Assignment 2

**Data and Implementation**

I created a simple e-commerce platform where customers can purchase various products. The Redis database will store the details of these products, such as their id, name, price, description, and availability.

Here's dataset with 10 key/value pairs, each representing a product with different data types.

1. SET product1 ‘{“id":1, name": "Wireless Headphones", "price": 39.99, "description": "High-quality wireless headphones", "available": true}’
2. SET product2 {“id":2, name": "Smart Watch", "price": 99.99, "description": "A multi-functional smartwatch", "available": true}
3. SET product3 ‘{“id"=3, name": "Fitness Tracker", "price": 49.99, "description": "A smart fitness tracker for your daily activities", "available": false}’
4. SET product4 ‘{"name": "Bluetooth Speaker", "price": 29.99, "description": "A portable Bluetooth speaker", "available": true}’
5. SET product5 ‘{"name": "Wireless Mouse", "price": 19.99, "description": "A comfortable and responsive wireless mouse", "available": true}’
6. SET product6 ‘{"name": "Gaming Keyboard", "price": 89.99, "description": "A high-performance gaming keyboard", "available": true}’
7. SET product7 ’{"name": "Wireless Charger", "price": 14.99, "description": "A fast and efficient wireless charger", "available": true}’
8. SET product8 ‘{"name": "Portable Power Bank", "price": 24.99, "description": "A compact and powerful portable power bank", "available": true}’
9. SET product9 ‘{"name": "USB Flash Drive", "price": 9.99, "description": "A reliable and secure USB flash drive", "available": true}’
10. SET product10 ‘{"name": "Smart Home Hub", "price": 149.99, "description": "A versatile and intelligent smart home hub", "available": false}

Queries

1. INCR product1:id

This command will increment the "id" value of the "product1" key by 1. Since the original value was 1, after this command is executed the "id" value of the "product1" key will be 2.

1. DECR product2:price

This command will decrement the "price" value of the "product2" key by 1. Since the original value was 25, after this command is executed the "price" value of the "product2" key will be 24.

1. EXISTS product3

This command will check if the key "product3" exists in the database. Since the "product3" key was set in the database, this command will return 1 (true).

1. SETNX product4 '{"name": "Bluetooth Speaker", "price": 29.99, "description": "A portable Bluetooth speaker", "available": true}'

This command will set the value of the "product4" key only if the key does not already exist in the database. Since the "product4" key was already set in the database, this command will not modify the value.

1. TTL product5

This command will return the time-to-live (TTL) of the "product5" key, which is the amount of time left before the key expires. Since no TTL was set for the "product5" key, this command will return -1 (the key has no expiration).

1. KEYS \*

This command will return all keys in the database. In this case, it will return a list of all the product keys ("product1", "product2", till “product10.).

1. TYPE product6

This command will return the type of the value stored in the "product6" key, which is a string.

1. DEL product7

This command will delete the key product7 and its pairs in the database

1. LPUSH cart1 product1 product2 product5

This command will add the values of the "product1", "product2", and "product5" keys to the beginning of a list with the key "cart1". This simulates adding these products to a shopping cart.

1. SADD wishlist1 product1 product3 product10

This command will add the values of the "product1", "product3", and "product10" keys to a set with the key "wishlist1". This simulates adding these products to a wishlist.

**Code Explanation**

I use the SET to insert value into the redis database comprising of different data types for the keys and pairs.

INCR product1:id: This command is used to increment the value of the "id" field of the "product1" key by 1. It is assumed that the "id" field is a unique identifier for each product, and this command ensures that each new product added to the database will have a unique "id" value.

DECR product2:price: This command is used to decrement the value of the "price" field of the "product2" key by 1. It is assumed that this command could be used to decrease the price of a product temporarily during a sale or promotion.

EXISTS product3: This command is used to check if the "product3" key exists in the database. It is assumed that this command could be used to check if a product is available before attempting to add it to a shopping cart or wishlist.

SETNX product4: This command is used to set the value of the "product4" key only if the key does not already exist in the database. It is assumed that this command could be used to add a new product to the database without overwriting any existing data.

TTL product5: This command is used to check the time-to-live (TTL) of the "product5" key, which is the amount of time left before the key expires. It is assumed that this command could be used to automatically remove a product from the database after a certain period of time.

KEYS \*: This command is used to return a list of all keys in the database. It is assumed that this command could be used to retrieve a list of all products in the database for display purposes.

TYPE product6: This command is used to return the data type of the value stored in the "product6" key, which is a string. It is assumed that this command could be used to check the data type of a value before performing any operations on it.

DEL product7: This command is used to delete the "product7" key and its associated data from the database. It is assumed that this command could be used to remove a product from the database if it is no longer available for purchase.

LPUSH cart1 product1 product2 product5: This command is used to add the values of the "product1", "product2", and "product5" keys to the beginning of a list with the key "cart1". It is assumed that this command could be used to simulate adding products to a shopping cart.

SADD wishlist1 product1 product3 product10: This command is used to add the values of the "product1", "product3", and "product10" keys to a set with the key "wishlist1". It is assumed that this command could be used to simulate adding products to a wishlist.

**Rationale of the design decision**

An e-commerce platform built on the Redis database is intuitive and easy to use. Redis commands can be used to quickly access and modify product data because it is stored as key/value pairs. The nature of the data being saved influenced the decision to utilize multiple data types for the values of each key. Product fields can be either numeric (the "id") or string-based (the "name" and "description"), as an example. This guarantees the data is kept in a manner that can be read and understood with minimal effort.

Benefits

* Superior efficiency: Since Redis operates in RAM, it can process data requests and updates at lightning speed. It is commonly used as a caching layer to boost application performance and can process millions of transactions per second.
* Redis is versatile in that it can store and retrieve data in many different formats, such as strings, hashes, lists, sets, and sorted sets. For this reason, it is a flexible database with many potential uses.
* Redis can scale horizontally, meaning more nodes can be added to a cluster, to accommodate more data and more users.
* Redis allows for data to be saved to disk for durability and recovery via snapshotting and AOF (append-only file) persistence.
* Redis's pub/sub messaging, Lua scripting, and transaction support are just a few of the characteristics that make it well-suited for real-time applications.

Drawback

* Redis's capacity is bounded by the amount of memory that is accessible to it, which is a major drawback. This can be a problem for programs that have to store a lot of information.
* Redis's lack of support for complicated queries, in comparison to standard SQL databases, might be problematic for software that has to perform sophisticated analyses of its data.
* Redis is meant to operate in a master-slave setup, thus if the master node goes down, the entire cluster could be affected. For highly available applications, this could be a problem.
* Redis does not support data validation or constraints, which can make it challenging to maintain data integrity in particular applications.
* Redis does allow transactions, but each one can only affect a single key or data structure. Complex transactions involving numerous data structures may be hindered by this.