Parte 0: Conexión

Para conectarme a redis lo hice mediante Docker utilizando la guia "Docker Quickstart" dentro de Redis Enterprise Software. O sea, una instalación local pero mediante Docker.

Parte 1: Introducción

1) ¿Qué tipo de bd es Redis?

Redis es una base de datos NoSQL de tipo clave-valor distribuida en memoria. La durabilidad de los datos es opcional y soporta varias estructuras de datos y transacciones.

2) ¿Dónde almacena los datos Redis?

Almacena los datos en la memoria RAM. Aunque se puede configurar para que vuelque la información en un archivo.

3) ¿Qué tipos de datos posee Redis?

String: secuencias de un tamaño máximo de 512 MB
Hash: entrada con varios campos
List: conjunto de strings clasificados por orden de introducción
Set: conjunto de strings sin clasificar
Sorted set (sets clasificado): conjunto de strings clasificado por el usuario
Bitmap: conjunto de operaciones a nivel de bit
HyperLogLog: estimación según valores unívocos
Stream: lista de strings o pares complejos de key-value

4) Enuncie las características de Redis

- Es muy rápido ya que los datos estan todos en memoria principal y no los tiene que buscar en ningun disco
- Se pueden guardar claves de distintos tipos y con un tamaño de 512MB para una cadena
- Permite replicación y persistencia bajo la arquitectura maestro-esclavo para mejorar las lecturas y recuperación
- Es compatible con casi cualquier lenguaje (Java, Python, PHP, C, C++, C#, Javascript, etc)

5) Comparar Redis con los RDBMS

Redis es una base de datos clave-valor, la información se guarda en pares de claves y valores mientras que las RDBMS tiene una estructura de tablas con relaciones entre ellas.

A pesar que lo permite, redis no es persistente ya que los datos están en memoria principal mientras que las RDBMS persisten todo al estar en memoria secundaria**.** Eso permite que las operaciones de lectura y escritura sean extremadamente rápidas mientras que en las RDBMS son más lentas. Pero en consecuencia, el espacio de redis es bastante menor y más costoso por lo que se suelen guardar

información de texto que tiene que ser accedida de manera inmediata mientras que en las RDBMS el espacio es mayor y se pueden guardar datos más tamaño.

6) **¿Redis tiene transacciones?**

Si, permite que la ejecución de una serie de comandos se realicen en un solo paso utilizando los comandos MULTI, EXEC, DISCARD y WATCH.

7) **¿Redis tiene persistencia?**

Si tiene persistencia, tiene un modo de guardar las entradas en un archivo appendonly

8) **¿Cuáles son los principales usos de Redis?**

Redis se suele utilizar para Caching, analíticas en tiempo real, guardar y controlar información de sesiones, se usa como índice secundario, entre otras.

Parte 2 - Manejo simple de valores String

1) **Agregue una clave package con el valor "Bariloche 3 days"**

Append agrega el valor al final si la key ya existe, sino agrega la key con el valor. Podría haber usado SET: si la key no existe, set y append se comportan igual

```
redis-12000.cluster.local:12000> APPEND package "Bariloche 3 days"
(integer) 16
redis-12000.cluster.local:12000> get package
"Bariloche 3 days"
```

2) **Agregue una clave user con el valor "Turismo BD2". Obtenga el valor de la clave user**

```
redis-12000.cluster.local:12000> set user "Turismo BD2"
OK
redis-12000.cluster.local:12000> get user
"Turismo BD2"
```

3) **Obtenga todos los valores de claves almacenadas**

No tengo un comando que me muestra todos los pares clave-valor almacenados. En este caso recuerdo las claves, pero lo que puede hacerse si uno no las recuerda es ver todas las claves almacenadas con el comando KEYS con el parametro "*", porque el comando KEYS sirve para buscar claves que cumplan con el pattern ingresado como parametro; y luego buscar todos los valores con el comando MGET y el valor de las claves. El comando MGET muestra el valor asociado a todas las claves ingresadas.

