XML Keyword Search

CIS650

Yong Cha

Introduction

- One of the goals of XML is to represent document information and we might want to perform a keyword search on particular elements.
- The goal of the paper is to combine database-style query language with free-text search.

Role of RDBMS in XML keyword search

- XML is rapidly becoming data format of choice, but retrieving data from large XML sources can be a painfully slow experience.
- What if you want to search 100MB XML file?
- To improve XML query performance, RDBMS is used to extend XML-QL.

Why use RDBMS

- It is easy to build an extended XML query processor on top of RDBMS because most of the functionality are already built in.
- RDBMSs are universally available.
- RDBMSs allow to mix XML data with other types of data.
- RDBMSs have very high performance.

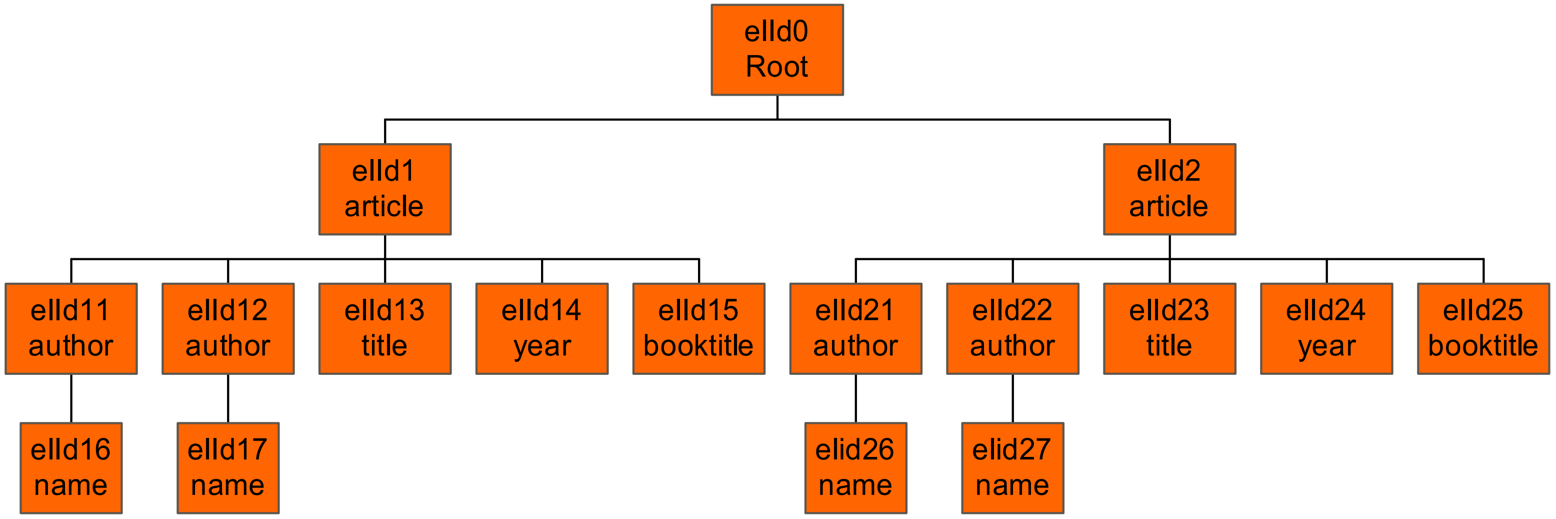
XML

- Extensible Markup Language
- XML and HTML are subsets of SGML (Standard Generalized Markup Language)
- Comparison of SGML and XML
- http://www.w3.org/TR/NOTE-sgml-xml-971215

Sample XML

```
<document>
<article id=“1”>
    <author><name>Adam Dingle</name></author>
    <author><name>Peter Sturmh</name></author>
    <author><name>Li Zhang</name></author>
    <title>Analysis and Characterization of Large-Scale Web
    Server Access Pattern</title>
    <year>1999</year>
    <booktitle>World Wide Web Journal</booktitle>
</article>
</document>
```

XML as Tree


elId0
Root
elId1
article
elId2
article
elId11
author
elId12
author
elId13
title
elId14
year
elId15
booktitle
elId21
author
elId22
author
elId23
title
elId24
year
elId25
booktitle
elId16
name
elId17
name
elid26
name
elid27
name

XML-QL

- where (XML-pattern [ELEMENT_AS $elem_var]) IN fileName, (predicate)* construct XML-pattern | variable
- Example: Retrive all articles written by Mr. Dingle in 1999 about the web.

```
where <article><author><name>$N
</name></author><title>$T</title><year>1999</year>
</article> ELEMENT_AS $E IN “bib.xml”,
$N like *Dingle* , $T like *web*
construct $E
```

Extending XML-QL with Keyword Search

- XML-QL is a good way to query XML document if you know the structure of the document.
- If user is not familiar with the structure queries get messy.

