Indonesian Journal of Electrical Engineering and Computer Science
Vol. 7, No. 2, August 2017, pp. 551 ~ 566
DOI: 10.11591/ijeecs.v7.i2.pp551-566

XMapDB-Sim: Performance Evalaution on Model-Based XML to Relational Database Mapping Choices

Haw Su-Cheng
Faculty of Computing and Informatics, Multimedia University, Malaysia
Corresponding author, e-mail: sucheng@mmu.edu.my, schaw@mmu.edu.my

Abstract

XML has emerged as the standard for information representation over the Internet. However, most enterprises today have long secured the use of relational databases. Thus, it is crucial to map XML data into relational data to provide seamless integration between these database infrastructures. Many mapping techniques have been proposed, yet, none has provides a unified view on these techniques. Ultimately, understanding how these techniques work is important especially if one needs to decide which technique to adopt in their organization. This paper (i) reviews on some existing model-based mapping schemes focusing on how the mapping technique works, the advantages and the disadvantages, (ii) present the simulation engine to evaluate the performance of selected mapping schemes, (iii) highlight the future direction of the related area.

Keywords*: XML-to-relational database, Data integration, Mapping scheme, Relational database*

1. Introduction

XML is the abbreviation for eXtensible Markup Language, a formal recommendation from the World Wide Web Consortium (W3C), which act as the standard for data presentation and exchange over the World Wide Web (WWW). It is extensible in nature, with great support for dynamic updates on the Create, Retrieve, Update and Delete (CRUD) operations of database [1-3]. Nevertheless, most organizations are still using relational database (RDB) as the storage. Yet, RDB is unable to cope with the high and fast pace demands of electronic business due to its independent data processing. As such, it is important to have effective mapping method to store and retrieve XML via RDB. Ultimately, the most challenging issue on this is to resolve the conflict between hierarchical structure of XML and the flat structure of RDB.

There exist various mapping approaches to efficiently map XML into RDB, which can be broadly classified into structural-based mapping (schema-based mapping) and model-based mapping (schema-less mapping). In structural-based mapping [4, 5], the XML is shredded into RDB based on the schema defined in either Document Type Definition (DTD) or XML schema (XSD). Intrinsically, this will result in additional complexity especially when managing different types of XML documents [6, 7]. On the other hand, the model mapping approaches are mapped to some fixed relational schemas based on the XML document itself. In this case, usually a fixed RDB scheme is employed.

The focus of this paper is on the model-based mapping approach since it supports the wider range of web applications, as most applications does not usually comes together with any DTD nor XML schema. Generally, the model-based mapping scheme can be further classified into path-based and node-based approaches. The mapping based on path expression is composed based on the idea to store the path for every node visited in one or more tables. In contrast, a node-based approach is constructed based on some annotation on the nodes to identify the position of the nodes relatively to the root of the tree.

The rest of the paper is organized as follows. Section 2 provides detailed review on three selected state-of-art model-based mapping approaches, focusing especially on how the approach works based on a running example. Section 3 proposes the architecture of simulation engine named XMapDB-Sim to measure the performance of the selected approaches. Section 4 presents the experimental results and analysis on the findings. Lastly, Section 5 summarizes the paper and points out the future direction on possible research in this area.

Received May 22, 2017; Revised July 5, 2017; Accepted July 23, 2017

2. Literature Review

Many mapping schemes have been proposed to resolve the conflict issues on transforming XML into RDB technologies [8-12]. Nevertheless, there is an absence of review to provide a unified view on these approaches to enable practitioner or organization to choose which approaches that suited them most. In view of this, this paper reviews some selected recent mapping schemes based on a running example. Figure 1 shows the illustration sample of XML which will be used throughout the paper. We will elaborate in detail how each scheme works based on the same example.

```
<university>
    <student id = "1140">
        <name> Amirah </name>
        <enrolment>
            <intake> October 2014 </intake>
            <detail>
                <faculty> FCI </faculty>
                <course> Computer Science </course>
                <specialization> Information System </specialization>
            </detail>
        </enrolment>
    </student>
    <student id = "1141">
        <name>Emyliana</name>
        <enrolment>
            <intake> October 2014 </intake>
        </enrolment>
    </student>
</university>
```

Figure 1. XML illustrative example

2.1. SMX/R Approach

SMX/R uses path-based labeling technique to track node to node [13]. The document is parsed using both Simple API for XML (SAX) and Document Object Model (DOM) parser. The SAX Parser is used to track start (symbol '<') and end (symbol '>') elements to assign the label based on sequence of appearance as illustrated in Figure 2, while DOM parser is used to track the path of the node and element value.

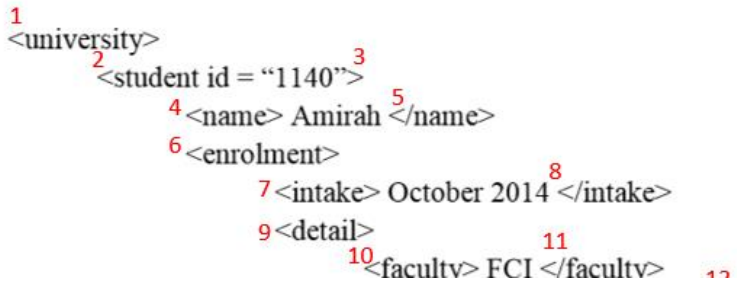

Figure 2. Partial XML document labelled with start and end element

SMX/R store data into two tables, which are path table and path index table. Table 1 depicts the Path_Index_Table, which consists of Document id (DocId), Path id (PID), Start Position (StartPos), End Position (EndPos), Node level (NodeLVL), Node type (NodeType) and Node value (NodeValue).

Table 1. Partial view of Path_Index_Table

DocId	PID	StartPos	EndPos	NodeLVL	NodeType	NodeValue
1	1	1	26	1	Element	
1	2	2	18	2	Element	
1	3	2	18	3	Attribute	1140
1	4	4	5	3	Element	Amirah
1	5	6	17	3	Element	

On the other hand, Table 2 depicts the Path_Table, which consists of PID (id of the path), Path (path value from root to the current element) and node name (the name of current node). The content of the tables were generated based on Algorithm 1.

