Keyword Search on Both XML and Relational Data

Liru Zhang, Tadashi Ohmori and Mamoru Hoshi

Graduate School of Information Systems, The University of Electro-Communications, Tokyo 1828585, Japan

Received: January 10, 2011 / Accepted: January 31, 2011 / Published: April 30, 2011.

Abstract: Keyword search is a familiar and effective method to retrieve information under databases for any user. Keyword search on structured/semi-structured databases has received significant attention in recent years. A number of solutions have been proposed and many prototypes have been developed. Building on growing user needs, recently several RDBMS (IBM DB2, etc.) have made themselves be compatible with relational and XML storages. However, existing keyword-search methods on RDB or XML DB cannot get appropriate answers when some substructures of XMLs are related by relational linkage information. In this paper, the authors begin by analyzing typical existing keyword-search methods on RDB and XML DB. Next, we design the data model of a relational database storing both XML and relational data. A relational database storing both XML and relational data is termed a hybrid XML-Relational Database (XML-RDB) in this paper. Next, we introduce our approach to realize keyword search on XML-RDB. Then, we propose a new join operator, named XRjoin, to join XML data with relational data under an XML-RDB. Finally, we present our experiments on DB2 V9.5 to demonstrate the effectiveness and feasibility of the proposed approach. Experimental results show that the hybrid XML-RDB system can get more reasonable answers than existing approaches.

Key words: Keyword search, hybrid database, XML-RDB, relational XMLDB, information integration.

1. Introduction

Keyword search is a popular function of databases to enable users to extract information under databases without any knowledge of the schema or query languages. Recently, keyword search on structured databases (RDB) or semi-structured databases (XML DB) has received significant attention. There are several studies on keyword search over databases such as DBXplorer [1], DISCOVER [2] on RDB, XRANK [3] on pure XML DB. These studies work on either XML or normalized relational data, and they cannot get appropriate answers when some substructures of XMLs are related by relational linkage information.

On the other hand, with the amount of XML data in relational databases growing rapidly, the need for user to search the information of both XML and relational data is dramatically increasing. Consequently, several major commercial RDBMS (e.g., IBM DB2 9.5[4], Microsoft SQL Server 2008[5]) have supported XML storage to store XML data without any change of their formats. Such relational databases storing both XML and relational data are termed hybrid XML-Relational databases (XML-RDB).

In this paper, the authors firstly analyze three previous main methods and clarify drawbacks of these existing studies. Then, under the background of XML-RDB, to effectively retrieve substructures of XMLs related by relational linkages, this paper proposes a new keyword-search method on XML-RDB. Our new keyword-search method not only can get the new information, but also can get the results obtained by DBXplorer [1] or XRANK [3]. Next, a new join operator, XRjoin is proposed to realize our proposed keyword-search method on XML-RDB. XRjoin uses...
the SQL/XML query language to do join between XML and relational data. This paper extends the normal type of XRjoin based on our previous study [6], into other three types to work in general cases. Our proposed system utilizes XRjoins on XML-RDB to do keyword search effectively. Our experiments show that our system can get more reasonable answers than existing approaches.

The rest of this paper is organized as follows. In section 2, we discuss the existing methods of keyword search on RDB and XML DB and describe their disadvantages and problems. To overcome the problems, in section 3, we build a hybrid XML-RDB system and state our approach of keyword search on this system. In section 4, we propose the new join operator XRjoin. In section 5, experiments show that our proposed system is effective and feasible. The conclusions are given in section 6.

2. Existing Techniques of Keyword Search

2.1 Keyword Search on XML Databases

Keyword search on XML DB returns minimal substructures (sub-trees) of XML which include all query keywords. XRANK [3], one of major techniques of keyword search on XML DB, employs the tree structure of XML to extract the LCA (Lowest Common Ancestor) for the elements that keywords hit, in such a way that every keyword appears at least once. It also generates a ranking score for each answer and can return deeply nested XML elements, not entire XML.

Fig. 1 shows an example of Query = {link, database}, when the contents of conferences are expressed as an XML tree structure. The result of Query = {link, database} is the sub-tree rooted by the session element SID:S005, which includes all query keywords (in red circles).

