IST769 Homework Problem Set 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 structure with common key structures to implement tabular structures that 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

# Prerequisites

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!**

1. Access Redis Commander admin UI: [http://localhost:8882](http://localhost:8888/)
2. Retwis, the Redis twitter clone: http://localhost:5082
3. To access the Cassandra shell:

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 three 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** E. When a ***user reads a message,*** we:
   3. Remove it from the end of their inbox key list, a FIFO queue.
   4. 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?!?!? |

A screenshot of a computer

Description automatically generated

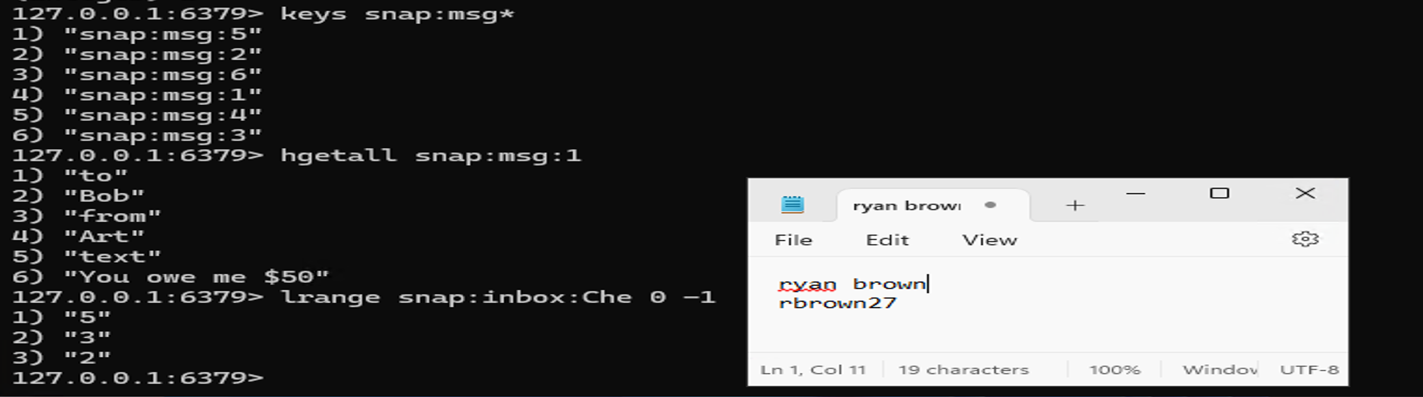
1. Using the Redis CLI, read messages for the following users, in the order listed. Make sure to perform both steps E.a and E.b.

Bob

Che

Art

Bob

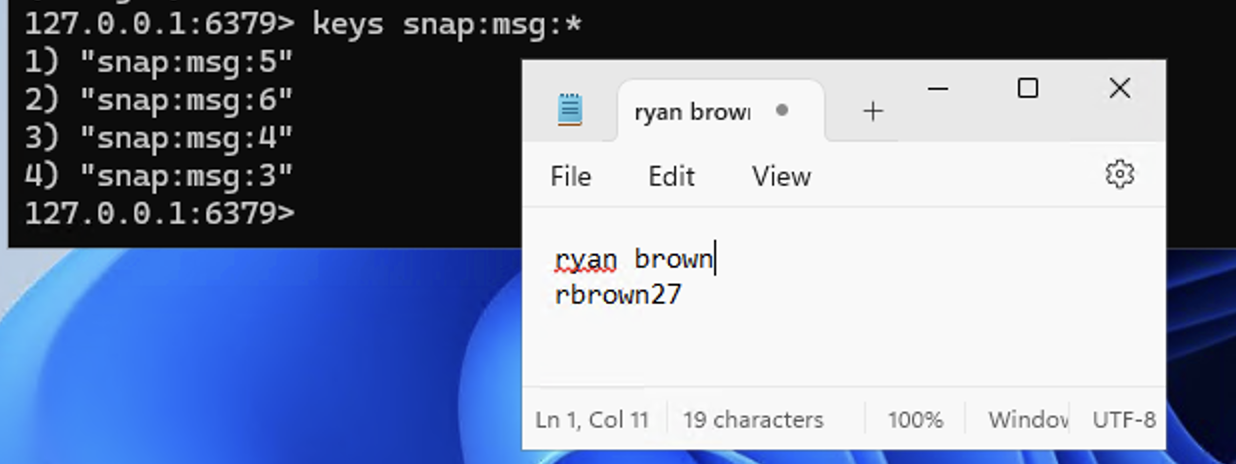


1. Provide a current state of the Redis database after Questions 1 and 2.

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

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

Display the message IDs in each users’ inbox.