Table 2. Partial view of Path_Table

PID	Path	Node Name
1	University	university
2	University.student	Student
3	University.student.@id=1140	@id=1140
4	University.student.name	Name
5	University.student.enrollment	enrollment
6	University.student.enrollment.intake	Intake

SMX/R approach begins by reading the XML document, and assign a unique ID in line 3 (see Algorithm 1). The path of a node is first being identified if it exists in Path_Index_Table (line 4). If the path exists in the Path_Index_Table, the path is identify in line 6 via PID and associate with the particular node. If the does not exist yet, new PID will be created and new entry will be inserted into Path_Index_Table. The process recursively repeated until it reaches the end of the document.

```
Algorithm 1: SMX/R Approach algorithm to map XML to RDB
1    Begin
2            Read XML document
3            Assign unique ID to the XML document
4            Check If the path is indexed in the Path_Index_Table
5            Identify path for the node
6            Identify existing path ID
7            Associate path ID with node
8            Else
9            Create new path ID
10           Create entry in Path_Index_Table associated with new path ID
11           End else
12           End if
13           End
14           End if
15           Else
16           Associate the entire path with the node
17           Store the node information
18           Store the hierarchical information
19           End else
20           End if
21   End
```

The authors have also conducted a comparison study between SMX/R with XRel [14]. XRel require more JOINs to retrieve the two set of test queries, while no join is required in SMX/R approach. This is due to the reason that the SMX/R has less number of tables and tuples in the tables compare to XRel.

2.2. XRecursive Approach

Fakharaldien et al. [15] proposed XRecursive, a model-based approach that store XML documents into RDB. XRecursive accept both fixed schema and non-fixed schema document. This approach identifies each path recursively by its parent id; hence, it does not need to store the path value or path structure.

Table 3 represents the Tag_structure, which consists of TagName (the name of the node), Id of the respective node (which is also the primary key), and pId (the parent id of the node). Since the root node does not have any parent id, the id of the root node and parent id of the root node are the same.

Table 3. Partial view of Tag_structure

TagName	Id	pId
university	1	1
student	2	1
Id	3	2
name	4	2
enrolment	5	2

Table 4 depicts the partial view of Tag_value, where by TagId is the foreign key which match the value Id in Table 3, value column represents the value of respective node, type column consists of two possible values, that is either 'Attribute' or 'Element'.

Table 4. Partial view of Tag_value

TagId	Value	Type
3	1140	Attribute
4	Amirah	Element
6	October 2016	Element
8	FCI	Element
9	Computer Science	Element

XRecursive decompose tree structure into nodes and all information of nodes are stored in RDB according to the node types in recursive manner as depicted in Algorithm 2. In order to allow addition of multiple XML file in the storage, XRecursive add document name in association with the id. Each and every element will have a unique id to represent it and there will also have a parent node associated with it. Yet, XRecursive does not store the path value because it will be determined by the parent id. In line 3 and 4, empty set tag_structure and tag_value are built to represent list of node and value. To begin the insertion of node into list, in line 10, it reads the element type. In line 11 to 16, the name, id and Pid are added into the list with condition if the element type is either element or attribute. Next, the id will be increased by 1. From line 17 to 22, if the element type of node is value, the id will be incremented by 1 and thus, the name, value and id are added into tag_value. Each list is stored into database in line 23 to 24.

The authors did experimental evaluation on comparing the storing method via SAX parser and DOM parser. From the result, SAX parser parsed XML document faster and uses less memory than DOM.

Algorithm 2: XRecursive Mapping Choice Algorithm to map XML to RDB

```
1   Begin
2    Let tag_structure = {} as empty set, where tag_structure represents the list of node of the XML File
3    Let tag_value = {} as empty set, where tag_value represents the list of the value of the node
4    Let String filename = null and int Parentid = 1;
5    Filename = read XML document name
6    While xml document is not null
7    Begin
8     Read element name as name
9     Read the element type
10    If element type is element or attribute
11    Begin
12      Read parent as pName
13      Id= id +1;
14      Add name, pName, id to the tag_structure
15    End
16    Else if element type is #text
17    Begin
18      Read text value as value
19      Id = id +1;
20      Add name, value, id to tag_value
21    End
22   Store tag_structure into database
23   Store tag_value into database
24  End
```

2.3. Approach Proposed by Suri and Sharma (SS Approach)

Suri and Sharma proposed a path-based approach, hereinafter abbreviated as SS approach [16], by firstly decompose the XML document into tree, and subsequently, map each node in the tree to RDB. Unlike the previous mapping scheme such as SMX/R [13], XRel [14] and XPEV [17] which store data based on path concept, this proposed mapping choice maintain parent and child relationship between elements, thus, avoids the needs to store the path information.

The two tables that needed to store XML document are Node table (Node id, Node name) and Data table (Doc id, Node id, Parent id, Node value, Node type, Node pos). Node table (see Table 5) stores all node id's with their respective names, whereby NodeID is the unique id assigned to each node of the XML document, while NodeName represents the name of the node.

Table 5. Partial content of Node table

NodeId	NodeName
1	university
2	student
3	Id
4	name
5	enrolment
6	intake

Data table stores values of each node. The DocId represent the id of the particular XML document, NodeID is the unique id assigned to each node of the XML document, Parent id is the id of parent node of a node, NodeValue represents the value of the node, NodeType is used to indicate type of each node (which is either element, attribute, or text), and NodePos is the position of the node among its siblings in the XML data tree.