However, if the linkages between elements are required to present important relationships, the resulting sub-trees related by those linkages cannot be explored. XRANK describes linkages using XLink or XPointer that are only utilized to calculate ranking scores. It cannot obtain the answer shown in Fig. 2. As the answer of Query = {system, link}, Fig. 2 means that, in the sub-tree SID:S005 having "link", the paper PID:P005 cites the sub-tree PID:P003 with "system" (by the red dotted arrow). But XRANK gets nothing other than the whole conference.

Other related researches [7-9] also cannot construct the sub-trees of XML by linkage information.

2.2 Keyword Search on Relational Databases

On the other hand, RDB normalizes all information into relations, represented by the Entity-Relationship (ER) model. With the data set of DBLP.xml [10] such as Fig. 1, the ER model for RDB is described in Fig. 3. The ER model in Fig. 3 consists of four entities (Conference, Session, Paper, Authors) and three relationships (Conf-Sess, Sess-Paper, Paper-Author). Because of database normalization, the XML hierarchy
Keyword search on RDB, where the answers are multiple tuples connected via joins, has been realized by two main ways. One is a simple way represented by DBXplorer [1]. The other is a complete method DISCOVER [2].

2.2.1 DBXplorer [1]

DBXplorer firstly identifies the entities (nodes) which include keywords. Then, sub-trees including those nodes are enumerated on the schema graph, under the constraint that each sub-tree must contain all keywords and its leaf-nodes must contain keywords. DBXplorer calls the sub-trees join-trees. Actually, DBXplorer uses Group-Steiner trees (GST) as join-trees [1]. According to join-trees, SQL statements are generated and executed to get resulting tuples.

For example, Fig. 4 shows the join-tree when Query = {commercial, sql} in Fig. 3. According to the join-tree of Fig. 4, the following SQL statement is created:

```
SELECT *
FROM Paper as P, Session as S, Sess-Paper as SP
WHERE contain(P.title, "sql")
AND contain(S.title, "commercial")
AND S.sid = SP.sid AND P.pid = SP.pid
```

Fig. 5 shows the resulting tuples of Query = {commercial, sql} in blue boxes that are gotten by joining two entities Session and Paper via the relationship Sess-Paper. This result means that the paper PID:P004 of keyword “sql” belongs to the session SID:S004 of keyword “commercial”.

However, appropriate answers cannot always be obtained. If keywords exist in the same entity but in different tuples, DBXplorer cannot get any results.

Fig. 6 shows the result of Query = {integrate, sql}. The keyword “integrate” hits the tuple of PID:P007 in Paper, and the keyword "sql" hits another tuple of PID:P004 in the same table. The tuples shown in the dotted line of Fig. 6 means that the two papers PID:P007 and PID:P004 belong to the same session SID:S004 by relationship Sess-Paper. Such information cannot be extracted by DBXplorer, but can be searched by XRANK easily.

2.2.2 DISCOVER [2]

To get complete results, DISCOVER enumerates all Candidate Networks (CNs) on an extended schema graph. The extended schema graph is derived from the schema graph of the database by dividing entities including one or more keywords into several tuple sets. The CNs are turned into SQL statements for joins in the same way as join-trees.
For example, Fig. 7 shows the extended schema graph when Query = \{integrate, sql\} according to [2]. In Fig. 7, Paper is divided into three tables (tuple sets): i.e., P without any keyword, P1 including "integrate" only, P2 including "sql" only.

In Fig. 7, the arrow-head curve presents a CN, which means to do four joins of P1 Sess-Paper Sess-Paper P2. So DISCOVER can get the result such as Fig. 6.

However, to return answers to users in a reasonable time, DISCOVER uses a parameter \(T_{max}\) (the maximum number of joins in a CN) to limit the number of joins to do the CN generation in a certain searching range. It makes DISCOVER not complete. According to [11], when \(T_{max}\) increases, DISCOVER's time grows exponentially. In fact, DISCOVER is not practical when \(T_{max}\) is greater than 5.

