Dewey order기법을 이용한 RBDMS 환경에서의 XQuery 질의 처리기 설계 및 구현

정민경, 홍동권
계명대학교 정보통신대학 컴퓨터 공학과
e-mail:skallet6050@hotmail.com

Design and Implementation of XQuery Processor on the RDBMS using Dewey order

Min-Kyoung Jung, Dong-Kweon Hong
Kei-Myoung University

요 약

본 논문에서는 Deway order기법을 이용하여 관계형 데이터베이스 환경에서 효율적으로 XML 문서를 저장, 검색, 결과값을 반환하기 위한 XQuery 질의 처리기를 설계하고 구현한다.

우선 첫 번째로 dewey order기법을 이용하여 XML문서를 저장하기 위한 색인 모델을 관계형 데이터베이스에 설계하고 XML문서를 저장한다. 두 번째로 이를 기반으로 XML 전문검색 언어인 XPath식을 SQL로 변환하는 전체적인 알고리즘을 제시한다. 세 번째로 위에서 변환된 SQL문의 질의 결과값을 처음에 저장될 당시의 XML문서의 형태와 Text를 그대로 유지하면서 사용자에게 반환하는 알고리즘을 제시한다. 이 부분은 기존의 발표된 논문에서는 좀처럼 보기 드문 내용으로 XQuery에 포함되는 다양한 형태의 Xpath식을 SQL문으로 변환할 수 있는 정확한 방법 뿐만 아니라 각각 한번의 질의로 얻고자 하는 엘리먼트들과 어트리뷰트들을 찾아 XML문서 그대로 출력하는 방법에 초점을 두어 본 논문을 기술한다.

마지막으로 이를 실제로 구현하고 Test한 결과를 바탕으로 Dewey order기법을 이용하여 XML 색인 모델을 설계 할 경우 SQL문으로 변환하여 질의를 처리하는 측면에서나 질의한 결과값을 XML문서 형태로 반환하는 측면에서나 이 기법을 사용하지 않는 Local order방식보다 성능이 훨씬 우수하다라는 결론을 제시한다.

1. 서론

W3C(World Wide Web Consortium)에 의해 제안된 XML(Extensible Markup Language)이 문서 교환의 표준으로 지정된 후 XML 문서를 저장하고 검색할 수 있는 XML 전용 데이터베이스가 많이 연구개발되고 있다. 하지만 그 성능이 아직까지는 정확히 검증된 바가 없으며 20~30년 동안 기술적, 상업적으로 급속한 발전을 이룬 RDBMS를 따라오지 못하고 있다. 이 두 가지를 좀더 구체적으로 비교하자면 관계형 데이터베이스로 XML문서를 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, 이십 년 이상 계속된 RDBMS에 대한 연구, 예를 들어 질의 최적화, 질의 수행, 병행 제어, 회복 기법 등을 그대로 XML 질의 처리 시 이용할 수 있으므로 시스템의 안정성을 보장할 수 있다. 셋째, RDBMS상에서 Text의 일부분을 포함하는 질의일 경우 XML전용 데이터베이스보다 효율적으로 처리 할 수 있다.. 마지막으로, RDBMS는 거의 모든 단체나 기관에서 그들의 데이터를 저장하기 위해서 사용되므로 추가적인 비용이 들지 않는다.

이런 이유로 현재 관계형 데이터베이스를 이용하여 XML문서를 저장하고 검색하기 위한 연구가 활발히 진행되고 있다. 하지만 기존 발표된 논문에서는 XML문서를 효과적으로 저장하는 기법 또는 제한된 형태의 XPath식만을 대상으로 질의를 처리하는 방법들만 제시할 뿐 그 결과값을 다시 XML형태로 반환하기까지의 전체과정에 대한 구체적인 알고리즘이 제시된 바가 거의 없었다.

이에 본 논문에서는 기존의 관계형 데이터베이스

*본 연구는 한국과학재단 목적기초연구
(R01-2003-000-10001-0)지원으로 수행되었음

환경에서 XML문서를 저장, 이를 검색하기 위한 XPath식의 처리, 결과값을 XML형태로 출력하기까지의 전 과정을 모두 다룬다. 그리고 이를 처리하는데 있어서 Dewey order 기법을 도입하여 실제 Test했을 경우 기존에 제시된 연구 방법들 보다 우수한 성능과 정확성을 보장한다.

