**Title:** Redis MRANGE

**Excerpt:** Redis ‘TS.MRANGE’ command is used to query data across multiple time series. It offers a wide range of features such as aggregation, filtering, and grouping. Its underlying data structure is a doubly-linked list which makes the command operations real fast.

**Permalink:** redis-mrange

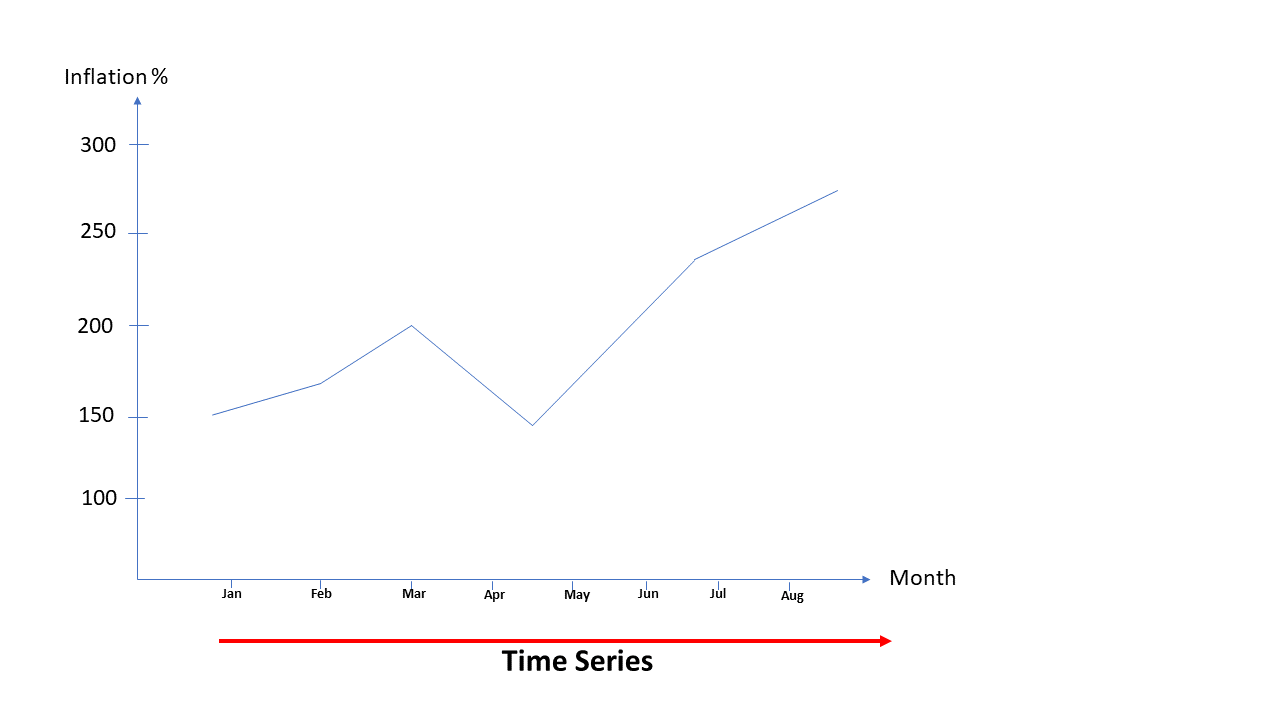
**Category:** Redis

# Time Series Data

Complex data structures use various indexing mechanisms. Most of them use numerical indexes. Time series data is another data structure that uses timestamp as its index. Several real-world scenarios use time-series data storage to store these types of data.

Example - Inflation rate over a while

Usually, the inflation rate variations for a given period are represented by the timestamp indexes as shown in the following. It varies with the time.



This is a very useful data representation methodology in applications such as stocks trading, weather data, CPU load over a period, etc.

# Redis and Time series

Redis has been offering time-series data storage with traditional sorted sets and streams. Both these data structures didn’t support advanced data manipulation queries. They were just able to retrieve the data based on a given upper and lower timestamp range. There was no support for aggregations, fetching data for different time ranges, and downsampling. With the arrival of Redis modules API, a time-series database has been introduced with some powerful features.

