XML 문서에 대한 RDBMS에 기반을 둔 효율적인 역색인 기법

(An Efficient Inverted Index Technique based on RDBMS for XML Documents)

서 치 영 † 이 상 원 †† 김 형 주 †††

(Chi-young seo) (Sang-won Lee) (Hyoung-joo Kim)

요 약 XML 정보검색 시스템이 XML 문서에 대한 포함질의를 지원하기 위해서는 기존의 정보검색 분야에서 널리 쓰이는 역색인 기법을 XML문서에 대해서도 적용이 가능하도록 확장해야 한다. 본 논문에서는 확장된 역색인 정보를 저장하고 XML 문서에 대한 포함질의를 처리하는 방법을 이전 연구에서와 같이 두 가지 관점에서 제시한다. 하나는 관계형 데이타베이스 관리 시스템(RDBMS)을 이용해서 역색인 정보를 저장하고 질의를 처리하는 방법이고 다른 하나는 RDBMS 대신 역 리스트 엔진(Inverted List Engine)을 이용하는 방법이다. 이전 연구에서 역색인을 확장한 방식은 두 가지 문제점이 존재한다. 하나는 RDBMS를 이용하는 방법이 역 리스트 엔진을 이용하는 방법에 비해 성능 상으로 많이 안 좋다는 점이고, 다른 하나는 RDBMS 상에서 포함질의를 처리 시, 질의의 경로길이에 비례해서 조인연산이 증가하고 조인연산도 크기가 큰 테이블간의 조인이 된다는 점이다. 본 논문에서는 이러한 문제점들을 해결하고자 이전연구와는 다르게 역색인을 확장하여 RDBMS를 이용하는 방법의 효율성을 밝힌다.

키워드 : 역색인, XML, 질의처리

Abstract The inverted index widely used in the existing information retrieval field should be extended for XML documents to support containment queries by XML information retrieval systems. In this paper, we consider that there are two methods in storing the inverted index and processing containment queries for XML documents as the previous work suggested: using a RDBMS or using an inverted list engine. It has two drawbacks to extend the inverted index in the previous work. One is that using a RDBMS is much worse in the performance than using an inverted list engine. The other is that when containment queries are processed in a RDBMS, there is an increase in the number of a join operation as the path length of a query increases and a join operation always happens between large tables. In this paper, we extend the inverted index in a different way to solve these problems and show the effectiveness of using a RDBMS.

Key word : inverted index, XML, query processing

1. 서 론

XML[1]이 웹 상에서 문서 교환의 표준으로 지정되고 난 후, 상당수의 회사나 단체들이 그들의 문서 포맷으로 XML을 채택하기 시작했다. 예를 들어, 자동차 회사에서 각종 자동차에 대한 정보를 나타내기 위해 사용될 수 있고, 도서관에서 도서목록에 대한 정보를 나타내기 위해 사용될 수 있다. 또한 전자상거래 사이트에서 각종 상품에 대한 정보를 나타내기 위해 사용될 수 있다. 결국, 이러한 추세로 방대한 XML 문서가 생성되며, 이로 인해 XML 문서 데이타베이스로부터 정보를 효율적으로 추출해내기 위한 연구가 필요하게 되었다.

· 본 연구는 서울대 두뇌한국-21 사업과 정보통신부 ITRC (e Business 기술 연구센터), 그리고 학술진흥재단 신진교수과제(KRF-2002-003-D00447)의 지원으로 수행되었음.

† 비 회 원 : ITCamp(주) 연구소 연구원
cyseo@oopsla.snu.ac.kr
†† 비 회 원 : 성균관대학교 정보통신공학부 교수
wonlee@ece.skku.ac.kr
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr

논문접수 : 2001년 11월 30일
심사완료 : 2002년 11월 1일

이러한 경향을 반영하듯 XML 문서들로부터 정보를 추출하기 위한 수단으로 많은 질의어가 제안되었다. 어떤 질의어가 결국 표준으로 될 지는 아직도 불투명하지만, 한가지 분명한 점은 이들 질의어들을 이용하여 XML 문서로부터 정보를 추출 시, 포함질의(containment query)가 XML 정보검색 시스템 상에서의 질의의 핵심이 된다는 것이다. 여기서 말하는 포함질의란[2], XML 문서상에서 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함(containment)관계에 기반을 둔 질의를 말한다.

이러한 포함질의가 XML 정보검색 시스템 상에서 XML 문서에 대한 질의의 핵심적인 부분이 되기 때문에 포함질의를 어떤 방법으로 지원하느냐 하는 것이 중요한 문제가 된다. 이것을 해결하기 위한 한 방법으로 기존의 정보검색분야에서 널리 쓰이는 역색인(Inverted index) 기법을 고려할 수 있다.

본 논문에서는, 전통적인 역색인 방법을 XML 문서에 맞게 확장하고 확장된 역색인 정보를 저장하고 포함질의를 처리하는 방법이 최근 연구[2]와 같이 크게 두 가지가 있다고 본다. 하나는 독립된 역 리스트 엔진(Inverted list engine 또는 IR engine 이라 부른다.)을 구축하여 역색인 정보를 저장하고 포함질의를 처리하는 방법이고 다른 방법은 RDBMS를 이용하는 방법이다.

[2]에서 전통적인 역색인 기법을 XML문서에 맞도록 확장하여 위에서 말한 두 가지 방식의 성능을 서로 비교하였다. 하지만 [2]에서 역색인을 확장한 방식은 두 가지 문제점이 존재한다. 하나는 RDBMS를 이용하는 방식이 대부분의 포함질의에 있어서 역 리스트 엔진을 이용하는 방식보다 성능이 안 좋다는 것이고 다른 하나는 RDBMS 상에서 포함질의 처리 시, 질의의 경로길이에 비례해서 조인연산의 횟수가 증가하고 조인연산도 크기가 큰 테이블간의 조인이 된다는 점이다. 본 논문에서는 이러한 문제점들을 해결하기 위해 [2]에서와는 다르게 역색인을 확장하여 RDBMS를 이용하는 방법이 역 리스트 엔진을 이용하는 방법보다 성능 면에서 별 차이가 없을 뿐만 아니라 [2]의 방법보다 성능 면에서 많은 향상을 보았음을 보인다.

본 논문의 구성은 다음과 같다. 먼저, 2장에서 관련 연구를 다루고 3장에서는 포함질의가 무엇인지를 살펴보고 포함질의를 효율적으로 처리하기 위한 역색인 기법을 제시한다. 또한, RDBMS에서 포함질의가 어떻게 처리되는지에 대해서도 3장에서 살펴본다. 4장에서는 실험을 통해서 우리가 제시하는 기법의 효율성을 밝힌다. 마지막으로 5장에서 결론을 내린다.

2. 관련 연구

XML 문서에 대한 포함질의를 지원하는 것과 관계된 연구[2,3]가 최근에 몇몇 있었다. 이것들은 모두 역색인을 저장하고 포함질의를 처리하기 위해 RDBMS를 사용했다. [2]는 본 논문에서와 같이 역색인 정보를 저장하고 질의를 처리하는 방법을 두 가지 종류로 보고 각각에 대해서 실험을 했지만 [3]은 RDBMS를 이용하는 방법에 대해서만 실험을 하였다.

