

# BBDD orientadas a clave-valor

## Redis

Ampliación a Bases de Datos

Profesor: Pablo Ramos

[pablo.ramos@u-tad.com](mailto:pablo.ramos@u-tad.com)

# INTRODUCCIÓN

- ¿Qué es Redis?
  - Base de datos de alto rendimiento
    - **In-Memory**: Optimizada para usar memoria principal frente al almacenamiento en disco.
  - **Cache**
    - Cache layer: Capa intermedia para sistemas de **alta demanda**
    - Least Recently Used (LRU) Cache
  - Sistema de **comunicación**
    - Publicación/subscripción de mensajes

# INTRODUCCIÓN

- **Persistencia:**
  - Almacenamiento de la BBDD en:
    - Memoria principal.
    - Memoria principal y virtual (disco duro).
  - Cualquiera de las dos versiones realiza de forma regular un volcado de la nueva información al disco duro.
- **Clústeres** (v3.x o mayor):
  - Información distribuida en varios nodos (BBDD grandes)
  - Robusto ante caídas parciales del cluster (Replicación)
- **Replicación**
  - Diseñado para trabajar en arquitecturas master-slave.
    - Arquitecturas tipo árbol. Cada nodo es master de las ramas que produce y slave de las ramas que lo preceden.

# INTRODUCCIÓN

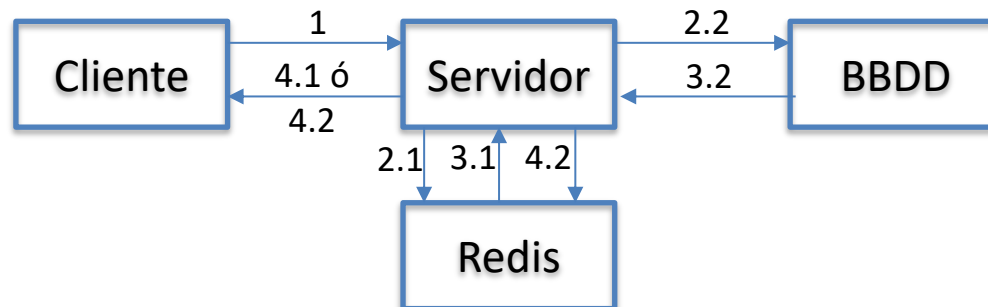
- Uso de **claves-valor** para almacenar la información:
  - **Tipo de datos** (valor):
    - Cadenas, cadenas binarias. (Números)
    - Listas
    - Conjuntos
    - Conjuntos ordenados
    - Tablas hash
    - Situaciones (Coordenadas)
  - **Durabilidad** de los datos opcional.

# CASOS DE USO

- Como **bases de datos**:
  - Almacenamiento de información de sesiones.
    - La sesión se almacena con una simple consulta en un único objeto.
    - Se puede establecer un tiempo de expiración.
    - La información de la sesión es accedida frecuentemente.
  - Perfiles de usuario y preferencias:
    - El perfil se almacena en un único objeto.
    - La información del perfil es accedida frecuentemente.
  - Información de cesta de la compra:
    - El perfil se almacena en un único objeto.
    - La información de la cesta es accedida frecuentemente.

# CASOS DE USO

- Como **cache**:
  - No es necesario serializar la información porque las estructuras de datos típicas son soportadas por Redis
    - Almacenamiento **datos** más consultados en un sistema informático.
    - Almacenamiento de las páginas más frecuentadas de una web.



# CASOS DE USO

- Como sistema de **mensajería**:
  - Sistema de procesamiento distribuido (colas de mensajes)
  - Sistema de comunicación como chats.
  - Timeline de una app social.
  - Mensajes en videojuegos multiusuario.
  - Sistemas de seguimiento en tiempo real

# TIPOS DE DATOS: Claves

- Binary-safe strings
  - Podemos utilizar cualquier cadena binaria como clave.
    - Cadenas (cadena vacía)
    - Imágenes
    - Etc.
- Tamaño de la clave (máx. 512Mb)
  - Cadenas muy largas (>1024bits,) no son eficientes.
  - No por mucho reducir mejoramos el rendimiento.
  - Mejor `miClave:2000` que `mK2000`
- Esquema de la clave
  - Es conveniente mantener un mismo esquema para todas las claves: e.g. `tipo_objeto:identificador`

