Keyword Search over Hybrid XML-Relational Databases

Liru Zhang [1] Tadashi Ohmori [1] and Mamoru Hoshi [1]

[1]Graduate School of Information Systems, The University of Electro-Communications, Tokyo, Japan
(Tel : +81-42-443-5618; E-mail: {zhangliru,omori} @ hol.is.uec.ac.jp)

Abstract: How to realize keyword search over XML databases (XML DB) or relational databases (RDB) is a today's hot topic. In this paper, we first point out that existing keyword-search methods over those databases cannot get sufficient results. Then, we propose a new keyword-search method over hybrid XML-Relational databases, and demonstrate that its answers are better than those of existing techniques. We propose a new join-operator for XML data, and utilize the new operator to enable keyword search in hybrid XML-Relational databases.

Keywords: hybrid database, keyword search, XML

1. INTRODUCTION

1.1 Background and Objective

Keyword search is a new popular function of databases to retrieve information by user-given keywords. A variety of semi-structured data, such as web contents expressed in XML (eXtensible Markup Language) format, has been stored in structural databases. Recently, major RDB management systems (e.g. [7], IBM DB2 V9 [9],) allow the residence of XML format data in relational tables. The RDB with tables including XML data is called a hybrid XML-Relational database (hybrid DB).

Currently, there are several keyword search methods over pure XML DB or traditional RDB such as DBXplorer [1], XRANK [2]. This paper firstly clarifies that these existing known methods cannot get sufficient answers. Next, we propose a new method which uses hybrid XML-Relational databases for getting enough answers.

1.2 Our goal

Our goal is to obtain more reasonable answers of keyword search over hybrid DB than XML DB or RDB does. To achieve it, we make good use of hierarchy of XML in hybrid tables (which contain XML format data). The following example illustrates a case of new challenge.

Example: Consider the hybrid database in Figure 1, which belongs to a search engine for digital library. The database includes two hybrid tables (**Conference**, **Authors**) and one relational table (**Paper**). We refer to all data formats except XML as *relational data*. In the hybrid table **Conference**, information of conference is listed as XML format, whose hierarchy is "Conf-Session-Paper". The relationship of Conference and Session is one to many, like several branches from a root of tree, and the relationship of Session and Paper is also one to many. In another hybrid table **Authors**, author information is listed as XML format of hierarchy "Author-Paper", which means the root of this XML is Author. The relationship of Authors and Paper is many to many, that is stored in a relationship table between Authors and Paper. The table **Paper** contains the detail of each paper such as id, title and so on.

Suppose a user wants to know about some relevance between an author named "Sanjay" and a subject of study about "link". We use the notation *Query* = {Sanjay, link} to express the set of query keywords "Sanjay" and "link". The user can issue *Query* = {Sanjay, link} to obtain a list of answers. As usual, if a writer named "sanjay" has written a paper or a book about "link", it is easy to get this answer. In our study, another new case of answers should be also obtained. As shown in Figure 1, one of such answers should include two tuples (Conference: V04) and (Authors: A002) containing the query keywords.

