Problem Set H Submission Form

Overview

Your Name	Hendi Kushta
Your SU Email	hkushta@syr.edu

Instructions

Put your name and SU email at the top. Answer these questions all from the lab. When asked to include screenshots, please follow the screen shot guidelines from the first homework.

Remember as you complete the homework it is not only about getting it right / correct. We will discuss the answers in class so it's important to articulate anything you would like to contribute to the discussion in your answer:

- If you feel the question is vague, include any assumptions you've made.
- If you feel the answer requires interpretation or justification provide it.
- If you do not know the answer to the question, articulate what you tried and how you are stuck.
- Highlight any doubts or questions you would like me to review.

This how you receive credit for answering questions which might not be correct. In addition, you must complete the reflection portion of the homework assignment for full credit. Since most answers will be similar this is an important part of your individual submission.

Complete Part II of this document first, then go back and complete the Reflection in Part I.

Part I - Reflection

Use this section to reflect on your learning. To achieve the highest grade on the assignment you must be as descriptive and personal as possible with your reflection.

1. As you completed this assignment, identify what you learned..

   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

2. What barriers or challenges did you encounter while completing this assignment?

3. How prepared were you to complete this assignment? What can you do to be better prepared?

4. Rate your comfort level with this week's material. Use the rubric provided.

4 ==> I understand this material and can explain it to others.
3 ==> I understand this material.
2 ==> I somewhat understand the material but sometimes need guidance from others.
1 ==> I understand very little of this material and need extra help.

Part II – Questions

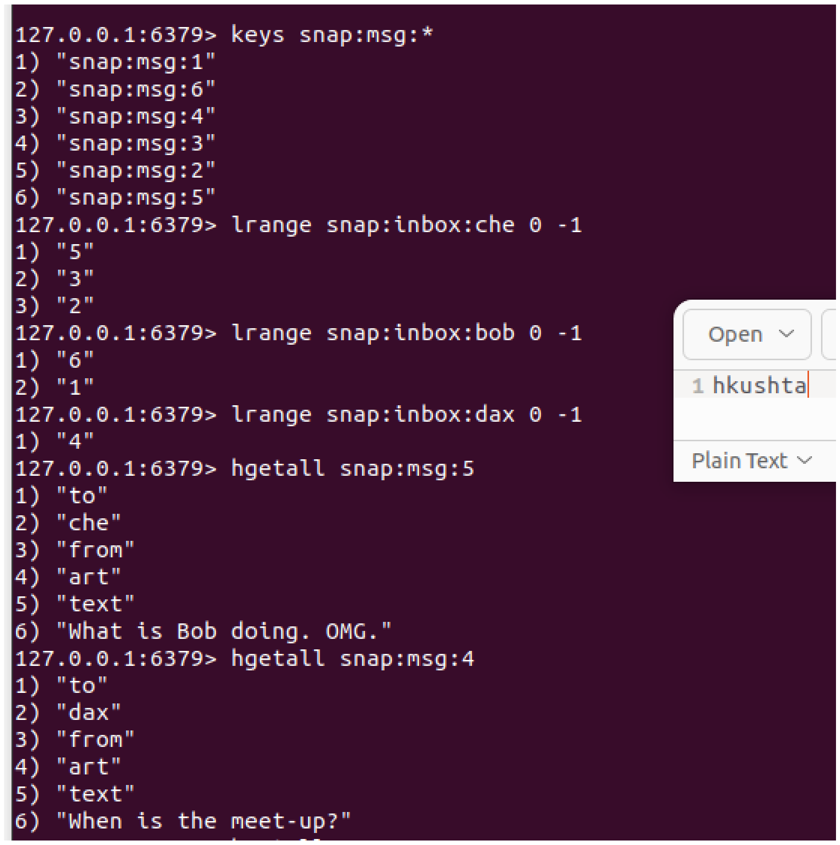
For each question, include a copy of the code required to complete the question along with a screenshot of the code and a screenshot of the output.

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:

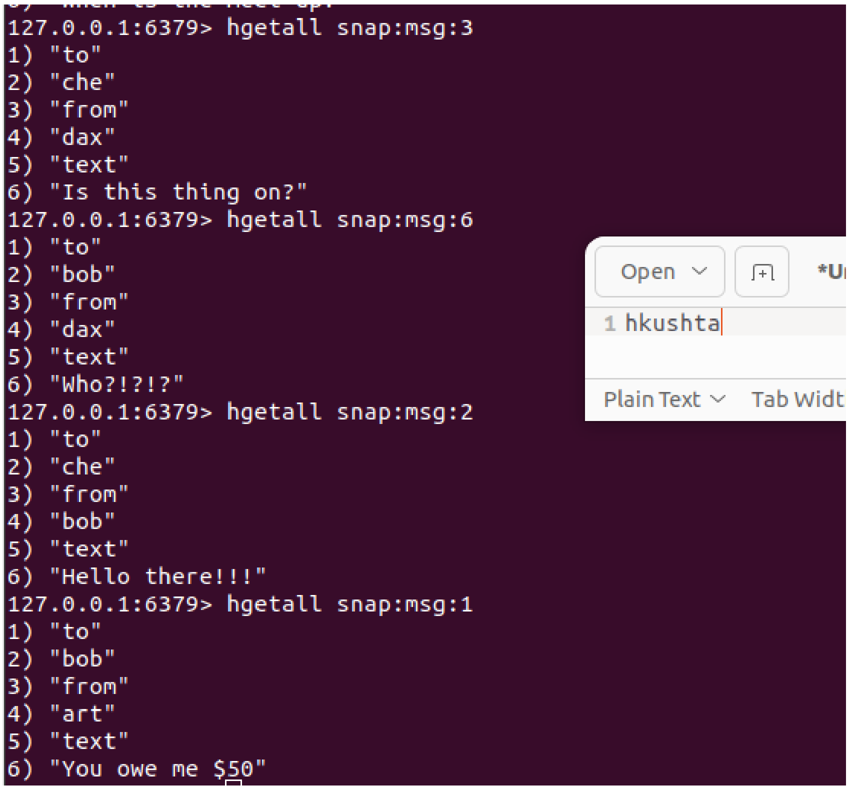
A. Each **message** should be keyed by an id (you can use an integer and control the ID yourself)
B. Each message key should be namespaced, like so: **`snap:msg:1`** where **1** is the ID in this case.
C. Each **message** has 3 hash fields:
    a. **To**: username of the recipient e.g bob
    b. **From**: username of the sender e.g mary
    c. **Text**: the message itself.
D. When a user ***sends a message,*** perform these Redis commands:
    a. A new key is added to namespace **snap:msg:*id*** with the fields set in the hash..
    b. 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:
    a. Remove it from the end of their inbox key list, a FIFO queue
    b. 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?!?!?



```
127.0.0.1:6379> keys snap:msg:*
1) "snap:msg:1"
2) "snap:msg:6"
3) "snap:msg:4"
4) "snap:msg:3"
5) "snap:msg:2"
6) "snap:msg:5"
127.0.0.1:6379> lrange snap:inbox:che 0 -1
1) "5"
2) "3"
3) "2"
127.0.0.1:6379> lrange snap:inbox:bob 0 -1
1) "6"
2) "1"
127.0.0.1:6379> lrange snap:inbox:dax 0 -1
1) "4"
127.0.0.1:6379> hgetall snap:msg:5
1) "to"
2) "che"
3) "from"
4) "art"
5) "text"
6) "What is Bob doing. OMG."
127.0.0.1:6379> hgetall snap:msg:4
1) "to"
2) "dax"
3) "from"
4) "art"
5) "text"
6) "When is the meet-up?"
```



```
127.0.0.1:6379> hgetall snap:msg:3
1) "to"
2) "che"
3) "from"
4) "dax"
5) "text"
6) "Is this thing on?"
127.0.0.1:6379> hgetall snap:msg:6
1) "to"
2) "bob"
3) "from"
4) "dax"
5) "text"
6) "Who?!?!?"
127.0.0.1:6379> hgetall snap:msg:2
1) "to"
2) "che"
3) "from"
4) "bob"
5) "text"
6) "Hello there!!!"
127.0.0.1:6379> hgetall snap:msg:1
1) "to"
2) "bob"
3) "from"
4) "art"
5) "text"
6) "You owe me $50"
```

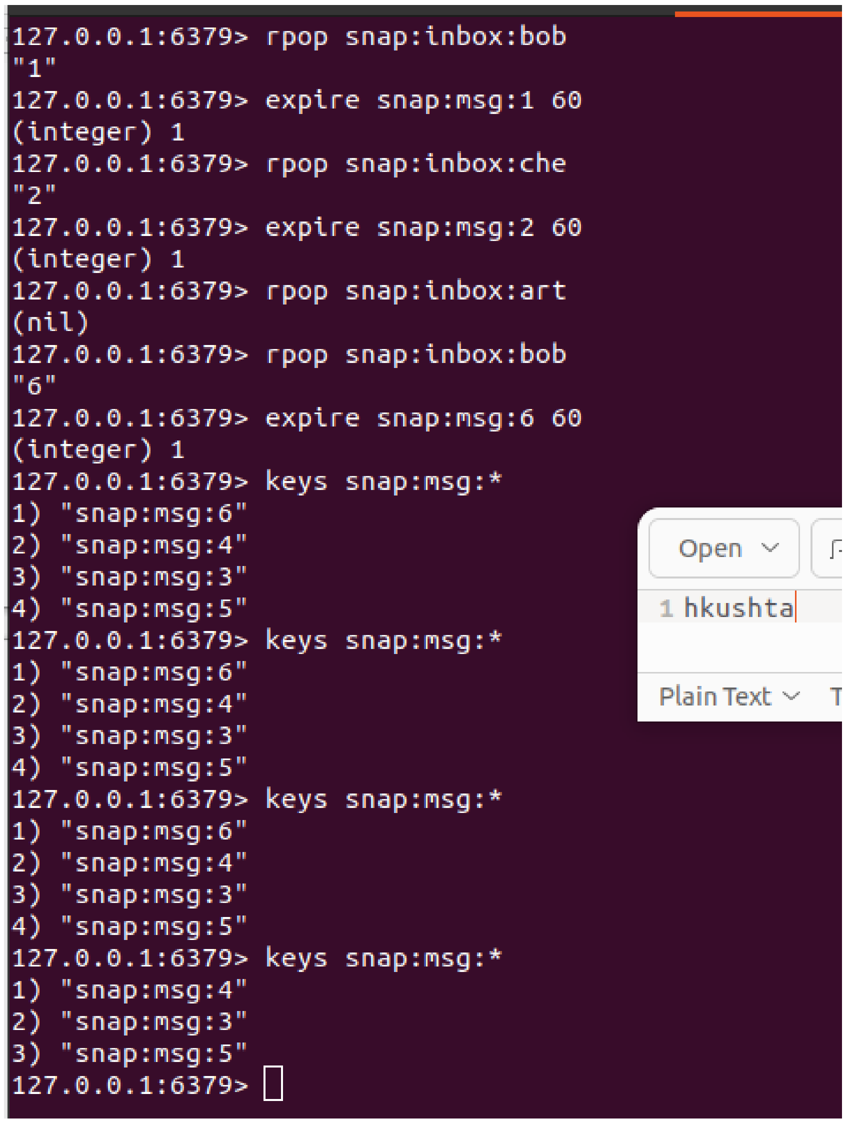
2. Using the Redis CLI, read messages for the following users, in the order listed. Make sure to perform both steps E.a and E.b.