# TIPOS DE DATOS: Cadenas

- El dato básico en Redis es la cadena.
- La cadena puede contener cualquier cadena o número, incluyendo datos binarios. (máx. 512MB)
- Comandos:
  - Asignar y obtener: SET(sobrescribe) and GET
    - > SET mykey "somevalue"
    - OK
    - > GET mykey
    - "somevalue"

# TIPOS DE DATOS: Cadenas

- Opciones comando SET:
  - EX segundos: Especifica un tiempo (segundos) en el que el dato expirará.
  - PX milisegundos: Especifica un tiempo (milisegundos) en el que el dato expirará.
  - NX: Asigna el valor solo si la clave no existe
  - XX: Asigna el valor solo si la clave existe.

```
> SET mykey "somevalue" xx
(nil)
> SET mykey "somevalue" nx px 1
Ok
> GET mykey
(nil)
```

# TIPOS DE DATOS: Cadenas

- Expiración de claves:
  - Además de las opciones asociadas al comando set para asignar el tiempo de expiración existen otros comandos.
    - EXPIRE: Asigna el tiempo de expiración a una clave en segundos.
    - PEXPIRE: Asigna el tiempo de expiración a una clave en milisegundos.
    - EXPIREAT (PEXPIREAT): Asigna el momento (unix timestamp) en el que la clave expirará.
    - PERSIST: Elimina el tiempo de expiración de una clave.
    - TTL (PTTL): Devuelve el tiempo restante hasta que la clave expire, -1 si no expira o -2 si la clave no existe.

```
> SET mykey "somevalue"
OK
> EXPIRE mykey 10
integer (1)
> TTL mykey
integer (9)
> PERSIST mykey
integer (1)
```

# TIPOS DE DATOS: Cadenas

- Incrementos y decrementos (atómico)
  - Obtiene el valor de una clave, lo convierte en un entero, incrementa o reduce su valor y lo vuelve a asignar.
    - INCR: Incrementa el valor en uno.
    - INCRBY n: Suma n al valor.
    - INCRBYFLOAT f: Suma f al valor.
    - DECR: Reduce el valor en uno.
    - DECRBY n: Resta n al valor.

```
> SET mykey "10"
```

```
OK
```

```
> INCR mykey
```

```
(integer) 11
```

```
> DECRBY mykey 3
```

```
(integer) 8
```

# TIPOS DE DATOS: Cadenas

- GETSET (atómico)
  - Asigna de forma atómica un valor a una clave y devuelve el valor antiguo.

```
> SET micontador "0"
OK
> INCR micontador
(integer) 1
> GETSET micontador "0"
"1"
> GET micontador
"0"
```

# TIPOS DE DATOS: Cadenas

- SET y GET múltiple (atómico)
  - MSET: Asigna de forma atómica una serie de valores a sus correspondientes claves. Si la clave existe, la sobrescribe. Para evitar sobrescritura se puede usar MSETNX.
  - MGET: Obtiene de forma atómica todos los valores correspondientes a las claves especificadas.

```
> MSET key1 "val1" key2 "val2"
OK
> GET key1
"val1"
> MGET key1 key2 key3
(1) "val1"
(2) "val2"
(3) (nil)
```

# TIPOS DE DATOS: Cadenas

- Existe una clave
  - EXISTS: Devuelve 1 si una clave existe 0 si no existe.

```
> SET key1 "val1"
```

```
OK
```

```
> EXISTS key1
```

```
integer (1)
```

```
> EXISTS key2
```

```
integer (0)
```

# TIPOS DE DATOS: Cadenas

- Eliminar claves
  - DEL: Elimina una o varias claves

```
> MSET key1 "val1" key2 "val2" key3 "val3"
```

```
OK
```

```
> DEL key1
```

```
integer (1)
```

```
> DEL key2 key3 key4
```

```
integer (2)
```

