Graph Database Indexing Layer for Logic-Based Tree Pattern Matching Over Intensional XML Document Databases

Abdullah Alrefae(✉), Jinli Cao, and Eric Pardede

Department of Computer Science and Information Technology,
La Trobe University, Melbourne, VIC 3086, Australia
afalrefae@students.latrobe.edu.au,
{j.cao,E.Pardede}@latrobe.edu.au

Abstract. Most XML query evaluation approaches are based on the technique of tree pattern query matching (TPQ) to find similar occurrences of the query's path and conditions. Mainly, two types of constraints are matched to evaluate a given query, including hierarchical structure constraints and value-based constraints. However, TPQ technique falls short when it comes to matching the logic-based constraints and non-hierarchical relationships between nodes and entities in the XML document and database. In this paper, we overcome this shortage by providing an abstract graph database layer that provides a logic-based graph relational model to inspect and resolve the logics of the query to choose most relevant nodes in the XML document. Only the subtrees of the relevant nodes will be traversed in the document and the other subtrees will be skipped. We propose the application of graph database as an indexing layer that defines conceptual linking between database entities, beside logic-based assertions and constraints to evaluate XML queries over this layer to find most related entities and traverse only their related nodes in the XML document. In addition, we propose a mapping criteria and algorithm between XQuery and Cypher, which is a query language for Neo4j graph database.

Keywords: Intensional XML · Tree pattern query (TPQ) · Graph database
Logic-based constraints · Neo4j

1 Introduction

Most of the XML query evaluation approaches are based on the technique of tree pattern query matching (TPQ) to find similar occurrences of the query's path and conditions. Typically, there are two types of constraints to be matched while traversing an XML document. First, match the hierarchical structure in the path expression, such as //A[/B]//C, which defines all nodes *C* under node *A* with at least one node *B* as child. In this example, the structural relationship between *A* and *B* is a parent-child (P-C) relation, while the structural relationship between *A* and *C* is ancestor-descendant (A-D) relation. Second, match the simple value condition, such as //A[/B]//C[X = y], which defines the node C with the aforementioned structure, and satisfies the predicate of having an element X with value y [1].

O. Gervasi et al. (Eds.): ICCSA 2018, LNCS 10963, pp. 577–588, 2018.
https://doi.org/10.1007/978-3-319-95171-3_45

A number of proposed techniques to improve the efficiency of traversing XML documents exploit the indexing of regional encoding scheme (described in Sect. 2.2, below) to optimize the evaluation of tree pattern matching process [2]. These techniques can be categorized based on their fundamental aims and methodology into two types. First, optimization by minimizing the size of the tree pattern into a small Twig pattern by using stack indexing algorithms, such as TwigStack [3] and Holistic Twig Joins [4]. Second, optimization by conducting a Global query pattern tree (G-QPT) based on collecting possible ordered pattern trees for a given query into a single rooted general query [5]. In general, most of these techniques proposed different approaches to improve the process of matching the hierarchical structure linking between nodes, and evaluating queries based on node-names and static values defined in the predicates [6].

However, in real-time systems that store their operational data in XML format, the predicate values can be stored as URI collections that need to be materialized on demand to overwrite its content, i.e. after receiving a related query, to find the most recent values [7]. This URI collections (a.k.a. intenstional nodes) may need to extract data from decentralized data sources that is prompted by the infrastructure of service-oriented systems and cloud computing [8].

Problem Definition: The implementation of traditional TPQ matching process will be impractical for documents with such URI collections. The difficulty is due to the absence of simple values in predicate nodes, the variation of the external sources to be invoked, and in some cases the enormous number of value nodes to be materialized in large document.

Assume the following example, in Fig. 1, we show a sample tree of an XML document for hospitals in Melbourne, Australia. This document is part of a proposed Intensional XML-enabled Web-based Real-time decision support system [9]. Let's say we want to check the current availability of Emergency Departments (ED), to help in workload distribution or reduce congestion in emergency departments and avoid overcrowded hospitals. The following XPath expression will be part of the query.