Bob

Che

Art

Bob



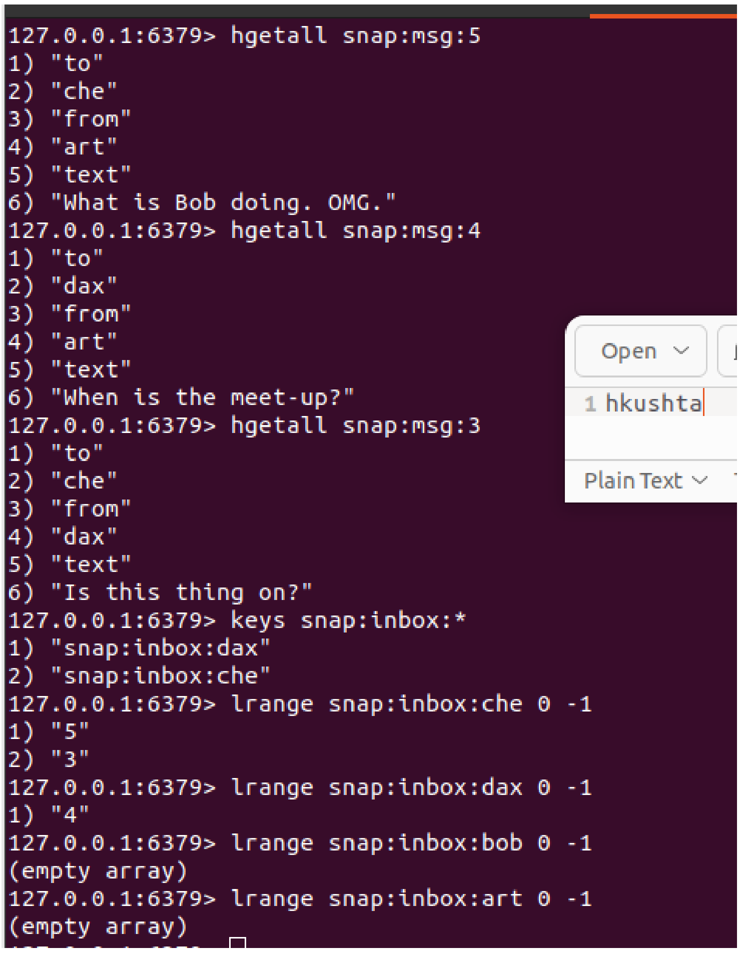
```
127.0.0.1:6379> rpop snap:inbox:bob
"1"
127.0.0.1:6379> expire snap:msg:1 60
(integer) 1
127.0.0.1:6379> rpop snap:inbox:che
"2"
127.0.0.1:6379> expire snap:msg:2 60
(integer) 1
127.0.0.1:6379> rpop snap:inbox:art
(nil)
127.0.0.1:6379> rpop snap:inbox:bob
"6"
127.0.0.1:6379> expire snap:msg:6 60
(integer) 1
127.0.0.1:6379> keys snap:msg:*
1) "snap:msg:6"
2) "snap:msg:4"
3) "snap:msg:3"
4) "snap:msg:5"
127.0.0.1:6379> keys snap:msg:*
1) "snap:msg:6"
2) "snap:msg:4"
3) "snap:msg:3"
4) "snap:msg:5"
127.0.0.1:6379> keys snap:msg:*
1) "snap:msg:6"
2) "snap:msg:4"
3) "snap:msg:3"
4) "snap:msg:5"
127.0.0.1:6379> keys snap:msg:*
1) "snap:msg:4"
2) "snap:msg:3"
3) "snap:msg:5"
127.0.0.1:6379>
```

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.



```
127.0.0.1:6379> hgetall snap:msg:5
1) "to"
2) "che"
3) "from"
4) "art"
5) "text"
6) "What is Bob doing. OMG."
127.0.0.1:6379> hgetall snap:msg:4
1) "to"
2) "dax"
3) "from"
4) "art"
5) "text"
6) "When is the meet-up?"
127.0.0.1:6379> hgetall snap:msg:3
1) "to"
2) "che"
3) "from"
4) "dax"
5) "text"
6) "Is this thing on?"
127.0.0.1:6379> keys snap:inbox:*
1) "snap:inbox:dax"
2) "snap:inbox:che"
127.0.0.1:6379> lrange snap:inbox:che 0 -1
1) "5"
2) "3"
127.0.0.1:6379> lrange snap:inbox:dax 0 -1
1) "4"
127.0.0.1:6379> lrange snap:inbox:bob 0 -1
(empty array)
127.0.0.1:6379> lrange snap:inbox:art 0 -1
(empty array)
```

