IST769 Lab H

# Key-Value Model: Redis

### In this lab, we will explore Redis a key-value data structure database. We will learn the various Redis data structures and how to leverage the Redis has structure with common key structures to implement tabular structures which can be integrated with Apache Spark DataFrames.

### Learning Outcomes

At the end of this lab you should be able to:

* Apply the string, list, hash and sorted set Redis data structures and determine typical scenarios under which they should be used.
* Use Apache Spark to import, and export data from Redis Hashes with common keys.
* Build complex data-oriented solutions by combining Redis structures.

### Pre-Requisites

Before you begin:

* Open a terminal window in the lab environment
* Set the current working directory to **advanced-databases**
* Start the following services required by the lab:   
  **jupyter redis rediscommander retwis**

### Tools Used In this Lab

The following tools will be used in this lab:

1. To access Jupyter Lab from your Windows host:  
   [http://localhost:8888](http://localhost:8888/)   
   The password is **SU2orange!**
2. Access Redis Commander Admin UI:  
   [http://localhost:888](http://localhost:8888/)2
3. Retwis, the redis twitter clone.  
   [http://localhost:](http://localhost:/)5082
4. To access the Redis Cli:

PS> docker-compose exec redis redis-cli

# Lab Problem Set

**QUESTIONS:   
  
For each question, include a copy of the code required to complete the question along with a screenshot of the code and a screenshot of the output.**

Snapchat clone! Let’s use Redis to create a data model like Snapchat. Basically, users send messages to each other and once the message is accessed by the receiver it expires in 60 seconds. The rules:

1. Each **message** should be keyed by an id (you can use an integer and control the ID yourself)
2. Each message key should be namespaced, like so: **snap:msg:1** where **1** is the ID in this case.
3. Each **message** has 3 hash fields:
   1. **To**: username of the recipient e.g bob
   2. **From**: username of the sender e.g mary
   3. **Text**: the message itself.
4. When a user ***sends a*** ***message,*** perform these Redis commands:
   1. A new key is added to namespace **snap:msg:*id*** with the fields set in the hash..
   2. Add the ***id*** of the message to the user’s inbox key, queue, which is a list.   
      For example, mary’s inbox key is **snap:inbox:mary**
5. When a ***user reads a message,*** we:
   1. Remove it from the end of their inbox key list, a FIFO queue
   2. Set the message id key to expire in 60 seconds.

1. Using the Redis CLI, send these messages in the order they are listed with Redis commands. Make sure to perform both steps D.a and D.b as separate commands.

**To From Text**

Bob Art You owe me $50

Che Bob Hello there!!!

Che Dax Is this thing on?

Dax Art When is the meet-up?

Che Art What is Bob doing. OMG.

Bob Dax Who?!?!?

hset snap:msg:1 to "bob" from "art" text "You owe me $50"

lpush snap:inbox:bob 1

hset snap:msg:2 to "che" from "bob" text "Hello there!!!"

lpush snap:inbox:che 2

hset snap:msg:3 to "che" from "dax" text "Is this thing on?"

lpush snap:inbox:che 3

hset snap:msg:4 to "dax" from "art" text "When is the meet-up?"

lpush snap:inbox:dax 4

hset snap:msg:5 to "che" from "art" text "What is Bob doing. OMG."

lpush snap:inbox:che 5

hset snap:msg:6 to "bob" from "dax" text "Who?!?!?"

lpush snap:inbox:bob 6

keys snap:msg:\*

lrange snap:inbox:che 0 -1

hgetall snap:msg:5

hgetall snap:msg:4

2. Using the Redis CLI, read messages for the following users, in the order listed. Make sure to perform both steps E.a and E.b.

Bob

Che

Art

Bob

rpop snap:inbox:bob

expire snap:msg:1 60

rpop snap:inbox:che

expire snap:msg:2 60

rpop snap:inbox:art

rpop snap:inbox:bob

expire snap:msg:6 60

keys snap:msg:\*

1) "snap:msg:6"

2) "snap:msg:4"

3) "snap:msg:3"

4) "snap:msg:5"

keys snap:msg:\*

1) "snap:msg:6"

2) "snap:msg:4"

3) "snap:msg:3"

4) "snap:msg:5"

keys snap:msg:\*

1) "snap:msg:6"

2) "snap:msg:4"

3) "snap:msg:3"

4) "snap:msg:5"

keys snap:msg:\*

1) "snap:msg:4"

2) "snap:msg:3"

3) "snap:msg:5"

3. Provide a current state of the Redis database after Questions 1 and 2.

Display the current keys under the **snap:** namespace.