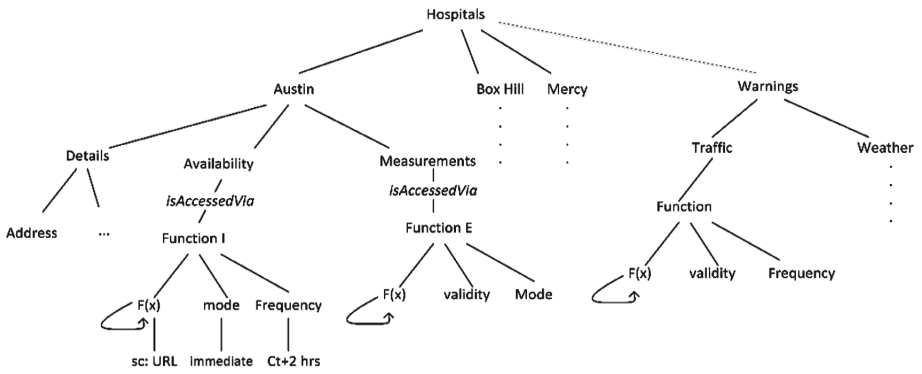

Fig. 1. Tree representation of an XML document with intensional data parts

doc("hospitals.axml")/hospitals/hospital[availability="available"]/measurement (1)

It is impractical to use most of the existing indexing approaches, that rely on key values, to index documents with intensional parts. The values for these parts, such as availability, are changeable or unavailable.

In addition, with data sizes incrementally increase and with more demand for real-time query answering, these intensional parts start to appear everywhere in the document. Therefore, the process of materializing all the intensional parts in the document, to prepare the document for the traditional TPQ matching traversing, will include many unnecessary materialization processes for unrelated nodes. Practically, we need to minimize the number of intensional nodes that will be materialized.

Proposed Solution: We propose the addition of logic-based relationships between database entities. The definition of assertions and constraints to build the logic-based relationships can vary from one database to another based on content type.

For instance, in our predefined hospitals database, assume the following XQuery, that is derived from the predefined XPath expression in (1), asking for the available hospitals:

```
let service Find-alternative($x) be
for $ed in doc("hospitals.axml")//Hospitals,
    $sub in $ed(name=$x)/suburb
where contains ($ed/support, $sub) and
        $ed/Availability=available and
        $ed/Measurements/LWBS<3 and
        $ed/Measurements/LOS<2
return <f>{$ed/name}</f>
```

In processing such query, we need to consider two main constraints: (1) Location-dependent constraint, i.e. alternative emergency department (ED) need to be in the nearby hospital; and (2) Time-dependent constraint, i.e. real-time measurements in the alternative emergency department need to be currently below predefined risk levels.

Traditionally, such queries are processed with the common TPQ techniques, which depend on the traversing of the document's tree, based on tag-names and values defined in the query. However, such hierarchical traversal will not consider the time and location constraints. The process of extracting the constraints' values from the query, based on predefined rules and assertions, will be described in the following sections.

In addition, we propose an abstract graph database layer that provides a relation-based indexing model to inspect and resolve the assertions and logics of the query and come up with most relevant nodes (subtrees) in the XML database to process the TPQ matching over them.

The remaining parts of this paper will be as follow. Preliminary techniques and concepts in relation to our proposal will be defined in the next section. The third section

will represent our proposed approach of graph database layer, and other related evaluation techniques that include conceptual data linking and logic-based assertions and constraints. Finally, section four concludes our paper. It will summarize the proposed solution of this paper and discuss the research issues for future directions.

2 Preliminaries

XML document database consists of multiple hierarchically structured, unranked, node-labeled trees to form a forest, where each tree represents a document. To facilitate the discussions of proposed solutions, the related concepts and techniques are presented in this section.

2.1 Intensional XML Document (Active XML)

The form of XML documents with URI collections, that define service calls to the sources of these parts of the document, has been defined as part of the so-called Active XML (AXML for short) [10, 11]. AXML is a markup language framework that is based on AXML documents and AXML services. AXML document is an extension of XML document, with embedded calls (AXML subtree) to interrogate web services, beside other parts of the data that are defined explicitly in the documents as in normal XML documents. AXML is a useful solution for integration of data and services over peer-to-peer environment because the embedded service calls in AXML documents will give the flexibility to authorize distributed systems cooperation and find other sources of data at run time (see Fig. 2).

