International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
https://doi.org/10.32628/CSEIT217116

Recent Trends in JSON Filters

Atul Jain*, Dr. ShashiKant Gupta
CSA Department, ITM University, Gwalior, Madhya Pradesh, India

Article Info
Volume 7, Issue 1
Page Number: 87-93
Publication Issue :
January-February-2021

Article History
Accepted : 11 Jan 2021
Published : 24 Jan 2021

ABSTRACT

JavaScript Object Notation is a text-based data exchange format for structuring data between a server and web application on the client-side. It is basically a data format, so it is not limited to Ajax-style web applications and can be used with API's to exchange or store information. However, the whole data never to be used by the system or application, It needs some extract of a piece of requirement that may vary person to person and with the changing of time. The searching and filtration from the JSON string are very typical so most of the studies give only basics operation to query the data from the JSON object.
The aim of this paper to find out all the methods with different technology to search and filter with JSON data. It explains the extensive results of previous research on the JSONiq Flwor expression and compares it with the json-query module of npm to extract information from JSON.
This research has the intention of achieving the data from JSON with some advanced operators with the help of a prototype in json-query package of NodeJS. Thus, the data can be filtered out more efficiently and accurately without the need for any other programming language dependency. The main objective is to filter the JSON data the same as the SQL language query.

Keywords: JSON, JSONiq, Json-Query, Node

I. INTRODUCTION

On the web when it required to send data between the webserver to the client, it needs some lightweight format to store and transport.[1] Here JSON to be used: JavaScript Object Notation which is self-describing and easy to use.
JSON Syntax Rules
1. JSON-Data Should name-value pairs
2. JSON-Data Should comma separated
3. objects hold by Curly braces
4. arrays hold by Square brackets

Basically, JSON is name-value pairs, separated with commas. Here Objects are held by Curly braces and arrays are hold by square brackets. The name-value pair of JSON is consists of a field name in " double quotes" which is and followed by a colon that also followed by a value

"username":"AtulJain".

The syntax of JSON syntax has been derived from JavaScript object notation [2], however JSON format is text only. It can use any programming language to write code for generating and reading JSON data.

JSON basic datatypes are Number, String, Boolean, Array, Object, and null.

The below section has an example with JSON representation to describe an Employee details

Employee.

```
{
 "fName": "Atul",
 "lName": "Jain",
 "isActive": 1,
 "dateOfBirth": "01-11-1989",
 "ResidenceAddress": {
 "street": "G T Colony",
 "city": "Gurgaon",
 "state": "Haryana",
 "pincode": "122001"
 },
 "phoneNumbers": [
 {
  "Pref": "Office",
  "number": "9111111111"
 },
 {
  "Pref": "Resi",
  "number": "999 555-4567"
 }
 ],
 "kids": ["Aman"],
 "maritalStatus": "married"
}
```

JSON as Object Tree:

As each JSON object in a JSON document is a set of key-value pairs, a Naturally, JSON document can be represented as a data tree structure called JSON tree. The value can be an atomic value, such as a string, an integer, a number, an array, or null. To capture the compositional structure of JSON data. Here each value of JSON can also be a set of JSON objects.

A JSON document D is defined as

D::={Object[,Object, ...]},

where

Object ::= {Key:{Value} [, Key:{Value}, ...]},

Key ::= String,

Value ::= String|Integer|Number|Array|Null|Object.

For example, consider the following JSON document Doc1:

```
{
“name”: {
“empFirstName”: “Atul”,
“empLastName”: “Jain”
},
“empAge”: 45
}
```

JSON document Doc1 contains 2 objects Ob1 and Ob2 as following:

```
Ob1::= {
“name”: {
“empFirstName”: “Atul”,
“empLastName”: “Jain”
},
}
```

with key “name” and

```
Ob2::={
“empAge”: 45
}
```

with thekey “empAge” and an atomic integer value “45”.

The value of key of Ob1 again contains 2 objects Ob11 and Ob12 as following:

```
Ob11::= {
“empFirstName”: “Atul”,
}
```

with the key “empFirstName” and an string value “Atul” and

```
Ob12::= {
“empLastName”: “Jain”
}
```

with the key “empLastName” and an string value “Jain”.

