DWG2XML: GENERATING XML NESTED TREE STRUCTURE FROM DIRECTED WEIGHTED GRAPH

Kate Y. Yang, Anthony Lo, Tansel Özyer, and Reda Alhajj
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada

Keywords: XML, directed graph, nested structure, constrained conversion, relational database.

Abstract: The overall XML file length is one of the critical factors when we need to transfer a large amount of data from relational database into XML. Especially in the nested tree structure of XML file, redundant data in the XML file can add more cost on database access, network traffic and XML query processing. Most previous automated relational to XML conversion research efforts use directed graphs to present relations in the database and nested trees in the XML structure. However, they all ignore that different combinations of tree structures in a graph can have a big impact on the XML data file size. This paper addresses this nested structure data file size problem. It proposes a module that can find the most convenient tree structure for the automated relational to XML conversion process. It provides a plan generator algorithm to list all the possible tree structures in a given directed weighted graph. Also it analyzes the data size of each plan and shows the convenient tree structure to the user. It can finally create the targeted XML documents for the user.

1 INTRODUCTION

XML is becoming one of the most widely used technologies for data exchange over the internet. But most business data is currently stored in relational database systems, which have been well developed for a long time. So, there are large demands for transforming such relational databases into XML documents. Considerable amount of work has been done to help people in this area. Basically, we can group those studies into two categories based on the amount of data to be transferred from relational database to XML.

The first covers the case when only part of the database is of interest. Specific database queries are needed in order to fetch the target data. XML Extender (Chaudhuri, 2003), SilkRoute (Fernandez et al, 2002) and DB2XML (Shanmugasundaram et al, 2001) are all under this category. However, all these tools need human experts working on mapping from the relational schema to the XML schema. Therefore, when large amount of relational schemas and data need to be translated into XML documents, a significant investment of human effort is required to initially design the target schema. The second group of approaches concentrates on automatically inferring XML schema out of the relational database schema using semantic constraints, such as Net & CoT (Lee et al, 2002a; Lee et al, 2002b; Lee et al 2001), ConvRel & Con2XMl (Duta, 2004) and the reverse engineering based approach for converting Legacy RDB to XML (Wang et al, 2004; Lo et al, 2004). They can convert data from the relational database to XML without human input. Finally, VIREX (Lo et al, 2004) is an approach capable of handling both strategies.

In this paper, we propose a method which focuses on the second group of approaches. To justify for the motivation of our method, we start with a brief introduction for each recent approach in this group; we mainly address the nested XML tree structure problem in those approaches. Then we propose our DWG2XMl method which extends existing studies. It has an algorithm that can generate all possible XML nested tree plans from a given directed graph. We provide each plan's data file size and compact rate to help choosing a good nested structure and then we generate XML document for the selected plan.

The rest of the paper is organized as follows. Section 2 is an overview of three different approaches for conversion from relational databases

Y. Yang K., Lo A., Özyer T. and Alhajj R. (2005).
DWG2XML: GENERATING XML NESTED TREE STRUCTURE FROM DIRECTED WEIGHTED GRAPH.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 19-26
DOI: 10.5220/0002512500190026

into XML, including their contributions and drawbacks. Our DWG2XML method is presented in Section 3. Section 4 is the conclusions.

2 OVERVIEW OF THE EXISTING APPROACHES

2.1 Reverse engineering approach

Alhajj (2003) presents a reverse engineering approach that extracts the entity-relationship (EER) schema from the relational schema. The concepts and mechanism provided contribute to legacy database maintenance, re-engineering or updating to another database technique. Based on the analysis of the relationships between tables in a legacy database, a relational intermediate directed (RID) graph consistent with the EER diagram is derived to express all possible unary, binary and nary relationships between the given relations. Then, it develops algorithms to eliminate the symmetry and transitivity in RID, if exist. It also identifies *is-a* links in the RID graph to deliver an optimized RID as the final outcome, which can be used to derive the XML schema. Such a conversion approach has been implemented by Wang, et al (2004). Then they translate the RID graph into XML schema in a process called forward engineering. A flat XML schema is automatically derived from the RID graph. Our DWG2XML approach can be easily seen as an extension to complete RID to nested XML schema translation; this is all described in Section 3.