No es recomendable el comando KEYS en databases grandes porque es poco

performático, se recomienda el comando SCAN o los sets para esto.

```
redis-12000.cluster.local:12000> keys *
1) "user"
2) "key1"
3) "package"
redis-12000.cluster.local:12000> MGET user key1 package
1) "Turismo BD2"
2) "123"
3) "Bariloche 3 days"
```

4) **Agregue una clave user con el valor “Cronos Turismo” ¿Cuál es el valor actual de la clave user?**
   Como lo agregue con SET, el valor se sobreescribe, por lo que va a ser “Cronos Turismo”

```
redis-12000.cluster.local:12000> set user "Cronos Turismo"
OK
redis-12000.cluster.local:12000> get user
"Cronos Turismo"
```

5) **Concatene “ S.A.” a la clave user. ¿Cuál es el valor actual de la clave user?**
   El comando APPEND agrega el valor ingresado al final del valor que la clave ya tenía.

```
redis-12000.cluster.local:12000> append user " S.A"
(integer) 18
redis-12000.cluster.local:12000> get user
"Cronos Turismo S.A"
```

6) **Elimine la clave user.**
   DEL user
7) **¿Qué valor retorna si queremos obtener la clave user?**
   Devuelve nil

Parte 3 - Manejo simple de valores Numericos

1) **Verificar si existe la clave visits**

```
127.0.0.1:6379> EXISTS visits
(integer) 0
```

2) **Agregue una clave visits con valor 0**

```
127.0.0.1:6379> SET visits 0
OK
```

3) **Incremente en 1 visits ¿Cuál es el valor actual de la clave visits?**

```
127.0.0.1:6379> INCR visits
(integer) 1
127.0.0.1:6379> GET visits
"1"
```

4) **Incremente en 5 visits. ¿Cuál es el valor actual de la clave visits?**

```
127.0.0.1:6379> INCRBY visits 5
(integer) 6
127.0.0.1:6379> GET visits
"6"
```

5) **Decremente en 1 visits ¿Cuál es el valor actual de la clave visits?**

```
127.0.0.1:6379> DECR visits
(integer) 5
127.0.0.1:6379> GET visits
"5"
```

6) **Incremente en 2 visits. ¿Cuál es el valor actual de la clave visits?**

```
127.0.0.1:6379> INCRBY visits 2
(integer) 7
127.0.0.1:6379> GET visits
"7"
```

7) **Agregue una clave “value package” con el valor 539789.32**

```
127.0.0.1:6379> SET "value package" 539789.32
OK
127.0.0.1:6379> GET "value package"
"539789.32"
```

8) **Incremente en 20000 la clave “value package”. ¿Cuál es el valor actual de “value package”?**

```
127.0.0.1:6379> INCRBY "value package" 20000
(error) ERR value is not an integer or out of range
127.0.0.1:6379> INCRBYFLOAT "value package" 20000
"559789.31999999999999318"
127.0.0.1:6379> GET "value package"
"559789.31999999999999318"
```

9) **¿Cuál es el tipo de datos de “value package”, visits y user?**

Como user fue eliminada devuelve ***none*** pero si estuviese creada sería ***string***

```
127.0.0.1:6379> TYPE user
none
127.0.0.1:6379> TYPE visits
string
127.0.0.1:6379> TYPE "value package"
string
```

Parte 4 - Manejo de claves

1. **Obtenga todas las claves que empiecen con v**

```
redis-12000.cluster.local:12000> keys v*
1) "visits"
2) "value package"
```

2. **Obtenga todas las claves que contengan la “t”**

```
redis-12000.cluster.local:12000> keys *t*
1) "visits"
```

3. **Obtenga todas las claves que terminen con “age”**

```
redis-12000.cluster.local:12000> keys *age
1) "value package"
2) "package"
```