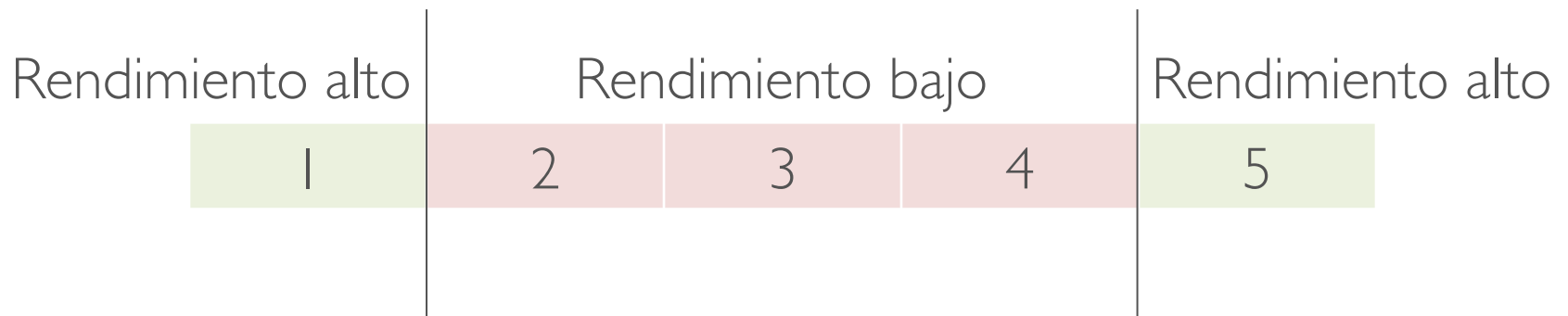
# TIPOS DE DATOS: Cadenas

- Modificación de cadenas
  - APPEND: Concatena una cadena a la cadena de una clave especificada.

```
> SET key1 "val1"
OK
> APPEND key1 " val2"
integer (9)
> GET key1
"val1 val2"
```

# TIPOS DE DATOS: Listas

- En Redis las listas son listas enlazadas
  - Para permitir inserción de datos de una forma muy eficiente (tiempo de inserción es constante)
  - El acceso por índice es muy costoso.
  - En caso de querer acceder a una lista por índice:
    - Devolver la lista
    - Usar sorted sets



# TIPOS DE DATOS: Listas

- Inserción de datos en listas: PUSH
  - R PUSH: Inserta por la cola/derecha de la lista
  - L PUSH: Inserta por la cabeza/izquierda de la lista
- Consulta de datos en listas
  - L RANGE: Devuelve una sublista comprendida entre el primer índice y el segundo. Los índices pueden ser negativos indicando que se inicia desde el final.

```
> LPUSH list "val1" "val2"  
integer (2)  
> RPUSH list "val3"  
integer (3)  
> LRANGE list 0 -1  
1)val2 2)val1 3)val3
```

# TIPOS DE DATOS: Listas

- Obtención de datos (eliminándolo de la lista): POP
  - RPOP: Obtiene un valor de la cola/derecha de la lista.
  - LPOP: Obtiene un valor de la cabeza/izquierda de la lista.

```
> LPUSH list "val1" "val2" "val3"
integer (3)
> RPOP list
"val3"
> LPOP list
"val1"
> LPOP list
"val2"
> LPOP list
(nil)
```

# TIPOS DE DATOS: Listas

- Recortar listas:
  - LTRIM: Recorta una lista eliminando los elementos más allá del primer índice y el segundo. Los índices pueden ser negativos indicando que se inicia desde el final.

```
> RPUSH list "1" "2" "3" "4" "5"  
(integer) 5  
> LTRIM list 0 2  
OK  
> LRANGE list 0 -1  
1) "1"  
2) "2"  
3) "3"
```

# TIPOS DE DATOS: Listas

- Obtención de datos con bloqueo: BxPOP
  - BRPOP n: Obtiene un valor de la cola/derecha de la lista. Si la lista está vacía, espera n segundos a que alguien inserte un valor, sino devuelve nil.
  - BLPOP n: Obtiene un valor de la cabeza/izquierda de la lista. Si la lista está vacía, espera n segundos a que alguien inserte un valor, sino devuelve nil.

```
> BRPOP list 20
```

```
1) "list"
```

```
2) "val1"
```

```
(14,83s)
```

```
> BRPOP list 1
```

```
(nil)
```

```
(1,26s)
```

