Autocomplete Text using Graph Database

Rahul Patel
K. J. Somaiya College of Engineering

Sanket Sheth
K. J. Somaiya College of Engineering

Kavita Kelkar
K. J. Somaiya College of Engineering

ABSTRACT

This paper gives the idea about autocomplete predictive text search coupled with using graph database for storing various nodes. Firstly, the paper focuses on the search bar developed using web technologies that is platform independent and can be deployed on any system. It deals with the Jquery and AJAX mechanisms to give the user immediate autocomplete results before he or she has finished typing. Secondly, the paper discusses the role of a graph database technology used named Neo4j that stores nodes and orders relationships between them. The use of Cypher queries has been explained to retrieve the data from the graph database and JSON encode it on the screen where the user's current focus remains.

Keywords

AJAX, HTML, PHP,JQuery

1. INTRODUCTION

There are range of predictive text completion products used in the industry most notable is the auto-complete software used by google.com who use natural language user interface to execute this and depend on tons of storage devices to store the database. Basic idea behind this is that a user cannot type as fast as he or she can think. An autocomplete text input comes real handy as it just doesn't save precious time but also gives a range of predictions to the user depending upon his or her preferences to choose from.

The web in today's world has become very important as there are billions of people using it everyday. The amount of data stored and that appears on the web is humongous. What is more important is to search for something in particular that a user wants from this ocean of data and even before the process of searching the data by indexing or some other mechanism takes place, giving the user a list of choices on what he or she wants to search becomes essential and that too before they have finished typing it.

All data irrespective of the data structure or the technology used, is stored and referred to as some or the other node. Using the functionality of an autocomplete search bar means that the software or the web page should be well aware of where the data is stored, how to establish a connection with and and how and when to access it which may include reading the data or manipulating it. Mechanisms including but not limited to flat files and relational databases can be used for the same reason. Choosing a data structure like an array and storing data in it in flat files is a very naïve way of doing it. Besides relational databases like SQL services work fine when it comes to dealing with fewer nodes and limited functionalities. Redundancy, scalability and flexibility becomes a problem when an efficient service needs to be delivered.

The use of graph database becomes thus empirical. It becomes easier to utilize the data, storing it, reading it or manipulating it. The graph database will have nodes in place which may be connected to each other via some sort of a relationship. A many-to-many relationship can thus be achieved. Properties can be assigned to each and every other node so that it becomes easier to map data which otherwise would be difficult. Besides, a graph database is not challenged by SQL functionalities that may hold relational databases aback with what they can do and how much can they do it.

2. EXISTING SYSTEMS

Although the autocomplete functionality is being provided with the use of AJAX calls, Jquery and JavaScript functions as far as web technology is concerned, there are various existing ways to store and retrieve the data from the data structures used.

A simple implementation of an array data structure can store nodes with difeerent labels that are used to define the parameters realted to that row. The data is indexed so that retrieval can be based on accessing these indices. Explicitly the search for any particular node against the term entered involves traversing the entire data structure everytime a new letter is typed in. Efficiency is greatly affected when the data becomes significantly large.

Another way of doing this is by the use of Xtensible Markup Language or XML. A tree structure to establish heirarchical relationship. This is a better way that is being used as data traversal doesn't mean that the entire data structure will be accessed. XML queries can be fired to retrieve a particular element. Although this achieves a greater efficiency but establishing connection between two child nodes whose parental heirarchy is not the same is not possible.

We hereby propose the usage of graph database powered by Neo4j technology to achieve autocomplete results.

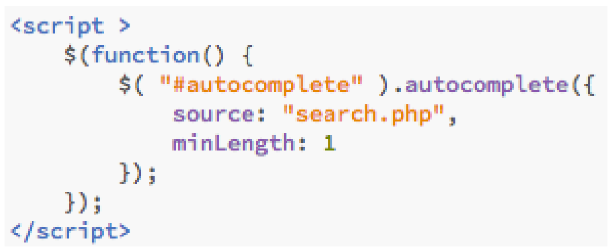
3. PROPOSED SYSTEM

3.1 Overview

Our system encompasses of languages like HTML, CSS, JavaScript, JQuery, AJAX, PHP, Cypher queries and Neo4j technology for the storage of data. The user can open the web page and find out a search bar provided.

Figure 1- The input search bar

Any input that the user makes will be subjected to couple of JavaScript functions and be stored in a PHP variable named 'term'. As this is going to make use of the form tag from HTML the data will be passed to a search.php file through the following implementation [4].



```
<script >
    $(function() {
        $( "#autocomplete" ).autocomplete({
            source: "search.php",
            minLength: 1
        });
    });
</script>
```

Figure 2 – JavaScript function that passes the input to the php file

The search.php file ensures the next step of establishing the connection with our Neo4j graph database and processing the variable ‘term’.