Of course, a value which contain nothing,can be represented by type “Null”. For type “Array” and “Number” see the below example.

```
Ob3::={
“name”: {
“empFirstName”: “Mohit”,
“empLastName”: “Jain”
},
“empAge”: 28
}
and
Ob4::={
“name”: {
“empFirstName”: “Vipul”,
“empLastName”: “Jain”
},
“empAge”: 23
}
```

To combine Ob3 and Ob4 in JSON document Doc1, add a new key “children” in Doc1 and arrange Ob3 and Ob4 as an array [Ob3,Ob4] and numbered by numbers “1” and “2” sequentially as {1:Ob3, 2:Ob4}:

```
{
“name”: {
“empFirstName”: “Atul”,
“empLastName”: “Jain”
},
“empAge”: 32,
“children”:{1:Ob3, 2: Ob4}
}
```

II. LITERATURE REVIEW

Most of the Previous work on querying data interchange formats has basically focused on XML data [3]. A very few research for querying JSON data. Zorba is one of the most well approved JSONiq processor. The system is normally a virtual machine for the purpose of query processing. Both the XML and JSON data can be processed by using the JSONiq & XQuery languages respectively. But, it is not scalable and optimized to formultiple nodes with the multiple data files, which is most important to the focus of this work.

In contrast, Apache VXQuery is a system that can be deployed on a multi-node cluster to exploit parallelism. There are many parallel approaches that came out as well for the JSON data querying . These systems can be divided into two categories. The first category includes SQL-like systems such as Jaql, Trill, Drill, Postgres-XL, MongoDB and Spark, which can process raw JSON data. Moreover, they have been well integrated with well-known JSON parsers like Jackson. When the parser goes to reads raw JSON data, it converts the whole part into an internal data model like a table[4] .Now it can be then easily been processed by queries.,when it is in tabular format. This system can also read raw JSON data, but it has the advantage that it does not require data conversion to another format since it directly supports JSON’s data model[5]. The Queries can thus be processed very quickly as the JSON file is read. However Postgres-XL (a scalable extension to PostgreSQL) has a limitation on how it exploits its parallelism feature. Specifically,It is not designed to scale on multiple cores, while it scales on multiple nodes .

A. JSONiq (FLWOR Expression)

1)JSONiq [6].is a query and processing language which is exclusively designed for the JSON data model .

2)JSONiq is not concentrated on collaborative data, it avoids the uncertainty about the element or attributes, order of the decedent and namespace’s complications. Nowadays JSONiq became expressive query language for the semi-structured database, because of its optimality to query and query processing.

3)JSONiq is not only semi-structured query language, but it could illustrate programs to process the data, from transformations, selections, projections, joins of heterogeneous data sets, cleansing of data, enhancement of data, extraction of data, and so on

The most dominant expression in JSONiq is the FLWOR expression, which is an acronym for "For, Let, Where, Order by, Return".

```
1. let $test1 : = <test1 JSON database>
2. for $t in $test1()
3. group by $Marks := $t.Marks
4. where count($t) gt 1
5. return {"Marks" : $Marks, "Total Count" count($t)}
```

Now consider two JSON to check how Its extracting information through jsoniq.Two JSON database namely test1 and test2
Test1 (Sno, RollNo, Name, Marks)
Test2 (Sno, RollNo, Name, Marks)

1)Json-query1: “Fetch and priint the marks in descending order”.(Projection and selection)

```
for $t in $test1()
order by $t.Marks descending
return $t.Marks
```

2)Json-query2: “Fetch print the students who got more than 45 marks”(Filtering)

```
for $t in $test1()
where $t.Marks gt 45
return $t.Marks
```

3)Json-query3: “Print the count-no of each mark obtain by the
student having more than th33” (Aggregation)

```
for $t in $test1()
group by $Marks := $t.Marks
where count($t) gt 3
return {"Marks" : $Marks, "Total Count" :
count($t)}
```

