Inverted Indexes for XML Updates and Full-Text Retrievals in Relational Model

Yun-Woo Cheon†, Dong-Kweon Hong‡

ABSTRACT

Recently there has been some efforts to add XML full-text retrievals and XML updates into new standardization of XML queries. XML full-text retrievals plays an important role in XML query languages. Not like tables in relational model an XML document has complex and unstructured nature. We believe that when we try to get some information from unstructured XML documents a full-text retrieval query is much more convenient approach than a regular structured query. XML update is another core function that an XML query have to have. In this paper we propose an inverted index to support XML updates and XML full-text retrieval in relational environment. Performance comparisons exhibit that our approach maintains a comparable size of inverted indexes and it supports many full-text retrieval functions very well. It also shows very stable retrieval performance especially for large size of XML documents. Foremost our approach handles XML updates efficiently by removing cascading effects.

키워드: XML 변경(XML Update). XML 전문 검색(XML Full-Text Retrieval). 관계형 모델(Relational Model). 역 인덱스(Inverted Index)

1. 서 론

XML(Extensible Markup Language)은 인터넷상에서 더 자유로운 정보를 표현하고 저장하기 위한 표준으로 확장된 자료를 구성한다[1]. 기존의 표준 중심의 HTML(HyperText Markup Language)과는 달리 XML은 데이터의 의미를 표현한다는 점에서 프로그램을 사용하여 데이터를 분석할 수 있는 장점은 가지고 있다. 이는 인터넷상의 데이터를 다양한 형태의 프로그램으로 자동 처리할 수 있는 새로운 컴퓨팅 패러다임을 제공하는 의기적인 것으로 평가되고 있으며 이러한 변화를 바탕으로 현재 XML 데이터를 여러 환경 분야에서 활용하려는 노력이 활발히 이루어지고 있다. 그 예로 XML은 자동차 회사들이 모든 자동차에 대한 정보를 제공하는데 사용되고, 도표와 도서목록을 만들거나 전자상거래(E-Commerce) 웹 사이트의 다양한 제품을 소개하는데 사용되고 있다. 이와 같은 분야에서의 사용으로 인해 XML 데이터의 구조 정보 표현 기법 및 저장 시스템에 대한 연구가 필요하게 되었다. 그 결과 XML 인덱스 기술에 대한 표준도 제정되고 있다[2-6].

본 논문에서는 최근 W3C에서 제안하고 있는 XQuery의 전문 질의 기능을 알아보고, 그 기능을 기존의 관계형 기술을 활용하는 방식을 제안하여 기존을 개선하고, 연구 결과를 활용하여 관계형 모델을 활용한 전문 검색 기능의 구현 가능성을 대상으로 한다. 또한, 현재까지의 연구들은 거의 고려되지 않았던 XML 문서의 변경을 지원할 수 있는 기법에 대해서도 알아본다. 본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 연구들에 대해 언급하고 3장에서는 본 논문에서 제안한 인덱스 구조의 설계와 구현에 대해서 인지하고 4장에서는 구현된 시스템에 대한 성능 평가를 위한 모델을 제시한다. 5장에서는 제시된 성능 평가 모델을 바탕으로 실험 결과를 분석하고 마지막으로 6장에서는 결론을 제시한다.

2. 관련 연구


2.1 XML 점의

데이터를 그려폰 형식으로 보는 XML 모델과 데이터를 데이터의 형태로 보는 관계형 모델사이에는 많은 차이점이 있다. 따라서 관계형 모델의 질의어로 사용되는 SQL을 XML 데이터를 위한 질의어로 그대로 사용하는 것은 매우 어렵다. 이로 인해 새로운 질의어가 W3C에서 XQuery 표준화로 진행되고 있으며 XQuery 구문 형태는 XML 스키마, XSLT, XPath의 내용을 바탕으로 특히 IBM의 Quilt 연구 내용을 많이 수용하고 있다. XQuery는 초기에는 데이터 검색 부분만 강조했으나 점차 질문 검색의 기능도 고려하고 있으며 2003년 2월에 “XQuery and XPath Full-Text Requirement”과 “XQuery and XPath Full-Text Use Cases” 2건의 Working Draft가 발표되었다[9, 10]. 이 2가지의 문서는 향후 XML Query에 추가될 전문 검색(full-text retrieval) 기능에 대한 부분이 포함되어 있다.

2.2 관련 기법 분석


3. XML 변경과 전문 검색을 위한 역 인덱스 설계 및 구현

관계형 데이터베이스에 XML 문서를 저장하는 여러 기
법을 통해 XML 문서를 테이블의 특정 콘텐츠에 저장하고, 그 컬럼에 대한 역 인덱스를 생성하는 기법은 에러코드, 에
트리뷰트, 그리고 명작된 에러코드 사이에서의 제층 구조를
효과적으로 관리하여 정확한 값의 검색을 할 수 있다는 장
점이 가지고 있다. 또한, 전체 XML 문서가 저장된 특정 컬
럼을 통해서 XML 문서 전체를 결과로 쉽게 반환할 수 있
다. 이러한 이유로 본 논문에서는 이 기법을 사용해서 XML
문서를 저장함(그림 1)은 본 논문에서 제안하는 기법을 설명하기 위한 예제 XML 문서이다. 본 논문에서는 2장에서 제시된 관련 기법[11-13]들이 전형 고려하지 못했
던 XML 문서의 변경과 다양한 종류의 전문 검색 기능의
동시에 효과적으로 처리할 수 있는 역인덱스 기법인 XFTS
(XML Full-Text Search)를 제안한다.