Neo4j enabled us to create a graph database with nodes, establish relationships among them and assign properties to each and every node. The data in the node gets stored in the file named neostore.nodestore.db on the disk. Every node is defined by its identification parameters and assigned various other properties unique to it. These properties are stored in the file named neostore.propertystore.db on the disk. Two nodes share a relationship that portray the manner in which they are connected to each other. These relationships are stored in the file named neostore.relationshipstore.db on the disk. An illustration of the above can be found below -

```
CREATE (Engg:Field {title:'Engineering', branch:'Science'})

CREATE (Comps:Department {title:'Computer Engineering',
field:'Engineering'})
CREATE (Mech:Department {title:'Mechanical Engineering',
field:'Engineering'})
CREATE (Etrx:Department {title:'Electronics Engineering',
field:'Engineering'})
CREATE (Extc:Department {title:'Electronics &
Telecommunication Engineering', field:'Engineering'})
CREATE (IT:Department {title:'Information Technology',
field:'Engineering'})

CREATE
  (Comps)-[:DEPARTMENT_IN]->(Engg),
  (Mech)-[:DEPARTMENT_IN]->(Engg),
  (Extc)-[:DEPARTMENT_IN]->(Engg),

  (Etrx)-[:DEPARTMENT_IN]->(Engg),

  (IT)-[:DEPARTMENT_IN]->(Engg)
```

The entire system is deployed on a server from where it will be accessible to the users. We have used the Apache server for demonstration purpose. All the project related files are deployed in the htdocs folder. The protocol for communication with the Apache server makes use of the port number 80 whereas that needed to access Neo4j makes use of port number 7474.

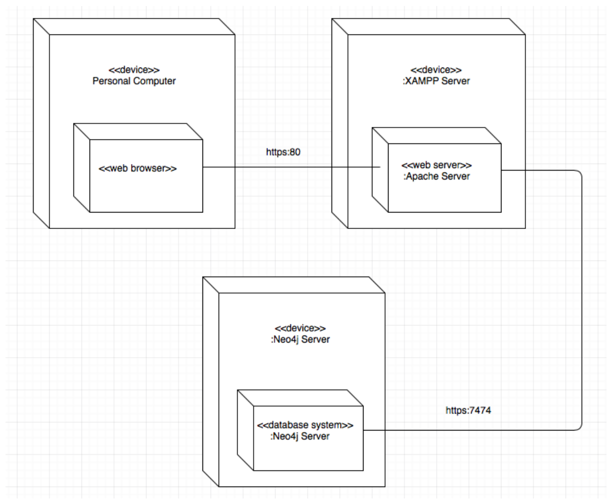


Figure 3 – The deployment diagram for the system

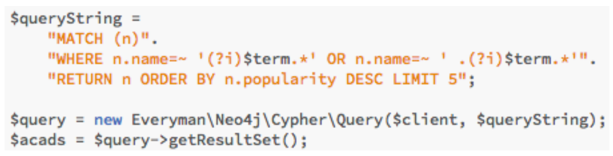
3.2 Algorithm

The information flow takes place form the input search bar to the Neo4j server and back to the input search bar where the results are displayed below it. A couple of more steps follow as far as the algorithm to it is concerned. It is described as –

```
1.  Start.
2.  Ask the user for a text input.
3.  Accept the text input and store it into a variable
    ('term’ in this case).
4.  Pass this term to the search.php page using method
    ‘GET’.
5.  Wrap this into a new php variable (say $term) on
    search.php.
6.  for every letter entered into the search bar i.e. for
    every $term do
7.  fire a query to match $term with the names of the
    nodes present in the graph database.
8.  Get the query result set and store it into another php
    set variable.
9.  Create new php variable (say $result) to store the
    result.
10. for every row in the result set variable do
11. push the matched node name to $result.
12. End for.
13. Throw the result back on the search bar using
    json_encode.
14. End for.
15. End.
```

3.3 Autocomplete process

Once the term is passed to the search.php file every time it is changed i.e. for every letter that a user enters into the search bar a Cypher query is executed that matches it with the node names present in the Neo4j graph database using a regular expression that ignores case sensitiveness [5, 8]. The result of this Cypher query is then stored into a result set which consists of rows with each row containing the matched corresponding node.

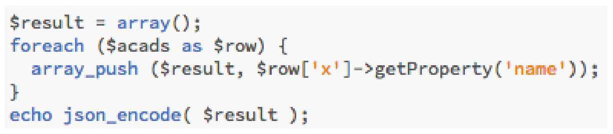


```
$queryString =
    "MATCH (n)".
    "WHERE n.name=~ '(?i)$term.*' OR n.name=~ ' .(?i)$term.*'".
    "RETURN n ORDER BY n.popularity DESC LIMIT 5";

$query = new Everyman\Neo4j\Cypher\Query($client, $queryString);
$acads = $query->getResultSet();
```

Figure 4 – Illustration of the Cypher query used to match the input

A new array variable is the created to store the final result. Using a for loop the node names from each of the rows present in the result set are extracted and pushed into this newly created array and finally the resultant array is json encoded (JavaScript Object Notation) to display the result on the same page where the user currently resides [5].



```
$result = array();
foreach ($acads as $row) {
  array_push ($result, $row['x']->getProperty('name'));
}
echo json_encode( $result );
```

Figure 5 – Extracting name of the matched nodes from the result set

3.4 Test cases

After the successful implementation of the system, it was tested for a few unit test cases as well as system integrated test cases.