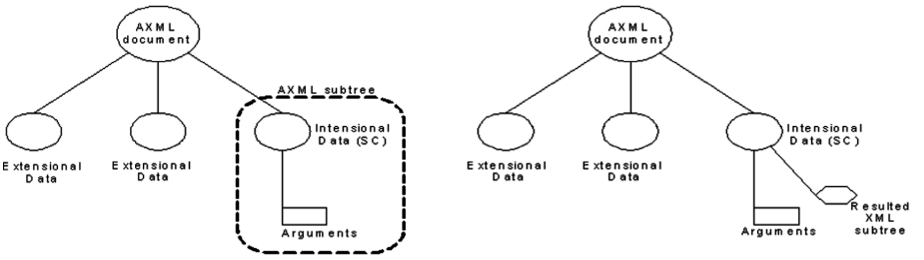

Fig. 2. AXML document before materialization (left) and after materialization (right)

The AXML framework has the following characteristics [12] that differentiate it from normal XML system and make it powerful project for data integration:

- The possibility to manage calls activation and the lifespan of the resulted data: by appending some parameters to determine *when* the service call should be activated and for *how long* the results must be considered valid.
- Support the services with intensional data: by providing AXML services that accept data with intensional input/output data.

– Allow continues services: by supporting the use and creation of Web services that may return a stream of answers, when such action is required.

These advantages of AXML framework promoted the adoption of its documents as metadata format in our proposed web-based real-time decision support system [13]. However, the nature of AXML document that stores service calls instead of static values invalidates the implementation of TPQ matching technique to traverse such documents, which we address in this paper.

2.2 Tree Pattern Matching Techniques

Most of recent proposed techniques to improve the efficiency of TPQ matching over XML databases share the following techniques to build their proposed approaches and define their rules.

Document Encoding and Position Representation of Occurrences of Elements and String Values in XML Database [3]. It is a join-based matching technique that depends on numerical annotation of elements in the XML tree. The numbering methodologies vary in position details provided for each element. In general, they show the regional encoding as a 3-tuple (DocId, LeftPos:RightPos, LevelNum). These encodings represent the flowing [14]:

DocID: Document's identifier
LeftPos: Counting number of the start word of the element, from the beginning of the document
RightPos: Counting number of the end word of the document, from the beginning of the document
LevelNum: Nesting depth of the element in the document

Recording these position details of elements facilitates the process of defining structural relationships between tree nodes [15]. For instance, for a node pair (a, d), node *d* is a descendent of node *a*, if and only if, $a.LeftPos < d.LeftPos < d.RightPos < a.RightPos$. This numbering scheme can also be a foundation for other techniques, such as inverted list and index structure, as shown below.

Inverted Lists Model. After annotating elements in XML document with their regional encoding, we can build an inverted list based on document's tags, with one sequence for each document tag, to reflect the regional encoding details for each element [16].

Index Structure. It is a part of the inverted list that defines the regional encoding for value nodes to satisfy a particular predicate [16].

The implementation of these techniques varies from one approach to another to according to indexing structure and matching format. Some approaches proposed to index XML document into join stacks or sequences. They map the XML document into stacks or B^+ tree element lists, respectively, to narrow the query matching on related stacks or subsequences [15]. However, these structural summaries only represent the encoding positions of elements as in B^+ -tree, and the connection of entities based on

their hierarchical structure as in stack-assisted matching. They do not consider the logical connections between entities, which we propose in this paper.

2.3 NoSQL Graph Database

NoSQL database can generally be categorized into four types, as explained in Table 1, below [17].

Table 1. NoSQL databases types.

NoSQL Database	Storing methodology	Example
Key-value stores	Store every node as a key with its value	Riak, Voldemort
Column Family\ BigTable Clones	Store data in columns to query large datasets	HBase, Hypertable
Document databases	Store every record with its data and complex data structure as a document with an associated key	CouchDB, MongoDB
Graph databases	Store information as a connected network of data	Allegro Graph, Neo4j

Because of its powerful, agile, and flexible data model, NoSQL provides solutions to most of the relational databases' issues [17]. NoSQL data model capabilities include, the manipulation of large data sets with different structures, faster schema iteration, frequent code pushes, and better treatment of geographically distributed resources.

Graph database, in particular, provides a powerful solution to overcome data complexity issues (see Fig. 3).

