**Title:** Redis SMEMBERS

**Excerpt:** The ‘SMEMBERS’ command operates on Redis sets. Redis set is a collection of unordered strings stored at a given key. The set contains unique values and duplicates are not permitted. We can use the ‘SMEMBERS’ command to return all the available members within a set at a specified key. In this article, we will be covering all the aspects related to the ‘SMEMBERS’ command.

**Permalink:** redis-smembers

**Category:** Redis

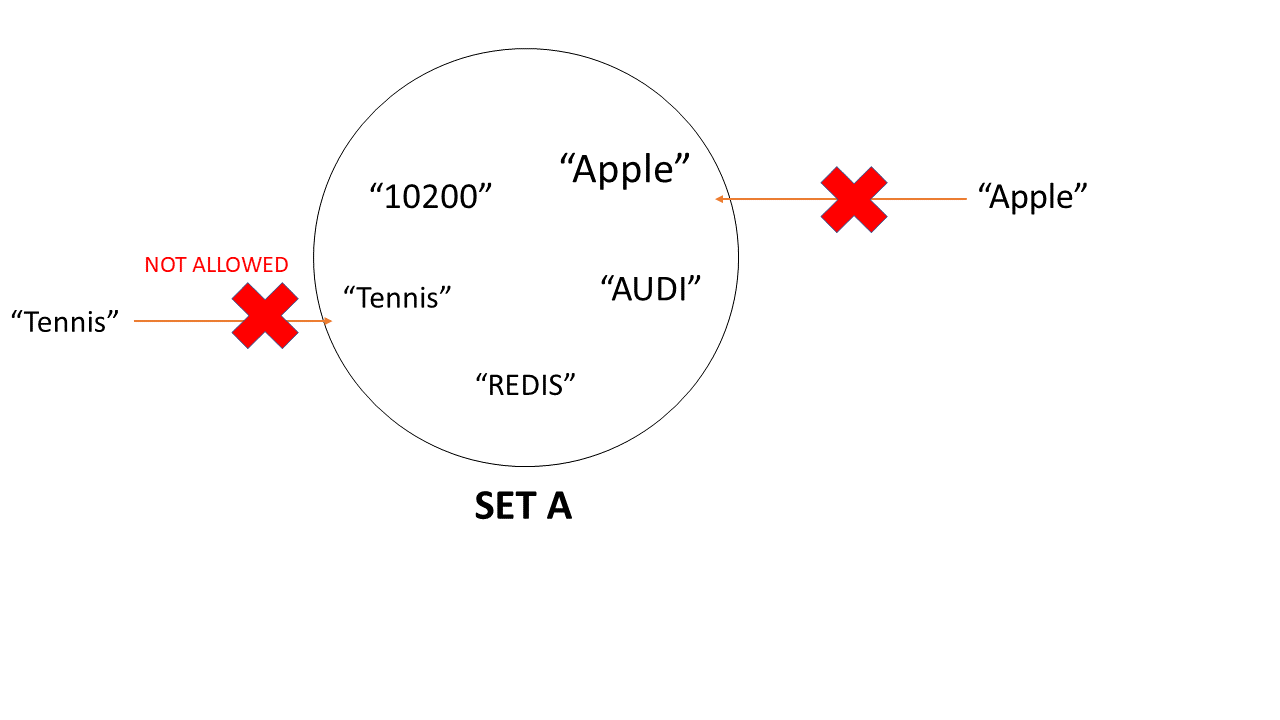
# 

# 

# 

# Redis sets

Redis supports several data types and data structures. The Set is one of the most beneficial among them. It allows you to store a collection of strings in unordered form. One of the handy features associated with sets is that no repeating elements are permitted.



A set can store more than four billion unique members. Major set operations such as add and remove are having constant time complexity which is very efficient. When you are trying to add an existing element to a set, it will keep only one copy of that element without creating a new element. Hence, no need to add explicit duplicate checks.