Display the messages which have not been read (and therefore have not expired)

Display the message ID’s in each users’ inbox.

hgetall snap:msg:5

hgetall snap:msg:4

hgetall snap:msg:3

keys snap:inbox:\*

lrange snap:inbox:che 0 -1

lrange snap:inbox:dax 0 -1

lrange snap:inbox:art 0 -1

lrange snap:inbox:bob 0 -1

The Department of Motor Vehicles has hired you to build a queue management system. You have decided the best system for this is Redis (a good choice, BTW). The system needs to manage a single queue of users, by username. Queued users can be served at one of 4 windows, A,B,C or D. The structure you build in Redis should support the queue and be able to display who is waiting in the queue. As people go to the window they should be removed from the queue and assigned to one of the 4 windows. You should be able to display who is at each window at any time.  
Namespace all keys with **dmv:**

**Example:**

Users In queue: Tom, Bill, Bart

Being Served at windows: A: Carl, B: Steve, C: Chuck, D: Dave

Event: When Dave is done at the window D, Bart is served next:

Users In queue: Tom, Bill

Being Served at windows: A: Carl, B: Steve, C: Chuck, D: Bart

Event: Mary arrives

Users In queue: Mary, Tom, Bill

Being Served at windows: A: Carl, B: Steve, C: Chuck, D: Bart

1. Its first thing in the morning and 8 people are waiting outside for the department to open! Add them to your queue: **amy, beth, chris, dee, erin, fran, greg, hela**Provide all the command required to accomplish this and a view of the queue

lpush dmv:queue amy

lpush dmv:queue beth

lpush dmv:queue chris

lpush dmv:queue dee erin fran greg hela

lrange dmv:queue 0 -1

1. The department is now open! Assign the first 4 people to windows A,B,C and D respectively. Oh, and Don’t forget to remove them from the Queue!  
   Provide all the steps required to accomplish these steps and a view of the queue and windows.

hset dmv:windows A empty B empty C empty D empty

hgetall dmv:windows

rpop dmv:queue 4

hset dmv:windows A amy B beth C chris D dee

lrange dmv:queue 0 -1

hgetall dmv:windows

1. Next, the following events occur:  
   a. iris arrives

lpush dmv:queue iris

lrange dmv:queue 0 -1

b. window C becomes available – move the next person from the queue to this window!

rpop dmv:queue

hset dmv:windows C erin

hgetall dmv:windows

c. window B becomes available – move the next person from the queue to this window!

rpop dmv:queue

hset dmv:windows B fran

hgetall dmv:windows

d. jake arrives

lpush dmv:queue jake

lrange dmv:queue 0 -1

e. window C becomes available – move the next person from the queue to this window!

rpop dmv:queue

hset dmv:windows C greg

hgetall dmv:windows

Provide all the steps required to accomplish these steps and a view of the queue and windows after the events.

1. Use spark to load the exam scores dataset `/home/jovyan/datasets/exam-scores/\*.csv` into Redis under the namesspace **examscores**. Use spark to Demonstrate the data is there by querying it back out.

import pyspark

from pyspark.sql import SparkSession

# REDIS CONFIGURATION

redis\_host = "redis"

redis\_port = "6379"

# Spark init

spark = SparkSession \

.builder \

.master("local") \

.appName('jupyter-pyspark') \

.config("spark.redis.host", redis\_host)\

.config("spark.redis.port", redis\_port)\

.config("spark.jars.packages","com.redislabs:spark-redis\_2.12:3.0.0")\

.getOrCreate()

sc = spark.sparkContext

sc.setLogLevel("ERROR")

examscores = spark.read.options(header = True, inferSchema = True) \

.csv("file:///home/jovyan/datasets/exam-scores/\*.csv")

examscores.printSchema()

examscores.toPandas().head()

examscores.createOrReplaceTempView("examscores")

query = '''

select

row\_number() over (order by Class\_Section) as rowid,

\*

from examscores

'''

examscores2 = spark.sql(query)

examscores2.printSchema()

keys examscores:\*

keys examscores:2

hgetall examscores:2

1. In Spark SQL, read the Redis **examscore** data into a temp view and get the min, max, and average exam score across all students. Write the data back out to Redis as **examscoresummary**, finally query the key in redis showing all values in the hash!

from pyspark.sql.functions import \*

examscores\_summary = examscores2.groupBy().agg(

min(col("Student\_Score")).alias("min\_score"),

max(col("Student\_Score")).alias("max\_score"),

avg(col("Student\_Score")).alias("avg\_score")

)

examscores\_summary.write.format("org.apache.spark.sql.redis")\

.mode("overwrite")\

.option("table", "exam\_summary")\

.save()

keys exam\_summary:\*

keys exam\_summary:b5476b81d0a743cb9c49e81b6dcdf45d

hgetall exam\_summary:b5476b81d0a743cb9c49e81b6dcdf45d

**IMPORTANT:** When you are finished with the lab, execute:

PS:> docker-compose stop

To turn off all running services, then shut down your Azure Lab instance.