To improve the efficiency of QTP matching over intensional XML database, we propose a graph database layer as covering index for element nodes. It defines structured conceptual linking between nodes based on their logic-based relationships, as will be demonstrated below.

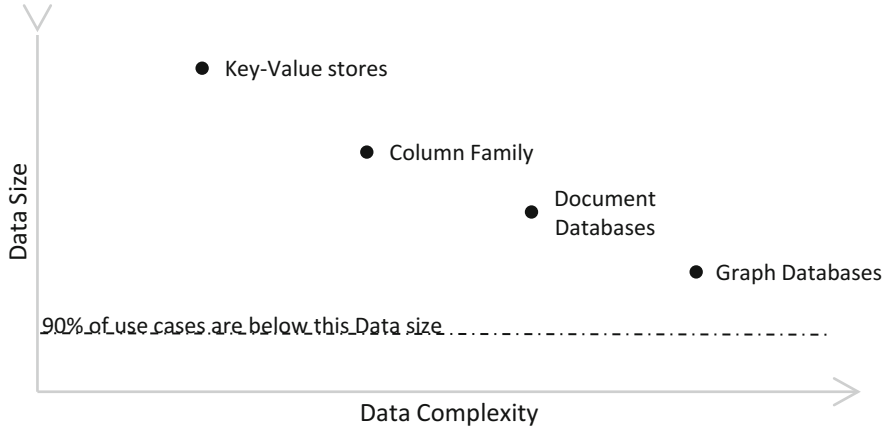

Fig. 3. NoSQL data models, in regards to data complexity and data size [18].

3 Graph Database Abstract Layer

In this section, we propose the application of logic-based definitions and connections in a form of a graph data model on top the XML data model (tree model). By using the model, we apply queries on the graph data model and filter out the most related tags in XML documents. Then we apply the query only on these matching XML tree nodes.

The application of logic-based graph database as a covering index layer empowers the system with a number of advantages, including the following.

- Schema matching is unnecessary, as graph database is schema free or schema optional. This advantage gives the system the required agility to change. Include, add or remove properties to nodes on the fly, without requirement to update other nodes to include the new properties.
- The ability to query high complex data, better than traditional TPQ or any other NoSQL query techniques.
- Provide a real relational database, as it connects any relational nodes from any level or type, in the form of triples (subject, object, predicate).
- The data mode is very intuitive, and the query processing speed is fast.

The graph database model maintains good performance with complex data sets, like social networks. Nevertheless, it can also be a promising solution for other applications, such as fraud detection, and our application of real time recommendations.

3.1 Linked Data Model

The graph database model represents data in triples (subject, object, and predicate) where each of the subject and object can represent a different entity or resource, while the predicate represents the relationship between them. This mechanism of linking data located in different resources builds a linked data model that can help in decision making process by manipulating the relationships between the different resources.

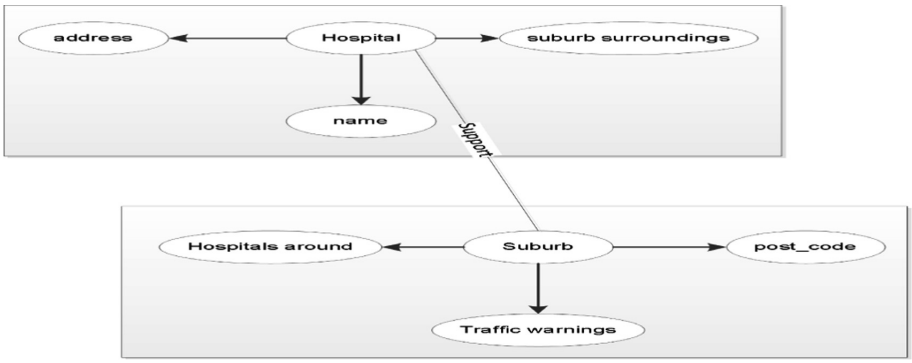

Fig. 4. Conceptual linking in a linked data model

In Fig. 4, we show a conceptual linking example between two different tags, where the hospital, suburb and support respectively represent the subject, object, and relationship type in triple.

3.2 Logical Assertions and Constraints Definition

To build a linked data model between different tags in a database, we need to define assertions and constraints that define the logic-based relationships between tags.