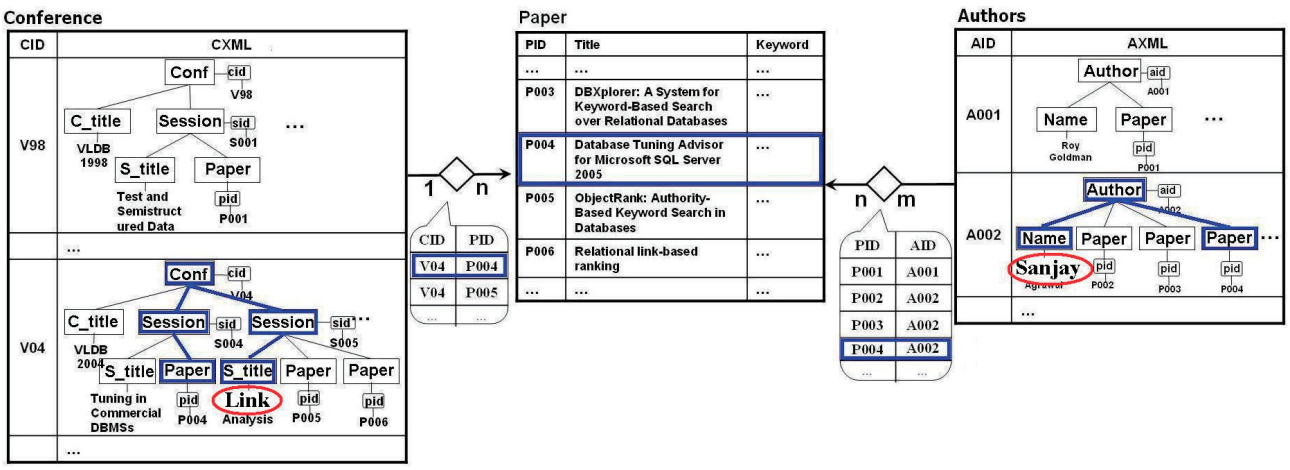


Figure 1 An example of keyword search over a hybrid DB

This answer of Figure 1 is depicted in blue lines, and it means that an author named "Sanjay" has written a paper, which has been published at a conference titled "VLDB 2004", and the conference has a session named "link analysis". Based on the relationship of the two tuples described above, our study chooses this answer by two kinds of join operation; one is to join XML data with relational data, and another is a foreign-key join between relational data.

This paper focuses on how to search for all the above appropriately-related information on hybrid XML-Relational databases. The rest of this paper is organized as follows. In Section 3, we make a keyword search comparison between XML DB and RDB. In Section 4, we present our new solution of keyword search over hybrid XML-Relational databases, and describe the processing component responsible for joining XML data with relational data (called *XRjoin* by us). And conclusions are given in Section 5.

2. RELATED WORKS

Many research efforts have studied the problem of keyword search over traditional RDB. Examples include [3, 5, 6]. They are based on the basic approach of DBXplorer [1], a fundamental solution for RDB. Keyword search over XML data has also attracted attention (e.g. [8]) and one of the most outstanding studies is XRANK [2]. We analyze merits and demerits of the techniques of DBXplorer [1] and XRANK [2] in Section 3.

Another approach in the literature is to process a keyword query on a weighted graph [4]. According to the database schema, tuple instances are mapped to a graph model. It finds connections of nodes where keywords occur on the graph for answer. In contrast, our approach is to find a sub-tree (called *join tree*) in the schema graph that contains tables as nodes. Then we construct join statements (XRjoin and foreign-key join) to select tuple sets that include all keywords (see Section 4).

3. ANALYSIS OF EXISTING KEYWORD SEARCH METHODS

3.1 XML DB

The eXtensible Markup Language (XML) is a general-purpose specification for creating custom markup languages, which allows users to define their own elements. In this paper, we express the contents of the table Conference in Figure 1 as an XML document of Figure 2.

The biggest element is "Conferences" which includes a lot of child-elements "Conference". Each "Conference" contains one "C_title" and many child elements "Session", and each "Session" contains one "S_title" and many child-elements "Paper".

```
<?xml version="1.0">
<Conferences>
  <Conference  cid="V04">
    <C_title>VLDB2004</C_title>
    <Session  sid="S004">
      <S_title> … </S_title>
      <Paper pid ="P004">
        <title>…</title>
        <keywords>…</keywords>
      </Paper>…
    </Session>…
  </Conference>…
</Conferences>
```

Figure 2 The XML document of Conference

Based on the same contents, Figure 3 indicates the tree structure of Figure 2. In the tree structure, the root node is "Conferences" still. The hierarchy of the tree structure lets the relationship between elements be easily understood.

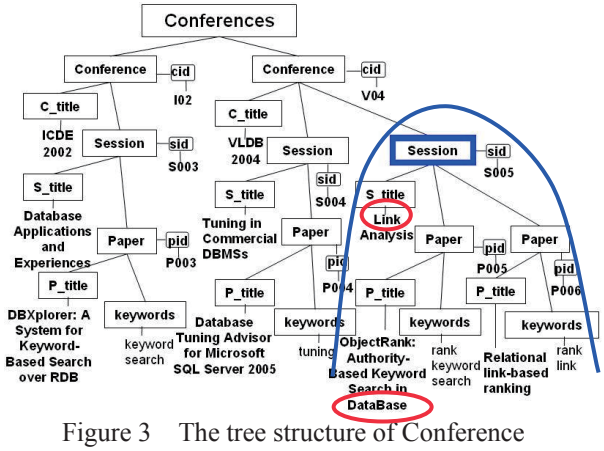


Figure 3 The tree structure of Conference

Keyword search over XML DB returns sub-trees of an XML tree structure, each of which includes all query keywords. As an illustration, consider *Query* = {link, database} issued on the XML in Figure 3. One of the results is the sub-tree in the blue line in the figure, whose root is the "Session" element in the blue box because it contains the two query-keywords in red circles of Figure 3.

