Magic query builder

With the following exercise we are meant to assess your ability to:

- Analyse and understand complex problems and be able to break them down into smaller chunks which are easier to solve
- Write clean and readable code easy to review
- Apply design principles in favour of reusability and scalability
- Think out-of-the-box to come up with smart and creative solutions
- Work on schedule, delivering quick high-quality products

There is no time limit for completing this exercise, but once you start, we suggest not to spend more than

- 2 hours

Start the timer. If you find yourself investing more than the allocated time for this exercise, you are probably doing a great job but missing your deadline! Do not stress and try to find the right balance that works for you.

You are required to complete this exercise by using either Go (our preferred choice) or Javascript/TypeScript.

Do not use any additional third-party package/library (e.g. mergo, lodash or similar).

Description

Your colleague, Tom, was tasked to build an endpoint that would query and filter from a NoSQL DB. The endpoint would be used by a FE application to allow users to filter over a set of data. This feature was roadmapped to be deployed to production in 4 weeks. Being a smart developer, Tom first enabled the endpoint and wrote a function to interface with the database. He was about to start writing the filtering logic; a function that will create a query based on query-params passed by the endpoint, before he came down with bad fever. The scrum master has indicated that you will now have to finish the job.

We would like to write a simple meta-language that will be interpreted by our program to build complicated queries. This meta-language consists of string-format text where `.` is the separator between keys (or different levels in a JSON) and `[]` is a special keyword that will tell our code the word that immediately precedes is a list.

Here an example of a filter:

`interests.[].sport.name = "football”`

And this will return all items containing this pattern (from the root level):

```
{
  "interests": [
    "sport": {
      "name": "football"
    }
  ]
}
```

equivalent to “dotted” version:

```
{
  "interests": [
    "sport.name": "football"
  ]
}
```

Goals

The final solution should include a function that will:

- take in input a list of query-filters as key-value pairs
- return as output the final query in a key-value format

We ask you to implement only the following filters:

- `__match__` which is the internal query keyword for `search in array`
- `__eq__` which is the internal query keyword for `element is equal to`