3. The Hybrid XML-RDB System

We design a new data model for an XML-RDB. Using this data model, our system aim at generating join-trees more effectively than DISCOVER and getting relationally-related sub-trees of XMLs, which are not obtained by DBXplorer.

3.1 Data Model

Based on the ER model of Fig. 3, we design the data model for an XML-RDB. Fig. 8 shows the data model where the corresponding data of DBLP.xml are stored. This schema includes three entities Conference, Paper, Authors, and two part-of\(^1\) relationships between Conference.XML1 and PID, Authors.XML2 and PID.

In Fig. 8, the hybrid entity is an entity storing one XML column (denoted by \[\text{XML}^1\]), the relational entity is an entity without XML (denoted by \[\text{XML}^2\]). Conference and Authors are hybrid entities storing XML1 and XML2 respectively. Paper is a relational entity without XML. The structure of XML1 is presented in Fig. 8, which has the “Conf-Session-Paper” hierarchy. XML1 makes four tables (Conference, Conf-Sess, Session, Sess-Paper) of RDB into one hybrid entity. XML2 contains the information between one author and one or more papers in one XML. “Author-Paper” hierarchy in Fig. 8 shows the structure of XML2. An instance of Fig. 8 is shown in Fig. 9.

3.2 Our Approach of Keyword Search

In this section, we outline our approach of keyword search. The processes of our approach are composed of the following three steps: (1) identify all entities including one or more keywords, (2) generate join-trees, (3) generate statements (i.e. SQL/XML statements) and execute them.

In this paper, the authors only discuss the join-tree generation on an acyclic schema graph without any self-loop relationship.

3.2.1 Keyword Entities

Keyword entities are the entities including one or more keywords. Keyword entities are identified first for all keywords by using auxiliary tables.

In this XML-RDB, three auxiliary tables are made as follows. Symtbl_relation (value, r, cn, rn) stores, for each tuple t in a relational entity rn, the information of t’s the tuple id r and t’s value of the column cn. Secondly, Symtbl_XML (value, h, DeweyID, hn) stores, for each element e of the XML of a hybrid tuple t in a hybrid entity hn, the information of e’s value, t’s tuple

\(^1\) Part-of is a relationship where one tuple id “belongs” to (is a part or member of) another object (element of XML).
id $h$, e’s corresponding DeweyID\(^2\) in the XML. 

Symtbl\_link $(r, h, \text{DeweyID})$ contains the relational tuple id $r$, corresponding hybrid tuple id $h$ and the DeweyID of $r$ in $h$. We use the full-text search supported by DB2 Net Search Extender (NSE) on the attribute $\text{value}$ of $\text{Symtbl\_relation}$ and $\text{value}$ of $\text{Symtbl\_XML}$ to identify which data including keywords.

### 3.2.2 Join-trees

Given keyword entities, the authors enumerate the join-trees in the schema graph. On our hybrid XML-RDB system, we generate two kinds of join-trees, primitive join-trees and extensive join-trees.

A primitive join-tree is defined as a tree which must contain all keywords and whose leaf-nodes must be keyword entities.

An extensive join-tree is defined as a tree whose leaf-nodes must be keyword entities or hybrid entities and which must contain all keywords. Namely, an extensive join-tree allows the leaf-node to be a hybrid entity without keywords.

The authors rank those join-trees according to the number of joins they involve. Users can select any join-tree to do next step to get detailed information. The details of join-tree generation will be given at Section 3.3.

### 3.2.3 Statements and Execution

Our system assumes that one join-tree is selected by a user. According to the join-tree, the join between XML and relational data will be done firstly. Then, the natural join will be done among the remaining tables and the temporary results. If final results exist, keyword-related tuple sets will be returned to the user interface.

### 3.3 Join-Tree Generation

In this section, we state the generation of join-trees on an acyclic XML-RDB schema.

For simplicity, this paper provides the join-tree generation without considering self-loop relationship such as Citation in Fig. 8.

Based on the algorithm of DBXplorer, we generate the GSTs on the acyclic schema graph as primitive join-trees. However, primitive join-trees do not include the information that is substructures of XMLs when multiple keywords hit a single relational entity such as Fig. 6.