Cliente 1

```
> LPUSH list "val1"
```

Cliente 2

# TIPOS DE DATOS: Listas

- Obtención de datos con bloqueo: BxPOP
  - Si se especifica tiempo 0, espera de forma indefinida.
  - Si se especifica más de una lista y todas están vacías, espera hasta que alguna de las listas tenga un valor.
  - Una lista puede ser bloqueada por dos clientes al mismo tiempo. El cliente que primero la bloquea, recibirá el primer elemento insertado.

```
> BRPOP list1 list2 0
```

```
1) "list1"
```

```
2) "val1"
```

```
(76,83s)
```

Cliente 1

```
> LPUSH list1 "val1"
```

Cliente 2

# TIPOS DE DATOS: Listas

- Obtención e inserción simultánea:
  - RPOPLPUSH: Inserta un valor de la cola/derecha de la primera lista especificada y lo inserta en la cabeza/izquierda de la segunda lista especificada.
    - Si la primera lista está vacía devuelve nil y no se realiza ninguna operación.
    - Si la primera y segunda lista son la misma, el valor de la cola es insertado en la cabeza creando así una lista circular.

```
> RPUSH list "1" "2" "3"  
(integer) 3  
> RPOPLPUSH list list  
"3"  
> LRANGE list 0 -1  
1) "3"  
2) "1"  
3) "2"
```

# TIPOS DE DATOS: Listas

- Otros operadores:
  - LINDEX: Devuelve el valor de una lista en el índice indicado.
  - LSET n: Asigna un valor especificado en la posición n especificada
  - LINSERT: Inserta un valor antes (BEFORE) o después (AFTER) un valor especificado.
  - LLEN: Devuelve la longitud de una lista
  - LREM n: Elimina los n primeros valores de una lista igual a un valor especificado. Si n es positivo empieza por la cabeza si es negativo por la cola. Si el valor es cero, elimina todas las apariciones.
  - LPUSHX y RPUSHX: Inserta un valor en la lista solo si existe la clave y es una lista.

# TIPOS DE DATOS: Tablas hash

- Inserción y consultas:
  - HSET: Inserta un par clave-valor en la tabla hash especificada. Devuelve 1 si el valor no existe y lo crea, 0 si existe y lo modifica.
  - HSETNX: Inserta un par clave-valor en la tabla hash especificada si la clave no existe. Devuelve 1 si el valor no existe y lo crea o 0 si existe.
  - HMSET: Inserta un conjunto de pares clave-valor en la tabla hash especificada.
  - HGET: Obtiene el valor para una clave de una tabla hash especificada.
  - HMGET: Obtiene los valores para un conjunto de claves de una tabla hash especificada.

```
HMSET hash "clave1" "valor1" "clave2" "valor2"
OK
> HGET hash "clave1"
"valor1"
> HSET hash "clave2" "valor3"
integer (0)
> HMGET hash "clave1" "clave2"
1) "valor1"    2) "valor3"
```

# TIPOS DE DATOS: Tablas hash

- Consultas:
  - HGETALL: Devuelve todas las claves y valores de una tabla hash consultada.
  - HKEYS: Devuelve todas las claves de una tabla hash consultada.
  - HVALS: Devuelve todos los valores de una tabla hash consultada.

```
HMSET hash "clave1" "valor1" "clave2" "valor2"  
OK
```

```
> HGETALL hash
```

```
1) "clave1" 2) "valor1" 3) "clave2" 4) "valor2"
```

```
> HKEYS hash
```

```
1) "clave1" 2) "clave2"
```

```
> HVALS hash
```

```
1) "valor1" 2) "valor2"
```

# TIPOS DE DATOS: Tablas hash

- Modificadores:
  - HINCRBY n: Incrementa el valor asociado a una clave en una tabla especificada por n.
  - HINCRBYFLOAT f: Suma f al valor asociado a una clave en una tabla especificada.

```
> HSET hash "key" 6.5  
integer (1)
```

```
> HINCRBYFLOAT hash "key" 0.1  
"6.6"
```

```
> HINCRBYFLOAT hash "key" 1.0e3  
"1006.6"
```

# TIPOS DE DATOS: Tablas hash

- Otros operadores:
  - HDEL: Elimina la clave especificada de la tabla hash. Devuelve 1 si la clave existe y la borra, 0 en caso contrario.
  - HEXISTS: Devuelve 1 si la clave existe en la tabla hash, 0 en caso contrario.
  - HLEN: Devuelve el número de elementos almacenados en la tabla hash.
  - HSTRLEN: Devuelve la longitud de una cadena almacenada en una clave de la tabla hash. En caso de no existir la clave devuelve 0.