(그림 1) 예제 books XML 문서

3.1 XFTS 역 인덱스 기법의 구성
본 논문에서 제안하는 역 인덱스 기법 XFTS는 다음의
표 1과 같이 4개의 테이블을 사용한다.

<table>
<thead>
<tr>
<th>표 1</th>
<th>XFTS 역 인덱스 기법을 위한 테이블</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>XML Documents (id, docname, isidx, contents)</td>
</tr>
<tr>
<td></td>
<td>Location (docid, pathid, path, depth, path_cnt)</td>
</tr>
<tr>
<td></td>
<td>Element (docid, eid, name, sbid, pathid, key_count, value)</td>
</tr>
<tr>
<td></td>
<td>Word (word, position, depth, docid, eid, pathid)</td>
</tr>
</tbody>
</table>

XFTS 역 인덱스 기법에서는 에트리뷰트에 대한 정보를
 저장하기 위한 테이블, 테이블에 대한 정보를 저장하는
테이블 Element와 유사하기 때문에 둘이도 생성하기로 한
다. 각각의 테이블에 대한 설명은 다음과 같다. 먼저 XML_
Documents 테이블은 실제 XML 문서를 저장하기 위한 것
으로 사용자가 XML 문서 전체를 결과로 반환하기를 원하
는 경우에 최대 4GB까지 저장할 수 있는 CLOB 타입의
contents 컬럼을 통해서 처리할 수 있다. 이 테이블의 id 컬
럼은 저장된 XML 문서의 고유한 식별자를 나타내며 doc-
name 컬럼은 저장된 XML 문서의 이름을 나타낸다. 그러
고 isidx 컬럼은 contents 컬럼에 저장되어 있는 XML 문서
에 대한 역 인덱스의 생성 여부를 나타낸다. isidx 컬럼의
값이 'true'인 XML 문서는 이미 역 인덱스가 생성되었다는
기반이므로 실제 역 인덱스를 생성해야 하는 문서는 isidx
컬럼의 값이 'false'로 되어 있어야 한다.

Location 테이블은 XML 문서 내에 존재하는 경로에 대
한 정보를 저장한다. 이 테이블의 docid 컬럼은 각 경로가
존재하는 XML 문서를 구별하기 위한 식별자이고 pathid 컬
럼은 각 경로들을 구별하기 위한 식별자를 나타내며, path 컬
럼은 두 엘리먼트에서부터 시작하여 최적 단단 엘리먼
트까지의 경로를 나타낸다. 또한 path 컬럼은 두 엘리먼
트에서부터 단단 엘리먼트까지의 길이를 나타내며 path_cnt 컬
럼은 XML 문서 내에 나타난 두 엘리먼트에서부터
단단 엘리먼트까지의 동일한 경로의 횟수를 표현한다. 본
논문에서는 path 컬럼의 값을 입력할 때 XPath에서 각 경
로를 구별하기 위해서 사용된 ‘/’ 문자가 대신에 점투자로 ‘’
문자를 입력한 ‘/’ 문자를 사용한다[14].

Element 테이블은 XML 문서 내에 존재하는 엘리먼트들
에 대한 세부 정보를 저장하며 sbid 컬럼은 XQuery의 기
능 중에서 '/author/family[2]'처럼 동일한 부모 엘리먼트의
자식들인 형제 엘리먼트 사이에서의 순서의 관련된 엘리먼
t을 처리한다. 형제를 찾는 XQuery를 처리하기 위해서는 '%/a-
uthor/family'인 경로를 Location 테이블의 path 컬럼에서
찾고 Element 테이블에서 sbid가 2인 엘리먼트를 찾으면
된다. value 컬럼은 각 엘리먼트의 전체 컨텐츠를 저장하며
key_count 컬럼은 문서여와 함께 동일한 키워드를 제외한
각 엘리먼트의 컨텐츠를 단수로 나타낸다. 이러한 컨텐츠
을 통해서 특정 경로 내에 특정 키워드나 구를 포함하는 엘리먼트를 검색하는 XQuery를 용용도 있게 처리할 수 있다.