The authors extend a primitive join-tree $T_o$ if:

1. $T_o$ has a relational entity $R$ where $R$ contains multiple keywords, and
2. $R$ does not have a “part-of” edge from any hybrid entity $H$ existing in $T_o$, and
3. in the schema graph, $R$ is directly linked by a “part-of” edge from a hybrid entity $X$ (where $X$ is not in $T_o$).

---

\(^2\) DeweyID is the ID created by an effective labeling method for XML, the Dewey Order [7].
If all of those conditions hold, by attaching the edge and $X^3$ to $R$ in $T_m$, we get a new extensive join-tree. The extensive join-trees can pick up the results missed by DBXplorer.

Consider an example of $Query = \{K1, K2, K3\}$ on the schema graph of the XML-RDB shown in Fig. 10.

According to our join-tree generation algorithm, Fig. 11 shows four primitive join-trees of $Query = \{K1, K2, K3\}$, which are Join-tree1, Join-tree2, Join-tree3 and Join-tree4. Fig. 12 shows two extensive join-trees (Join-tree5 and Join-tree6) from one primitive join-tree Join-tree1, because Join-tree1 satisfies the extension condition.

The semantics of these join-trees are understood as follows, using the real instances from DBLP.

Join-tree1 means to get tuples in the relational entity $Paper$, when the tuples include $K1$, $K2$ and $K3$. If $K1$, $K2$ and $K3$ exist in different tuples of $Paper$, the system extends it to Join-tree5 to get the XML sub-trees (substructures) in $Conference$ to present the relationships among different tuples of $K1$, $K2$ and $K3$. If the results of Join-tree5 are not null, they mean that the papers of $K1$, $K2$ and $K3$ belong to the same session or the same conference. Similarly, Join-tree6 gets the XML sub-trees in $Authors$ if the papers of $K1$, $K2$ and $K3$ are written by the same author.

Join-tree2 means to extract the hybrid information that contains the $K1$-related sub-trees in $Conference$ and the tuples of $K2$ and $K3$ in $Paper$. These XML sub-trees directly include $K1$ but do not include $K2$ and $K3$ directly. They contain the elements of tuple ids ($ID_{K2}$, $ID_{K3}$) which link the tuples in $Paper$ including $K2$ and $K3$.

Join-tree3 means to extract the hybrid information that contains the $K3$-related sub-trees in $Authors$ and the tuples of $K1$ and $K2$ in $Paper$. These XML sub-trees include $K3$ but do not include $K1$ and $K2$ directly. They contain the elements of tuple ids ($ID_{K1}$, $ID_{K2}$).

Join-tree4 means to extract the hybrid information that contains the $K1$-related sub-trees in $Conference$ and the $K3$-related sub-trees in $Authors$, where both the $K2$-related paper and a common paper between $Conference$ and $Authors$ must belong to the same sub-tree of $Conference.XML1$ or that of $Authors.XML2$.

How to interpret Join-tree4 is not trivial. We decide that Join-tree4 means to extract the hybrid information that contains the $K1$-related sub-trees in $Conference$ and the $K3$-related sub-trees in $Authors$, where both the $K2$-related paper and a common paper between $Conference$ and $Authors$ must belong to the same sub-tree of $Conference.XML1$ or that of $Authors.XML2$.

Fig. 13 (a) is one interpretation of Join-trees 4. It means that both a paper written by a $K3$-related author and a $K2$-related paper belong to the same $K1$-related conference. There are two kinds of sub-trees in Fig. 13 (a). One is the sub-tree of $Conference.XML1$ that contains a $K2$-related paper element, a common paper

---

3 If there are multiple $X$s on the schema graph linked by $R$, for each $X$, a new extensive join-tree will be generated.
Keyword Search on Both XML and Relational Data

Fig. 13  The interpretations of Join-tree4.

The interpretations of Join-tree4 is the union of set of the Fig. 13 (a)’s result and the Fig. 13 (b)’s result.

The remaining task we must do is to prepare new XRjoin operators to satisfy the demands of these join-trees.

