경로정보의 중복을 제거한 XML 문서의 저장 및 질의처리 기법

이 혜 자[†] · 정 병 수[††] · 김 대 호[†††] · 이 영 구[††]

요 약

본 논문에서는 대용량 XML 문서를 저장하고 그로부터 원하는 정보를 효율적으로 찾기 위한 방법으로, 경로정보의 중복을 제거하면서 역 인덱스를 함께 이용한 방법을 제안한다. XML 문서는 트리구조에 기반한 노드로 분해되어, 노드 타입에 따라, 루트에서 각 노드까지의 경로정보와 함께 관계형 테이블에 저장된다. 경로정보를 이용한 기존의 XML 질의 기법들에서는 모든 엘리먼트 노드들에 대해 경로정보를 저장함에 따라 정보의 양이 증가하여 질의 처리의 성능을 저하시키는 요인이 되고 있다. 제안 방법에서는 경로정보 중 가장 긴 단말 엘리먼트 노드까지의 경로인 단말 엘리먼트 경로(leaf element path)만 저장하고 내부 엘리먼트 노드까지의 경로인 내부 엘리먼트 경로들(internal element paths)은 저장하지 않는다. 단말 엘리먼트 경로만을 대상으로 하여 역 인덱스를 구성함에 따라, 기존의 역 인덱스 이용 기법에 비해 키워드별 포스팅 리스트(posting lists)의 수를 줄이게 된다. 제안 방법에서는 XML 문서의 저장과 질의를 위하여 XML 문서에 대한 스키마 정보가 없어도 되며, 관계형 데이터베이스의 어떤 확장도 요구하지 않는다. 실험을 통해 제안 방법은 실험 범위 내에서 기존 기법들에 비해 좋은 성능을 보인다.

키워드 : XML 문서 질의 처리, 경로정보, 단말 엘리먼트 경로, 내부 엘리먼트 경로, 역 인덱스

Storage and Retrieval of XML Documents Without Redundant Path Information

Hiye-Ja Lee[†] · Byeong-Soo Jeong[††] · Dae-Ho Kim[†††] · Young-Koo Lee[††]

ABSTRACT

This paper proposes an approach that removes the redundancy of path information and uses an inverted index, as an efficient way to store a large volume of XML documents and to retrieve wanted information from there. An XML document is decomposed into nodes based on its tree structure, and stored in relational tables according to the node type, with path information from the root to each node. The existing methods using path information store data for all element paths, which cause retrieval performance to be decreased with increased data volume. Our approach stores only data for leaf element path excluding internal element paths. As the inverted index is made by the leaf element path only, the number of posting lists by key words become smaller than those of the existing methods. For the storage and retrieval of XML data, our approach doesn't require the XML schema information of XML documents and any extension of relational database. We demonstrate the better performance of our approach than the existing approaches within the scope of our experiment.

Key Words : Retrieval of XML Documents, Path Information, Leaf Element Path, Internal Element Path, Inverted Index

1. 서 론

XML(eXtensible Markup Language)은 인터넷상의 데이터 표현과 교환에 있어서 사실상의 표준이 되고 있다. 이와 함께 XML문서를 저장하고 질의하는 것과 같은 XML관련 데이터 관리 요구사항들이 생기고 있다. 대량의 XML 문서를 저장하고 그로부터 원하는 정보를 효율적으로 찾는 기법을 개발하는 것은 데이터베이스와 XML과 관련된 주요 연구과제 중의 하나이다. 이러한 문제들을 해결하기 위하여 그동안 많은 연구자들은 관계형 데이터베이스를 사용하여 XML 문서를 관계들로 분해하고, XML 질의를 그 관계들에 대한 SQL 질의로 번역하는 방법들을 제안해 왔다[1, 2].

XML 문서의 주요 특징 중의 하나는 문서의 논리적 구조에 기초를 두고 연산을 할 수 있다는 것이다. 이 특징에 기초하여, XML 문서를 다루는 데이터베이스는 문서의 논리적 구조와 콘텐츠에 대한 질의를 지원해야 한다. 논리적 단위

※ 이 논문은 2004년도 경희대학교 지원에 의한 연구결과임.
† 정 회 원 : 용인송담대학 의료정보시스템과 교수
†† 종신회원 : 경희대학교 전자정보학부 컴퓨터공학과 교수
††† 준 회 원 : 경희대학교 전자정보학부 컴퓨터공학과 박사
논문접수 : 2005년 4월 14일, 심사완료 : 2005년 9월 1일

와 재이용성을 고려하면, XML 문서를 문서의 트리구조에 따라 분해하여 저장하는 것이 적절하다[2, 3].

하나의 고정된 데이터베이스 스키마가 모든 XML 문서의 구조를 저장하는 모델 매핑 방법(Model-mapping approach)에 따라 관계형 스키마를 설계할 경우의 주된 문제들 중의 하나는 트리모델의 기본적인 구성체를 어떻게 (객체)관계형 스키마로 대응시키는가에 관한 것이다. 이에 대한 기존의 방법들로는, 에지(edge) 저장하기[4], 노드 저장하기[5], 경로와 영역(region)의 조합으로 표현하는 XRel[2], 경로와 노드 식별자로 표현하는 XParent[6] 등이 있다.

일반적으로 XML 문서에 대한 질의는 경로 표현식(path expressions)을 포함하고 있다. XRel 이전의 방법들은 경로 표현식을 처리하기 위하여, 경로 표현식의 길이에 비례하여 조인 연산을 요구한다. 그러나 XRel에서는 모든 가능한 경로 표현식들이 데이터베이스에 스트링으로 저장되어 있어서, 이들 질의를 스트링 매칭으로 처리할 수 있다[2]. 더 최근의 연구에서는 스트링 매칭의 횟수를 줄이기 위해 경로정보에 대한 인덱스를 구성하는 방법들을 제안하고 있다[7-12]. 이 방법들은 특히 대용량의 XML 문서에서 필요한 정보를 검색할 때 효율적이다.

그러나 경로정보를 이용하고 있는 기존의 XML 질의 기법들에서는 루트부터 각 노드까지의 모든 경로에 대해 정보를 저장하고 있다. 이는 정보의 중복을 초래하여 경로 정보의 양을 증가시키게 되며, 따라서 경로 정보를 이용하는 XML 질의 처리의 성능을 저하시키는 요인이 된다.

본 논문에서는 정보의 중복을 제거한 간결한 경로정보와 역 인덱스를 이용하여 XML 질의를 효율적으로 처리하는 새로운 방법에 대하여 서술한다. 제안 방법에서는 저장되는 XML 문서에 대하여 어떤 제약도 부과하지 않으며, XML 문서의 저장 및 질의는 현재 가용한 관계형 데이터베이스를 이용하여 가능해야 하고, 질의 처리는 효율적이어야 한다는 데에 설계 목표를 두고 있다.

우리의 제안 방법과 기존 연구와의 차이점은 기존 방법들에서 성능 저하의 요인이 되는 경로정보의 중복성을 제거하여 질의 처리의 성능을 향상시킨다는 것이다. 경로정보 중에서 내부 엘리먼트 노드까지의 경로는 저장하지 않고, 전체경로를 나타내는 단말 엘리먼트 노드까지의 경로만을 저장한다. 경로정보에 대한 역 인덱스는 단말 엘리먼트 경로만을 대상으로 하여 구성함으로써 기존의 역 인덱스 이용 기법에 비해 키워드별 포스팅 리스트의 수를 줄이게 되고 따라서 질의 처리의 성능도 향상시키게 된다.

본 논문의 기여점은 다음과 같이 요약할 수 있다.