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

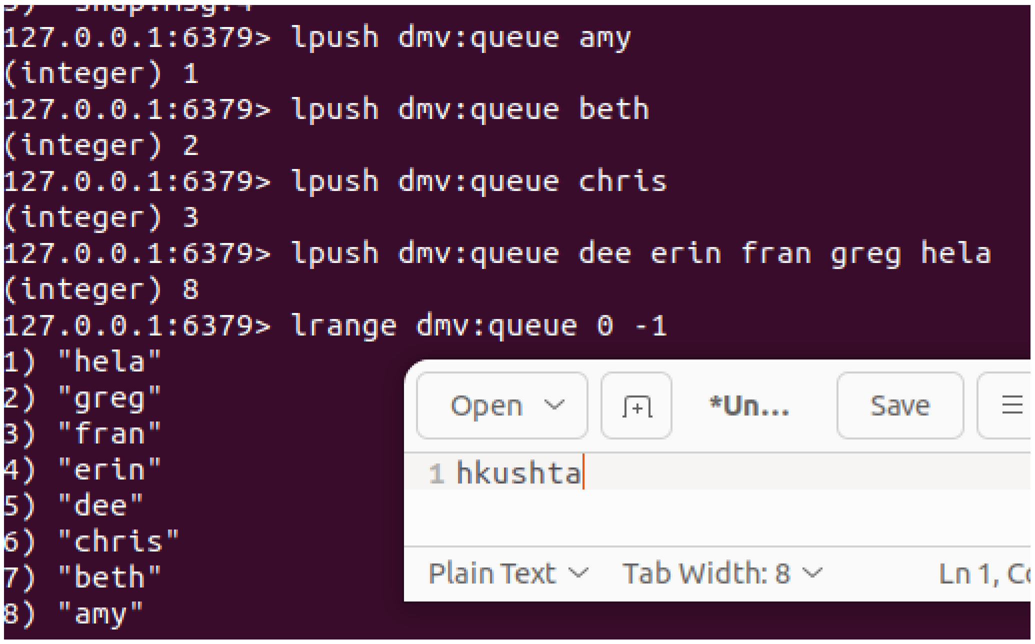
Event: When Dave is done at the window D, Bart is served next:

Users In queue: Tom, Bill

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

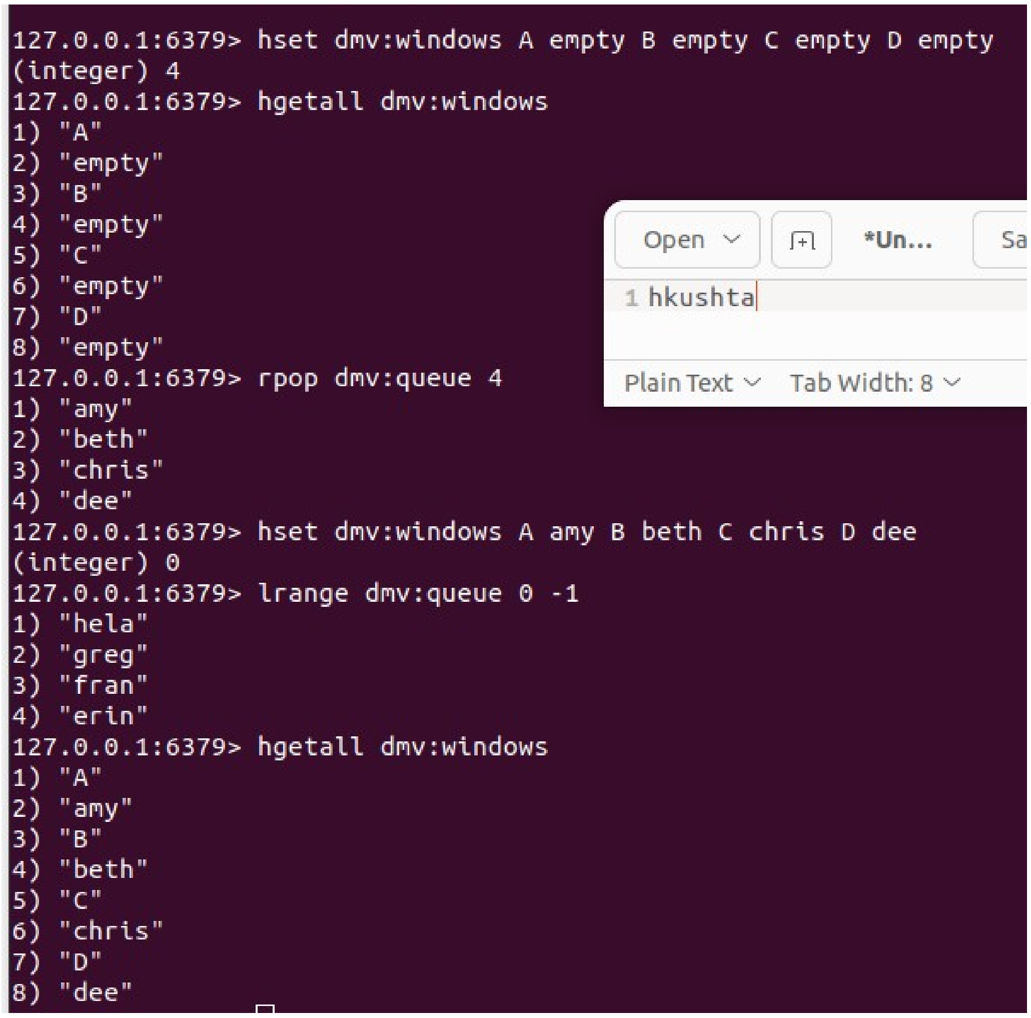
Event: Mary arrives
Users In queue: Mary, Tom, Bill
Being Served at windows: A: Carl, B: Steve, C: Chuck, D: Bart