XRANK [2] can find such answer sub-trees as described above. The tree structure makes search easy, when the relation of nodes containing query keywords can be presented as edges in answer sub-trees. We can easily discover the relevance of user-given keywords, based on the hierarchy of answer sub-tree.

However, XRANK [2] can only get an answer formed by a sub-tree. Thus, if there are relations of contents containing query keywords that cannot be connected on a sub-tree, XRANK [2] cannot get such information as an answer.

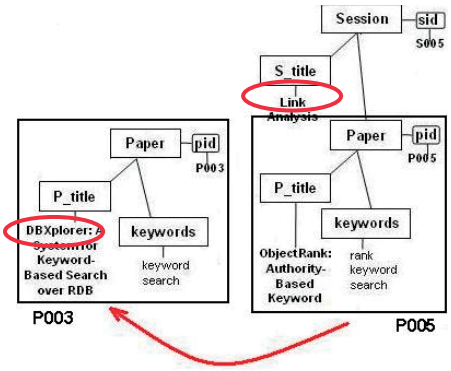


Figure 4 An unavailable answer of XML DB

For instance, suppose that *Query* = {DBXplorer, link}. Then it is difficult to obtain the answer (two sub-trees) shown in Figure 4, because the two sub-trees are connected by a citation link. The citation link is presented by a red arrow in Figure 4. Existing methods of XML DB cannot find this type of answers.

3.2 RDB

On the other hand, RDB places information into tables, based on a relational schema modeling method, Entity-Relationship (ER) model. Storing the same contents of Figure 1, we suggest the ER model for RDB in Figure 5.

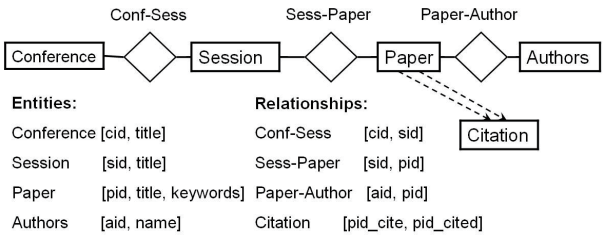


Figure 5 Entity-Relationship model

The ER model consists of four entities (**Conference**, **Session**, **Paper**, **Authors**) and three relationships (**Conf-Sess**, **Sess-Paper**, **Paper-Author**). Because of database normalization, the hierarchy "Conference-Session-Paper" has been divided into five parts (**Conference**, **Session**, **Paper**, **Conf-Sess**, **Sess-Paper**).

The approach of DBXplorer [2] is to look up a preprocessing index table to identify the tables where keywords occur, firstly. Then, all potential subsets of tables in the RDB are enumerated, that might contain all keywords. For each subset of tables, there is a sub-tree in the schema graph that contains these tables as nodes. These sub-trees are referred to as *join trees*. Along a join-tree, the subsets of tables can be joined.

Finally, for each enumerated join tree, a SQL statement is generated and executed. The results are selected and presented to the user.

As shown in Figure 6, given a *Query* = {commercial, SQL}, we can find a result of tuple sets including **Paper**: P004, **Session**: S004 and **Sess-Paper**: S004. The tuples (**Paper**: P004, **Session**: S004) have keywords "commercial" and "SQL" respectively, and are selected because of the tuple of relationship **Sess-Paper**: S004.

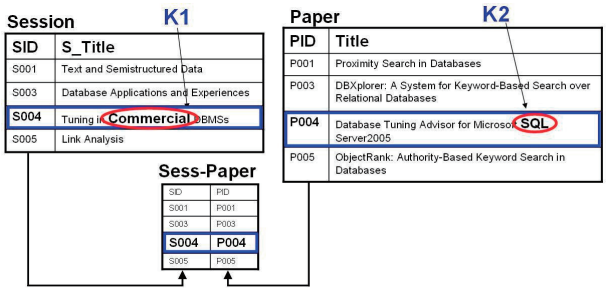


Figure 6 An example of keyword search over RDB

This method is effective when keywords occur in different entities which are connected with each other by relationships or when keywords exist in one tuple of the same entity.