[2]는 역 리스트 엔진을 이용하는 방법이 RDBMS를 이용하는 방법보다 실행된 대부분의 질의들에 대하여 성능이 훨씬 좋았고 몇몇 질의들에 있어서는 RDBMS 방법이 역 리스트 엔진 방법보다 성능이 더 좋다고 하였다. 그리고 이러한 성능에 있어서의 차이의 원인을 2가지로 보았다. 하나는 역 리스트 엔진에서 사용된 조인 방법이 RDBMS에서 사용된 조인방법보다 성능이 더 좋다는 것이고 다른 하나는 하드웨어 캐쉬 사용률이 RDBMS가 역 리스트 엔진 보다 낮다는 것이다. 그러므로 RDBMS에서 쓰이는 조인 알고리즘을 역 리스트 엔진에서 사용된 조인 알고리즘으로 대체하고 RDBMS에서 캐쉬 사용률을 높이는 방법을 개발하면 역색인 정보를 저장하고 포함질의를 처리하기 위하여 RDBMS를 이용하는 것도 역 리스트 엔진을 사용하는 것에 비해 성능 상에서 크게 차이가 나지 않을 것이다 라고 하였다. 하지만 [2]에서 제안한 방법은 1장에서 언급한 문제점들이 존재하여 결국, 성능 상에 문제가 발생하게 된다.

반면, [3]은 [2]와는 다르게 역색인을 확장해서 RDBMS에 저장했지만 데이타베이스의 테이블 수가 많아지는 단점이 존재하게 된다. 더구나 역 리스트 엔진을 사용하는 것과의 성능비교를 하지 않았다.

XML 문서를 저장하고 질의를 처리하기 위해 RDBMS를 이용하는 방법에 관계된 연구[4,5,6,7,8]가 이전에 나왔었다. 하지만, 이들 논문들은 주로 XML 문서를 릴레이션(relation)으로 사상(mapping)하는 방법, 또는 그 반대로 릴레이션 데이타를 XML 문서로 변환하는 방법에 대해서 주로 다루었다.

SGML [9]의 등장으로 텍스트 검색에서 문서의 내용과 구조를 함께 고려해야 했고 이에 대한 연구가 많이 나오게 되었다 [10,11,12,13]. SGML 문서에 대한 포함질의 처리와 관계된 연구들은 [14,15,16]등을 들 수 있다. SGML에 대한 이전연구들과 본 논문과의 차이점은 이전연구들은 포함 알고리즘(containment algorithm)의 개발에 초점을 맞추었고, 본 논문은 RDBMS에서 포함

알고리즘을 구현하는 방법에 대해서 주로 다룬다는 점이다.

3. 포함질의 처리

3.1 포함질의

포함질의는 XML문서를 구성하고 있는 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계에 기반을 둔 질의를 말한다. 이러한 포함질의는 XML 정보검색 시스템에 의해 처리되는 질의의 핵심이 되기 때문에 정보검색 시스템 구축 시 XML문서에 대한 포함질의를 효과적으로 처리할 수 있는 방법을 모색해야 한다. 이 방법에 대해서는 3.2절에서 살펴볼 것이다.

본 논문에서는 포함질의를 나타내기 위해 XQuery[17]와 비슷한 경로 표현식을 사용한다. XQuery는 W3C에 의해 제안된 XML 질의어들 중의 하나로 질의를 비교적 간결하고 이해하기 쉽도록 표현하기 위해서 고안되었다. XQuery는 기존에 있던 XML 질의어들 및 데이타베이스 질의어들이 가지고 있었던 특징들을 도입하였다. 예를 들어, XPath[18]와 XQL[19]로부터 경로 표현식을 나타내는 방법을 도입했고 XML-QL[20]로부터는 변수 바인딩(variable binding)하는 방법을 도입했다. SQL로부터는 SELECT-FROM-WHERE 구문을 도입하였다. 그러므로 다양한 기능을 제공할 수 있는 새로운 XML 질의어라 할 수 있다.

본 논문에서는 기본 포함질의의 종류를 [2]에서와 같이 크게 네 가지로 분류한다. 그림 1의 XML문서에 대한 질의 예제로 살펴본다.

• **간접 포함질의**: 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계가 간접 포함관계(조상-자손 관계)로만 이루어진 질의

예제: /companies//profile//'scanners'

맨 앞의 "/"는 companies가 문서의 루트 엘리먼트라는 것을 의미하고 "//"는 간접 포함관계(조상-자손 관계)를 의미하므로 "companies" 루트 엘리먼트가 자손 엘리먼트로 "profile"을 가지며 "profile" 엘리먼트의 자손 엘리먼트들의 내용(content) 중에 "scanners" 단어가 있는 XML 문서를 추출하라는 질의이다.

• **직접 포함질의**: 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계가 직접 포함관계(부모-자식 관계)로만 이루어진 질의

예제: /companies/company/profile/description/'printers'

루트 엘리먼트 "companies"는 자식 엘리먼트로 "company"를 가지며 "company"는 자식 엘리먼트로 "profile"을 가진다. "profile"은 자식 엘리먼트로 "description"을 가지고 "description"은 그것의 내용(content)에 "printers"이라는 단어를 포함해야 한다.

결국, 위 조건들을 만족하는 XML 문서를 추출하라는 질의이다.

• **완전 포함질의**: 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계가 완전 포함관계로만 이루어진 질의

예제: //symbol='AAPL'

"symbol" 엘리먼트가 "AAPL" 단어를 완전포함("symbol" 엘리먼트의 내용이 "AAPL"단어만 포함하고 있는 경우)하고 있는 XML 문서를 검색하라는 질의이다.

```
<Companies>
  <Company>
    <Symbol>AAPL</Symbol>
    <Name>Apple Computer, Inc.</Name>
    <Profile>
      <Address>
        <State>Texas</State>
        <City>Austin</City>
      </Address>
      <Description> Designs, develops, produces, markets and services
        microprocessor based personal computers, related software and peripheral
        products, including laser printers, scanners, compact disk read only memory
        drives and other related products </Description>
    </Profile>
  </Company>
</Companies>
```

그림 1 예제 XML 문서

• **k-근접 포함질의**: 텍스트 단어간의 근접도(proximity)에 기반을 둔 질의

예제(k=3): Distance("printers","scanners") ≤ 3

"printers"와 "scanners"가 단어단위 거리로 3이내에 있는 XML 문서를 추출하라는 질의이다.

XML 정보검색 시스템 상에서의 질의의 대부분은 위의 네 가지 기본 포함질의들이 혼합된 형태가 핵심을 이룬다고 볼 수 있다. 예를 들면, 간접과 직접 포함질의가 혼합된 예로서 "/companies/company//description/'plastic'"을 둘 수 있고 간접, 직접, 그리고 완전 포함질의가 혼합된 예로 "/companies/company//address/city='Austin'"을 둘 수 있다.

3.2 포함질의 처리를 위한 역색인의 확장

3.2.1 이전 연구의 문제점

역색인은 정보검색 시스템에서 가장 널리 쓰이는 색인 방법으로 탐색 작업의 속도를 향상시키기 위해 텍스트에 대한 색인을 만들기 위한 단어기반 메커니즘이다[21]. 일반적인 역색인 구조는 어휘와 출현빈도의 두 요소로 구성된다. 어휘는 텍스트에 나타나는 모든 단어의 집합이며 각 단어에 대해 해당 단어들이 나타난 문서의 위치를 리스트 형태로 저장하는데, 이러한 리스트의 집합을 출현빈도(occurrence)라고 한다. 그리고 어떤 단어들은 색인되지 않는데 이 단어들은 정보검색 분야에서 보통 "불용어" 라고 한다. 예를 들어, 관사, 전치사, 접속사 등은 불용어의 자연스러운 후보가 된다. 그 이유는 거의 모든 문서에서 출현하고 출현 횟수 또한 매우 많으므로 그리 의미가 있는 단어들은 아니기 때문이다. 그러므로, 본 논문에서는 XML 문서에 나타나는 불용어에 대해서는 역색인화를 하지 않았고 불용어 리스트는 [22]에서 제공한 것을 사용하였다.