4. **Renombre la clave “package” por “bariloche package”**

```
redis-12000.cluster.local:12000> RENAME package "bariloche package"
OK
redis-12000.cluster.local:12000> keys *age
1) "bariloche package"
2) "value package"
```

5. **Si quisiera evitar renombrar una clave por otra existente, ¿qué comando utilizaría?**
   Si la clave por la que estoy cambiando ya existe, RENAME elimina la clave y realiza el cambio de nombre. Puedo usar RENAMENX, que en lugar de eliminarla, no realiza el RENAME si la newkey ya existe.
6. **Elimine todas las claves**

```
redis-12000.cluster.local:12000> keys *
1) "visits"
2) "bariloche package"
3) "key1"
4) "value package"
redis-12000.cluster.local:12000> flushdb
OK
redis-12000.cluster.local:12000> keys *
(empty array)
```

Parte 5: Expiración de Claves

1) **Agregue una clave agency con el valor “Cronos Tours”**

```
127.0.0.1:6379> SET agency "Cronos Tour"
OK
127.0.0.1:6379> GET agency
"Cronos Tour"
```

2) **¿Cúal es el tiempo de vida de la clave agency?**

Como no tiene un tiempo de vida asociado devuelve -1

```
127.0.0.1:6379> TTL agency
(integer) -1
```

3) **Agregue una expiración de 30 segundos a la clave agency**

```
127.0.0.1:6379> EXPIRE agency 30
(integer) 1
```

4) **¿Cúal es el tiempo de vida de la clave agency?**

```
127.0.0.1:6379> TTL agency
(integer) 28
127.0.0.1:6379> TTL agency
(integer) 26
127.0.0.1:6379> TTL agency
(integer) 25
```

5) **Pasados los 30 segundos, ¿Cúal es el tiempo de vida de la clave agency? ¿Qué retorna si pido el valor de agency?**

```
127.0.0.1:6379> TTL agency
(integer) -2
```

Retorna un -2 ya que la clave agency no existe

```
127.0.0.1:6379> GET agency
(nil)
```

6) **Agreguemos una clave agency con el valor “Cronos Tours” que expire en 20 segundos**

(Me equivoque en el primer TTL el nombre de la key)

```
127.0.0.1:6379> SET agency "Cronos Tour" EX 20
OK
127.0.0.1:6379> TTL agnency
(integer) -2
127.0.0.1:6379> TTL agency
(integer) 10
127.0.0.1:6379> TTL agency
(integer) 6
127.0.0.1:6379> TTL agency
(integer) 4
127.0.0.1:6379> TTL agency
(integer) -2
```

Parte 6 - Listas

1. **Inserte una lista llamada pets con el valor dog.**

```
redis-12000.cluster.local:12000> LPUSH pets dog
(integer) 1
```

2. **¿Qué sucede si ejecuto el comando get pets? ¿Cómo obtengo los valores de la lista?**
   Si ejecuto el comando GET, Redis me devuelve un error diciendo que ese comando no puede operar con el valor que tiene esa key. LRANGE me devuelve los valores de la key que señalo, en el rango que marco: en este caso, desde 0, que es la primera posición, a -1, que es la última.

```
redis-12000.cluster.local:12000> get pets
(error) WRONGTYPE Operation against a key holding the wrong kind of value
redis-12000.cluster.local:12000> LRANGE pets 0 -1
1) "dog"
```

3. **Agregue a la lista pets el valor cat a la izquierda.**
   LPUSH agrega “a la izquierda”: es decir, agrega al principio, “prepends”, antepone.
4. **Agregue a la lista pets el valor fish a la derecha.**

```
redis-12000.cluster.local:12000> LPUSH pets cat
(integer) 2
redis-12000.cluster.local:12000> RPUSH pets fish
(integer) 3
redis-12000.cluster.local:12000> LRANGE pets 0 -1
1) "cat"
2) "dog"
3) "fish"
```

