IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 13, 2020, accepted September 3, 2020, date of publication September 23, 2020, date of current version October 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3026006

XML-REG: Transforming XML Into Relational Using Hybrid-Based Mapping Approach

EMYLIANA SONG AND SU-CHENG HAW, (Member, IEEE)
Faculty of Computing and Informatics, Multimedia University, Cyberjaya 63000, Malaysia
Corresponding author: Su-Cheng Haw (sucheng@mmu.edu.my)

This work was supported in part by Telekom Malaysia (TM) Research and Development from TM, Malaysia.

ABSTRACT eXtensible Markup Language (XML) is one of the most used standards for information sharing between applications and devices, both on the internet and local network. However, relational database (RDB) has been used by many enterprises as their data management system and will require an amount of cost to change the system completely, if they are to change to XML technology solely. Thus, a mapping scheme is required to provide seamless integration on bridging XML technologies and RDBs. In this paper, an efficient model-based mapping scheme named XML-REG is proposed. The XML document will first be read and parsed into the parser, namely Streaming API for XML (StAX) parser. Then, each node will then be assigned with unique identification label to show the exact position of nodes in the document. Subsequently, by employing the proposed algorithm, data will then be transformed into tables in the RDB storage. As the result, two tables, namely (i) value table to store information carried by text node of the document, and (ii) path table to store the hierarchy structure of the document will be created. Experimental evaluations demonstrated that XML-REG outperformed some existing approaches, such as Mini-XML, XAncestor, XMap and XRecursive in terms of data storage size, mapping time and query retrieval time. In addition, the scalability test has also been conducted to show the capability of these approaches in supporting huge datasets, by scaling the DBLP dataset by times 5, times 10 and times 15. The results showed that XML-REG has the closest to linear graph compared to other existing approaches. On average, XML-REG showed the best performance in terms of query retrieval time and database storage size.

INDEX TERMS XML mapping, XML to RDB, XML database, XML labeling, model-based mapping, XML transformation, XML extraction.

I. INTRODUCTION

eXtensible Mark-up Language or better known by its abbreviation, XML, is a mark-up language that allow data transferring from one platform such as database and website to another platform. This is achievable as it is cross-platform in nature and allows XML to be able to bridge differences in system and devices. XML is widely applied in web services and on the Internet, it is used to store, carry and transport data. The data carried in XML document is separated by the start tag <> and end tag </ >, which define when the element begins and ends. The data carried by the element are commonly known as text.

XML mapping is a technology that used in transforming XML data into any other format as the underlying storage. Among some of the existing database technologies are relational database (RDB), object-oriented database, object-relational database and Not Only SQL (NoSQL) database. Among these databases, RDB is still the popular storage. With the emergence of cloud computing, RDBs are still the back-end architecture of cloud computing architecture. RDB has been and still widely used in many organizations, thus, these organizations require an effective mapping scheme to transform XML into RDB storage.

There are two types of mapping techniques, which are structural-based mapping (schema-based mapping) and model-based mapping (schema-less mapping) [1]. The main difference between these mapping choices is the existence of XML Schema (XSD) or Document Type Definition (DTD) to help define the structure of the document. Structural-based mapping approach requires existence of DTD to transform XML document into RDB storage. Nevertheless, DTD file are not usually provided along with the XML document. Thus, if user wants to run a mapping with DTD file, ones need

The associate editor coordinating the review of this manuscript and approving it for publication was Genoveffa Tortora.

to create it. Creating a new DTD file is often complicated and requires skills. This will create additional complexity when managing various types of XML documents [2], [3].

Meanwhile, DTD is not required for the model-based mapping as it can define document structure of XML document independently, based on the respective model constructed. Using this approach, the XML are mapped to some fixed relational schemas.

XML can be categories into data-centric or document-centric. Data-centric consists of highly structured content, and the meaning of value depends on the structured data represented in it. It is often used for data exchange purpose, transferring data from one system to another. This type of XML is commonly found in enterprise applications. To give an example, enterprises usually have data on sales orders, flight schedules, stock quote and sometimes, scientific data analysis. Aside from that data-centric, another category of XML is document-centric. This type of XML is loosely structures; it contains a large amount of text. Examples are legal document, product catalog or news like CNN RSS Feed. For this research, data-centric XML is given priority as we mainly focus on elevating productivity of enterprises in term of data management and analysis.

The ability on processing semi- structured, unstructured, and structured data is vital as well as extracting information from the data. As for XML, although native XML databases are present, cost of shifting between database management hardly made it as come-and-go category. The primary goal of XML databases is to enable XML content to be stores and retrieved as per requested by users.

In recent years, there are many authors that kept improving existing XML-RDB mapping algorithm in term of storing method and labeling system. Storing methods of an algorithm will give huge impact on how query retrieval needs to be structure and the time taken to process each query. Common improvement of the proposed algorithm is on how the data is stored and the technique used. Most of the time, authors use existing labeling system to label the nodes in the document, this gives author limited improvement on the mapping algorithm. For this research, we target to achieve faster storing time, faster query retrieval time and yet able to support dynamic updates effectively.

The simplicity of XML syntax enables both human and machine to understand the language easily. XML encodes data in plain text format, which gives the advantage of platform independence, bridges changes of format for different computer system. Flexibility of XML has also benefitted to the data sharing process, whereby, the data are transported without losing any descriptive information. Nevertheless, the recommendations of W3C to employ XML as a standard across the internet have brought challenges in data processing. As such, integration of XML into various formats is essential through the mapping scheme.

Figure 1 shows a sample of the XML document that will be used throughout this paper for data representation and

```
<root>
 <listing>
  <seller_info>
     <seller_name>jenzen12 </seller_name>
     <seller_rating>new </seller_rating>
  </seller_info>
  <payment_types>Accepts Credit Cards (MC, VISA)
  </payment_types>
  <shipping_info>* Buyer Pays Shipping * Seller Ships on Payment
  </shipping_info>
  <buyer_protection_info> Standard Protection Coverage
  </buyer_protection_info>
  <auction_info>
    <current_bid>$310.00 </current_bid>
    <time_left>4 days </time_left>
    <high_bidder>
       <bidder_name>mike_and_colleen </bidder_name>
       <bidder_rating>1 </bidder_rating>
    </high_bidder>
    ...
```

FIGURE 1. Sample of XML document.

experimentation. This document is extracted from parts of yahoo dataset that is obtained from XML Repository [4].

II. REVIEW ON EXISITNG APPROACHES

There are four types of mapping approaches, edge-based mapping scheme, node-based mapping scheme, path-based mapping scheme and hybrid-based mapping scheme [5]. Edge-based mapping scheme is the simplest yet modest technique. The edges of the XML tree will be mapped into single table. Designated tables store information of document by using node identifier, source and target to get the edge label between nodes. The drawback of this technique is huge storage space is required as all the document edges are stored in a single table. This will be incurred higher query processing time, especially in retrieving complex queries. Excessive self-joins are required, which is the most expensive operation in RDBMS [5].

Path-based mapping scheme tracks the hierarchical structure of document by tracking the trail of node to node. This labelling approach utilizes path table. The path store is often divided into two, namely (i) root to non-leaf node and (ii) root to leaf node. With storing node information into a path table, it is able to reduce the search space of node when it comes to query retrieval process. This technique can be divided into two sub-categories: (i) root to node, and (ii) root to leaf node.

For the first sub-category, the path expression of nodes is stored in a table, while the node information of the document is stored in another table. The second sub-category will store path expression in one table, while storing only the leaf node information in the other table. For this approach, the information of inner node is not significant, and thus, it is not stored.

Node-based mapping scheme allocates an identifier to indicate the absolute position of the node in the document. This technique normally uses high storage space, which may cause the column size to appear overhead for some label in the RDB. By looking at the positional identifier, the hierarchical relationships between pair of nodes can be determined easily. Nevertheless, for complex queries (consists of at least

a branching edge), the query response time will be longer as the structural join will be performed based on pair of nodes.

Hybrid-based mapping scheme is the combination of two or more of any techniques. For example, XParent [6] and XPEV [7] combine both edge and path-based techniques. All edge between nodes and path information are stored into separate table. Containment relationship is utilized by these approaches to preserve node relationships. Obvious drawback of this technique is it requires huge storage space to store all the edge information. Nevertheless, with the employment of path table, the query response time is expedited by reducing the query search space [5].

The following sub-sections elaborate some of the recent mapping schemes in detail.

A. MINI-XML MAPPING APPROACH

Mini-XML is a path-based mapping scheme that reduces data redundancy by storing leaf node separately from the data table [8]. The Document Object Model (DOM) parser is used to check well form-ness of XML document; that is the document must follow syntax rules to be identify as well formed for the parser. However, DOM Parser Application Programming Interface (API) in Java provides function to traverse input of the XML file and creates DOM object corresponding to the nodes. The object is stored into memory, resulting in longer time and memory space requires torun larger dataset. Nevertheless, this parser allows users to navigate nodes in the document back and forth. The nodes are then annotated using first version Persistent labelling scheme [9] and traverse the XML tree in depth first manners.

Zhu *et al.* [8] compared their proposed approach against s-XML [10] using six datasets with size ranging from 2.2MB to the largest by 683 MB. The experimental results revealed that Mini-XML successfully reduce storage time and space; it is attainable because Mini-XML avoids data redundancy. In terms of storage time, Zhu *et al.* stated that s-XML uses more fields that stores duplicated information of nodes. As such, as the dataset grow larger, efficiency of Mini-XML become dominant as compared to s-XML.

Storage space that used by Mini-XML are comparably lesser than s-XML. This is because Mini-XML keeps only crucial path information and fields that are sufficient to identify the relationship of nodes. Yet, data redundancy still occurs when the same path expression is stored into path table with different node position. Figure 2 illustrates the data model annotated for Mini-XML with node annotation that describes each position of node.

Table 1(A) and Table 1(B) depict the partial data of path table and leaf table respectively. Path table stores only unique path on the three attributes (PathId, Path and Pos of non-leaf nodes). Each path has its own unique id (PathID) and the position of the element node (Pos). The path expression (Path) defines the top-down hierarchy of data from the root to its respective node.