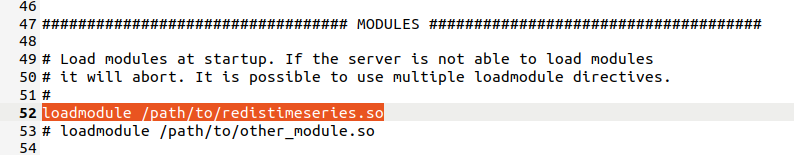
# Redis Modules API

Redis core comes with numerous features in-built. Still, the Redis functionality can be extended with external modules. These modules would add a new feature set and commands to the Redis core. These libraries can be loaded to Redis in two ways.

1. Using the redis.conf configuration file
2. Using the ‘MODULE LOAD’ command

1. The redis.conf configuration file can be used to load third-party libraries by modifying the following line in the Modules section.

| loadmodule /path/**to**/redistimeseries.**so** |
| --- |



The ‘redistimeseries.so’ module can be downloaded from the official Redis website. Also you need to restart the Redis server.

2. The ‘MODULE LOAD’ command can be used to plug a new module into the Redis server as well. This command can be used at runtime.

| MODULE **LOAD** /**path**/**to**/mymodule.so |
| --- |

Then you can list down the available module list as shown in the following.

| MODULE LIST |
| --- |

Output:



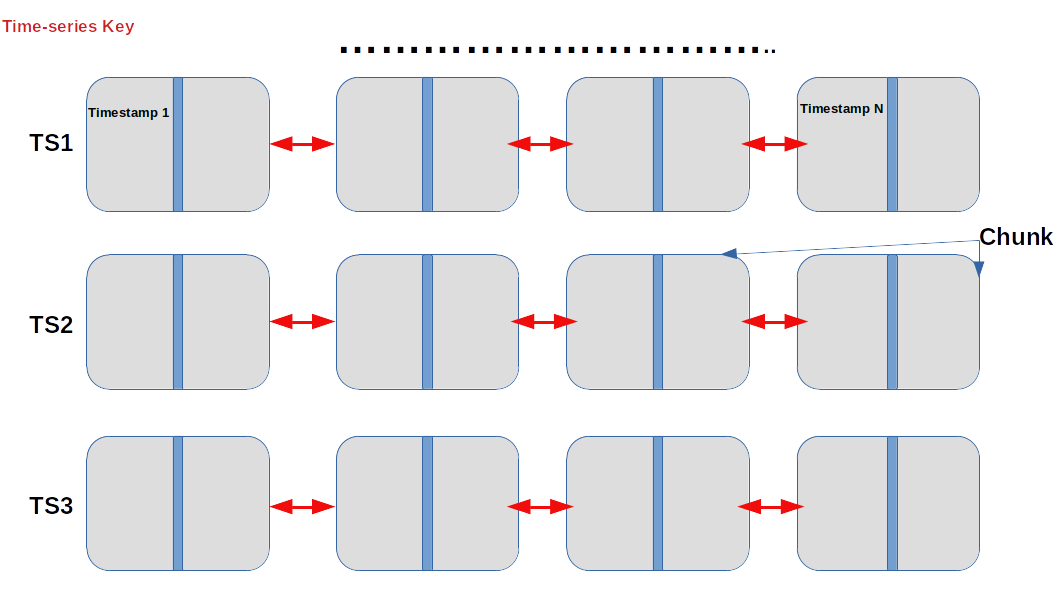
The ‘timeseries’ module is listed which means it has been plugged successfully.

## RedisTimeSeries Module

The ‘RedisTimeSeries’ module implements a time-series data store with a rich set of commands. It maintains a doubly-linked list at its core that is lightweight. Each node of the linked list consists of two corresponding arrays with the size of 128 bits. One array is to store the timestamps and the other one is for sample values. The timestamp and its value together are called a sample.

# The TS.MRANGE command

Several commands are available in the RedisTimeSeries library to manipulate time series data rows. The ‘TS.MRANGE’ is one of the most popular commands that is used to query a range of data values. It is capable of querying data values across multiple time series which is the advanced version of ‘TS.RANGE’ command. It also provides aggregation features such as AVG, FIRST, LAST, SUM, COUNT, etc.