2. Dewey order기법을 이용한 XML색인보델 설계

본 논문에서는 비슷한 내용의 XML문서들을 모아 하나의 콜렉션을 생성하여 관리한다. 이는 여러개의 XML문서들을 쓰임새에 따라 일정하게 분류하여 효율적으로 관리하기 위함이다.

본 연구에서 제안하는 Deway order방식을 이용한 XML 색인모델은 다음과 같이 5개의 table을 사용한다.

Collection_Document(id, name, contens)
Collection_Llocation(id, pathid, path, depth, path_cnt)
Collection_Word(id, position, depth, word, eid, pathid)
Collection_Attribute(id, eid, name, value)
Collection_Element(id,eid, name, pathid, info, value key_count ,sibord, pid, numbering)

위의 table들을 간단히 설명하자면 Collection_Document 테이블은 비슷한 내용의 XML문서들마다 유일하게 부여된 id값과 실제 내용이 저장된다. CollectionName_Location테이블은 XML문서를 트리형태로 표현했을 경우 루트노드부터 자식들까지의 깊이와 경로가 저장된다. 이때 각 각 경로를 구별하는 Step('/') 앞에는 '~'문자를 결합하여 저장한다. 예를 들면 book/title/price와 같은 경로일 경우 깊이가 3이며 각 '/'앞에는 '~'를 삽입하여 book~/title~/price와 같은 형태로 저장된다. 이는 XPath식을 처리 할 때 '//'와 같은 Step이 올 경우 '~%/'로 바꿔 처리되므로 정확한 결과값을 찾을 수 있다.. Collection_Word테이블은 각 엘리먼트들이 가지는 Text와 이를 이루는 키워드 ,키워드의 위치 정보를 가지고 있다. Collection_word테이블은 엘리먼트들의 어트리뷰트의 정보를 나타낸다. Collection_Element 테이블은 각 엘리먼트들의 Text와 XML트리상에서 순서대로 부여된 즉 Local order기법으로 매겨진 유일한 eid값, 그리고 그들의 부모 id, 자신의 위치와 순서 및 영역을 나타내는 numbering값으로 구성된다. 여기서 numbering 컬럼은 eid컬럼과 마찬가지로 각 엘리먼트들이 가지는 유일한 값을 가지고 있으며 지금부터 소개 할 Dewey order기법을 이용하여 이 값들이 생성된다. 즉 최고 루트노드에는 #1#부터 시작하여 그의 첫 번째 자식은 #1#1#, 두 번째 자식은 #1#2#을 부여한다. '#'문자는 자신의 numbering값과 부모의 numbering값을 구별하기 위한 용도로 사용된다. 이렇게 각 엘리먼트들의 위치, 형제들간의 순서, 그리고 자신의 부모를 포함한 조상들까지의 순서값을 모두 포함하면서 일련의 유일한 값을 매기는 방식을 Dewey order방식이라 부른다.

```
<bib>
 <book>
  <title>TCP/IP Illustrate</title>
 <author>Stevens</author>
 </book>
<bib>
```

id	eid	name	.....	pid	numbering
1	1	bib			#1#
1	2	book		1	#1#1#
1	3	title		2	#1#1#1#
1	4	author		3	#1#1#2#

<Collection_Element table>

위의 예에서 보는 봐와 같이 각 엘리먼트마다 Dewey order방식으로 매겨진 numbering 값은 XMLtree상에서 자신의 위치, 형제들간의 순서를 나타낼 뿐만 아니라 부모의 numbering 값을 포함하고 있으므로 자신이 소속된 영역도 같이 나타낸다. 이런 Dewey order만의 특징은 뒤에서 소개 할 XQuery에 포함 되는 XPath식을 처리하는 부분이나 질의한 결과값을 XML형태로 처리하는 부분에서 성능을 높이는데 큰 역할을 하게 된다.

