**Title:** Redis ZADD

**Excerpt:** Redis sorted sets are very fast in adding, updating, and deleting members. Hence, these can be used in low latency applications efficiently. The ZADD command is used to add members to a sorted set at a specified key. Several options can be used with the ZADD command which makes it very powerful.

**Permalink:** redis-zadd

**Category:** Redis

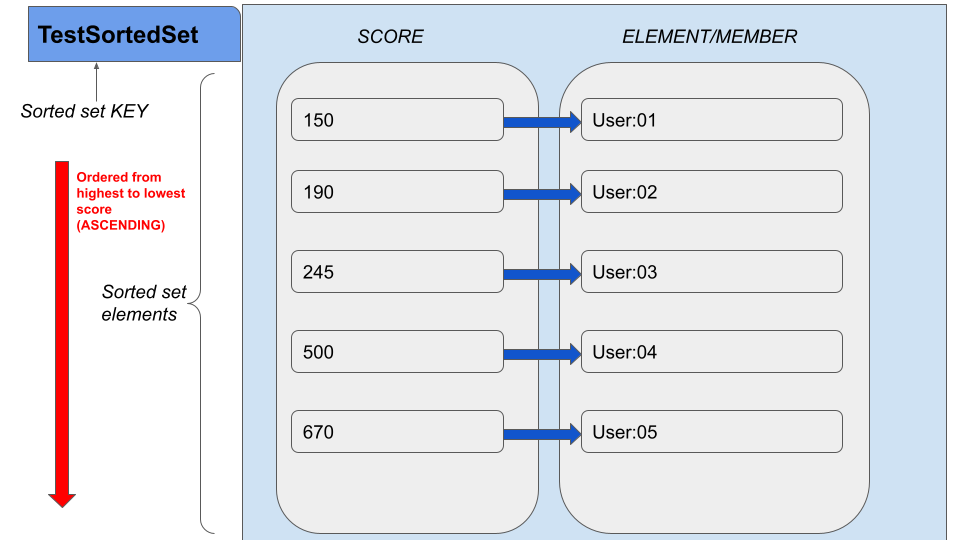
# 

# 

# 

# Introduction to Redis sorted sets

Redis sorted sets are a more advanced version of sets. A sorted set inherits all the properties from the Redis set data structure. In addition, they maintain the order of kept elements. Each member of a sorted set has a score that is used to order the set elements in ascending order.



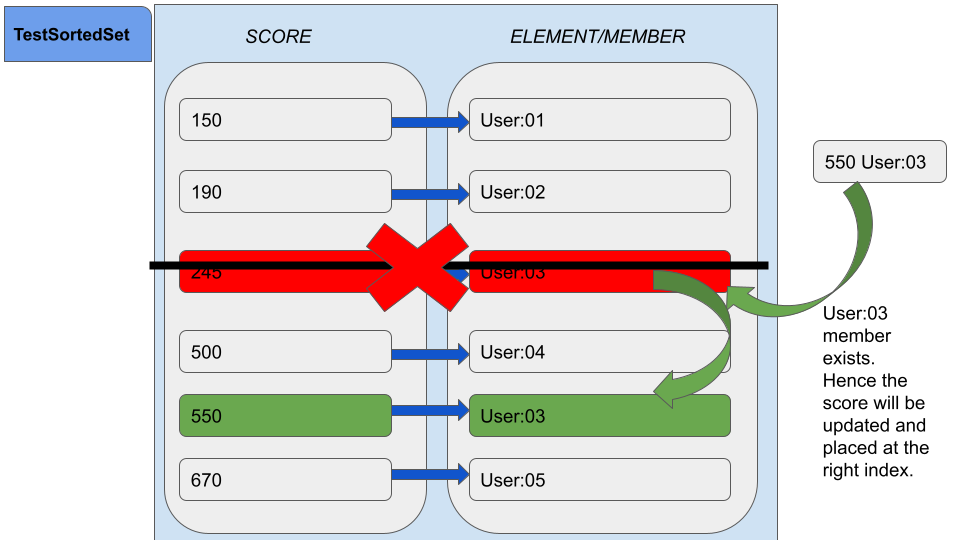
The sorted sets are very fast in adding, updating, and removing their members. It has logarithmic time complexity on all these operations. Since the members are ordered, accessing the middle element is very efficient as well. Hence, the sorted set would be ideal for implementing real-time applications such as online gaming leaderboards, low latency priority queues, and secondary indexes.