Notes

- “Dotted” version can be used to simplify JSON building

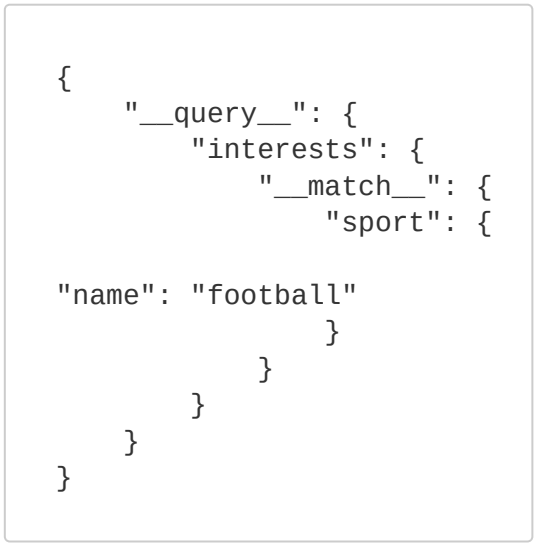
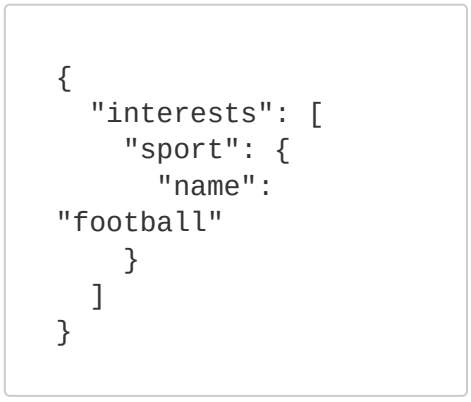
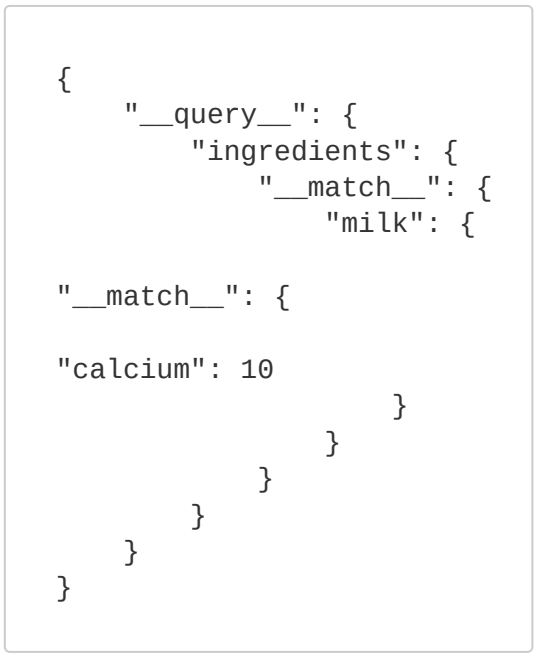
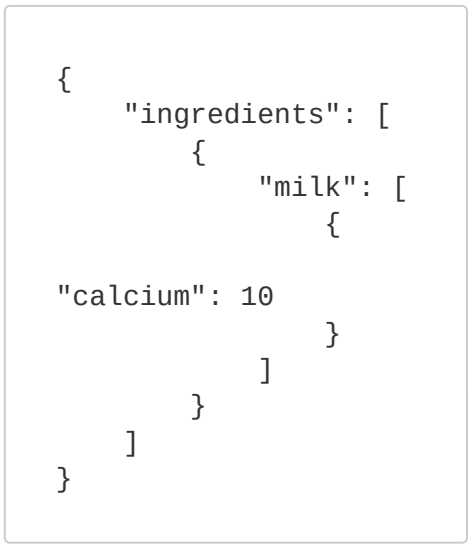
Requirements

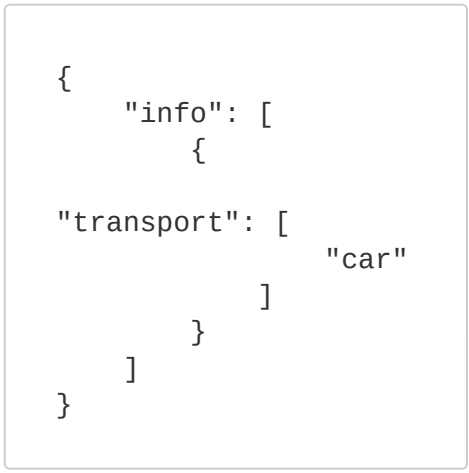
- The output query should have `__query__` as the starting key (i.e. `{ “__query__”: { .. } }`
- It should work for *n* levels of a query statement

Examples

Query params	Generated query	Output matched

`name.first_name = "Sam”`
 	
`address.country = "United Kingdom”`
	<pre>{
	"__query__": {
	"name": {
	"first_name": "Sam"
	},
	"address": {
	"country": "United Kingdom"
	}
	}
	}</pre>
		<pre>{
		"name": {
		"first_name": "Sam"
		},
		"address": {
		"country": "United Kingdom"
		}
		}</pre>

interests.[].sport.name = "football”		
ingredients.[].milk.[].calcium = 10		

info.[].transport.[] = ”car”		
Note: Common fields should be grouped together		
name.first_name = "Sam” name.last_name = "Watson”		

info.[].transport.[] = ”car”
 	
info.[].owners.[] = ”Tracy”
	{
	"__query__": {
	"info": {
	"__match__": {
	
	"transport": {
	
	"__eq__": "car"
	},
	"owners": {
	
	"__eq__": "35"
	}
	}
	}
	}
	}
		{
		"info": [
		{
		
		"transport": [
		"car"
		],
		"owners": [
		"Tracy"
		]
		}
		]
		}