Project #2 – Redis / Key-Value Stores
Due Date: May 5th, 2019

A key-value store is a system that manages a collection of (*key,value*) pairs, where *key* is unique in this universe. Redis – and other systems – allow the value to be a single value (e.g. string, number), a set of values, a list of values, a hash, etc.

Assume a collection of (key, list) pairs, i.e. the value is a list of values, namely strings. Assume that key is a string as well. Let's call such a collection a *Key-List Store (KL Store)*. This is a special case of a multi-map data structure, where several values are mapped to a key.

Example:

Key	**List**
12	[t12, t67]
34	[t87, t12, t98]
…	…
76	[t121, t72, t99, t179]

a Key-List Store

Assume two domains of values D_1 and D_2

e.g. D_1 = {all possible customer ids}, D_2 = {all possible transaction ids}

Assume that there is a process P that generates a collection of value1, value2 pairs

$S = \{ (u,v): u \in D_1 , v \in D_2\}$

Examples of such processes:

- SELECT custID, transID FROM SALES
- Reading a CSV file and getting for each line forming a pair using columns i and j.
- Running any program that produces a stream of pairs of values

Given a collection S described as above, one can define two KLStores, $KL_1(S)$ and $KL_2(S)$ as follows:

$KL_1(S) = \{(x, L_x), \forall x \in U = \{u: (u,v) \in S\}, L_x$ = the list of values v, such that $(x,v) \in S\}$
$KL_2(S) = \{(x, L_x), \forall x \in V = \{v: (u,v) \in S\}, L_x$ = the list of values u, such that $(u,x) \in S\}$

We want to implement in Python (or some other language) the following functions/methods that get one or more KL stores and "return" (or update) a KL store. All these KL stores should exist in Redis.

Create_KLStore (name, data-source, query-string, position1, position2, direction)

This function creates in Redis a KL store with name <name> using the data source found in <data-source>. Data sources can be found in an XML file described later and for the scope of this project can be either a csv file, a relational database or an excel file. In the case of a csv file, <query-string> is empty and <position1> and <position2> two integer numbers specifying the column positions that will be used to form the (u,v) pairs of S (as described earlier). In the case of an excel, <query-string> contains the index of the worksheet and <position1> and <position2> two integer numbers specifying the column positions that will be used to form the (u,v) pairs of S (as described earlier). In the case of a relational database, <query-string> is an SQL statement in the form SELECT col1, col2 WHERE <etc>. <direction> has the value 1 or 2, specifying whether $KL_1(D)$ or $KL_2(D)$ should be implemented.

Filter_KLStore(name1, expression)

This function gets a KL store in Redis named <name1> and a string called <expression> representing a valid python boolean expression and applies this expression on each element of each list of <name1>. If the return value is **true**, the element remains in the list, otherwise it is removed. Come up with a convention on how the element of the list is mentioned within the <expression>.

Apply_KLStore (name1, func)

This function gets a KL store in Redis named <name1> and a python function named <func> - which gets a string and returns a string – and applies <func> on each element of a list, for all lists of the KL store $<\text{name1}>$, transforming thus the lists of the KL store.

Aggr_KLStore (name1, aggr)

This function gets a KL store in Redis named <name1> and a string named <aggr> that can have one of the values “avg/sum/count/min/max” and aggregates each list of the KL store <name1> according to the specified aggregate, updating the list with just one item, the result of the aggregation. You can implement a more general version of this function that also gets a python function <func> that operates on a list of strings and returns a string; in this case you should modify the signature of Aggr_KLStore appropriately (e.g. Aggr_KLStore (name1, aggr, func), if <aggr> is an empty string then use <func> for aggregation).

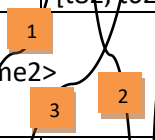
LookUp_KLStore(name1, name2)

This function gets two KL stores named <name1> and <name2> and for each element e of a list L in <name1>, performs a lookup for e in the keys of <name2>, gets the list L’ of the matched key,

and replaces e in L with the elements of L'. This should happen for all lists in <name1>. This is graphically shown in the figure below.

<name1>

Key	List
12	[t12, t13]
18	[t28]
…	…
56	[t82, t62, t71]



<name2>

Key	List
t11	[120, 67, 98]
t12	[68, 139, 65]
t13	[55, 12]
…	…
t99	[45, 89]

New list for 12 → [68, 139, 65, 55, 12]

ProjSel_KLStore(output_name, pname1, pname2, …, pnamek, expression)

Let's assume n KL stores, nm_1, nm_2, …, nm_n, sharing the same keys, i.e. key column is drawn from the same domain D. The figure below shows such an example for three KL stores.

nm_1

Key	List
t11	[31, 62, 9]
t15	[75, 91]
t22	[55, 12, 112]
t39	[44]
t44	[42, 98]

nm_2

Key	List
t11	[12, 6, 95]
t15	[128]
t22	[43]
t32	[39, 77]
t44	[129]

nm_3

Key	List
t11	[182]
t16	[7, 9]
t22	[56, 29]
t38	[32, 82]
t44	[66, 121, 22]

The goal of this function (operator) is to perform a join on the common keys of some KL stores, creating a new KL store having keys the common keys and corresponding list the concatenation of the individual lists in nm_1, nm_2,…, nm_n. In other words, if there is a key k in *all* KL stores nm_1, nm_2,…, nm_n, and L_k^1, L_k^2,…L_k^n the corresponding lists, then we insert a key-list pair in the new KL store as (k, concatenation(L_k^1, L_k^2,…L_k^n)). In the example above, the new KL store would be the following:

Key	List
t11	[31, 62, 9, 12, 6, 95, 182]
t22	[55, 12, 112, 43, 56, 29]
t44	[42, 98, 129, 66, 121, 22]

In addition, we would like to specify at the same time a filtering condition for this new KL store, based on the key and the contents of the lists involved, for example “key <> ‘t22’ and nm_2 > 20”. Such an expression is ill-defined because lists are not atoms. However, we would like to keep these semantics for reasons that have to do with user experience and understanding at the conceptual level (not discussed here). For the scope of this assignment, translate it as “any element of the list, randomly chosen, for example the first one”.

<output_name> is the name of the KL store that will be created
<pname1>, <pname2>,…,<pnamek> are the names of the KL stores that will be used for the output, i.e. their lists will be concatenated on common keys
<expression> is a valid python boolean expression with the following convention: within the expression, *key* and *KL stores* involved should be prepended by some special symbol(s) – e.g. “##key <> ‘t22’ and ##age > 20”.

Dictionary of Data Sources

There is a dictionary in XML describing data sources. Feel free to change it to JSON if you feel more comfortable with JSON.

```
<datasources Repository=”SomeName”>
        <datasource name=”DSName” id=”someID” type=”db/excel/csv”>
        <dbconnect>
                <username> username </username>
                <password> username </password>
                <request string>
                <database>
        </dbconnect>
        <filename> someFileName </filename>
        <path> somePath </path>
        <delimiter> someDelimiter </delimiter>
        </datasource>
</datasources>
```