(1) XML 문서의 구조적 정보를 관계형 모델을 이용하여 표현하는 방안을 제안한다.
(2) 중복을 제거한 경로정보의 역 인덱스를 이용한 XML 데이터 질의 처리 알고리즘을 제공한다.
(3) 실제 XML 문서와 가상 XML 문서를 이용한 실험을 통하여 제안 방법의 성능을 입증한다.

본 논문에서는 루트부터 각 노드까지의 모든 경로들 중에서 가장 긴 경로인 단말 엘리먼트 노드까지의 경로를 **단말 엘리먼트 경로(leaf element path : LEP)**라고 정의하며, 모든 경로 중에서 단말 엘리먼트 경로를 제외한 내부 엘리먼트 노드까지의 경로를 **내부 엘리먼트 경로(internal element path : IEP)**라고 정의한다.

2. 관련 연구

XML 문서 저장을 위한 접근방식은 XML 문서를 텍스트 파일 그대로 저장하는 텍스트 기반의 접근방식, 관계형 데이터베이스를 기반으로 한 접근방식, 그리고 각 XML 엘리먼트를 분리된 객체로 저장하는 객체지향 접근방식으로 구분할 수 있다. 텍스트 기반 접근방식은 XML 문서를 저장하는 데 별 노력이 들지 않지만 질의를 위해 특별히 지원하는 것은 없다. Lore[13], Tamino[14] 등과 같은 객체지향 접근방식의 데이터베이스 시스템은 아직 대규모 데이터베이스에 대한 복잡한 질의를 처리할 만큼 성숙하지 않았다. 그에 반해 관계형 데이터베이스를 기반으로 한 접근방식은 실무에서 많이 이용하고 있는 데이터베이스 시스템에 적용할 수 있는 등 여러 가지 장점이 있어서 가장 활발히 연구되고 있는 분야이다[4, 15].

일반적으로 XML 문서에 대하여 데이터베이스 스키마를 설계하는 방법은 구조 매핑 방법(Structure-mapping approach)과 모델 매핑 방법(Model-mapping approach)으로 분류된다[2]. 구조 매핑 방법에서는 XML 문서의 각 엘리먼트 타입마다 관계나 클래스를 생성하거나 DTD(Document Type Definition)에 대한 상세분석을 기초로 하여 데이터베이스 스키마를 설계한다[16, 17]. 모델 매핑 방법은 DTD에 관한 정보 없이 하나의 고정된 데이터베이스 스키마가 모든 XML 문서의 구조를 저장하는 데 사용되는 방법으로, XML 문서 트리의 에지를 관계형 튜플로 저장하는 에지 접근방식[4], 트리구조를 경로와 노드의 영역 또는 식별자의 조합으로 표현하는 경로기반 접근방식[2, 6] 등이 여기에 속한다.

구조 매핑 방법은 한정된 숫자의 문서구조나 DTD를 갖는 대량의 XML 문서를 저장할 경우와 문서 구조나 DTD가 비교적 정적일 때에 적합하다. 그러나 수많은 정교한 웹 애플리케이션들은 융통성 있고 동적인 XML 사용을 전제로 하고 있다. 즉, 여러 종류의 XML 문서를 저장할 수 있어야 하고 논리적 구조가 자주 바뀌는 XML 문서를 다룰 수 있어야 한다. 구조 매핑 방법은 이렇듯 동적이고 구조적으로 변하는 대량의 XML 문서를 저장하기에는 적합하지 않다.

XML 질의를 위해 경로 표현식을 처리하는 기법들은 크게 2가지, 스키마-레벨 방법과 인스턴스-레벨 방법으로 구분할 수 있다[8]. 스키마-레벨 방법은 경로표현식과 매치하는 노드들을 찾기 위해 레이블 경로와 같은 구조적인 정보를 사용하는 방법을 가리키며, 그 중 경로들을 저장하기 위해 관계형 테이블을 사용하는 방법으로는 XRel[2], Xparent[6], XIR-Linear[8] 등이 있으며, 특별한 목적의 인덱스를 사용하는 방법으로는 Index Fabric[18], APEX[19] 등이 대표적이다. 인스턴스-레벨 방법은 노드 구분 정보만을 사용하

는 방법을 가리키며, Element Numbering Scheme[9] 등이 여기에 속한다.

본 연구에서는 저장되는 XML 문서에 대하여 어떤 제약도 부과하지 않으며, XML 문서의 저장 및 질의는 현재 가용한 관계형 데이터베이스를 이용하여 가능해야 하고, 질의처리는 효율적이어야 한다는 데에 설계 목표를 두고 있다. 따라서 제안 방법에서는 XML 문서 저장 방법 중 구조적으로 변하는 대량의 XML 문서를 저장하기에 적합한 모델 매핑 방법을 이용하며 관계형 테이블을 사용하는 스키마-레벨 방법에 기반을 두면서 역 인덱스 기술을 이용하고 있다. 관계형 테이블을 사용하는 스키마 레벨 방법 중 제안 방법의 기초가 되는 XRel과 XIR-Linear에 대하여 좀 더 살펴본다.

2.1 XRel

XRel에서는 루트로부터 각 노드까지의 경로들을 열거하고, 경로 표현식을 관계형 테이블의 한 속성으로 저장하며, 이들 정보와 영역정보를 조합하여 트리구조를 표현한다. 모든 가능한 경로 표현식들이 데이터베이스에 스트링으로 저장되어 있어서, 스트링 매칭으로 질의를 처리할 수 있다. XRel에서 XML 트리 정보를 저장하기 위한 데이터베이스 스키마는 다음과 같다.

```
Element (docID, pathID, start, end, index, reindex)
Attribute (docID, pathID, start, end, value)
Text (docID, pathID, start, end, value)
Path (pathID, pathexp)
```

다음과 같은 질의 예제 1(Q1)을 XRel에서 처리하려면 아래와 같은 SQL 문장이 필요하다. XRel에서는 스트링 매칭를 이용하여 동일한 처리를 하므로, 조인 연산의 횟수를 줄여 질의 처리를 효율적으로 할 수 있다.

예제 1 (Q1) : /PLAY/ACT/EPILOGUE/SPEECH

```
SELECT   e1.docID, e1.start, e1.end
FROM     Element e1, Path p1
WHERE    p1.pathexp='#/PLAY#/ACT#/EPILOGUE#/SPEECH%'
AND      e1.pathID=p1.pathID;
```

XRel에서 start, end 속성은 하나의 XML 문서 내에서 각 노드의 순서를 유지하기 위해, index, reindex 속성은 형제 노드들 사이의 순서를 유지하기 위해 필요하다. 그러나 제안 방법에서는 노드 순서와 형제노드들 사이의 순서를 유지하고 있는 노드 ID를 이용하므로 start, end, index, reindex 속성은 필요하지 않다.

2.2 XIR-Linear

XIR-Linear 방법은 대규모의 이질적인 XML 문서들에 대한 부분 매치 질의를 효과적으로 처리하기 위한 방법으로, 관계형 테이블을 이용한 스키마 레벨 방법에 기반을 두고 역 인덱스 기술을 이용하여 질의 처리의 효율성을 향상시켰다. 레이블 경로를 텍스트로 간주하고, 레이블 경로 내의 레이블들을 텍스트 내에 있는 키워드로 간주한 후, 정보검색 기술을 이용하여 레이블들을 인덱싱함으로써, 기존의 스트링 매치보다 효율적인 방법으로 레이블 경로들을 찾는다.