Table 6. Partial content of Data table

DocId	NodeId	ParentId	NodeValue	NodeType	NodePos
10	1	Null	Null	Element	1
10	2	1	Null	Element	1,1
10	3	2	1140	Attribute	1,1,1
10	4	2	Amirah	Element	1,1,2
10	5	2	Null	Element	1,1,3
10	6	5	October 2016	Element	1,1,3,1
10	7	6	Null	Element	1,1,3,2
10	8	7	FCI	Element	1,1,3,2,1

The content of both tables are generated through Algorithm 3. This approach firstly assigns a unique identifier to each node, identifies the root element of the document and then stores the corresponding node values along with each node id in relational table. The tree is traversed based on depth first traversal manner. There are three main cases to be take note as follows:

1. If node is an element (Line 12)
2. If the node is an attribute (Line 13)
3. The element node has a text value (Line 14)

For case 1 in line 12, if the node is an element, the node type will be store as “element”. Meanwhile for case 2, if the node is an attribute, the node type will appear as “attribute” in line 13. For case 3, in line 14, if the element node has a text value, the node type will appear as “text”. Each node, regardless of the type, node id and node name are stored into the Node table.

They have conducted the performance evaluation of their proposed approach as compared to XRel [14] and XPEV [17]. In the evaluation test, the proposed algorithm uses the least database size among the three mapping schemes.

2.4. Other More Recent Approaches

Ying et al. [18] proposed a hybrid approach combining path-based approach with node-based to transform the XML documents to a RDB scheme. In their approach, the Path table is utilized to store each distinct path expression of the leaf node, while the InnerNode (internal node) is annotated with labeling scheme to maintain only the unique absolute path expression. This further decreases the storage space, and ultimately faster retrieval is possible based on the basis of the relationship maintained by the labels of inner nodes.

Bousalem & Cherti [7] proposed XMap to store and retrieve XML in relational database. In their approach, the XML is shredded into three tables, namely Data Table (stores value of the node), Vertex Table (stores node information) and Path Table (stores path information). In the Data Table, OrdPath [19] is employed as the labeling scheme to avoid re-generating the content whenever dynamic updates happen, and thus, achieves lower storage consumption. Though no experimental evaluation has been carried out, they proved by complexity analysis that their proposed algorithm is linear.

XAncestor [6] is a path-based labeling technique to create mapping on only distinct ancestor paths (ignoring the inner nodes to reduce the storage space), for all leaf nodes of the XML tree into its RDB. It consists of two main algorithms namely (i) XtoDB and (ii) XtoSQL. XtoDB maps XML documents to a fixed RDB using a DOM parser to decompose the XML document into a predefined RDB scheme: the Ancestor_Path (Ances_PathID, Ances_PathExp) table and a Leaf_Node (Node_Name, Ancest_PathID, Ances_Pos, Node_Value) table. XtoSQL translates XPath queries into corresponding SQL queries based on the Ancestor_Path and Leaf_Node tables, in order to achieve a shorter response time. XAncestor is evaluated with five other related approaches in terms of the RDB storage space and query response time. From their evaluation, XAncestor outperformed XRel [14], SMX/R [13], XRecursive [15], s-XML [20], and approach proposed by Ying et al. [18], in terms of RDB storage space, and query response time for various types of queries.

In a more recent work, Zhu et al. [21] proposed mini-XML, a path-based model mapping approach to identify the path among the non-leaf nodes. As the result, two tables namely PathTable and LeafTable are created to store the nodes. In constructing the tables, the position information annotated with their proposed labeling scheme of (Level, [P-pathID, S-order]), where Level is the depth of the current leaf node in the tree, P-pathId is the path of the direct

parent node, and S-order is the position number of the current leaf node in direct node. This serves as the crucial identification on the complex node relationship. They have experimentally demonstrated that their approach is better than s-XML [20] in terms of storage space and storage time. Nevertheless, there is no evaluation being carried out on the query retrieval yet.

Algorithm 3: Mapping Algorithm by Suri and Sharma

1 Begin
2 Read the XML document
3 Assign a unique id to the XML document say 10 here i.e. doc id = 10.
4 Create new object of the NODE table say node
5 Create new object of DATA table say data
6 While XML document is not null Repeat step 7 to 12
7 Identify the root element of the XML document
  i. Node id = 1
  ii. Node type = element
  iii. Node pos = 1
  iv. Parent id = null
  v. Node name = root node name
8 Store Node id, Node name to NODE
9 Store Node id, Parent id, Node type, Node pos to DATA
10 Traverse the whole XML tree in the depth first manner from node to node
11 If the node is an element
  i. Node id = Node id +1
  ii. Node type = element
  iii. Node pos = (parent Node pos, child Node pos)
  iv. Node name = node name
  v. Store Node id, Node name to NODE
  vi. Store Node id, Node type, Node pos to DATA
12 If the node is an attribute
  i. Node id = Node id + 1
  ii. Node type = attribute
  iii. Node pos = (Parent Node pos, child Node pos)
  iv. Node value = text value
  v. Node name = node name
  vi. Store Node id, node name to NODE
  vii. Store Node id , node type , Node pos, Node value to DATA
13 If the node element has a text value
  i. Node id = Node id +1
  ii. Node type = text
  iii. Node pos = (parent Node pos , child Node pos)
  iv. Node value = text value
  v. Node name = node name
  vi. Store Node id with Node Name to NODE
  vii. Store Node id, Node type, Node pos, Node value to DATA
14 End

Table 7 shows the comparison of the three selected mapping schemes reviewed earlier. All these approaches use only two tables in RDB to store data. SS and XRecursive approaches use node labeling technique while SMX/R approach uses path-node labeling. Nevertheless, the efficiency on these approaches is determined based on how the translation of XPath query into SQL statement based on their respective relational scheme [6]. In doing so, some factors such as number of join operations required, number of columns accessed, number of tuples involved, and the ease to determine the structural relationships (such as Parent-Child (P-C), Ancestor-Descendant (A-D), Level, and Sibling relationships) among nodes, will have effect on the performance.

