Redis

Extension to Databases

Professor: Pablo Ramos

pablo.ramos@u-tad.com

INTRODUCTION

- What is Redis ?
  - High performance database
    - **In- Memory** : Optimized to use main memory over disk storage.
  - **Cache**
    - Cache layer : Intermediate layer for **high demand systems**
    - Least Recently Used (LRU) Cache
  - **Communication** system
    - Posting/Subscribing to Messages

INTRODUCTION

- **Persistence** :
  - Database storage in:
    - Main memory.
    - Main and virtual memory (hard disk).
  - Either version regularly dumps new information to the hard drive.
- **Clusters** (v3.x or greater):
  - Information distributed across multiple nodes (large databases)
  - Robust against partial cluster failures (Replication)
- **Replication**
  - Designed to work in master- slave architectures .
    - Tree-type architectures. Each node is master of the branches it produces and slave of the branches that precede it.

INTRODUCTION

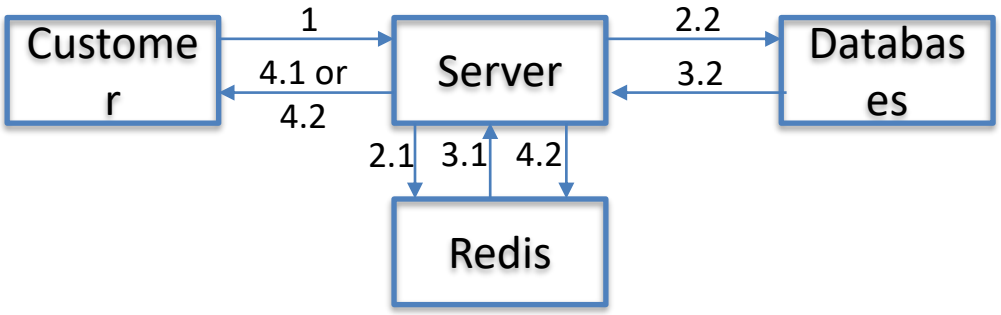
- Using **key-values** to store information:
  - **Data type** (value):
    - Strings, binary strings. (Numbers)
    - Lists
    - Sets
    - Ordered sets
    - Hash tables
    - Situations (Coordinates)
  - Optional data **durability .**

USE CASES

- As **databases** :
  - Storage of session information.
    - The session is stored with a simple query in a single object.
    - An expiration time can be set.
    - Session information is accessed frequently.
  - User profiles and preferences:
    - The profile is stored in a single object.
    - Profile information is frequently accessed.
  - Shopping cart information:
    - The profile is stored in a single object.
    - Basket information is accessed frequently.

USE CASES

- As **a cache** :
  - There is no need to serialize the data because typical data structures are supported by Redis
    - Storing the most frequently consulted **data** in a computer system.
    - Storing the most frequently visited pages on a **website** .



USE CASES

- As a **messaging system** :
  - Distributed processing system (message queues)
  - Communication system such as chats.
  - Timeline of a social app.
  - Messages in multiplayer video games.
  - Real-time tracking systems

DATA TYPES: Keys

- Binary-safe strings
  - We can use any binary string as a key.
    - Strings (empty string)
    - Images
    - Etc.
- Key size (max 512Mb)
  - Very long strings (>1024bits,) are not efficient.
  - Reducing a lot does not improve performance.
  - Better **MiClave:2000** than **mK2000**
- Key Schematic
  - It is convenient to maintain the same scheme for all keys: eg . **object_type:identifier**

DATA TYPES: Strings

- The basic data in Redis is the string.
- The string can contain any string or number , including binary data. (max. 512MB)

- Commands:
  - Set and Get: SET(overwrite) and GET

```
> SET mykey " somevalue "
OK
> GET mykey
" some value "
```

DATA TYPES: Strings

- SET command options:
  - EX seconds: Specifies a time (seconds) in which the data will expire.
  - PX milliseconds: Specifies a time (milliseconds) in which the data will expire.
  - NX: Assigns the value only if the key does not exist
  - XX: Assign the value only if the key exists.