However, we cannot obtain an answer, when keywords exist in different tuples of the same entity. Figure 7 shows this situation.

Figure 7 is an answer tuple set which cannot be obtained by the approach mentioned above. *Query* = {integrate, SQL} consists of K1 "integrate" in **Paper**: P007 and K2 "SQL" in **Paper**: P004. Since P004 and P007 are in the same entity, the relationship **Sess-Paper** and the entity **Session** will not be used to do foreign-key join between P004 and P007**.** Thus the answer will be lost shown in the blue dot lines, although it means that P004 and P007 containing "integrate, SQL" have been published in the same session S004.

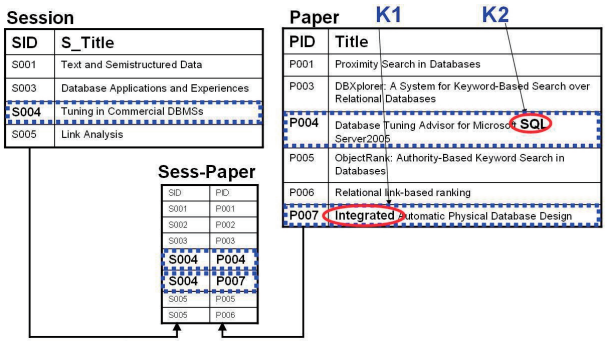


Figure 7 An unavailable answer of RDB

Note that in case of XML DB, the above sub-tree can be obtained, which is rooted by Session S004 containing the Papers P004 and P007.

Unlike XML DB, RDB finds answers of keyword search by using relationships to join and select tuple sets. The above answer tuple sets ought to be found, because contents of XML have been exploited to be stored in appropriate tables. However, DBXplorer [2] does not consider such important hierarchical information when building a *join tree*. So RDB cannot get appropriate answers in the case of Figure 7.

Concerning keyword search over XML data and relational data, we utilize the tables containing XML data and propose a new method to retrieve information effectively.

4. USING HYBRID XML-RELATIONAL DATABASES

4.1 Approach of keyword search

Since XML data can reside in tables of RDB, we propose a new function to deal with keyword-based search for hybrid DB systems. Based on some techniques of DBXplorer [1], we offer a solution for hybrid DBs as follow.

Firstly, we design the schema for hybrid DBs from the ER model of traditional RDBs. According to the feature of XML data, in the example of this paper, we utilize "XML1" as an XML format which contains entities (**Conference**, **Session**) and relationships (**Conf-Sess**, **Sess-paper**) in ER of RDB (see Figure 8), and use "XML2" which contains (**Authors**, **Paper-Author**). The details of papers are stored in the relational entity **Paper** still. Notice that there are only the ids of papers to be presented in XMLs.

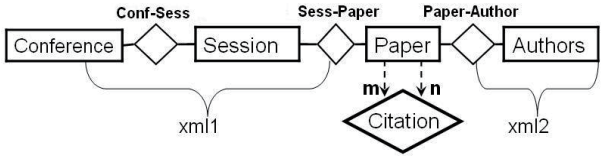


Figure 8 The data corresponding to XMLs in RDB

The ER model for hybrid DB has been made as shown in Figure 9. The new hybrid tables are "**Conference**" and "**Authors**" which contain XMLs.

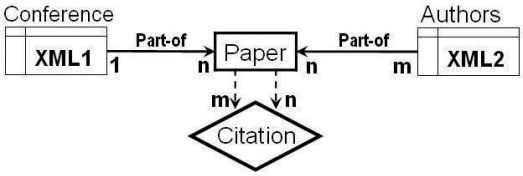


Figure 9 The ER model for hybrid DB

Secondly, to enable keyword search in hybrid DB, we make use of an auxiliary table that identifies entities containing query keywords. If a possible keyword exists in a node (v) of an XML, we store the table name, the attribute name and the ids of all ancestor nodes of the node (v). For example, as shown in Figure 11(a), the information to be stored for a keyword "tuning" is the table name **Conference**, the attribute name **XML1**, and the ancestor node ids **S004** and **V04**. Additionally, if a tuple-id of relational data exists in an XML, we store the table name, the attribute name, the tuple id and all ancestor- node ids of the tuple id. This auxiliary table is looked up to identify the tables of the database that contain the query keywords.