2.2 CoT and NeT

Lee, *et al* (2002a; 2002b; 2001) proposed an approach for creating both flat and nesting XML structures from the relational database schema. The Flat Translation (FT) converts each table into a flat element structure. The Nesting-based Translation (NeT) derives nested structures from a flat relational model by the use of the nest operator. This nest operator process is applied to a single table at a time and it can create nested structures only for non-normalized tables in normalized databases. Net is useful to decrease data redundancy in non-fully normalized relational databases. But it only works on tables one by one and depends on the relational schema as well as the actual data stored in the database.

Then Lee *et al* extended the nesting approach to multiple tables, using Constraints-based Translation (CoT) algorithm. It is one of the first approaches that deal with relationships. The source database contains several interconnected tables and based on the cardinality of the binary relationships, two types are identified one-to-one (1:1) and one-to-many (1:M). A directed Inclusion Dependency (IND) Graph of tables is created from which an empirical way to nest XML structures is identified. However, a table can only have one child. If there are more children relations for a particular parent table, these relationships are simulated by using reference key expression.

2.3 ConvRel and Conv2XML

Conv2XML and ConvRel are two algorithms proposed by Duta, *et al* (2004) for converting relational schema to XML Schema, focusing on preserving the source relationships and their structural constraints.

ConvRel analyzes each type of relationship and determines a set of candidate XML structures capable of representing the analyzed relationship type. The possible XML structures are classified as Parent-Child, Child-Parent nested structures, flat structure using keyref references and combination nested with keyref structure. Those structures are filtered depending on criteria such as the nested and compact structure, and the size of XML data file. ConvRel classifies each type of possible relationship in the database into the best XML structure spot. But this approach only works with a single relationship at a time; it is not applicable for relationships involving more that two tables.

Conv2XML algorithm extends ConvRel to create a nested structure for the entire database. It uses a graph representation that combines all structures discovered previously in ConvRel. In this graph, the vertices are tables and edges represent connections between tables as defined by ConvRel. Two categories of edges exist in this directed graph: 1) full edges representing nested structures; and 2) dotted edges representing relationships for the reference key. The ConvRel algorithm is thereby transformed into the problem of discovering trees in a directed graph.

Compared to the NeT and CoT approach, ConvRel and Conv2XML approach solved the unary relationship problem between tables. It also can present multiple tables as a tree structure. However, from the directed graph, there exist different nested tree structures. The method proposed by Duta *et al* is depth-first algorithm, which ends up with only one tree structure solution. As a result, DWG2XML as described in this paper is more comprehensive; it considers all possible tree structures instead.

3 THE DWG2XML APPROACH

3.1 The motivation

In Section 2, we briefly discussed three different approaches for automatically converting relational database to XML structure. Even though those systems vary in terms of the kind of database (legacy versus catalog-based) they can convert, they all share a common feature of using the directed graph to capture the relationships in the database. The directed graph is different from the XML tree structure we need. In a directed graph, for any given node there is no restriction on the number of parent and child nodes. But in the XML tree structure, a node can have only one parent. So when relationships between tables in the database are presented as a directed graph, there can be different ways to construct nested tree structures. Selecting a particular one as the most appropriate choice for the XML structure will have impact on the overall XML data file length and on the database query access time. This problem has not received enough attention in the literature yet.

As a result, DWG2XML approach presented in this paper may be considered as extending the previous work that concentrates on the directed graph. In particular, our approach focuses on analyzing the given directed graph, finding all possible nested tree structures that can be used to construct the XML document and selecting the most appropriate one to generate an XML document.

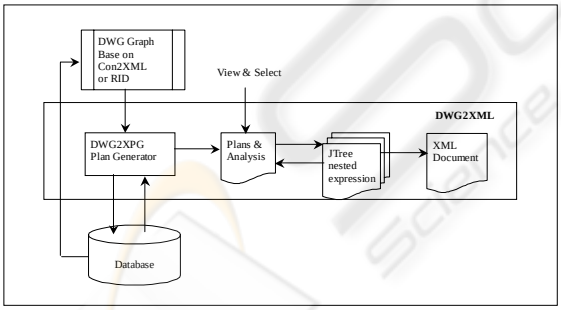

Figure 1: DWG2XML System Architecture

3.2 DWG2XML overview

The architecture of DWG2XML is shown in Figure 1. The DWG graph is generated by the Con2XML algorithm. It can be either RID graph from reverse engineering, or a directed graph from Net with some minor changes. An example RID graph is shown in Figure 2. DWG2XPG plan generator finds all tree structure combinations for the input DWG graph and saves each combination as a tree structure plan. Then, it queries the database to analyze each plan's data file size and data compactness rate; the results are summarized and displayed to the user as a plan data table. Users can view the tree structure of each plan as a JTree expression. From plans with the same data file length and compactness rate, the user can always pick the one that has more semantics.