Table 7. Comparison of three chosen mapping choices

Approach	SMX/R	XRecursive	SS
Number of tables	two tables	two tables	two tables
Mapping technique	Path-based labeling	Node-based labeling	Node-based labeling
Labeling method	Depth first traversal	Depth first traversal	Depth first traversal with node position information
Advantages	Accurate retrieval of element.	Fast extraction time.	Less join operation. Less storage space.
Disadvantages	Take times to store and retrieve data as it uses 2 parser- SAX and DOM.	Nested query is needed for all retrieval, which may slow down the performance.	High query processing time for large document.

From the review, most researchers only compared their approach with other approaches that are based on their own selection. In the absent of this, we propose XMapDB-Sim to perform the evaluation to test the performance of the selected mapping approaches on various sizes of datasets, especially large size.

3. XMAPDB-SIM: The Mapping Performance Evaluation Engine

Figure 3 depicts the flow of XMapDB-Sim, which consists of two main components, the (i) Data Mapping, and (ii) Query Execution. Firstly, the XML document will be loaded and read by document builder factory, and subsequently parsed into tree data model by the parser. Next, the XML trees will be converted into RDB using one of the three selected mapping algorithms. The storing time is then recorded. Successively, the query can be retrieve and the response time will be record and display in simulation engine. Figure 4 depicts the interface of XMapDB-Sim main screen.