Then, when keywords are given, all potential subsets of tables in the database are identified and enumerated. These subsets can be joined only if they are connected in the schema as join trees. Based on the example data of this paper, Figure 10 shows an instance of enumerated join trees by the way of DBXplorer [1]. In Figure 10, each keyword of Query = {K1, K2} exists in the schema twice. So we enumerate four types of join trees.

Next, if join trees include hybrid entity containing XML data, it is necessary to solve the problem of joining XML data with relational data. We define a new operator XRjoin (see subsection 4.2) to construct a new hybrid table which contains appropriate sub-trees of XMLs and essential relational data. The result table of XRjoin is a hybrid entity, and thus it is used for succeeding join operations in a join-tree.

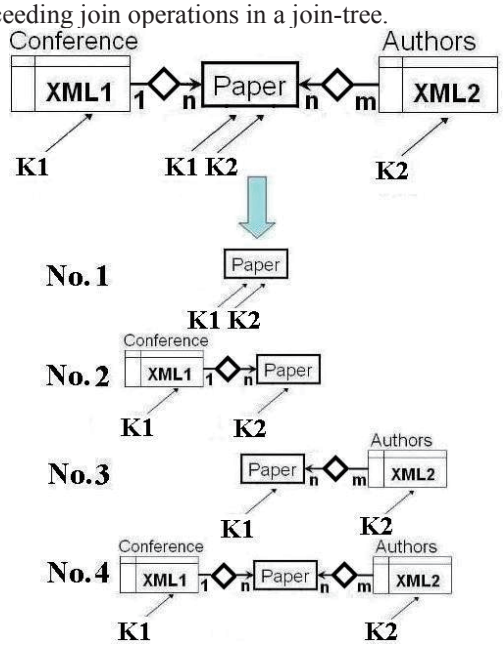


Figure 10 An example of enumerated join trees

Finally, for each enumerated join tree, a SQL/XML statement is constructed and is executed. It joins the tables in the tree and selects those rows containing all query keywords. The final rows are provided for the user.

4.2 XRjoin

We produce XRjoin to select tuple sets from XML data and relational data. We describe XRjoin in detail in this subsection.

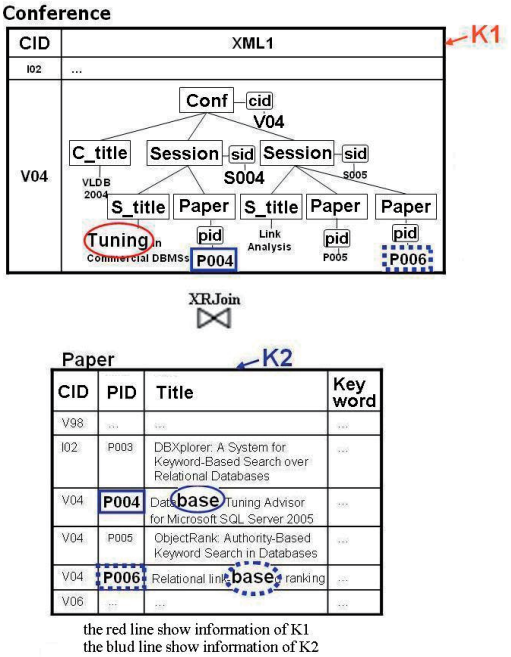
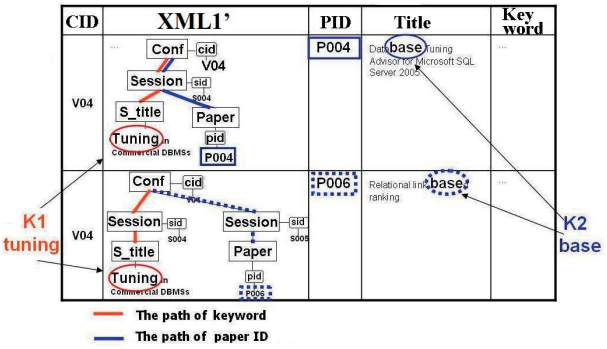


Figure 11 (a) Before XRjoin



(b) After XRjoin

Figure 11 An instance of XRjoin

Firstly, we present an illustration of concrete contents before and after XRjoin in order to show our idea clearly. The instance of Figure 11(a) comes from No.2 join tree of Figure 10.