4. XRjoin

4.1 XRjoin Definitions

Here, we provide a new join operator XRjoin for presenting the semantics of join-trees. We design four types of XRjoin and show their notations.

In Fig. 14 and Fig. 17, $X$ presents a hybrid entity, and $K_1$. The other is the sub-tree of Authors.XML2 that contains the common paper element and $K_1$. Fig. 13 (b) is the other interpretation of Join-trees4. It means that both a paper belonging to the $K_1$-related conference and a $K_2$-related paper are written by the same $K_3$-related author.

The final result of Join-trees4 is the union of the Fig. 13 (a)’s result and the Fig. 13 (b)’s result.

The XRjoin has keywords in both $X$ and $R$. Fig. 14 shows the normal type of XRjoin.

Each XML in the resulting table of Fig. 16 has a red path from the element of “tuning” in Conference and a blue path from the paper element whose relational tuple in Paper includes keyword “SQL”. As an important part of results, relevant relational data (PID, Title) in Paper are attached. The first hybrid tuple in Fig. 16 means that a paper of “SQL” belongs to a session Type 1.

XRjoin ($(\textit{Conference}, \{\textit{tuning}\}), (\textit{Paper}, \{\textit{SQL}\}))$

(Fig. 16) is to extract relevant sub-trees from “tuning”-related XMLs in Conference and the elements of “SQL”-related tuple ids in Paper. It extracts the LCA between the element “S\_title” of SID:S004 and one of the paper elements PID:P004 and PID:P006. The sub-tree rooted by the LCA becomes a new XML in CXML of Fig. 16.

Then, the following XRjoin:

XRjoin ($(\textit{Conference}, \{\textit{tuning}\}), (\textit{Paper}, \{\textit{SQL}\}))$

(Fig. 16) is to extract relevant sub-trees from “tuning”-related XMLs in Conference and the elements of “SQL”-related tuple ids in Paper. It extracts the LCA between the element “S\_title” of SID:S004 and one of the paper elements PID:P004 and PID:P006. The sub-tree rooted by the LCA becomes a new XML in CXML of Fig. 16.

Each XML in the resulting table of Fig. 16 has a red path from the element of “tuning” in Conference and a blue path from the paper element whose relational tuple in Paper includes keyword “SQL”. As an important part of results, relevant relational data (PID, Title) in Paper are attached. The first hybrid tuple in Fig. 16 means that a paper of “SQL” belongs to a session Type 1.

XRjoin ({$X$, {$K_{11}, K_{12}, \ldots$}), ($R$, {$K_{21}, K_{22}, \ldots$}))

(1)

The expression (1) means to combine the LCA’s sub-trees of XMLs in $X$ including {K11, K12, ..., ID21, ID22, ...} with tuple sets in $R$ satisfying {K21, K22, ...}. Fig. 14 shows that XRjoin of Type 1 has restrictions on both $X$ and $R$. Join-tree2 and Join-tree3 in Fig. 11 use XRjoins of Type 1.

Fig. 15 shows an example of several tuples of Paper, and an XML in Conference. The keyword “tuning” exists in $S\_title$ of the session SID:S004 in XML1. The keyword “SQL” exists in $Title$ of two tuples PID:P004 and PID:P006 in Paper.

Then, the following XRjoin:

XRjoin ($(\textit{Conference}, \{\textit{tuning}\}), (\textit{Paper}, \{\textit{SQL}\}))$

(Fig. 16) is to extract relevant sub-trees from “tuning”-related XMLs in Conference and the elements of “SQL”-related tuple ids in Paper. It extracts the LCA between the element “S\_title” of SID:S004 and one of the paper elements PID:P004 and PID:P006. The sub-tree rooted by the LCA becomes a new XML in CXML of Fig. 16.

4. XRjoin

4.1 XRjoin Definitions

Here, we provide a new join operator XRjoin for presenting the semantics of join-trees. We design four types of XRjoin and show their notations.

In Fig. 14 and Fig. 17, $X$ presents a hybrid entity, and $K_1$. The other is the sub-tree of Authors.XML2 that contains the common paper element and $K_1$. Fig. 13 (b) is the other interpretation of Join-trees4. It means that both a paper belonging to the $K_1$-related conference and a $K_2$-related paper are written by the same $K_3$-related author.