# The ZADD command

Several commands are available to operate on sorted sets. The *ZADD* command is used to add one or multiple members with scores to a sorted set stored at a given key. This command’s time complexity is proportional to the logarithm of the number of elements. Hence, it is much faster than most of the other Redis commands.

Whenever we add members using the ZADD command, there are some direct effects that take place in the sorted set.

* Since a sorted set holds a unique set of members, it doesn't allow adding already specified members into the sorted set. Instead, it will update the score of that particular member and place that element at the right index to maintain the correct order.



* If the sorted set key doesn’t exist, then the ZADD command will create the sorted set and add all the specified members.
* If the key exists but isn't supposed to hold a sorted set type value then it will raise an error.

## Syntax

| ZADD <sorted\_set\_key> [NX | XX] [GT | LT] [CH] [INCR] <score> <member> [score member …] |
| --- |

Usually, the ZADD command returns the number of members added to the sorted set. Hence, it ignores the score updates for the already existing members. This return value will change if the *CH* option is specified. Hence the count of all the changed members will be returned by the ZADD command. This count includes the sum of newly added and the score changed members.

## Example 01 - Online gaming leaderboard

Let’s assume a scenario where we need to manage a user leaderboard for an online adventure game that is played by thousands of users around the world. The nature of the game is that each user earns gold per successful completion of each mission. Redis sorted set would be the ideal data structure that we can use for this type of real-time low latency application.

We will be creating a sorted set identified by the key ***gameleaderboard****.* Furthermore, several gamers will be added as sorted set members with different scores. Each user's earned gold amount will be mapped as the sorted set score.

### Adding multiple members with ZADD

| **zadd** **gameleaderboard** 2300 **gamer**:1 1400 **gamer**:2 800 **gamer**:3 3500 **gamer**:4 4000 **gamer**:5 |
| --- |

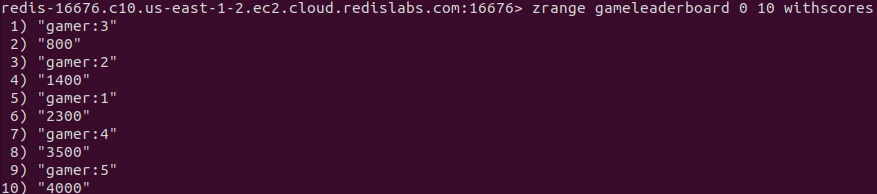
Output:



As expected, the return value is 5 that is the number of members added to the sorted set stored at key *gameleaderboard.*

Let’s check whether the sorted set contains all the members in an ordered manner. We can use the ZRANGE command to query all the members with their scores as shown in the following.

| zrange gameleaderboard 0 10 withscores |
| --- |



As mentioned, the *gameleaderboard* sorted set stores its members in ascending order based on their scores.

### Adding a new member with the same score as the existing member

Let’s try to add another user *gamer:6* with the gold amount of 3500. Redis sorted sets allow inserting members with the same score value. Hence this operation should successfully add the *gamer:6*.

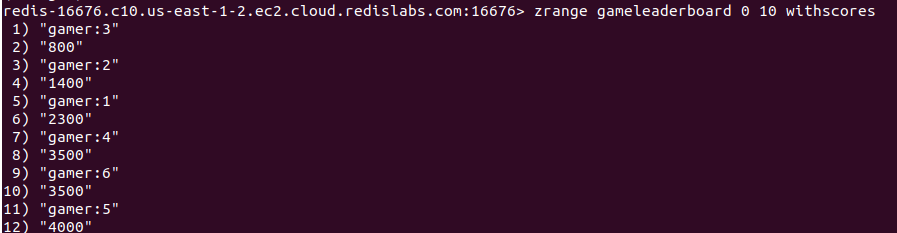
| **zadd** **gameleaderboard** 3500 **gamer**:6 |
| --- |

Output:



As expected, the return value is 1 which verifies the member has been added successfully.

Let’s inspect the sorted set members using the ZRANGE command again.



The member *gamer:6* has been inserted just after the *gamer:4.* Redis sorted sets use the lexicographical ordering if the score values are the same for the specified members. It compares member strings as an array of bytes and orders them accordingly.