XML 문서에 대한 포함질의를 처리하기 위해 [2]에서는 두 개의 색인을 두었다. 하나는 텍스트 단어를 색인하기 위한 T-INDEX, 다른 하나는 엘리먼트를 색인하기 위한 E-INDEX 이다. T-INDEX는 XML문서 안의 각각의 텍스트 단어에 대해 그것이 나타나는 문서의 번호, 문서에서 단어단위로 센 위치, 문서의 루트 엘리먼트를 기준으로 한 문서 안에서의 레벨 등을 하나의 레코드로 하여 기록하므로 (term, docno, wordno, level)의 구조를 이룬다. 반면, E-INDEX는 XML문서 안의 각각의 엘리먼트에 대하여 그것이 나타나는 문서의 번호, 문서에서 단어단위로 센 시작과 끝 위치, 문서 안에서의 레벨 등을 하나의 레코드로 하여 기록하므로 (term, docno, begin, end, level)의 구조를 이룬다. 그림 2는 그림 1의 XML문서에 대해 위 방식으로 구축된 역색인의 간단한 예를 보이고 있다.

E-INDEX(term, docno, begin, end, level)
→ (companies,1,1,52,0), (company,1,2,51,1), (symbol,1,3,5,2)
(name,1,6,10,2), (profile,1,11,50,2), (address,1,12,19,3)
(state,1,13,15,4), (city,1,16,18,4), (description,1,20,49,3)
T-INDEX(term, docno, wordno, level)
→ (AAPL,1,4,3), (Apple,1,7,3), (Computer,1,8,3),
(Inc.,1,9,3), (Texas,1,14,5), (Austin,1,17,5)
(designs,1,21,4), (develops,1,22,4), (produces,1,23,4)
(markets,1,24,4), (services,1,26,4), ...

그림 2 E-INDEX와 T-INDEX

위에서 살펴본 E-INDEX와 T-INDEX를 [2]에서는 RDBMS상에서 다음의 2개의 테이블에 각각 사상(mapping)을 하였다.

Elements 테이블(term,docno,begin,end,level)
--> 기본 키(Primary Key): (term,docno,begin,end,level)
Texts 테이블(term,docno,wordno,level)
--> 기본 키: (term,docno,wordno,level)

Elements 테이블이 XML 엘리먼트의 출현에 관한 정보를 저장하고 있는 반면, Texts 테이블은 텍스트 단어의 출현에 관한 정보를 저장하고 있다.

이와 같이 역색인을 할 경우, 앞에서 살펴본 기본 포함관계는 각각 다음과 같은 조인 조건을 만족해야 한다. 우선, 엘리먼트 EL1의 발생을 (T1,D1,B1,E1,L1), 엘리먼트 EL2의 발생을 (T2,D2,B2,E2,L2), 텍스트 단어 W3의 발생을 (T3,D3,P3,L3), 그리고 텍스트 단어 W4의 발생을 (T4,D4,P4,L4)라고 각각 색인 했다고 가정한다.

1. 간접 포함

EL1이 EL2를 간접 포함하기 위한 조건을 다음과 같다.
(1) D1=D2, (2) B1 < B2, 그리고 (3) E1 > E2
EL1이 W3을 간접 포함하기 위한 조건은 다음과 같다.
(1) D1=D3, (2) B1 < P3, 그리고 (3) E1 > P3
예를 들어, 그림 2에서 (companies,1,1,52,0)는 (symbol,1,3,5,2)를 간접 포함한다.

2. 직접 포함

EL1이 EL2를 직접 포함하기 위한 조건은 다음과 같다.
(1) D1=D2, (2) B1 < B2, (3) E1 > E2, 그리고 (4) L1 = L2 - 1
EL1이 W3을 직접 포함하기 위한 조건은 다음과 같다.
(1) D1=D3, (2) B1 < P3, (3) E1 > P3, 그리고 (4) L1 = L3 - 1
예를 들어, 그림 2에서 (name,1,6,10,2)는 (Apple,1,7,3)를 직접 포함한다.

/companies/company/profile /description/'software'

companies와 company
직접 포함관계, Elements 테이블 상에서 자기조인(self-join)
조인 조건: companies.docno=company.docno and companies.begin < company.begin
and companies.end > company.end and companies.level = company.level - 1

company와 profile
직접 포함관계, Elements 테이블 상에서 자기조인
조인 조건: company.docno=profile.docno and company.begin < profile.begin
and company.end > profile.end and company.level = profile.level - 1

profile과 description
직접 포함관계, Elements 테이블 상에서 자기조인
조인 조건: profile.docno=description.docno and profile.begin < description.begin
and profile.end > description.end and profile.level = description.level - 1

description과 software
직접 포함관계, Elements 테이블과 Texts 테이블간의 조인
조인 조건: description.docno=software.docno and description.begin < software.wordno
and description.end > software.wordno and description.level = software.level - 1

그림 3 질의예제 및 조인연산

3. 완전 포함

E1.1이 W3을 완전 포함하기 위한 조건은 다음과 같다.

(1) D1=D3, (2) B1 = P3 - 1, 그리고 (3) E1=P3 + 1

예를 들어, 그림 2에서 (symbol,1,3,5,2)는 (AAPL,1,4,3)을 완전 포함한다.

4. k-근접 포함

W3과 W4가 단어 단위거리로 k이내에 있어야 하는 조건은 다음과 같다.

(1) D3=D4, (2) P4 - P3 ≥ 0, (3) P4 - P3 ≤ k

예를 들어, 그림 2에서 (printers,1,38,4)와 (scanners, 1,39,4)는 k가 3인 경우, 조인조건을 만족한다.일반적인 포함질의는 3.1절에서 살펴본 네 가지 기본 포함질의들이 혼합된 형태가 되므로 엘리먼트간의 포함관계가 포함질의의 핵심을 이루게 된다. 그러므로 엘리먼트간의 포함관계를 효과적으로 처리할 수 있는 방안을 모색해야 한다. 하지만, 위와 같은 역색인 방식은 각각의 두 개의 엘리먼트간의 포함관계를 처리하기 위해 Elements 테이블 상에서 한 번의 자기조인(self-join)을 필요로 하므로 다음과 같은 두 가지 문제점을 야기 시킨다.

우선, 포함질의의 경로길이에 비례해서 조인 횟수가 증가한다는 점이다. 여기에서 질의의 경로길이는 질의에서 포함관계의 개수를 말한다. 그림 3은 "/companies/company/profile/description/'software'"(질의경로길이=4)와 같은 질의를 처리하기 위해서 몇 번의 조인 연산이 발생하는지를 보여준다. "companies"와 "company", "company"와 "profile", "profile"과 "description", 그리고 "description"과 "software" 포함관계들을 조사해야 하므로 총 네 번의 조인 연산이 필요로 하게 된다.