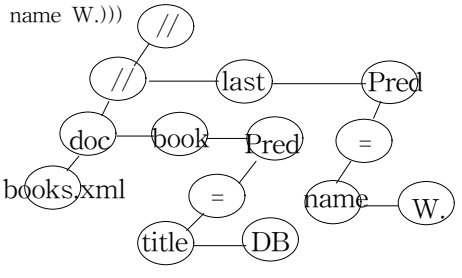
3.XPath식을 SQL로 변환하는 알고리즘

사용자가 입력한 XPath식을 처리하기 앞서 우선 XML전용 데이터베이스인 eXist의 parser 일부분을 가져와서 XPath식을 AST 라는 Tree형태로 변환한다. 이 AST Tree는 자식과 형제 노드만을 가지는 트리로 이들의 형태를 LISP언어의 형태와 유사하게 표시할 수 있다.

예를 들면 다음과 같다.

<Ex1>

doc('books.xml')//book[title='DB']//last[name='W.']
(//(//(doc books.xml') book (Pred (= title DB)))last (Pred (= name W.)))



위의 <Ex1>는 books.xml이라는 XML문서에서 book의 2번째 title을 찾고 그의 자식 노드중 lastest의 값이 Tom인 last노드를 검색하는 구문이다.

Tree형태로 변환된 XPath식을 다음과 같은 3개의 알고리즘을 거치면 완전한 SQL문이 생성된다.

(1)OutputNodeSeek(AST Node)

AST트리에서 찾고자 하는 결과노드를 검색하는 알고리즘으로 루트노드부터 시작하여 부모 우측자식, 좌측자식 순으로 운행하되 가장 처음으로 만나는 Qname타입(엘리먼트명을 가진 노드)을 출력할 Output노드로 정한다. <EX1> 경우 Output노드는 last이다.

(2)SQL_Creator(Collection_Name, AST node)

Output노드를 찾은 다음 AST트리를 Preorder방행으로 운행하면서 다음과 같은 타입의 노드를 만날 경우 각 과정을 거치면서 from절과 where절을 생성하게 된다.

A. '/' 또는 '//': step을 Stack에 push한다

B. Qname타입: Stack을 pop하여 Qname과 pop한 step을 연결하여 하나의 Path식을 형성한다. 그리고 Output 노드와 자신을 비교하여 같으면 from절과 where절을 생성한다.

C. Pred타입: Qname타입을 거치면서 생성된 Path로 from절과 numbering 값, path를 비교하는 where절을 생성한다. 만약 이 노드의 자식으로 숫자타입의 노드가 오면 I_TH_CHILD프로시져를 호출하는 구문을 생성한다. 이는 특정 엘리먼트의 I번째 자식노드를 검색하는 프로시져로 book /title[2]와 같은 식을 예로 들 수 있다.

D. Text타입: 현재까지 Qname타입을 거치면서 생성된 Path로 from절과 where절을 생성하되 Text를 검색하는 구문을 추가한다.

E. contains타입: Collection_Word테이블의 word컬럼을 참조하는 SQL문을 생성한다.

F. doc타입 : books.xml 문서의 id를 찾는 구문을 생성한다.

(3)Concatenation(Output_node)

위의 알고리즘에서 생성된 where절에 id를 비교하는 구문을 삽입한 뒤 from절과 Output노드를 찾는 select문을 결합하여 완전한 SQL문을 형성한다.