# TIPOS DE DATOS: Sets

- Inserción y consulta:
  - SADD: Añade uno o varios a elementos a un set. Si existe algún elemento repetido, no lo inserta. Devuelve el número total de elementos insertados.
  - SPOP: Devuelve y elimina un elemento aleatorio del set especificado.

```
> SADD set "val1" "val2" "val3" "val3"
integer (3)
> SPOP set
"val2"
> SPOP set
"val1"
```

# TIPOS DE DATOS: Sets

- Consultas:
  - SRANDMEMBER: Devuelve un elemento aleatorio del set especificado.
  - SMEMBERS: Devuelve el listado de elementos que constituyen el set especificado.
  - SISMEMBER: Devuelve 1 si el elemento consultado esta en el set, 0 en caso contrario.

```
> SADD set "val1" "val2" "val3"
(integer) 3
> SISMEMBER set "val2"
(integer) 1
> SRANDMEMBER set
"val1"
> SMEMBERS
1) "val1"    2) "val3"
```

# TIPOS DE DATOS: Sets

- Operadores sobre conjuntos:
  - SDIFF: Devuelve un conjunto con el resultado de la diferencia del primer set especificado y los siguientes sets.
  - SDIFFSTORE: Almacena en un set especificado el resultado de la diferencia del primer set especificado y los siguientes sets. Devuelve un número entero especificando el tamaño del set resultante.
  - UNION: Devuelve un set con el resultado de la unión de todos los sets especificados.
  - UNIONSTORE: Almacena en un set especificado el resultado de la unión de todos los sets especificados. Devuelve un número entero especificando el tamaño del set resultante.

```
> SADD set1 "a" "b" "c"
(integer) 3
> SADD set2 "a" "c" "d"
(integer) 3
> SDIFF set1 set2
1) "b"
> SUNIONSTORE set3 set1 set2
(integer) 4
```

# TIPOS DE DATOS: Sets

- Operadores sobre conjuntos:
  - SINTER: Devuelve un set con el resultado de la intersección de todos los sets especificados.
  - SINTERSTORE: Almacena en un set especificado el resultado de la intersección de todos los sets especificados. Devuelve un número entero especificando el tamaño del set resultante.

```
> SADD set1 "a" "b" "c"
(integer) 3
> SADD set2 "a" "c" "d"
(integer) 3
> SINTER set1 set2
1) "c" 2) "a"
> SINTERSTORE set3 set1 set2
(integer) 2
```

# TIPOS DE DATOS: Sets

- Otros operadores:
  - SREM: Elimina uno o varios elementos de un set especificado. Devuelve el número de elementos eliminados o 0 si no se elimina ningún elemento o el set no existe.
  - SCARD: Devuelve la cardinalidad del set especificado.
  - SMOVE: Mueve un elemento de un set origen especificado a otro set destino. Devuelve 1 si existe el elemento y se mueve, 0 en caso contrario.

```
> SADD set1 "a" "b" "c"
(integer) 3
> SREM set1 "a"
(integer) 1
> SMOVE set1 set2 b
(integer) 1
> SCARD set1
(integer) 1
```

# TIPOS DE DATOS: Sorted sets

- Los sorted set son sets en el que los elementos tienen una puntuación, “score”, asociada por el cual se ordenan los elementos dentro del set.
- Tienen cierta similitud con las tablas hash salvo porque en los sorted sets los elementos estarían ordenados por el valor. La clave sería la puntuación.
- Las puntuaciones son números en coma flotante.
- Si dos valores tienen la misma puntuación, entonces se ordena por orden lexicográfico del valor del elemento.

