Sistemas de Big data

Redis

Conectar Redis con Jupiter

```
Import redis
r = redis.Redis(
        host=”Redis”,
        port=6379,
        db=0,
        decode_responses=True
)
```

Redis es una base de datos clave-valor

Clave = Siempre es una cadena

Valor = Puede ser de diferentes tipos, depende de la información que queramos almacenar

Comandos (Recordemos todo es clave-valor)

Los ejemplos van a estar siempre en modo cuadernos jupiter

SET = Almacena una clave y su valor asociado

r.set(“nombre”, “Angel”)

Opciones con SET

EX = Establece un tiempo de expiración en segundos

r.set(“nombre”, “Angel”) EX 300

PX = EX pero en milisegundos

r.set(“nombre”, “Angel”) PX 300

NX = Solo guarda la clave si no existe ya

r.set(“nombre”, “Angel”) NX

XX = Solo guarda la clave si existe ya

r.set(“nombre”, “Angel”) XX

GET = Devuelve los valores de las claves indicadas

r.get(“nombre”)

KEYS = Busca claves que coincidan con un patrón

Ejemplo clave

r.SET(“user:name” “Angel”)

r.KEYS(user:*)

Resultado

“user:Angel”

EXISTS = Verifica si una clave existe

r.EXISTS(“user:name”)

DEL = Elimina una clave

r.DELETE(“user:apellidos”)

EXPIRE = Establece un TTL de una clave en segundos

r.expire(“clave”,tiempo en segundos)

TTL = Te indica el ttl de esa clave(Si es numero positivo, te da los segundos que le quedan, si es -1, la clave existe pero no tiene caducidad y si es -2, la clave no existe)

r.TTL(“user:nombre”)

Type = Vemos el tipo de datos que almacena una clave

r.TYPE(“user:name”)

Flushbd() = Elimina todas las claves de la base de datos actual

r.flushbd()

flushall() = Elimina todas las claves de todas las bases de datos

COMANDO TRABAJAR CON **CADENAS**

SET

GET

DEL

INCR = Incrementa el valor numérico de una clave en 1

DECR = Decrementa el valor numérico de una clave en 1

EXPIRE

TTL

COMANDO TRABAJAR CON **LISTAS**

LPUSH = Añade uno o más valores al inicio de la lista

r.LPUSH("clave_pedidos",”valor_pedido”)

RPUSH = Añade uno o mas valores al final de la lista

r.RPUSH("clave_pedidos",”valor_pedido”)

LPOP = Elimina y devuelve el primer elemento de la lista

r.LPOP(“Clave_de_la_lista”)

RPOP = Elimina y devuelve el primer elemento de la lista

r.RPOP(“Clave_de_la_lista”)

LRANGE = Obtiene un rango de elementos de la lista

r.LRANGE(“clave_lista”,0,3)

Clave, valor por el que empieza el rango, valor por el que acaba

LLEN = Obtiene la longitud de la lista

r.LLEN(“clave_de_la_lista”)

BLPOP = Version de LPOP que es bloqueante, espera hasta que haya elementos

```
r.BLPOP(“Clave_de_la_lista”,timeout=0)
```

COMANDOS PARA **CONJUNTOS (SETS)**

Un conjunto es una colección desordenada de cadenas únicas

No mantienen un orden

SADD = Añade uno o más miembros a un conjunto

r.SADD(“Clave_conjunto”,”valor1,”valor2”)

SREM = Elimina uno o más miembros de un conjunto

r.SREM(“Clave_conjunto”,”valor1,”valor2”)

SMEMBERS = Obtiene todos los miembros del conjunto

r.smember(“Clave_conjunto”)

SISMEMBER = Comprueba si un miembro existe en un conjunto

r.SISMEMBER(“Clave_conjunto”, “miembro”)

SCARD = Obtiene el número de miembros en un conjunto

r.SCARD(“Clave_conjunto”)

SINTER = Compara Conjuntos y te dice los valores en común

r.SINTER(“Clave_conjunto”,”Clave_conjunto2”)

SUNION = Devuelve la unión de dos o más conjuntos

r.SUNION(“Clave_conjunto”,”Clave_conjunto2”)

COMANDOS PARA **CONJUNTOS ORDENADOS (SORTED SETS)**

SORTED SETS = SETS, pero cada miembro único tiene un SCORE asociado (Valor numérico en flotante)

Los elementos se ordenan según su score, Si varios miembros tienen el mismo score se ordena lexicográficamente

ZADD = Añade uno o más miembros a un conjunto ordenado con su score

r.ZADD(“Clave_Sorted_SETS, {Clave_miembro:Valor_Miembro, Clave_miembro2:Valor_Miembro2})

ZRANGE = Obtiene un rango de miembros por índice (Ordenado por score)

r.ZRANGE(“Clave_Sorted_SETS”,0,1, WITHSCORES = True)

clave_sorted_sets, Inicio,final y con valores

ZREVRANGE = ZRANGE pero de mayor a menor

r.REVRANGE(“Clave_Sorted_SETS”,0,1, WITHSCORES = True)

clave_sorted_sets, final, inicio y con valores

ZRANGEBYSCORE = Obtiene miembros dentro de un rango de scores

r.ZRANGEBYSCORE(“clave_SORTED_SETS”, Valor inicial,Valor final, WITHSCORES = True)