The final result of Join-trees4 is the union of the Fig. 13 (a)’s result and the Fig. 13 (b)’s result.

The remaining task we must do is to prepare new XRjoin operators to satisfy the demands of these join-trees.

Fig. 13  The interpretations of Join-tree4.
Keyword Search on Both XML and Relational Data

271

Fig. 15  The instances of Conference and Paper.

Fig. 16  XRjoin ((Conference, {tuning}), (Paper, {SQL})).

Type 2
XRjoin ((X, {K11, K12, ...}), (R, NULL))

(a) The XRjoin has keywords in X.

Type 3
XRjoin ((X, NULL), (R, {K21, K22, ...}))

(b) The XRjoin has keywords in R.

Type 4
EXRjoin ((X, {K11, K12, ...}), (R, {K21, K22, ...}))

(c) EXRjoin linked by another hybrid entity in the join-tree

Fig. 17  The other notations of XRjoin.

including “tuning”. The second tuple means that a paper of “SQL” belongs to a conference with a session of “tuning”.

Based on the normal type Type 1, we apply XRjoin to various join-trees by other three types.

The Type 2 is written as follows:
XRjoin ((X, {K11, K12, ...}), (R, NULL))

(2)

The expression (2) means to combine the LCA’s sub-trees of XMLs in X including {K11, K12, ..., KDfree} with the tuple of IDfree in R. IDfree refers to any tuple of R; namely, this tuple is not given any restriction. Fig. 17 (a) shows a part of a join-tree where XRjoin of Type 2 happens. XRjoin of Type 2 has restrictions on X only, and provides temporary results for the next join that will be done on IDfree.

The Type 3 is written as follows:
XRjoin ((X, NULL), (R, {K21, K22, ...}))

(3)

The expression (3) means to combine the LCA’s sub-trees of XMLs in X including {ID21, ID22, ...} with tuple sets in R satisfying {K21, K22, ...}. Fig. 17 (b) shows that XRjoin of Type 3 has restrictions on R only. Join-tree5 and Join-tree6 in Fig. 12 use this XRjoin.

The Type 4 is written as follows:
EXRjoin ((X, {K11, K12, ...}), (R, {K21, K22, ...}))

(4)

When an XRjoin including keywords in both X and R is further linked with another entity in a join-tree, we use a specialized XRjoin of Type 4, denoted by EXRjoin.

The expression (4) shown in Fig. 17 (c) means to combine the XML data x with the relational data r and r’, where

- x is a LCA’s sub-tree of XMLs in X including {K11, K12, ..., KD21, KD22, ..., KDfree}, and
- r is a tuple-set in R satisfying K2 = {K21, K22, ...}, and
- r’ is the tuple of IDfree (i.e., any tuple) in R.

The resulting hybrid entity of (4) is a temporary table which will be joined next on the tuple r’ of IDfree in R.

In fact, the tuple-id IDfree of r’ must appear in the XMLs of X that satisfy K1 = {K11, K12, ...} and all KD2 for {K21, K22, ...}. Thus the computation of the expression (4) is efficient.
Our system uses XRjoin of Type 4, EXRjoin, to interpret the Join-tree 4 in Fig. 11. As has been stated, we union the results of two join-trees in Fig. 13 (a) and (b) for the Join-tree 4. Fig. 13 (a) first executes EXRjoin \( ((Conference, \{K1\}), (Paper, \{K2\})) \) and XRjoin \( ((Authors, \{K3\}), (Paper, NULL)) \) (Type 2). Then, between the result of EXRjoin and the result of XRjoin of Type 2, the natural join will be done on the tuple of ID\textsubscript{free} in Paper. In the same way, Fig. 13 (b) executes EXRjoin between Authors and Paper, and XRjoin of Type 2 between Conference and Paper, and the natural join between the results of the two XRjoins.

Here, the author present an illustration for an EXRjoin. Based on the contents of Fig. 15, we do EXRjoin \( ((Conference, \{link\}), (Paper, \{authority\})) \).