A screenshot of a computer program

Description automatically generated

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 four 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 four 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. It’s first thing in the morning and eight people are waiting outside for the department to open! Add them to your queue: **Amy, Beth, Chris, Dee, Erin, Fran, Greg, Hela**

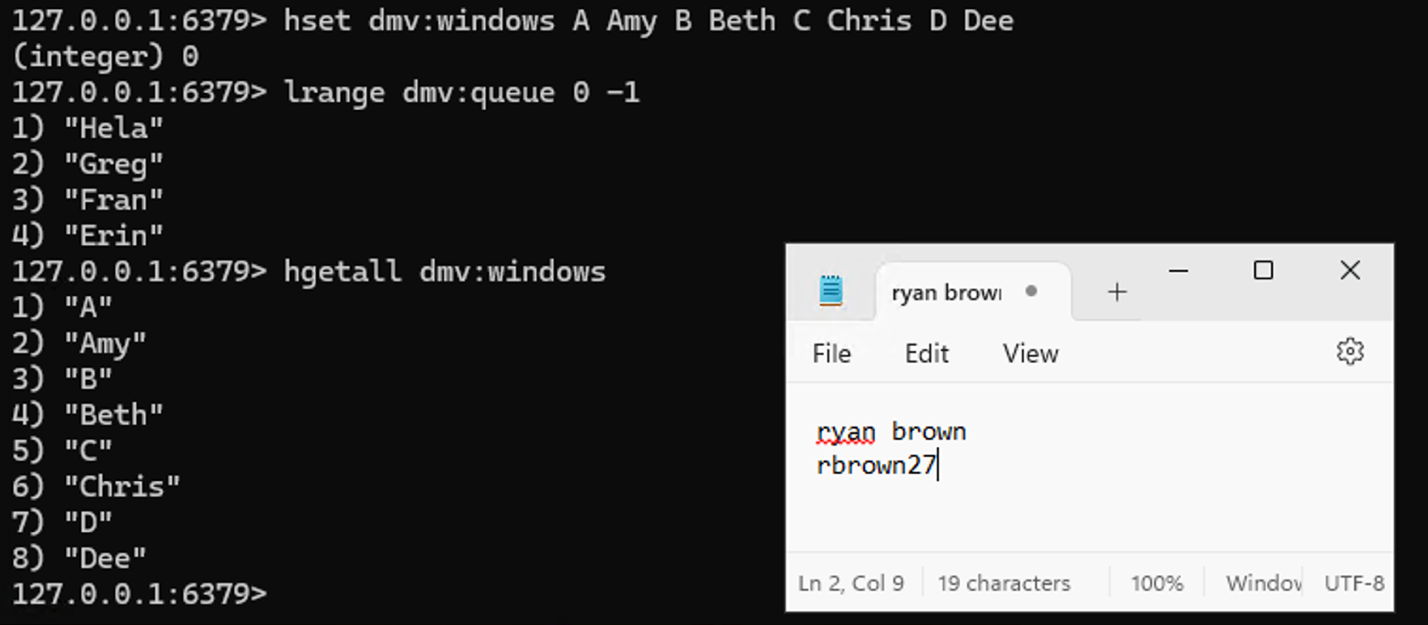
Provide all the commands required to accomplish this and a view of the queue.

A screenshot of a computer

Description automatically generated

1. The department is now open! Assign the first four 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.



1. Next, the following events occur:
   1. Iris arrives.
   2. Window C becomes available—move the next person from the queue to this window!
   3. Window B becomes available—move the next person from the queue to this window!
   4. Jake arrives.
   5. Window C becomes available—move the next person from the queue to this window!

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

A screenshot of a computer

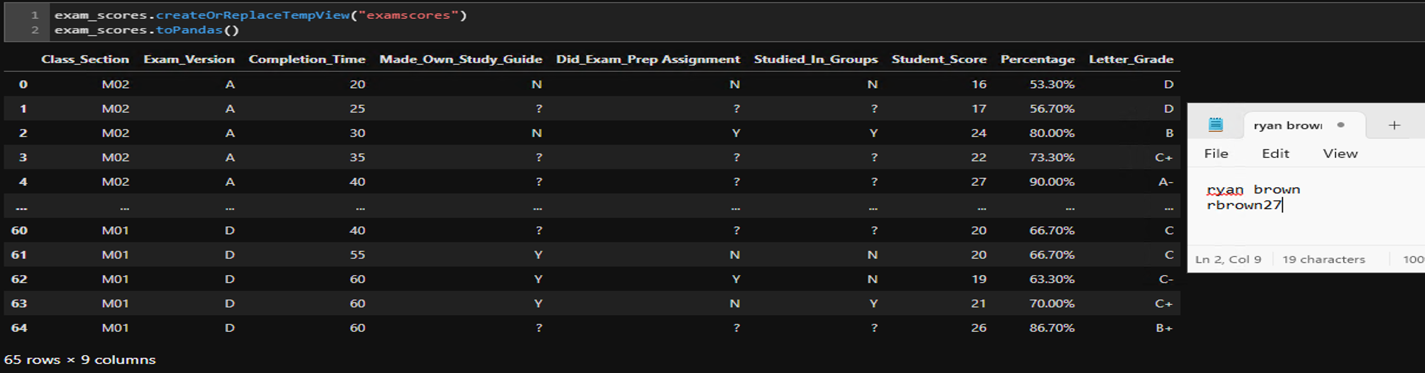
Description automatically generated

1. Use Spark to load the exam scores data set `/home/jovyan/datasets/exam-scores/\*.csv` into Redis under the namespace **examscores**. Use Spark to demonstrate the data is there by querying it back out.

A computer screen shot of a program

Description automatically generated

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!



A screenshot of a computer screen

Description automatically generated

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**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.