5. **¿Qué tipo de datos es el valor de pets?**

```
redis-12000.cluster.local:12000> TYPE pets
list
```

6. **Elimine el valor a la izquierda de la lista.**

```
redis-12000.cluster.local:12000> LPOP pets
"cat"
```

7. **Elimine el valor a la fecha de la lista.**

```
redis-12000.cluster.local:12000> RPOP pets
"fish"
redis-12000.cluster.local:12000> LRANGE pets 0 -1
1) "dog"
```

8. **Agregue una clave “vuelo:ar389” los valores: aep, mdz, brc, nqn y mdq.**
9. **Ordene los valores de lista “vuelo:ar389”. ¿Qué sucede si solicito todos los valores de la lista?**
   El comando SORT ordena los elementos de la lista por el criterio especificado, pero si no le indico otra cosa, solo los proyecta, dejando la lista de igual manera a la que estaba. Con el parametro STORE puedo indicarle donde guardar la lista ordenada, en este alfabeticamente. Si luego de STORE pongo la key de la misma lista, la

sobreescribe.

```
redis-12000.cluster.local:12000> SORT vuelo:ar389 ALPHA
1) "aep"
2) "brc"
3) "mdq"
4) "mdz"
5) "nqn"
redis-12000.cluster.local:12000> LRANGE pets 0 -1
1) "dog"
redis-12000.cluster.local:12000> LRANGE vuelo:ar389 0 -1
1) "aep"
2) "mdz"
3) "brc"
4) "nqn"
5) "mdq"
```

10. **Inserte el valor “fte” luego de “brc”**

```
redis-12000.cluster.local:12000> LINSERT vuelo:ar389 AFTER "brc" "fte"
(integer) 6
redis-12000.cluster.local:12000> LRANGE vuelo:ar389 0 -1
1) "aep"
2) "brc"
3) "fte"
4) "mdq"
5) "mdz"
6) "nqn"
```

11. **Inserte el valor “ush” antes de “fte”**

```
redis-12000.cluster.local:12000> LINSERT vuelo:ar389 BEFORE "fte" "ush"
(integer) 7
redis-12000.cluster.local:12000> LRANGE vuelo:ar389 0 -1
1) "aep"
2) "brc"
3) "ush"
4) "fte"
5) "mdq"
6) "mdz"
7) "nqn"
```

12. **Modifique el último elemento por “sla”**

```
redis-12000.cluster.local:12000> LSET vuelo:ar389 -1 "sla"
OK
redis-12000.cluster.local:12000> LRANGE vuelo:ar389 0 -1
1) "aep"
2) "brc"
3) "ush"
4) "fte"
5) "mdq"
6) "mdz"
7) "sla"
```

13. **Obtenga la cantidad de elementos de “vuelo:ar389”**

```
redis-12000.cluster.local:12000> LLEN vuelo:ar389
(integer) 7
```

14. **Obtenga el 3 valor de “vuelo:ar389”**

Como la numeración de las listas comienza en 0, el tercer valor es el ubicado en el índice 2 de la lista.

```
redis-12000.cluster.local:12000> LINDEX vuelo:ar389 2
"ush"
```

15. **Elimine el valor “aep” de “vuelo:ar389”**

```
redis-12000.cluster.local:12000> LREM vuelo:ar389 0 "aep"
(integer) 1
redis-12000.cluster.local:12000> LRANGE vuelo:ar389 0 -1
1) "brc"
2) "ush"
3) "fte"
4) "mdq"
5) "mdz"
6) "sla"
```

16. **Quédese con los valores de las posiciones 3 a 5 de vuelo:ar389**

```
redis-12000.cluster.local:12000> LRANGE vuelo:ar389 3 5
1) "mdq"
2) "mdz"
3) "sla"
```