Table 1(B) depicts the leaf table, which stores the value of all the leaf nodes. The table contains four attributes, namely

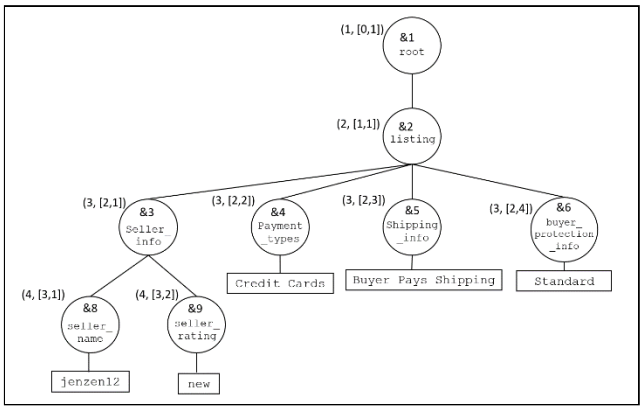

FIGURE 2. Data model labelled based on Mini-XML approach.

TABLE 1. (a) Partial view of path table. (b) Partial view of leaf table.

(a)

PathId	Path	Pos
1	./root	(1,[0,1])
2	./root/listing	(2,[1,1])
3	./root/listing/seller_info	(3,[2,1])
7	./root/listing/auction_info	(3,[2,5])
12	./root/listing/auction_info/high_bidder	(4,[7,3])

(b)

LeafId	name	Value	Pos
4	payment_types	Accepts credit cards (MC, VISA)	(3, [2,2])
5	shipping_info	*Buyer Pays Shipping* Seller Ship on Payment	(3, [2,3])
6	buyer_protection_info	Standard Protection Coverage	(3, [2,4])
8	seller_name	Jenzen12	(4, [3,1])
9	seller_rating	New	(4, [3,2])

(i) leaf id, which is the unique self id of nodes, (ii) node name, which is the name of the node, (iii) value, which is the text value, and (iv) pos, which is the position of the node in the document. Each node pos is composed of node level, parent id and self id of node among the siblings.

Mini-XML stores all nodes and its precise position in the document into path and value table. Unlike approaches that will be discussed later, each element node with different position label will be kept into path table, thus it increases the size of space consumption. Other than that, it resulted in longer query retrieval time compared to other approaches.

B. XANCESTOR MAPPING APPROACH

Qtaish and Ahmad [11] proposed XAncestor, an approach that consists of three main components, namely (i) fixed RDB scheme, (ii) XtoDB mapping, and (iii) XtoSQL query processing algorithm. The fixed RDB scheme is designed with the aim to store the relations optimally. The XtoDB mapping component, maps XML document into RDB scheme. Prior to the mapping process, a DOM parser API is adopted to validate the document. The data then stored into two tables.

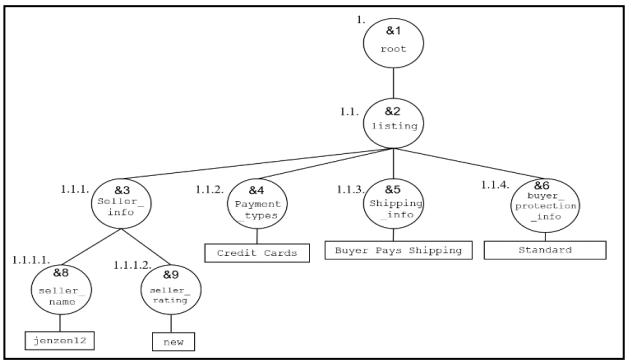

FIGURE 3. Data model labelled based on XAncestor approach.

The uniqueness of XAncestor is that it manages path of leaf node in a pre-defined RDB scheme (also known as fixed RDB scheme). This reduces the size of storage consumption when the dataset is huge. The third component, XtoSQL, is a query processing algorithm to transform XPath queries into SQL queries representation to retrieve relevant results against the pre-defined RDB scheme.

The experimental comparison results on XAncestor against other approaches, XRel [12], SMXR [13], approach proposed by Ying *et al.* [14], XRecursive [15] and s-XML [10], show that XAncestor uses the least storage space and time. For instance, the storage space consumed by XAncestor is half of the storage space as compared to s-XML approach. For complex query processing, XAncestor achieve the best results. For simple query search, it is observed that approach proposed by Ying *et al.* [14] is compatible with XAncestor. However, as the query complexity increases, the difference can be seen between the two approaches. Figure 3 shows a data model using XAncestor annotation. This approach implements Dewey order labelling [16] to annotate each node.

Table 2(A) and Table 2(B) depict the resulted tables using XAncestor mapping approach. XAncestor stores the data into two tables, namely Ancestor_Path and Leaf_Node. Table 2(A) represents the Ancestor_Path_Table, whereby this table shows unique path for every node in the path expression. This table consists of Ancestor_PathID that shows unique ID of path expression in Ancestor_PathExp.

Table 2(B) shows the Leaf_Node table. This table is made up of four attributes, which is Node_Name, Anc_PathID, Ancestor_Pos and Node_Value. However, information of the inner node is not stored in this approach.

XAncestor approach able to reduce storage space as it only stores inner nodes and query retrieval time able to be lessen by utilizing path expression in path table. Nevertheless, the column Ances_Pos requires to retrieve parent node via level using recursive join.

C. XMAP MAPPING APPROACH

Bousalem and Cherti [3] proposed XMap, which utilize the ORDPATH labelling scheme [17] to annotate the data in

TABLE 2. (a) Partial view of Ancestor_Path Table. (b) Partial view of Leaf_Node Table.

(a)

Ancestor_PathID	Ancestor_PathExp
1	/root/listing
2	/root/listing/seller_info

(b)

Node_Name	Anc_pathID	Node_Value	Ances_Pos
payment_types	1	Accepts credit cards (MC,VISA)	(1.1.2)
shipping_info	1	*Buyer Pays Shipping* Seller Ship on Payment	(1.1.3)
buyer_protection_info	1	Standard Protection Coverage	(1.1.4)
seller_name	2	Jenzen12	(1.1.1.1)
seller_rating	2	New	(1.1.1.2)

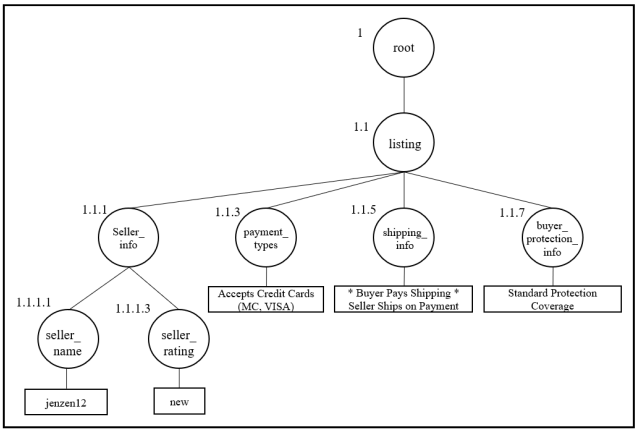

FIGURE 4. Data model labelled based on XMap approach.

XML document. The authors categorized the method into three main components as follows: (i) XML to RDB, (ii) Translate the XML Query to SQL query, and (iii) Reconstructs XML document from RDB by transforming from the SQL result.

The authors have not done any experimental comparison against related approaches. Instead, the author compared the efficiency of using DOM parser and Simple API for XML (SAX) in their evaluation. From the evaluation result, it is observed that SAX outshined DOM in both storage and time consumption. Nevertheless, theoretical comparisons were also conducted among XRel [12], Edge [18], XParent [19] and their proposed approach, XMap. The authors stated that the proposed algorithm has the benefit over utilizing ORDPath as its labelling scheme. The design of ORDPath supports structural identification and dynamic updates efficiently compare to the rest of the approaches. Figure 4 shows the data model based on XMap mapping scheme, which is based on ORDPath labelling.

Table 3 (A) to Table 3 (C) are the tables that stores data of each node. XMap stored data into three tables, which are data,

TABLE 3. (a) Partial view of Data table. (b) Partial view of Vertex table. (c) Partial view of Path table.

(a)

ordpath	value	order	No.element	No.attribute	pathId
1		1	1	0	1
1.1		2	5	0	2
1.1.1		3	2	0	3
1.1.1.1	Jenzen12	4	0	0	4
1.1.1.3	New	5	0	0	5

(b)

id	name
1	root
2	listing
3	seller_info
4	seller_name
5	seller_rating

(c)

id	path
1	1
2	1.2
3	1.2.3
4	1.2.3.4
5	1.2.3.5

vertex and path table. The data table consists of six attributes, namely, ordpath, value, order, no.element, no.attribute and pathId. The ordpath is the path labelling of node from root to the leaf node using ORDPath labeling [17], the value is the text of the node, the order column is the unique id given to each node in the order of its appearance among its sibling node, while both the no.element and no.attribute are the number of element and attribute the node nesting respectively. Lastly, the pathId is the id of the path that leads to the element. This id act as the foreign key to join to Path table (Table 3 (C)).

The name and id of each element are stored in Vertex table (see Table 3 (B)), while the path id and path expression are store in Path table (see Table 3 (C)).

D. XRECURSIVE MAPPING APPROACH

Fakharaldien *et al.* [15] proposed XRecursive, which is a model-based approach that store XML into RDB storage. XRecursive stores data of the XML document with only two tables, which are Tag_structure table and Tag_value table. The path of each node is identified recursively by using the parent id without storing path value and structure of the path.

Fakharaldien *et al.* performed an experimental evaluation to compare XRecursive with SUXCENT [20]. The results indicated that database storage using XRecursive approach is much more memory saving. SUCXENT uses five tables for data storing while XRecursive only use two tables. The time required for mapping and insertion is improved by XRecursive. Other than that, they also did a comparison on storing method via DOM and SAX parsers. The result revealed that SAX parser uses less size compared to DOM parser. In addition, SAX parser is faster and uses less memory since the model is traversed in depth first manner and unique label is uniquely assigned to each node.

Figure 5 shows data model labelled with simple depth first traversal labelling method.

Table 4 (A) shows the partial view of Tag_structure table. TagName represents the name of the node, ID represents the id of the respective node, while PID represents the parent id

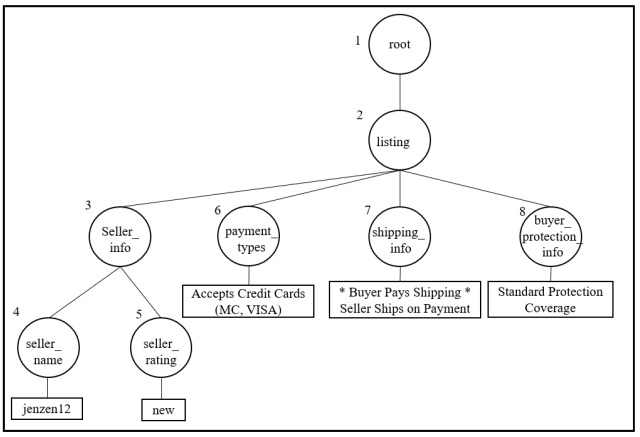

FIGURE 5. Data model labelled based on XRecursive approach.