지금까지 살펴본 3개의 알고리즘을 거쳐 생성된 <EX1>의 SQL문과 그에 적용된 단계는 다음과 같다. 단 books.xml문서는 how콜렉션에 존재한다.

```
SELECT E2.docid, E2.numbering((3))
FROM how_ELEMENT E0, how_LOCATION L0, (B)
how_ELEMENT E1, how_LOCATION L1, (D)
how_ELEMENT E2, how_LOCATION L2, (C)
how_ELEMENT E3, how_LOCATION L3(D)
WHERE L0.path like '~%/book‘(B)
and E0.id = L0.id(B)
and E0.pathid = L0.pathid(B)
and L1.path like '~%/book~/title'(D)
and E1.numbering like E0.numbering || '%'(D)
and E1.pathid = L1.pathid(D)
and E1.id = L1.id(D)
and (trim(E1.value)='Data on the Web' )(D)
and L2.path like '~%/book~%/last'(C)
and L2.pathid = E2.pathid(C)
and L2.id = E2.id(C)
and E2.id in (Select id from how_documents where docname = 'books.xml')(F)
and E2.numbering like E0.numbering || '%'(C)
and L3.path like '~%/book~%/last~/lastest' (D)
and E3.numbering like E2.numbering || '%' (D)
and E3.pathid = L3.pathid (D)
and E3.id = L3.id (D)
and (trim(E1.value)='Data on the Web' or trim (E3.value) ='Abite' (D)
)
and E2.id = E3.id((3))
and E3.id = E1.id((3))
and E1.id = E0.id;((3))
```

위의 SQL문을 통해 알 수 있듯이 title이 DB인 book의 자손노드인 last를 numbering컬럼을 조인하여 한번의 질의로 결과값을 찾고 있다. 만약 Dewey order기법을 적용하지 않고 XML트리상에서 일정한 순서대로 유일한 id값을 각 노드에게 부여하는 Local order방식을 적용한다면 title이 DB인 book의 id를 찾고 이를 부모로 가지는 노드의 id를 찾고 다시 이 를 부모로 가지는 노드의 id를 찾는 등 last를 찾을 때까지 subquery를 생성하여 반복적으로 질의해야된다. 즉 XML트리상에서 book 노드의 깊이와 last의 깊이의 차이가 많아 질 수록 질의를 계속 수행해야하므로 그 만큼 성능이 떨어지게된다. 물론 이를 프로시져를 통해 처리 할 수도 있다. 하지만 프로시져를 구현하여 Test해 본 결과 결과노드를 찾을 때 까지 질의가 반복적으로 수행되므로 한번의 질의로 처리하는 Dewey order방식보다 성능이 떨어지게 된다. 위의 예제 뿐만 아니라 Predicate 즉 어떤 조건을 만족하는 노드의 자손을 찾는 유형의 XPath식일 경우 Dewey order방식은 한번의 질의로 결과값을 정확히 찾는 반면 Local order방식은 질의를 하고 다시 조인하는 작업을 반복적으로 수행하여 결과값을 찾으므로 성능이 저하된다.

4. 질의한 결과를 XML형태로 반환하는 알고리즘

질의한 결과를 XML형태로 반환하기 위해 본 논문에서는 Dewey order방식의 장점을 최대한 활용하여 하나의 SQL문장으로 저장 될 당시의 XML문서의 형태와 순서를 그대로 유지하면서 결과값을 출력한다. 우선 <EX1>을 예로 들자면 SQL문으로 변환하여 처리한 결과값으로 books.xml문서의 id와 last의 numbering값을 추출한다. 그리고 이 값을 아래의 XML_TRANSFORM 알고리즘을 적용하여 완전한 XML문서 형태로 출력한다.

XML_TRANSFORM(id, numbering)

1)하나의 SQL문으로출력할 last의 하위 엘리먼트들과 그들이 가진 값 들을 저장 될 당시의 순서를 그대로 유지하면서 한번에 로드한다.

```
with inline_view as
(select e1.docid, e1.numbering, e1.name, e1.value, e1.eid, e1.info, e1.pid
from how_element e1
where e1.numbering like '#1#1#3#' || '%'
and e1.id = 1)
select * from inline_view I
start with i.numbering = '#1#1#3#'
connect by prior i.eid = i.pid order siblings by i.numbering;
```

2)추출해야 할 어트리뷰트들을 로드한다.

```
select a1.aname, a1.eid, a1.value
from how_attribute a1, how_element e1
where e1.numbering like '#1#1#3#' || '%'
and e1.docid = 1
and a1.eid = e1.eid
and a1.id = e1.id;
```

3)한번의 질의로 얻어낸 결과를 바탕으로 엘리먼트와 어트리뷰트들 연결하고 '<'와 '>'를 삽입하는 간단한 작업을 거친 뒤 완전한 XML문서 형태로 사용자에게 반환한다.