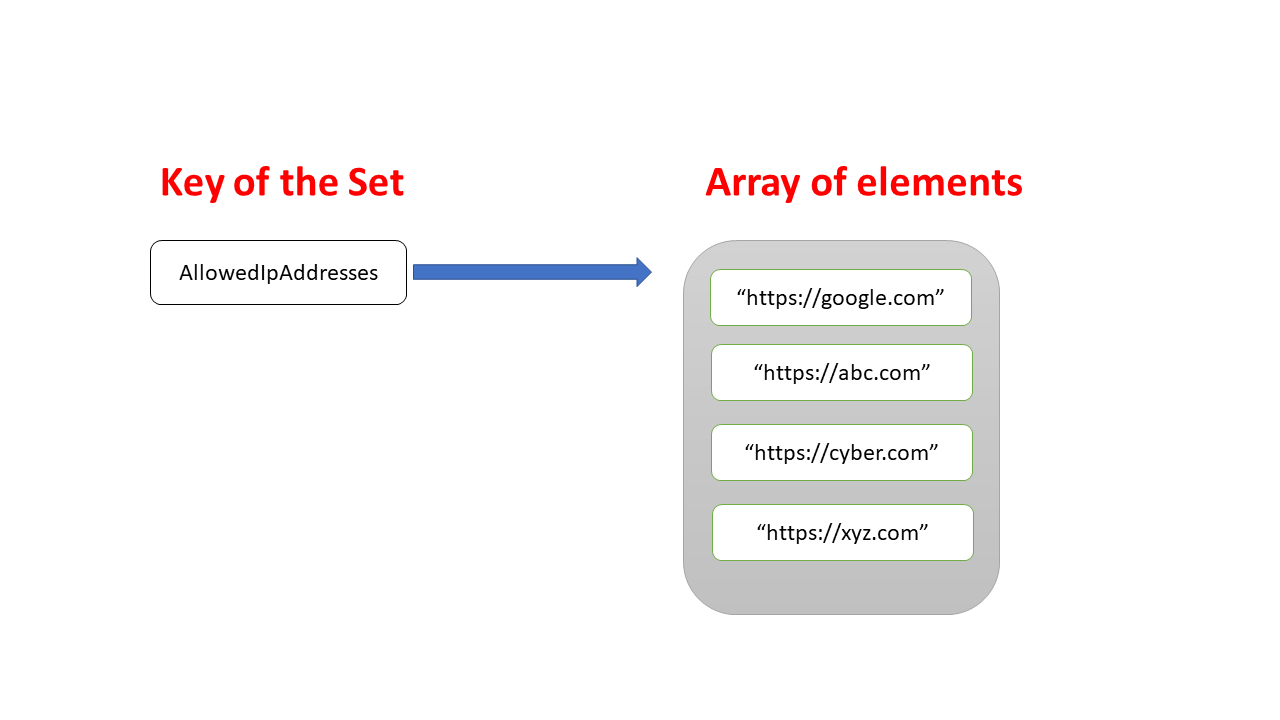
# The ‘SMEMBERS’ command

There are several commands available to operate on Redis sets. It is always useful to fetch all the elements stored in a set. Usually, the ‘SMEMBERS’ command is used to retrieve elements in a Redis set.

## Syntax

| SMEMBERS <**set\_key**> |
| --- |

Every set has a key that acts as the reference for the array of elements.



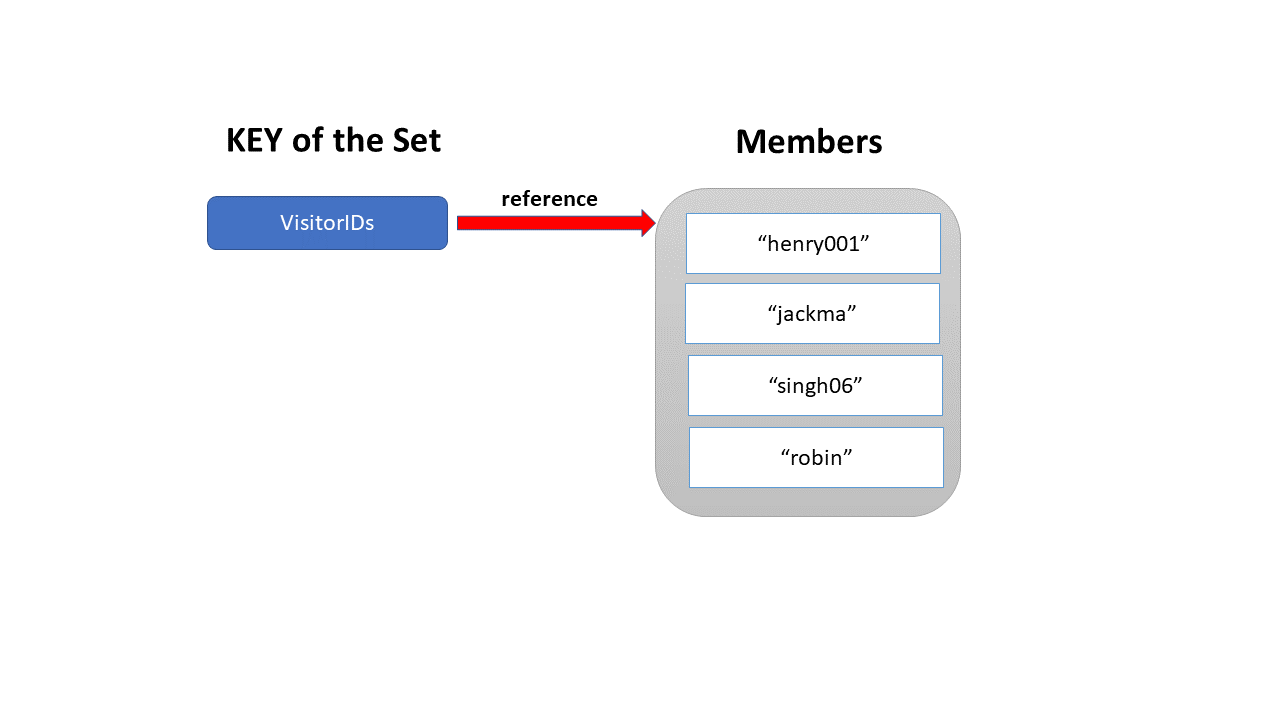
## Example - Count page views from unique users