TABLE 4. (a). Partial view of Tag_structure table. (b) Partial view of Tag_value table.

(a)

TagName	ID	PID
Root	1	1
Listing	2	1
seller_info	3	2
seller_name	4	3
seller_rating	5	3

(b)

TagId	Value	Type
4	New	E
5	Jenzen12	E
6	Standard Protection Coverage	E
7	*Buyer Pays Shipping* Seller Ship on Payment	E
8	Accepts credit cards (MC, VISA)	E

of the node. Since the root node does not have any parent id, the ID of the root node and PID of the root node are the same.

Table 4 (B) depicts the partial view of Tag_value table. This table represents the values associated with the elements or types. The TagId is the primary key of the table, and it is obtained from the ID attribute in the Tag_structure table. The Value attribute represents the value of respective node. The Type attribute consists of either two alphabets, 'A' or 'E', whereby 'A' indicates attribute and 'E' represents element.

E. OTHER EXISTING APPROACHES

Subramaniam *et al.* [10] proposed simple XML (s-XML), which annotate each node based on Persistent labelling. In s-XML, there a two tables, which are parent table and child table. All non-leaf nodes will be stored into parent table while the leaf node will be stored into child table. The authors compared s-XML against Edge, Attribute and DTD approaches. The experimental results showed that s-XML is better in term of time and storage consumption. This is especially proven on query retrieval on complex chain and twig queries evaluation. However, Zhu *et al.* [8] did a comparison test on their proposed approach, Mini-XML against s-XML. It is

observed that s-XML require more time and space, which might be caused by the duplicated information stored in both child and parent table. On a separate research, Qtaish and Ahmad [11] revealed that s-XML take up the most RDB storage space to map XML documents as compared to some mapping approaches.

Suri and Sharma [21] proposed a path-based approach that adopted DOM model to uniquely identify each node with positive numbers. Unlike previous approaches, the proposed approach maintains the P-C relationship by annotating each node based on depth first traversal order. Performance comparisons against existing approaches (XRel and XPEV) were conducted. The evaluation outcomes demonstrated that their proposed algorithm uses the least database storage size. This may be due to their approach only uses two tables to store the data, which resulting in lesser join during query processing.

Ying *et al.* [14] proposed a mapping using hybrid labeling, which combines both path and node labelling technique. The document is first modelled into tree and subsequently, orderly labelled each node. The approach then maps the data into four tables: File, Path, LeafNode and InnerNodes table. The path expressions in path table adopted the XPath representation to generate the path of element node. The approach was compared against XRel [12], XParent [19], and SUXCENT [20] using two datasets and five queries. The results showed that their proposed approach managed to store the document more efficiently and consume lesser storage.

Abduljwad *et al.* [13] proposed SMX/R approach, which uses path labelling technique to track node to node in an XML document. XPath is adopted to identify the path information starting from the root to the leaf node. SMX/R uses two tables to store data, which is (i) Path_Table, and (ii) Path_Index_Table. The mapping approach provides a generic solution in storing XML efficiently while utilizing XPath to extract fragments of the data. The comparison tests performed by the authors indicated that SMX/R outperformed XRel [12] in various aspects such as number of join operations required, number of paths and number of predicates required to accomplish some designated queries.

Jiang *et al.* [19] proposed XParent mapping approach, which is an edge-oriented mapping technique. XParent stores information into four tables, the tables are, (i) label path table, (ii) data table, (iii) element table and, (iv) data path table. XParent enables easy retrieval of regular path queries and utilizing the path identifier to identify value attached on the leaf node. XParent maintain the P-C relationship in label path table to reduce the join operation during query processing.

Yoshikawa *et al.* [12] proposed XRel, which is a path-based approach to map XML data into relational tuples. XRel stores XML data graphs into four tables, (i) Text table, (ii) Attribute table, (iii) Path table, and (iv) Element table. This approach does not require any special indexing structures as each node is orderly labelled in depth first traversal manner. XRel uses Path table with the aims of reducing the cost of join operation. The relationship in the document is maintained by using region method, utilizing start and end position of a node.

In terms of experimental evaluation, the authors have run a comparison test on XRel against Edge [18] approach. The results showed that XRel have some improvement in term of time and space cost to store and query process. However, in 2016, Qtaish and Ahmad [11] demonstrated that XRel consumed the most storage space compared to their approach, XAncestor and XRecursive [15]. This is due to number of tables that XRel required to store the data, which increases the time needed to map XML into RDB, and subsequently involved more join operation to query the data. It is also observed that, the Attribute table will only be shown if a document contains attribute node.

Florescu and Kossmann [18] proposed Edge mapping approach to store all information in XML document into one single table. They named the table as Edge. Edge table only keep label of edge, rather than path label. Huge amount of join operations is required to perform query processing. Over the time, path-based mapping technique was introduced to overcome extensive amount of join operation for querying problem. By using path table to store all possible paths in the document, it reduces query time and search space for retrieving data.

F. SUMMARY OF MAPPING SCHEMES

Table 5 summaries and compare existing approaches discussed. These approaches are grouped into its labelling technique, which are edge, node, path and hybrid labelling. Also, the advantages and weakness of each approach is tabulated in the table based on chronological order of the year being proposed.

On the other hand, a good labelling scheme is essential to provide quick determination on the relationship between query nodes for fast retrieval. Nevertheless, the size of the label increases space consumption on the database and potentially leads to longer time taken for query processing. Moreover, an efficient labelling should be able to avoid relabelling of node and capable to deal with dynamic updates.

Dynamic updates can be identified in a few categories; such are insertion, update and deletion. Most existing works [22, 23, 24] focus on support for dynamic updates on insertion operation as the deletion and modification operations will not cause any re-labeling problem. Our work concurred with existing works, which focus on addressing the insertion operation (to be elaborated further in Section III). Generally, there are three types of insertion: (i) in-between insertion, (ii) left-most insertion, and (iii) right-most insertion.

III. PROPOSED APPROACH

There are two stages involved in this process, mapping process and query retrieval process. In the mapping process, XML document will be mapped and transformed into a RDB, that is, necessary data are extracted and stored in tables. The XML document will first be read and parsed into the parser,

TABLE 5. Summary of Mapping Approaches.

Approach	Scheme	Number of Table	Advantages	Weakness
Edge (1999)	Edge-based	1	Fast and efficient for simple and small document.	Require multiple join operation to retrieve data from one table.
XREL (2004)	Path-based	4	Maintain hierarchical of nodes using region.	High query response times and multiple joins operation needed.
XParent (2002)	Hybrid (Edge-based and Node-based)	4	Reduce join operation by using path table.	Redundant data in different tables.
SMX/R (2010)	Path-based	2	Reduce join operation with path table.	Unnecessary data stored in data table.
XRecursive (2011)	Node-based	2	Less space consuming, only two table used to store data.	Time consuming during query processing for recursive search.
Suri et al. (2012)	Node-based	2	Less space for search process and storage space.	Does not support dynamic update.
s-XML (2012)	Node-based	2	Efficient dynamic update and query for large datasets.	Redundant attributes and duplicated data stored in tables.
Ying et al. (2012)	Hybrid (Path-based and Node-based)	4	Utilizing path table to reduce query process space.	Unnecessary leaf node path information stored.
XMap (2015)	Hybrid (Node-based and Path-based)	3	Support dynamic updates with ORDPath.	Unnecessary data store in data table.
XAncestor (2016)	Path-based	2	Reduce storage space & query time by utilizing path table.	Labelling technique does not support best dynamic update
Mini-XML (2017)	Path-based	2	Support dynamic update effectively with usage of Persistent labelling technique.	Redundant data occur in path table when different node with same path expression exist.
Hammad et al. (2018)	Hybrid-based (Node-based and path-based)	2	Able to query desire data via path table to speed up the retrieval process.	Do not support dynamic update and node label size increase as the tree grow deeper.

namely Streaming API for XML (StAX) parser [25]. Similar to SAX parser, StAX is an event-driven parser, while DOM parser is memory-based parser. The main difference between StAX and SAX parsers is that SAX is a push API, while StAX is a pull API. StAX allows retrieval of data on the available pointer while SAX parser provides the data that it encountered. In another word, StAX parser is able to filter XML data, in which unnecessary data can be ignored.

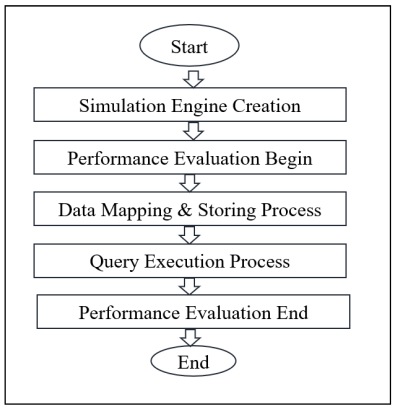

FIGURE 6. Detailed structure of process architecture.

Then, each node will then be assigned with unique identification label to show the exact position of nodes in the document. This process is also called as tree annotation. By employing the proposed algorithm, data will then be transformed into tables in the RDB.

In the query retrieval process, data will be retrieved from the database by using Structure Query Language (SQL). SQL provides commands that justify what data to be retrieved and how to acquire. From the Grapical User Interface (GUI), the user is required to select database that the data has been stored on. Then, the user may input the SQL command to retrieve particular data. It is then translated and retrieved data from the database. The answer of the query will then be returned through the GUI to the user. Figure 6 illustrates the overall view of system architecture design.

This research aims to implement a model-based mapping algorithm that allows dynamic updates without the need to re-labelled the nodes. The proposed approach consists of three components, which is reading XML documents, tree annotation and database design.

A. NODE ANNOTATION

The nodes in the document are then added with unique ID as the label to define its position in the document. This identification values act as one of the main keys in query retrieval process. Without these values, a node with similar names but different position will affect the correctness of retrieved data. Other than that, labelling system is crucial for dynamic updates and making sure that each node existed in the database. Figure 7 shows XML data model with our proposed node labelling system.