Word 테이블은 XQuery 중에서 경로 정보가 주어지지
없는 단순한 키워드 검색을 위한 것이다. 이 테이블의 word
컬럼은 엘리먼트의 컨텐츠 중에서 동일한 키워드를 포함한
각 키워드를 나타내며 position 컬럼은 특정 엘리먼트 내에 존재
하는 각 키워드에 대한 상대적 위치 정보를 나타낸다. XQuery에서 연금하는 키워드간의 거리 연산(distance[12])
을 위하여 본 논문에서는 특정 엘리먼트 내에 존재하는 키
워드들 간의 거리 연산을 고려한다. 본 논문에서는 position
컬럼을 통해서 특정 엘리먼트에 있는 키워드들 간의 거리
연산을 처리할 수 있다. depth 컬럼은 각 경로의 구별자로
pathid가 동일한 Location 테이블의 depth 컬럼 값에 1을
증가시킬 것으로 각 키워드의 값이 나타난다.
(그림 2) (그림 3) (그림 4)는 각각 예제 books XML 문서를 구현하여 얻은 Location, Word, Element 역 인덱스의 내용을 나타내고 있다.

Location (docID, pathID, path, depth, path_cnt)
(1, 1, "book", 1, 1)
(1, 2, "books"/book, 2, 1)
(1, 3, "books"/book/title, 3, 1)
(1, 4, "books"/book/author, 3, 1)
(1, 5, "books"/book/author/family, 4, 3)
(1, 6, "books"/book/author/given, 4, 3)
(1, 7, "books"/book/summary, 3, 1)
(1, 8, "books"/book/summary/keyword, 4, 3)

(그림 2) books XML 문서의 Location 역 인덱스 내용

Word (word, position, depth, docID, eID, pathID)
(Data, 1, 4, 1, 3, 3) (GIL, 1, 5, 1, 10, 6)
(Web, 2, 4, 1, 3, 3) (Dong, 2, 5, 1, 10, 6)
(Kim, 1, 5, 1, 5, 5) (book, 1, 4, 1, 11, 7)
(Young, 1, 5, 1, 6, 6) (mentions, 2, 4, 1, 11, 7)
(Chul, 2, 5, 1, 16, 6) (semistructured, 1, 5, 1, 12, 8)
(Lee, 1, 5, 1, 7, 5) (data, 2, 5, 1, 12, 9)
(Eun, 1, 5, 1, 8, 6) (database, 1, 5, 1, 13, 8)
(Suk, 2, 5, 1, 8, 6) (XML, 1, 5, 1, 14, 8)
(Hong, 1, 5, 1, 9, 5)

(그림 3) books XML 문서의 Word 역 인덱스 내용

Element(docID, eID, name, sibord, pathID, key_count, value)
(1, 1, books, 1, 1, 0, null)
(1, 2, book, 1, 2, 0, null)
(1, 3, title, 1, 3, 2, Data on the Web)
(1, 4, author, 2, 4, 0, null)
(1, 5, family, 1, 5, 1, Kim)
(1, 6, given, 2, 6, 2, Young Chul)
(1, 7, family, 3, 5, 1, Lee)
(1, 8, given, 4, 6, 2, Eun Suk)
(1, 9, family, 5, 5, 1, Hong)
(1, 10, given, 6, 6, 2, Gil Dong)
(1, 11, summary, 3, 7, 2, This book mainly mentions)
(1, 12, keyword, 1, 8, 2, semistructured data)
(1, 13, keyword, 2, 8, 1, database)
(1, 14, keyword, 3, 8, 1, XML)

(그림 4) books XML 문서의 Element 역 인덱스 내용

다음의 예는 본 논문에서 제안하는 XFTS 역 인덱스 기법을 통한 XML 전문 검색의 대표적인 것들이다.

1) Word 테이블의 word 키를 통해서 단순 키워드 검색을 처리할 수 있다.

contains(text, 'XML')
SELECT E.DOCID, E.EID
FROM ELEMENT E, WORD W
WHERE W.WORD LIKE '%XML%'
AND W.DOCID = E.DOCID AND W.EID = E.EID ;

(그림 5) 단순 키워드를 검색하는 질의의 SQL으로의 변환

2) Element 테이블의 value와 key_count 키를 통해서 tight 중속 질의를 처리할 수 있다[12].

//keyword for 'semistructured data'
SELECT E.DOCID, E.EID
FROM LOCATION L, ELEMENT E
WHERE L.PATH LIKE '%/keyword'
AND E.VALUE LIKE '%semistructured data%' AND
E.KEY_COUNT = 2
AND L.DOCID = E.DOCID AND L.ID = E.PATHID ;

(그림 6) tight 중속 질의의 SQL으로의 변환

3) Word 테이블의 position 키를 통해서 엘리먼트 범위 내에서 키워드간의 거리 인산을 수행한다.