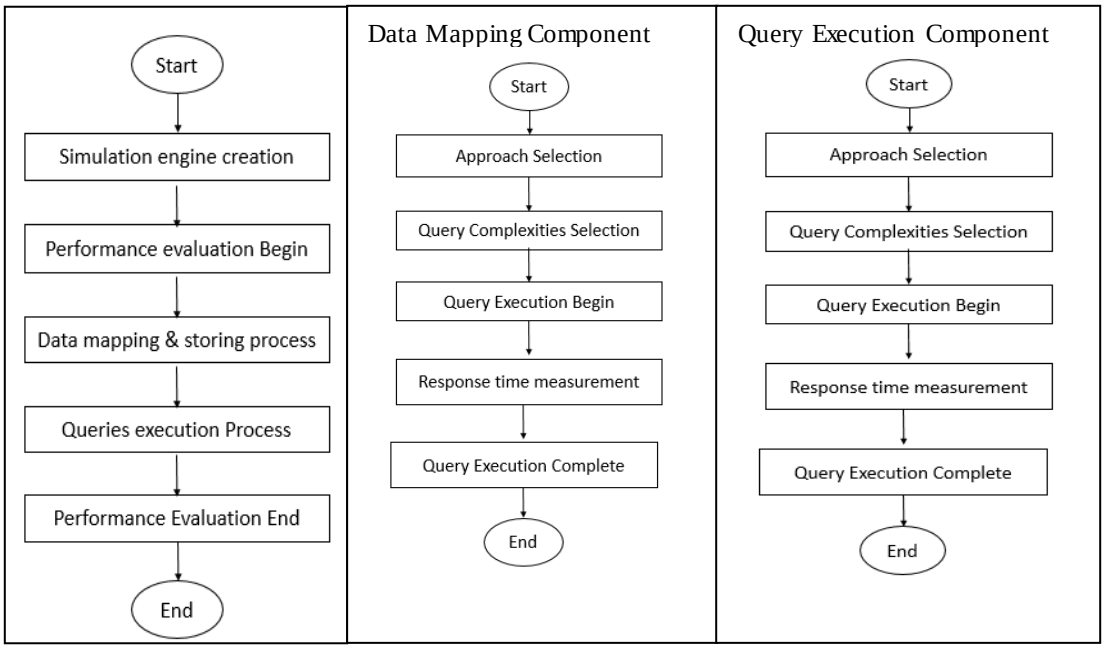

Figure 3. The flow of XMapDB-Sim evaluation

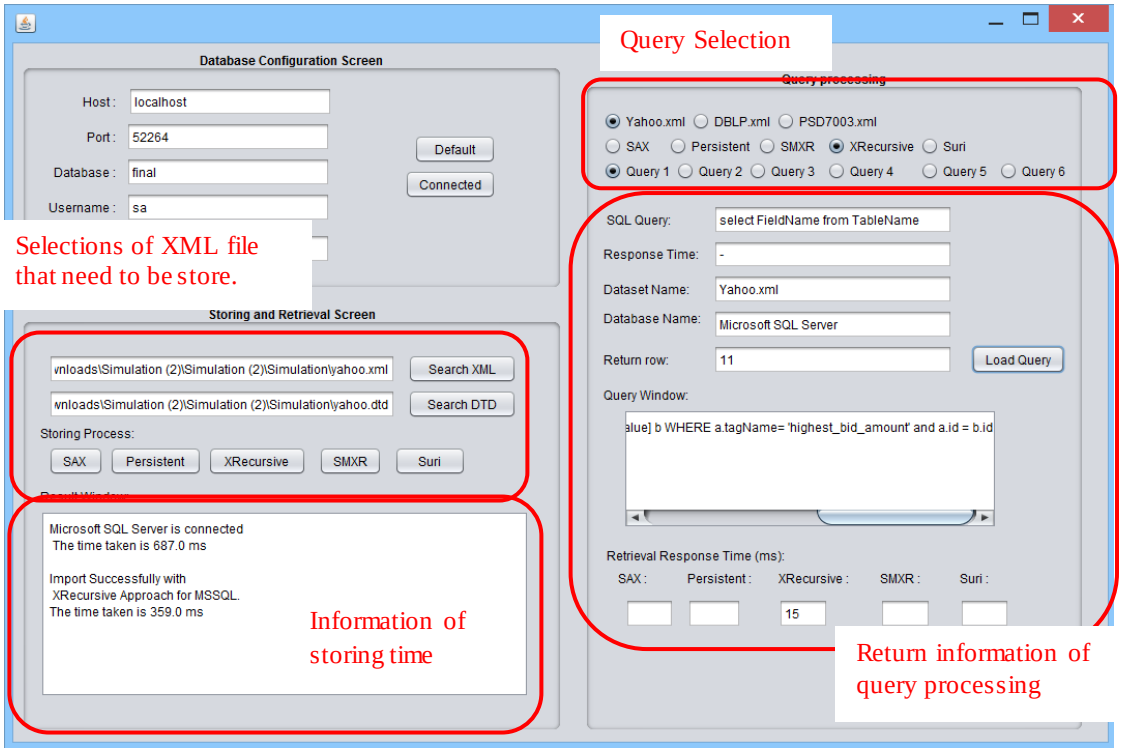

Figure 4. Main screen for the dataset and query selection

4. Performance Evaluation: Results and Discussion

Figure 3 depicts the flow of XMapDB-Sim, which consists of two main components, the (i) Data Mapping, and (ii) Query Execution. Firstly, the XML document will be loaded and read by document builder factory, and subsequently parsed into tree data model by the parser. Next, the XML trees will be converted into RDB using one of the three selected mapping algorithms. The storing time is then recorded. Successively, the query can be retrieve and the response time will be record and display in simulation engine. Figure 4 depicts the interface of XMapDB-Sim main screen.

4.1. Experimental Setup

XMapDB-Sim has been built to calculate response time of storing and retrieving of data by two different sizes of XML document: yahoo (small - 25KB), and protein (large - 0.7GB), which were obtained from University of Washington repository [22]. The characteristic of the dataset is depicted in Table 8. The evaluation test are perform on Intel i7-3630QM processor with 16 GB RAM running on Windows 8 64 bits machine.

Table 8. Selected Dataset for Experimental Evaluation

Dataset	File (MB)	Characteristic Description
Yahoo	0.024	Max. tree level: 5, structured
Protein	700	Max. tree level: 5, unstructured and recursive data

4.2. Experimental Results and Discussion

Table 9 shows storing time for the three approaches on the two datasets. The storing time is captured by taking the average of three consecutive running times. From the result, we observed that among the approaches, XRecursive stores data the fastest while SMX/R approach is the least efficient in data storing. All of these approaches apply DOM parser in parsing XML document; as such, the heap size needs to be increase to load PSD7003. For more consistent and accurate results, unused programs and connection have been shut off

during this process. All three approaches takes a large amount of time on PSD7003; this can conclude the statement that DOM parser works best on small size dataset but not suitable for large size dataset.

Table 9. Data Storing Time

Approach	Yahoo	PSD7003
XRecursive	313 ms	2 hr 21 min
SS	344 ms	3 hr 41 min
SMX/R	422 ms	6 hr 20 min

In evaluating the retrieval, six queries are prepared in the function buttons (see Figure 4) in the evaluation screen. Through clicking on the respective query button, the simulation engine process-es the query from the relational database. The time taken for retrieval is recorded and repeated for three times. Then, the average time based on the three consecutive runs is calculated; usually the first run result is discarded to ensure that the cache memory does not contain any unnecessary data that can affect the response time. In addition, the total number of rows returned from each query is recorded to check for the correctness.
The performance evaluations for Yahoo dataset to test on six queries are not resulting into much different values since the dataset size and the returned result are small.

4.2.1. Retrieval Evaluation on Yahoo Dataset

Table 10 depicts the six queries description and the corresponding query node. Q1 to Q3 are path queries (simple queries with P-C, A-D and mixed) while Q4 to Q6 are twig queries (complex or branching queries with P-C, A-D and mixed).

Table 10. Query Description on Yahoo Dataset

Query No.	Query Pattern	Query Description	XPath Expression
Q1	Path query with P-C relationship	List the highest_bid_amount for bid_history under listing.	root/listing/bid_history/highest_bid_amount
Q2	Path query with A-D relationship	List listing with any memory node	root/listing//memory
Q3	Path query with both P-C and A-D relationship	List out all the high_bidder with its respective immediate node bidder_rating	root//high_bidder/bidder_rating
Q4	Twig query with P-C relationship	List bid_history and item_info with their respective immediate node highest_bid_amount and item_info with memory	root/listing[/bid_history/highest_bid_amount]/item_info/memory
Q5	Twig query with A-D relationship	List highest_bid_amount and current_bid.	root[//highest_bid_amount]//current_bid
Q6	Twig query with both P-C and A-D relationship	List out high_bidder and seller_name where high_bidder contains immediate node bidder_rating and bidder_name.	root/listing[//high_bidder[/bidder_rating]/bidder_name]//seller_name

After the mapping process, the XML data were shredded and stored into relational table. As such, query retrieval will be performed using SQL command. Table 11 illustrates the SQL commands on various approaches. Even though these queries were retrieved from two tables (since these three approaches store the shredded data into at most two tables), yet, the attributes in each table are different. Henceforth, the time to retrieve these queries varies.