The proposed method utilizes a hybrid labelling system, where both path-based and node-based labelling schemes are combined. A tree will be traversed in the depth-first traversal order. Each node will be given position annotation where the

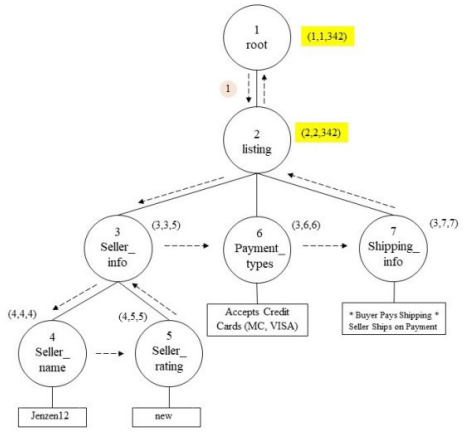

FIGURE 7. XML data model with proposed node labelling system.

label is represented as (*l, s, e*). with *l* represents the level of the node, *s* represents the startid and *e* represents the endid. To identify a leaf node, the startid will be equal to the endid.

In order to identify relationships between the nodes, these labels will provide quick determination on the structural relationship. For instance, for the Parent-Child (P-C) relationship, the level between parent and child node has the difference of one, while for the siblings, it can be identified with the same value of level and the node of its parent node. To identify if A-D relationship exist between nodes, one need to be in the range of another. For instance, to identify if node 4 has AD relationship with node 2, id of node 4 must be in the range of starts and end id of node 2.

On top of that, XML allows nesting of elements, which means that the element can contain another element; this relationship between the elements is described as P-C relationship. If an element nested more than one level, the relationship between the highest-level node and the lowest level node is called as A-D relationship.

B. DATABASE DESIGN

Two tables are used to store the necessary information to ensure lossless of data. These tables are: (i) Element_path table and (ii) Value table. By limiting the number of tables, the join operation required for the query retrieval can be minimized as well, thus, unnecessary storage of data can be avoided. Element path table stores all distinct path information of nodes in the XML document. An unique path ID, PathId is assigned to each unique path expression (Pathexp). Table 6 shows the partial view of the Element_path table.

Table 7 shows the partial view of the Value table. Value table consists of four attributes, which are level, self id, node value and path id. Path id is retrieved from path table where the leaf node is equal to the node name of value node.

C. DYNAMIC UPDATES

To support dynamic updates, we adopted the idea from the scheme proposed by Khanjari and Gaeini [23]. Figure 8

TABLE 6. Partial view of the Element_path table.

PathId	Pathexp
1	/root
2	/root/listing
3	/root/listing/seller_info
4	/root/listing/seller_info/seller_name
5	/root/listing/seller_info/seller_rating
6	/root/listing/shipping_info
7	/root/listing/payment_types

TABLE 7. Partial view of Value table.

Level	ID	Value	RPathId
4	4	jenzen12	4
4	5	new	5
3	6	* Buyer Pays Shipping * Seller Ships on Payment	6
3	7	Accepts Credit Cards (MC, VISA)	7

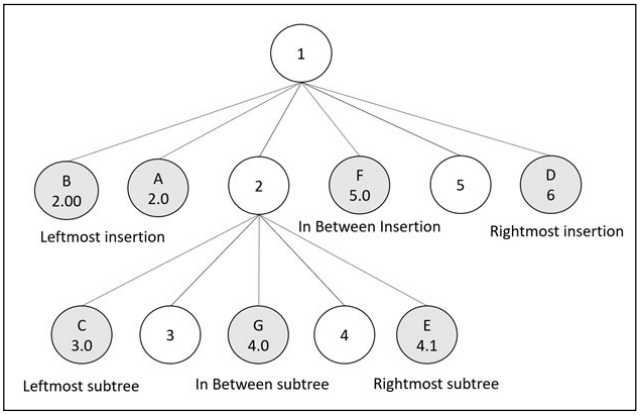

FIGURE 8. Dynamic updates with node instance.

illustrates how dynamic updates take place in various situations. The algorithm is designed to insert data anywhere in the document without any re-labeling.

From Figure 8, if leftmost insertion were to happen, the value of startid will be added with '.0' at the end of the initial label. In case of leftmost insertion happens again, the id will be added '0' at the end. For instance, first insertion of left insertion can be seen for node A and node C. The label was added with '.0' at the end. Node B illustrates the subsequent leftmost insertion of additional node. Label of node is added with value '0' at the end. This process will be repeated for all insertion on the leftmost.

On the case of insertion on the rightmost, there are two situations to be considered. Firstly, it is the situation where startid does not exist. Node D represent the situation, id given label will be added into the value table as it is. Second situation is the insertion of right node if id existed, the level of id needs to be check. If the level is not the same, this situation will be the right insertion of a subtree. The id will be added with value '.1' at the end of the label.

Meanwhile, for the case of the id is within the same level with the new node, but node value does not exist (like node F), this in-between insertion will be added .0 at the

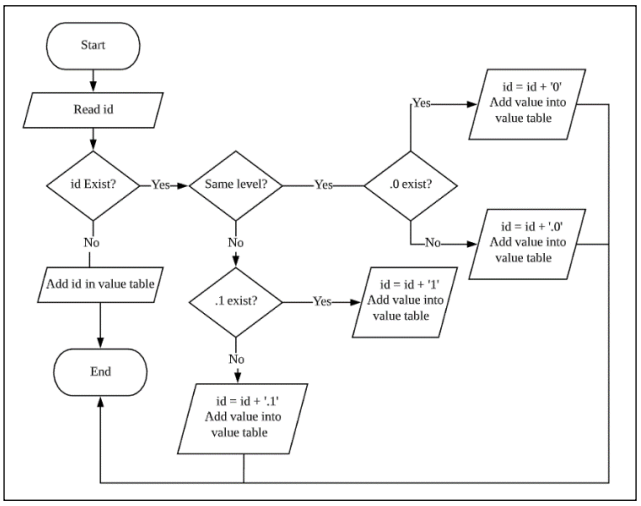

FIGURE 9. Flow of dynamic updates operation.

end of the node label. Figure 9 shows the flow of the insertion if occurs.

IV. IMPLEMENTATION

A. ALGORITHM

An XML-REG is a hybrid of path-based and node-based mapping scheme. The path of each inner and leaf node is tracked and stores in the path table as path expression. This will maintain the hierarchical nature of XML document and can easily locate text node stored in the value table. Parent and ancestor node can be track by using the rpathid in the Value table and pathid in the Element_path table. On the other hand, node-based mapping in this proposed approach is used when it comes to storing the value nodes. Each node is uniquely labelled as id in the value table. Figure 10 shows the pseudocode of XML-REG approach.

First and foremost, for all approaches to be implemented, the connection of database needs to be established. Then, the XML dataset is loaded and parsed using StAX parser. Parser is used to read and extract the data to be mapped into RDB.

In Figure 10, a stack named stackPath is created to store the path as in line 3. It stores the entire element name from root to the current node. StAX parser uses the function getEventType to get the type of the node. There are three event types: (i) startElement in line 6, (ii) character in line 26, and (iii) endElement in line 32.

The startElement retrieves element that exist in the angle bracket tag (< >). It basically retrieves all element name and id will be incremented for each startElement. Local name of startElement is assign to variable qName, and after that, it will be stored in string path. Nevertheless, as shown in line 11, path will only be stores in stackPath if the path does not exist in stackPath. In addition, the attribute node is labelled in the start tag too. Thus, attribute node will be retrieved when the startElement tag is encountered. While attribute exist in startElement, attribute information will be retrieved

```
1.  Create Database connection
2.  Input: An XML document file
3.  Stack stackPath = new Stack()
4.  Create table
5.  Get event type

6.  If is start element;
7.  Id ++
8.  Level ++
9.  qName = startElement.getName()
10. Path = currentpath + "/" + qName
11. if !stackPath.contains(path)
12. Pathid ++
13. stackPath.add(path)
14. while (attributes.hasNext())
15. Id ++
16. Level ++
17. attrName = "@" + attr.getName()
18. attrPath =  path + "/" + attrName
19. if !stackPath.contains(path)
20. Pathid ++
21. stackPath.add(path)
22. Pathvalue = stackPath.indexOf(attrPath) + 1
23. Insert into Value table
24. Level = level - 1
25. End if

26. If character
27. elementValue = characters.getData();
28. If !elementValue.isEmpty()
29. pathvalue = stackPath.indexOf(path) + 1;
30. Insert into Value table
31. End if

32. If end element
33.  pathlast = path.substring(path.lastIndexOf("/") +
    1);
34. if pathlast.equals(eName)
35. path = path.substring(0, path.lastIndexOf("/"))\
36. Level = level - 1
37. End if

38. InsertPath();
```

FIGURE 10. Pseudocode of XML-REG Approach.

and stores into table value with RPathid of existing path in stackPath.

The second EventType is character, where character is the text value of a leaf node. Text node information is stored in variable value and inserted into value table. Finally, for the third EventType, endElement, it takes the element in end tag of XML which is </ >. In endElement, the last qName in string path is removed and follow by deducting level by 1. Unlike value table, path expression and id are stored into path table at the end of the algorithm.

Definition 1: Each node in XML document is denoted as q. For each node q, by following the q type, the information on the name, selfid, level and value are extracted accordingly. The node name will be used for path expression.

Definition 2: Each node will be given position annotation where the label is represented as (l, s, e). with l represents the level of the node, s represents the startid and e represents the endid. To identify a leaf node, the startid will be equal to the endid.

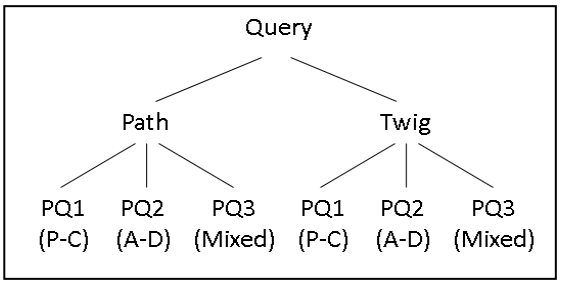

FIGURE 11. Query design.

Definition 3: Each node q is then checked for existence of attribute with attribute.hasNext() function.

Definition 4: Path expression can be denoted as $p_1\ n_1 p_2\ n_2 \ldots p_k n_k$, where p is denoted as the element name and n denoted as relationship between node to node, '/' for P-C relationship and '//' for A-D relationship.