For our emergency department's example, we can define the following functions and axioms to organize the relationships between database elements: (1) Domain: Hospitals (Emergency departments); (2) Unary predicates: Hospital(x), Suburb(x); (3) Relationships: nearby(x, y), adjacent(x, y), support(x, y); and (4) Functions: alternate(x), can_replace(x, y).

In our example, the domain of interest is hospital emergency departments in Melbourne (Victoria, Australia). Our master data include, Hospitals and Suburbs, therefore we define these entities as unary predicates (Hospital(x) and Suburb(x)). Types of connections that have been used to build the graph database for our system include, nearby(x, y) where x and y are hospitals and this relationship represents a connection between geographically nearby hospitals, as shown in Fig. 5.

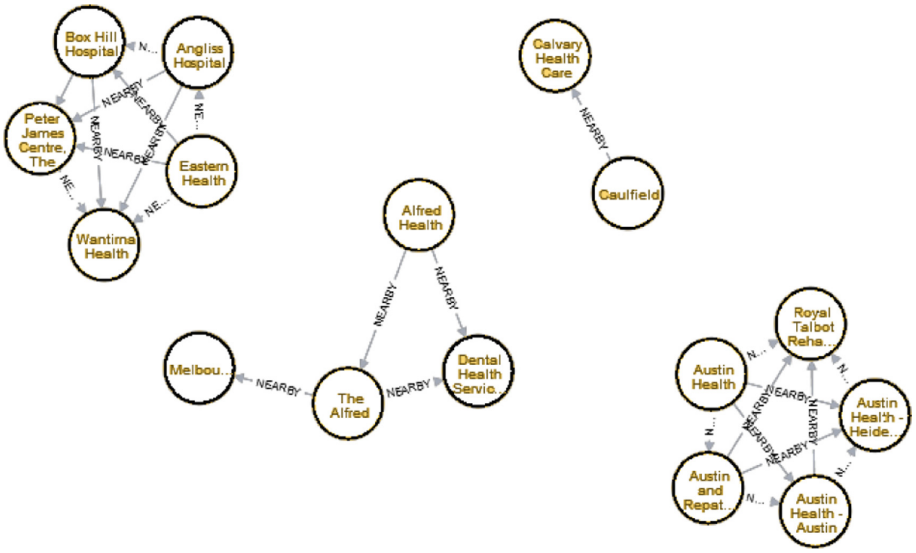

Fig. 5. Nearby relationships between hospitals

The second type of relationships is adjacent(x, y), were x and y represent suburbs, and this connection defines geographically adjacent suburbs.

The other relationship type is support(x, y), were x represents hospital and y represents suburb that is supported by the hospital x, because x is the closest hospital(s) to the suburb y, as shown in Fig. 6.