# TIPOS DE DATOS: Sorted sets

- Inserción y consultas:
  - ZADD: Inserta uno varios elementos puntuación/valor al sorted set especificado. Devuelve el número de elementos insertados. Si alguno de los valores ya existe, se actualiza la puntuación.
  - ZRANGE: Devuelve un subconjunto ordenado comprendido entre el primer índice y el segundo. Los índices pueden ser negativos indicando que se inicia desde el final. Si se añade el argumento WITHSCORES devuelve las puntuaciones asociadas a cada elemento.
  - ZREVRANGE: Devuelve un subconjunto ordenado en orden inverso comprendido entre el primer índice y el segundo. Los índices pueden ser negativos indicando que se inicia desde el final.

```
> ZADD ss 1 "a" 3 "c" 2 "b"
(integer) 3
> ZRANGE ss 0 -1
1) "a"    2) "b"    3) "c"
> ZREVRANGE ss 0 -1 WITHSCORES
1) "c"    2) 3      3) "b"    4) 2      5) "a"    6) 1
```

# TIPOS DE DATOS: Sorted sets

- Consultas avanzadas:
  - ZRANGEBYSCORE min max: Devuelve un subconjunto ordenado cuya puntuación está comprendida entre el min y el max especificado. Acompañado de ( el min o max será exclusivo, si no será inclusivo por defecto. Los valores de comparación pueden ser  $-\text{inf}$  o  $+\text{inf}$ . Si se añade el argumento WITHSCORES devuelve las puntuaciones asociadas a cada elemento. También se le puede especificar un offset y límite de elementos a devolver con el argumento LIMIT.
  - ZREVRANGEBYSCORE max min: Devuelve un subconjunto ordenado en orden inverso cuya puntuación está comprendida entre el min y el max especificado. Los valores de comparación pueden ser  $-\text{inf}$  o  $+\text{inf}$ .

```
> ZADD ss 1 "a" 3 "c" 2 "b" 4 "d"  
(integer) 4  
> ZRANGEBYSCORE ss -inf (3 LIMIT 1 2  
1) "b"
```

# TIPOS DE DATOS: Sorted sets

- Consultas avanzadas:
  - ZRANGEBYLEX min max: Para listas con elementos con la misma puntuación, devuelve un subconjunto ordenado por valor cuyos valores están lexicográficamente comprendidos entre el min y el max especificado. Los valores de comparación pueden ser – o +. También se le puede especificar un offset y límite de elementos a devolver con el argumento LIMIT. Es necesario especificar ( o [ en min y max para indicar si es inclusivo o exclusivo.
  - ZREVRANGEBYLEX max min: Para listas con elementos con la misma puntuación, devuelve un subconjunto ordenado por valor cuyos valores están lexicográficamente comprendidos entre el min y el max especificado. Los valores de comparación pueden ser – o +.

```
> ZADD ss 1 "a" 1 "c" 1 "b" 1 "d"  
(integer) 4  
> ZRANGEBYLEX ss [a + LIMIT 0 2  
1) "a"    2) "b"
```

# TIPOS DE DATOS: Sorted sets

- Operadores sobre conjuntos:
  - ZINTERSTORE: Almacena en un sorted set especificado el resultado de la intersección de todos los sorted sets especificados.
    - Devuelve un número entero especificando el tamaño del set resultante.
    - Se debe indicar el número de sorted set implicados en la operación.
    - La puntuación de elementos que existen en varios de los conjuntos es por defecto igual a la suma sus puntuaciones. Se puede ponderar mediante el argumento WEIGHTS. Se puede especificar otra operación diferente a la suma con AGGREGATE, siendo las posibilidades MIN y MAX.

```
> ZADD ss1 1 "a" 1 "b"
(integer) 2
> ZADD ss2 2 "a" 2 "c"
(integer) 2
> ZINTERSTORE ss3 2 ss1 ss2 WEIGHTS 1 2
(integer) 1
> ZRANGE ss3 0 -1 WITHSCORES
1) a      2) 5
```