Definition 5: A query is given by, Q = (Nd, Ed) where Nd is a set of nodes in the query tree with n0 ∈ Nd, and Ed is a set of edges that connects Nd with e0 ∈ Ed denotes the association between nodes. The type types of association between the nodes are represented with '/' for P-C relationship and '//' for A-D relationship.

Defintion 6: A P-C relationship existed in a Query Q, if and only if:

- q selfid is in the range of ≥ startid and ≤ endid
- level difference is equal to 1.

Defintion 7: An A-D relationship existed existed in a Query Q, if and only if:

- q selfid is in the range of ≥ startid and ≤ endid
- level difference is more than 1.

The definitions are used to identify hierarchical structure between nodes in XML document.

B. QUERY EXPRESSION

Query execution process evaluates the retrieval time for data to be searched in the newly created tables. The efficiency and effectiveness of this process are influenced by a few factors, such as, number of tables, the information stored, labelling technique and so on. Six queries were prepared for the evaluation as depicted in Figure 11 [26].

Similarly, each query runs six times consecutively. Yet, the first result will be eliminated as it calculates the execution plan of the query before it is executed. Then, the average of these remainder results is calculated.

Table 8 depicts the query patterns used in the evaluation process. Generally, there are two main types of queries, namely Path Query and Twig Query. As for our evaluation, PQ1 to PQ3 are path queries with P-C, A-D, and mixed relationships, while TQ1 to TQ3 are twig queries with P-C, A-D and mixed relationships.

V. RESULTS AND DISCUSSION

All evaluations are carried out on the machine with AMD Ryzen 7 processor (64 bit) with maximum memory capacity of 237 GB and RAM volume of 32 GB. During evaluation test, machine will not work on other tasks and all connection of internet and devices are removed in order to get standardize result for each test. The system is implemented in JAVA language, using the Java SE Development Kit, while Microsoft SQL Server is chosen as the DBMS. This is because Microsoft SQL server is more scalable and reliable compared to other DBMS [27].

TABLE 8. Description of query patterns.

Query number	Query Pattern
PQ1	Path query with P-C relationship
PQ2	Path query with A-D relationship
PQ3	Path query with both P-C and A-D relationship
TQ1	Twig query with P-C relationship
TQ2	Twig query with A-D relationship
TQ3	Twig query with both P-C and A-D relationship

TABLE 9. Various dataset sizes for test evaluation.

Dataset	File Size	Elements	Attribute	Max-Depth	Size
Sigmod	467 KB	11,526	3,737	6	Small
DBLP	130.7MB	3,332,130	404,276	6	Medium
PSD7003	722.6MB	21,305,818	1,290,647	7	Large

Three benchmark datasets were selected for the test evaluation [4]. The smallest size is Sigmod dataset (467 KB), followed by DBLP (130.73 MB) as the medium sized, and PSD7003 (722.59 MB) as the large sized. Details of datasets are shown in Table 9. We have selected these datasets due to several reasons. One of the criteria is the depth of the XML document must be at least three so that the query with A-D relationship can be constructed. Next, the selection of datasets must also contain attributes in an element. This is needed to identify if an approach is able to handle any attributes element.

A. DATA STORING RESULTS

In this section, each dataset will be stored seven times and the first reading will be eliminated to avoid calculation of execution plan and buffering effects. Thus, the average of six times of data storing are taken as the final result. Table 10 shows the result of data storing for all three selected datasets. The best reading in the table are in bold. From the table, it can be observed that our proposed approach (XML-REG) shows the fastest storing results as compared to the rest of the approaches.

Bar charts are constructed for each dataset for illustration to ease the visual comparison. Figure 12 shows the result of insertion on Sigmod dataset in millisecond (ms). As mentioned earlier, Sigmod dataset represent small size XML document. The result shows that XML-REG leading as the fastest approach to store the document, followed by XMap [3], Mini-XML [8], XAncestor [11] and lastly, XRecursive [15].

TABLE 10. Data storing time for Sigmod, DBLP and PSD7003 datasets IN MS.

Dataset	XML-REG	Mini-XML	XAncestor	XMap	XRecursive
Sigmod	**4354.3**	5786.8	5862.3	5609.5	9013.5
DBLP	**673932.5**	1115032.8	943377.3	900419.5	1566933
PSD7003	**3537953.2**	5858550.8	5479149	5516890.3	9204814.3

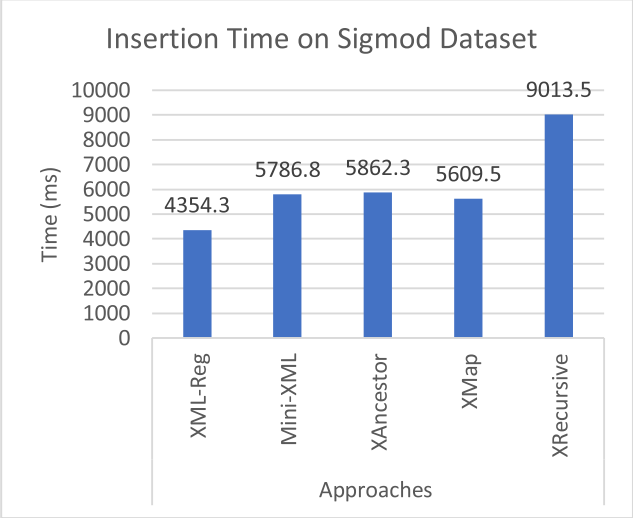

FIGURE 12. Storing result on Sigmod dataset.

Other datasets such as Mondial and Yahoo datasets (from Washington University XML repository) were also used as the representation of small-sized dataset, nevertheless, when it comes to storing the data, it shows inaccurate result, many attributes were not stored correctly. This is because XAncestor algorithm unable to support more than one attributes in an element. For fairness comparison, Sigmod was then employed to represent the small-sized dataset instead.

Figure 13 illustrates the result of approaches on storing DBLP dataset. The result can be seen showing slight differences for three recent approaches. Similarly, to Sigmod, XML-REG shows the fastest storing result, next is XMap [3], XAncestor [8], Mini-XML [11] and finally, XRecursive [15]. As the dataset grows, the efficiency of the approach can be perceived clearly. XRecursive approach takes longer time as it recursively calling the child node and stores unnecessary data into the database. On the other hand, XML-REG only stores unique path of element node and node values with its unique label id to identify its position and hierarchy in the document. Thus, XML-REG takes lesser time than compare to XRecursive [15], and ultimately all other approaches.

To represent large dataset, PSD7003 dataset was selected as it is the largest dataset in the benchmark dataset repository [4]. The storing result of this dataset can be viewed in Figure14. The pattern of the result on PSD7003 dataset was quite similar to DBLP dataset, except that XMap approach comes first prior to XAncestor. Previously, the reason on

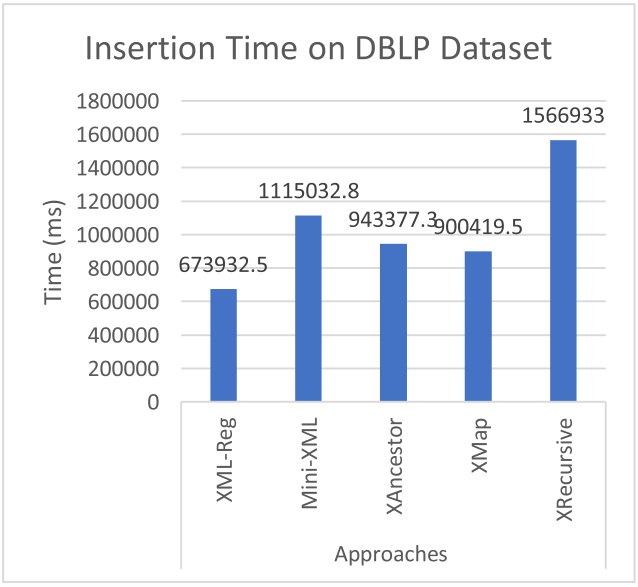

FIGURE 13. Storing result on DBLP dataset.

TABLE 11. Overall result of database size on all approaches in kb.

	XML-REG	XAncestor	Mini-Xml	XMap	XRecursive
Sigmod	**480**	664	864	872	1000
DBLP	**162384**	197064	223112	240416	312208
PSD7003	**677240**	1009464	1362552	1292776	1627488

why XRecursive tends to take more time compare to other approaches was explained. Contrarily, for XMap approach, it takes longer time as compare to XML-REG due to the time taken to store the data into three tables, namely Path table, Vertex table and Data table. For XAncestor, despite of having the drawback which unable to support more than one attributes for each element, this approach efficiency is reduced as it stores its hierarchy via Ances_Pos, which requires it to store and regularly retrieve its ancestor position via Parent position. Come last among the three recent existing approaches is Mini-XML. The result was affected on how the data were stored in the table, especially for the path table, whereby it stores all unique position of the path. In another word, all the inner nodes of the document need to be stored.

B. DATABASE SIZE RESULTS

Aside from the time taken to store XML document into RDB, storage space consumption of each approach is also being evaluated in the evaluation test. Table 11 shows the full results of database size on all the approaches. The smallest storage consumption results among the approaches are in bold. From the overall view, it is noticeable that XML-REG utilizes the least space, followed by XAncestor, Mini-XML, XMap and XRecursive respectively.

Figure 15 illustrates the bar charts on database size of all the approaches. In the chart, we can see that result for XML-REG, Mini-XML and XRecursive approaches are constant

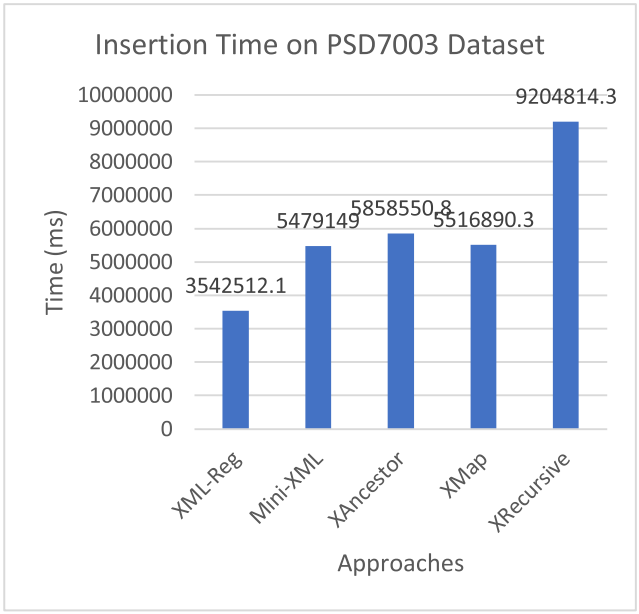

FIGURE 14. Storing result of PSD7003.

