CS Games 2017

Relay Programming II

Participants	3
Workstations	3
Value	7%
Duration	3 hours

Key-Value datastore

We are a Silicon Valley start-up that wants to implement its own Key-Value datastore for our application back-end to maintain a cache of the most used data. You must implement the command-line client for this datastore, so that feasibility testing can proceed. Keys are GUIDs (Globally Unique Identifier) and values are objects. Objects may be persisted in any format but object input and output must be JSON formatted.

* You’re not obligated to use a GUID for the key of an object, but the key should be unique across the database.

Key-Value datastore concept

The first argument to your command line client shall be the name of the operation to execute on the datastore. Input values shall be provided as arguments and output shall be written to standard output. The following operations shall be available (each is accompanied by an example):

insert : Insert a new entry and returns the generated key to it. Accepts as a parameter the value to insert.
For example, adding this value:

```
{
   "name": "john doe",
   "phoneNumber": "514-555-5555"
}
```

```
$ kvdb insert "{\"name\": \"john doe\", \"phoneNumber\": \"514-555-5555\"}"
```

Returns :

```
{
   "key" : "910bd4c8-34cc-42b4-b434-d4b6ec964f6f"
}
```

insert-with-key : Insert a new entry with a specified key. If a value already exists for the key, previous data is overwritten.
For example:

```
$ kvdb insert-with-key "{ \"key\": \"910bd4c8-34cc-42b4-b434-d4b6ec964f6f\", \"value\": {
\"name\": \"john doe\", \"phoneNumber\": \"514-555-5555\" } }"
```

This operation shall not return anything.

select-by-key : Selects the value associated with the specified key. If it exists, its value is returned. If none found, you can return an empty JSON object or returns nothing.

```
$ kvdb select-by-key 910bd4c8-34cc-42b4-b434-d4b6ec964f6f
```

Returns:

```
{
  "name": "john doe",
  "phoneNumber": "514-555-5555"
}
```

select-by-field : Selects one or many values and their key if they satisfy the condition passed as an argument.

```
$ kvdb select-by-field "name=john doe"
```

Returns:

```
[{ "key": "910bd4c8-34cc-42b4-b434-d4b6ec964f6f", "value": { "name": "john doe", "phoneNumber": "514-555-5555" } }, ...]
```

delete-by-key : Deletes the value that corresponds to the key passed as an argument.

```
$ kvdb delete-by-key 910bd4c8-34cc-42b4-b434-d4b6ec964f6f
```

This operation shall not return anything.

delete-by-field : Deletes the value that corresponds to the key passed as an argument.

```
$ kvdb delete-by-field "name=john doe"
```

Returns:

```
["910bd4c8-34cc-42b4-b434-d4b6ec964f6f", ...]
```

update-by-key : Updates the value that corresponds to the key passed as an argument. As opposed to **insert-with-key**, does not change the values of fields not specified in the input. For example:

```
$ kvdb update-by-key "{\"key\": \"910bd4c8-34cc-42b4-b434-d4b6ec964f6f\", \"value\": { \"name\": \"bob\", \"ipAddress\": \"127.0.0.1\" }}"
```

This operation overwrote the value of the "name" field, which is now "bob". However, the "phoneNumber" field remains unchanged and the "ipAddress" field is added to the value.

This operation shall not return anything.

update-by-field : Updates one or many values that satisfy the condition passed as an argument. Returns the Keys to the updated values.
For example:

```
$ kvdb update-by-field name=bob "{ \"name\": \"john doe\" }"
```

Returns:
["910bd4c8-34cc-42b4-b434-d4b6ec964f6f", ...]

db-info : Returns information about the datastore, such as the count of values and space used in bytes.

```
$ kvdb db-info
```

Returns:

```
{
   "values_count": 1,
   "size": 16
}
```

Features

- ❏ Insertion of a new value (**insert**) return of its generated key. (2 pts)
- ❏ Insertion of a value with a specified key (**insert-with-key**). If the key already exists, its value is overwritten. (3 pts)
- ❏ Retrieval of a value by its key (**select-by-key**). Returns the value in standard output. (2 pts)
- ❏ Retrieval of all Key-Value pairs satisfying a case-insensitive condition (**select-by-field**) (for example: "name=john doe"). Returns all corresponding values and their keys. The only supported operator is equality. (5 pts)
- ❏ Delete a value from its key (**delete-by-key**). (2 pts)
- ❏ Delete one or more values that satisfy a specified condition (**delete-by-field**). Returns the affected keys in standard output. (3 pts)
- ❏ Update a value from its key (**update-by-key**). (2 pts)
- ❏ Update one or more values satisfying a condition (**update-by-field**). Returns the affected keys in standard output. (5 pts)

- ❏ Support union and intersection (OR/AND) for conditions. For example: "name=john doe and phoneNumber=514-555-5555" or "name=john doe or name=bob". (6 pts)

- ❏ Support "greater than" (>) operator in conditions, if the field is a number. (3 pts)

- ❏ Support "lesser than" (<) operator in conditions, if the field is a number (3 pts)

- ❏ Support the inequality operator (!=) in conditions for number and string fields. (3 pts)

- ❏ Implement the **db-info** operation that returns the count of stored values and the size of the data in bytes. (2 pts)

- ❏ Addition of an order argument for the **select-by-field** operation. This argument allows specifying which number field the returned list is ordered by, increasing. (5 pts)
    - Document this argument in a readme file.

- ❏ Support nested fields in conditions. Nested fields are accessed with the "." character. (6 pts)

    - For example, given the following value:

```
{
   "name": "john doe",
   "phoneNumber": "514-555-5555",
   "address": {
      "civicNumber": 1050,
      "street": "Notre-Dame",
      "city": "Montreal" } ,
}
```

      The condition "address.street=Notre-Dame" may be used to select it.

- ❏ Implement the function “startswith” for the selector condition. (3 pts)
    - For example:

    $ kvdb select-by-field startswith(name, ‘john’)

    Returns:
    [{ "key": "910bd4c8-34cc-42b4-b434-d4b6ec964f6f", "value": { "name": "john doe", "phoneNumber": "514-555-5555" } }, ...]

- ❏ Encrypt the file containing the values of your database. (2 pts)

- ❏ Include a “readme” file in your solution, document completed features and the execution of your software. (1 pt)

- ❏ Include a “run.sh” file to start your software. (1 pt)