//book/title/Distance(Data, 'Web') <= 1
SELECT E.DOCID, E.EID, E.VALUE
FROM LOCATION L, ELEMENT E, WORD Data, WORD Web
WHERE L.PATH LIKE '%/book/title'
AND Data.WORD = 'Data' AND Web.WORD = 'Web'
AND L.DOCID = E.DOCID AND L.ID = E.PATHID
AND Data.DOCID = Web.DOCID AND Data.EID = Web.EID
AND Data.PATHID = LID AND Web.PATHID = LID
AND Web.POSITION - Data.POSITION > 0
AND Web.POSITION - Data.POSITION <= 1
AND E.DOCID = Data.DOCID AND E.EID = Data.EID
AND E.DOCID = Web.DOCID AND E.EID = Web.EID ;

(그림 7) 엘리먼트 내의 키워드간 거리 인산 질의의 SQL으로의 변환

3.2 XML 문서의 변경 처리
3.2.1 기존 엘리먼트 내의 키워드 변경 처리

키워드의 추가와 삭제는 유사하기 때문에 추가에 대한 절차만을 살펴본다. 키워드의 추가 시에는 추가되어야 할 엘리먼트에 대한 정보를 바탕으로 Element 테이블의 Word 테이블에서 관련된 품목을 변경한다. (그림 8)에서 에제 books XML 문서의 keyword 엘리먼트 중에서 키워드가 'database'인 것을 'database system'으로 변경하기 위한 절차를 보이고 있다.

```sql
1. Location 테이블에서 키워드를 찾아HAND path_id 를 얻는다. / SELECT DOCID, PATH_ID FROM LOCATION
   WHERE PATH = '/books/book/summary' AND KEYWORD = 'keyword';
2. docid, path_id를 바탕으로 elid를 얻는다. / SELECT ELID FROM ELEMENT
   WHERE NAME = 'keyword' AND VALUE = 'database' AND DOCID = L_DOCID
   AND PATHID = L_PATHID;
3. Element 테이블에 키워드를 추가하기 위한 변경을 한다. / UPDATE ELEMENT (KEY_COUNT, VALUE)
   SET (KEYCOUNT + 1, 'database system'
   WHERE DOCID = L_DOCID AND ELID = E_ELID AND PATHID = L_PATHID;
4. Word 테이블에 키워드를 추가하기 위한 정보를 얻는다. / SELECT POSITION, DEPTH INTO W_POSITION, W_DEPTH
   FROM WORD
   WHERE DOCID = L_DOCID AND ELID = E_ELID AND PATHID = L_PATHID;
5. Word 테이블에 키워드를 추가한다. / INSERT INTO WORD (SYSTEM, W_POSITION, W_DEPTH, L_DOCID, E_ELID, L_PATHID);
```

(그림 8) keyword 엘리먼트 내의 키워드 'system'을 추가하기 위한 절차

3.2.2 엘리먼트와 XML 문서의 업데이트 처리

XML 템플릿은 다른 XML 문서에 추가하는 경우를 살펴보자. 이러한 변경을 처리하기 위해서 Location 테이블의 path 테이블에서 추가될 엘리먼트까지의 경로가 한 번도 나타나지 않은 경우에는 추가될 엘리먼트의 경로 정보를 삽입하고 이 정보를 바탕으로 Element 테이블과 Word 테이블의 관련 품목들을 변경한다. 그러나 이미 추가될 엘리먼트까지의 경로가 존재할 경우에는 Location 테이블에서 삽입될 경로의 path_cnt 값을 1만큼 증가시키고 Element 테이블에서 삽입된 엘리먼트 이후의 엘리먼트에 대한 sibord 절차를 값을 1씩 증가시킨다. 그 다음으로 Element 테이블과 Word 테이블에서 관련 품목들을 변경한다.

(그림 9)는 추가될 엘리먼트까지의 경로가 이미 존재하는 경우의 예로 books XML 문서에 책의 저자로 'Cheor'이라는 키워드를 가진 family 엘리먼트를 추가하기 위한 절차를 보이고 있다.

```sql
1. 추가될 엘리먼트까지의 경로가 이미 존재하는지 검사한다. / SELECT PATH_CNT INTO L_PATH_CNT FROM LOCATION
   WHERE PATH = '/books/book/author';
2. 이미 존재(L_PATH_CNT > 0)하기 때문에 path_cnt를 1만큼 증가시킨다. / UPDATE LOCATION(L_PATH_CNT) SET PATH_CNT = L_PATH_CNT + 1
   WHERE PATH = '/books/book/author';
3. 긴 부모 엘리먼트를 가진 엘리먼트 중에서 삽입된 엘리먼트 이후의 엘리먼트에 대해 sibord 절차를 값을 1만큼 증가시킨다. / UPDATE ELEMENT(SIBORDER) SET SIBORDER = SIBORDER + 1
   WHERE PATH LIKE '/books/book/author/AND SIBORDER > 0';
4. 이후의 절차는 엘리먼트 내에 키워드를 추가하는 절차와 동일하다. /
```