Path(path, pathID)
(/companies,0)
(/companies/company,1)
(/companies/company/symbol,2)
(/companies/company/name,3)
(/companies/company/profile,4)
(/companies/company/profile/address,5)
(/companies/company/profile/address/state,6)
(/companies/company/profile/address/city,7)
(/companies/company/profile/description,8)

PathIndex(pathID, docID, begin, end)
(0,1,1,52), (1,1,2,51), (2,1,3,5), (3,1,6,10), (4,1,11,50)
(5,1,12,19), (6,1,13,15), (7,1,16,18), (8,1,20,49)
Term(term, termID)
(AAPL,0), (Apple,1), (Computer,2), (Inc.,3)
(Texas,4), (Austin,5), (designs,6), (develops,7)
(produces,8), (markets,9), (services,10)
TermIndex(termID,docID,pathID,position)
(0,1,2,4),(1,1,3,7),(2,1,3,8),(3,1,3,9)
(4,1,6,14),(5,1,7,17),(6,1,8,21),(7,1,8,22)
(8,1,8,23),(9,1,8,24),(10,1,8,26)

그림 4 4개의 역색인 리스트

두 번째 문제점은 방대한 XML문서에 대해서 위와 같은 역색인 방식이 쓰이게 될 경우, Elements 테이블과 Texts 테이블은 크기가 매우 커지게 되므로 조인 연산 시, 항상 크기가 큰 테이블간의 조인이 발생한다는 점이다. 예를 들어, 4.1 실험환경의 표 2는 XML 데이타 크기가 113MB 일 때, Elements 테이블과 Texts 테이블의 크기를 보여준다.

본 논문에서는 [2]에서 제안한 역색인 기법이 두 개의 역색인을 두므로 이 이후로 이것을 "2-INDEX 방법"으로 정의한다.

3.2.2 본 논문에서 제시한 방법

본 논문에서는 2-INDEX 방법의 문제점을 해결하고자 역색인을 다르게 확장한다. 그림 4의 예에서 보듯이 네 개의 역 색인 리스트를 유지한다. 그리고 네 개의 리스트를 RDBMS상에서 다음의 4개의 테이블에 각각 사상(mapping)을 한다.

Path 테이블(path, pathID)
--> 기본 키(Primary Key): path
PathIndex 테이블(pathID, docID, begin, end)
--> 기본 키: (pathID,docID,begin,end)
Term 테이블(term, termID)
--> 기본 키: term
TermIndex 테이블(termID, docID, pathID, position)
--> 기본 키: (termID,docID,pathID,position)

네 개의 리스트 중에 Path와 PathIndex 리스트는 XML문서 안에서 엘리먼트의 절대경로의 발생에 대한 정보를 기록하고 있다. 여기에서, 엘리먼트의 절대경로가 의미하는 바는 XML 문서를 트리로 보았을때, 문서의 루트 엘리먼트부터 시작해서 해당 엘리먼트까지의 경로 상에 있는 모든 엘리먼트들의 이름을 "/"를 구분자로 하여 합친 것을 말한다. 예를 들어, 그림 1에서 "symbol" 엘리먼트의 절대경로는 "/companies/company/symbol"이 된다. Path 리스트는 각각의 엘리먼트의 절대경로에 대해서 pathID라는 유일한 식별자를 부여하고 PathIndex 리스트는 Path 리스트에 있는 절대경로들의 발생에 관한 정보를 담고 있다. 즉, docID는 절대경로가 발생한 문서번호를 나타내고 begin과 end는 각각 XML문서 안에서의 그 절대경로의 시작 과 끝 위치를 의미한다. 참고적으로, 본 논문에서 텍스트 단어나 엘리먼트의 절대경로의 위치정보는 문자단위가 아닌 단어단위로 센 위치를 말한다.

전에도 언급했듯이, 엘리먼트간의 포함관계를 효과적으로 처리할 수 있는 방법을 모색해야 하는데 본 논문에서는 이를 위해 위와 같이 엘리먼트의 이름 대신, 절대경로를 사용한다. 엘리먼트간의 포함관계를 처리하는데 있어서 2-INDEX 방법은 각각의 두 개의 엘리먼트간의 포함관계를 처리하기 위해 Elements 테이블 상에서 한 번의 자기조인(self-join)을 필요로 하므로 엘리먼트간의 포함관계의 개수만큼 조인연산을 필요로 한다. 하지만, 본 논문의 방법은 엘리먼트간의 포함관계의 개수와 무관하게 Path와 PathIndex 테이블간의 한 번의 조인만으로 처리할 수 있다. 그림 5는 여러 경우의 엘리먼트간의 포함관계가 본 논문의 방법으로 RDBMS 상에서 어떻게 처리되는지를 보여주고 있다. 여기서 주목할 점은 포함질의가 "//" 문자열을 가지고 있으면 "//"는 SQL문의 WHERE 절에서 "%/"으로 대체가 된다는 점이다.

한편, 나머지 두 개의 리스트는 XML문서 안의 각각의 텍스트 단어의 발생에 관계된 정보를 저장하기 위해서 쓰인다. Term 리스트, TermIndex 리스트가 바로 그것인데 각각의 텍스트 단어에 대해 Term 리스트에서는 termID 라는 유일한 식별자를 부여해서 관리하고 TermIndex 리스트에서는 실질적인 텍스트 단어의 발생에 관한 정보를 기록한다. 즉, docID는 단어가 나타난 문서의 번호를 나타내며 pathID는 단어가 나타난 엘리먼트의 절대경로의 식별자를 나타내고 position은 문서

/companies/company/profile/address
→Path 테이블로부터 SQL 조건문으로 path 필드 값이 "/companies/company/profile/address"인 레코드의 pathID를 얻어낸 후, 그것과 같은 값을 pathID로 갖는 레코드들을 PathIndex 테이블로부터 추출
/companies/company//city
→Path 테이블로부터 SQL 조건문으로 path 필드 값이 "/companies/company%/city"인 레코드의 pathID를 얻어낸 후, 그것과 같은 값을 pathID로 갖는 레코드들을 PathIndex 테이블로부터 추출
//company//description
→Path 테이블로부터 SQL 조건문으로 path 필드 값이 "%/company%/description"인 레코드의 pathID를 얻어낸 후, 그것과 같은 값을 pathID로 갖는 레코드들을 PathIndex 테이블로부터 추출

그림 5 RDBMS 상에서 엘리먼트간의 포함관계 처리

안에서 단어가 나타난 위치를 의미한다.

엘리먼트의 절대경로의 식별자를 TermIndex 리스트에 둔 이유는 RDBMS 상에서 포함질의 처리 시, 질의의 경로길이에 비례해서 조인횟수가 증가하지 않는 장점을 얻을 수 있기 때문이다. 그림 6은 "/companies/company/profile/description/'software'"와 같은 질의를 처리하기 위해서 몇 번의 조인이 발생하는지를 보여주고 있다. TermIndex와 Path 테이블간의 조인, 그리고 TermIndex와 Term 테이블간의 조인, 총 2번의 조인 연산이 발생한다.

방대한 XML 문서에 대해서 2-INDEX 방법으로 역색인을 구축 시, Texts 테이블과 Elements 테이블은 크기가 매우 커지게 되므로 포함질의 처리 시, 발생하는 조인 연산은 항상 크기가 큰 테이블간의 조인이 되는 단점이 존재한다. 본 논문에서는 이 문제점을 해결하고자 위와 같이 4개의 리스트로 나누어서 각각을 RDBMS의 테이블에 사상을 한 것이다. PathIndex와 TermIndex 테이블은 모든 필드가 정수형이므로 각각, Elements 테이블과 Texts 테이블에 비해서 크기가 상대적으로 작다. 4.1 실험환경의 표 2로부터 본 논문의 방법의 전체 테이블 크기가 2-INDEX 방법의 전체 테이블 크기보다 크지 않고 해당 테이블간(PathIndex와 Elements, TermIndex와 Texts)의 크기도 본 논문의 방법이 상대적으로 작음을 확인할 수 있다.