17. **Agregue en “vuelo:ar389” el valor “fte”. ¿Cuántas veces aparece?**

Se agrega normalmente y aparece dos veces.

```
redis-12000.cluster.local:12000> RPUSH vuelo:ar389 "fte"
(integer) 7
redis-12000.cluster.local:12000> LRANGE vuelo:ar389 0 -1
1) "brc"
2) "ush"
3) "fte"
4) "mdq"
5) "mdz"
6) "sla"
7) "fte"
```

Parte 7 - Conjuntos

1. **Agregue un conjunto llamado airports los valores: eze aep nqn mdz mdq ush fte sla aep nqn brc cpc juj aep tuc eqs**

```
127.0.0.1:6379> SADD airports eze aep nqn mdz msq ush fte sla aep nqn brc cpc juj aep tuc eqs
(integer) 13
```

2. ¿Cuántos valores tiene el conjunto?

La entrada es de 16 valores pero el conjunto se termina armando con 13 ya que hay algunos repetidos

```
127.0.0.1:6379> SCARD airports
(integer) 13
```

3. Liste los valores del conjunto airports.

```
127.0.0.1:6379> SMEMBERS airports
 1) "eze"
 2) "aep"
 3) "nqn"
 4) "mdz"
 5) "msq"
 6) "ush"
 7) "fte"
 8) "sla"
 9) "brc"
10) "cpc"
11) "juj"
12) "tuc"
13) "eqs"
```

4. Quite el valor cpc del conjunto airports.

```
127.0.0.1:6379> SREM airports cpc
(integer) 1
127.0.0.1:6379> SMEMBERS airport
(empty array)
127.0.0.1:6379> SMEMBERS airports
 1) "eze"
 2) "aep"
 3) "nqn"
 4) "mdz"
 5) "msq"
 6) "ush"
 7) "fte"
 8) "sla"
 9) "brc"
10) "juj"
11) "tuc"
12) "eqs"
```

5. Quite un valor aleatorio del conjunto airports.

```
127.0.0.1:6379> SPOP airports 1
1) "msq"
127.0.0.1:6379> SMEMBERS airports
 1) "eze"
 2) "aep"
 3) "nqn"
 4) "mdz"
 5) "ush"
 6) "fte"
 7) "sla"
 8) "brc"
 9) "juj"
10) "tuc"
11) "eqs"
```

6. ¿Qué cantidad de valores tiene ahora airports?

Ahora tiene 11 elementos

```
127.0.0.1:6379> SCARD airports
(integer) 11
```

7. Compruebe si cpc es miembro del conjunto airports.

```
127.0.0.1:6379> SISMEMBER airports cpc
(integer) 0
```

Devuelve 0 ya que el elemento no está en el conjunto

8. Mueva los valores sla y juj a un conjunto denominado noa_airports.

```
127.0.0.1:6379> SMOVE airports noa_airports sla
(integer) 1
127.0.0.1:6379> SMOVE airports noa_airports juj
(integer) 1
127.0.0.1:6379> SMEMBERS airports
1) "eze"
2) "aep"
3) "nqn"
4) "mdz"
5) "ush"
6) "fte"
7) "brc"
8) "tuc"
9) "eqs"
127.0.0.1:6379> SMEMBERS noa_airports
1) "sla"
2) "juj"
```

9. Retorne la unión de los conjuntos airports y noa_airports. ¿Modifica los conjuntos base?

```
127.0.0.1:6379> SUNION airports noa_airports
 1) "eze"
 2) "aep"
 3) "nqn"
 4) "mdz"
 5) "ush"
 6) "fte"
 7) "brc"
 8) "tuc"
 9) "eqs"
10) "sla"
11) "juj"
127.0.0.1:6379> SMEMBERS airports
1) "eze"
2) "aep"
3) "nqn"
4) "mdz"
5) "ush"
6) "fte"
7) "brc"
8) "tuc"
9) "eqs"
127.0.0.1:6379> SMEMBERS noa_airports
1) "sla"
2) "juj"
```