XIR-Linear 방법에서 경로정보에 대한 역 인덱스 구성을 위한 저장 구조는 다음과 같다.

```
LabelPath (pid, labelpath)
LabelPath Inverted Index (keyword, posting list)
* posting list =
      {pid, occurrence_count, offsets, label_path_length}
 pid : 레이블이 존재하는 레이블 경로의 식별자
 occurrence_count : 레이블 경로 내에서 그 레이블이 나타난 개수
 offsets : 레이블의 시작부터 해당 레이블들까지의 거리들의 집합
 label_path_length : 레이블 경로 내에 존재하는 레이블들의 개수
```

레이블 경로 매치 시간에 영향을 미치는 요소가 XRel에서는 경로 테이블 내에 존재하는 경로의 개수인 반면, XIR-Linear 방법에서는 경로 표현식에 주어진 레이블들의 개수와 이들 레이블들과 연관된 포스팅 리스트들의 길이의 합이 된다. XIR-Linear 방법에서는 기존 방법의 경로 표현식 처리 부분을 역 인덱스 검색으로 대치하므로, XML 문서의 양이 증가할수록 비용이 많이 드는 기존 방법의 스트링 매치를 사용하지 않아도 되어 효율적으로 질의를 처리할 수 있다.

본 논문에서 제안하는 방법이 기존 방법들과 다른 점은 루트에서 각 노드까지의 모든 경로를 관계형 테이블에 저장하지 않고, 루트부터 가장 긴 단말 엘리먼트 노드까지의 경로만을 저장하여 경로의 수를 줄이고, 경로 레이블에 대한 역 인덱스를 만들 때도 단말 엘리먼트 경로만을 대상으로 함으로써 키워드별 포스팅 리스트의 수를 줄인다는 점이다.

3. XML 문서의 데이터 모델

본 논문에서는 XML 문서를 표현하기 위하여 Tatarinov 등[1]이 제안한 순서화된(ordered) XML 데이터 모델 중 듀이 순서 코딩(Dewey order encoding) 방법을 이용한 데이터 모델에 기반을 두고 있다. XML 문서는 트리로 볼 수 있다. 단말노드는 자료 값 즉, 텍스트에 해당되고, 내부노드는 XML 엘리먼트에 해당된다.

순서는 XML 데이터 모델의 중요한 특징이다. XML 문서 트리는 XML 문서의 엘리먼트들의 순서에 따라 내부적으로는 순서가 정해진다. XML 문서 트리는 엘리먼트 노드와 텍스트 노드 외에 애트리뷰트 노드를 포함할 수 있는데, 애트리뷰트 노드는 서브엘리먼트가 없는 엘리먼트 노드와 비슷하다. 문서의 내부적 순서는 XML 문서가 조각들로 나누어진 형태로 저장되어 있을 경우, 선택 후 추출하는 재구성 모드의 질의에서 아주 중요하다[1].

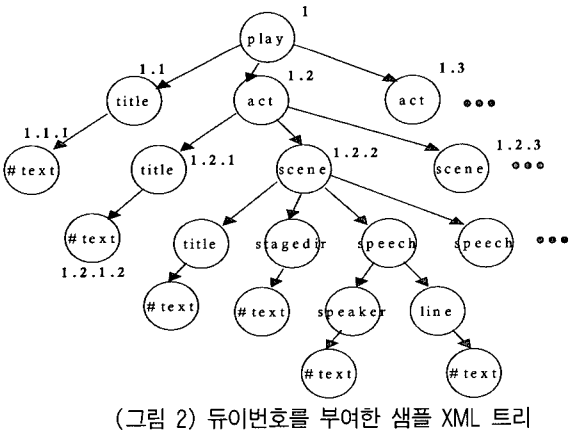
XML 문서의 각 노드의 순서를 코드화하는 방법으로는 듀이 순서 코딩 방법 외에 전역 순서 코딩(Global order encoding), 지역 순서 코딩(Local order encoding), 같은 이름의 형제노드 순서 코딩(Same sibling order encoding) 방

법이 제안되고 있다. 전역 순서 코딩은 질의 위주 작업에 최적이고, 지역 순서 코딩은 갱신 위주 작업에 최적이며, 듀이 순서 코딩은 질의와 갱신이 혼합된 작업에 최적이다. 같은 이름의 형제노드들 사이의 순서 코딩은 같은 이름을 가진 형제노드에 대해 부분적으로 순서를 코드화하는 것으로, 다른 3가지 방법들에서 보완적으로 사용될 수 있다[1].

본 연구에서 XML 문서의 각 노드의 순서를 코드화하는 방법은 질의와 갱신이 혼합된 작업에 최적인 듀이 순서 코딩 방법을 기본으로 하되, 구현의 편의를 위해 부분적으로 수정하였다. 각 노드에 대한 듀이 순서는 XML 문서 내에서의 순서와 조상 노드에 대한 정보를 나타내다. 그러나 일반적으로 많이 이용하는 질의의 형태 중에는 같은 이름의 형제노드들 사이의 순서를 알고 있으면 쉽게 찾을 수 있는 경우가 있다. 이러한 배경에서 본 연구에서는 듀이 순서 코딩을 보완하여 같은 이름의 형제노드들 간의 순서도 함께 나타낼 수 있게 수정하였으며, 구현의 편의를 위해 노드 ID를 구성하는 각 자리수의 자리 크기도 같게 수정하였다. 수정

```
<play>
  <title>Hamlet</title>
  <act>
    <title>ACT I</title>
    <scene>
      <title>SCENE I. Elsinore. A platform ...</title>
      <stagedir>FRANCISCO at his post ...</stagedir>
      <speech>
        <speaker>BERNARDO</speaker>
        <line>Who's there?</line>
      </speech>
      <speech>
        <speaker>FRANCISCO</speaker>
        <line>Nay, answer me: stand, and ...</line>
      </speech>
      ...
    </scene>
    ...
  </act>
  <act>
    ...
  </act>
</play>
```

(그림 1) 샘플 XML 문서



(그림 2) 듀이번호를 부여한 샘플 XML 트리

〈표 1〉 기존 듀이번호와 수정된 듀이번호 비교

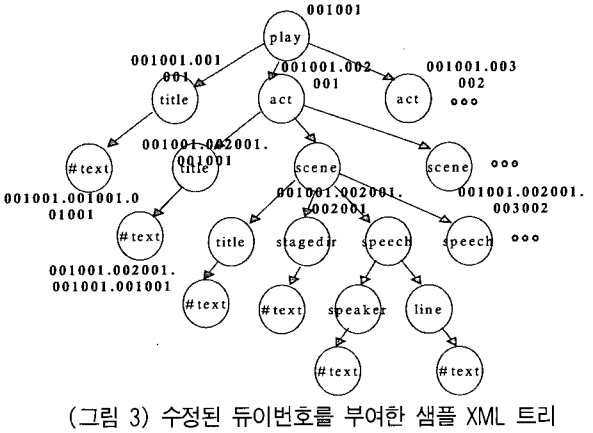
노드 이름	기존 듀이번호에 의한 노드ID	수정된 듀이번호에 의한 노드ID
Play	1	001001
title	1.1	001001.001001
act	1.2	001001.002001
act	1.3	001001.003002
#text	1.1.1	001001.001001.001001
title	1.2.1	001001.002001.001001
scene	1.2.2	001001.002001.002001
#text	1.2.1.1	001001.002001.001001.001001
title	1.2.2.1	001001.002001.002001.001001
stagedir	1.2.2.2	001001.002001.002001.002001
speech	1.2.2.3	001001.002001.002001.003001
speech	1.2.2.4	001001.002001.002001.004002
...	...	...