Table 11. Comparison on various approaches for query retrieval on Yahoo dataset

Query	SMX/R	XRecursive	SS	Returned Result
Q1	SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.nodeName = 'highest_bid_amount' and a.id = b.id	SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'highest_bid_amount' and a.id = b.id	SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pid = 'bid_history' and a.id = b.id and b.name = 'highest_bid_amount'	11
Q2	SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.path LIKE '%item_info%' and b.nodeName = 'memory' and a.id=b.id	SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'memory' and a.id = b.id and a.pid='item_info'	SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pos LIKE '%item_info%' and b.name = 'memory' and a.id=b.id	11
Q3	SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.path LIKE '%high_bidder%'and a.id=b.id and b.nodeName = 'bidder_rating'	SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'bidder_rating' and a.id = b.id and a.pid= 'high_bidder'	SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pos LIKE '%high_bidder%' and a.pid = 'high_bidder' and a.id=b.id and b.name = 'bidder_rating'	11
Q4	SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.nodeName = 'memory' and a.id = b.id UNION all SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.path LIKE '%bid_history%' and b.nodeName = 'highest_bid_amount' and a.id=b.id	SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'highest_bid_amount' and a.id = b.id UNION all SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'memory' and a.id = b.id	SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pos LIKE '%listing%' and b.name = 'highest_bid_amount' and a.id=b.id UNION all SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pid = 'item_info' and a.id = b.id and b.name = 'memory'	22
Q5	SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.path LIKE '%bid_history%' and b.nodeName = 'highest_bid_amount' and a.id=b.id UNION all SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.nodeName = 'current_bid' and b.path LIKE '%auction_info%' and a.id = b.id	SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'highest_bid_amount' and a.id = b.id UNION all SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'current_bid' and a.id = b.id	SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pid = 'bid_history' and b.name = 'highest_bid_amount' and a.id = b.id UNION ALL SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pos LIKE '%listing%' and b.name = 'current_bid' and a.id=b.id and a.pid ='auction_info'	22
Q6	SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.path LIKE '%listing%' and b.nodeName = 'seller_name' and a.id=b.id UNION all SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.nodeName =	SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'bidder_rating' and a.id = b.id UNION all SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'bidder_name' and a.id = b.id	SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pid = 'high_bidder' and b.name = 'bidder_name' and a.id = b.id UNION ALL SELECT a.value FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pos LIKE '%listing%' and b.name = 'seller_name' and a.id=b.id UNION ALL SELECT a.value	33

Query	SMX/R	XRecursive	SS	Returned Result
	'bidder_name' and b.path LIKE '%high_bidder%' and a.id = b.id UNION all SELECT a.value FROM [fyp].[dbo].[pathIndexTable_yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.nodeName = 'bidder_rating' and b.path LIKE '%high_bidder%' and a.id = b.id	UNION all SELECT b.value FROM [fyp].[dbo].[tag_structure_yahoo] a, [fyp].[dbo].[tag_value_yahoo] b WHERE a.tagName= 'seller_name' and a.id = b.id	FROM [fyp].[dbo].[data_yahoo] a, [fyp].[dbo].[node_yahoo] b WHERE a.pid = 'high_bidder' and b.name = 'bidder_rating' and a.id = b.id	

Figure 5 depicts the query retrieval results. Based on the figure, SS has better performance, while XRecursive and SMX/R are comparable. To ensure correctness of implementation, we checked and observed that all the approaches returned the same number of retrieval results.

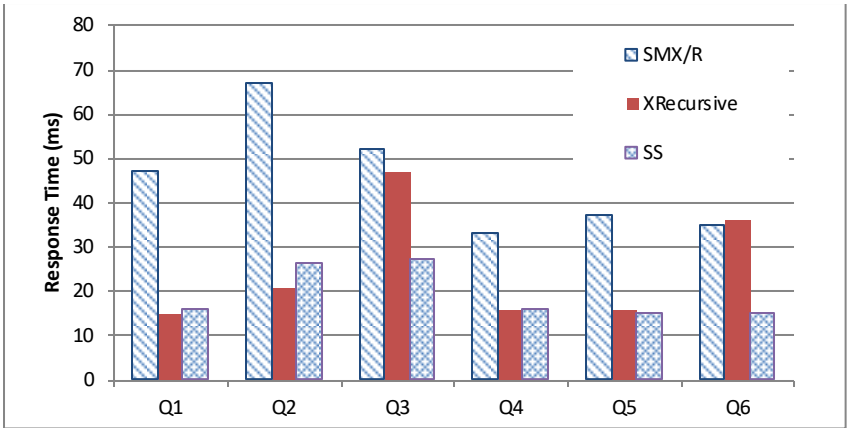

Figure 5. Query Evaluation on Yahoo dataset

From the result obtained, we notice the following:

1. For cases involving simple queries (Q1 to Q3), the SMX/R approach perform less good compared to the others.
    - This is due to the fact that it requires more time to handle join as it needs to retrieve the Path ID (PID) in the Path_Index_Table table to match the Path ID (PID) in the Path_Table. Similarly, the same join is required in XRecursive approach, yet, the performance is better due to the use of the attribute ParentID recursively to form nested query.
2. For cases involving complex queries (Q4 to Q6), the SS approach has the best performance.
    - The SS approach has the best performance due to the fact the labeling scheme on node position is effective to provide quick determination on the relationship among the nodes.
    - The XRecursive approach perform less good compared to SS approach because it uses the ParentID column recursively to form a nested query, which resulted into the highest storage space for RDB scheme. This increases the search space that is required to answer the necessary query. Subsequently, this will produce unnecessary intermediate results, which does not participate in the final solution.
3. All approaches has rather similar mapping scheme, i.e., uses two tables for storage, thus for smaller dataset, result shown would not be significant.

4.2.2. Retrieval Evaluation on Protein Dataset

Table 12 depicts the six queries description and the corresponding query node. Q1 to Q3 are path queries (simple queries) while Q4 to Q6 are twig queries (complex or branching queries).

Table 12. Query Description on Protein Dataset

Query No.	Query Pattern	Query Description	XPath Expression
Q1	Path query with P-C relationship	List all organism for proteinEntry.	ProteinDatabase/ProteinEntry/organism
Q2	Path query with A-D relationship	List out all refinfo under ProteinDatabase	ProteinDatabase//reinfo
Q3	Path query with both P-C and A-D relationship	List the ProteinEntry which consists of immediate reference node with any citation.	//ProteinEntry/reference//citation
Q4	Twig query with P-C relationship	List the ProteinEntry with its respective immediate node header and reference which consists of refinfo with its respective immediate citation node.	//ProteinEntry[/header]/reference/refinfo/citation
Q5	Twig query with A-D relationship	List out all the information that consists of accinfo and refinfo with their respective immediate accession and volume node.	ProteinDatabase[//accinfo/accession]//refinfo/volume
Q6	Twig query with both P-C and A-D relationship	List out all the information that consists of accinfo and refinfo with their respective immediate xrefs and authors nodes by which xrefs has immediate xref node with its immediate uid and db nodes while authors has immediate author node.	ProteinDatabase[//accinfo/xrefs[/xref/uid]/db]//refinfo/authors/author

Table 13 illustrates the query retrieval using SQL command on various approaches while Figure 6 depicts the query retrieval results. Based on the figure, XRecursive has the best performance in all cases. To ensure the correctness of the implemented algorithm, we counted on the number of returned result. It can be observed that all the approaches returned the same number of retrieval result.