/companies/company/profile/description/'software'

I) Path 테이블로부터 "/companies/company/profile/description"의 pathID, p1을 얻어냄
II) Term 테이블로부터 "software"의 termID, t1을 얻어냄
III) TermIndex 테이블로부터 p1과 같은 값을 pathID로 가지며 동시에 t1과 같은 값을 termID로 가지는 레코드들을 추출함 (2번의 조인 발생)

그림 6 질의예제 및 조인연산

3.3 RDBMS에서의 기본 포함질의 처리

네 가지 종류의 기본 포함질의가 어떻게 SQL 문으로 변환되어 RDBMS상에서 처리되는지에 대해서 2-INDEX 방법과 본 논문의 방법, 각각을 살펴본다. 아래의 예제들은 3.1절에서 살펴본 기본 포함질의 예제를 바탕으로 한 것이다.

3.3.1 간접 포함질의

/companies//profile//'scanners' 질의를 처리하기 위해 2-INDEX 방법에서는 그림 7의 (가)와 같이 "companies"와 "profile" 포함관계를 처리하기 위해서 Elements 테이블 상에서 자기조인 한번, "profile"와 "scanners" 포함관계들을 처리하기 위해서 Elements 테이블과 Texts 테이블간의 한 번의 조인이 필요하다. 결국, 총 2번의 조인이 발생한다. 하지만, 여기에서 발생하는 조인은 본 논문의 방식에 의해 생성된 테이블에 비해서 상대적으로 크기가 큰 테이블간의 조인으로 조인에 따른 오버헤드가 발생한다.

반면, 본 논문에서는 그림 7의 (나)와 같이 Path 테이블로부터 "/companies//profile//"에 해당하는 pathID들을 얻고 Term 테이블에서 "scanners"에 해당하는 termID를 얻은 후, 그것들과 같은 값을 pathID, termID로 가지는 투플(tuple)들을 TermIndex상에서 추출하면 된다. 결국, TermIndex와 Path 테이블간의 한 번의 조인, 그리고 TermIndex와 Term 테이블간의 한 번의 조인이 발생하지만, 여기서의 조인은 위의 2-INDEX 방법에 비해서 상대적으로 작은 테이블간의 조인이 된다. 비록, 위의 질의예제는 2-INDEX 방법과 본 논문의 방식의 조인 횟수가 두 번으로 같지만, 간접 포함질의의 경로가 길어지게 되면 2-INDEX 방법은 그 만큼 비례해서 조인횟수가 증가하지만 본 논문의 방식은 조인 횟수가 질의의 경로길이에 관계없이 항상 두 번이 된다.

```
select scanners.docno
from Elements companies, Elements profile, Texts
scanners
where companies.term='companies' and
        profile.term='profile'
        and scanners.term='scanners'
        -- "companies"는 "profile"를 간접 포함한다.
        and companies.docno=profile.docno
        and companies.begin < profile.begin
        and companies.end > profile.end
        -- "profile"는 "scanners"를 간접 포함한다.
        and profile.docno=scanners.docno
        and profile.begin < scanners.wordno
        and profile.end > scanners.wordno
```

(가)

```
select TI.docID
from Path P, Term T, TermIndex TI
where P.path like '/companies%/profile%' and
T.term='scanners' and TI.pathID=P.pathID
      and TI.termID=T.termID
```

(나)

그림 7 간접 포함질의 처리

3.3.2 직접 포함질의

/companies/company/profile/description/'printers' 질의를 처리하기 위해 2-INDEX 방법에서는 그림 8의

(가)와 같이 "companies"와 "company", "company"와 "profile", 그리고 "profile"과 "description" 포함관계들을 처리하기 위해서 Elements 테이블 상에서 각각 자기조인 한 번, "description"과 "printers"의 포함관계를 처리하기 위해서 Elements 테이블과 Texts 테이블간의 한 번의 조인, 총 네 번의 조인이 발생하게 된다.

반면, 본 논문에서는 그림 8의 (나)와 같이 Path 테이블로부터 "/companies/company /profile/description"에 해당하는 pathID, p1을 얻고 Term 테이블로부터 "printers" 단어의 termID, t1을 얻은 후, TermIndex상에서 p1을 pathID로 가지면서 t1을 termID로 가지는 투플들만 추출하므로 총 2번의 조인연산만 필요로 한다. 2-INDEX 방법에 비해서 조인횟수도 줄고 조인연산 또한 상대적으로 크기가 작은 테이블간의 조인이 된다. 조인횟수도 간접 포함질의에서와 같이 질의의 경로길이에 관계없이 항상 두 번이 발생하지만 2-INDEX 방법은 질의의 경로길이에 비례해서 조인연산의 횟수가 증가한다.

```
select printers.docno
from Elements companies, Elements company, Elements
profile, Elements description, Texts printers
where companies.term='companies' and
      company.term='company'
      and profile.term='profile' and
      description.term='description'   and
      printers.term='printers'
      -- "companies"는 "company"를 직접 포함한다.
      and companies.docno=company.docno
      and companies.begin < company.begin
      and companies.end > company.end
      and companies.level=company.level-1
      ... "company" 와 "profile", "profile"과 "description"
      포함관계를 처리하기 위해 Elements 테이블 상에서
      2번의 자기조인(self-join)
      -- "description"은 "printers"을 직접 포함한다.
      and description.docno = printers.docno
      and description.begin < printers.wordno
      and description.end > printers.wordno
      and description.level = printers.level-1
```

(가)

```
select TI.docID
from Path P, Term T, TermIndex TI
where P.path='/companies/company/profile/description'
      and T.term='printers'
      and TI.pathID = P.pathID
      and TI.termID = T.termID
```

(나)

그림 8 직접 포함질의 처리

3.3.3 완전 포함질의

```
select AAPL.docno
from Elements symbol, Texts AAPL
where symbol.term='symbol' and AAPL.term='AAPL'
      -- "symbol"은 "AAPL"을 완전 포함한다.
      and symbol.docno=AAPL.docno
      and symbol.begin=AAPL.wordno-1
      and symbol.end =AAPL.wordno+1
```

(가)

```
select TI.docID
from Term T, TermIndex TI, Path P, PathIndex PI
where T.term='AAPL' and P.path like '%/symbol'
      and TI.termID=T.termID and PI.pathID=P.pathID
      -- 완전 포함관계
      and TI.docID=PI.docID
      and TI.position=PI.begin+1
      and TI.position=PI.end-1
```

(나)

그림 9 완전 포함질의 처리

//symbol='AAPL' 질의를 처리하기 위해 2-INDEX 방법에서는 그림 9의 (가)와 같이 "symbol"과 "AAPL"의 완전 포함관계를 처리하기 위해서 Elements 테이블과 Texts 테이블간의 한 번의 조인이 발생한다. 반면, 본 논문에서는 그림 9의 (나)와 같이 Term 테이블로부터 "AAPL"의 termID를 얻고 Path 테이블로부터 엘리먼트의 경로이름이 "symbol"로 끝나는 경로들의 pathID를 얻은 후, 그것과 같은 값을 termID, pathID로 가지는 투플들을 각각 TermIndex 테이블과 PathIndex 테이블로부터 추출한다. 그리고, TermIndex 테이블과 PathIndex 테이블로부터 추출된 투플들 중에 위의 완전 포함관계 조인조건을 만족하는 투플들을 추출하면 된다.