(그림 9) family 엘리먼트를 추가하기 위한 절차

3.2.3 엘리먼트의 삭제 처리

본 논문에서는 엘리먼트의 삭제에 있어서 XML 문서가 변경될 경우에 삭제될 엘리먼트가 테스트를 제외한 자식 엘리먼트를 가지고 있다면 반드시 그 자식 엘리먼트도 함께 삭제되어야 한다고 가정한다. 삭제될 엘리먼트가 하위 자식 엘리먼트를 가지고 있지 않은 경우에는 삭제될 엘리먼트에 대한 역 인덱스 정보만을 삭제하면 된다. 그렇지 않으면 삭제된 엘리먼트가 자식 엘리먼트를 가지고 있을 경우에는 부모 엘리먼트를 삭제함으로써 발생되는 트리거를 통해 자식 엘리먼트에 대한 트리플을 삭제한다. 삭제될 엘리먼트까지의 경로가 처음 나타난 것인가에 따라 존재하는 경로를 알아보기 위해서는 Location 테이블의 path_cnt 칼럼을 사용한다. 이 칼럼의 값이 0이면 삭제될 엘리먼트까지의 경로가 이미 두 번 존재하는 것이기 때문에 path_cnt를 1만큼 감소시키고 삭제될 엘리먼트 이후에 나타나는 엘리먼트의 sibord 칼럼의 값을 1씩 증가시킨다. 그 다음 절차로 품목 기반 트리거를 통해가 Element 테이블, Word 테이블에 관련 품목들을 변경한다. 그러나 path_cnt가 1이면 삭제될 엘리먼트까지의 경로가 한 번 나타난 것이기 때문에 Location 테이블에서 관련 경로 정보를 가진 품목을 삭제하고 path_cnt가 2이상일 때의 절차가 동일하게 처리된다.

이처럼 본 논문에서 제안하는 기법은 XML 문서의 변경을 처리하기 위해서 변경된 엘리먼트가 키워드의 역 인덱스에 대해서 부분적인 변경이 필요하기 때문에 상당히 효율적이다.
4. 성능 평가 모델

본 장에서는 본 논문에서 제안하고 있는 XFTS 기법의 실험 환경을 기술한다. XML 전문 검색을 위한 역 인덱스
를 생성하기 위해서는 Java 1.4.2와 JDOM API를 사용하였 다. 실험에 사용한 상용 데이터베이스 시스템은 오라클9에
이고 이것은 리눅스 2.4.20, 코질 periods III 1.2GHz 듀얼 CPU,
메인 메모리 1.5GB에 설치되어 있다. 각 테이블의 인덱스
는 주키와 함께 조인이 발생되는 컬럼에 생성되었다. 클라
이언트는 윈도우 2000 Professional, 코질 periods IV 1.4GHz에
설치되어 있고 메인 메모리는 768MB이다. 본 장의 나머지
부분에서는 실험에 사용된 XML 문서와 XML 결의, 그리
고 성능 평가 척도에 대해서 설명한다.

4.1 실험 XML 문서
본 논문의 실험에서는 세 개의 XML 문서를 사용하였다.
첫 번째 문서는 세로스의 유명한 내용을 포함하고 있는
Shakespeare’s Work이고 두 번째 문서는 DBLP의 서지 목
록 내용을 담고 있는 DBLP이고 세 번째 문서는 실제 데이터
베이스가 아닌 가상의 메타데이터를 담고 있는 Auction이다
[16-18]. <표 2>는 실험에 사용된 문서의 태입과 개수, 전
체 크기, 그리고 각 문서의 평균 엘리먼트의 길이를 나타내
고 있다.

<table>
<thead>
<tr>
<th>문서명</th>
<th>태입</th>
<th>문서의 개수</th>
<th>전체 크기(kB)</th>
<th>평균 길이</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakespeare’s work</td>
<td>Real</td>
<td>37</td>
<td>753</td>
<td>59.5</td>
</tr>
<tr>
<td>DBLP</td>
<td>Real</td>
<td>1</td>
<td>50.3</td>
<td>4.00</td>
</tr>
<tr>
<td>Auction</td>
<td>Synthetic</td>
<td>1</td>
<td>1155</td>
<td>7.02</td>
</tr>
</tbody>
</table>

4.2 실험 XML 질의
본 논문에서는 XML 전문 검색을 위한 언어로서 XQuery
의 문법과 유사한 언어를 사용하였다. 전문 검색 기법 중에
서 이론 검색 언어와 오퍼 림 간 동일 연구에 있는 검
색은 제외하고 XQuery의 전문 검색 기법 중에서
XML 문서의 자료 구조 특성에 의하여 추구된 기법에 대하
서는 실험하였다. <표 3>은 실험에 사용된 XML 질의를
타입하고 있다.

<table>
<thead>
<tr>
<th>질의명</th>
<th>태입</th>
<th>문서의 개수</th>
<th>전체 크기(kB)</th>
<th>평균 길이</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains(text)</td>
<td>'XML'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/PLAY/TITLE/'The Comedy of Errors'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/PLAY/ACT//SCENE/SPEECH/SPEAKER/'DUKE SOLNUS'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'SPEAKER = &quot;ARBEON&quot;'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/TITLE/'ACT'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/ACT//SPEECH/SPEAKER='KING JOHN'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/SCENE/*//LINE/'love'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contains(text)</td>
<td>'SQL'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/DBLP/ARTICLE/TITLE/'SQL Reunion'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/AUTHOR/'Frank Manola'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/INPROCEEDINGS/TITLE/'Dynamite'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contains(text)</td>
<td>'pencil'</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/SITE/REGIONS/ASIA/ITEM/LOCATION/'Korea'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>name = 'preventions'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'/item/payment/'Personal Check'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3 성능 평가 척도
본 논문에서 제안하는 역 인덱스 기법의 성능을 평가하
기 위해서 다음과 같은 3가지의 성능 평가 척도를 사용하
였다.