위의 1)과정에서 알 수 있듯이 numbering값과 '%'연산자를 통해 last의 자손 엘리먼트들을 한꺼번에 로드하여 inline_view를 생성한다. 이에 start with~ connect by 절을 통해 부모와 자식레벨에 따라 그리고 같은 레벨일 경우 numbering값으로 XML고유의 순서 그대로 정렬한다.. 여기서 생성되는 inline_ view는 임시공간으로 쿼리문의 실행이 끝나면 자동소멸된다.

디스크와 메모리상에서 작업을 처리하는 속도를 고려한다면 XML_TRANSFORM알고리즘은 단 한번의 질의로 원하는 값을 정렬된 상태로 로드하기때문에 I/O시간이 줄어드므로 성능이 향상된다. 반면 Local order방식에서는 last의 모든 자손노드들을 찾을 때까지 질의하고 다시 조인하는 작업을 반복적으로 수행하므로 성능이 저하된다.

4. 성능평가 및 결론

본 논문에서 제안한 Dewey order기법을 이용한 XML 데이터베이스 시스템의 구현 환경은 팬티엄 4 1.2GHz, 메인 메모리 1GB, window2000운영체제이며 jdk 1.4.05와 jdbc를 이용하여 본 시스템을 구현하였다. 이 실험의 시간 측정은 오라클 9i에서 행해졌다. 우선 본 논문의 실험은 다음의 XPath식을 SQL문으로 변환한 것을 질의한 후 그 결과 값을 XML형태로 출력하기까지의 전체 시간을 Dewey order방식과 Local order방식으로 각각 나누어 Test 하였다. 이에 이용되는 문서는 shakespears의 약 193KB 크기인 as.xml문서이다

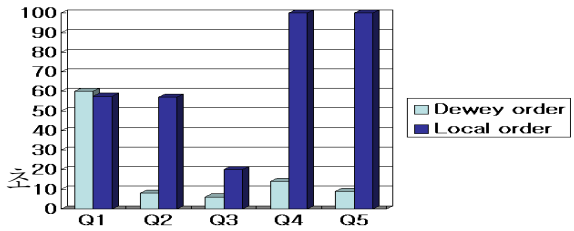
Q1: doc('as_you')//ACT/SCENE/SPEECH/LINE

Q2: doc('as_you')//ACT/SCENE[STAGEDIR = 'Exit']

Q3: doc('as_you')//ACT[TITLE = 'ACT I' and SCENE//SPEAKE = 'ORLANDO']

Q4: doc('as_you')//SCENE[TITLE = 'SCENE II. Lawn before the Duke's palace.']//SPEAKER

Q5: doc('as_you')//ACT[TITLE = 'ACT I']//SPEECH [SPEAKER='ORLANDO']



<질의 처리 후 XML형태의 결과 출력시간 비교>

위의 성능 평가를 통해 Dewey order방식이 Local order방식보다 성능이 훨씬 우수함을 알 수 있다. 즉 Predicate이 삽입된 질의일 경우 Dewey order기법으로 매겨진 numbering값을 통해 쉽게 결과값을 찾을 수 있다. 또한 XML문서로 반환하는 경우 각각 한번의 질의로 XML문서의 순서에 맞게 Data를 로드하므로 Local order방식보다 성능이 훨씬 우수하다. 하지만 미래에 XML문서의 삽입 삭제를 위한 구문을 사용하게 된다면 갱신이 발생된 위치부터 numbering값의 일부분이 변경되어야 하는 경우가 발생한다. 그러므로 앞으로의 향후 과제는 Dewey order 방식의 장점을 최대한 활용하되 문서의 변경이 일어날 경우를 대비한 연구가 이루어져야 할 것이다.

참고문헌

[1]Yoshikawa, M., Amagasa, T,. "XRel: path-based 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 Orderd XML Using a Relational Database System"2002

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

[4]"XML path Language (XPath) Version 1.0" http://w3c.org/TR/xpath 2005년 2월 검색