The keyword “link” exists in “S_title” of the session: SID:S005 in XML1. The keyword “authority” exists in “Title” of the tuple PID:P005 in Paper.

**EXRjoin ((Conference, \{link\}), (Paper, \{authority\}))** (Fig. 18) is to extract relevant sub-trees from "link"-related XMLs in Conference, with attaching the corresponding relational data in Paper. These relational data are the "authority"-related tuple \( P_{K2} \) and a tuple \( P_{free} \) (i.e., any tuple in Paper) that exists in the same XML as \( P_{K2} \) exists. We use \( P_{free} \) (in a red box) to do the next join in the join-tree. Note that the resulting XML sub-trees are rooted by the LCAs satisfying all of “link”, “authority” and \( P_{free} \).

In this example, it extracts the LCA from the element "S_title" of SID:S005 (including "link") and the paper element PID:P005 (linking "authority") and a free paper element (denoted by \( P_{free} \) in Fig. 18). The sub-tree rooted by the LCA appears as a new XML in CXML column. The resulting table in Fig. 18 consists of the sub-trees including “link”, the tuples including “authority” and the free tuples used to join with other entities in the join-tree.

As a result, the first hybrid tuple in Fig. 18 means that a paper PID:P005 including “authority” belongs to the session SID:S005 including “link”. This result must be joined on the tuple of the same paper PID: P005 with other entities in the join-tree. The second hybrid tuple means that the paper PID:P005 including “authority” belongs to a session SID:S005 including “link” and that the session has a keyword-unrelated paper PID:P006. This result will be joined on the paper PID:P006. The third is understood by the same way.

### 4.2 The XRjoin Procedure

Basically, we execute the XRjoin between \( X \) satisfying \( K_1 \) and \( R \) satisfying \( K_2 \) as follows.

**Step 1:** XRjoin selects the tuple sets of \( R \) that satisfy \( K_2 = \{K_{21}, K_{22}, \ldots\} \) and those corresponding IDs in \( X \).

In case of \( K_2 = \emptyset \) or EXRjoin, we further find all relevant ID\textsubscript{trees} in \( R \).

**Step 2:** XRjoin selects XMLs that must include both all keywords in \( K_1 = \{K_{11}, K_{12}, \ldots\} \) and all IDs of relevant relational tuple set of step 1. If \( K_2 = \emptyset \), the XML sub-trees just are restricted only by the IDs.

**Step 3:** XRjoin gets the DeweyID of a LCA for each combination of the elements satisfying all keywords, and calculates the corresponding score for the sub-tree rooted by the LCA.

**Step 4:** Finally, the SQL/XML query statement will be created and executed to get the hybrid results of XRjoin, which combine the sub-tree of XML with relational tuple sets related to keywords, ordered by the scores of sub-trees.

The authors use SELECT statements to do Step1 and Step2 and store the results in temporary tables.
builds two stored procedures to finish calculation at Step 3. Step 4 publishes the SQL/XML query statement to get the resulting hybrid tuples of XRjoin.

At Step 3, given an LCA lca of an XML x, we set score(lca) = 10^m, where m is the difference of the tree's height between lca and x's root node. The greater the difference is, the lower score is. The higher score means that keywords are more concentrated in XML sub-trees. So we list the hybrid tuples in descending order of the score.

The execution plan of our system is to do XRjoins first between all hybrid entities and related relational entities in join-trees, and then do natural joins between the results of XRjoins and remaining entities of join-trees.

5. Experiments

The authors make a real data test on a machine with an Intel Core(TM) i7 CPU of 2.80GHz and 3.24GB of RAM, running the Windows XP operating system. We construct our system on a hybrid XML-RDB by using IBM DB2 V9.5 and JAVA 1.5.

The authors use a real dataset of DBLP in our experiments, based on the schema of Fig. 8 without Citation. We store all information of VLDB, SIGMOD and ICDE from DBLP.xml, which is represented by 101 hybrid tuples in Conference, and 9,212 hybrid tuples in Authors, and 8,267 tuples of Paper.