된 듀이번호의 각 자리수의 크기는 6바이트이며, 그 중 앞 3바이트는 기존 듀이번호의 각 자리수를 정규화한 것이고, 뒤의 3바이트는 같은 이름을 가진 형제노드간의 순서를 나타내기 위한 것이다. (그림 1)는 샘플 XML 문서이고 (그림 2)와 (그림 3)는 샘플 XML 문서를 트리구조로 나타낸 것으로, 각 노드의 ID를 기존의 듀이번호를 사용한 경우와 본 연구에서 사용하고 있는 수정된 듀이번호를 사용한 경우를 비교하여 보여주고 있다. <표 1>은 샘플 XML 문서의 각 노드에 대한 수정된 듀이번호를 기존의 듀이번호와 비교하여 보여주고 있다.

4. XML 문서 저장을 위한 데이터베이스 스키마

4.1 경로 정보에 대한 데이터베이스 스키마

일반적으로 XML 문서에 대한 질의는 경로 표현식을 포함하고 있으므로, 제안 방법에서도 XML 트리의 분해 단위로서 경로를 사용한다. 경로정보를 저장하는 데 있어서 기존 방법의 문제점은 모든 노드에 대해 루트부터 각 노드까지의 경로 정보를 저장함에 따라 정보의 양이 증가하여 질의 처리의 성능을 저하시킨다는 점이다. 그런데, 이러한 경로정보들은 중복된 데이터를 가지고 있다. 예를 들면, <표 2>에서 노드 9에 대한 경로정보는 노드 1, 3, 5, 8에 대한 경로정보를 포함하고 있다. 제안 방법에서는 내부 엘리먼트



(그림 3) 수정된 듀이번호를 부여한 샘플 XML 트리

노드에 대한 경로인 내부 엘리먼트 경로들은 저장하지 않고 전체경로를 나타내는 단말 엘리먼트 노드에 대한 경로정보만 저장함으로써 경로정보의 양을 줄인다. 또한 경로정보에 대한 역 인덱스도 단말 엘리먼트 경로만을 대상으로 하여 구성함으로써 포스팅 리스트의 수가 줄어들게 된다.

제안 방법에서 경로정보에 대한 관계형 스키마는 다음과 같다.

```
Path (LEP_ID, pathExp)
PathIndex (keyword, {LEP_ID, level})
```

경로 테이블(Path Table)은 경로에 대한 역 인덱스 테이블(PathIndex Table)을 만들기 위한 중간과정에서 필요하다. 경로 테이블과 역 인덱스 테이블에서 LEP_ID는 단말 엘리먼트 경로의 ID(Leaf Element Path ID; LEP_ID)를 가리키며, pathExp는 해당 단말 엘리먼트 경로의 경로 표현식을 가리킨다. level은 각 엘리먼트(키워드)가 단말 엘리먼트 경로 내에서, 루트부터 몇 번째에 해당되는 엘리먼트인지를 나타내는 번호이다.

정의 1. 단말 엘리먼트 경로의 ID(Leaf Element Path ID; LEP_ID)는 각 단말 엘리먼트 경로를 식별하기 위해 부여한 일련의 번호이다.

정의 2. 레벨(level)은 각 엘리먼트가 단말 엘리먼트 경로(LEP) 내에서, 루트부터 몇 번째에 해당되는 엘리먼트인지를 나타내는 번호이다.

<표 2>는 앞에서 서술한 샘플 XML 문서의 모든 경로를 저장한 경로 테이블이고, <표 3>는 내부 엘리먼트 경로를 제외하고 단말 엘리먼트 경로만을 저장한 경로 테이블이다. 10개의 경로가 6개로 줄어든 것을 볼 수 있다.

〈표 2〉 모든 경로를 저장한 경로 테이블의 예

번호	경로 표현식
1	/play
2	/play/title
3	/play/act
4	/play/act/title
5	/play/act/scene
6	/play/act/scene/title
7	/play/act/scene/stagedir
8	/play/act/scene/speech
9	/play/act/scene/speech/speaker
10	/play/act/scene/speech/line
...	...

〈표 3〉 단말 엘리먼트 경로만 저장한 경로테이블의 예

번호	경로 표현식
1	/play/title
2	/play/act/title
3	/play/act/scene/title
4	/play/act/scene/stagedir
5	/play/act/scene/speech/speaker
6	/play/act/scene/speech/line
...	...

〈표 4〉 경로에 대한 역 인덱스 테이블의 예

키워드	포스팅 리스트
play	<1,1> <2,1> <3,1> <4,1> <5,1> <6,1>
title	<1,2> <2,3> <3,4>
act	<2,2> <3,2> <4,2> <5,2> <6,2>
scene	<3,3> <4,3> <5,3> <6,3>
stagedir	<4,4>
speech	<5,4> <6,4>
speaker	<5,5>
line	<6,5>
...	...

경로에 대한 역 인덱스 테이블(PathIndex Table)은 경로 테이블(Path Table)의 경로 표현식(pathExp)에 나타나는 각 레이블을 키워드로 취급하고, 각 레이블 즉, 키워드에 대한 역 인덱스를 구성한 것이다. 각 레이블에 대한 포스팅 리스트는 <LEP_ID, level>로 구성된다. LEP_ID는 내부 엘리먼트 경로를 제외한 단말 엘리먼트 경로의 ID이고, level은 단말 엘리먼트 경로 중 해당 레이블이 나타난 레벨을 나타낸다. 샘플 XML 문서에 대한 역 인덱스 테이블은 <표 4>과 같이 구성된다.

4.2 노드 정보에 대한 데이터베이스 스키마

제안 방법에서 각 노드에 대한 정보를 저장하기 위한 데이터베이스 스키마는 각 노드 타입별로 하나의 관계형 스키마를 생성한다. 노드 타입별 정보를 저장하는 데 있어서 기존 방법과의 차이는 내부 엘리먼트 경로를 제외한 단말 엘리먼트 경로만을 저장함에 따라 경로 ID가 대표 단말 엘리먼트 경로의 ID(Representative Leaf Element Path ID; RLEP_ID)이고, 각 노드에 대한 속성으로 대표 단말 엘리먼트 경로에서의 레벨과 노드이름이 추가되어 있는 점이다. 각 엘리먼트 노드의 단말 엘리먼트 경로는 여러 개가 있을 수 있으므로, 그 중에서 가장 작은 값을 갖는 단말 엘리먼트 경로의 ID를 RLEP_ID로 사용한다. 노드 ID(nodeID)는 노드간의 순서와 부모/조상간의 관계, 그리고 같은 이름을 가진 형제노드간의 순서 관계를 보존하기 위해 3장에서 서술한 수정된 듀이번호로 부여한다. (그림 3)와 <표 1>에서 볼 수 있듯이 노드 ID는 엘리먼트 노드, 애트리뷰트 노드, 텍스트 노드 등 노드 타입에 관계없이 모든 노드에 대하여 같은 형식으로 부여한다. 레벨은 단말 엘리먼트 경로 중 해당 노드가 위치한 레벨을 가리키고, 노드이름은 연산자 '//'가 있는 경우 정확한 레벨을 알 수 없으므로 경로 ID 검색 후 노드를 찾을 때 해당하는 노드만 찾기 위해 필요하다.

정의 3. 대표 단말 엘리먼트 경로의 ID(Representative Leaf Element Path ID; RLEP_ID)는 각 엘리먼트 노드가 포함되어 있는 단말 엘리먼트 경로가 여러 개가 존재할 때, 그 중에서 경로 ID(LEP_ID)의 값이 가장 작은 경로의 ID를 말한다.

정의 4. 노드 ID는 하나의 XML 문서 내에서 각 노드의 순서와 부모/조상간의 관계, 그리고 같은 이름을 가진 형제 노드간의 순서 관계를 보존하고 있는 번호로, 3장에서 서술

한 수정된 듀이번호로 부여된다.

