**Title:** Redis HSCAN

***Excerpt:*** The HSCAN command is used to retrieve elements of a hash without blocking the client or server. It has optional arguments such as MATCH and COUNT to filter out the returned elements based on a pattern match and the element count.

**Permalink:** redis-hscan

**Category:** Redis

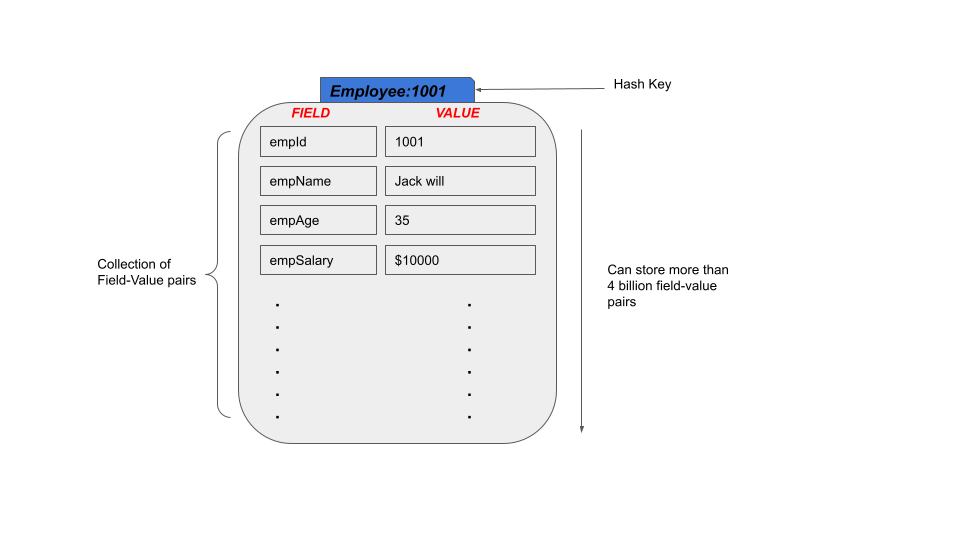
# 

# 

# 

# Redis Hashes

Redis hashes are a collection of string fields and string value pairs. It is an ideal candidate to represent real-world entities inside the memory. Hashes are very memory efficient where each can hold more than four billion field-value pairs.



Since Redis hashes allow you to store vast amounts of field-value pairs, iteration over this type of collection has to be facilitated as well. So, the cursor-based incremental iteration has been implemented with the HSCAN command. In this guide, we will be focusing more on the HSCAN command and its use cases.

# The HSCAN Command

The HSCAN command is used to iterate over fields and values of a Redis hash stored at a given key. It doesn’t block the server like KEYS or SMEMBERS commands. Instead, the HSCAN returns a portion of elements in each iteration with O(1) constant time complexity. The only downside of this behavior is that the HSCAN command has a limited guarantee of the returned hash elements at a given time because the hash can be changed during the iterations. It is not the case with blocking commands like SMEMBERS which guarantee that all the elements will be returned at a given time.

## Syntax

The HSCAN command derives from the SCAN command. Hence the HSCAN command has almost the same syntax as SCAN except that the SCAN command doesn’t require a *key* as the first argument. Because it iterates over the keys stored in the current Redis database.

| HSCAN *hash\_key* *cursor* [MATCH *pattern*] [COUNT *count\_per\_iteration*] |
| --- |

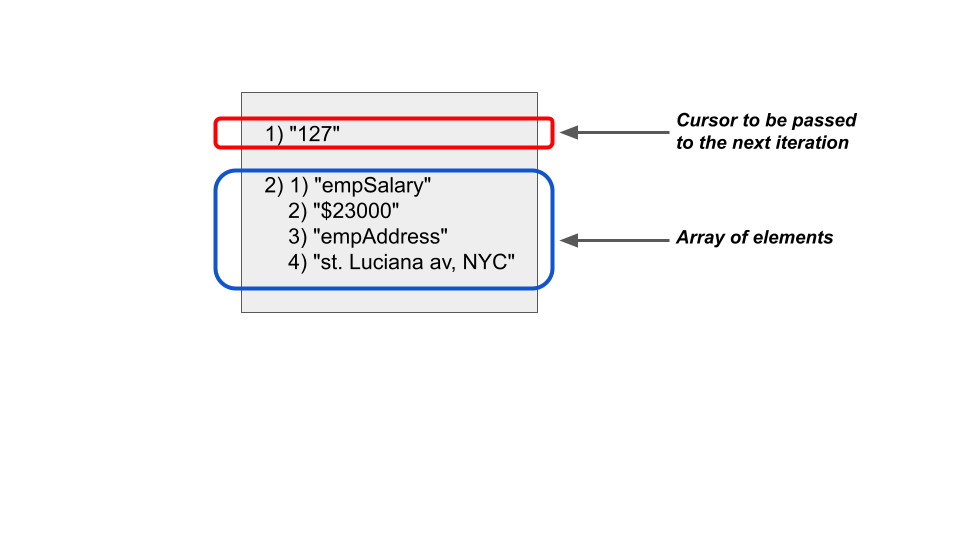
*Hash\_key*:The key of the hash.