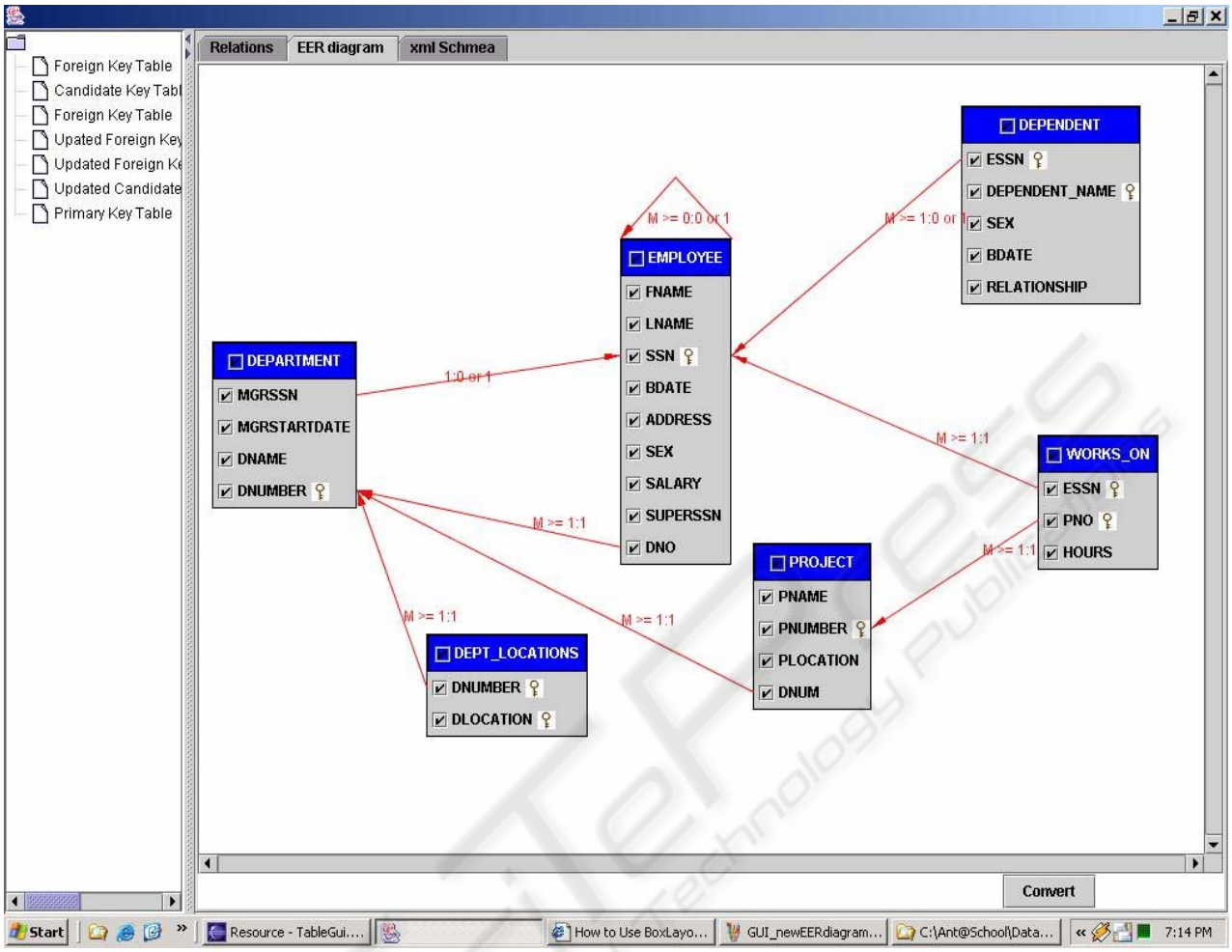

Figure 2: Example RID graph generated by VIREX (Lo et al, 2004)