No modifica los conjuntos base

10. Realice la unión de los conjuntos airports y noa_airports en un conjunto llamado total_airports.

```
127.0.0.1:6379> SUNIONSTORE total_airports airports noa_airports
(integer) 11
127.0.0.1:6379> SMEMBERS total_airports
 1) "eze"
 2) "aep"
 3) "nqn"
 4) "mdz"
 5) "ush"
 6) "fte"
 7) "brc"
 8) "tuc"
 9) "eqs"
10) "sla"
11) "juj"
```

11. Realice la intersección entre los conjuntos total_airports y noa_airports.

```
127.0.0.1:6379> SINTER total_airports noa_airports
1) "sla"
2) "juj"
```

12. Realice la diferencia entre los conjuntos total_airports y noa_airports.

```
127.0.0.1:6379> SDIFF total_airports noa_airports
1) "eze"
2) "aep"
3) "nqn"
4) "mdz"
5) "ush"
6) "fte"
7) "brc"
8) "tuc"
9) "eqs"
```

Parte 8 - Conjuntos ordenados

1. **En un conjunto ordenado llamado passengers agregue los siguientes datos: 2.5 federico 4 alejandra 3 julian 1 ivan 2 andrea 2 luciana 2.4 natalia**

```
redis-12000.cluster.local:12000> ZADD passengers 2.5 "federico" 4 "alejandra" 3
"julian" 1 "ivan" 2 "andrea" 2 "luciana" 2.4 "natalia"
(integer) 7
```

2. **Obtenga los valores del conjunto passengers**

```
redis-12000.cluster.local:12000> ZRANGE passengers 0 -1
1) "ivan"
2) "andrea"
3) "luciana"
4) "natalia"
5) "federico"
6) "julian"
7) "alejandra"
```

Puedo agregar el parametro WITHSCORES, en cuyo caso también me muestra los scores.

3. **Actualice el score de luciana a 2.7**

```
redis-12000.cluster.local:12000> ZADD passengers XX 2.7 luciana
(integer) 0
redis-12000.cluster.local:12000> ZRANGE passengers 0 -1 WITHSCORES
 1) "ivan"
 2) "1"
 3) "andrea"
 4) "2"
 5) "natalia"
 6) "2.4"
 7) "federico"
 8) "2.5"
 9) "luciana"
10) "2.7"
11) "julian"
12) "3"
13) "alejandra"
14) "4"
```

El parametro XX indica que solo actualice si el valor ya existe, no agrega elementos nuevos. En este caso es innecesario

4. **Agregue al conjunto passengers a silvia con score 5.1**

```
redis-12000.cluster.local:12000> ZADD passengers 5.1 "silvia"
(integer) 1
```

5. **Incremente en 2 el score de alejandra en el conjunto passengers.**
INCRBY incrementa el score existente en la cantidad señalada, y devuelve el valor actualizado del score

```
redis-12000.cluster.local:12000> ZINCRBY passengers 2 "alejandra"
"6"
```

6. **Obtenga los valores del conjunto passengers con sus scores.**

```
redis-12000.cluster.local:12000> ZRANGE passengers 0 -1 WITHSCORES
 1) "ivan"
 2) "1"
 3) "andrea"
 4) "2"
 5) "natalia"
 6) "2.4"
 7) "federico"
 8) "2.5"
 9) "luciana"
10) "2.7"
11) "julian"
12) "3"
13) "silvia"
14) "5.1"
15) "alejandra"
16) "6"
```

7. **Obtenga los valores del conjunto passengers con sus scores en orden inverso.**

```
redis-12000.cluster.local:12000> ZRANGE passengers 0 -1 REV WITHSCORES
 1) "alejandra"
 2) "6"
 3) "silvia"
 4) "5.1"
 5) "julian"
 6) "3"
 7) "luciana"
 8) "2.7"
 9) "federico"
10) "2.5"
11) "natalia"
12) "2.4"
13) "andrea"
14) "2"
15) "ivan"
16) "1"
```