The ‘TS.MRANGE’ command can query the range from timestamp 1 to timestamp N. At the same time, it can fetch data across all three time-series TS1, TS2, and TS3.

## Syntax

| **TS**.MRANGE **startingTimestamp** **endTimestamp**  [FILTER\_BY\_TS]  [FILTER\_BY\_VALUE]  [WITHLABELS | SELECTED\_LABELS label1 label2...] [COUNT rowCount] [AGGREGATION aggregator] [FILTER filter] [GROUPBY label] |
| --- |

The ‘startingTimestamp’ and ‘endTimestamp’ parameters specify the range of the data to be queried. These two parameters are mandatory. All the other parameters are optional to the command.

## 

## Example - Fetching stock market data

Let’s assume a real-world application where we need to store the hourly selling prices of heavily traded currency pairs. Redis time-series library can be used to store and manipulate these data efficiently.

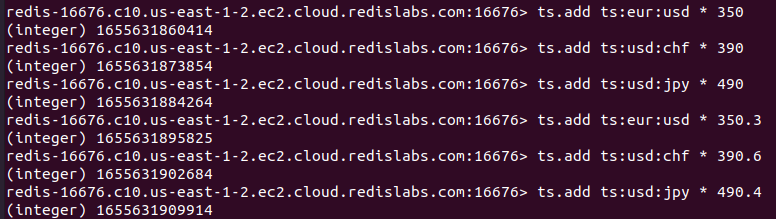
Let’s use the ‘TS.CREATE’ command to create time series per currency pair.

| **ts.create ts:eur:usd labels type "forex"** **ts.create ts:usd:chf labels type "crypto"** **ts.create ts:usd:jpy labels type "forex"** |
| --- |

We have added a label called type for each of the above time series’. Next, we should add sample data to the above time series. The ‘TS.ADD’ command can be used.

| **ts**.add **ts**:eur:usd \* 350 **ts**.add **ts**:usd:chf \* 390 **ts**.add **ts**:usd:jpy \* 490 **ts**.add **ts**:eur:usd \* 350.3 **ts**.add **ts**:usd:chf \* 390.6 **ts**.add **ts**:usd:jpy \* 490.4 |
| --- |

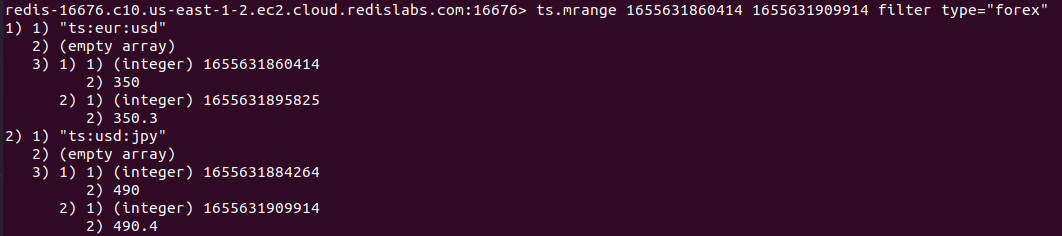
Output:



The ‘\*’ commands the Redis server to use the current server time as the timestamp. You could see the timestamp value has been returned by each command. Let’s use the ‘TS.MRANGE’ command to query the data across all time series’ where the type is 'forex' and the start and end timestamps are 1655631860414, 1655631909914 respectively.

| ts.mrange 1655631860414 1655631909914 filter type="forex" |
| --- |

Output:



As expected the ‘TS.MRANGE’ command queried data rows across both time series ts:eur:usd and ts:usd:jpy.

Let’s fetch the data rows where the type is anything in the ‘forex’ or ‘crypto’. The start and end timestamps are the same as in the previous scenario.

| ts.mrange 1655631860414 1655631909914 filter type=(forex,crypto) |
| --- |

Output:



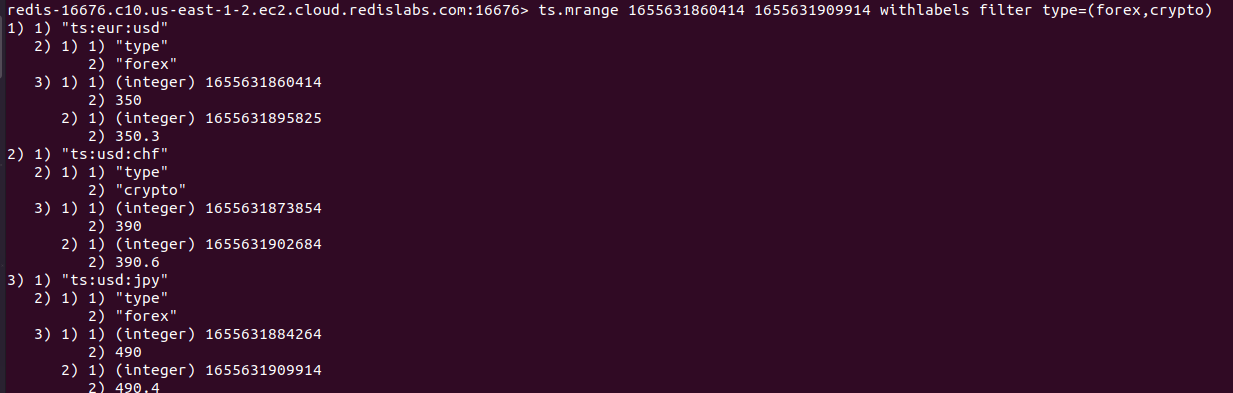
As you could see, the command has fetched data from all the time series that we've created previously.

### The ‘WITHLABELS’ flag

We can use the ‘WITHLABELS’ optional parameter to show the labels in the output.

| ts.mrange 1655631860414 1655631909914 withlabels filter type=(forex,crypto) |
| --- |

Output:



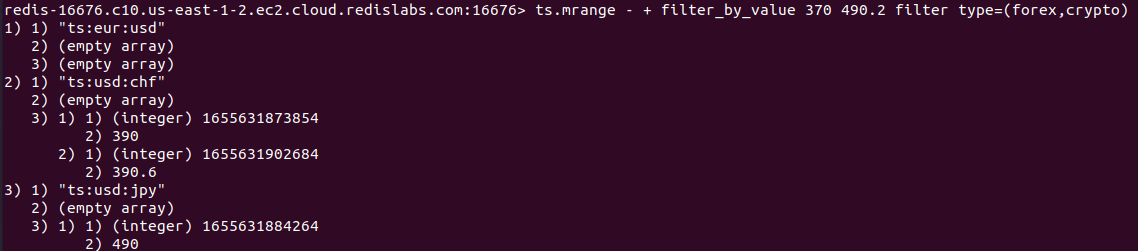
The ‘FILTER\_BY\_VALUE’ flag

This flag would filter the results based on the range of values specified. You can specify a minimum and maximum value for this flag.

| ts.mrange - + filter\_by\_value 370 490.2 filter type=(forex,crypto) |
| --- |

You can use the ‘-’ and ‘+’ symbols that indicate the minimum and maximum possible timestamps. In this example, the ‘FILTER\_BY\_VALUE’ parameter takes values 370 and 490.2 which commands the Redis server to query the time series data where the values lie within the specified range.

Output:



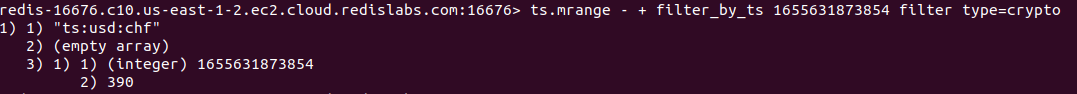
The ‘FILTER\_BY\_TS’ flag

You can filter the results by its exact timestamp value as shown in the following.

| ts.mrange - + filter\_by\_ts 1655631873854 filter type=crypto |
| --- |

In this case, we have specified the timestamp value 1655631873854.

This would filter out the resulting data as shown in the following.



# Conclusion

Redis offers time-series database capabilities through its external modules API. The ‘TimeSeriesModule’ can be plugged into the usual Redis store using a configuration file or runtime command. Time-series data can be stored within 128 bits of chunks. The ‘TS.MRANGE’ command is used to query data across multiple time series. It is the advanced version of the ‘TS.RANGE’. This command offers Several features such as aggregations, grouping, filtering, etc.