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

DATA TYPES: Strings

- Key expiration:
  - In addition to the options associated with the set command to assign the expiration time, there are other commands.
    - EXPIRE: Assigns the expiration time to a key in seconds.
    - PEXPIRE: Assigns the expiration time to a key in milliseconds.
    - EXPIREAT (PEXPIREAT): Assigns the time (unix timestamp) at which the key will expire.
    - PERSIST: Removes the expiration time from a key.
    - TTL (PTTL): Returns the time remaining until the key expires, -1 if it does not expire or -2 if the key does not exist.

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

DATA TYPES: Strings

- Increments and decrements (atomic)
  - Gets the value of a key, converts it to an integer, increments or decrements its value, and assigns it back.
    - INCR: Increases the value by one.
    - INCRBY n: Adds n to the value.
    - INCRBYFLOAT f: Adds f to the value.
    - DECR: Decreases the value by one.
    - DECRBY n: Subtracts n from the value.

```
> SET mykey “10”
OK
> INCR mykey
(integer) 11
> DECRBY mykey 3
(integer) 8
```

DATA TYPES: Strings

- GETSET (atomic)
  - Atomically assigns a value to a key and returns the old value.

```
> SET my counter “0”
OK
> INCR my counter
(integer) 1
> GETSET mycounter “0”
“1”
> GET mycounter
“0”
```

DATA TYPES: Strings

- Multiple SET and GET (atomic)
  - MSET: Atomically assigns a series of values to their corresponding keys. If the key exists, it overwrites it. To avoid overwriting, MSETNX can be used.
  - MGET : Atomically gets all values corresponding to the specified keys.

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

DATA TYPES: Strings

- There is a key
  - EXISTS: Returns 1 if a key exists 0 if it does not exist.

```
> SET key1 “val1”
OK
> EXISTS key1
integer (1)
> EXISTS key2
integer (0)
```

DATA TYPES: Strings

- Delete keys
  - DEL: Deletes one or more keys

```
> MSET key1 “val1” key2 “val2” key3 “val3”
OK
> DEL key1
integer (1)
> DEL key2 key3 key4
integer (2)
```

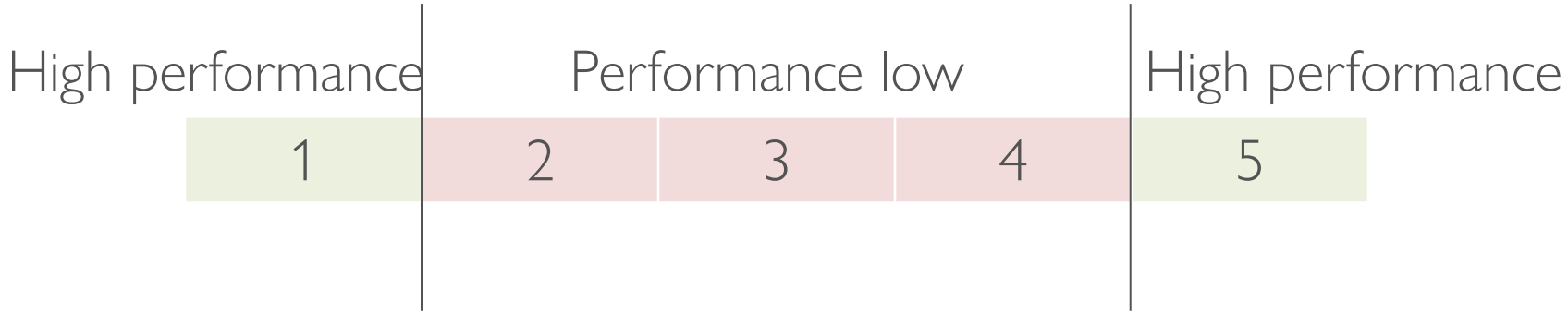
DATA TYPES: Strings

- Chain modification
  - APPEND: Concatenates a string to the string of a specified key.

```
> SET key1 “val1”
OK
> A PPEND key1 “ val2”
integer (9)
> GET key1
“val1 val2”
```

DATA TYPES: Lists

- In Redis lists are linked lists
  - To allow data insertion in a very efficient way (insertion time is constant)
  - Index access is very expensive.
  - If you want to access a list by index:
    - Return the list
    - Using sorted sets



DATA TYPES: Lists

- Inserting data into lists: PUSH
  - RPUSH: Insert from the tail /right of the list
  - LPUSH: Insert from the head/left of the list
- Querying data in lists
  - LRANGE: Returns a sublist between the first index and the second. The indexes can be negative, indicating that it starts from the end.

