RDBMS의 질의 결과 값을 XML 형태로 변환하는 XML Reconstruction 기법

이재호° 홍동권 남재열
계명대학교 정보통신학부 컴퓨터공학과 데이터베이스 실험실
sieg0623@naver.com

The XML Reconstruction technique which converts the question result price of the RDBMS in XML form

Jaeho Lee° D.K. Hong J.Y. Nam
Kei-Myoung University. The Department of Computer Engineering

요 약

오늘날 HTML을 대체하기 위해 등장한 XML은 디지털 정보교환 형식의 표준으로 자리 잡은 후 XML 문서를 데이터베이스에 저장하고 원하는 정보를 효율적으로 질의한 후 결과를 출력하는 연구가 활발히 진행되고 있다. 본 논문에서는 XML 문서를 미리 설계되어진 Analyzer를 이용해서 관계형 테이블에 저장한 후 사용자가 XQuery를 사용하여 질의를 한다. 변환기에 의해서 SQL로 데이터베이스에 질의를 하게 되고 그 결과는 테이블에서 다시 XML 형태로 재생성하여 사용자는 XML 형태의 결과를 볼 수 있다. 본 논문에서는 XQuery로 질의한 결과를 다시 XML로 재생성하는 기법에 대한 설명과 관련 테이블의 구조와 구현 과정을 비롯하여 여기서 제시한 재생성 기법을 recursive function으로 구현한 경우와 반복문으로 구현한 경우를 테스트하여 recursive function으로 구현한 경우가 반복문으로 구현한 경우보다 재생성하는 시간이 빠르다는 것을 확인하고 보다 효율적이라는 결론을 제시한다.

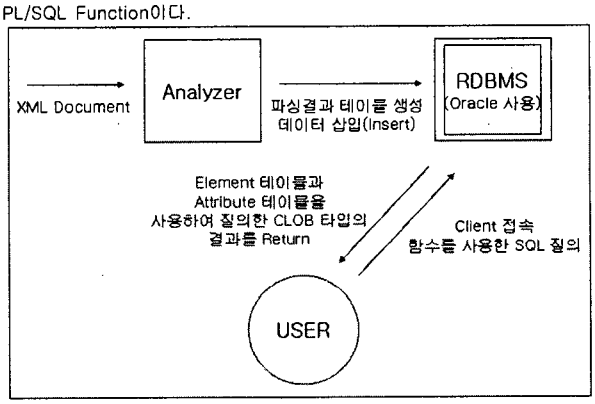
1. 서 론

W3C(World Wide Web Consortium)에 의해 제안된 XML(Extensible Markup Language)은 HTML을 대체하는 디지털 정보교환 형식의 표준으로 지정되었다. 현재 XML 문서를 저장하고 검색할 수 있는 XML 전용 데이터베이스가 많이 연구 개발되고 있다. 하지만 그 성능이 아직까지는 정확히 검증된 바가 없으며 20~30년 동안 기술적, 상업적으로 급속한 발전을 이룬 관계형 데이터베이스를 따라오지 못하고 있다. 이 두 가지를 좀 더 구체적으로 비교하자면 관계형 데이터베이스로 XML 문서를 관리 할 경우 다음과 같은 장점을 가질 수 있다. 첫 번째로 XML 문서에 대한 질의 처리를 하나의 시스템 상에서 구현할 수 있으므로 하나의 통합된 검색 시스템을 구축할 수 있다. 둘째, 20년 이상 계속된 RDBMS에 대한 연구의 결과라고 할 수 있는 질의 최적화, 질의 수행, 병행 제어, 회복 기법 등을 그대로 XML 질의 처리 시 이용할 수 있으므로 시스템의 안정성을 보장한다. 셋째, RDBMS상에서 텍스트의 일부분을 포함하는 질의일 경우 XML 전용 데이터베이스보다 효율적으로 처리 할 수 있다. 마지막으로 RDBMS는 거의 모든 단체나 기관에서 데이터를 저장하기 위해 사용되므로 장비의 교체 같은 추가적인 비용이 들지 않는다. 이런 이유로 현재 관계형 데이터베이스를 이용하여 XML 문서를 저장하고 검색하기 위한 연구가 활발히 진행되고 있다.

