**COMP1835**

**Lab 3. Working with Redis database**

**Overview**

In this lab you will work with different data types in Redis data store.

**Lab 3.1 Ensure your Redis server is running.**

1. Your Redis server should be started in your VM 1.

If not, you can always start Redis server by using : $redis-server

2. Check if Redis is working, by opening Redis client:

$redis-cli

This will open a redis prompt.

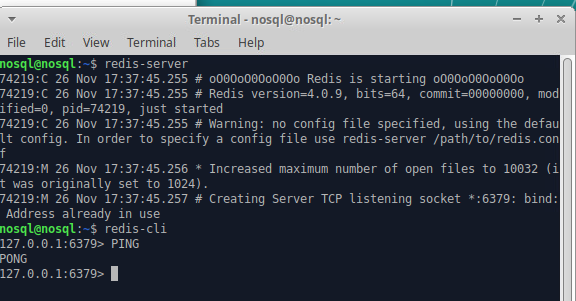
redis 127.0.0.1:6379>

3. In the above prompt, 127.0.0.1 is your machine's IP address and 6379 is the port on which Redis server is running.

Now type the following **PING** command.

**redis 127.0.0.1:6379> ping**

**PONG**



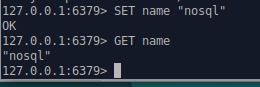
This shows that Redis is successfully installed on your machine.

**Lab 3.2 Working with strings.**

Redis string is a sequence of bytes. Strings in Redis are binary safe, meaning they have a known length not determined by any special terminating characters. Thus, you can store anything up to 512 megabytes in one string.

1. Create a key-value pair name=nosql

In the Terminal:



1. Create another key-value pair of your choice, where the value is a string.

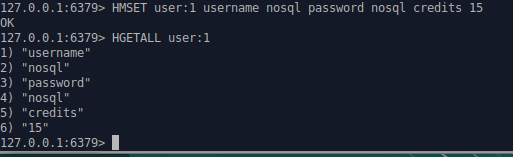
**Lab 3.3 Working with hashes.**

A Redis hash is a **collection of key value pairs**. Redis Hashes are maps between string fields and string values. Hence, they are used to represent objects.

1. Create a hash data type to store the user's object which contains basic information of the user. Here **HMSET, HGETALL** are commands for Redis, while **user − 1** is the key.

Every hash can store up to 232 - 1 field-value pairs (more than 4 billion).

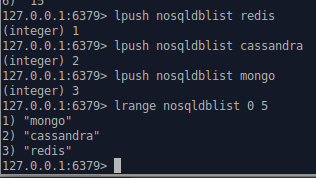




**Lab 3.4 Working with lists.**

Redis Lists are simply lists of strings, sorted by insertion order. You can add elements to a Redis List on the head or on the tail.

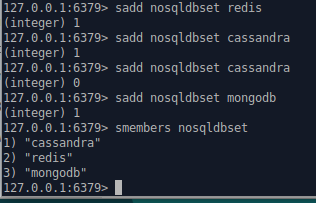
### Create a list called nosqldblist as shown below:



**Lab 3.5 Working with sets.**

Redis Sets are an unordered collection of strings. In Redis, you can add, remove, and test for the existence of members in O(1) time complexity.

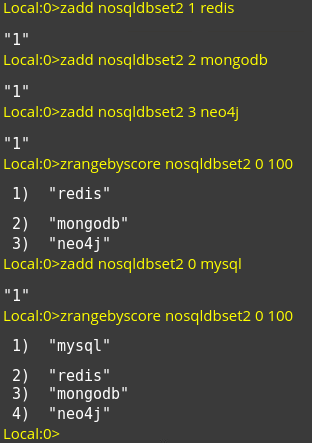
### Create a set called nosqldbset as shows below:



**Lab 3.6 Working with Sorted Sets.**

Redis Sorted Sets are similar to Redis Sets, non-repeating collections of Strings. The difference is, every member of a Sorted Set is associated with a score, that is used in order to take the sorted set ordered, from the smallest to the greatest score. While members are unique, the scores may be repeated.

### Create a sorted set called nosqldbset2 as shown below:



**Retrieving All Existing Keys**

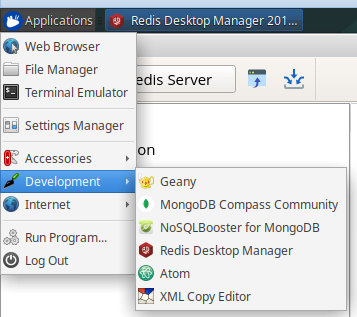
To get a list of all current keys that exist, simply use the **KEYS** command:

**127.0.0.1:6379> KEYS \***

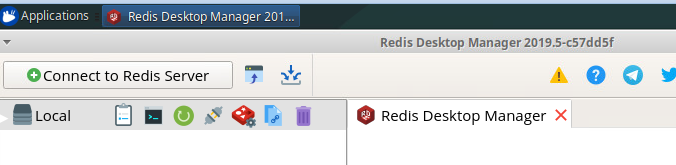
The syntax following **KEYS** can be used to search for specific words or phrases within the key, or the exact match as well. Here we want all keys that contain the text 'title':