8. **Obtenga la cantidad de elementos del conjunto passengers.**

```
redis-12000.cluster.local:12000> ZCARD passengers
(integer) 8
```

9. **Obtenga la cantidad de elementos que tienen scores entre 2 y 3.**

```
redis-12000.cluster.local:12000> ZCOUNT passengers 2 3
(integer) 5
```

10. **Obtenga el ranking de julian en el conjunto passengers.**

```
redis-12000.cluster.local:12000> ZRANK passengers "julian"
(integer) 5
```

De devuelve la posición pero teniendo en cuenta que el ranking comienza en 0. Puedo agregarle el parametro WITHSCORE

11. **Obtenga el score de andrea en el conjunto passengers.**

```
redis-12000.cluster.local:12000> ZSCORE passengers "andrea"
"2"
```

12. **Extraiga el valor de menor score del conjunto passengers.**

```
redis-12000.cluster.local:12000> ZPOPMIN passengers
1) "ivan"
2) "1"
```

13. **Extraiga el valor de mayor score del conjunto passengers.**

```
redis-12000.cluster.local:12000> ZPOPMAX passengers
1) "alejandra"
2) "6"
```

14. **Elimine del conjunto passengers al valor silvia.**

```
redis-12000.cluster.local:12000> ZREM passengers "silvia"
(integer) 1
redis-12000.cluster.local:12000> ZRANGE passengers 0 -1
1) "andrea"
2) "natalia"
3) "federico"
4) "luciana"
5) "julian"
```

Parte 9 - Hashes

1. **Agregue a un hash llamado user:cronos los valores: “razon social” “cronos s.a”, domicilio “47 236 La Plata”, “teléfono” 2215556677**

```
127.0.0.1:6379> HSET user:cronos "razon social" "cronos s.a" domicilio "47 236 La Plata" telefono 2215556677
(integer) 3
```

2. **Agregue el mail info@cronos.com.ar a user:cronos**

```
127.0.0.1:6379> HSET user:cronos mail info@cronos.com.ar
(integer) 1
```

3. **Obtenga todos los valores de user:cronos**

```
127.0.0.1:6379> HGETALL user:cronos
1) "razon social"
2) "cronos s.a"
3) "domicilio"
4) "47 236 La Plata"
5) "telefono"
6) "2215556677"
7) "mail"
8) "info@cronos.com.ar"
```

4. **Obtenga el mail de user:cronos**

```
127.0.0.1:6379> HGET user:cronos mail
"info@cronos.com.ar"
```

5. **Elimine el teléfono de user:cronos**

```
127.0.0.1:6379> HDEL user:cronos telefono
(integer) 1
127.0.0.1:6379> HGETALL user:cronos
1) "razon social"
2) "cronos s.a"
3) "domicilio"
4) "47 236 La Plata"
5) "mail"
6) "info@cronos.com.ar"
```

6. **Obtenga la cantidad de campos de user:cronos**

```
127.0.0.1:6379> HLEN user:cronos
(integer) 3
```

7. **Obtenga las claves de los campos de user:cronos**

```
127.0.0.1:6379> HKEYS user:cronos
1) "razon social"
2) "domicilio"
3) "mail"
```

8. **Determine si existe el campo cuil en user:cronos**

```
127.0.0.1:6379> HEXISTS user:cronos cuil
(integer) 0
```

Devuelve 0 ya que el campo “cuil” no existe

9. **Obtenga todos los valores de los campos de user:cronos**

```
127.0.0.1:6379> HVALS user:cronos
1) "cronos s.a"
2) "47 236 La Plata"
3) "info@cronos.com.ar"
```

10. **Obtenga la longitud del campo mail de user:cronos**

```
127.0.0.1:6379> HSTRLEN user:cronos mail
(integer) 18
```