3.3 An example database and the corresponding DWG graph

There are six tables in our example database from a time schedule system; it has the following relational schema in which primary keys are underlined and foreign keys are italic:

Groups (groupName)

Users(loginName, name, email, accessLevel, password, *groupName*)

Appointments(appointmentId, loginName, sDate, startTime, endTime, note, *meetingId*)

Meetings(meetingId, type, chairLoginName, *meetingRoomNum*)

Notices(noticeId, *loginName*, *meetingId*, readMark, message)

MeetingRooms(meetingRoomId, seatNumber, projector, multimedia)

The structural constraints of all relationships that exist in the example relational schema are:

Groups (1; 1): (1; M) Users

Users (1; 1): (0; M) Appointments

Users (1; 1): (0; M) Notices

Appointments (1; M): (0; 1) Meetings

Meetings (1; 1): (1; M) Notices

MeetingRooms (1; 1): (0; M) Meetings

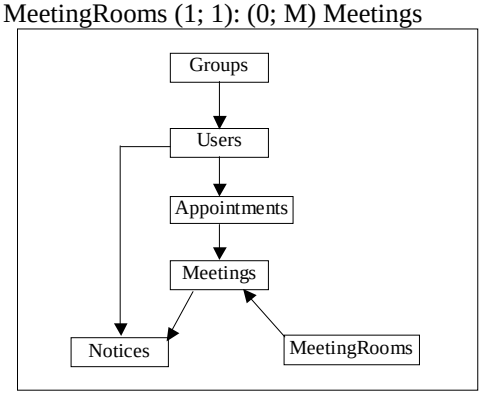

Figure 3: Input DWG graph

We can generate a corresponding directed graph (shown in Figure 3), which includes all the above relationships. We implement this graph as a directed weighted graph (DWG), which has the following properties:

1. It is a directed graph; implemented by having each edge connecting source node to target node. Here, source represents the relation of the primary key and target is the relation that contains the corresponding foreign key.
2. Weight has predefined values based on the relationship type. In this implementation, one-to-one relationships get the weight 1, one-to-many relationships get the weight 2, and dotted (keyref) relationships are assigned the weight 3.
3. It is possible to have several un-connected trees in one DWG graph

3.4 Plans generating algorithm

In this section, we present the plan generator algorithm, which generates all possible nested tree XML structures. We analyze the incoming edges of a node. Having more than one incoming edge for a node means that there is more than one path to reach this node in the graph. So, we have to list all possible paths to reach each node in the graph. Steps of the plans generation algorithm are given next:

Input: DWG graph

Output: Vector Plans, each plan has nested tree XML structure.

Variables: - Plans[ ] is DWG graph Vector
- processingNodeQ[ ] is a vector to keep all nodes waiting for process;
- unprocessingNodeQ[ ] is for the unconnected nodes in the graph. It is initialized with all nodes in DWG

Steps:

1. Select a starting node, push it into processingNodeQ, and mark it as a working node.
2. For each outgoing edge of the working node, get the target node of the outgoing edge, i.e., the child of the working node. Add the child node and connecting edge into the corresponding DWG graph to Plans [ ]. For the first node, create a DWG graph and add to Plans [ ]. Push each child node into a processing Queue.
3. For each incoming edge of the working node, get the source node of the incoming node, i.e., the parent node. Push this source node into processingNodeQ.
   a. If there are more than two incoming edges, for each plan in the plans vector, make a new copy.
   b. In the existing old plan, add the source node in plan, add the incoming edge as weight 3, presenting no parent-child relationship.
   c. In the duplicated new plan, add source node in the plan, add outgoing edge as edge with weight 2, and mark all other outgoing edges as weight 3.
4. After checking both incoming and outgoing edges of the working node, remove it from the processingNodeQ and unprocessingNodeQ; pop up the next node from processingNodeQ; mark it as the new working node; and go to Step 1.
5. If unprocessingNodeQ is not empty, pop the next node from unprocessingNodeQ, if any, and go to Step 1.

Step 3 of the plan generator algorithm guarantees only one parent for each node. Step 5 guarantees that all disconnected nodes have been processed.

After we apply the plan generator algorithm to the example input graph, we can generate the four plans shown in Figures 4-7. As we can see, the nested structure plan 3 (Figure 6) has the most nested structure. MeetingRooms table has no nested data. It will be converted to a flat structure in the XML document.