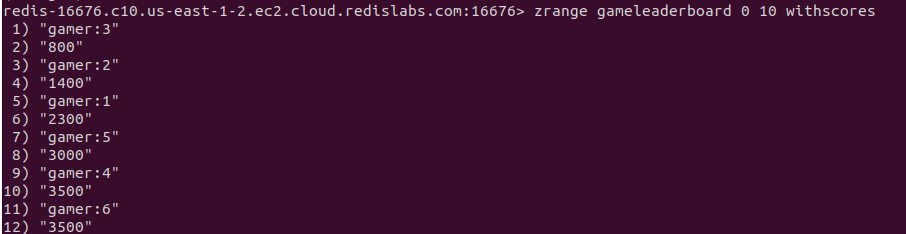
### Usage of NX and XX options with ZADD

Let’s assume that we only need to update the score of an existing member and not add any new members to the sorted set *gameleaderboard.* The ***XX*** option is used to achieve this.

| **zadd** **gameleaderboard** **xx** 3500 **gamer**:7 3000 **gamer**:5 |
| --- |



As expected the return value is 0 which means no new members added. We will be inspecting the sorted set again.

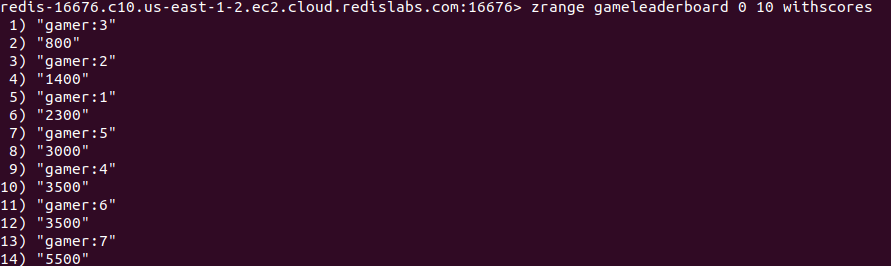


The *gamer:7* member has not been added to the sorted set, but the *gamer:5* member’s score has been modified and it is placed accordingly.

The ***NX*** option does the exact opposite of the ***XX.***

| **zadd** **gameleaderboard** **nx** 5500 **gamer**:7 4000 **gamer**:5 |
| --- |

Let’s inspect the sorted set again.



As stated above, the new member *gamer:7* has been added successfully. The *gamer:5* score value has not been modified.

### Usage of LT and GT options with ZADD

The LT and GT options are very useful when you need to update score values conditionally. Any of these two flags will not prevent adding new elements to the sorted set.

Whenever you specify the LT option with the ZADD command, it will modify the score value if and only if the new score is less than the current score of that element. The GT option will modify the score only if the new score is greater than the current score.

| **zadd** **gameleaderboard** **lt** 2100 **gamer**:1 1500 **gamer**:2 |
| --- |

Let’s inspect the *gameleaderboard* sorted set.



As you can see, the *gamer:1* member’s previous score was 2300. Hence the score has been changed with this operation and it has been changed to 2100. The *gamer:2* member score is not changed since its previous score was less than the new score.

### The CH option

Usually, the ZADD command returns the number of members added. With the CH option, it will return the sum of newly added and the existing members whose scores are modified.

| **zadd** **gameleaderboard** **ch** 2100 **gamer**:8 1500 **gamer**:2 3550 **gamer**:4 |
| --- |

Upon the execution of the above command, the *gamer:8* member has to be added. The *gamer:2* and *gamer:4* members' score values have to be modified. Hence, the sum of newly added and modified members is 3.



### Usage of INCR option in ZADD

The *INCR* option will increment the score of a member by the specified increment number. The ZADD command behaves exactly like ZINCRBY.

Let’s increment the *gamer:7* member’s score by another 100 as shown in the following

| **zadd** **gameleaderboard** **incr** 100 **gamer**:7 |
| --- |



As expected the previous score value has been incremented by 100. The new score is returned as 5600.

# Conclusion

Redis sorted set is a more advanced data structure that inherits all the properties from ordinary sets. The sorted sets are much faster than most of the Redis commands. Hence, the sorted sets are widely used in real-time low latency applications. The ZADD command is used to create a sorted set at a specified key with multiple members. The members are ordered based on their score values. Whenever the score values are the same for multiple members, ordering will be done using lexicographical ordering.