Table 13. Comparison on various approaches for query retrieval on Protein dataset

Query	SMX/R	XRecursive	SS	Returned Result
Q1	SELECT b.value FROM [psd].[dbo].[pathTable] a LEFT JOIN [psd].[dbo].[pathIndexTable] b on a.id = b.id where a.nodeName = 'organism' and a.path LIKE '%ProteinEntry%'	SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id where a.pid = 'ProteinEntry' and a.tagName = 'organism'	SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id where b.pid = 'ProteinEntry' and a.name = 'organism'	262525
Q2	SELECT b.value FROM [psd].[dbo].[pathTable] a LEFT JOIN [psd].[dbo].[pathIndexTable] b on a.id = b.id where a.nodeName = 'refinfo' and a.path LIKE '%reference%'	SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id where a.pid = 'reference' and a.tagName = 'refinfo'	SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id where b.pid = 'reference' and a.name = 'refinfo'	314763
Q3	SELECT b.value FROM [psd].[dbo].[pathTable] a LEFT JOIN [psd].[dbo].[pathIndexTable] b on a.id = b.id where a.nodeName = 'citation' and	SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id where a.pid = 'refinfo' and a.tagName =	SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id where b.pid = 'refinfo' and a.name = 'citation'	314763

Query	SMX/R	XRecursive	SS	Returned Result
	a.path LIKE '%refinfo%'	'citation'		
Q4	SELECT b.value FROM [psd].[dbo].[pathTable] a LEFT JOIN [psd].[dbo].[pathIndexTable] b on a.id = b.id w here a.nodeName = 'citation' and a.path LIKE '%refinfo%' UNION ALL SELECT b.value FROM [psd].[dbo].[pathTable] a LEFT JOIN [psd].[dbo].[pathIndexTable] b on a.id = b.id w here a.nodeName = 'header' and a.path LIKE '%ProteinEntry%'	SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id w here a.pid = 'refinfo' and a.tagName = 'citation' UNION ALL SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id w here a.pid = 'ProteinEntry' and a.tagName = 'header'	SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id w here b.pid = 'refinfo' and a.name = 'citation' UNION ALL SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id w here b.pid = 'ProteinEntry' and a.name = 'header'	577288
Q5	SELECT b.value FROM [psd].[dbo].[pathTable] a LEFT JOIN [psd].[dbo].[pathIndexTable] b on a.id = b.id w here a.nodeName = 'accession' and a.path LIKE '%accinfo%' UNION ALL SELECT b.value FROM [psd].[dbo].[pathTable] a LEFT JOIN [psd].[dbo].[pathIndexTable] b on a.id = b.id w here a.nodeName = 'volume' and a.path LIKE '%refinfo%'	SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id w here a.pid = 'accinfo' and a.tagName = 'accession' UNION ALL SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id w here a.pid = 'refinfo' and a.tagName = 'volume'	SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id w here b.pid = 'accinfo' and a.name = 'accession' UNION ALL SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id w here b.pid = 'refinfo' and a.name = 'volume'	550383
Q6	SELECT a.value FROM [fyp].[dbo].[pathIndexTable_ yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.path LIKE '%listing%' and b.nodeName = 'seller_name' and a.id=b.id UNION all SELECT a.value FROM [fyp].[dbo].[pathIndexTable_ yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.nodeName = 'bidder_name' and b.path LIKE '%high_bidder%' and a.id = b.id UNION all SELECT a.value FROM [fyp].[dbo].[pathIndexTable_ yahoo] a, [fyp].[dbo].[pathTable_yahoo] b WHERE b.nodeName = 'bidder_rating' and b.path LIKE '%high_bidder%' and a.id = b.id	SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id w here a.pid = 'xref' and a.tagName = 'uid' UNION ALL SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id w here a.pid = 'xref' and a.tagName = 'db' UNION ALL SELECT b.value FROM [psd].[dbo].[tag_structure] a LEFT JOIN [psd].[dbo].[tag_value] b on a.id = b.id w here a.pid = 'authors' and a.tagName = 'author'	SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id w here b.pid = 'xref' and a.name = 'db' UNION ALL SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id w here b.pid = 'xref' and a.name = 'uid' UNION ALL SELECT b.value FROM [psd].[dbo].[node] a LEFT JOIN [psd].[dbo].[data] b on a.id = b.id w here b.pid = 'authors' and a.name = 'author'	8655551

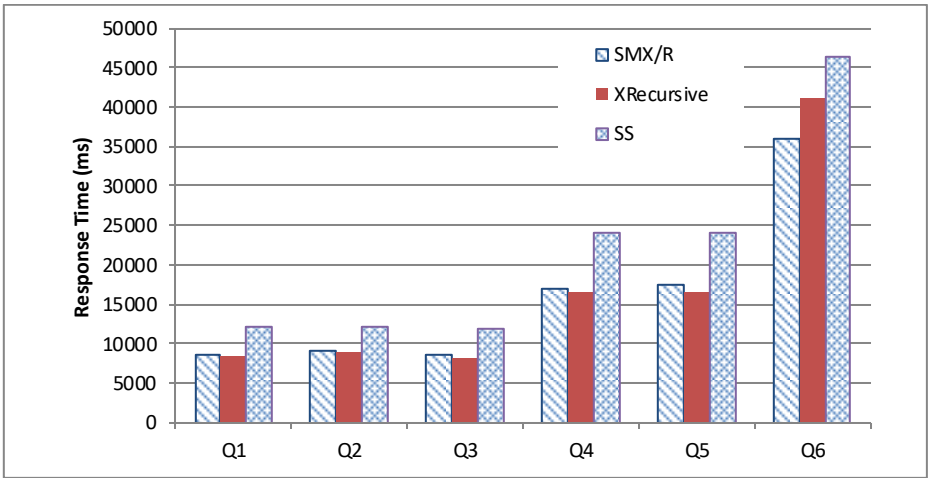

Figure 6. Query Evaluation on Protein dataset

From the result obtained, in the Protein dataset, we notice the following:

a. Protein dataset is huge, unstructured and contains recursive nodes. The table structure of XRecursive facilitates the retrieval for recursive query. It also uses ParentID recursively to form nested queries in order to maintain the P-C and A-D relationships inside these queries.
b. The SS approach performance degraded especially on the huge and unstructured dataset because they store more information (six columns in Data table) in their RDB scheme. This definitely increases the search space.