ZREM = Elimina uno o más miembros

r.ZREM(“Clave_Sorted_sets”,”Clave_miembro”)

ZSCORE = Obtiene el score de un miembro

r.ZCORE(“Clave_Sorted_sets”,”Clave_miembro”)

ZCARD = Obtiene el número de miembros

r.ZCARD(“Clave_SORTED_SETS”)

COMANDOS PARA **HASHES (HASH MAPS)**

Un hash es un mapa de campos de valores

HSET = Establece el valor de uno o más campos en un hash

r.HSET(“clave_hash”,campo,valor)

r.hset("usuario:1001", "nombre", "Alice")

r.hset("usuario:1001", "edad", 30)

r.hset("usuario:1001", "email", "alice@example.com")

HGET = Obtiene el valor de un campo específico

r.HGET(“Clave_hash”,campo)

HGETALL = Obtener los valores de todos los campos

r.HGETALL(“Clave_hash”)

HDEL = Elimina uno o más campos de un hash

r.HDEL(“Clave_hash”,campo)

HKEYS = Obtiene todos los nombres de campo de un hash

r-HKEYS(“Clave_hash”)

HVALS = Obtiene todos los valores de un Hash

r.HVALS(“Clave:hash”)

HLEN = Obtiene el número de campos de un hash

r.HLEN(“Clave_Hash”)

COMANDOS PARA **STREAMS**

Un stream en Redis es una estructura de datos que almacena secuencias de mensajes de manera ordenada. Puedes imaginarlo como un registro de eventos o una cola de mensajes, donde cada mensaje tiene:

Un ID único (basado en timestamp y secuencia).

Un conjunto de campos y valores (similar a un diccionario o hash).

XADD = Añade una nueva entrada al stream (El id puede ser * para generar uno automáticamente)

```
r.xadd("Clave_stream", {"usuario": "Bob", "mensaje": "Hola Alice"},id=”*”)
```

XRANGE = Lee un rango de entradas de un stream

```
r.XRANGE(“Clave_stream”,min=)
```

("mimensaje_stream", min='id_inicial', max='id_final')

XREAD = Lee entradas de uno o mas streams opcionalmente bloqueando hasta que hasta nuevas entradas

```
r.XREAD = ({"mimensaje_stream": "$"}, count=1, block=5000)
```

XGROUP CREATE = Crea un grupo de consumidores para un Stream

```
r.XGROUP_CREATE (name="mimensaje_stream", groupname="grupo1", id="$", mkstream=True)
```

XREADGROUP GROUP = Lee entrada de un stream usando un grupo de consumidores

```
r.xreadgroup(groupname="grupo1",
                consumername="consumidor1",
    streams={"mimensaje_stream": ">"},  # ">" = mensajes nuevos no entregados
    count=10)
```

COMANDOS PARA **BITMAPS**

Son un tipo de dato que permite tratar las cadenas de Redis como un array de bits

SETBIT = Establece o borra el bit en u offset dado

```
r.setbit("Clave_bit", 10, 1)
```

GETBIT = Obtiene el valor del bit en un offset dado

```
r.GETBIT(“Clave_bit”)
```

BITCOUNT = Cuenta el número de bits establecidos (a 1) en una cadena

```
r.BITCOUNT(“Clave_bit”,bit inicio,bit final)
```

BITPOS = Encuentra la posición del primer bit establecido o no establecido

```
r.BITPOS(“Clave_bit”,bit_inicio,bit_final)
```

COMANDOS PARA **HYPERLOGS**

Es un algoritmo probabilístico para estimas la cardinalidad de un conjunto(números de elementos únicos)

PDADD = Añade elementos a un Hyperlog

```
r.pfadd(“Clave_Hyperlog”,”valor”)
```

PFCOUNT = Estima la cardinalidad de uno o mas hyperloglog

```
r.pfcount(“Clave_Hyperlog”,”Clave_hyperlog2”)
```

PFMERGE = Combina varios Hyperloglog en uno

```
r.pfmerge("Clave_hiperlogunion ", "Hyperlog1", " Hyperlog2")
```

COMANDOS PARA Datos Geoespaciales

Permite almacenar coordenados de latitud y longitud y realizar consultas basadas en la distancia

GEOADD = Añade uno o más puntos geoespaciales

r.geoadd("Clave_datos”, (-3.70379, 40.41677, "Usuario:"))

clave, longitud,latitud,id

GEODIST = Calcula la distancia entre dos miembros

r.GEODIST ("Clave_datos", "Miembro1", "Miembro2", unit="km")

GEORADIUS = Encuentra miembros dentro de un radio dado

r.GEORADIUS(“Clave_datos”,longitud,latitud, distancia,unit= km)

GEORADIUSBYMEMBER = Georadius pero centrado en un miembro existente

r.georadiusbymember("Clave_datos", "Miebro", 1500, unit="km")

GEOHASH = Devuelve el Geohash de uno o mas miembres

r.GEOHASH(“clave_datros”,”Miembro”)

Geosearch = georadius mejorado

```
r.geosearch(
    "poi:locations",
    longitude = lon,
    latitude = lat,
    radius=100,
    unit = "m"
)
```