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

DATA TYPES: Lists

- Getting data (removing it from the list): POP
  - RPOP: Gets a value from the tail/right of the list.
  - LPOP: Gets a value from the head/left of the list.

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

DATA TYPES: Lists

- Trimming lists:
  - LTRIM: Trims a list by removing elements beyond the first and second indices. Indices can be negative indicating that the list is started from the end.

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

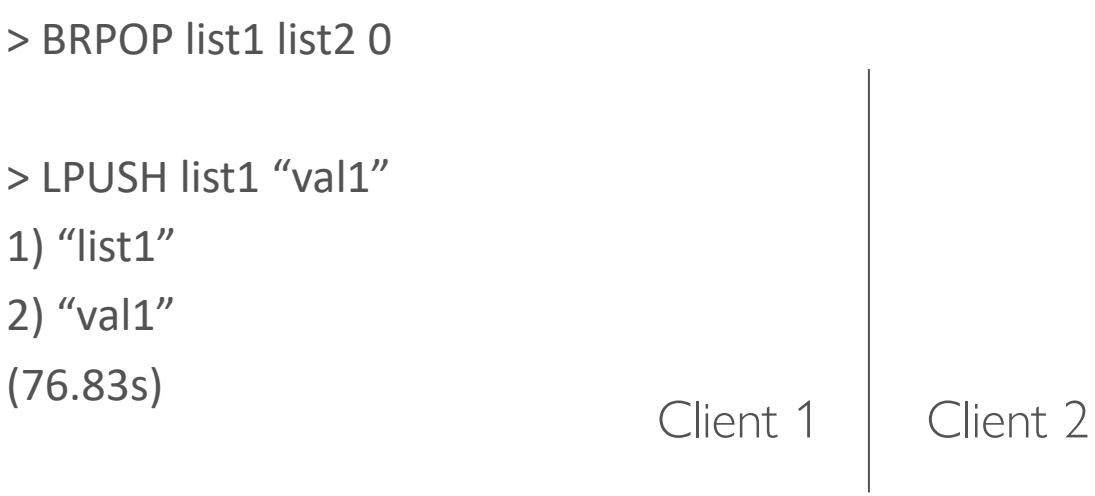
DATA TYPES: Lists

- Blocking data fetching: BxPOP
  - BRPOP n: Gets a value from the tail/right of the list. If the list is empty, waits n seconds for someone to insert a value, otherwise returns nil .
  - BLPOP n: Gets a value from the head/left of the list. If the list is empty, waits n seconds for someone to insert a value, otherwise returns nil .

Client 1	Client 2
> BRPOP list 20	
	> LPUSH list “val1”
1) “list”	
2) “val1”	
(14.83s)	
> BRPOP list 1	
(nil)	
(1.26s)	

DATA TYPES: Lists

- Blocking data fetching: BxPOP
  - If time 0 is specified, wait indefinitely.
  - If more than one list is specified and all are empty, wait until one of the lists has a value.
  - A list can be locked by two clients at the same time. The client that first locked it will receive the first inserted element.



DATA TYPES: Lists

- Simultaneous obtaining and insertion:
  - RPOPLPUSH: Inserts a value from the tail/right of the first specified list and inserts it into the head/left of the second specified list.
    - If the first list is empty, nil is returned and no operation is performed.
    - If the first and second lists are the same, the value at the tail is inserted into the head thus creating a circular list.

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

DATA TYPES: Lists

- Other operators:
  - LINDEX: Returns the value of a list at the indicated index.
  - LSET n: Assigns a specified value at the specified position n
  - LINSERT: Inserts a value before (BEFORE) or after (AFTER) a specified value.
  - LLEN: Returns the length of a list
  - LREM n: Removes the first n values from a list equal to a specified value. If n is positive, it starts at the head; if it is negative, it starts at the tail. If the value is zero, it removes all occurrences.
  - LPUSHX and RPUSHX: Insert a value into the list only if the key exists and is a list.