**127.0.0.1:6379> KEYS \*nosql\***

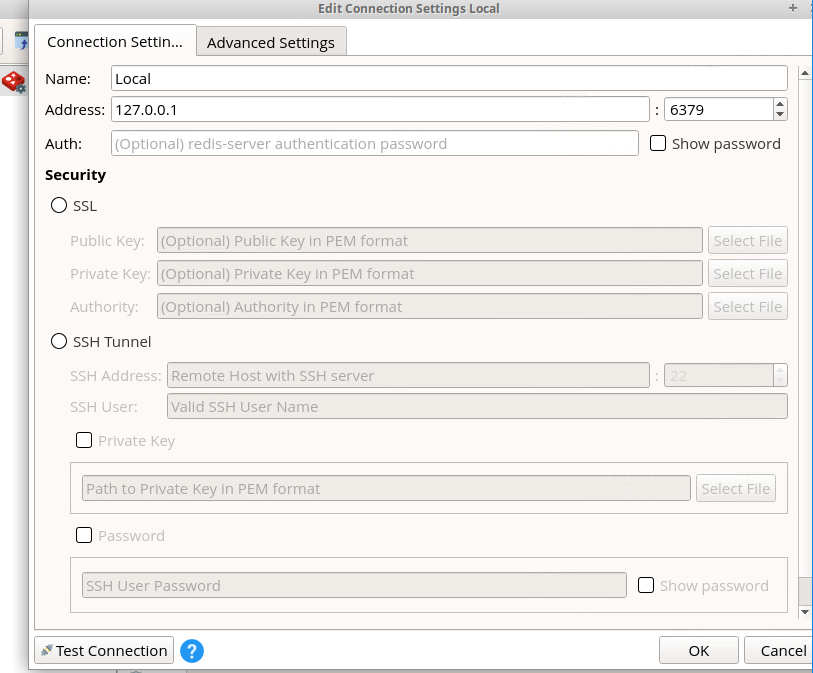
**Lab 5.7 Working Redis Desktop Manager.**



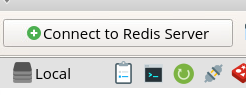
Press **Connect to Redis Server** button



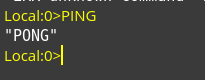
On the Connection page don’t change anything, just press Test Connection and then OK:



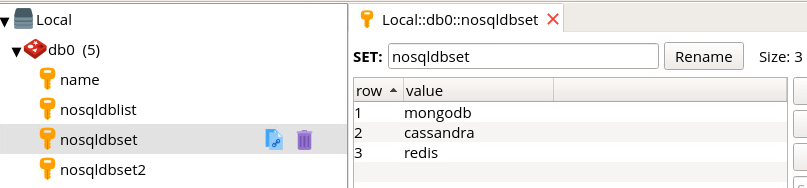
You can Open Console:

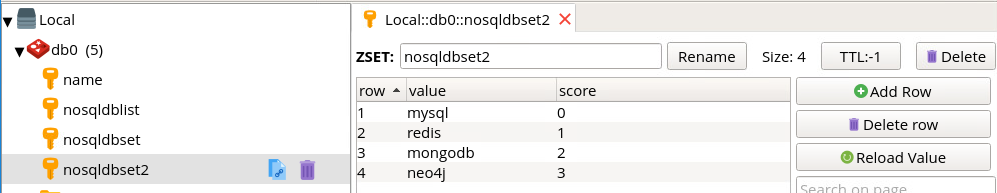


And work in the command line environment:



Or you can use a User Interface:





**Lab 3.7 Setting up Expiry**

A common use case for a key-value system like Redis is as a fast-access cache for data that’s more expensive to retrieve or compute. Expiration helps keep the total key set from growing unbounded, by tasking Redis to delete a key value after a certain time has passed. Marking a key for expiration requires the EXPIRE command, an existing key, and a time to live in seconds.

1. Set a key and set it to expire in ten seconds. We can check whether the key EXISTS within ten seconds and it returns a 1 (true). If we wait to execute, it will eventually return a 0 (false).

127.0.0.1:6379> SET ice "I'm melting…"

OK

127.0.0.1:6379> EXPIRE ice 10

(integer) 1

127.0.0.1:6379> EXISTS ice

(integer) 1

127.0.0.1:6379> EXISTS ice

(integer) 0

2. Setting and expiring keys is so common that Redis provides a shortcut command called SETEX.

127.0.0.1:6379> SETEX ice 10 "I'm melting…"

3. You can query the time a key has to live with TTL. Setting ice to expire as shown earlier and checking its TTL will return the number of seconds left.

127.0.0.1:6379> TTL ice

(integer) 4

At any moment before the key expires, you can remove the timeout by running command PERSIST key\_name .

127.0.0.1:6379> PERSIST ice

**Lab 3.8 Working with Database Namespaces**

So far, we’ve interacted only with a single namespace, but sometimes we need to separate keys by namespace. For example, if you wrote an internationalized key-value store, you could store different translated responses in different namespaces. The key greeting could be set to “guten tag” in a German namespace and “bonjour” in French. When a user selects their language, the application just pulls all values from the namespace assigned. In Redis nomenclature, a namespace is called a database and is keyed by number. So far, we’ve always interacted with the default namespace 0 (also known as database 0). Here we set greeting to the English hello.

127.0.0.1:6379> SET greeting hello

OK

127.0.0.1:6379> GET greeting "hello"

But if we switch to another database via the SELECT command, that key is unavailable.

127.0.0.1:6379> SELECT 1

OK

127.0.0.1:6379[1]> GET greeting

(nil)

And setting a value to this database’s namespace will not affect the value of the original.

127.0.0.1:6379[1]> SET greeting "guten tag"

OK

127.0.0.1:6379[1]> SELECT 0

OK

127.0.0.1:6379> GET greeting

"hello"

Since all databases are running in the same server instance, Redis lets us shuffle keys around with the MOVE command.

Here we move greeting to database 2:

127.0.0.1:6379> MOVE greeting 2

(integer) 2

127.0.0.1:6379> SELECT 2

OK

127.0.0.1:6379[2]> GET greeting

"hello"

This can be useful for running different applications against a single Redis server but still allow these multiple applications to trade data between each other.