Consider the simplified XML tree of Conference VLDB 2004 (whose id is **CID: V04**) in Figure 11(a). The figure shows one keyword K1 "tuning" in a red circle and another keyword K2 "base" in a blue circle, which are stored in the hybrid entity **Conference** and the relational entity **Paper**, respectively. In contrast, the result of XRjoin in Figure 11(b) is one hybrid entity; this table has two tuples (rows) as answers, each of which contains K1 and K2.

In each tuple of Figure 11(b), the attribute **XML1'** has a reconstructed tree. This tree is a part of the XML tree in hybrid entity **Conference** of Figure 11(a). The tree of **XML1'** has two parts. One is the path of keyword K1 "tuning" shown in a red line and the other is the path of "PID" shown in a blue line. This "PID" is the tuple ID of relational data containing K2 "base". We extract these paths from the original **XML1** of Figure 11(a), and reconstruct the tree **XML1'** in Figure 11(b). In this way, we can select the answer tuple sets of Figure 11(b) containing all query keywords.

Next, we describe the general model of XRjoin.

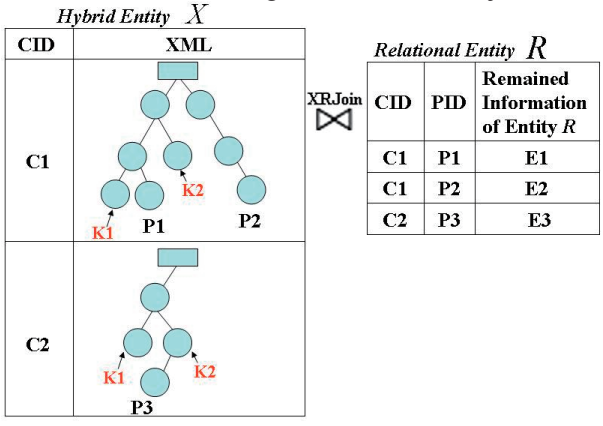


Figure 12 Operand Entities of XRjoin

Figure 12 is a pair of operand entities of XRjoin, a hybrid entity and a relational entity. **PID** is a tuple id in the relational entity "**R**" and it is also a node of XML tree in the hybrid entity "**X**". In the relational entity "**R**", we use an attribute of **CID** to present the one-to-many relationship between **CID** and **PID**. In Figure 12 and Figure 13Figure 12, the circle symbol means a node (XML element), and the box symbol means the root node.

In Figure 12, assume we have known two keywords K1 and K2 hit the hybrid entity "**X**", and this XRjoin is referred to as *XRjoin(X, R)*.

Figure 13 is a result of the XRjoin. Generally, the result has its attribute "***x***" having a sub-tree, which has paths from nodes of keywords and nodes of joined tuple-ids to their nearest common ancestor. To put it concretely, in Figure 13, the attribute "***x***" is constructed by two paths (shown in orange and red) from the two keyword nodes satisfying K1 or K2 and the blue path from one ID (which corresponds to id of one tuple in relational entity "**R**"). We also add the path from the nearest common ancestor to the root node of the original XML in "***x***". The tuple sets of Figure 13 provide all relevant information of both the hybrid entity "**X**" and the relational entity "**R**", when the keywords K1 and K2 exist in the XML of the hybrid entity "**X**".