DATA TYPES: Hash tables

- Insertion and queries:
  - HSET: Inserts a key-value pair into the specified hash table. Returns 1 if the value does not exist and creates it, 0 if it exists and modifies it.
  - HSETNX: Inserts a key-value pair into the specified hash table if the key does not exist. Returns 1 if the value does not exist and creates it or 0 if it exists.
  - HMSET: Inserts a set of key-value pairs into the specified hash table.
  - HGET: Gets the value for a key from a specified hash table.
  - HMGET: Gets the values for a set of keys from a specified hash table.

```
HMSET hash ”key1” ”value1” ”key2” ”value2”
OK
> HGET hash “key1”
”value1”
> HSET hash ”key2” ”value3”
 integer (0)
> HMGET hash ”key1” ”key2”
1) ”value1” 2) ”value3”
```

DATA TYPES: Hash tables

- Questions:
  - HGETALL: Returns all keys and values from a queried hash table.
  - HKEYS: Returns all the keys of a queried hash table.
  - HVALS: Returns all values from a queried hash table.

```
HMSET hash ”key1” ”value1” ”key2” ”value2”
OK
> HGETALL hash
1) ”key1” 2) ”value1” 3) ”key2” 4) ”value2”
> HKEYS hash
1) ”key1” 2) ”key2”
> HVALS hash
1) ”value1” 2) ”value2”
```

DATA TYPES: Hash tables

- Modifiers:
  - HINCRBY n: Increments the value associated with a key in a table specified by n.
  - HINCRBYFLOAT f: Adds f to the value associated with a key in a specified table.

```
> HSET hash ” key ” 6.5
 integer (1)
> HINCRBYFLOAT hash “ key ” 0.1
”6.6”
> HINCRBYFLOAT hash “ key ” 1.0e3
”1006.6”
```

DATA TYPES: Hash tables

- Other operators:
  - HDEL : Removes the specified key from the hash table. Returns 1 if the key exists and deletes it, 0 otherwise.
  - HEXISTS: Returns 1 if the key exists in the hash table, 0 otherwise.
  - HLEN: Returns the number of elements stored in the hash table.
  - HSTRLEN: Returns the length of a string stored in a hash table key. If the key does not exist, it returns 0.

DATA TYPES: Sets

- Insertion and consultation:
  - SADD: Adds one or more elements to a set. If there are any duplicate elements, it does not insert them. Returns the total number of elements inserted.
  - SPOP: Returns and removes a random element from the specified set.

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

DATA TYPES: Sets

- Questions:
  - SRANDMEMBER: Returns a random element from the specified set.
  - SMEMBERS: Returns the list of elements that make up the specified set.
  - SISMEMBER: Returns 1 if the queried element is in the set, 0 otherwise.

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

DATA TYPES: Sets

- Operators on sets:
  - SDIFF: Returns a set with the result of the difference of the first specified set and the following sets.
  - SDIFFSTORE: Stores in a specified set the result of the difference of the first specified set and the following sets. Returns an integer specifying the size of the resulting set.
  - SUNION: Returns a set with the result of the union of all the specified sets.
  - SUNIONSTORE: Stores the result of the union of all the specified sets in a specified set. Returns an integer specifying the size of the resulting set.

```
> 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
```

DATA TYPES: Sets

- Operators on sets:
  - SINTER: Returns a set with the result of the intersection of all the specified sets.
  - SINTERSTORE: Stores the result of the intersection of all the specified sets in a specified set. Returns an integer specifying the size of the resulting set.

```
> 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
```

DATA TYPES: Sets

- Other operators:
  - SREM: Removes one or more elements from a specified set. Returns the number of elements removed or 0 if no elements are removed or the set does not exist.
  - SCARD: Returns the cardinality of the specified set.
  - SMOVE: Moves an element from a specified source set to a specified destination set. Returns 1 if the element exists and is to be moved, 0 otherwise.

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

DATA TYPES: Sorted sets

- Sorted sets are sets in which the elements have an associated score by which the elements are ordered within the set.
- They have some similarity with hash tables except that in sorted sets the elements would be ordered by value. The key would be the score.
- Scores are floating point numbers.
- If two values have the same score, then they are sorted by lexicographical order of the element value.