Table 1. Unit test cases

Case ID	Scenario	Step	Expected Result	Actual Result
1	Verify that the search bar can accept an input	Click on the search bar and type in something	The text typed appears in the search bar	The text typed appears in the search bar
2	Verify that the search bar accepts all characters	Type in a letter, number or a symbol	All characters should be accepted	All characters are accepted
3	Verify that the XAMPP server is running	Access the web page form a different IP on the same network	The web page should be loaded in the browser	The web page gets loaded in the browser
4	Verify that the Neo4j server is running	Load the connection test php file in a browser	The server settings should be displayed on the browser	The server settings are displayed on the web browser

Table 2. System Integrated test cases

Cas e ID	Scenario	Step	Expected result	Actual result
5	Verify that auto complete results are displaye d	Type in a characte r in the search bar	Matching results to that character should be displayed	Matching results to that character are displayed
6	Verify that the database can be updated	Type in, choose a result and hit enter	The popularity index of the correspondin g node should be incremented by a unit	The popularity index of the correspondin g node is incremented by a unit
7	Verify that the style sheets included get loaded	Access the web page in a browser	All the elements should be properly arranged and oriented	The user interface is just as designed with all elements properly aligned and styled

3.5 Results

The entire system was verified for its functionalities and was successfully deployed. The autocomplete search bar was successfully achieved.

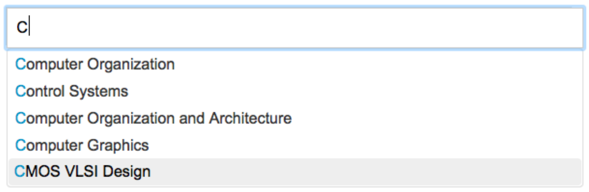


Figure 6 – The autocomplete search bar

4. CONCLUSION

Auto complete and predictive text has become quite necessary in today's digital presence. Whenever a user tries to type in something, he/she wishes that without much effort the exact results be displayed corresponding to what he/she wants. The use of JQuery and AJAX have made this possible as the entire webpage doesn't need to load again and again each time the user enters a character. A simple query is appended to the end of the url and is fired simultaneously. Furthermore, provision of self learning is also practical as the system will constantly keep on adding new input if it were previously unavailable for prediction. Thus the user's history will also be taken into consideration as a part of the auto-complete suggestions.

Along with that the use of graphical database has helped create a more connected database with a hierarchical structure which enables the search to be more refined and deep. Also the use of neo4j made the entire implementation much more easy with user friendly tutorials and smooth implementation. Also Neo4j is easily integrated with many other platforms enables the project to work on multiple and varied platforms without any issues. This helped us reach our conceptual and intellectual milestone we began the development with, in the beginning.

5. REFERENCES

[1] Hongcheng Huang, Chongqing Univ. of Posts & Telecommun., Chongqing, China, Ziyu Dong. Research on architecture and query performance based on distribute graph database Neo4j. Consumer Electronics, Communications and Networks (CECNet), 3rd International Conference. 2013.

[2] H. Komatsu, Tokyo Inst. of Technol., Japan, S. Takabayashi, T. Masui. Corpus-based predictive text input. Proceedings of the 2005 International Conference on Active Media Technology. 2005

[3] Neelu Nihalani, Comput. Applic., UIT, Bhopal, India, Sanjay Silakari, Mahesh Motwani. Integration of Artificial Intelligence and Database Management System: An Inventive Approach for Intelligent Databases. Computational Intelligence, Communication Systems and Networks, CICSYN '09. 2009

[4] Tutorialspoint. JQueryUI – Autocomplete. http://www.tutorialspoint.com/jqueryui/jqueryui_autocomplete.htm

[5] Josh Adell. Neo4jphp. Github. https://github.com/jadell/neo4jphp

[6] C. Irniger, Dept. of Comput. Sci., Bern Univ., Switzerland H. Bunke. Graph database filtering using decision trees. Pattern Recognition, ICPR, 2004.

[7] Sadhana Priyadarshini, Debahuti Mishra. An approach to graph mining using gSpan algorithm, Computer and Communication Technology (ICCCT), 2010.

[8] Volker Pacher, Stack Overflow. Running a case insensitive cypher query. http://stackoverflow.com/questions/13439278/running-a-case-insensitive-cypher-query