제안 방법에서 각 노드 타입별 관계형 스키마는 다음과 같다.

Element (docID, nodeID, RLEP_ID, level, nodeName)
Text (docID, nodeID, RLEP_ID, level, value, nodeName)
Attribute (docID, nodeID, RLEP_ID, level, value, nodeName)

5. XML 문서에 대한 질의 처리

5.1 질의 처리 알고리즘

제안방법에서 역 인덱스를 이용하여 경로 표현식의 질의를 처리하는 알고리즘은 (그림 4)와 같다. 먼저, 경로 표현식의 각 레이블 즉, 엘리먼트별로 레벨이 맞는 경로 ID들

```
procedure SearchNodes(PE: path expression of query, NOL: number
of labels of PE)
returns nodeSet: set of nodes
begin
    // find LEP_IDs with the same label and level in PathIndex
Table
    initialize matchedPathIDset[1..NOL];
    i = 0;
    foreach label of PE do
    begin
        if (PE has '//' and the label appears after the first '//')
then
            matchedPathIDset[i++] =
findMatchedPathID_GTE(label, i+1);
        else matchedPathIDset[i++] = findMatchedPathID_EQ(label,
level);
    end
    // filter LEP_IDs that are stored in all matchedPathIDsets for
all labels of PE
    initialize filteredPathIDset;
    foreach LEP_ID of matchedPathIDset[0] do
    begin
        filter LEP_ID that is stored in all matchedPathIDsets;
        if (filtered) then
            store LEP_ID in filteredPathIDset;
    end
    // find nodes with the same LEP_ID, level and value in
Element/Text/Attribute Table
    initialize nodeSet;
    for (i = 0; filteredPathIDset[i] != NULL; i++) do
    begin
        if (PE has '//') then
            nodeSet +=
findMatchedNode_GTE(matchedPathIDset[i], level);
        else if (PE has same-named sibling index) then
            nodeSet +=
findMatchedNode_IDX(matchedPathIDset[i], level, idx_level,
idx_value);
        else if (PE has attribute value) then
            nodeSet +=
findMatchedNode_ATT(matchedPathIDset[i], level, att_value);
        else if (PE has text value) then
            nodeSet +=
findMatchedNode_TXT(matchedPathIDset[i], level, txt_value);

        else nodeSet +=
findMatchedNode_EQ(matchedPathIDset[i], level);
    end
    return nodeSet;
end
```

(그림 4) 질의 처리 알고리즘

```
procedure findMatchedPathID_EQ (label, level)
returns matched LEP_IDs in PathIndex Table
begin
    SELECT      pi1.LEP_ID
    FROM        PathIndex pi1
    WHERE       pi1.keyword = label AND pi1.level = level;
end

procedure findMatchedPathID_GTE (label, level)
returns matched LEP_IDs in PathIndex Table
begin
    SELECT      pi1.LEP_ID
    FROM        PathIndex pi1
    WHERE       pi1.keyword = label AND pi1.level >= level;
end
```

(그림 5) 엘리먼트별 레벨이 맞는 경로ID 찾는 절차들

```
procedure findMatchedNode_EQ (LEP_ID, level)
returns matched (docID, nodeID)s in Element Table
begin
    SELECT      e1.docID, e1.nodeID
    FROM        Element e1
    WHERE       e1.RLEP_ID = LEP_ID AND e1.level = level;
end

procedure findMatchedNode_TXT (LEP_ID, level, txt_value)
returns matched (docID, nodeID)s in Text Table
begin
    SELECT      t1.docID, t1.nodeID
    FROM        Text t1
    WHERE       t1.RLEP_ID = LEP_ID AND t1.level = level
    AND         t1.value LIKE '%"txt_value"%';
end
```

(그림 6) 노드테이블에서 해당 노드를 찾는 절차의 예

(LEP_IDs)을 역 인덱스 테이블에서 찾아서 경로 ID 집합(matchedPathIDset)을 만든다. 이 때 만들어지는 경로 ID 집합의 개수는 경로 표현식에 포함되어 있는 엘리먼트들의 수와 같다. 각 엘리먼트별로 레벨이 맞는 경로 ID를 찾는 SQL 문장은 (그림 5)에 있는 바와 같다. 경로 표현식에 연산자 '//'이 포함되어 있으며 대상 엘리먼트가 '//' 뒤에 나타나는 경우에는 엘리먼트의 레벨보다 크거나 같은 경우를 찾고, 그렇지 않으면 엘리먼트의 레벨과 같은 경우를 찾는다.

경로 표현식의 각 레이블 즉, 엘리먼트별로 레벨이 맞는 경로 ID들을 역 인덱스 테이블에서 찾은 다음에는, 엘리먼트별로 찾아진 경로 ID들을 저장해 놓은 집합(matched PathIDset)들에서 모든 집합에서 나타나는 경우의 경로 ID들만을 걸러낸다. 이렇게 걸러진 경로 ID들이 찾고자 하는 목표의 경로 ID들이 되며, 이 목표 경로 ID들은 filtered PathIDset에 저장한다.

목표 경로 ID들을 찾은 다음에는, 경로 ID와 경로 표현식의 레벨을 이용하여 엘리먼트/애트리뷰트/텍스트 테이블에서 해당 노드들을 찾는다. 이 때 각 테이블에서 해당 노드를 찾는 SQL 문장은 (그림 6)에 있는 예와 같다. 연산자 '//'가 있는 경우에는 정확한 레벨을 알 수 없으므로 노드를 찾을 때 해당하는 노드만 찾기 위해 노드이름도 비교하는 것이 필요하다.

5.2 질의 처리 예제

다음과 같은 일반적인 정규 경로식으로 표현된 질의 예제

1(Q1)을 처리하는 경우를 예를 들어 질의 처리 과정을 살펴본다.

예제 1 (Q1) : /PLAY/ACT/EPILOGUE/SPEECH

먼저, 경로에 대한 역 인덱스 테이블에서, 레이블(엘리먼트) PLAY, ACT, EPILOGUE, SPEECH 각각에 대하여 레벨이 맞는 (Q1에서 레벨은 PLAY부터 순서대로 1, 2, 3, 4로 부여됨) 경로 ID들을 찾아서 경로 ID 집합(matchPathIDset)들을 만든다. Q1 질의에서는 경로 표현식 상에 엘리먼트가 4개이므로 만들어지는 matchPathIDset의 개수도 4개이다. 다음은, 앞 단계에서 만들어진 4개의 경로 ID 집합에서 경로 ID가 모두 나타나는 경우의 경로 ID만을 걸러냄으로써 목표 경로 ID를 찾아낸다. 그 다음 단계에서는, 목표 경로 ID와 최대 레벨 값(Q1에서는 4)을 이용하여 엘리먼트 테이블에서 해당 노드들을 찾는다.

XRel에서는 경로 테이블에서 해당 경로식이 있는 경로ID를 찾을 때, 모든 경로식에 대한 스트링 매치가 필요하다. 제안방법에서는 역 인덱스를 검색함으로써 기존의 방법에서 필요한 스트링 매치를 하지 않아도 되며, 기존의 역 인덱스 방법과의 차이는 대상 경로정보의 중복을 제거하여 엘리먼트별 포스팅 리스트의 수를 줄임으로써 성능을 향상시킨다는 점이다.

연산자 '//'가 있는 질의 예제 2(Q2)를 처리하는 경우, XRel에서 처리하려면 다음과 같은 SQL 문장이 필요하다.