본 논문에서는 XML 문서를 Full Text Search를 위하여 미리 설계되어진 Analyzer를 이용해서 관계형 테이블에 저장한 후 사용자가 XQuery를 사용하여 질의를 한다. 변환기에 의해서 SQL로 데이터베이스에 질의를 하게 되고 그 결과는 테이블에서 다시 XML 형태로 재생성하여 사용자는 XML 형태의 결과를 볼 수 있다. 본 논문에서는 XQuery로 질의한 결과를 다시 XML로 재생성하는 기법에 대한 설명과 관련 테이블의 구조와 구현 과정에 대해 설명하고 여기서 제시한 재생성 기법을 recursive function으로 구현한 경우인 'recursive_sql'와 반복문으로 구현한 경우인 'reiterate_sql'를 테스트했을 경우 recursive function으로 구현한 경우가 우수한 성능을 보장한다.

2. 전체 구성 및 테이블 구성

XML 문서는 Analyzer에 의하여 Element, Attribute, Location, Word, XML_DOCUMENTS에 저장된다. 본 글에서 다루고자하는 XML 재생성 기법은 Element table과 Attribute table을 이용하여 특정 Element의 정보와 모든 자식 정보를 XML 형식으로 Return하는 PL/SQL Function이다.



<그림 1. 전체 구성 및 환경>

XML문서를 Analyzer를 이용한 결과 다음의 다섯 가지 테이블에 저장된다.

본 연구는 산업자원부 지역연구개발클러스터구축사업 중 경북대학교 첨단진단/예측 의료기술 클러스터 사업단의 연구비 지원을 받아 수행되었음.

XML_DOCUMENTS (ID, DOCNAME)
WORD (WORD, POSITION, DEPTH, DOCID, EID, PATHID)
LOCATION (DOCID, PATHID, PATH, DEPTH, PATH_CNT)
ELEMENT (DOCID, EID, NAME, SIBORD, PATHID, KEY_COUNT, PID, INFO, VALUE)
ATTRIBUTE (DOCID, EID, AID, ANAME, AVALUE)

각각에 대해서 설명하면 XML_DOCUMENTS 테이블은 XML 문서의 식별자, 문서이름과 문서 내용 등이 저장된다. WORD 테이블은 단순한 키워드 검색에 사용되며 단어와 단어의 위치정보를 저장한다. LOCATION 테이블은 XML 문서 내에 존재하는 경로에 대한 정보를 저장하고 경로 식별자와 경로, 루트 element에서 단말 element까지의 깊이, 동일 경로의 횟수 등의 경로에 대한 정보를 저장한다. ELEMENT 테이블은 XML 문서 내에 존재하는 element의 세부 정보를 저장하는 테이블이다. ATTRIBUTE 테이블은 element에 해당하는 attribute의 정보를 저장하는 테이블이다.

ELEMENT 테이블에서 DOCID는 XML 문서를 구별을 위한 식별자이고, EID는 element의 구별을 위한 식별자이다. NAME은 엘리먼트의 이름, SIBORD는 동일한 부모 element의 자식들 사이에서의 순서를 나타낸다. PID는 element의 부모 element의 EID 값을 가지는(예: PID가 10인 element의 부모는 EID 값이 10인 element가 된다) 컬럼이다. INFO는 한 element가 자신의 텍스트가 있고 element가 오고 또 다시 자신의 텍스트 값이 나오는 것과 같은 특이한 경우에 텍스트의 위치를 파악하기 위한 위치정보를 나타내는 컬럼이다. VALUE는 element의 텍스트를 가지는 컬럼이다. ELEMENT 테이블의 Primary Key는 DOCID, EID, INFO로 구성된다.

ATTRIBUTE 테이블에서 DOCID와 EID는 ELEMENT 테이블과 같고, AID는 DOCID와 EID와 더불어 Key 필드를 구성하는 속성 구별 값이다. ANAME은 속성의 이름이고 AVALUE는 속성 값을 가지는 컬럼이다. ATTRIBUTE 테이블의 Primary Key는 DOCID, EID, AID로 구성된다.