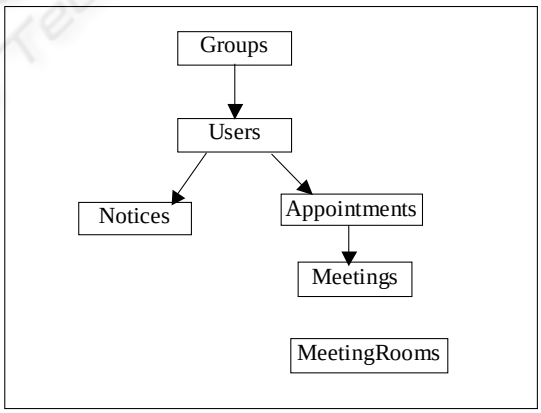

Figure 4: Nested structure plan 1

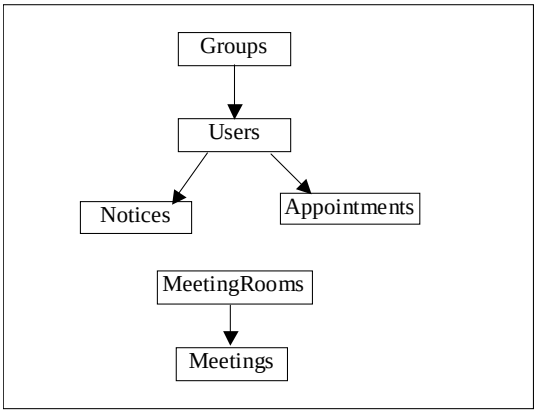

Figure 5: Nested structure plan 2

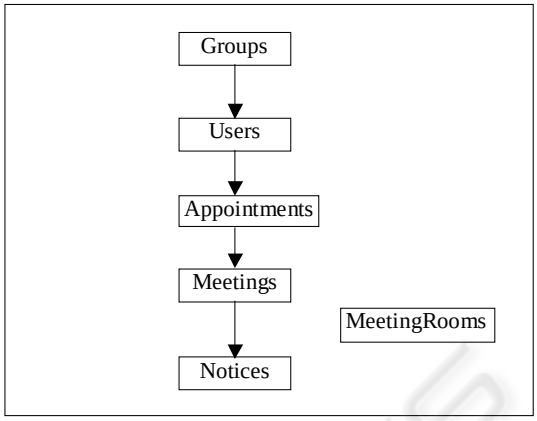

Figure 6: Nested structure plan 3

In nested structure plan 1 (Figure 4), Users data has Notices data set and Appointments subtree. In plan 2 (Figure 5), Users data has two nested datasets and MeetingRooms data has one data set. The last plan (Figure 6) has two trees with equal number of nodes. These different nested plans have different data file sizes and compactness rates. This is analyzed in more details in the next section.

3.5 Find a good plan

After we have all the possible solutions for XML nested structures. We have to choose the best one to convert to XML document. The XML data file size is one of the most important factors for choosing the good plan. Since we convert the whole database into XML, the smallest the data file size is, the less are duplicated data and relational database query/access time. Considering the participation ratio we have for each given relationship, if all parents and their children nodes have all mandatory one-to-one and one-to-many relationships, then all plans can have almost the same data file size. However, if there is -Parent(0,1):(1,M)Child- kind of relationship, then the data in the child branch can have a heavy duplication, depending on the number of levels below this child node.

