* Why did you choose that language?

Ans:- I choose C++ for implementing the project because it is fast and as this shares the same characteristics with that of python, Java etc like inheritance, polymorphism, and related data structures like map, set etc. We can also use generic functions to extend the use of all the functions in future. Although python would be a great choice to implement it but I have not used python for a long time so due to less time, I choose the C++ as I have been using it for quite a long time and I am quite confident in it.

* What are the further improvements that can be made to make it efficient?

Ans:- The improvements that can be made to make this project efficient are-

1. We can use some kind of caching algorithm like LRU (Least Recently Used), LFU (Least Frequently Used) according to the usage by the users to efficiently cache most frequent data requested by the user. It will reduce the traffic load on the server and will make the application more fast and reliable.
2. We can reduce the memory consumption by deleting the unnecessary data structure used during the implementation of this assignment.
3. As the most frequently used data structure is map so we can use an efficient hash function to compute the hash of the key efficiently. This will make the application fast and reliable.
4. We can use python programming language to incorporate the wide range of data types to be stored as key and value. It may also reduce the number of lines of codes.
5. In the provided implementation there is no way to do the eviction of the keys which are expired. As it would be a problem if we have lot of expire key.
6. The given data for a particular session can be store in some file and a function can be constructed to import the data when the somehow the connection is lost.

* What data structures have you used and why ?

Ans:- The data structure used during the implementation of this project are as follows-

1. **Unordered\_map** :- This data structure is used to efficiently hash the key and store the value so that it can be retrieved efficiently in future.

The time complexity for average if there are very few hash collisions is O(1) and it will be O(N) {Where N is the number of keys in the hash table} in the worst case when there will be a lot of hash collisions.

1. **Map** :- This data structure has been used to string the rank of values for a particular key. It also searches if a value for the particular key is already present in the set so that it can be deleted when a value with the different rank will be called in the future.

The map is implemented using the balanced binary search tree so the time complexity for searching any value is O(log(N)) where N is the number of the entries in the map.

1. **Set** :- This data structure has been used for storing the values in sorted order corresponding to the key. The pair of values and its rank has been stored for sorting the values in set using the comparator function. This sorts the values according to the rank associated with the value.

* Does your implementation support multi threaded operations? If No why can’t it be? If yes then how ?

Ans :- This implementation does not support multi-threaded operations. If more than one connection simultaneously tries to change the value of a particular key value pair then it can lead to undesirable results. Let there be multiple connections for the same database then it might lead to situations and will give wrong output. The operations should be atomic in nature so that no other operation can interrupt in the middle of execution of particular instructions.

Although it does not support the multi-threaded operations, we can use the solutions of problems like the Reader-Writer Problem in transactions. Using locks and permissions we can efficiently solve this problem.