type similar to a Sorted Set. Provides an estimated count of unique items in a collection. **The advantage of HyperLogLogs** is that they're **very efficient in memory** (maximum size is 12K bytes) and **do not require a proportional amount of memory** (memory-to-items in the set) to perform a population count.

PFADD key element (element, element, ...)

Adds one or more elements to an HyperLogLog.

```
 PFADD Conference:Atendees People:TumTumTum:Members People:Agoiros:Members
`
**Key**: Conference:Atendees ;

### PFCOUNT key
Returns an approximation of the count in the HyperLogLogs with an standard error of .81%
```

PFCOUNT Conference:Atendees ``` **Return**: 4

PFMERGE destination_key key1 (key2 key3, ...)

Merge multiple HyperLogLogs into a single one specified by destination_key.

```
 SADD    People:Staff:Members  "Chico Silva" "André Baião" "Rafael Simões"
PFADD   Conference:Staff People:Staff:Members
PFMERGE Conference:Everyone Conference:Staff Conference:Atendees
PFCOUNT Conference:Everyone
`
**Return**: <br>"3"<br>

# Mass Insertion
Redis has a very kewl function that allows you to do mass insertion of data read from a
```

sudo cat initials4redis.txt | redis-cli --pipe

```

You might wanna treat data before feeding it, for example using a small python (or whate
```

name_counter = {} #Dictionary that stores, as keys, the initial letters of each name and as values the total amount of names that start with that letter

with open("female-names.txt","r") as reader: #Add data to name_counter for line in reader: if line[0].upper() not in name_counter: name_counter[line[0].upper()] = 0

```
    name_counter[line[0].upper()] += 1
```

with open("initials4redis.txt","w") as writer: for key, value in name_counter.items(): writer.write(f" SET {key} {value}\n") ```

More info can be found over at https://redis.io/topics/mass-insert

Programatic Access to Redis

Support for using Redis with (virtually) any programming language can be found over at https://redis.io/clients Personally, I used https://pypi.org/project/redis/ Documentation over at: https://redis-py.readthedocs.io/en/latest/index.html#redis.Redis.hgetall

References

http://intro2libsys.info/introduction-to-redis/redis-data-types - Redis Data Types

Author

Diogo Silva