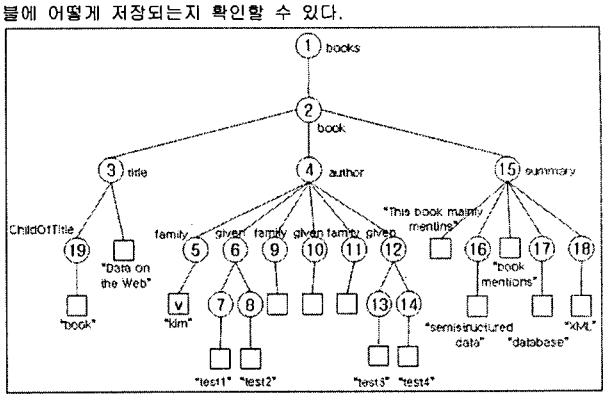
XML 문서가 파싱되어 ELEMENT 테이블과 ATTRIBUTE 테이블에 저장되는 간단한 예를 위한 XML 문서인 'books.xml' 문서가 그림 2에 있다.

```
<books>
  <book>
    <title><ChildOfTitle>book</ChildOfTitle>
      Data on the Web
    </title>
    <author first = "smith", last = "robert">
      <family>kim</family>
      <given>
        <test1>test1</test1>
        <test2>test2</test2>
      </given>
      <family>Lee</family>
      <given>Eun Suk</given>
      <family>Hong</family>
      <given>
        <test3>test3</test3>
        <test4>test4</test4>
      </given>
    <summary>
      This book mainly mentions
      <keyword>semistructured data</keyword>
      Book mentions
      <keyword>database</keyword>
      <keyword>XML</keyword>
    </summary>
  </book>
</books>
```

<그림 2. books.xml 문서의 구조>

'○'는 element를 나타내고, '□'는 텍스트를 나타낸다. 이 문서는 INFO의 용도와 attribute를 보여주기 위해 약간의 변경이 가해진 문서이다. ELEMENT 테이블에서 INFO는 3번 element처럼 텍스트의 출력 순서가 시작 element 바로 다음에 나오는 것이 아닐 경우 INFO 값을 보고 출력 순서를 제어할 수 있다. 15번 element도 마찬가지 경우이다.

표 1, 2를 보면 XML 문서가 ELEMENT 테이블과 ATTRIBUTE 테이블에 어떻게 저장되는지 확인할 수 있다.



<그림 3. books.xml 문서의 트리 표현>

DOCID	EID	NAME	SIBORD	···	PID	INFO	VALUE
1	19	ChildOfTitle	1		3	19	book
1	3	title	1		2	19	Data on the Web
1	5	family	1		4	5	kim
1	7	makeXMLForm	1		6	7	makeX
1	8	test2	2		6	8	test2
:	:	:	:	:	:	:	:
1	15	summary	3		2	15	This book mainly mentions
1	16	keyword	1		15	16	semistructured data
1	15	summary	3		2	16	book mention
1	17	keyword	2		15	17	database
1	18	keyword	3		15	18	XML

<표 1. books.xml이 저장되는 ELEMENT 테이블>

DOCID	EID	AID	ANAME	AVAULE
1	4	1	first	smith
1	4	2	last	robert

<표 2. books.xml이 저장되는 ATTRIBUTE 테이블>

3. 알고리즘 및 구현

3.1 recursive function으로 구현된 'recursive_sql'

Step 1. element의 정보와 element의 텍스트와 attribute 정보로 PL/SQL 커서를 생성한다.
Step 2. element의 속성(attribute) 값이 있다면 값을 출력한다.
Step 3. 다음 출력이 텍스트이면 value 값을 출력한다.
element의 텍스트 자식의 순서가 고려되어야 한다.
Step 4. element를 출력해야한다면 Step 3을 거치지 않고 순환적으로 함수를 실행한다.

<'recursive_sql' 알고리즘>

Step 1. - 1단계에서는 Collection별로 XML 문서를 구분하여 저장하고 질의하기 위하여 ELEMENT 테이블은 'Collection명_element', ATTRIBUTE 테이블도 마찬가지로 'Collection명_attribute'로 구성된다. 문제 해결을 위해 컬렉션이 정해졌을 때 동적으로 PL/SQL 커서를 생성하기 위하여 '커서변수'를 사용하였다. 커서변수를 사용하면 커서를

본 연구는 산업자원부 지역연구개발클러스터구축사업 중 경북대학교 첨단진단/예측 의료기술 클러스터 사업단의 연구비 지원을 받아 수행되었음.