# TIPOS DE DATOS: Sorted sets

- Operadores sobre conjuntos:
  - ZUNIONSTORE: Almacena en un sorted set especificado el resultado de la unión de todos los sorted sets especificados.
    - Devuelve un número entero especificando el tamaño del set resultante.
    - Se debe indicar el número de sorted set implicados en la operación.
    - La puntuación de elementos que existen en varios de los conjuntos es por defecto igual a la suma sus puntuaciones. Se puede ponderar mediante el argumento WEIGHTS. Se puede especificar otra operación diferente a la suma con AGGREGATE, siendo las posibilidades MIN y MAX.

```
> ZADD ss1 1 "a" 1 "b"
(integer) 2
> ZADD ss2 2 "a" 2 "c"
(integer) 2
> ZUNIONSTORE ss3 2 ss1 ss2 AGGREGATE MAX
(integer) 3
> ZRANGE ss3 0 -1 WITHSCORES
1) b    2) 1    3) a    4) 2    5) c    6) 2
```

# TIPOS DE DATOS: Sorted sets

- Eliminar elementos:
  - ZREM: Elimina si existe/n los elemento/s con valor/es especificado/s del sorted set. Devuelve el número de elementos eliminados.
  - ZREMRANGEBYRANK min max: Elimina los elementos comprendidos entre las posiciones min y max de una sorted set especificada. Devuelve el número de elementos eliminados.

```
> ZADD ss 1 "a" 1 "c" 1 "b" 1 "d"
(integer) 4
> ZREM ss "a"
(integer) 1
> ZREMBYRANK ss 0 1
(integer) 2
> ZREVRANGE ss 0 -1
1) c    2) d
```

# TIPOS DE DATOS: Sorted sets

- Eliminar elementos:
  - ZREMRANGEBYSCORE min max: Elimina los elementos cuya puntuación esta comprendida entre el min y el max especificado.
  - ZREMRANGEBYLEX min max: Para listas con elementos con la misma puntuación, elimina los elementos lexicográficamente comprendidos entre el min y el max especificado.

```
> ZADD ss 1 "a" 1 "c" 1 "b" 1 "d"  
(integer) 4  
> ZREMRANGEBYLEX ss - (c  
integer (2)  
>ZRANGE ss 0 -1  
1) "c"    2) "b"
```

# TIPOS DE DATOS: Sorted sets

- Otros operadores:
  - ZSCORE: Devuelve la puntuación de un valor especificado si este existe en el set.
  - ZRANK: Devuelve la posición de un valor especificado si este existe en el set.
  - ZREVRANK: Devuelve la posición empezando por el final del sorted set de un valor especificado si este existe.
  - ZCARD: Devuelve el número de elementos que integran el sorted set.
  - ZCOUNT min max: Devuelve el número de elementos cuyas puntuaciones están comprendidas entre min y max.
  - ZLEXCOUNT min max: En un sorted set con la misma puntuación para todos los elementos, devuelve el número de elementos cuyos valores están comprendidas entre min y max.
  - ZINCRBY n: Incrementa la puntuación de un valor especificado por n.

# TIPOS DE DATOS: Bits

- Inserción y consultas:
  - SETBIT: Inserta un bit en la posición especificada. Devuelve el valor del bit antes de modificarlo.
  - GETBIT: Obtiene el bit de la posición especificada.
  - BITOP: Realiza la operación específica (AND, OR, XOR y NOT) sobre las claves especificadas y las almacena en otra clave.
  - BITCOUNT: Cuenta el número de bits activos entre las posiciones especificadas.
  - BITPOS b: Devuelve la posición del primer bit igual a b entre las posiciones especificadas.

```
>SETBIT bits 0 1
(integer) 0
>SETBIT bits 1 0
(integer) 0
>BITOP not res bits
(integer) 1
>BITPOS res 1 0 -1
(integer) 1
```

# TIPOS DE DATOS: Iteradores

- Iteradores:
  - Uno de los argumentos que toman es el identificador del cursor. Si este es igual a 0, crea un nuevo cursor.
  - El iterador devuelve un array de dos elementos. El primero es el cursor que se ha de llamar para continuar iterando. El segundo es la lista de elementos iterados.
  - Se puede especificar sobre cuantos elementos iterar con el argumento COUNT.
  - Se puede restringir el conjunto de elementos sobre el que iterar con el argumento MATCH.