*Cursor*: The cursor value which starts from 0 and ends by 0 if a full iteration has been finished.

MATCH: A pattern to match when retrieving elements in each iteration. Only the matched elements will be returned to the client.

COUNT: Approximate number of elements to be returned in each iteration.

The returned result set per iteration contains a couple of elements. The first one is a 64-bit unsigned integer that represents the cursor to be passed into the next call. Apart from that, an array of hash field-value pairs will be returned.



## Use Case 01 - Get All the Information of an Employee

Let’s assume that a company needs to maintain its employee details in a data store with limited memory consumption. So, the Redis hash is the ideal data structure to use in this case because we are going to represent real-world employee objects with their properties.

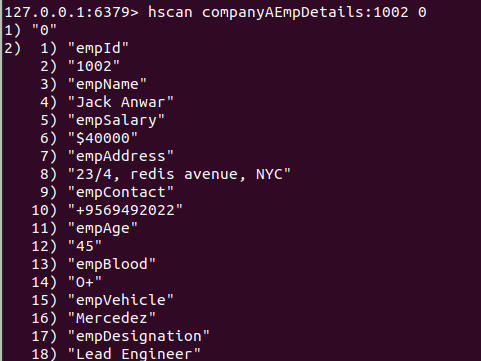
First we will create a Redis hash *companyAEmpDetails:1002* with some field-value pairs as follows. The HMSET command can be used to add multiple field-value pairs to a hash.

| HMSET companyAEmpDetails:1002 empId 1002 empName "Jack Anwar" empSalary $40000 empAddress "23/4, redis avenue, NYC" empContact +9569492022 empAge 45 empBlood O+ empVehicle Mercedez empDesignation "Lead Engineer" |
| --- |

Let’s iterate over the employee object and retrieve all the information using the HSCAN command as follows. We need to start from the cursor 0.

| hscan companyAEmpDetails:1002 0 |
| --- |

Output:



The cursor value is 0 in the returned result set which means all the elements have been returned by the first iteration. Because the number of elements is small, Redis hashes represent these elements using a single-allocation packed encoding. Therefore, until a maximum pack size or number of elements meet, the command will return all the elements in the hash. If the COUNT argument is specified, it will be ignored in that case. This is called a Full iteration. Because at the end of the first iteration, we received all nine elements. If we had tens of field-value pairs, it would be represented as a hash table in memory. So, it would take several iterations to return all the elements.

## Use Case 02 - Get Visitor Count for Web Sites Whose Top Level Domain is “.com”

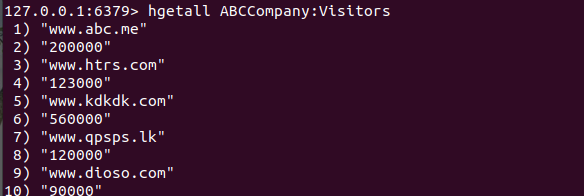
The HSCAN command can be used to filter out the returned elements based on a pattern match. A MATCH argument should be passed.

Let’s assume a company needs to track the visitor counts for their websites. We can create a hash ABCCompany:Visitors and add elements as follows.

| hmset ABCCompany:Visitors "www.abc.me" 200000 "www.htrs.com" 123000 "www.kdkdk.com" 560000 "www.qpsps.lk" 120000 "www.dioso.com" 90000 |
| --- |

We can use the HGETALL command to check how the hash looks like.

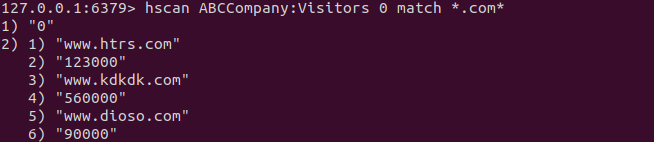
| hgetall ABCCompany:Visitors |
| --- |



Now the company has a requirement to iterate over this hash and retrieve the visitor counts for websites with the *“.com”* top-level domain. Let’s pass the MATCH argument to the HSCAN command as follows.

| hscan ABCCompany:Visitors 0 match \*.com\* |
| --- |

This should ideally return three members out of five members as shown in the following.



# Conclusion

In summary, the HSCAN command is used to iterate over a collection of hash field-value pairs. This command is derived from the parent SCAN command but with a slight change in the first argument to be passed where HSCAN takes the hash key as the first argument. As discussed in the use cases, the HSCAN command can be used in different ways by specifying the MATCH and COUNT arguments where you can retrieve elements that match a specific pattern and limit the returned element count per iteration. Overall, the HSCAN command can be useful when retrieving elements of a collection without blocking the server or client.