3.3.4 k-근접 포함질의

```
select printers.docno
from Texts printers, Texts scanners
where printers.term='printers'
      and scanners.term='scanners'
      -- "printers"와 "scanners"는 단어단위 거리로 3이내
      and printers.docno=scanners.docno
      and scanners.wordno - printers.wordno >= 0
      and scanners.wordno - printers.wordno <= 3
```

(가)

```
select TI1.docID
from Term T1, Term T2, TermIndex TI1, TermIndex
TI2
where T1.term='printers' and T2.term='scanners'
      and TI1.termID = T1.termID
      and TI2.termID = T2.termID
      -- "printers"와 "scanners"는 단어단위 거리로 3이내
      and TI1.docID = TI2.docID
      and TI2.position - TI1.position >=0
      and TI2.position - TI1.position <= 3
```

(나)

그림 10 k-근접 포함질의 처리

Distance("printers","scanners") ≤ 3 질의를 처리하기 위해 2-INDEX 방법에서는 그림 10의 (가)와 같이 "printers"와 "scanners"간의 인접 포함관계를 처리해야 하므로 "Texts" 테이블 상에서 한 번의 자기조인 연산을 필요로 한다. 한편, 그림 10의 (나)의 SQL문에서 보듯이 본 논문의 방법은 완전 포함질의에서와 같이 세 번의 조인연산을 필요로 한다.(T1과 TI1간의 조인, T2와 TI2간의 조인, 그리고 TI1과 TI2간의 조인)

비록, 완전 포함질의와 k-근접 포함질의의 경우, 2-INDEX 방법은 한 번의 조인이 필요하고 본 논문의 방식은 세 번의 조인이 필요하지만, 다른 기본 포함질의와 혼합된 경우, 예를 들어, "/companies/company/profile/address/state='California'"와 같은 질의를 처리하기 위해서 2-INDEX 방법은 다섯 번의 조인이 필요하지만 본 논문의 방식은 경로길이에 관계없이 항상 세 번의 조인만 발생한다.

4. 실험 및 분석

본 논문에서 제안한 방식대로 확장된 역색인을 RDBMS를 이용해서 저장하고 포함질의를 처리하는 방법이 역 리스트 엔진을 이용하는 방법에 비해서 성능 상으로 거의 비슷하다는 것을 보인다. 그리고, 2-INDEX 방법의 RDBMS나 역 리스트 엔진을 이용하는 방법에 비해서 본 논문의 RDBMS를 이용하는 방법이 성능 상에서 많은 향상을 보았음을 보인다.

4.1 실험 환경

실험 환경에 대해서 말하기 전에 우선 XML 문서에 대한 역색인 정보를 저장하고 포함질의를 처리하기 위한 방법들로 본 실험에서 성능을 비교한 네 가지 방법들을 살펴본다.

4Table 방법: 본 논문에서 제안한 방식대로 그림 4와 같이 네 개의 리스트로 확장된 역 색인을 RDBMS를 이용해서 네 개의 테이블 즉, Path 테이블, PathIndex 테이블, Term 테이블, TermIndex 테이블에 저장하고 질의를 처리하는 방법

4BTree 방법: 본 논문에서 제안한 방식대로 그림 4와 같이 네 개의 리스트로 확장된 역색인을 각각 B+ 트리를 이용해서 역 리스트 엔진에 저장하고 질의를 처리하는 방법

2Table 방법: 2-INDEX 방법대로 그림 2와 같이 두 개의 리스트로 확장된 역색인을 RDBMS를 이용해서 두 개의 테이블 즉, Elements 테이블과 Texts 테이블에 저장하고 질의를 처리하는 방법

2BTree 방법: 2-INDEX 방법대로 그림 2와 같이 두 개의 리스트로 확장된 역색인을 각각 B+ 트리를 이용해서 역 리스트 엔진에 저장하고 질의를 처리하는 방법(조인연산 시, [2]에서 제안한 방법인 MPMGJN 조인을 사용)

우선, 위에서 말한 4Table 방법과 2Table 방법을 구현하기 위해 RDBMS로 Oracle8.1.7을 사용하였고 Oracle 8.1.7에서 제공하는 IOT(Index-Organized Table)를 사용하였다. 기존 관계형 테이블에서 기본 키(primary key)에 해당하는 색인과 데이타 테이블을 따로 두는 것과는 달리, IOT [23]는 이것들을 하나로 통합시킨 방식이다. 본 논문에서는 테이블의 색인을 기본 키에 대해서만 두었으므로 색인을 위한 추가적인 저장 공간이 필요하지 않았다. 한편, Oracle은 1400MHZ PIV 컴퓨터의 Windows 2000 Professional 운영체제에서 수행되었으며 컴퓨터의 메인 메모리 크기는 768 MB 이었다.

한편, 위에서 말한 4BTree 방법과 2BTree 방법을 구현하기 위해서 역 리스트 엔진을 BerkeleyDB 라이브러리[24]를 이용하여 각각 자바로 구현하였다. BerkeleyDB 라이브러리는 접근 메소드(access method)로서 B+ 트리, 확장된 선형 해싱(Extended Linear Hashing), 큐(Queue) 등을 제공하는데 여기서는 역색인을 저장하기 위한 접근 메소드로서 B+ 트리를 사용하였다. 이 역 리스트 엔진 또한 메인 메모리 크기가 768 MB인 1400MHZ PIV 컴퓨터의 Windows 2000 Professional 운영체제에서 수행되었다.

표 1은 실험에서 사용된 XML 데이타의 부류와 네 가지 방법의 역색인 구축 시 필요한 저장공간을 나타낸다. 실험 데이타는 네 가지 부류로 나뉜다. 각종 회사들에 대한 정보를 담고 있는 XML 문서들의 집합인 Companies, 자동차에 대한 카달로그 정보를 갖고 있는 XML 문서들의 집합인 Cars, Shakespeare 연극의 대본을 XML로 표현한 문서들의 집합인 Shakespeare, 그리고 DBLP XML 문서들이다. 테이블의 크기는 IOT의 크기이며 4BTree, 2BTree 방법의 역색인 크기는 Berkeley

DB B+ 트리를 이용해서 역 리스트엔진에 저장한 역색인의 총 크기를 의미한다.

표 2는 4Table 방법의 네 개의 테이블과 2Table 방법의 두 개의 테이블의 크기를 보여준다. 특히, Path 테이블의 크기가 26 KB로 다른 테이블들에 비해 상대적으로 무척 작음을 알 수 있다. 이것의 주된 이유는 Path 테이블의 크기가 XML 문서들의 크기에 비례하는 것이 아니라 그것들의 DTD(Document Type Definition) 문서의 크기에 비례하기 때문이다. DTD 문서의 크기는 일반적으로 XML 문서의 크기에 비해서 훨씬 작다. 그리고 XML 데이타의 가장 큰 특징중의 하나는 많은 XML 문서들이 일반적으로 하나의 같은 DTD 문서를 공유한다는 것이다. 결국, 4-Table 방법은 2-Table 방법에 비해 전체 테이블들의 총 크기도 작을 뿐만 아니라, 조인도 상대적으로 크기가 작은 테이블들간에 발생하게 된다. 하지만, 역색인의 크기가 XML 데이타의 크기 보다 큰 것은 어떤 응용프로그램에서는 문제가 될 수 있다. 이것을 해결하기 위해 향후 중요한 연구로서 효율적인 데이타베이스 압축 기법과 근사 역색인 기법(approximate inverted index technique)에 대해서 연구할 계획이다 [3].