예제 2 (Q2) : /PLAY//SPEECH

```
SELECT  e1.docID, e1.st, e1.ed
FROM    Element e1, Path p1
WHERE   p1.pathexp LIKE '#/PLAY#%/SPEECH%'
AND     e1.pathID = p1.pathID;
```

Q2 질의에 대해서도 XRel에서는 경로 테이블에서 해당 경로식이 있는 경로ID를 찾을 때, 모든 경로식에 대한 스트링 매치가 필요하다. 제안방법에서는 연산자 '//'가 있는 Q2에서도 '//'가 없는 Q1에서와 마찬가지로 (그림 4)와 같은 알고리즘으로 처리할 수 있다. 다만 연산자'//' 이후에 나타나는 엘리먼트의 경우 레벨의 정확한 값을 알 수 없으므로 레벨을 비교할 때 레벨이 같거나 큰 경우에 해당하는 것으로 처리해야 하며, 노드를 찾을 때에도 정확한 레벨을 모르므로 노드이름도 함께 비교해야 한다.

같은 이름을 가진 형제노드간의 관계를 알아야 하는 질의 예제 3(Q3)을 처리하는 경우는 Q1과 같은 방식으로 처리하되, 찾아진 경로 ID와 경로 표현식의 레벨을 이용하여 엘리먼트 테이블에서 해당 노드들을 찾을 때, 노드 ID를 비교하는 부분만 추가하면 된다. 제안 방법의 노드 ID에는 문서 내에서의 노드 순서와 이름이 같은 형제노드의 순서가 포함되어 있기 때문이다. Q3 예제에서 엘리먼트 테이블에서 노드를 찾는 SQL 문장은 다음과 같다.

예제 3 (Q3) : /PLAY/ACT/SCENE[3]/SPEECH

```
SELECT      e1.docID, e1.nodeID
FROM        Element e1
WHERE       e1.RLEP_ID = LEP_ID
AND         e1.level = level
AND         e1.nodeID LIKE '%003._____';
```

애트리뷰트의 값을 찾거나 텍스트 값을 찾는 질의에서도 Q1과 같은 방식으로 처리하되, 찾아진 경로 ID와 경로 표현식의 레벨을 이용하여 엘리먼트 테이블에서 해당 노드들을 찾을 때, 애트리뷰트 테이블이나 텍스트 테이블의 값을 비교하는 부분만 추가하면 된다.

연산자 '//'가 있으면서 텍스트 값을 찾는 복잡한 질의에서는 연산자 '//'가 있으면서 단순한 질의인 Q2과 같은 방식으로 처리하되, 찾아진 경로 ID와 경로 표현식의 레벨을 이용하여 텍스트 테이블에서 해당 노드들을 찾을 때, 텍스트 테이블의 값을 비교하는 부분만 추가하면 된다.

지금까지 살펴보았듯이 기존의 XRel과 같은 방법에서는 경로 테이블에서 해당 경로식이 있는 경로ID를 찾을 때, 모든 경로식에 대한 스트링 매치가 필요하다. 제안방법에서 역 인덱스를 이용하지 않는다면, 질의에 연산자 '//'가 없을 경우에는 단말 엘리먼트 경로만 저장된 경로 테이블을 이용해도 되므로 스트링 매치를 해야 하는 경로식의 수를 줄임으로써 경로ID 찾는 시간을 줄일 수 있다. 그러나 질의에 연산자 '//'가 있는 경우에는 단말 엘리먼트 경로뿐만 아니라 내부 엘리먼트 경로에 대한 스트링 매치가 필요하므로 XRel과 비교 시 별 차이가 없다. 그렇지만 제안방법에서 역 인덱스를 이용하는 경우에는, 경로식의 레이블에 대한 역 인덱스 테이블에서 해당 경로식의 레이블(키워드)을 이용하여 경로 ID를 찾을 때, 모든 경로에 대해 포스팅 리스트를 만들지 않고 단말 엘리먼트 경로에 대해서만 포스팅 리스트를 만들기 때문에, 기존의 역 인덱스 방법에 비해 키워드별 포스팅 리스트의 수를 줄이게 되어 경로ID 찾는 시간을 줄일 수 있다.

6. 실험 및 분석

6.1 실험 설정

본 연구에서 실험을 위해 사용한 하드웨어는 Pentium4 3GHz, 512 RAM이고, 운영체제는 Windows XP, 데이터베이스 관리 시스템은 MS SQL-Server 2000이었다. 프로그래밍 언어는 Java와 Visual Basic을, XML Parser는 APACHE Xerces2를 사용하였다.

실험용 데이터는 셰익스피어의 희곡 모음에 관한 XML 문서 등을 포함하고 있는 Wisconsin XML Data Set[20]과 가상 데이터를 사용하였으며, 실험 데이터의 상세 내용은 <표 5>와 같다.

성능 평가를 위해 사용한 질의 예제는 <표 6>와 같이 6가지로, 전형적인 XPath 형태의 질의이다. Q1은 일반적인 경로 표현식으로 표현되는 정규 경로식의 질의이고, Q2는 경로 표현식에 재귀적 내림 연산자인 '//'가 포함되어 있는 질의이며, Q3는 같은 이름의 형제 노드 중의 어느 것을 찾

〈표 5〉 실험에 사용된 XML 문서에 대한 상세정보

구분	실제 데이터 (Wisconsin XML Data Set)	가상 데이터 (Depth 깊고 옆으로 적은 경우)	가상 데이터 (Depth 낮고 옆으로 많은 경우)
문서의 종류	10	300	300
문서의 개수	320	12,000	12,000
엘리먼트 노드 수	428,558	589,507	754,032
애트리뷰트 노드 수	36	55,967	70,176
텍스트 노드 수	339,631	283,131	617,496
문서당 평균 엘리먼트 수	1,339.2	49.1	62.8
문서당 평균 애트리뷰트 수	0.1	4.7	5.8
문서당 평균 텍스트 수	1,061.3	23.6	51.5

〈표 6〉 성능 평가를 위해 사용한 질의 예제

경로 표현식	특징
Q1: /PLAY/ACT/EPILOGUE/SPEECH	정규 경로식 질의
Q2: /PLAY//SPEECH	연산자 '//' 포함 질의
Q3: /PLAY/ACT/SCENE[3]/SPEECH	SAME SIBLING 관계 질의
Q4: /Element446/Element399/ Element930/[@Attribute642 = 'nrzmmup']	애트리뷰트트 값을 찾는 질의
Q5: PLAY/ACT/SCENE/SPEECH/ [SPEAKER = 'LEPIDUS']	텍스트 값을 찾는 질의
Q6: /PLAY/ACT//SPEECH/ [SPEAKER = 'LEPIDUS']	Q2 + Q4

는 질의이다. Q4는 애트리뷰트 값을 가지고 있는 노드를 찾는 질의이고, Q5는 텍스트 값을 가지고 있는 노드를 찾는 질의이며, Q6는 Q2와 Q5가 혼합된 형태의 질의이다.

성능 평가의 방법은 기존의 모든 경로를 저장하는 역 인덱스 방법과 LEP만 저장하여 역 인덱스를 만드는 제안 방법을 비교하는 방식으로 실시하였다. 일반적인 데이터베이스 환경에서 인덱스를 만드는 오버헤드 혹은 인덱스 자체의 검색 시간 등을 고려할 때, 적은 양의 데이터베이스인 경우에는 인덱스를 이용하지 않고 스트링 매칭으로 직접 검색하는 것이 빠를 수도 있다. 그러나 우리 실험은 대용량 XML을 고려하고 있으며, 문서가 많아질수록 스트링 매칭을 이용하는 방법보다 역 인덱스를 이용하는 방법이 타당하다는 것은 이미 기존의 연구 XIR-Linear[8]에서 확인되었기에, 우리는 여기서 스트링 매칭을 이용하는 방법과의 비교는 하지 않고, 역 인덱스를 사용하는 기존의 방법과 제안 방법을 비교하였다.