# TIPOS DE DATOS: Iteradores

- Iteradores:
  - SCAN: Itera sobre las claves almacenadas en la base de datos actual.
  - SSCAN: Itera sobre los elementos almacenados en un set.
  - HSCAN: Itera sobre los elementos almacenados en una tabla hash.
  - ZSCAN: Itera sobre los elementos almacenados en un set ordenado.

```
> SADD set a1 a2 a3 a4 a5 b1 b2 b3 b4 c1
(integer) 10
> SSCAN set 0 MATCH a* COUNT 1
1) 4      2) 1) "a5"
> SSCAN set 4 MATCH a* COUNT 1
1) 2      2) 1) "a3"
```

# PIPELINING

- En redis no es necesario esperar a obtener la respuesta de las operaciones realizadas. Se puede enviar **cadenas de operaciones** y esperar más adelante a obtener la respuesta.

```
> SET a 1
OK
> INCR a
(integer) 2
> INCR a
(integer) 3
> INCR a
(integer) 4
```

```
> SET a 1
OK
> INCR a
> INCR a
> INCR a
(integer) 2
(integer) 3
(integer) 4
```

# TRANSACCIONES

- Permite ejecutar un conjunto de comandos en un solo paso.
  - Los comandos son **serializados** y ejecutados **secuencialmente** como una operación única y **atómica**.
  - **Todos** los comandos son ejecutados, en caso contrario, **ningún** comando es ejecutado.
  - No existe la posibilidad de **rollback**. Una vez se ejecuta una transacción no hay posibilidad de fallo en ejecución.
    - **Cuidado:** Puede haber fallos de programación, pero deberían haber sido detectados en desarrollo.

# TRANSACCIONES: Ejemplo de transacción

- La transacción se comienza con el comando MULTI.  
    > MULTI  
    OK
- Se encolan las operaciones que se deseen  
    > INCR foo  
    QUEUED  
    > INCR bar  
    QUEUED
- Se ejecutan  
    > EXEC  
    1) (integer) 1  
    2) (integer) 2
- O bien se descartan  
    > DISCARD  
    OK

# TRANSACCIONES: Watch

- Permite controlar el acceso de otros clientes a ciertas variables durante una transacción.
  - Método **Compare & Swap**: Comprueba si ha habido algún cambio en las variables desde que se bloqueó, y si no es así, se realiza la operación.
  - A esta operación se le denomina **bloqueo optimístico** porque se espera que ningún otro cliente acceda a esas variables.
  - En el caso de que otro cliente acceda, se desecha la transacción y tendremos que repetirla.

# TRANSACCIONES: Ejemplo con Watch

- Situación hipotética en el que no existe el operador INCR
  - > WATCH mykey
  - > val = GET mykey
  - > val = val + 1
  - > MULTI
  - > SET mykey \$val
  - > EXEC

# PUBLICACION Y SUSCRIPCIÓN

- Redis permite la **suscripción** y **publicación** de mensajes por canales
  - SUBSCRIBE: Suscribirse a canales especificados.
  - PSUBSCRIBE: Suscribirse a canales que cumplan los patrones especificados.
  - PUBLISH: Enviar un mensaje al canal especificado.
  - UNSUBSCRIBE y PUNSUBSCRIBE: Finalizar la suscripción de los canales especificados.

```
> PSUBSCRIBE chanel.*
```

```
1) psubscribe
```

```
2) chanel.*
```

```
3) (integer) 1
```

```
4) pmessage
```

```
5) chanel.*
```

```
6) chanel.1
```

```
7) hi!
```

```
> PUBLISH chanel.1 hi!
```

```
(integer) 1
```

# Cache

- Redis permite implementar diferentes modalidades de Cache.
  - Cuando se llega al límite de memoria, se desecha (**evict**) uno de los elementos almacenados.
  - El límite de memoria se define con **config set**:  
`config set maxmemory 100mb`
  - Redis proporciona diversos metodos de desalojo (**Eviction Policies**) :
    - **noeviction**: Cuando se llega al límite da error. (usar DEL)
    - **allkeys-lru**: Least Recently Used puro.
    - **volatile-lru**: LRU puro sobre elementos que expiran.
    - **allkeys-random**: Aleatorio sobre todos los elementos.
    - **volatile-random**: Aleatorio sobre elementos que expiran.
    - **volatile-ttl**: Elimina los elementos que expiran antes.