4.2 실험 결과

4.1에서 살펴본 4가지 부류로 나뉘어진 XML 데이타에 대하여 각각 세 개의 질의가 수행되었다. 표 3은 성능비교를 위해서 수행된 질의 집합에 대한 정보를 보여주고 있다. 질의의 경로길이가 짧게는 1부터 긴 것은 5

표 1 XML 데이타 부류 및 역색인 구축 시 필요한 저장공간

	Companies	Cars	Shakespeare	DBLP
데이타크기	5 MB	19 MB	8 MB	81 MB
총 데이타크기	113 MB			
4Table 방법의 테이블 크기	190 MB			
4BTree 방법의 역색인 크기	133 MB			
2Table 방법의 테이블 크기	215 MB			
2BTree 방법의 역색인 크기	155 MB			

표 2 4Table 방법과 2Table 방법의 테이블 크기 비교

		크 기
4Table 방법	Term 테이블	25 MB
	TermIndex 테이블	110 MB
	Path 테이블	26 KB
	PathIndex 테이블	55 MB
2Table 방법	Texts 테이블	138 MB
	Elements 테이블	77 MB

표 3 실험에 사용된 질의 집합

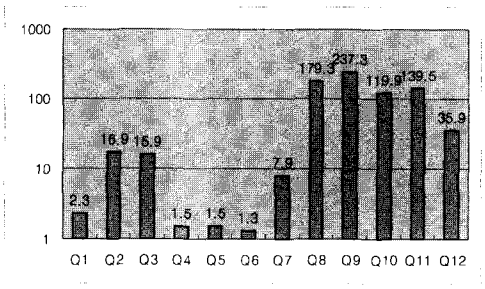
		질 의 문
Companies	Q1	/companies/company//'asphalt'
	Q2	/companies/company/profile/description//'car'
	Q3	/companies/company/profile/address/city='milwalkee'
Cars	Q4	/cars//'coupe'
	Q5	/cars/class/'convertible'
	Q6	/cars/drivetype='fwd'
Shakespeare	Q7	/play/act/scene//'exeter'
	Q8	/play/act/scene/speech/line/'love'
	Q9	/play/act/scene/speech/speaker='salarino'
DBLP	Q10	/article//'buneman'
	Q11	/article/journal/'artificial'
	Q12	/article/month='april'

까지 다양하게 선택되었고 XML 문서상에서 엘리먼트의 발생 빈도 및 텍스트 단어의 발생 빈도의 여러 조합을 고려하여 만든 것이어서 특정 질의 패턴으로 한정되는 경향에 대한 가능성을 배제하였다. 표 4는 네 가지 방법의 질의 수행시간을 각각 비교한 것이다. 참고적으로 수치의 단위는 msec 이다. 그림 11은 4Table 방법에 대한 2Table, 2BTree, 그리고 4BTree 방법의 질의 수행시간의 성능비율을 표 4를 토대로 각각 그래프로 도식화 한 것이다.

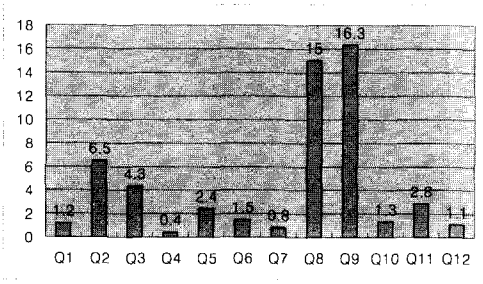
그림 11(가)로부터 2Table 방법은 4Table 방법에 비해 질의의 경로길이가 긴 경우(Q2,Q3,Q7,Q8,Q9) 뿐만 아니라, 질의의 경로길이가 짧은 경우(Q1,Q4,Q5,Q6, Q10,Q11,Q12)에도 성능이 안 좋음을 알 수 있다. 그 이유는 질의의 경로가 짧은 경우라도 4Table 방법에 비해 상대적으로 크기가 큰 테이블간의 조인이고 질의의 경로가 길어지게 되면 비례적으로 테이블간 조인횟수가

표 4 질의수행시간 비교

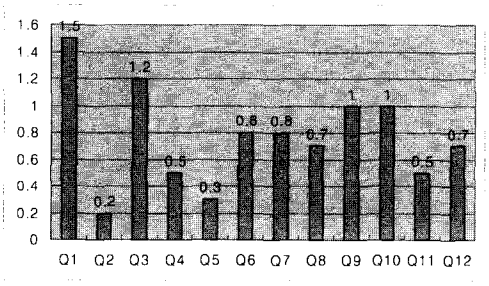
XML 데이타	질의	4Table	4BTree	2Table	2BTree
Companies	Q1	90	140	210	106
	Q2	150	30	2533	982
	Q3	250	290	3976	1072
Cars	Q4	220	110	230	80
	Q5	60	20	90	141
	Q6	90	70	120	131
Shakespeare	Q7	60	50	471	50
	Q8	230	150	41229	3461
	Q9	60	61	14240	980
DBLP	Q10	40	40	4797	51
	Q11	190	90	26508	540
	Q12	140	100	5037	153



(가) 2Table방법/4Table방법 성능비율(로그 단위)



(나) 2BTree방법/4Table방법 성능비율



(다) 4BTree방법/4Table방법 성능비율

그림 11 성능 비율

증가하기 때문이다. 반면, 4Table 방법은 테이블들간의 조인횟수가 질의의 경로길이에 비례해서 증가하지 않고 조인연산 시에도 상대적으로 크기가 작은 테이블간의 조인이므로 질의수행시간에 있어서 큰 증가는 발생하지 않는다.

2Table 방법이 4Table 방법에 비해 성능차이가 많이 나지 않는 질의의 특성은 질의의 경로길이가 2이하면서 조인에 참가하는 투플의 수가 대략 10,000 이하인 경우이다. 위의 수행된 질의 중에서 Q1, Q4, Q5, 그리고 Q6이 바로 그런 예이다. 반면, 질의의 경로길이가 2이상인 경우(Q2,Q3,Q7,Q8,Q9), 또는 질의의 경로길이가 2이하라도 조인에 참가하는 투플의 수가 대략 10,000 이상인 경우(Q10,Q11,Q12)에는 2Table 방법이 4Table 방법에 비해 성능이 많이 떨어지게 된다.

그림 11(나)로부터 2BTree 방법이 4Table 방법에 비해 대부분의 질의에 있어서 성능이 안 좋음을 알 수 있다. 특히, 질의의 경로길이가 2이상인 경우는 대부분 성능차이가 비교적 많이 난다. 위의 예에서 Q2, Q3, Q8, 그리고 Q9등이 그런 경우이다. 반면, 질의의 경로길이가 2이하인 경우는 대부분 성능차이가 두 배 이내거나 몇몇 경우(Q4,Q7)에는 2BTree 방법이 성능이 더 좋음을 알 수 있다.

[2]에서 실험을 통해 보인바와 같이 2BTree 방법이 대부분의 질의에 있어서 2Table 방법 보다 성능이 좋다는 것이 본 논문에서도 다시 한 번 확인된다. 이것의 주된 이유로 [2]에서는 두 가지를 들었다. 하나는 역 리스트 엔진에서 사용된 조인방법이 RDBMS에서 사용된 조인방법보다 성능이 더 좋았고 다른 하나는 하드웨어 캐쉬 사용률에 있어 역 리스트 엔진이 RDBMS 보다 좋다는 점이다.

하지만, 본 논문에서는 그림 11(다)에서 보듯이 4Table 방법이 4BTree 방법에 비해 성능 상으로 많은 차이가 나지 않는다. 그 이유는 본 논문에서 역색인을 확장한 방식은 조인횟수가 질의의 경로길이에 비례해서 증가하지 않고 또한, 조인발생 시, 2Table방법에 비해 상대적으로 작은 테이블간의 조인이므로 조인연산으로부터 발생하는 RDBMS와 역 리스트 엔진간의 성능상의 차이를 감소시키기 때문이다.

그림 12는 질의의 경로길이가 증가할수록 4Table 방법과 2Table 방법의 질의수행시간의 변화를 보여준다. 그림 8에서 보듯이 2Table 방법은 질의의 경로가 길어질수록 질의수행시간이 갑자기 증가하는 경향을 보이고 반면, 4Table 방법은 질의의 경로길이에 관계없이 질의수행시간이 거의 일정함을 알 수 있다.