Parte 10 - GeoSpatial

1. **Agregue en un conjunto denominado cities las siguientes localidades:**
   Luego del comando y el nombre del conjunto se ingresan los miembros, con el orden de los parámetros siendo: longitud, latitud, nombre

```
redis-12000.cluster.local:12000> GEOADD cities -58.37723 -34.61315 "Buenos Aires" -64.18105 -31.4135 "Cord
oba" -60.63932 -32.94682 "Rosario" -68.82717 -32.89084 "Mendoza" -65.2226 -26.82414 "San Miguel de Tucuman
" -57.95453 -34.92145 "La Plata" -57.5562 -38.00042 "Mar del Plata" -65.41166 -24.7859 "Salta" -60.70868 -
31.64881 "Santa Fe" -68.53639 -31.5375 "San Juan" -58.98389 -27.46056 "Resistencia" -64.26149 -27.79511 "S
antiago del Estero" -55.89608 -27.36708 "Posadas" -65.29712 -24.19457 "San Salvador de Jujuy" -62.27243 -3
8.71959 "Bahia Blanca" -60.52897 -31.73271 "Parana"
```

2. **Obtenga los valores del conjunto cities**

   Puedo usar los mismo comando que en los sorted set porque la estructura del índice Geo es un conjunto ordenado

```
redis-12000.cluster.local:12000> ZRANGE cities 0 -1
 1) "Mendoza"
 2) "San Juan"
 3) "Cordoba"
 4) "San Miguel de Tucuman"
 5) "Santiago del Estero"
 6) "Salta"
 7) "San Salvador de Jujuy"
 8) "Bahia Blanca"
 9) "Mar del Plata"
10) "Buenos Aires"
11) "La Plata"
12) "Rosario"
13) "Santa Fe"
14) "Parana"
15) "Resistencia"
16) "Posadas"
```

3. **Obtenga las coordenadas de Santa Fe**

```
redis-12000.cluster.local:12000> GEOPOS cities "Santa Fe"
1) 1) "-60.70868164300918579"
   2) "-31.64881101691540977"
```

4. **Obtenga la distancia en km entre Buenos Aires y Cordoba**

   Tengo que indicar por parámetro en que unidad (M, KM, pies(FT), millas (ML)) quiero la respuesta. En este caso kilómetros.

```
redis-12000.cluster.local:12000> GEODIST cities "Buenos Aires" "Cordoba" KM
"647.6303"
```

5. **Obtenga las ciudades que están en un radio de 100 km de la coordenada -27.37 -55.9 con su distancia.**

   Indico primero la key del conjunto, la longitud y latitud desde la que quier buscar, el valor del radio en que quiero buscar, en este caso 100, la unidad del radio, KM, y puedo marcar por parametro que me devuelva la distancia a la ubicación mandada (WITHDIST) y las coordenadas de la ciudad que encontro (WITHCOORD)

```
redis-12000.cluster.local:12000> GEORADIUS cities -55.9 -27.37 100 KM WITHDIST
1) 1) "Posadas"
   2) "0.5055"
```

6. **Obtenga las ciudades que están a menos de 700 km de Córdoba.**

En lugar de agregar la coordenada, nombro uno de los miembros del set.

```
redis-12000.cluster.local:12000> GEORADIUSBYMEMBER cities "Cordoba" 700 KM WITHDIST
 1) 1) "Cordoba"
    2) "0.0000"
 2) 1) "San Miguel de Tucuman"
    2) "520.3835"
 3) 1) "Santiago del Estero"
    2) "402.5353"
 4) 1) "Parana"
    2) "347.8771"
 5) 1) "Santa Fe"
    2) "330.2202"
 6) 1) "Resistencia"
    2) "668.2173"
 7) 1) "Buenos Aires"
    2) "647.6303"
 8) 1) "La Plata"
    2) "698.5502"
 9) 1) "Rosario"
    2) "374.4698"
10) 1) "San Juan"
    2) "413.3545"
11) 1) "Mendoza"
    2) "467.3004"
```