Let’s take an example where your website needs to count the views coming from unique users. Hence, you do not count on views coming from the same user id. We can use a set to keep all the unique users visiting your website.

We can use the ‘SADD’ command to create a new set at a specified key and to add some members.

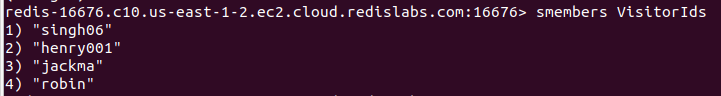
| **SADD** VisitorIds "henry001"  **SADD** VisitorIds "jackma"  **SADD** VisitorIds "singh06"  **SADD** VisitorIds "robin" |
| --- |

This would create a set at key ‘VisitorIds’ with four members as shown in the following.



Let’s execute the ‘SMEMBERS’ command to check the available members in the set identified by the key ‘VisitorIds’.

Output:

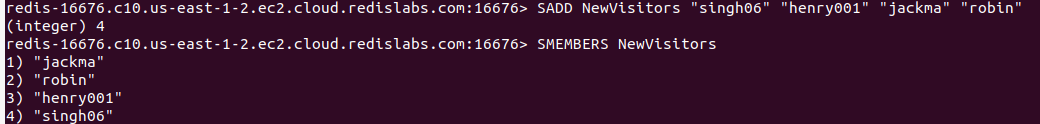


The returned array doesn’t contain any duplicate user IDs because the ‘SADD’ command didn’t allow duplicate members to go in. Hence you can identify the unique users who visited your website by the above command output.

We can use the ‘SADD’ command to add multiple members at once without executing the same command repeatedly.

| **SADD** NewVisitors "singh06" "henry001" "jackma" "robin" |
| --- |

Output:

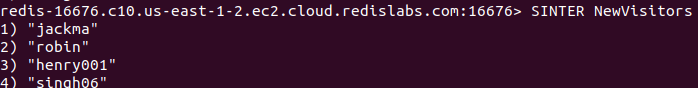


The same result is given by the ‘SMEMBERS’ command.

The ‘SINTER’ command can be used instead of the ‘SMEMBERS’ command. Usually, the ‘SINTER’ command returns the intersection of two sets. But you can use it by passing one key. It would return the members of the set stored at the passed key.

| SINTER NewVisitors |
| --- |

Output:



## The ‘SMEMBERS’ command performance

This command has linear time complexity that can be represented as O(N). It means, when the member count increases, the time it takes to return the array of elements will go up.

## The ‘SMEMBERs’ command on a non-set type

Usually, the ‘SMEMBERS’ command will only work with set data types. It will raise an error for other data types. Let’s create a ‘Hash’ as shown in the following.

| HSET myHash id 1 name funnyhash |
| --- |

We have created a hash called ‘myHash’. Next, we will pass this key to the ‘SMEMBERS’ command.

| SMEMBERS myHash |
| --- |

Output:



As expected, the command returns an error message by indicating that the key is not holding a set type value. It is a hash. Hence, the ‘SMEMBERS’ command is terminated with the above error message. So, this command is specific to set data types.

# Conclusion

Redis comes with several in-built data types. It supports sets. A set is an unordered collection of strings. It doesn’t store duplicate members. If we try to insert a duplicate element into a set, it will keep only a single copy of that member. Redis set operations are very efficient. The add and remove operations are having constant time complexity. We can use the ‘SMEMBERS’ command to return all the members stored in a set at a specified key. This operation has linear time complexity and is still very useful in real-world applications.