The SMX/R approach consumed more storage space than XRecursive and SS, as it stores distinct path information for every node (root-to any node) of the document and all the information about elements, attributes and texts in Path and Path_index table.

Although the XRecursive approach used the least storage space due to its simplicity scheme that contains only two tables, it still stores all the information about inner nodes and the Type column. On the other hand, SS is good for smaller dataset, but is not scalable for larger dataset with recursive or nested nodes.

5. Conclusion and Future Works

Using the correct mapping scheme is essential to ensure that the best performance based on various datasets under various environments. From the paper, one may decide which approach to adopt depends on the nature of business, i.e., frequent updating, real-time and ad-hoc retrieval, unstructured data, streaming data and so on. To facilitate XML for full-fledged support as data exchange over the Web, it is crucial that the mapping scheme is robust enough to support dynamic updates.

Our future work is to implement more recent approaches such as XAncestor, XMap and mini-XML into XMapDB-Sim. In addition, we will perform the complexity analysis on these approaches.

References

[1] JC Chemiavsky and CH Smith. "A Binary String Approach for Updates in Dynamic Ordered XML Data". *IEEE Transactions on Knowledge and Data Engineering*. 2010; 22: 602-607.
[2] J Liu and XX Zhang. "Dynamic labeling scheme for XML updates", *Knowledge-Based System Journal*. 2016; 106: 135-149. doi: 10.1016/j.knosys.2016.05.039.
[3] GZ Qadah. "Indexing techniques for processing generalized XML documents". *Computer Standards & Interfaces*. 2017; 49: 34-43.
[4] A Chebotko, M Atay, S Lu and F Fotouhi. "XML subtree reconstruction from relational storage of XML documents". *Data & Knowledge Engineering*. 2007; 62: 199-218.
[5] I Dweib, A Awadi and J Lu. "MAXDOR: Mapping XML Document into Relational Database". *The Open Information Systems Journal*. 2009; 3: 108-122.

[6] A Qtaish and K Ahmad. "XAncestor: An efficient mapping approach for storing and querying XML documents in relational database using path-based technique". *Knowledge-Based Systems*. 2016; 114: 167-192.
[7] Z Bousalem and I Cherti. "XMap: a novel approach to store and retrieve xml document in relational databases". *Journal of Software*. 2015; 10(12): 1389-1401.
[8] I Tatarinov, SD Viglas, K Beyer, J Shanmugasundaram, E Shekita and C Zhang. "*Storing and querying ordered XML using a relational database system*". in Proceedings of the 2002 ACM SIGMOD International conference on Management of data. 2002: 204-215.
[9] S Prakas, SS Bhowmick and S Madria. "SUCXENTdatabase mapping techniques". *Advances in Computer Science and Information Technology*. 2015; 2(2): 162–166.
[10] SC Haw and CS Lee. "Data storage practices and query processing in XML databases: A survey". *Knowledge-Based System*. 2011; 24(8): 1317-1340.
[11] M Mourya and P Saxena. "Survey of XML to relational database mapping techniques". *Advances in Computer Science and Information Technology.* 2015; 2(2): 162–166.
[12] X Yuan, X Hu, D Wu, H Zhang and X Lian. "XML data storage and query optimization in relational database by XPath processing model". *Journal of Software*. 2015; 8(4): 809-816.
[13] F Abduljwad, N Wang and D Xu. "*SMX/R: Efficient way of storing and managing XML documents using RDBMSs based on paths*". in Proceedings of International Conference on Computer Engineering and Technology. 2010; 1: 143-147.
[14] 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 Transactions on Internet Technology*. 2001; 1(1): 110-114.
[15] MAl Fakharaldien, JM Zain and N Sulaiman. "XRecursive: An efficient method tostore and query XML documents". *Australian Journal of Basic and Applied Sciences*. 2011; 5(12): 2910-2916.
[16] P Suri and D Sharma. "A Model Mapping Approach for storing XML documents in Relational databases". *International Journal of Computer Science Issues*. 2012; 9(3): 495-498.
[17] J Qin, SM Zhao, SQ Yang and WH Dou. "XPEV: A Storage Model for Well-Formed XML Documents". *Lecture Notes in Computer Science*. 2005; 3613.
[18] J Ying, S Cao and Y Long. "*An efficient mapping approach to store and query XML documents in rela-tional database*". in Proceedings of International Conference on Computer Science and Network Technology. 2012: 2140–2144.
[19] P O'Neil, E O'Neil, S Pal, I Cseri, G Schaller and N Westbury. "*ORDPATHs: Insert-friendly XML node labels*". in Proceedings of ACM SIGMOD International Conference on Management of Data. 2004: 903-908.
[20] S Subramaniam, SC Haw and KH Poo. "s-XML: an efficient mapping scheme to bridge XML and relational database". *Knowledge-Based Systems*. 2012; 27: 369–380.
[21] H Zhu, H Yu, G Fan and H Sun. "*Mini-XML: An efficient mapping approach between XML and relational database*". in Proceedings of International Conference on Computer and Information Science. 2017: 839-843.
[22] UW (University of Washington XML Repository). 2017. Retrieved from http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/www/repository.html