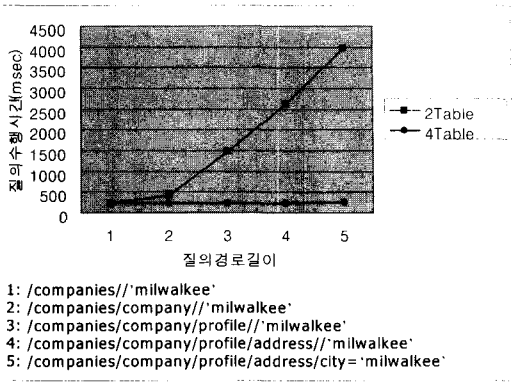


그림 12 질의의 경로길이에 따른 질의수행시간 변화

5. 결 론

본 논문에서는 XML 문서에 대한 포함질의를 효과적으로 처리하기 위해 RDBMS를 이용해서 역색인을 저장하고 포함질의를 처리하는 방법이 역 리스트 엔진을 이용하는 방법에 비해서 성능 상으로 거의 비슷하다는 것을 보였고 또한, 기존의 2-INDEX 방법에 비해서 성능 상으로 많은 향상을 보았음을 보였다. 이것을 가능하게 하기 위해 기존의 정보검색 분야에서 많이 쓰이는 색인 기법인 역색인을 XML 문서에 맞도록 2-INDEX 방법에서 확장한 방식과는 다르게 확장해서 질의처리속도에 있어서 많은 향상을 보았다. 그러므로 본 논문의 방식대로 확장된 역색인을 RDBMS를 이용해서 저장하고 질의를 처리하는 방식이 충분히 XML 정보검색 시스템에서 사용될 수 있다.

앞에서 XML문서를 저장하고 질의를 처리하기 위해 RDMBS를 이용하는 것에 대한 장점들에 대해서 언급하였다. 기관이나 기업들은 머지않아 그들의 문서 포맷으로 XML를 채택하게 될 것이고 RDBMS가 XML 문서를 저장하고 질의를 처리하는데 있어 중추적인 역할을 하게 될 것이다. 하지만, 역색인의 크기가 XML 데이타의 크기보다 크게 되는 점은 문제가 될 수 있다. 그러므로 역색인의 크기를 줄이기 위해 효율적인 데이타베이스 압축 기법과 근사 역색인 기법에 대해 향후 중요한 연구로 다룰 것이다.

참 고 문 헌

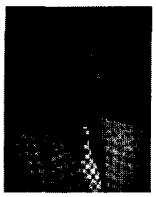
[1] T. Bray, J. Paoli, and C. Sperberg-McQueen, "Extensible markup language(XML) 1.0", Technical report, W3C Recommendation, 1998.

[2] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, "On supporting containment queries in relational database management systems", ACM

SIGMOD, 2001.
[3] Daniela Florescu, Donald Kossman, and Ioana Manolescu, "Integrating keyword search into XML query processing", WWW9/Computer Networks, 33(1-6):119-135, 2000.
[4] Takeyuki Shimura, Masatishi Yoshikawa, and Shunsuke Uemura, "Storage and retrieval of XML documents using object-relational database", Dexa, pp. 206-217, 1999.
[5] Daniela Florescu and Donald Kossman, "Storing and querying XML data using anrdbms", IEEE Data Engineering Bulletin, 22(3):27-34, 1999.
[6] Jayavel Schanmugasundaram, He Gang, Kristin Tuffe, Chun Zhang, David DeWitt, and Jeffrey Naughto, "Relational database for quering XML documents: limitation and opportunities", VLDB, 1999.
[7] Jayavel Schanmugasundaram, E. Schekita, R.Barr, Michael J. Carey, Bruce G.Lindsay, Hamid Pirahesh, and Berthold Reinwald, "Efficiently publishing relational data as XML documents", VLDB, 2000.
[8] Alin Deutsch, Mary F. Fernandez, and Dan Suciu, "Storing semistructured data with STORED", ACM SIGMOD, 1999.
[9] International Organization for Standardization, "Information processing-text and office systems -standard generalized markup language(sgml)", iso/iec 8879, 1986.
[10] Ricardo Baeza-Yates and Gonzalo Navarro, "Integrating contents and structure in text retrieval", SIGMOD Record, 25(1):67-69, March 1996.
[11] G. E. Blake, M. P. Consens, P. Kilpelainen, P. A. Larson, T. Snider, and F. W. Tompa, "Text/relational database management system: Harmonizing sql and sgml", In Proceedings of the International Conference on Applications of Database, pages 267-280, June 1994.
[12] Ron Sacks-Davis, Timothy Arnold-Moore, and Justin Zobel, "Database systems for structured documents", In Proceedings of the International Symposium on Advanced Database Technologies and Their Integration, pages 272-283, October 1994.
[13] T. Arnold-Moore, M. Fuller, B. Lowe, J. Thom, and R. Wilkinson, "The elf data model and sgql query language for structured document database", In Proceedings of the Australasian Database Conference, Adelaide, Australia, pages 17-26, 1995.
[14] C. L. Clarke, G. V. Cormack, and F. J. Burkowski, "An algebra for structured text search and a framework for its implementation", The Computer Journal, 38(1):43-56, 1995.
[15] C. L. Clarke, G. V. Cormack, and F. J. Burkowski, "Schema-independent retrieval from heterogeneous structured text", In Fourth Annual Symposium on Document Analysis and Information Retrieval, pages 279-289, 1995.
[16] Tuong Dao, Ron Sacks-Davis, and James A. Thom, "Indexing structured text for queries on containment relationships", In Proceedings of the 7th Australasian Database Conference, 1996.
[17] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu, "XQuery 1.0: An XML Query Language", W3C Working Draft, June 2001.
[18] J. Clark and S. DeRose, "XML Path Language (XPath) Version 1.0", W3C Recommendation, November 1999.
[19] J. Robie, J. Lapp, and D. Schach, "XML Query Language (XQL)", http://www.w3.org/TandS/QL/QL98/pp/xql.html.
[20] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, and Dan Suciu, "XML-QL: A Query Language for XML", World Wide Web Consortium, August 1998.
[21] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, "Moderm Information Retrieval", Addison- Wesley Longman Inc, 1999.
[22] W. B. Frakes and R. Baeza-Yates, "Information Retrieval: Data Structures & Algorithms", Prentice Hall, Englewood Cliffs, NJ, USA, 1992.
[23] Oracle, Oracle Documentation Library, Release 8.1.7, http://otn.oracle.co.kr/docs/Oracle817/index.htm.
[24] Sleepycat Software, The berkeley database, http://www.sleepycat.com.

서 치 영
2000년 충남대학교 정보통신공학과 학사. 2002년 서울대학교 전기/컴퓨터공학부 석사. 2002년~현재 ITCamp(주) 연구소 연구원.

이 상 원
1991년 서울대학교 컴퓨터공학과 학사. 1994년 서울대학교 컴퓨터공학과 석사. 1999년 서울대학교 컴퓨터공학과 박사. 1999년~2001년 한국 오라클(주) 연구원. 2001년~2002년 이화여자대학교 BK 계약교수. 2002년~현재 성균관대학교 정보통신공학부 전임강사



김 형 주
1982년 서울대학교 전자계산학과 학사. 1985년 Univ. of Texas at Austin 전산학과 석사. 1988년 Univ. of Texas at Austin 전산학과 박사. 1988년 5월~1988년 9월 Univ. of Texas at Austin. Post-Doc.1988년 9월~1990년 12월 Georgia Institute of Technology 부교수. 1991년 1월~현재 서울대학교 컴퓨터공학부 교수