The authors use 100 keyword queries having from two to five keywords. The keywords are selected randomly from the set of words that are of frequent occurrences in the XML-RDB.

For each query, we issue it to the system and execute all join-trees without interaction. Namely, the execution time of a query is set to the time from when the query is issued to when results of all join-trees are returned. As a result, the execution time of a query is from 0.234 seconds to 4.468 seconds. The average execution time of a query is 1.15 seconds.

In the above test, multiple XRjoins of different types are executed in total. We focus on the time of executing XRjoins of each type. Fig.19 shows the average execution time for one XRjoin, whose type is from Type1 to 4, in the above test. This figure excludes the case where a result of XRjoin becomes empty. The x-axis is the number of keywords given to one XRjoin of Type-k (=1 to 4), and the y-axis is its average execution time.

XRjoins of Type 1 and Type 3 get hybrid results that are returned to users directly. XRjoin of Type 1 has keywords in both XMLs and the relational data, which gets the results using 0.28 seconds when the number of keywords is 4. XRjoin of Type 3 has keywords in the relational data only and uses 0.27 seconds when the number of keywords is 4. XRjoins of Type 2 and Type 4 get temporary hybrid results that will be utilized to do a next join. XRjoin of Type 2 uses 1.96 seconds, when XMLs include four keywords. XRjoin of Type 4 uses 1.30 seconds when the relational data and XMLs are hit by four keywords totally. It is a reasonable time for users to get exact results for searching hybrid data under XML-RDB. In order to speed up the execution of XRjoins, we save the cost of communications between databases and applications by utilizing our defined Stored Procedure to calculate the Dewey ID of LCA in DB2.

Finally, we present a snapshot of our experimental system as shown in Fig. 20.

This is an example of Query = \{link, sanjay\}. The original data about “link” is a session title of conference VLDB 2004, and "sanjay" is the first name of an author. The results are derived from a join-tree (in the red box) which does XRjoin twice and natural join once (in the blue box).
The answer of No.1 in Fig. 20 means that an author named “sanjay Agrawal” has written a paper $p$ of PID:conf/vldb/AgrawalCKMNS04 in the conference VLDB 2004, which has a session $s$ of “link”. The answer cannot be obtained by existing techniques.

DBXplorer cannot get this answer, because the paper $p$ does not belong to the session $s$ directly. DISCOVER ($T_{\text{max}} = 5$) also cannot get the answer of Fig. 20, because it must do six joins as stated in section 2.2.2. If $T_{\text{max}} = 6$, to get such an answer, the cost is expensive to calculate many intermediate CNs. XRANK cannot get the answer of Fig. 20, because it contains two kinds of XML sub-trees with different hierarchies that are linked by relations.

Consequently, our proposed system has reasonable performance without any limitation such as $T_{\text{max}}$ of DISCOVER, even if XMLs are deep and heterogeneous.

6. Conclusions

The paper firstly pointed out the problems of keyword search on RDB and XML DB and analyzed the difference of results among those methods.

In view of information integration between XML and relational data, we proposed a keyword-search method on XML-RDB including the ability of both DBXplorer and XRANK. Our unique advantage is that our system can get appropriate answers when some substructures of XMLs are related by relational linkage information, which the existing keyword-search methods cannot get.

To do so, based on the algorithm of DBXplorer, we challenged to generate join-trees on an acyclic schema of XML-RDB without considering the self-loop relationship. We extended the join-trees to retrieve information in XMLs to cover the ability of XRANK. For these various join-trees, we designed four types of XRjoin to interpret their semantics. As XML data can be stored in relational databases, we utilized our join-tree generation and XRjoins to formalize keyword search on an XML-Relational database.

Based on our proposal, an XML-RDB system was built on DB2 V9.5 and stored a real data set DBLP [10]. Experiments showed that our proposed system is effective and feasible to do keyword search on XML and relational data. In our experimental system, we not only can get the answers that both DBXplorer and XRANK, but also can get relationally-related XML sub-trees that the two systems cannot obtain.

As our future work, we will consider the self-loop relationship in join-trees and generate join-trees on a cyclic schema.

References