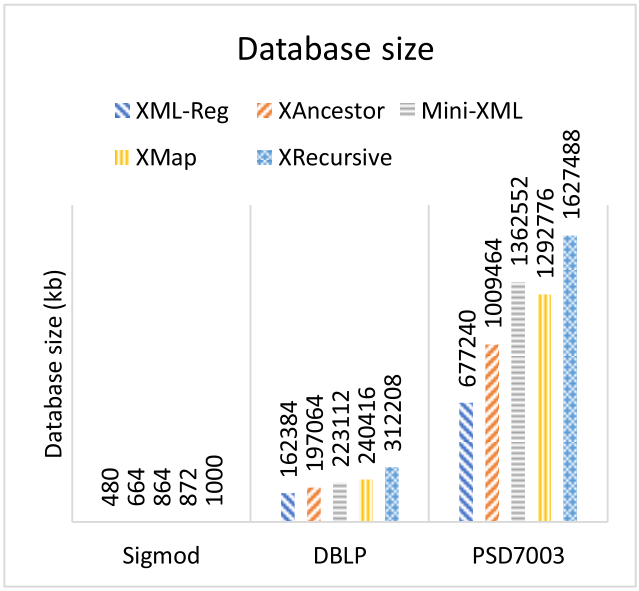

FIGURE 15. Bar chart of database size storage for all approaches.

throughout all the datasets. Nevertheless, XAncestor and XMap approaches have varying results depending on the structure of datasets. XAncestor keeps track of the hierarchy of XML document by storing inner node path in path table and leaf node details in leaf table. Meanwhile, XMap stores XML data into three table, one for data table to store node values and two other tables are used to store hierarchy of document. The design of data table causes XMap database size increase as it stores unnecessary information in the table. The result can be compared for datasets DBLP and PSD7003. XMap require more database size in DBLP and lesser for PSD7003 dataset compares to Mini-XML.

Table 12, Table 13 and Table 14 show the details of database size for each approach. The number of tables and

TABLE 12. Details of database size on Sigmod.

Approach	Mapping scheme	No. of tuples	Total no. of tuples	RDB storage space (KB)
XML-REG	Path Value	12 12120	12132	480
Mini-XML	Path Leaf	3143 12120	15263	864
XAncestor	Ancestor_Path Leaf_Node	4 12120	12124	664
XMap	Data Path Vertex	15607 32 12	15651	864
XRecursive	Tag_structure Tag_value	15263 12120	27383	1000

TABLE 13. Details of database size on DBLP.

Approach	Mapping Scheme	No.of tuples	Total no. of tuples	RDB storage space (KB)
XML-REG	Path Leaf_Node	145 3284834	3284979	159088
Mini-XML	Path Leaf	331243 3410124	3741367	223112
XAncestor	Ancestor_Path Leaf_Node	29 3410124	3410153	197064
XMap	Data Path Vertex	3861590 40 40	3861670	240416
XRecursive	Tag_structure Tag_value	56146176 51151860	107298036	312208

tuples used in each approach, partially give effects on the storage consumption. Moreover, to reduce the storage space of the database, it is needed to be strategically design of the table column, so that, it is able to keep the document structure while minimizing the use of space in the database.

C. DATA RETRIEVAL RESULTS

As mentioned in previous section, the structure of the six queries consists of three path queries and three twig queries. Among the three queries, three types of relationship will be tested. These are the P-C relationship, A-D relationship and mixture of both. Each query will be tested six times and the average of the result is taken as the final result.

In the retrieval evaluation of every dataset, XRecursive is unable to support any A-D relationship related query. As the name of the approach indicated, XRecursive stores the data in such a way that it needs to recursively find the parent and ancestor node. As a matter of fact, one will not know what is the nodes that existed in-between a particular node to its ancestor node. Thus, it is impossible to retrieve the results for any query involving A-D relationship.

a: Query response time on Sigmod dataset

Table 15 shows the query response time of each approach on Sigmod dataset.

TABLE 14. Details of database size on PSD7003.

Approach	Mapping scheme	No.of tuples	Total no. of tuples	RDB storage space (KB)
XML-REG	Path	97	16936016	677240
	Leaf_Node	16935919		
Mini-XML	Path	5195144	22203020	1362552
	Leaf	17007876		
XAncestor	Ancestor_Path	20	17007896	1009488
	Leaf_Node	17007876		
XMap	Data	1293008	1293152	1292776
	Path	72		
	Vertex	72		
XRecursive	Tag_structure	22351500	39359376	1627488
	Tag_value	17007876		

TABLE 15. Query response time on Sigmod dataset.

Query	XML-REG	Mini-XML	XAncestor	XMap	XRecursive
PQ1	2.2	8.6	3.8	3.4	8
PQ2	67	108	105.4	85.4	not supported
PQ3	57.2	121	90.2	85	not supported
TQ1	6.4	95.8	7.4	7.4	25.6
TQ2	6	120.8	20	8.6	not supported
TQ3	6.8	117	41.6	17.2	not supported

Figure 16 illustrates the result of path queries (PQ1-PQ3), while Figure 17 illustrates the result of twig queries (TQ4-TQ6). The same query patterns were prepared on DBLP and PSD7003 datasets. Apart from the XRecursive drawback in handling A-D relationship, the fastest query retrieval approach is XML-REG, followed by XMap, XAncestor, Mini-XML and at the last place is, XRecursive. XRecursive takes the longest time to retrieve data as it needs to recursively join the tables to find the parent and ancestor of the node.

As for twig query, it is shown obviously that Mini-XML takes the longest time (see Figure 17). This is because Mini-XML approach needs to select the data from the database and compare first part of twig query with second part of the query with join operation. Still, the design of approach in mapping and data storing is inadequate enough for query retrieval process. It is important to note that query retrieval could not be performed on XRecursive approach, as this approach could not support A-D retrieval.

b: Query response time on DBLP dataset

Table 16 shows query response time for DBLP dataset. The order of the results is same as Sigmod dataset result where

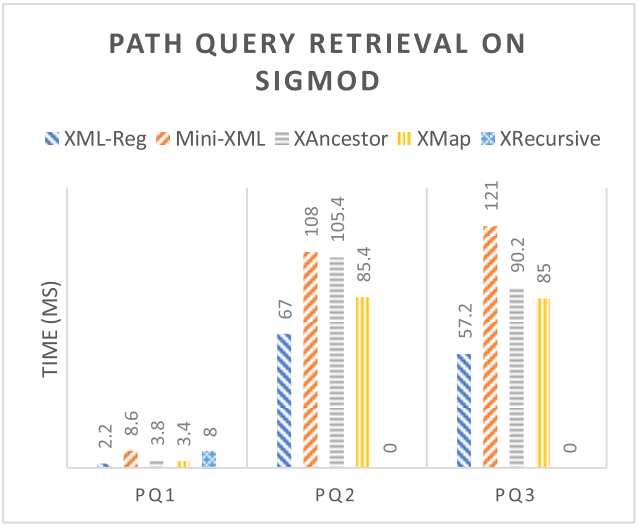

FIGURE 16. Path query retrieval on Sigmod dataset.

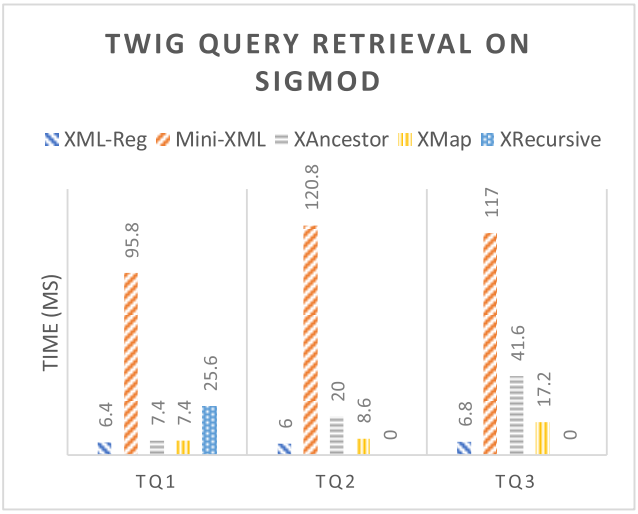

FIGURE 17. Twig query retrieval on Sigmod dataset.

XML-REG come first, then XMap, XAncestor, Mini-XML and finally, XRecursive. Figure 18 and Figure 19 shows result of query retrieval on DBLP dataset in bar chart to ease the viewing of results. The reason on why XMap is more efficient than XAncestor is because, XAncestor store path of nodes until the last inner node, in addition of that, instead of appointing unique id node for the element, approach utilize path-based technique, which stores node position via ancestor path position. When retrieving data from the table, join operation between tables with multiple of where condition needs to be done, thus, it makes sense on why XAncestor require more time to retrieve data.

Figure 19 illustrates the result of twig queries for DBLP dataset. From the figure, we observed that for TQ3, the result is not in the same pattern as the result exhibited from the other datasets. This is due to the complexity of query. Since XMap is using three tables, for TQ3, the amount of join operation used are more as it need to find two paths from joining path

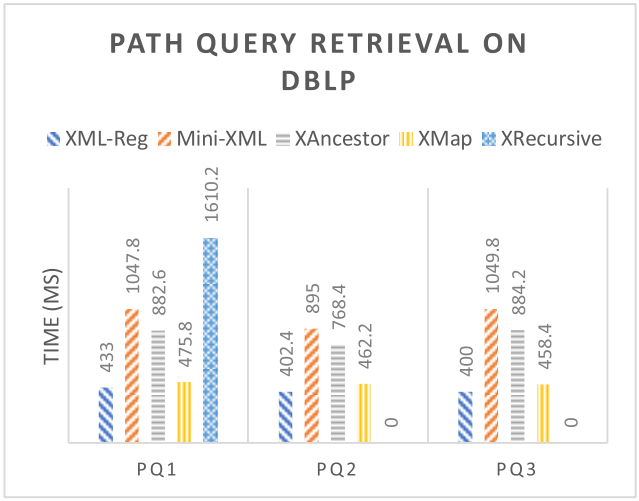

FIGURE 18. Result of path query for DBLP dataset.