역 인덱스를 사용하는 2가지 방법에 대하여 성능에 영향을 미치는 경로의 수와 역 인덱스의 포스팅 리스트의 수를 비교하고, XML 문서 개수의 증가에 따른 데이터베이스 크기와 질의 처리 시간의 변화를 측정하여 비교하였다. 여기서 질의 처리 시간은 조건에 부합하는 XML 문서의 노드를 찾아서 해당 노드 ID 값을 돌려주는 데 걸리는 시간으로, 인터넷 검색과 같은 질의 즉, XML 문서들에 대한 키워드 검색 등과 같은 질의의 결과를 생성하는 시간은 포함하지 않는다.

〈표 7〉 문서 종류별 경로 수의 비교 - 실제 데이터

문서 종류	모든 경로의 수	단말 엘리먼트 경로의 수 (%)	내부 엘리먼트 경로의 수 (%)
Shakes	23	15 (65.2)	8 (34.8)
club	13	9 (69.2)	4 (30.8)
bib	14	10 (71.4)	4 (28.6)
cq	20	15 (75.0)	5 (25.0)
Department	40	29 (72.5)	11 (27.5)
Edmunds	84	74 (88.1)	10 (11.9)
Personnel	15	10 (66.7)	5 (33.3)
Movies	12	6 (50.0)	6 (50.0)
Actors	20	10 (50.0)	10 (50.0)
Bom	32	19 (59.4)	13 (40.6)
(경로 수의 합계)	273	197 (72.2)	76 (27.8)
(문서별 %의 평균)		(66.8)	(33.2)

〈표 8〉 문서 종류별 경로 수의 비교 - 가상 데이터

데이터 종류	모든 경로의 수	단말 엘리먼트 경로의 수 (%)	내부 엘리먼트 경로의 수 (%)
가상 데이터 세트 (Depth 깊고 옆으로 적은 경우)	788	351 (44.5)	437 (55.5)
가상 데이터 세트 (Depth 낮고 옆으로 많은 경우)	1,585	1,235 (77.9)	350 (22.1)

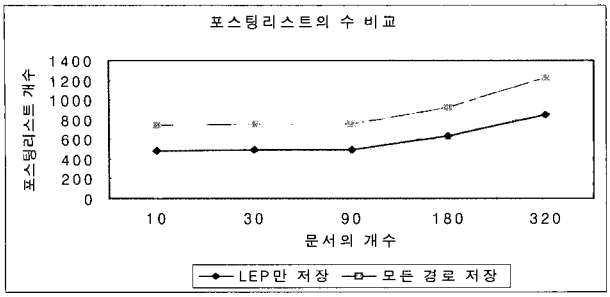
6.2 실험 결과

<표 7>은 실험에 사용한 실제 데이터의 XML 문서의 종류별로 모든 경로, 단말 엘리먼트 경로, 내부 엘리먼트 경로로 구분하여 경로의 수를 보여주고 있다. 모든 경로를 이용하는 기존의 방법과 비교 시, 단말 엘리먼트 경로만을 이용하는 제안 방법에서는 경로의 수를 11.9%에서 50%까지 줄일 수 있었다. 모든 문서의 경로의 수를 합한 경우에는 평균 27.8% 정도 줄일 수 있으며, 각 문서별 비율의 평균값으로 추정하면 약 33.2% 정도 줄일 수 있다.

<표 8>은 실험에 사용한 가상 데이터의 경로의 수를 모든 경로, 단말 엘리먼트 경로, 내부 엘리먼트 경로로 구분하여 경로의 수를 보여주고 있다. 모든 경로를 이용하는 기존의 방법과 비교 시, 단말 엘리먼트 경로만을 이용하는 제안 방법에서는 문서 개수가 12,000개일 때, XML 문서 트리의 깊이가 깊고 옆으로 적은 경우에는 55.5%, XML 문서 트리의 깊이가 낮고 옆으로 많은 경우에는 22.1%의 경로를 줄일 수 있었다.

(그림 7)은 질의 처리 시 성능에 영향을 미치는 역 인덱스의 포스팅 리스트 개수의 차이를 기존의 모든 경로를 저장하는 경우와 비교하여 보여주고 있다. XML 문서 개수의 증가와 관계없이 비슷한 차이를 보여주고 있으며, 기존의 모든 경로를 저장하는 경우와 비교 시, 단말 엘리먼트 경로만을 저장하는 제안 방법에서는 포스팅 리스트의 개수를 실제 데이터의 문서의 개수가 320개일 때 31.1% 정도 줄일 수 있었다(1,232에서 849개로 줄임).

<표 9>는 XML 문서 개수의 증가에 따른 데이터베이스 크기를 비교하여 보여주고 있다. 모든 경로를 이용하는 기



(그림 7) 문서 수의 증가에 따른 포스팅리스트 수 변화

〈표 9〉 문서 수별 DB 크기의 비교 - 실제 데이터

문서의 수	DB 크기 (MB)	
	LEP만 저장	모든 경로 저장
10	21.06	21.08
30	22.34	22.36
90	54.37	54.39
180	108.74	108.78
320	172.33	172.37

존의 방법과 비교 시, 단말 엘리먼트 경로만을 이용하는 제안 방법에서의 데이터베이스 크기는, 엘리먼트, 텍스트, 애트리뷰트 테이블은 같고 다만 경로 테이블과 경로에 대한 역 인덱스 테이블만 차이가 있으므로, 좀 작아지기는 하지만 거의 차이가 없는 수준이었다.

(그림 8)은 XML 문서 개수의 증가에 따른 질의 처리 시간의 변화를 보여주고 있다. 기존의 모든 경로를 저장하는 경우와 비교 시, 단말 엘리먼트 경로만을 저장하는 제안 방법에서는 실험을 한 모든 질의 예제에서 더 좋은 성능을 보이고 있다. 문서의 수가 많아질수록 성능의 차이는 커지고 있다. XML 문서의 깊이가 깊고 옆으로 적은 경우 문서의 수가 많아질수록 더 큰 차이를 보이고 있다.

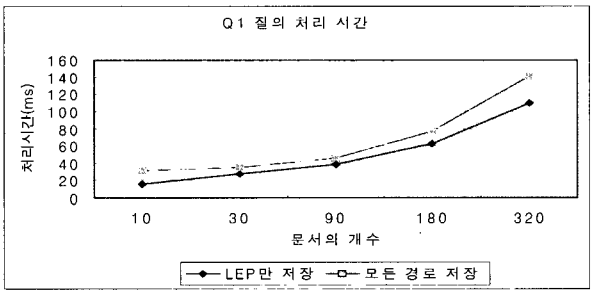
7. 결 론

본 논문에서는 XML 문서에 대한 효율적인 질의처리를 위해 기존의 경로 기반 기법을 좀더 보완할 수 있는 방법을 제안하였다. 제안 방법에서는 경로정보 중 반드시 필요하지 않은 내부 엘리먼트 경로들은 제외하고 단말 엘리먼트 경로(leaf element path)만을 대상으로 하여 역 인덱스를 구성함으로써, 기존의 XRel과 같은 스트링 매치 방법보다 빠를 뿐만 아니라 부분매치에 아주 우수한 기존의 역 인덱스 방법보다도 빠른 성능을 보였다.