정의하는 내용을 바꿀 수 있어서 동적인 프로그램을 가능하게 한다. 즉 상황에 따라서 다른 조건을 가지는 질의들을 기반으로 커서를 OPEN할 수 있게 된다. 두 번째로 테이블명이 동적으로 결정되기 때문에 PL/SQL 블록 내에서 SELECT문을 테이블에 따라 동적으로 사용할 수 있게 하기 위해서 '고유 동적 SQL'을 사용하였다.

Step 2. - 2단계에서는 함수가 실행되면서 각 element가 속성 값이 있는지를 검사하고 제어문을 사용하여 속성 값이 있으면 속성 값까지 붙여 결과에 저장하고 속성 값이 없으면 element 이름만 출력하고 넘어가는 형식으로 함수를 수정하였다. ATTRIBUTE 테이블에도 역시 EID 컬럼이 존재하므로 함수의 인자로 들어가는 EID와 같은 값이 ATTRIBUTE 테이블의 EID의 값과 같은 것이 있으면 해당 EID는 속성이 존재하게 되는 것이다.

Step 3. - 3단계에서는 XML 문서에서 하나의 element가 여러 개의 텍스트를 가지고 있고 텍스트가 그 element의 자식 element 사이에 랜덤하게 여러 개가 들어 있을 수 있는 경우를 고려하였다.

DOCID	EID	NAME	VALUE	···	PID	INFO
			···			
			···			
1	19	ChildOfTitle	book		3	19
1	3	title	Data on the Web		2	19

그림 2에서도 알 수 있듯이 3번 element의 텍스트인 'Data on the Web'이 19번 element 다음에 오는 것을 알 수 있다. 위의 ELEMENT 테이블의 일부를 봤을 때에도 마찬가지로 3번 element의 텍스트의 INFO 컬럼의 값은 19이다. 그러므로 INFO 필드를 이용하여 element의 텍스트가 여러 개가 오더라도 위치를 나타내어 XML 문서의 순서와 동일하게 출력되도록 하였다.

Step 4. - Step 2.를 거친 후 다음 출력 내용이 element 일 경우에는 다시 'recursive_sql' 함수를 호출하여 인자로 같은 Collection과 DOCID와 출력해야하는 해당 EID를 넘겨주고 위의 똑같은 단계를 거쳐 인자로 받은 EID에 대한 정보를 출력한다. 이전의 결과는 모두 CLOB 타입의 result 변수에 저장되어 계속 출력내용이 있을 때마다 result의 내용에 덧붙인다. 더 이상의 element가 존재하지 않으면 종료 element를 출력하고 return하게 된다.

3.2 반복문으로 구현된 'reiterate_sql'

Step 1. element의 EID를 저장하는 table open_table, close_table, close2_table을 생성한다.
Step 2. element의 정보와 element의 텍스트의 정보와 attribute 정보로 PL/SQL 커서를 생성한다.
Step 3. element의 속성(attribute) 값이 있다면 값을 출력한다.
Step 4. 출력할 텍스트가 있다면 value 값을 출력하고, element의 자식 element가 있다면 open_table에 저장한다.

<'reiterate_sql' 구현 단계>

Step 1. - 반복문으로 구현하기 위해서는 시작 element, 종료 element에 관련된 정보를 저장하는 곳이 필요하다. 'reiterate_sql'에서는 PL/SQL의 임시적인 테이블을 선택하여 사용하였다. 이 테이블은 인덱스를 이용하여 스택처럼 후입선출 방식으로 사용하며 element의 EID를 보관한다. open_table은 다음 어떤 element가 출력되어야하는지 순서를 저장하며, close_table은 종료 element의 순서를 저장한다. close2_table은 element 종료 순서만 아는 것으로는 언제 종료 element를 출력해야하는지 알 수가 없다는 점을 보완하기 위하여 back tracking 위치를 open_table의 인덱스로 저장하여 close2_table의 값과 open_table의 인덱스 값이 일치하면 close_table의 마지막 element를 출력하도록 한다.

Step 2. - 'recursive_sql' 함수와 마찬가지로 테이블명이 동적으로 결정되므로 커서를 사용하여 attribute, text, element 정보를 알 수 있다.

Step 3. - 이 단계도 'recursive_sql' 함수와 마찬가지로 각 element가 속성 값을 가지는 지를 검사하고 제어문을 사용하여 속성 값이 있으면 속성 값까지 함께 출력하고 속성 값이 없으면 element 이름만 출력한다.