(1) 결과의 정확성: 결과의 결과의 내용이 같은가를 비교하
었다. 종합 결과를 제거한 후 그 결과가 같은지 비교하
하였다.
(2) 역 인덱스 생성 성능 척도: XML 질의를 처리하기 위
해서 사용되어지는 역 인덱스의 공간 크기와 역 인덱
스를 만들기 위해서 사용되어진 생성 시간을 측정하
였다.
(3) 질의 처리 시간 복잡도 척도: 역 인덱스가 저장되어진
데이터에 접근하고 결과를 얻는 데까지 걸린 시간을 측
정하였다. 측정 시간의 오차를 줄이기 위해서 반복적
으로 5회 실험하였고 결과 시간은 5회의 평균으로 나
타내었다.

5. 실험 결과
본 장에서는 4장에서 열거한 성능 평가 척도를 바탕으로
본 논문에서 제안하는 XFTS 기법의 성능을 평가하였다. 또
한, XML 문서의 크기 변화와 결과 길이의 변화에 따른 질
의 처리 시간 복잡도의 성능도 함께 평가한다.

5.1 역 인덱스 생성 성능
표 4는 역 인덱스를 생성하는데 걸린 시간을 나타내고 있다. 여기서 C_INDEX 기법, XFTS_INDEX 기법, LINDEX 기법 순으로 생성 시간이 오래 걸린다는 것을 알 수 있다. LINDEX 기법은 XML 문서의 값이가 길이에 따라서 그 값만큼 높은 정보를 저장하기 때문에 역 인덱스를 생성하는데 시간이 오래 걸린다. C_INDEX 기법과 XFTS_INDEX 기법은 XML 문서의 값이가 상관없이 임정하게 증가하는 것을 볼 수 있다. C_INDEX 기법이 XFTS_INDEX 기법보다 시간이 적게 걸리는 이유는 질의를 처리하기 위해서 보다 적은 정보를 생성하기 때문이다.

<table>
<thead>
<tr>
<th></th>
<th>LINDEX 기법</th>
<th>C_INDEX 기법</th>
<th>XFTS_INDEX 기법</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakespeare's work</td>
<td>800.256</td>
<td>200.619</td>
<td>486.392</td>
</tr>
<tr>
<td>DBLP</td>
<td>4899.672</td>
<td>1322.540</td>
<td>3725.732</td>
</tr>
<tr>
<td>Auction</td>
<td>43387.022</td>
<td>1913.814</td>
<td>3673.301</td>
</tr>
</tbody>
</table>

표 5는 제안 기법별 역 인덱스의 크기(MB)이다.

<table>
<thead>
<tr>
<th></th>
<th>LINDEX 기법</th>
<th>C_INDEX 기법</th>
<th>XFTS_INDEX 기법</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakespeare's work</td>
<td>101</td>
<td>13</td>
<td>21</td>
</tr>
<tr>
<td>DBLP</td>
<td>429</td>
<td>78</td>
<td>110</td>
</tr>
<tr>
<td>Auction</td>
<td>3000</td>
<td>243</td>
<td>373</td>
</tr>
</tbody>
</table>

표 5에서 각각의 XML 문서에 대해서 생성되어진 역 인덱스의 크기를 나타내고 있다. 이 표를 통해서 LINDEX, XFTS_INDEX, C_INDEX 기법의 순서로 작은 성능을 보인다는 것을 알 수 있다. 이런 성능 차이의 원인은 LINDEX 기법은 같은 엘리먼트에 대해서 중복된 Word_Type 역 인덱스를 생성하기 때문이다. 최근의 경우에 런트 엘리먼트에 대해서 단일 엘리먼트까지의 값이가 길이의 경우에는 단일 엘리먼트에 포함된 어워드의 패턴을 역 인덱스에 저장하기 위해서 Word_Type 역 인덱스를 생성하게 된다. 그리고 C_INDEX 기법이 XFTS_INDEX 기법보다 약 25%의 성능 향상을 보이며 그 이유는 C_INDEX에서는 포함된 연산을 위한 계층 정보로(begin, end) 숫자만 사용하여 계층 정보를 위한 역 인덱스를 따로 만들어 주지 않기 때문이다. 반면에 XFTS_INDEX는 계층 정보를 무작위 단말까지의 패스 트리임을 따로 저장한다. 하지만 C_INDEX 기법은 XML 문서가 변경되었을 때 변경된 부분 다음에 있는 모든 원소의 (begin, end)를 다시 부여해야 하는 어려움이 있지만 XFTS_INDEX 기법은 C_INDEX 기법에 비해 추가적으로 생성된 Location 데비오에 새로운 패스 트리를 추가하지 아니면 기존에 존재하는 패스 트림의 path_cnt 컬럼 값을 증가시킴으로써 효과적으로 처리할 수 있다. 이처럼 C_INDEX 기법은 공간 복잡도에 있어서는 XFTS_INDEX 기법보다 좋은 성능을 보임으로서 XML 문서의 변경을 처리하기 위하여 각 원소의 (begin, end) 값을 새로 부여해야 하므로 매우 비효율적이다.