4. 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



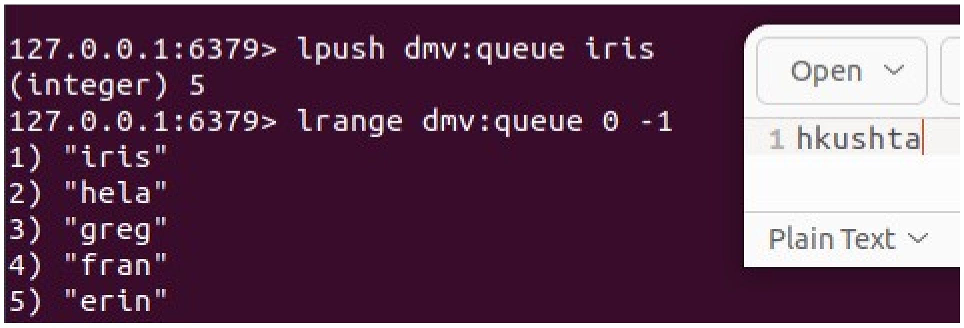
```
127.0.0.1:6379> lpush dmv:queue amy
(integer) 1
127.0.0.1:6379> lpush dmv:queue beth
(integer) 2
127.0.0.1:6379> lpush dmv:queue chris
(integer) 3
127.0.0.1:6379> lpush dmv:queue dee erin fran greg hela
(integer) 8
127.0.0.1:6379> lrange dmv:queue 0 -1
1) "hela"
2) "greg"
3) "fran"
4) "erin"
5) "dee"
6) "chris"
7) "beth"
8) "amy"
```

5. 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.



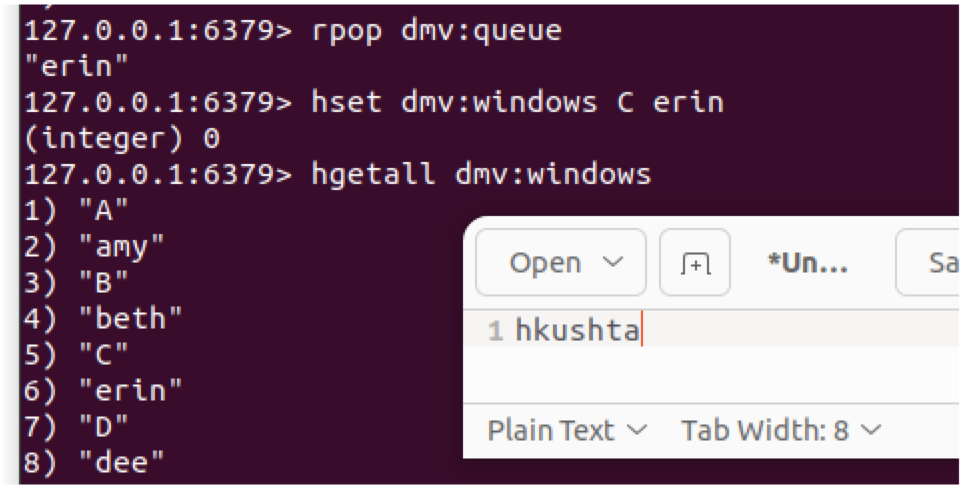
```
127.0.0.1:6379> hset dmv:windows A empty B empty C empty D empty
(integer) 4
127.0.0.1:6379> hgetall dmv:windows
1) "A"
2) "empty"
3) "B"
4) "empty"
5) "C"
6) "empty"
7) "D"
8) "empty"
127.0.0.1:6379> rpop dmv:queue 4
1) "amy"
2) "beth"
3) "chris"
4) "dee"
127.0.0.1:6379> hset dmv:windows A amy B beth C chris D dee
(integer) 0
127.0.0.1:6379> lrange dmv:queue 0 -1
1) "hela"
2) "greg"
3) "fran"
4) "erin"
127.0.0.1:6379> hgetall dmv:windows
1) "A"
2) "amy"
3) "B"
4) "beth"
5) "C"
6) "chris"
7) "D"
8) "dee"
```