DATA TYPES: Sorted sets

- Insertion and queries:
  - ZADD: Inserts one or more score/value elements into the specified sorted set. Returns the number of elements inserted. If any of the values already exist, the score is updated.
  - ZRANGE: Returns an ordered subset between the first and second indexes. The indexes can be negative, indicating that the value starts from the end. If the WITHSCORES argument is added, the scores associated with each element are returned.
  - ZREVRANGE: Returns a subset sorted in reverse order between the first index and the second. The indexes can be negative indicating that it starts from the end.

```
> 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
```

DATA TYPES: Sorted sets

- Advanced queries:
  - ZRANGEBYSCORE min max : Returns an ordered subset whose score is between the specified min and max . If accompanied by (the min or max will be exclusive, otherwise it will be inclusive by default. The comparison values can be – inf or + inf . If the WITHSCORES argument is added, the scores associated with each element are returned. An offset and limit of elements to return can also be specified with the LIMIT argument.
  - ZREVRANGEBYSCORE max min: Returns a subset sorted in reverse order whose score is between the specified min and max . Comparison values can be – inf or + inf .

```
> ZADD ss 1 ”a” 3 ”c” 2 ”b” 4 ”d”
( integer ) 4
> ZRANGEBYSCORE ss – inf (3 LIMIT 1 2
 1) “b”
```

DATA TYPES: Sorted sets

- Advanced queries:
  - ZRANGEBYLEX min max : For lists with elements with the same score, returns a subset ordered by value whose values are lexicographically between the specified min and max . Comparison values can be – or +. An offset and limit of elements to return can also be specified with the LIMIT argument. It is necessary to specify (or [in min and max to indicate whether it is inclusive or exclusive.
  - ZREVRANGEBYLEX max min: For lists with elements with the same score, returns a subset sorted by value whose values are lexicographically between the specified min and max . Comparison values can be – or +.

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

DATA TYPES: Sorted sets

- Operators on sets:
  - ZINTERSTORE: Stores in a specified sorted set the result of the intersection of all the specified sorted sets.
    - Returns an integer specifying the size of the resulting set.
    - sorted sets involved in the operation must be indicated .
    - The score of elements that exist in several sets is by default equal to the sum of their scores. Weighting can be done using the WEIGHTS argument. An operation other than summation can be specified with AGGREGATE, the possibilities being MIN and 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) to 2) 5
```

DATA TYPES: Sorted sets

- Operators on sets:
  - ZUNIONSTORE: Stores in a specified sorted set the result of the union of all the specified sorted sets.
    - Returns an integer specifying the size of the resulting set.
    - sorted sets involved in the operation must be indicated .
    - The score of elements that exist in several sets is by default equal to the sum of their scores. Weighting can be done using the WEIGHTS argument. An operation other than summation can be specified with AGGREGATE, the possibilities being MIN and 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
```

DATA TYPES: Sorted sets

- Delete items:
  - ZREM: Removes if any element(s) with specified value(s) from the sorted set. Returns the number of elements removed.
  - ZREMRANGEBYRANK min max : Removes elements between the min and max positions from a specified sorted set. Returns the number of elements removed.

```
> 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
```

DATA TYPES: Sorted sets

- Delete items:
  - ZREMRANGEBYSCORE min max : Removes items whose score is between the specified min and max .
  - ZREMRANGEBYLEX min max : For lists with elements with the same score, removes the elements lexicographically between the specified min and max .

```
> 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”
```

DATA TYPES: Sorted sets

- Other operators:
  - ZSCORE: Returns the score of a specified value if it exists in the set.
  - ZRANK: Returns the position of a specified value if it exists in the set.
  - ZREVRANK: Returns the position starting from the end of the sorted set of a specified value if it exists.
  - ZCARD: Returns the number of elements that make up the sorted set.
  - ZCOUNT min max : Returns the number of items whose scores are between min and max .
  - ZLEXCOUNT min max : In a sorted set with the same score for all elements, returns the number of elements whose values are between min and max .
  - ZINCRBY n: Increments the score by a value specified by n.