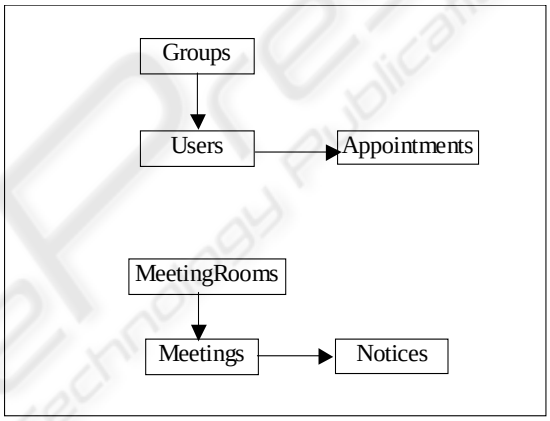

Figure 7: Nested structure plan 4

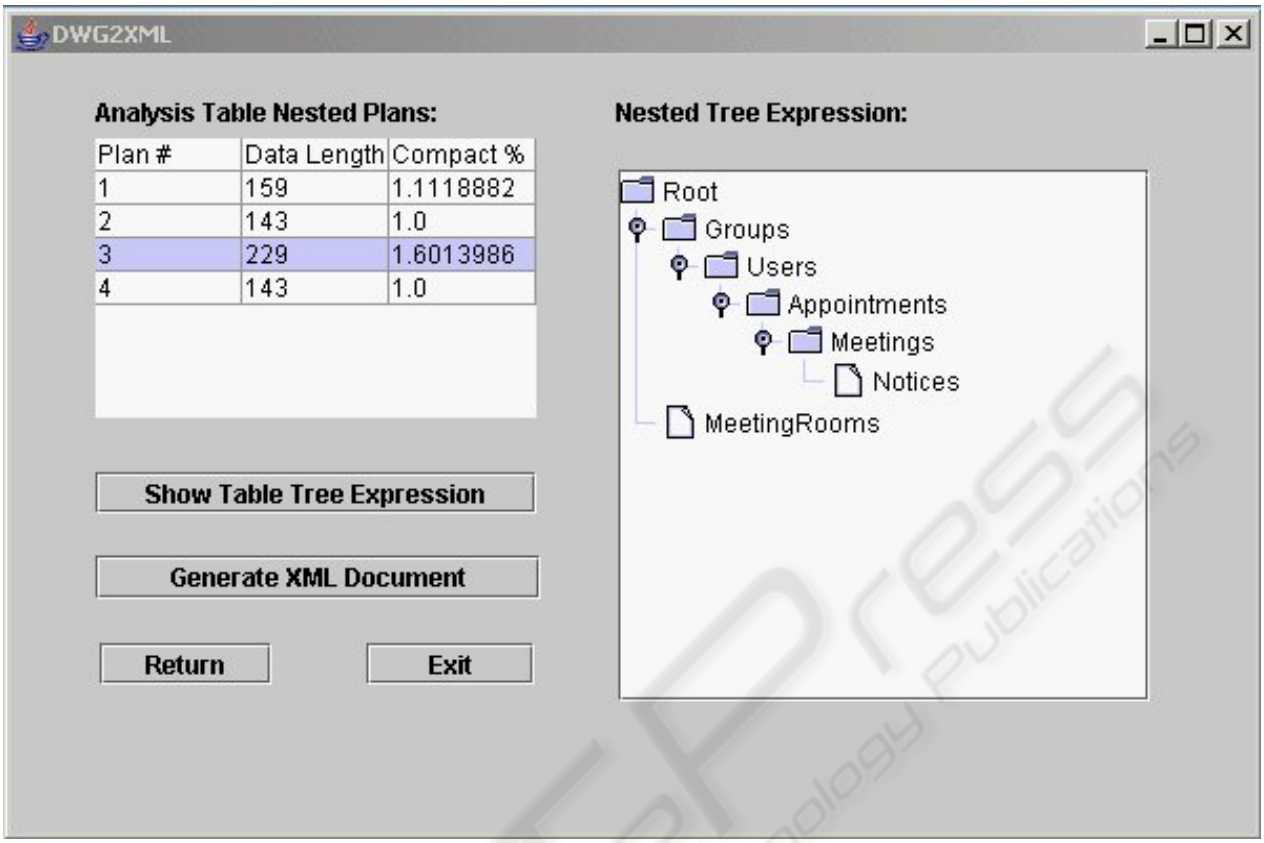

Figure 8: User interface

In our implementation, we use three hash tables to keep all the metadata we need from the relational database, including primary keys, foreign keys, and columns for each table. We only query the metadata once to save the database access time. By using this information, we can calculate simple data file size for one-to-one relationship. For partially duplicated nested tables, we can compose SQL query from primary-foreign keys set constraint and query the data file size from the database.

One of the screen from the developed approach is shown in Figure 8; it displays the data files size and compactness rates for the different plans produced by our approach. The right hand panel has a JTree expression for the nested XML structure plan 3 (Figure 6). In our example, one appointment can have zero or one meeting and one meeting can involve more than one appointment in the relational database. So the Meetings table data are duplicated.

