Exercise 1: Build a simple key-value database using Redis.

The process of designing the database includes deciding on what keys to use and what is the type of their values. Hence, let us read the following case study and build the suggested key-value pairs in your database. (Please use your student ID as part of the key instead of the account number for each key and read the instruction at the end of the HW to submit your solution).

Start this exercise by following the steps in this sheet numbered 1-4.

Case Study: Key-Value Databases for Mobile Application Configuration (TransGlobal Transport and Shipping (TGTS))

TGTS coordinates the movement of goods around the globe for businesses of all sizes. Customers of TGTS contact the shipper and provide detailed information about packages and cargo that need to be shipped.

To help their customers track their shipments, TGTS is developing a mobile app called TGTS Tracker. TGTS Tracker will run on the most popular mobile device platforms. To allow customers to monitor their shipments from any of their mobile devices, application designers have decided to keep configuration information about each customer in a centralized database. This configuration information includes:

- Customer name and account number
- Default currency for pricing information
- Shipment attributes to appear in the summary dashboard
- Alerts and notification preferences
- User interface options, such as preferred color scheme and font

In addition to configuration information, designers want the app to quickly display summary information in a dashboard. Slower response times are acceptable when customers need to look up more detailed information about shipments. The database supporting TGTS Tracker should support up to 10,000 simultaneous users, with reads making up 90% of all I/O operations.

The range of data that is required by the mobile app is fairly limited so the designers felt confident that a single namespace would be sufficient. They chose **TrackerNS** as the name of the app's namespace.

1. **Build a Redis DB named TrackerNS.**

Each customer has an account number, so this was selected as a unique identifier for each customer. The app designers decided to use the following naming convention for keys:

entity type:account number.

Replace the account number with your student ID

Given the list of data types the tracker manages, the designers decided the database should support four entity types:

- Customer information, abbreviated ‘cust’
- Dashboard configuration options, abbreviated ‘dshb’
- Alerts and notification specifications, abbreviated ‘alrt’
- User interface configurations, abbreviated ‘ui’

The designers then moved on to decide on the structure of values. After reviewing preliminary designs of the user interface, they determined that name and account number appear frequently together, so it made sense to keep them together in a single list of values. The default currency is also frequently required, so it is included in the *list of values* along with customer name and account number. The next step in the design process is determining attributes for each entity.

The **customer** entity maintains the customer name and preferred currency. The account number is part of the key, so there is no need to store it in a *list of attribute value pairs*.

2. **Build the first key-value pair of type list to store the name, and currency for each customer using the key “cust:accountNo”. Hint, use LPUSH command in Redis to set the value of this key to a list of values.**

The **dashboard** configuration detail is a *list of up to six attributes about a shipment* that will appear on a summary screen. Save this information using JSON structure. The following are options, with abbreviations in parentheses:

- Ship to company (shpComp)
- Ship to city (shpCity)
- Ship to state (shpState)
- Ship to country (shpCountry)
- Date shipped (shpDate)
- Expected date of delivery (shpDelivDate)
- Number of packages/containers shipped (shpCnt)
- Type of packages/containers shipped (shpType)
- Total weight of shipment (shpWght)
- Note on shipment (shpNotes)

3. **Build the key-value pair of type JSON to store the dashboard configuration for each customer using the key “dshb:accountNo”. Hint, use JSON.set command in Redis to set the value of this key to attribute-value design in JSON ex. {“shpComp”: “My company”, “shpWght”: “20kg”}. Here is an example of how to set a JSON value in Redis. The $ set the path of the JSON file to be created and this sign indicate the root folder.**

JSON.set dshb:2222 $ ‘{“shpComp”: “My company”, “shpWght”: “20kg”}’

The **alerts** and notification data indicate when messages should be sent to a customer. An alert and notification can be sent via text message when a shipment is delivered. The phone number is modelled with a *simple value*.

4. **Build the key-value pair of type simple string to store the alert configuration for each customer using the key "alrt:accountNo".**

Finally, the **user interface** configuration options are a *simple list of attribute value pairs*, such as font name, font size, and color scheme. A key-value pair for a user interface specification could be a Hash map or JSON file.

5. **Build the key-value pair of hash map to store the user interface configuration for each customer using the key "ui:accountNo". Hint, use HSET command in Redis to set the value of this key to hash map structure. You can list all the attribute- value pairs next to this command. Ex.** *HSET ui:222 att1 val1 att2 val2* **. You can retrieve the value of an attribute using the command** *HGET ui:222 att1*

To submit your solution

After building the TrackerNS DB in your Redis cloud or server and creating the above four key-value pairs as indicated, paste a snippet (Picture) of your Keys created in Redis into a word file with the SET command printed in the file. Name the word file **Ex1_Redis_YourStudentID** and upload it in the HW opened on e-learning.