4)Json-query4: “Print the students’ test1 marks and test2
marks”(Joins)

```
for $t1 in $test1(), $t2 in $test2()
where $t1."Roll No" eq $t2."Roll No"
return {
"Roll No" : $t1."Roll No",
"Test1Marks" :$t1.Marks,
"Test2 Marks":$t2.Marks
}
```

B. Node js Library

The JSON-query module of the node used to retrieves values from JSON objects. It also facilitates us with offer parameters, nested, and deep Queries. It gives the opportunity to create some custom filter function which can be used to make some other type of filtration and update the code as per the requirements. It can simply install it via npm [7] and used in out in code by including it above the code with the help of the “require” function.
Sytnax for install

```
npm install json-query
```

Just copy this command and run in the terminal.npm latest version will install into the system.
Syntax

```
var jsonQuery = require('json-query');
jsonQuery(query, options);
```

The above lines specify how to add the module in the code with the "require" function and how to use it in the application to the filtration of required object in the main object. Second-line describing what to query and from which data it needs to be queried. The return type of this function will also an Object which will be the result of that query.
Example :
Sample Data ->

```
var dataObj = {
 employee: [
 {username: 'Mohit', location: 'IN'},
 {username: 'Vipul', location: 'AU'},
 {username: 'Prashant', location: 'NZ'}
 ]
}
```

Query ->

```
jsonQuery('people[location=IN].username', {
 data: dataObj
})

Result ->
{
 value: 'Mohit',
 key: 'username',
 references: [ { username: 'Mohit', location: 'IN' } ],
 parents:
 [
  { key: null, value: [Object] },
  { key: 'employee', value: [Array] },
  { key: 0, value: [Object] }
 ]
}
```

1) Options

There are 7 options some are mandatory and some are optional.

- data or rootContext: This is the query's main Object.
- source or context (optional): This is the current object which is interested or which is accessed in query with
- parent : Its an additional optional context for looking further up the tree.
- locals: It is specify an object containing helper functions. Accessed by ':filter Name'. It Expects function(inputValue, arguments...) with this set to original passed in options.
- globals: when no local function found , it falls back to globals.
- force : It is Specify an optional object to be returned from the query if the query fails. It will be store into the place where query expected the object to be.
- allowRegexp (optional): Enable the ~ operator. Before enabling regexp match to anyone

2)Queries:

Queries in the npm manager can be defined as strings that used to describe an object or value to pick out, or manipulate from the given object having some mixed syntax of CSS and little bit JS but in more formatted and beaudiful way.

1) Accessing properties (dot notation) : Object name with dot(.) operator

person.username

2) Array accessors : Array name with index

employee[0]

3) Array pluck: : Array name with dot(.) operator

employee.username => return all the username of employee

4) Get all values of a lookup

lookup[*]

5) Array filter: By default only the first matching item will be returned:

employee[username=Mohit]

Here when add an "asterisk" (*), all matching items will be returned:

employee[*location=NZ]

6) comparative operators: To compare with some numeric value and return the result

employee[*rating>=3]

7) use boolean logic: To check the boolean value of some specific elements

employee[* rating >= 3 & starred = true]

8) Negate

It can also negate any of the object value by adding a ! before the = or ~:

employee[*location!=NZ]

9) Deep queries

Search through multiple levels of Objects/Arrays using [**]:

```
var data = {
 candidate: {
 'student': [
  {name: 'Atul', country: 'NZ'},
  {name: 'Vipul', country: 'US'},
  {name: 'Mohit', country: 'AU'},
  {name: 'Mayank', country: 'NZ'},
 ],
 'teacher': [
  {name: 'Prashant', country: 'AU'}
  {name: 'Vishal', country: 'NZ'},
 ]
 }
}
var result = jsonQuery('candidate[**][*country=NZ]', {data: data}).value
```

The result will be:

```
[
 {name: 'Atul', country: 'NZ'},
 {name: 'Mayank', country: 'NZ'},
 {name: 'Vishal', country: 'NZ'}
]
```

III. INDUCTION OF PROBLEM

Json-Query manager package providing us the basic operators like equals to (=), greater than(>), less than (<), both logical operator (AND, OR) only, which are not sufficient to filter some specific information from the large data to give an efficient output to the user, these operation only are able to handle single and straight forward data Extraction, but when handling with a huge amount of data, there are various scenarios which deal with conditional Data.