5.2 질의 처리 시간 복잡도
질의 처리 시간 성능 비교는 XFTS_INDEX 기법과 기존 기법인 LINDEX 기법, C_INDEX 기법을 각각 따로 실험하였다. 이렇게 비교한 이유는 LINDEX 기법을 통한 역 인덱스 생성 시에 문서의 크기가 115MB인 Auction 문서의 경우에 다른 기법들에 비해서 역 인덱스의 크기가 훨씬 커서 Auction 문서에 대한 질의 처리 시간을 측정할 수 없었기 때문이었다. (그림 10)은 LINDEX 기법과 XFTS_INDEX 기법의 질의 처리 시간 성능을 비교한 것이다. (그림 10)에서 알 수 있듯이 Q5와 Q8 질의를 제외하고 본 논문에서 제안하는 XFTS_INDEX 기법이 더 좋은 성능을 보인다. Q1과 Q10은 같은 종류의 질의로서 LINDEX 기법과 XFTS_INDEX 기법에서 상당한 성능 차이를 보임으로써 그 이유는 LINDEX 기법에서는 특정 단어가 존재하는 엘리먼트를 검색할 때 있어서 XML 문서에 대한 복잡한 그리고 단일 엘리먼트의 고유한 반호를 검색하기 위해서 'IN' 연산을 수행해야하기 때문이다. 그에 비해 XFTS_INDEX 기법은 'IN' 연산을 통해 수행하기 때문에 실행 시간을 단축할 수 있다. Q1과 Q10을 비교해 보면 실현 XML 문서의 크기가 커지기 때문에 그를 실행 시간이 길어질 것을 알 수 있다. 나머지 Q3, Q5, Q12, Q14도 이와 같은 이유로 성능 차이를 보인다. 그러나 Q4와 Q9는 tight 종속 질의로 LINDEX 기법은 XML 문서에 나타나는 각 엘리먼트에 대한 정보를 EleName 역 인덱스로 생성한다. 그래서 한 번의 'IN' 연산을 통해서 빠르게 검색할 수 있다. 그에 비해 XFTS_INDEX 기법은 정보에 대한 정보를 검색하는데 'LIKE' 연산을 한 번 수행해야하기 때문에 실행 시간이 더 길어 진 것이다. (그림 11)은 C_INDEX 기법과 XFTS_INDEX 기법의 질의 처리 시간 성능을 비교한 것이다. (그림 11)에서 볼 수 있듯이 XFTS_INDEX 기법이 C_INDEX 기법보다 Q1, Q11, Q12, Q14에서 훨씬 좋은 성능을 보인다. 또한 나머지 질의에서도 같은 경향을 보인다. 이러한 결과의 주된 원인은
인은 다음과 같이, QD1와 QA1 질의에서는 조인을 할 때 XFTS_INDEX 기법은 'n' 연산을 통해 이루어지지만 C_INDEX 기법은 수 비교 연산을 통해 이루어진다. 같은 종류의 질의인 QD1과 QA1에서 실행 XML 문서의 크기 변화가 실행 시간에 상당한 영향을 미친다. QD2, QD4, QA2, QA4 질의에서도 역시 실행 시간에 문서의 크기가 영향을 미쳤음을 알 수 있다. 특히, QA2와 QA4 질의에 대한 C_INDEX 기법은 문서의 크기 변화와 함께 복잡한 경로에 대한 질의 처리를 위해서 여러 빈의 조인이 발생한다. 그러나 XFTS_INDEX 기법은 Location 데이터에서 'n' 연산을 통해 복잡한 경로를 한정해 쉽게 찾을 수 있기 때문에 좋은 성능을 보인다.

기관 동영

(그림 10) I_INDEX기법과 XFTS_INDEX 기법의 시간 복잡도 성능 비교

C_INDEX와 XFTS_INDEX 시간 복잡도 비교

5.3 XML 문서의 크기 변화와 경로 길이에 따른 시간 복잡도 성능

(그림 12)와 (그림 13)은 XML 문서의 크기 변화와 사용자 취향의 경로 길이에 따른 시간 복잡도를 분석한 것이다. (그림 12)와 (그림 13)의 X축에 사용된 질의 종류에서 가운데 문서는 XML 문서를 나타내고 마지막 숫자는 경로 길이를 나타낸다. 성능 비교는 C_INDEX 기법과 XFTS_INDEX 기법을 사용한다. 실험에 사용된 XQuery는 반복 및 결과의 수를 고려해서 사용되었다.