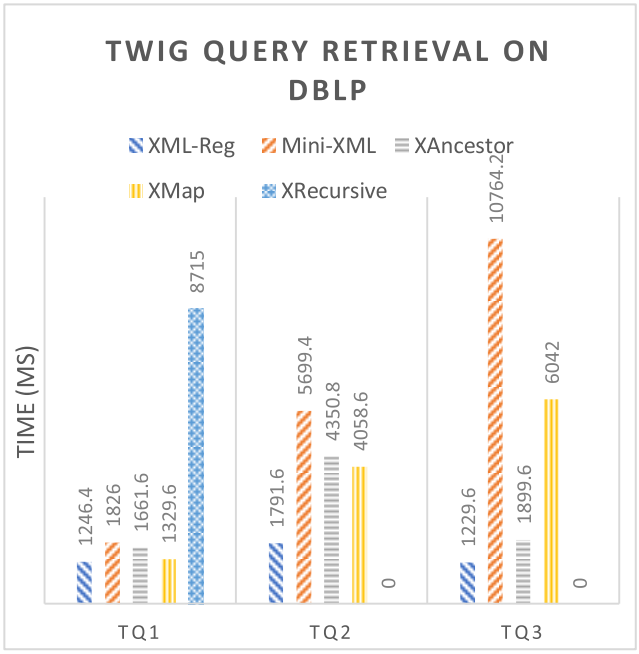

FIGURE 19. Result of twig query for DBLP dataset.

table and vertex table, and then, find the intersection data to retrieve the desired data.

c: Query response time on PSD7003 dataset

Last but not least, for the largest dataset, PSD7003, Table 17 shows the query response time for the dataset, followed by Figure 20 and Figure 21 to illustrate the results into bar chart. The results depict the similar pattern as in the previous dataset, which indicated that XML-REG is the most efficient approach in data retrieval process. This is due to the fact that XMap stores the data into three tables, consequently, increase the number of join operation to retrieve data. Moreover, the design of data storing in XMap value table increase the retrieval time as query needs

TABLE 16. Query response time for dblp dataset.

Query	XML-REG	Mini-XML	XAncestor	XMap	XRecursive
PQ1	433	1047.8	882.6	1237.4	1610.2
PQ2	402.4	5530	768.4	1294	not supported
PQ3	400	1049.8	884.2	458.4	not supported
TQ1	1246.4	1826	1661.6	3979.8	8715
TQ2	1791.6	5699.4	4350.8	4168.8	not supported
TQ3	1229.6	10764.2	1899.6	4198.2	not supported

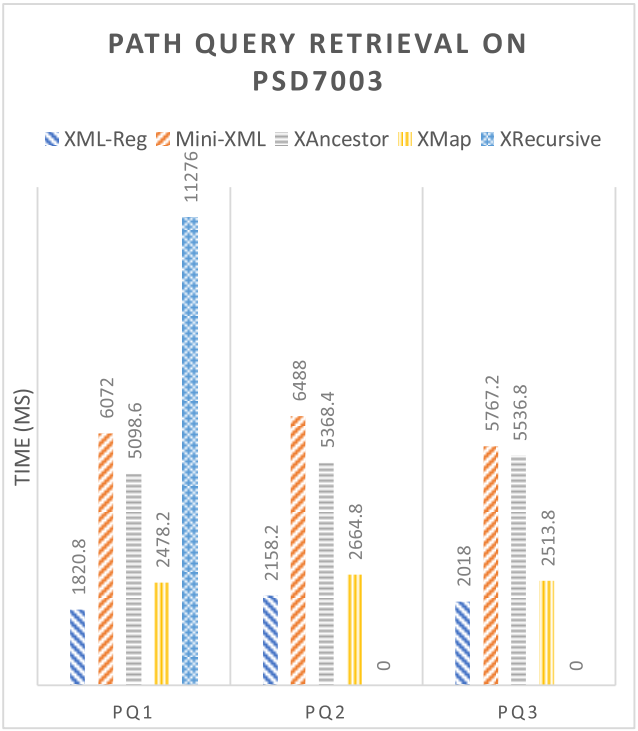

FIGURE 20. Result of path query for PSD7003 dataset.

to browse through empty value field rows and eliminating the null value at the end of the query.

D. DYNAMIC UPDATES

The ability of an approach to support insertion dynamically and yet no changes to the existing row is one of the criteria for good mapping scheme. Assume that Table 18 shows the content of original table before the dynamic updates take place. During dynamic updates, some new nodes are to be inserted as depicted in Table 19. Three types of insertions are tested against XML-REG, (i) Left insertion, (ii) right insertion and (iii) in-between insertion.

Table 20 depicts the original content of Path table before insertion takes place. For any new element inserted, it will be updated in the Path table (see Table 21). The position of path id is defined with.0 for new path and consecutively adding

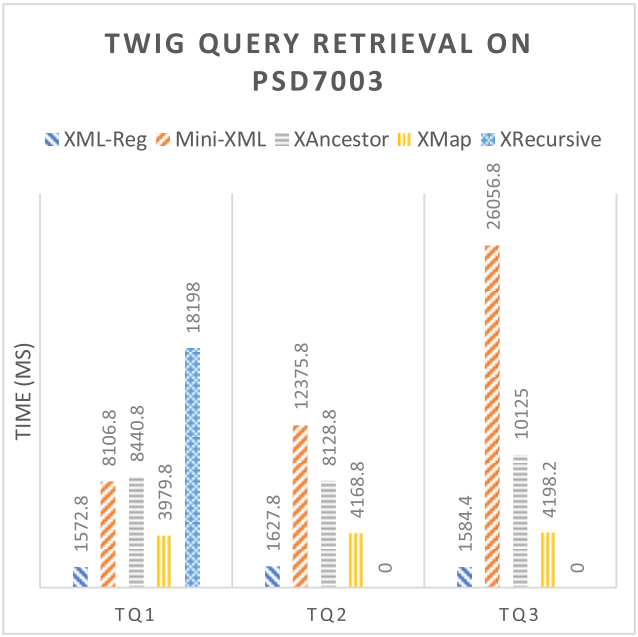

FIGURE 21. Result of twig query for PSD7003 dataset.

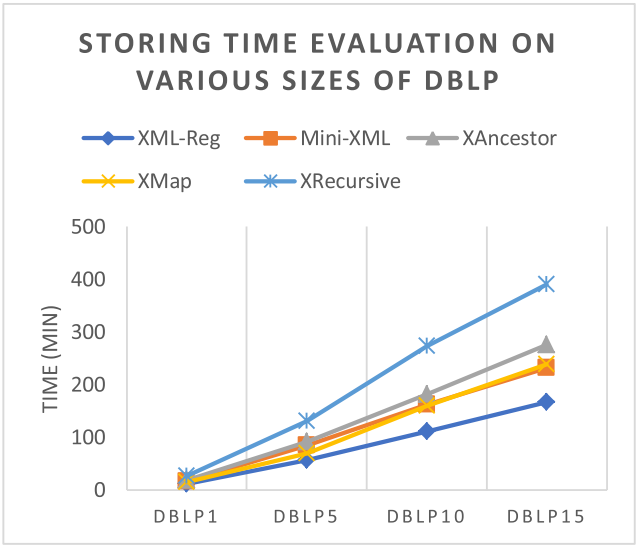

FIGURE 22. Storing time evaluation on various data sizes of DBLP dataset.

0 as the path with the same id is added. There is no restriction on how the path id should be updated as the most important part of this table is path expression column that stores the hierarchy of the document.

E. SCALING RESULTS

d: Datasets for scalability evaluation

Finally, the last part of the evaluation test is to perform scaling on each approach and evaluate on its scalability as data size grows larger. Scalability evaluation is done to check on how each approach handle huge datasets as it grows. For this purpose, the DBLP dataset are multiple by 5 times up to 15 to demonstrate the scalability performance of each approach. Table 22 shows the document size of the scaled datasets. The ability of an approach to support insertion dynamically and yet no changes to the existing row is one of the criteria for good mapping scheme.

TABLE 17. Query response time for PSD7003 dataset.

Query	XML-REG	Mini-XML	XAncestor	XMap	XRecursive
PQ1	1820.8	6072	5098.6	2478.2	11276
PQ2	2158.2	6488	5368.4	2664.8	not supported
PQ3	2018	5767.2	5536.8	2513.8	not supported
TQ1	1572.8	8106.8	8440.8	3979.8	18198
TQ2	1627.8	12375.8	8128.8	4168.8	not supported
TQ3	1584.4	26056.8	10125	4198.2	not supported

TABLE 18. Original Content of Value Table Before Insertions.

Level	ID	Value	RPathId
4	4	jenzen12	4
4	5	new	5
3	6	* Buyer Pays Shipping * Seller Ships on Payment	6
3	7	Accepts Credit Cards (MC, VISA)	7
3	8	Standard Protection Coverage	8

TABLE 19. New Content Of Value Table After Insertions.

Level	id	Value	RPathId	Operation
4	4.00	PosLaju	7.00	Left second insert
4	4.0	4	6.0	Left first insert
4	4	jenzen12	4	
4	5.0	liu	4	In-between insert
4	5	new	5	
4	5.1	KL	7.0	In-between insert
3	6	* Buyer Pays Shipping * Seller Ships on Payment	6	
3	7	Accepts Credit Cards (MC, VISA)	7	
3	8	Standard Protection Coverage	8	
3	9	* Seller Pays Shipping	6	Right first insert

e: Scalability Evaluation Result

Scalability test are conducted to evaluate the capability of approaches in handling the situation where dataset is growing. Table 23 shows the time taken for each approach to store the datasets while Figure 22 depicts the results in line graph.

TABLE 20. Original Content Of Value Table Before Insertions.

PathId	Pathexp
1	/root
2	/root/listing
3	/root/listing/seller_info
4	/root/listing/seller_info/seller_name
5	/root/listing/seller_info/seller_rating
6	/root/listing/shipping_info
7	/root/listing/payment_types

TABLE 21. New Content Of Value Table After Insertions.

PathId	Pathexp	Operation
1	/root	
2	/root/listing	
3	/root/listing/seller_info	
4	/root/listing/seller_info/seller_name	
5	/root/listing/seller_info/seller_rating	
6	/root/listing/payment_types	
6.0	/root/listing/seller_info/seller_rank	First insertion
7	/root/listing/shipping_info	
7.0	/root/listing/seller_info/area	First insertion
7.00	/root/listing/seller_info/Delivery	Second insertion

TABLE 22. Scalability of DBLP datasets.