Step 4. - 이 단계에서는 출력할 텍스트가 있다면 텍스트를 출력한다. 그러나 INFO가 같은 텍스트가 두개 이상인 경우는 마지막 텍스트를 제외한 나머지 텍스트를 출력 후 종료 element도 같이 출력하고 마지막 텍스트는 다시 element가 올수도 있고 종료 될 수도 있으므로 종료 element를 출력하지 않는다. 텍스트 출력 후 자식 element가 있다면 자식 element의 EID를 SIBORD의 역순으로 open_table에 추가한다. 추가된 element는 Loop가 반복될 때 하나씩 open_table에서 삭제되며 해당 element에 대한 정보를 출력하게 된다.

출력 내용은 'recursive_sql'과 마찬가지로 CLOB 타입의 result 변수에 덧붙여지며 마지막에 return한다.

4. 성능평가 및 결론

본 논문에서 제안한 recursive function으로 구현한 'recursive_sql'과 반복문으로 구현한 'reiterate_sql' 두 함수를 테스트를 위한 실행 환경은 펜티엄 4 2.4GHz, 메인메모리 512M, 운영체제 Window XP이며 jdk 1.4.05와 jdbc를 이용하여 오라클 9i에서 시간을 측정하였다.

'recursive_sql'과 'reiterate_sql' 두 함수는 동일한 결과를 return하며, 원하는 결과 값을 return 받을 수 있다. 그러나 두 함수는 지금의 테이블 구조와 알고리즘으로는 문서의 크기가 커지면 결과를 얻는데 시간이 많이 걸리게 된다. 'recursive_sql'은 함수를 재귀적으로 호출할 때마다 호출 전 상태를 보존해야하므로 문서의 크기가 커지고 XML 문서의 트리가 깊어지면 메모리 사용이 증가하여 실행시간이 증가한다. 'reiterate_sql'은 재귀적 호출대신에 임시 table을 사용하여 element 정보를 저장한다. 마찬가지로 문서가 커지면 table에 저장되어지는 정보의 양도 증가하고 element가 시작되고 닫히고, 자식 element가 생성될 때마다 세 table을 모두 변경해야 하기 때문에 실행시간이 증가한다.

No	라인수	문서의 크기	'recursive_sql' 실행시간	'reiterate_sql' 실행시간
1	8885 lines	254 KB	00:05:06.61	00:14:21.13
2	16599 lines	507 KB	00:10:43.25	00:29:51.22
3	25055 lines	782 KB	00:16:23.39	00:41:13.09
4	32975 lines	1.01 MB	00:49:51.38	-
5	48395 lines	1.50 MB	00:55:13.66	-
6	63815 lines	2.00 MB	01:39:18.18	-

<표 3. 문서 크기에 따른 'recursive_sql', 'reiterate_sql' 함수의 실행시간 비교>

표 3.에서 보여주는 실행시간은 최악의 경우인 루트 element를 인자로 받아 함수를 실행한 결과이다. 이 실행결과를 통해서 recursive function으로 구현된 'recursive_sql'이 반복문으로 구현된 'reiterate_sql' 함수보다 성능이 더 좋음을 알 수 있다. 'reiterate_sql'의 '-'부분은 실행시간이 너무 오래 걸려 시간 측정을 포기한 부분이다.

현재 구현한 알고리즘에서 발생하는 몇 가지의 문제점을 해결하고 앞으로 여기에서 구현한 방식이 아닌 새로운 알고리즘으로 지금 보다 더 빠른 시간에 수행되는 프로그램을 개발 및 구현하여야겠다.

참고문헌

[1] Yoshikawa, M., Amagasa, T,. "XRel: pathbased approach to storage and retrival of XML Documents using relation database" 2001

[2] Igor Tatarinoy, Stratis D. Yiglas, Kevin B eyer, Javavel Shanmugasundaram, Eugene Shekita, Chun Zhang "Storing and Querying Ordered XML Using a Relational Database System" 2002

[3] David Dehaan, David Toman, Mariano P., Consens, M.Tamer Ozsu "A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding" 2003

본 연구는 산업자원부 지역연구개발클러스터구축사업 중 경북대학교 첨단진단/예측 의료기술 클러스터 사업단의 연구비 지원을 받아 수행되었음.