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



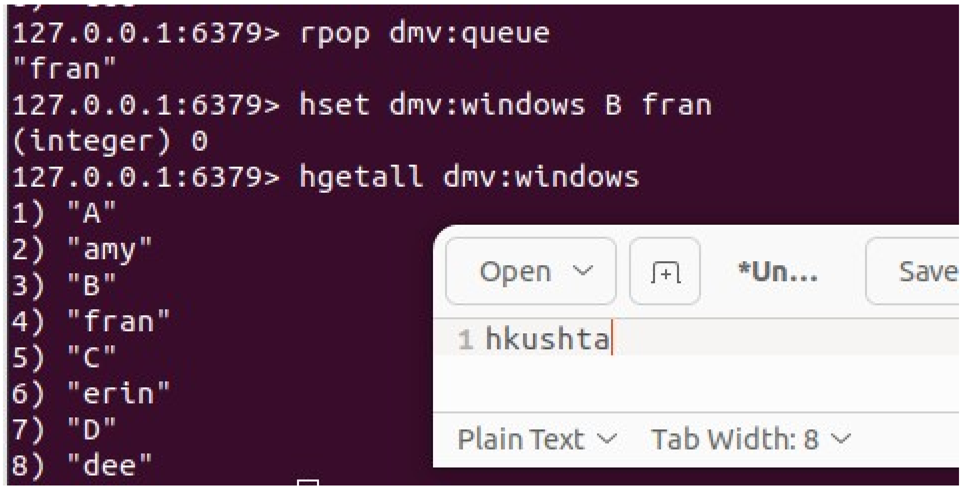
```
127.0.0.1:6379> lpush dmv:queue iris
(integer) 5
127.0.0.1:6379> lrange dmv:queue 0 -1
1) "iris"
2) "hela"
3) "greg"
4) "fran"
5) "erin"
```

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



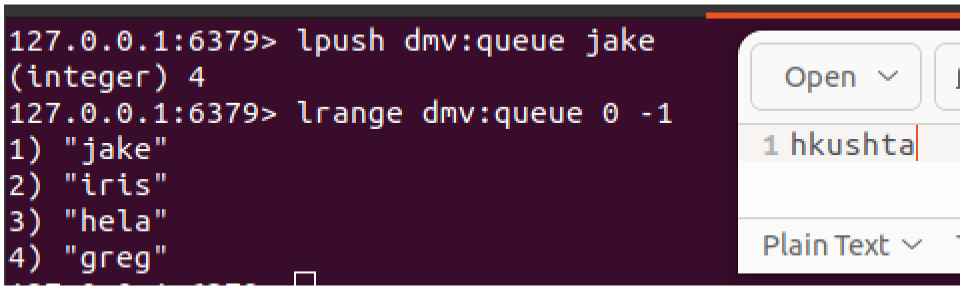
```
127.0.0.1:6379> rpop dmv:queue
"erin"
127.0.0.1:6379> hset dmv:windows C erin
(integer) 0
127.0.0.1:6379> hgetall dmv:windows
1) "A"
2) "amy"
3) "B"
4) "beth"
5) "C"
6) "erin"
7) "D"
8) "dee"
```

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



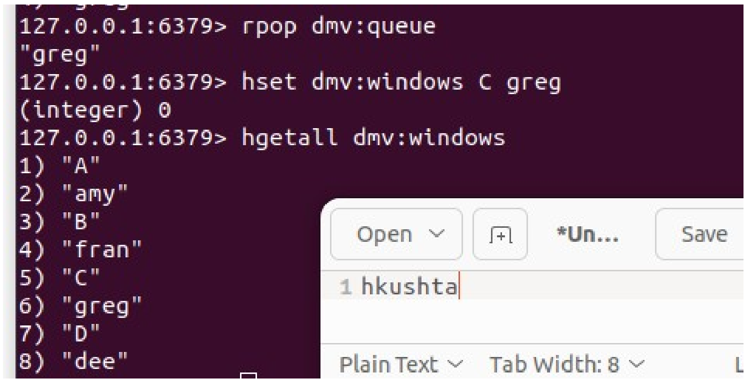
```
127.0.0.1:6379> rpop dmv:queue
"fran"
127.0.0.1:6379> hset dmv:windows B fran
(integer) 0
127.0.0.1:6379> hgetall dmv:windows
1) "A"
2) "amy"
3) "B"
4) "fran"
5) "C"
6) "erin"
7) "D"
8) "dee"
```

d. jake arrives



```
127.0.0.1:6379> lpush dmv:queue jake
(integer) 4
127.0.0.1:6379> lrange dmv:queue 0 -1
1) "jake"
2) "iris"
3) "hela"
4) "greg"
```

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

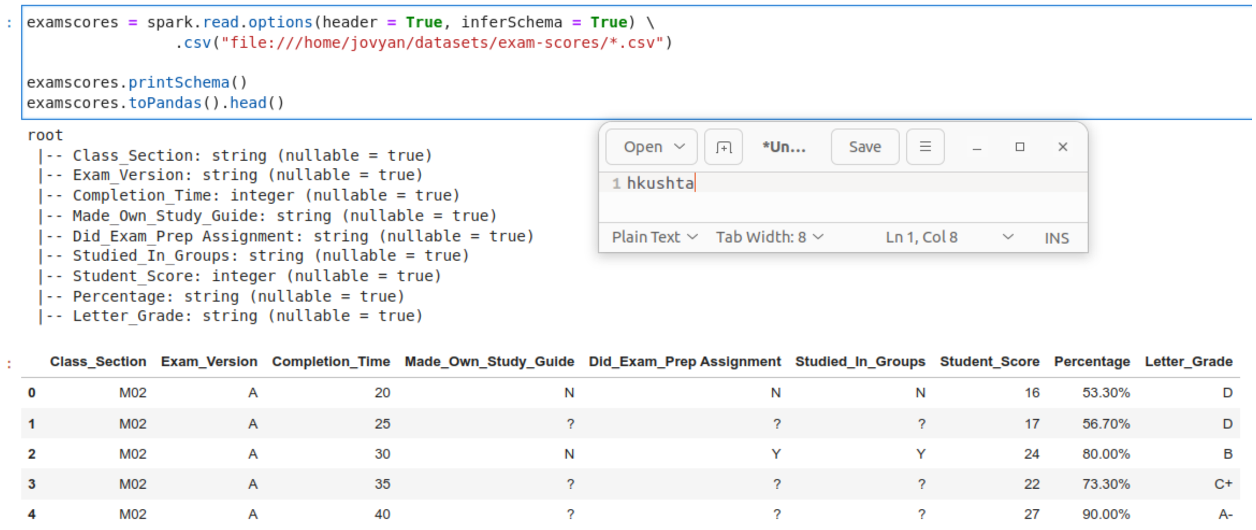


```
127.0.0.1:6379> rpop dmv:queue
"greg"
127.0.0.1:6379> hset dmv:windows C greg
(integer) 0
127.0.0.1:6379> hgetall dmv:windows
1) "A"
2) "amy"
3) "B"
4) "fran"
5) "C"
6) "greg"
7) "D"
8) "dee"
```

```
1 hkushta
```