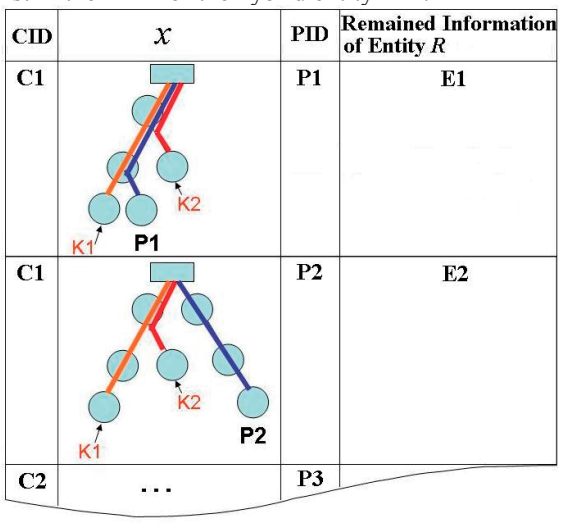


Figure 13 Result Entity of XRjoin

In this way, an XRjoin operator puts the appropriate tuple sets into a new hybrid entity, which contain reconstructed sub-trees of the XML.

As an implementation of XRjoin, we first look up query keywords in the auxiliary table, and then we extract the nearest common ancestors by comparing ancestor ids stored in the auxiliary table by SQL/XML queries. Last, necessary sub-trees are constructed and appropriate tuple sets are selected.

Finally, we show XRjoin plays an important role to execute a join tree.

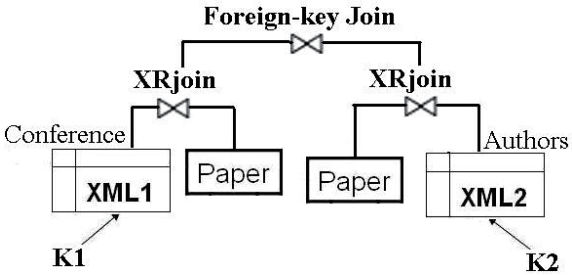


Figure 14 join steps by No.4 join tree of Figure 10

Figure 14 is the join tree of No.4 of Figure 10, written by XRjoins and a foreign-key join. In this case we must consider that the foreign-key join must wait for XRjoin's turn. Figure 14 shows that we firstly do XRjoin twice to refine tuple sets containing the query keywords and then execute the foreign-key join using a SQL statement to select the answer tuple sets.

In practice, Figure 14 presents the join tree for getting the answer of Figure 1 by our method.

5. CONCLUSION

In this paper, we firstly described that the existing keyword-search methods of XML DB or RDB cannot get sufficient results. Then we proposed a new keyword-search method of hybrid XML-Relational databases to retrieve information on both XML data and relational data. As a major idea, we proposed a new operator XRjoin, which joins XML data with relational data. By using XRjoins, we showed that a modified join-tree can get new better answers that contain both relationship information and hierarchy information in a hybrid XML-Relational database.

Our current implement is done on IBM DB2 v9 [9]. We are currently improving implementation of XRjoin operators. An algorithm of generating appropriate join-trees is also under our improvement.

REFERENCES

[1] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, "DBXplorer: A System for Keyword-Based Search over Rela-tional Databases", Proceedings of the 18th International Conference on Data Engineering, pp.05-17, 2002.

[2] Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram, "XRANK: Ranked Keyword Search over XML Documents", Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.16-27, 2003.

[3] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan, Luis Gravano, "Efficient Keyword Search Aross Heterogeneous Relational Databases", Proceedings of the 23rd International Conference on Data Engineering, pp.346-355, 2007.

[4] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, Xuemin Lin, "Finding Top-k Min-Cost Connected Trees in Databases", Proceedings of the 23rd International Conference on Data Engineering, pp.836-845, 2007.

[5] Andrey Balmin, Vagelis Hristidis, Yannis Papakon stantinou, "ObjectRank: Authority-Based Keyword Search in Databases", Proceedings of the Thirtieth International Conference on Very Large Data Bases, pp.564-575, 2004.

[6] G.Bhalotia, A.Hulgeri, S.Chakrabarti, S.Sudarshan, "Keyword searching and browsing in databases using BANKS", Proceedings of the 18th International Conference on Data Engineering, pp.431-440, 2002.

[7] Mirella M.Moro, Lipyeow Lim, Yuan-Chi Chang, "Schema Advisor for Hybrid Relational-XML DBMS", Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.959-970, 2007.

[8] S.Amer-Yahia, E.Curtmola, A.Deutsch, "Flexible and efficient XML search with complex full-text predicates", Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 575-586, 2006.

[9] IBM DB2 Database Information Center, "http://publib.boulder.ibm.com/infocenter/db2luw/v9/"