(그림 12)와 (그림 13)에서 볼 수 있듯이 C_INDEX 기법은 경로 길이가 길어질수록 처리 시간도 길어졌다. 그러나 본 논문에서 제안하는 XFTS_INDEX 기법은 경로 길이가 길어지는 것과는 상관없이 일정한 처리 시간이 걸렸다. 여기에 (그림 12)와 XFTS_INDEX 기법의 경우 결과가 가정치 중간값 이상의 결과 튜플의 수가 다른 질의에 비해 적게 남아서 결과를 가져오는데 오랜 시간이 걸렸기 때문이다. (그림 13)에서의 C_INDEX 기법은 처리 시간에 있어서 (그림 12)에서의 처리 시간보다 더 빠른 처리를 보였다. 이 이유는 실험에서 사용된 XML 문서의 크기가 상당히 커질 때 때문이다. (그림 13)에서의 C_INDEX 기법은 XQuery의 경로 길이가 길어짐에 따라 처리 시간도 상당히 증가하였다. 그렇지만 XFTS_INDEX 기법은 경로 길이와는 상관없이 일정한 처리 시간을 보였다. 종합 해보면 C_INDEX 기법은 XML 문서의 크기가 커지면 경로 길이가 길어질수록 XQuery를 처리하는 데 오랜 시간이 걸린다. 그러나 본 논문에서 제안하는 XFTS_INDEX 기법은 문서의 크기와 경로 길이와는 상관없이 항상 일정한 처리 시간을 유지한다.
5.4 XML 변경 성능

INDEX와 C_INDEX는 XML 변경을 전처리 고려하지 않고 만들어진 인덱스에므로 직접적인 성능을 비교하는 것은 매우 어렵다. 하지만 XISTS 범위는 기존의 INDEX에서 사용하는 번호 부여 방식(Numbering scheme)과는 달리 XML 변경 연산이 발생할 때 삽입 또는 삭제가 이루어지는 부분 주변에는 전혀 영향을 주지 않는다. 따라서 XML 문서가 발생할 경우 변경되어야 할 인덱스가 최소화 되어 XML 문서 변경이 쉽게 이루어질 수 있다.

6. 결론

본 논문에서는 XML 데이터를 저장하기 위한 기법으로 지금까지 구축하게 발전해온 관계형 데이터베이스를 사용하여 XML 문서의 변경시 중 인덱스의 재생성 없이 부분적인 변경만으로 XML 문서의 변경 내용을 반영하면서 다양한 기능의 XML 전문 검색 엔진의 효율적으로 처리 할 수 있는 XML 인덱스 기법을 제안하였다. 본 논문에서 제안한 XISTS_INDEX 기법은 기존에 연구되었던 INDEX 기법과 C_INDEX 기법에 비해 항상된 점은 다음과 같다. 첫째, 검색 시간이 단축되었다. 둘째, 옵션 분야에서 XML 데이터를 활용하기 위해서는 데이터의 분할 및 재구성에 XML 데이터의 크기도 상당히 커지고 있는 추세이다. 이러한 상황에서 XML 데이터에 대한 검색 성능 평가를 최적화 하는 중요한 요소는 검색에 걸린 시간이다. 둘째, XML 데이터의 변경을 효율적으로 처리 할 수 있다. 현재 XML 데이터는 주로 디지털 정보를 포함하고 교환하기 위한 수단으로 사용되고 있다. XML 데이터 사용의 보편화에는 XML 형식의 데이터는 언제든지 확장 또는 변경 될 수 있다는 장점이 큰 역할을 했다. 이러한 점에서 XML 데이터는 변경이 자주 일어날 수 있다. 이러한 변경은 본 논문에서 제안한 기법이 쉽게 처리한다는 사실은 상당한 성과라고 할 수 있다.

향후 연구 과제로는 성능의 향상 위해서 원래의 XML 데이터보다 상당히 큰 역 인덱스 저장 공간이 필요하기 때문에 현재의 XML 검색 기능을 그대로 유지 또는 발전시키면서 상대적으로 역 인덱스의 크기를 줄일 수 있는 압축 기법의 연구가 필요하다.

참고 문헌

천윤우
E-mail: ywchoon@idatabank.com
2002년 계명대학교 컴퓨터공학과(학사)
2004년 계명대학교 컴퓨터공학과(공학 석사)
2004년-현재 (주)에이터뱅크 시스템즈
관심분야: RDBMS, XQuery

홍동권
E-mail: dихong@kmuc.leeimywng.ac.kr
1985년 경북대학교 전자공학과(학사)
1992년 University of Florida 전자계산학과(석사)
1995년 University of Florida 전자계산학과(박사)
1985년-1990년 한국전자동통신연구원
1996년-1997년 한국전자동통신연구원
1997년-현재 계명대학교 공학부 컴퓨터공학 전공 교수
관심분야: 능동 실시간 데이터베이스, 병렬 처리, 성능 평가, 시뮬레이션, 멀티미디어 처리