Table 1: Pseudo-code of extend post order traversal algorithm

```
ExtandPostOrderTravers (parentNode, tableTag)
{
    for (each outgoing edge of the working node)
    {
        workingNode=outgoing edge.getTargetNode()
        sqlQuery=composeQuery (parentNode, workingNode)
        dataset=queryDataSet(sqlQuery, workingNode)
        while(dataset is not empty)
        {
            rowData=getNextRowData (dataset)
            workingTableTag=createNewTag( )
            for (each column in the rowData)
            {
                columnTag=createNewTag( )
                workingTableTag.appandChild(columnTag)
            }
        }
        ExtandPostOrderTravers (workingNode, workingTag)
        parentTableTag.appandChild(workingTableTag)
    }
}
```

More over, in plan 3, all duplicated data in Meetings table have duplicated data in Notices table. Note that plan 2 and plan 4 have the same data file size without any duplicated data in our example. At this point, we can make decisions based on the

semantic meaning of the data. From Figure 5 and Figure 7, we can see that Notices data can be grouped under different nested trees. In this case, the one which is more meaningful to the user is the one that can be the best candidate to be selected.

3.6 XML data process

When we have a good plan, we can transform all content of the relational database into XML document according to the tree structure in the plan. We extend the post order traversal algorithm using recursion to tag table data in an XML file.

As shown in Table 1, we create a table tag for each tuple of the data in the relation, and query the dataset from its nested relation until we reach the leaves. Then when we return to the upper level, we attach the data to the tag as well as we close the tag until we reach the root.

4 CONCLUSIONS

We have presented our DWG2XML approach that can derive all possible nested XML structure plans from a given directed graph in order to minimize the XML file size and database access time. This approach extends and completes previous work on relational database to XML conversion. It improves the performance of the conversion technique by the ability of finding the smallest data file size nested XML structure for a relational database. The DWG2XML approach presented here has been implemented in Java with JDBC driver for MSDE database. It is capable of handling unary, one-to-one, one-to-many, many-to-many and nary (n>2) relationships. Using our approach, it is possible to produce the desired XML schema and document ranging from flat to nested structures.

REFERENCES

Fernandez M., Kadiyska Y., Suciu D. and Tan W., "SilkRoute: A Framework for Publishing Relational Data in XML," *ACM Transactions on Database Systems*, Vol.27, No.4, pp.438-493, 2002.

Shanmugasundaram J., el al, "Efficiently Publishing Relational Data as XML Documents," *The VLDB Joural*, Vol.10, pp.133-154, 2001.

Chaudhuri S., Kaushik R. and Naughton J., "On Relational Support for XML Publishing: Beyond Sorting and Tagging," *Proceedings of ACM SIGMOD Conference on Management of Data*, San Diego, CA, June 2003.

Alhajj R., "Extracting the Extended Entity-Relationship Model from a Legacy Relational Database," *Information Systems*, Vol.28, No.6, pp.597-618, 2003.

Wang C., Lo A., Alhajj R. and Barker K., "Converting Legacy Relational Database into XML Database through Reserve Engineering," *Proceedings of the International Conference on Enterprise Information Systems*, Porto, Portugal, Apr. 2004.

Duta A., Barker K., and Alhajj R., "ConvRel: Relationship Conversion to XML Nested Structures," *Proceedings of the ACM Annual Symposium on Applied Computing*, Cyprus, Mar. 2004.

Lo A., Alhajj R. and Barker K., "Flexible User Interface for Converting Relational Data into XML," *Proceedings of the International Conference on Flexible Query Answering Systems*, Springer-Verlag, Lyon, France, June 2004.

Lee D., Mani M. and Chu W.W., "Effective Schema Conversions between XML and Relational Models," *Proceedings of the European Conference on Artificial Intelligence* (ECAI), *Knowledge Transformation Workshop* (ECAI-OT), Lyon, France, July 2002.

Lee D., Mani M., Chiu F. and Chu W.W., "NeT & CoT: Translating Relational Schemas to XML Schemas Using Semantic Constraints," *Proceedings. of ACM International Conference on Information and Knowledge Management*, McLean, VA, Nov. 2002.

Lee D., Mani M., Chiu F., and Chu W.W., "Nesting-based Relational-to-XML Schema Translation," *Proceedings of the International Workshop on the Web and Databases*, Santa Barbara, CA, May 2001.