What if user wants to search for keyword ‘name’ without knowing the structure

Things get messy!!!

```
/* search for “name” as a subelement, at any depth */
where <article><author><*><name>$N</name></*></author>
        <title>$T</title><year>1999</year>
        </article>ELEMENT_AS $E IN “bib.xml”
        $N like *Dingle*, $T like *web*
construct $E
union
/* search for “name” as an attribute name, at any depth */
where <article<author><*><! name=$N></_></*></author>
        <title>$T</title><year>1999</year>
        </article> ELEMENT_AS $E IN “bib.xml”,
        $N like *Dingle*, $T like *web*
construct $E
```

Contains predicate

- Contains predicate is introduced to simplify queries.
- contains (tag_name, attribute_name content, attribute_value)

```
where <article> </article> ELEMENT_AS $E IN “bib.xml”,
contains($E,”Dingle”, 3, any),
contains($E, “1999”, 3, any),
contains($E, “web”, 3, any)
construct $E
```

Inverted Files

- What happens when there are too many big documents?
- Use inverted files.
- The classic index structure for keyword search.
- In the simplest form, inverted files contain word and document.
- For XML data, the paper used form <word, elId, depth, location>

<"article", elId1, 0, tag>

<"id", elId1, 1, attr>

How to create inverted files

- You want to index following text:
- A cookie is just a cookie, but fig newtons are fruit and cake.
- <a, 1> <a, 17>
- <cookie, 3> <cookie, 19>
- <is, 10>
- <just, 13> etc.

Storing Inverted Files in RDBMS

- Inverted files can be easily stored as tables.
- To improve the performance, inverted files are partitioned by words.
- Further partitioning is done by combining word with type.
- word-type(elId, depth, location)
- Adam-name(elId16, 4, 1)

Problem with Inverted Files

- Indexes can grow to be bigger than the source.
- Size of original XML file: 7.7MB
- Size of relation database: 90MB

Extended XML-QL Query Processing

- First, create binary table for all XML tags.
- tag-name(source, target, value)
- article(“article”, “author”, null)
- author(“author”, “name”, null)
- name(“name”, null, “Adam Dingle”)
- title(“title”, null, “Analysis … “)
- Once the tables are filled XML-QL queries can be executed as SQL.

```
select art.source
from article art, author aut, name n, title t, year y
where art.target=aut.source and art.target=t.source and
art.target = y.source and aut.target=n.source and y.value=1999
and t.value like “web” and n.value like “Dingle”
```

Keyword search

- Once you have inverted file, keyword search become simple select.
- where <article> </article> ELEMENT_AS $E IN “bib.xml”, contains($E, “Dingle”, 3, any) construct $E
- simply becomes
- select elId from Dingle-article

Combining XML Query with Keyword Search

- Both query and keyword search can be combined easily into a single SQL statement.
- Where <article> <year>1999</year><author>$A</author> </article> ELEMENT_AS $E IN “bib.xml”, contains($E, “Web”, 4, any) construct $A
- becomes
- select aut.target from article art, year y, author aut, web-article c where art.target=year.source and art.target=aut.source and art.target=c.elId and year.value=“1999” and c.depth<=4

Distributed XML Query Processing

- Problem arises when XML data cannot be stored in RDBMS.
- Web site might give a permission to index the data without giving a permission to store the content.
- Two scenarios arise.
- XML data source without query capabilities and XML data source with query capabilities.

XML data sources without query capabilities

- Prefilter: User the inverted file stored in the RDBMS to find the relevant documents and/or elIDs.
- Retrieve: Get the relevant documents (or elements) from the data sources.
- Execute: Extract the query results from the retrieved documents (or elements).

XML data sources with query capabilities

- Prefilter: User the inverted file stored in the RDBMS to find the relevant documents and/or elIDs.
- Push Down: Pass elIds to the data sources and let the data sources execute the whole or parts of the query on these elIds.
- Refinement: Refine the results returned by the data sources if the data sources could not execute the whole query.

Query performance

- Experiments were done on three scenarios.
- Structured: The query exploits the full XML structure.
- Partially structured: The query exploits some structure.
- Unstructured: The query has no structure.

Performance

	Query 1		Query 2	
	Running Time	Query Result	Running Time	Query Result
Structured	3 secs	38 lines	7 secs	208 scenes
Part. Structured	1.5 secs	19 scenes	2.5 secs	1108 scenes
Unstructured	4 secs	56 elements	10 secs	10909 elements

Questions