1) Here its need to check any value in the given array or if need any value except then that array then its required to write an extra algorithm in the development code which increases complexity of the code and is relatively time consuming which in terms affects the overall performance of the application.

2)There is no operator provided for extracting information with a specific range and to find Distinct values.

3)There are many operators equivalent to SQL which are not provided by npm manager. Json-Query Manager has no solution for retrieval of these Data and for these extractions, users have to go through various stages or create their own structural piece of code which is always time-consuming and demands a good skill set to write such a piece of code.

IV. PROPOSED TECHNIQUE

The objective of this research is to take NPM json-query manager extension to be as close and as easy as SQL is for RDBMS in terms of Data Extraction.

NPM JSON-query package is a packaged prototype to manage and merge advanced operators to filter data on the basis of filters and queries. This research is to find some advanced techniques and operators to provide efficient filtration with more operators by using a basic Javascript function to develop a more efficient and optimized filtration and provide custom filters like DISTINCT, IN, NOT IN, BETWEEN.

Further research aims to reduce the time complexity of programmers by providing the complied library function. The function will be developed prior to the use of basic JavaScript methods. To maintain the speed of result, it needs to first convert the JSON into a collection and implement all the related functions of collection to sort and filter according to given queries. JavaScript Collection already has inbuilt methods that are optimized to work on collections. These advanced methods will help to filter data to the maximum level and can be used in conjunction with each other as well.

V. CONCLUSION

By using existing methods and models to filter out the JSON, a developer can work only on limited JSON data and with limited methods to filter out.

However, in today's digital era,most of the web application using JSON as a data format on a big scale, So data manipulation and filtration need some mode advanced techniques to work with such data. It always needs to merge two or many operators to fulfill the requirements of the current filter with some extra models. This paper compares the existing JSON filters with the SQL operators and found there are many SQL operators equivalent that can be implemented in the JSON filters and save the time of code execution and increases the overall performance.

VI. REFERENCES

[1] Eko Wahyudi,Sfenrianto Sfenrianto, M. Jundi Hakim, Reko Subandi, Okky Robiana Sulaeman, Rochmat Setiyawan ."Information Retrieval System for Searching JSON Files with Vector Space Model Method" Computer Science Faculty, STMIK Nusa Mandiri, Jakarta,IEEE Indonesia.September 2019

[2] [online] Available: http://www.json.org/.

[3] Aayush Goyal,Curtis Dyreson USA,"Temporal JSON",February 2020

[4] Pierre Bourhis,Juan L. Reutter ,"JSON: Data model, Query languages and Schema specification" www.researchgate.net ,May 2017

[5] R. Vinothsaravanan,C. Palanisamy,"Extracting information from JSON database as simple as extracting in SQL using JSONiq", IEEE Xplore: April 2020.

[6] Daniela Florescu,Ghislain Fourny,"JSONiq: The History of a Query Language" Switzerland IEEE Internet Computing (Volume: 17, Issue: 5, Sept.-Oct. 2013)

[7] [online] Available Npm package library for json-query filter https://www.npmjs.com/package/json-query.

[8] Mahfuzul Amin, Rashedur M Rahman Bangladesh "Universal database access layer to facilitate query" IEEE December 2014

[9] Khiem Minh Nguyen,Thanh-Hai Nguyen,Xuan Hiep Huynh "Automated translation between RESTful/JSON and SPARQL messages for accessing semantic data", Cantho University,IEEE, Vietnam September 2016

[10] Tang Lv, Ping yan , "Survey on JSON Data Modelling",www.researchgate.net, August 2018

Cite this article as :

Atul Jain, Dr. ShashiKant Gupta, "Recent Trends in JSON Filters", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 1, pp. 87-93, January-February 2021. Available at

doi : https://doi.org/10.32628/CSEIT217116

Journal URL : http://ijsrcseit.com/CSEIT217116