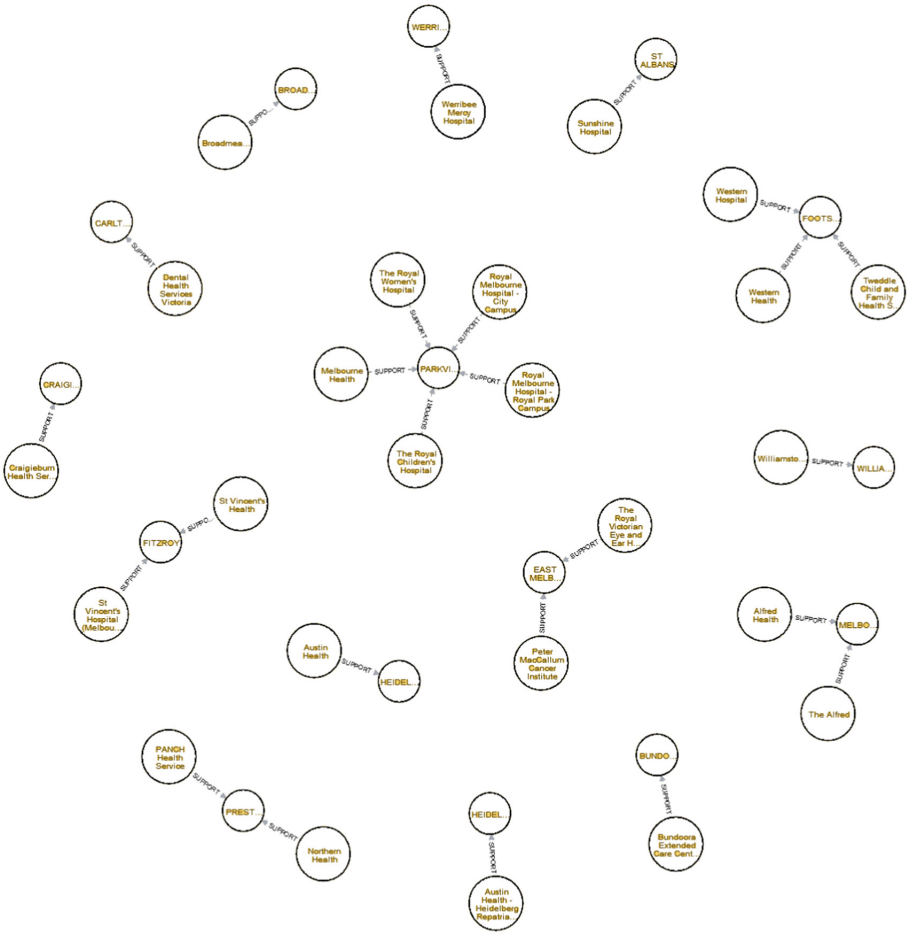

Fig. 6. Support relationships between hospitals and suburbs

After the graph database is built, based on the aforementioned predicates and relationships, we can define a number of axioms to find real-time recommendations and answer logic-based queries. Axioms can include:

$$\text{Axioms} : \forall x, y \, alternate(x) = y \leftrightarrow hospital(y) \wedge nearby(x, y) \tag{2}$$

$$\text{OR} \, \forall \, x, y \, \text{can\_replace}(x, y) \leftrightarrow x \neq y \wedge \exists p \, support(x, \, p) \wedge support(y, p) \tag{3}$$

Axiom (2) defines an alternative for hospital x, if hospital x is unavailable for any reason (e.g. overcrowded) and need to find best possible recommendations for alternatives, and axiom (3) has almost a similar purpose, but with the use of different relationships.

To answer queries and recommendation requests based on these functions and axioms, we need to extract the required details from the original XQuery as follows:

```
let service Find-alternative(Austin) be
// variable $x from general query has been declare
for $ed in doc("hospitals.axml")//Hospitals,
    $sub in $ed(name=$x)/suburb
where contains ($ed/support, $sub) and
        $ed/Availability=available and
        $ed/Measurements/LWBS<3 and
        $ed/Measurements/LOS<2
return <f>{$ed/name}/f>
```

(4)

Variables and predicates can be extracted to build an equivalent graph database query. Below, we present an equivalent Cypher query that is used to query a graph database called Neo4j:

```
Match (x:Ed {name:"Austin"})-[:SUPPORT]->(s:Suburb)<-[:SUPPORT]-(h:ED)
RETURN DISTINCT h.name;
```

(5)

As we can see in the two equivalent queries, the variable declaration in the XQuery (Austin, in a circle) has been extracted and used in the Cypher query as a start node. The node tag in the XQuery (Ed, in a solid square) has been used in the Cypher query as a label name. In addition, the property name (support, in dashed rectangle) has been extracted and used in the Cypher query as a predicate, i.e. relationship type.

In general, Table 2 shows the mapping criteria between XQuery for XML databases and Cypher for Neo4j graph databases.

The Cypher query (5) is equivalent to the XPath expressions in the XQuery (4). It is a replacement of the traversal process. The resulted nodes will then be the only subtrees to be checked for other predicates in the XQuery.

Table 2. XQuery and Cypher mapping criteria

XQuery	Cypher
x: variable name	Start node
Node tag or label	Label name
Property node	Relationship (predicate)

For the automation of the mapping, we defined an algorithm to transform XQuery to Cypher, as follows.

Mapping Algorithm

```
1. for (variable (X)                 //variable_name
        of type(T)                   //node_tag
        and constraint (C)           //property Condition
        with value(V)                //property value
2.    do (match ((X)
        of lable(T)
        with relation(C)
        to (V)
        and (Y)
        of label(T)
        With relation(C)
        to (V))
3.        Return Y))
4. Output: List of [Y]
```

In the evaluation of the predefined Cypher query (5) on the graph database, it has returned three nodes (Mercy Public Hospital, St. Vincent's Health, and Western Health) in 122 ms.