Dataset	DBLP	DBLP5	DBLP10	DBLP15
Data size	130.726 MB	653.625 MB	1307.25 MB	1960.874 MB

TABLE 23. Time taken for approaches to store the datasets in Min.

Dataset	DBLP1	DBLP5	DBLP10	DBLP15
XML-REG	11.23	56.01	111.25	167.01
Mini-XML	15.72	83.66	161.67	231.66
XAncestor	18.58	90.52	181.67	275
XMap	15.01	68.78	159.72	238
XRecursive	26.12	130.18	273.33	390

From Figure 22, XML-REG shows the closest to linear graph compared to the other approaches. On the other hand, the XRecursive approach has a sharper increase, and hence, it is the least scalable. To calculate the bearing of angle, assuming the result of DBLP is the start point and result of DBLP15 as the end point, the smallest the degree of angle is considered as the best. Calculation can be done by using following formula:

First, get the radian of linear line by using formula:

$$\Theta = \text{atan2}(b1 - a1, b2 - a2)$$

TABLE 24. Degree of angle for each approach.

Approach	XML-REG	Mini-XML	XAncestor	XMap	XRecursive
Value	89.63	89.73	89.77	89.74	89.84

Then, get the degree of line from previous radian:

$$\text{Degrees} = (\text{radian}/\pi) \times 180$$

Table 24 records the degree of angle calculation. It can be seen that the smallest angle is XML-REG, followed by Mini-XML, XMap, XAncestor and XRecursive respectively.

In all the evaluations, proposed approach XML-REG shows the best results. For data storing and query retrieval process, XML-REG takes the least time to process. This is due to the number of tables and data selected to be stored. Apart from that, for storage space usage, XML-REG able to save space by storing selective data and the result shows that as dataset size increases, space usage for the approach grows consistently. On scalability evaluation, graph presented depicts time taken for the compared approaches when datasets are scaled to larger size. Proposed approach responds a linear graph and takes the least time compares to other. The complexity of proposed algorithm is represented with O(n).

VI. CONCLUSION

This paper has three main objectives. The first objective of this research is to study existing mapping approaches on model-based mapping approaches. Some existing mapping approaches were reviewed and analysed on the drawbacks and advantages. Although the aim of this research is to propose and design an efficient mapping approach, labelling schemes were studied so that the proposed approach is able to support dynamic updates operations especially on insertion. Thus, lead this paper to its second objective, which is to propose an efficient model-based mapping approach to bridge the technology of XML and RDB.

A new XML to RDB mapping approach named XML-REG is proposed in this research. XML-REG is a hybrid of node-based and path-based mapping approaches, which means that this approach takes the best of path-based mapping and node-based mapping to produce a hybrid outcome. XML-REG stores data into two tables which are Path and Value table. Path table stores the unique path expression of the XML. As STaX parser does not support XPath technology directly, the path is stored into string and stack before it is map into RDB. On the other hand, for the Value table, it stores data of text node. It adopted the node-based technique, whereby each node is uniquely assigned with an id to represent their position in the document.

For the final objective of this paper, the proposed approach is compared against four existing approaches, Mini-XML [8], XAncestor [11], XMap [3], and XRecursive [15].

Performance tests were conducted on storing process, query retrieval process, database size and scalability test. The result of each test shows that XML-REG outperformed the existing approaches as it able to store XML into database with the least time required and least storage space. For retrieval process, proposed approach returns accurate result in the shortest time. Scalability test were also being conducted to see how each approach perform as the data sizes grows on the DBLP dataset. XML-REG shows also linear result as compared to the rest of the approaches. This indicated XML-REG is scalable to support huge datasets.

The contributions of the research presented in this paper can be summarized as follows:

1) Summary of existing mapping approaches and labelling method.
2) A new mapping technique which exploits the advantage of path and node-based mapping technique. It has been demonstrated that the XML-REG approach can efficiently transform XML document into RDB.
3) An improved labelling technique was created and adapted within XML-REG to support dynamic updates.

REFERENCES

[1] C. Li and T. W. Ling, "A novel quaternary encoding to completely avoid re-labeling in XML updates," in *Proc. 14th ACM Int. Conf. Inf. Knowl. Management.*, New York, NY, USA, 2005, pp. 501–508.

[2] A. Qtaish and K. Ahmad, "Query mapping techniques for XML documents," in *Proc. 5th Int. Conf. Electr. Eng. Informat.*, 2015, pp. 529–534.

[3] Z. Bousalem and I. Cherti, "XMap: A novel approach to store and retrieve XML document in relational databases," *J. Softw.*, vol. 10, no. 12, pp. 1389–1401, 2015.

[4] UW. *Washington University XML Repository*. (2015). [Online]. Available: http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/www/repository.html

[5] A. Qtaish and K. Ahmad, "Model mapping approaches for XML documents: A review," *J. Inf. Sci.*, vol. 41, no. 4, pp. 444–466, Aug. 2015.

[6] H. Lu, W. Wang, and J. X. Yu, "Path materialization revisited: An efficient storage model for XML data," in *Australian Computer Science Communications*. Darlinghurst, Australia: Australian Computer Society Inc., 2002, pp. 85–94.

[7] J. Qin, S. M. Zhao, S. Q. Yang, and W. H. Dou, "XPEV: A storage model for well-formed XML documents," in *Proc. Int. Conf. Fuzzy Syst. Knowl. Discovery*, 2005, pp. 360–369.

[8] H. Zhu, H. Yu, G. Fan, and H. Sun, "Mini-XML: An efficient mapping approach between XML and relational database," in *Proc. IEEE/ACIS 16th Int. Conf. Comput. Inf. Sci. (ICIS)*, May 2017, pp. 839–843.

[9] A. Gabillon and M. Fansi, "A New Persistent Labelling Scheme for XML," *J. Digit. Inf. Manage., Digit. Inf. Res. Found.*, vol. 4, no. 2, pp. 112–116, 2006.

[10] S. Subramaniam, S.-C. Haw, and P. K. Hoong, "S-XML: An efficient mapping scheme to bridge XML and relational database," *Knowl.-Based Syst.*, vol. 27, pp. 369–380, Mar. 2012.

[11] A. Qtaish and K. Ahmad, "XAncestor: An efficient mapping approach for storing and querying XML documents in relational database using path-based technique," *Knowl.-Based Syst.*, vol. 114, pp. 167–192, Dec. 2016.

[12] M. Yoshikawa, T. Amagasa, S. Shimura, and S. Uemura, "XRel: A path-based approach to storage and retrieval of XML documents using relational databases," *ACM Trans. Internet Technol.*, vol. 1, no. 1, 2001, pp. 110–114.

[13] F. Abduljwad, W. Ning, and X. De, "SMX/R: Efficient way of storing and managing XML documents using RDBMSs based on paths," in *Proc. 2nd Int. Conf. Comput. Eng. Technol.*, Apr. 2010, pp. 143–147.

[14] J. Ying, S. Cao, and Y. Long, "An efficient mapping approach to store and query XML documents in relational database," in *Proc. 2nd Int. Conf. Comput. Sci. Netw. Technol.*, Dec. 2012, pp. 2140–2144.

[15] M. A. I. Fakharaldien, J. M. Zain, and N. Sulaiman, "XRecursive: An efficient method to store and query XML documents," *Austral. J. Basic Appl. Sci.*, vol. 5, no. 12, 2011, pp. 2910–2916.

[16] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, "Storing and querying ordered XML using a relational database system," in *Proc. ACM SIGMOD Int. Conf. Manage. Data SIGMOD*, 2002, pp. 204–215.

[17] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: Insert-friendly XML node labels," in *Proc. ACM SIGMOD Int. Conf. Manage. Data SIGMOD*, 2004, pp. 903–908.

[18] D. Florescu and D. Kossmann, *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*. Paris, France: Institut National de Recherche en Informatique et en Automatique, 1999.

[19] H. Jiang, H. Lu, W. Wang, and J. Xu Yu, "XParent: An efficient RDBMS-based XML database system," in *Proc. 18th Int. Conf. Data Eng.*, Feb. 2002, pp. 335–336.

[20] S. Prakas, S. S. Bhowmick, and S. Madria, "SUCXENT: An efficient path-based approach to store and query XML documents," in *Proc. Int. Conf. Database Expert Syst. Appl.*, 2004, pp. 285–295.

[21] P. Suri and D. Sharma, "A model mapping approach for storing XML documents in relational databases," *Int. J. Comput. Sci. Issues*, vol. 9, no. 3, pp. 495–498, 2012.

[22] J. Ahn, D.-H. Im, T. Lee, and H.-G. Kim, "A dynamic and parallel approach for repetitive prime labeling of XML with MapReduce," *J. Supercomput.*, vol. 73, no. 2, pp. 810–836, Feb. 2017.

[23] E. Khanjari and L. Gaeini, "A new effective method for labeling dynamic XML data," *J. Big Data*, vol. 5, no. 1, pp. 1–17, Dec. 2018.

[24] J. Ahn, D.-H. Im, T. Lee, and H.-G. Kim, "Parallel prime number labeling of large XML data using MapReduce," in *Proc. 6th Int. Conf. IT Converg. Secur. (ICITCS)*, Sep. 2016, pp. 1–2.

[25] *Sun Microsystems, Streaming APIs for XML Parsers, Java Web Services Performance, Team White Paper*, Oracle Corp., Redwood City, CA, USA, 2005.

[26] S. C. Haw and C. S. Lee, "Node labeling schemes in XML query optimization: A survey and open discussion," *IETE Tech. Rev.*, vol. 26, no. 2, 2009, pp. 89–101.

[27] A. Saikia, S. Joy, D. Dolma, and R. Mary. R, "Comparative performance analysis of MySQL and SQL server relational database management systems in windows environment," *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 4, pp. 160–164, Mar. 2015.

EMYLIANA SONG is currently pursuing the master's degree with the Faculty of Computing and Informatics, Multimedia University. Her research interest includes XML data mapping scheme in transforming XML document into relational database.

SU-CHENG HAW (Member, IEEE) is currently an Associate Professor with the Faculty of Computing and Informatics, Multimedia University, where she leads several funded researches on the XML databases. She has published more than 120 articles in reputable journals and conferences. Her research interests include XML databases, query optimization, data modeling, semantic web, ontology, data management, and data warehousing.