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

7. 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.



```
examscores = spark.read.options(header = True, inferSchema = True) \
                .csv("file:///home/jovyan/datasets/exam-scores/*.csv")

examscores.printSchema()
examscores.toPandas().head()
```

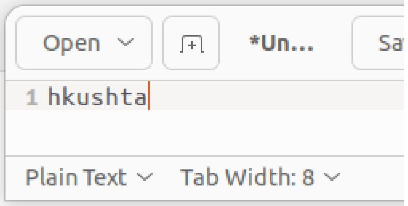
```
root
 |-- Class_Section: string (nullable = true)
 |-- Exam_Version: string (nullable = true)
 |-- Completion_Time: integer (nullable = true)
 |-- Made_Own_Study_Guide: string (nullable = true)
 |-- Did_Exam_Prep Assignment: string (nullable = true)
 |-- Studied_In_Groups: string (nullable = true)
 |-- Student_Score: integer (nullable = true)
 |-- Percentage: string (nullable = true)
 |-- Letter_Grade: string (nullable = true)
```

```
1 hkushta
```

	Class_Section	Exam_Version	Completion_Time	Made_Own_Study_Guide	Did_Exam_Prep Assignment	Studied_In_Groups	Student_Score	Percentage	Letter_Grade
0	M02	A	20	N	N	N	16	53.30%	D
1	M02	A	25	?	?	?	17	56.70%	D
2	M02	A	30	N	Y	Y	24	80.00%	B
3	M02	A	35	?	?	?	22	73.30%	C+
4	M02	A	40	?	?	?	27	90.00%	A-

```
examscores.createOrReplaceTempView("examscores")
```

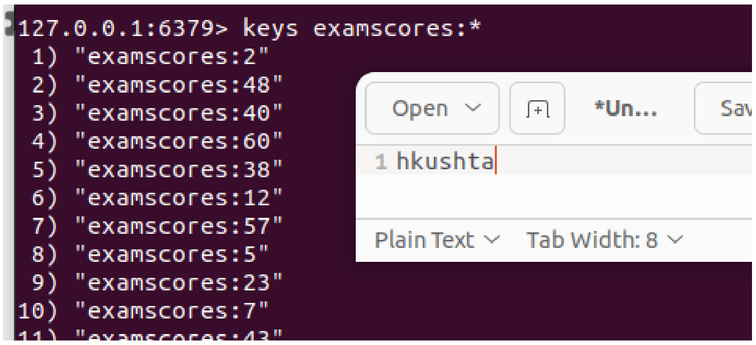
```
query = '''

select
row_number() over (order by Class_Section) as rowid,
*
from examscores

'''

examscores2 = spark.sql(query)
examscores2.printSchema()
```



```
1 hkushta
```

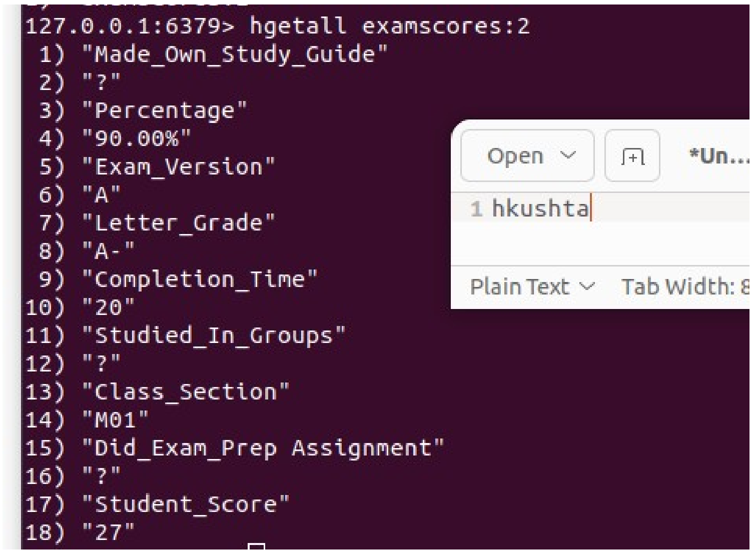
```
root
 |-- rowid: integer (nullable = true)
 |-- Class_Section: string (nullable = true)
 |-- Exam_Version: string (nullable = true)
 |-- Completion_Time: integer (nullable = true)
 |-- Made_Own_Study_Guide: string (nullable = true)
 |-- Did_Exam_Prep Assignment: string (nullable = true)
 |-- Studied_In_Groups: string (nullable = true)
 |-- Student_Score: integer (nullable = true)
 |-- Percentage: string (nullable = true)
 |-- Letter_Grade: string (nullable = true)
```

```
# Write to back to redis as a hash under the following key stocks
examscores2.write.format("org.apache.spark.sql.redis")\
  .mode("overwrite")\
  .option("table", "examscores")\
  .option("key.column","rowid")\
  .save()
```