DATA TYPES: Bits

- Insertion and queries:
  - SETBIT: Inserts a bit at the specified position. Returns the value of the bit before it was modified.
  - GETBIT: Gets the bit at the specified position.
  - BITOP: Performs the specified operation (AND, OR, XOR and NOT) on the specified keys and stores them in another key.
  - BITCOUNT: Counts the number of active bits between the specified positions.
  - BITPOS b: Returns the position of the first bit equal to b among the specified positions.

```
>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
```

DATA TYPES: Iterators

- Iterators :
  - One of the arguments they take is the cursor ID. If this is equal to 0, a new cursor is created.
  - The iterator returns an array of two elements. The first is the cursor to be called to continue iterating. The second is the list of elements iterated over.
  - You can specify how many elements to iterate over with the COUNT argument.
  - The set of elements to iterate over can be restricted with the MATCH argument.

DATA TYPES: Iterators

- Iterators :
  - SCAN: Iterates over the keys stored in the current database.
  - SSCAN: Iterates over the elements stored in a set.
  - HSCAN: Iterates over the elements stored in a hash table.
  - ZSCAN: Iterates over the elements stored in a sorted set.

```
> 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

- In Redis, it is not necessary to wait for the response of the operations performed. You can send **chains of operations** and wait later to get the response.

> SET to 1 > SET to 1

OK OK

> INCR to > INCR to

(integer) 2 > INCR a

> INCR to > INCR to

(integer) 3 (integer) 2

> INCR a (integer) 3

(integer) 4 (integer) 4

TRANSACTIONS

- Allows you to execute a set of commands in a single step.
  - Commands are **serialized** and executed **sequentially** as a single, **atomic operation** .
  - **All** commands are executed, otherwise **no** commands are executed.
  - There is no possibility of **rollback** . Once a transaction is executed there is no possibility of failure in execution.
    - **Warning** : There may be bugs, but they should have been caught in development.

TRANSACTIONS: Transaction example

- The transaction is started with the MULTI command.

```
> MULTI
OK
```

- The desired operations are queued

```
> INCR foo
QUEUED
> INCREASE bar
QUEUED
```

- They are executed

```
> EXEC
1) ( integer ) 1
2) ( integer ) 2
```

- Or they are discarded

```
> DISCARD
OK
```

TRANSACTIONS: Watch

- Allows you to control other clients' access to certain variables during a transaction.
  - **Compare & Swap** Method : Checks if there have been any changes to the variables since the lock was placed, and if not, the operation is performed.
  - This operation is called **optimistic blocking .** because no other client is expected to access those variables.
  - If another client accesses the transaction, it will be discarded and we will have to repeat it.

TRANSACTIONS: Example with Watch

- Hypothetical situation in which the INCR operator does not exist

```
> WATCH mykey
> val = GET mykey
> val = val + 1
> MULTI
> SET mykey $val
> EXEC
```

PUBLICATION AND SUBSCRIPTION

- Redis allows **subscription** and **publication** of messages by channels
  - SUBSCRIBE: Subscribe to specified channels.
  - PSUSCRIBE: Subscribe to channels that match the specified patterns.
  - PUBLISH: Send a message to the specified channel.
  - UNSUSCRIBE and PUNSUSCRIBE: Unsubscribe from the specified channels.

> PSUBSCRIBE chanel .* > PUBLISH chanel.1 hi!

1) psubscribe (integer) 1 2) chanel .*

3) (integer) 1

4) pmessage

5) chanel .*

6) chanel.1

7) Hi!

Cache

- Redis allows you to implement different Cache modalities.
  - When the memory limit is reached, one of the stored elements is **evicted** .
  - The memory limit is defined with **config set** :

```
config set maxmemory 100mb
```

  - Redis provides various eviction methods (**Eviction Policies**) :
    - **noeviction** : When the limit is reached, an error occurs. (use DEL)
    - **allkeys-lru** : Least Recently Used pure.
    - **volatile-lru** : Pure LRU on expiring items.
    - **allkeys-random** : Random over all elements.
    - **volatile-random** : Random on items that expire.
    - **volatile-ttl** : Remove items that expire early.