Consequently, only these three nodes will be traversed in the XML database and the predefined conditions in the XQuery (4) will be applied on these nodes to match the most relevant node after materializing the corresponding intensional web services.

4 Conclusion

This paper presented logic-based graph database indexing for XML database. It overcomes the issue of processing TPQ matching on large XML databases with intensional nodes that required to be materialized before processing the query.

We have defined the conceptual linking methodology that is based on the triples data model (subject, object, predicate) to build the graph database. In addition, we defined the logical assertions and constraints to organize the connections between the database's entities.

The contributions of the paper also include the proposal of the mapping techniques and the general algorithm to extract the logical assertions and constraints from the XQuery and build the relevant graph database's query. The example of Melbourne metropolitan's hospitals has been presented to evaluate our proposed technique.

Acknowledgments. This research paper is part of my research candidature, which is financially sponsored by the Ministry of Higher Education in Saudi Arabia.

References

1. Lu, J., Ling, T.W., Bao, Z., Wang, C.: Extended XML tree pattern matching: theories and algorithms. IEEE Trans. Knowl. Data Eng. **23**(3), 402–416 (2011)
2. Hachicha, M., Darmont, J.: A survey of XML tree patterns. IEEE Trans. Knowl. Data Eng. **25**(1), 29–46 (2013)
3. Bruno, N., Koudas, N., Srivastava, D.: Holistic twig joins: optimal XML pattern matching. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, pp. 310–321 (2002)
4. Jiang, H., Wang, W., Lu, H., Yu, J.X.: Holistic twig joins on indexed XML documents. In: Proceedings of the 29th International Conference on Very Large Data Bases, vol. 29, Berlin, Germany, pp. 273–284
5. Chen, Y.: Discovering ordered tree patterns from XML queries. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 559–563. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24775-3_66
6. Zhang, N., Kacholia, V., Ozsu, M.T.: A succinct physical storage scheme for efficient evaluation of path queries in XML. In: Proceedings of the 20th International Conference on Data Engineering, Boston, MA, USA, pp. 54–65
7. Milo, T., Abiteboul, S., Amann, B., Benjelloun, O., Ngoc, F.: Exchanging intensional XML data. ACM Trans. Database Syst. **30**(1), 1–40 (2005)
8. Demirkan, H., Delen, D.: Leveraging the capabilities of service-oriented decision support systems: putting analytics and big data in cloud. Decis. Support Syst. **55**(1), 412–421 (2013)
9. Alrefae, A., Cao, J.: Intensional XML-enabled web-based real-time decision support system. In: 2017 International Conference on Computing Networking and Informatics (ICCNI), 29–31 October 2017, pp. 1–10 (2017)
10. Abiteboul, S., Benjelloun, O., Milo, T.: The active XML project: an overview. VLDB J. **17**, 1019–1040 (2007)
11. Phan, V.B., Pardede, E.: Active XML (AXML) research: survey on the representation, system architecture, data exchange mechanism and query evaluation. J. Netw. Comput. Appl. **37**(1), 348–364 (2014)
12. Milo, T.: Peer-to-peer data integration with active XML. In: Grumbach, S., Sui, L., Vianu, V. (eds.) ASIAN 2005. LNCS, vol. 3818, pp. 11–18. Springer, Heidelberg (2005). https://doi.org/10.1007/11596370_2
13. Alrefae, A., Cao, J.: Web-based real-time decision support system active XML-based metadata. In: 2014 Global Summit on Computer & Information Technology (GSCIT), 14–16 June 2014, pp. 1–4 (2014)
14. Lu, J.: XML Tree Pattern Processing. In: An Introduction to XML Query Processing and Keyword Search, pp. 90–156 (2013). 4.2 XML structural join
15. Moro, M.M., Vagena, Z., Tsotras, V.J.: Tree-pattern queries on a lightweight XML processor. In: Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, pp. 205–216
16. Wu, X., Theodoratos, D., Hui Wang, W., Sellis, T.: Optimizing XML queries: bitmapped materialized views vs. indexes. Inf. Syst. **38**(6), 863–884 (2013)
17. NoSQL Database Explained. MongoDB, Inc. https://www.mongodb.com/nosql-explained
18. Webber, J., Robinson, I.: How graph databases relate to other NoSQL data models