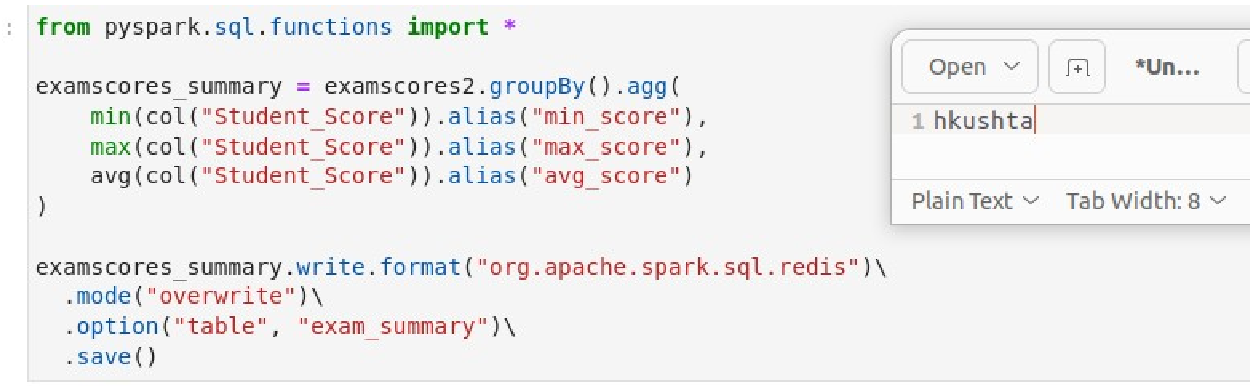


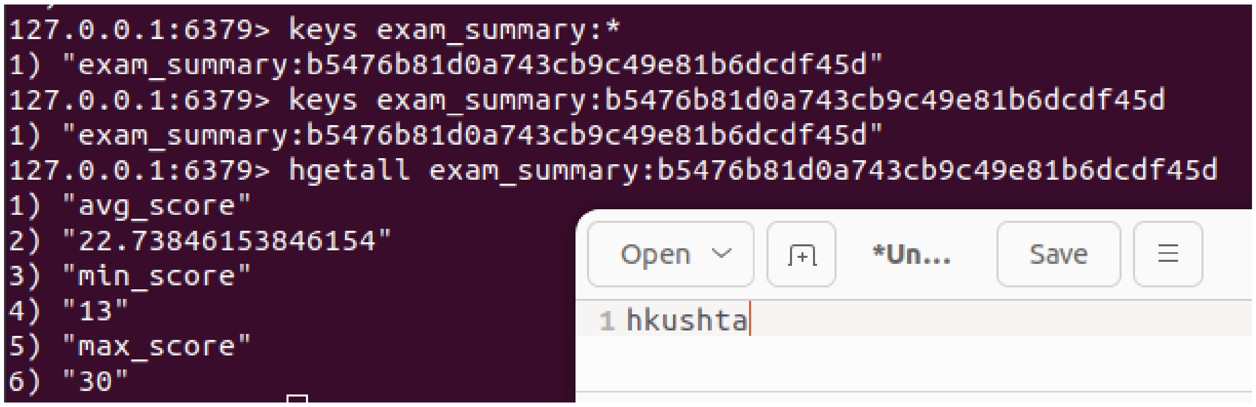
```
127.0.0.1:6379> keys examscores:*
 1) "examscores:2"
 2) "examscores:48"
 3) "examscores:40"
 4) "examscores:60"
 5) "examscores:38"
 6) "examscores:12"
 7) "examscores:57"
 8) "examscores:5"
 9) "examscores:23"
10) "examscores:7"
11) "examscores:43"
```

```
1 hkushta
```


```
127.0.0.1:6379> hgetall examscores:2
 1) "Made_Own_Study_Guide"
 2) "?"
 3) "Percentage"
 4) "90.00%"
 5) "Exam_Version"
 6) "A"
 7) "Letter_Grade"
 8) "A-"
 9) "Completion_Time"
10) "20"
11) "Studied_In_Groups"
12) "?"
13) "Class_Section"
14) "M01"
15) "Did_Exam_Prep Assignment"
16) "?"
17) "Student_Score"
18) "27"
```

8. 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()
```


```
127.0.0.1:6379> keys exam_summary:*
1) "exam_summary:b5476b81d0a743cb9c49e81b6dcdf45d"
127.0.0.1:6379> keys exam_summary:b5476b81d0a743cb9c49e81b6dcdf45d
1) "exam_summary:b5476b81d0a743cb9c49e81b6dcdf45d"
127.0.0.1:6379> hgetall exam_summary:b5476b81d0a743cb9c49e81b6dcdf45d
1) "avg_score"
2) "22.73846153846154"
3) "min_score"
4) "13"
5) "max_score"
6) "30"
```