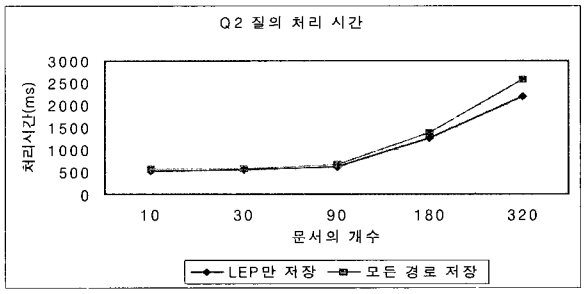
제안 방법은 상용 관계형 데이터베이스의 타입이나 기능의 확장은 필요하지 않으나 질의처리를 위한 별도의 인덱싱은 필요하다. 향후 연구에서는 XML 문서의 갱신에 대한 지원이 포함되어야 할 것이다. 일반적으로 XML 문서의 콘텐츠는 시간과 함께 변하므로, XML 문서상의 시간적 변화를 관리하는 것은 많은 애플리케이션에서 대단히 유용할 것이다. 이러한 시간에 따라 변하는 XML 문서를 효율적으로 관리하는 것도 XML 문서의 갱신에 대한 지원과 함께 연구할 주제가 될 것으로 본다.

참 고 문 헌

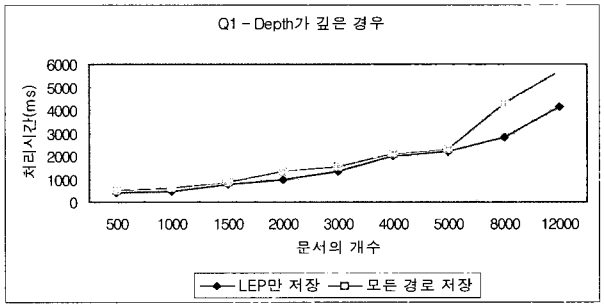
[1] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita and C. Zhang, "Storing and Querying Ordered



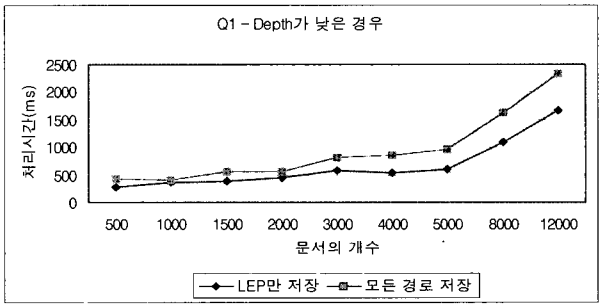
(a) 실제데이터에 대한 Q1 질의처리의 변화



(b) 실제데이터에 대한 Q2 질의처리의 변화



(c) depth 깊은 가상데이터에 대한 Q1 질의처리의 변화



(d) depth 낮은 가상데이터에 대한 Q1 질의처리의 변화

(그림 8) 문서 수의 증가에 따른 질의처리시간의 변화

XML Using a Relational Database System," ACM SIGMOD 2002.
[2] M. Yoshikawa, T. Amagasa, T. Shimura and S. Uemura "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases," ACM Transactions on Internet Technology, Vol.1, pp.110-141, August, 2001.
[3] WWW Consortium, XML Query Data Model, http://www.w3.org/
[4] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS," IEEE Data Engineering Bulletin 22(3), pp.27-34, 1999.
[5] J. Zhang, "Application of OODB and SGML Techniques in Text Database: An Electronic Dictionary System," SIGMOD Record 24, pp.3-8, 1995.
[6] H. Jiang, H. Lu, W. Wang and J. Yu, "XParent: An Efficient RDBMS-Based XML Database System," ICDE 2002.
[7] 민경섭, 김형주, "상이한 구조의 XML 문서들에서 경로 질의 처리를 위한 RDBMS 기반 역 인덱스 기법", 정보과학회논문지: 데이터베이스, 30(4):420-428, 2003.
[8] 박영호, 한욱신, 황규영, "정보 검색 기술을 이용한 대규모 이질적인 XML 문서에 대한 효율적인 선형 경로 질의 처리", 정보과학회논문지: 데이터베이스, 31(5):540-552, 2004.
[9] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expression," VLDB 2001.
[10] S. Sundara, Y. Hu, T. Chorma and J. Srimivasan, "Developing an Indexing Scheme XML Document Collections Using the Oracle8i Extensibility Framework," VLDB 2001.
[11] M. G. Bauer, F, Ramsak and R. Bayer, "Multidimensional Mapping and Indexing,"
[12] S. Pal, I. Cseri, O. Seeliger, G. Schaller, L. Giakoumakis, V. Zolotov, "Indexing XML Data Stored in a Relational Database," VLDB 2004.
[13] J. McHugh, S. Abiteboul, R. Goldman, D. Quass and J. Widom, "Lore: A Database Management System for Semistructured Data," 1997.
[14] H. Schoning, "Tamino - a DBMS Designed for XML"
[15] F. Tian, D. J. Dewitt, J. Chen and C. Zhang, "The Design and Performance Evaluation of Alternative XML Storage Strategies"
[16] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl, "From Structured Documents and to Novel Query Facilities", ACM SIGMOD 1994.
[17] J. Shanmugasundaram et al., "Relational Databases for Querying XML Documents: Limitation and Opportunities," VLDB 1999.
[18] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason and M. Shadmon, "A Fast Index for Semi-structured Data," VLDB 2001.
[19] C. Chung, J. Min and K. Shim, "APEX: An Adaptive Path Index for XML Data," ACM SIGMOD 2002.
[20] Wisconsin XML Data Set, http://www.cs.wisc.edu/niagara/data.html

이 혜 자

e-mail : hjlee@ysc.ac.kr
1982년 서울대학교 간호학과(학사)
1989년 연세대학교 산업대학원 전산전공 (공학석사)
2005년 경희대학교 대학원 컴퓨터공학과 (공학박사)
1982년~1996년 ㈜데이콤, ㈜삼보컴퓨터 등 근무
1996년~1998년 한국보건의료관리연구원 수석연구원
1998년~현재 용인송담대학 의료정보시스템과 조교수
관심분야 : 의료정보시스템, XML 데이터베이스, 정보 모델링, 전문가시스템, 지식관리 등

정 병 수

e-mail : jeong@nms.kyunghee.ac.kr
1983년 서울대학교 전자계산기공학과 (공학사)
1985년 한국과학기술원 전산학과(석사)
1995년 Georgia Institute of Technology, College of Computing(박사)
1985년~1989년 한국데이타통신(주) 정보통신연구소 선임연구원
1996년~현재 경희대학교 전자정보학부 부교수
관심분야 : 병렬 데이타베이스, 실시간 데이터베이스

김 대 호

e-mail : techwing@khu.ac.kr
1997년 경희대학교 전자계산공학과(학사)
1999년 경희대학교 대학원 전자계산공학과 (공학석사)
2005년 경희대학교 대학원 컴퓨터공학과 (공학박사)
관심분야 : 실시간 데이터베이스, 이동 데이터베이스, 동시성 제어 등

이 영 구

e-mail : yklee@khu.ac.kr
1992년 한국과학기술원 전산학과(학사)
1994년 한국과학기술원 전산학과(석사)
2002년 한국과학기술원 전산학과(박사)
2002년 3월~2004년 2월 한국과학기술원 박사 후 연구원
2002년 9월~2004년 2월 미국 University of Illinois at Urbana-Champaign, Postdoctoral Research Associate
2004년~현재 경희대학교 전자정보대학 전임강사
관심분야 : 유비쿼터스 데이터베이스, 데이터 웨어하우징, 데이터 마이닝, Bioinformatics