博士學位論文

RDBMS에서의 효율적인 XML 부분 매치 질의 처리

濟州大學校 大學院

컴퓨터공학과

朴 忠 熙

2008年 2月

RDBMS에서의 효율적인 XML 부분 매치 질의 처리

指導敎授 李 尙 俊

朴 忠 熙

이 論文을 工學 博士學位 論文으로 提出함

2008年 2月

朴忠熙의 工學 博士學位 論文을 認准함

審査委員長 郭 鍾 榮

審 査 委 員 김 도 현

審 査 委 員 김 철 민

審 査 委 員 박 경 쾌

審 査 委 員 이 상 조

濟州大學校 大學院

2008年 2月

The Efficient XML Partial Match Query Processing on RDBMS

Chung-Hee Park

(Supervised by professor Sang-Joon Lee)

A thesis submitted in partial fulfillment of the requirement

for the degree of Doctor of Computer Engineering

2008. 2.

This thesis has been examined and approved

Thesis director, Ho-young Kwak

Thesis director, Do-Hyeun Kim

Thesis director, Cheol Min Kim

Thesis director, Gyung-Leen Park

Thesis director, Sang-Joon Lee

2008. 2.

Department of Computer Engineering

GRADUATE SCHOOL

CHEJU NATIONAL UNIVERSITY

목 차

그 림 목 차

표 목 차

국문초록

RDBMS에서의 효율적인 XML 부분 매치 질의 처리

컴퓨터공학과 박충희

지도 교수 이상준

부분 매치 질의는 경로 질의 상에 조상-자손 관계성 '//'를 가지는 질의로 정의되며, 선형 경로 질의와 분기 경로 질의로 구분된다.

본 논문에서는 상이한 구조를 가지는 대규모 문서들에 대해서 부분 매치 질의를 효과적으로 처리하기 위하여 2가지 방법을 제안한다. 제안 방식은 기존의 순수 RDBMS를 기반을 두고 새로운 인덱스 기술과 조인 알고리즘을 사용하여 질의 처리 효율성과 확장성을 개선한다.

하나는 인덱스 구축을 위해 사용된 기법으로 경로 레이블 정보를 저장함에 있어서, 기존의 연구에서 사용된 순방향 레이블 경로 대신 역방향 레이블 경로를 사용하여 경로 인덱스를 구축한다. 구축된 인덱스는 대량의 상이한 구조의 문서가 존재하는 상황에서도 전체 매치 질의는 물론 부분 매치 질의 처리 시 해당되는 레이블 경로들에 대하여 빠른 접근을 허용하여 질의 처리 성능을 개선시킨다.

또 하나는 선형 경로 질의 결과 간의 결합을 위해 사용되는 조인 알고리즘을 제안한다. 제안된 조인 알고리즘은 먼저 두 레이블 경로 간에 유효하지 않은 경로 연결을 식별하고 나서, 유효한 레이블 경로에 소속된 레이블 경로 인스턴스들만을 이용함으로써 조인 연산에 참여하는 튜플들의 비교 횟수를 줄인다. 제안 알고리즘은 단독 또는 조인 연산의 성능을 개선시키기 위하여 기존 조인 알고리즘과 함께 사용될 수 있다.

마지막으로, 제안방식의 효율성 검증을 위해 기존 연구들과의 비교 분석을 수행하였다.

ABSTRACT

The Efficient XML Partial Match Query Processing on RDBMS

Park, Chung Hee
Department of Computer Engineering
Graduate School
Cheju National University

A partial match query is defined as the one having the descendant-or-self axis '//' in its path query. It can be classified as a linear path query or a branching path query.

In this thesis, we suggest two methods for processing partial match queries efficiently on a large amount of the differently-structured documents.

One is the new index structure constructed using backward label paths instead of forward label paths used in previous researches for storing the path information. It allows for finding the label paths efficiently than the conventional methods and improves the performance of query processing.

The other is the join algorithm used for merging results of linear path queries. The proposed algorithm reduces the number of the comparison of tuples that participate in join operation firstly by identifying invalid connections among the two label paths and then, using only abel path instances belong to valid path-connections. It can be used alone or to improve the performance of the join operation with the conventional join algorithm.

Finally, We demonstrated the efficiency of the proposed method by comparing it with the conventional methods.

I. 서 론

1. 연구 배경 및 목적

XML[1]이 인터넷 상에서 데이타 표현 및 교환을 위한 실질적 표준 언어로 자리를 차지함에 따라 인터넷(XHTML)뿐만 아니라 건강, 은행, 화학, 통신 산업 등 많은 기관과 그룹에서 XML을 적용하고 있으며, 심지어 비즈니스에서 필요로 하는 많은 문서들과 웹 사이트들의 내부 문서들도 XML 형식으로 변화하고 있다. 이러한 상황에서 유사한 정보를 가진 대규모의 XML 문서들은 서로 상이한 구조를 가지며 이들로부터 필요한 정보를 빠르게 추출하는 것은 매우 중요한 이슈가 되고 있다[2].

현재, XML 문서를 관리하기 위한 접근법은 세 가지로 유형으로 구분될 수 있다. 첫 번째 접근법은 네이티브 XML 데이타베이스 시스템을 사용하는 방식으로 이 접근법은 전형적으로 트리 혹은 그래프 표현을 가지는 XML 문서에 대해 좀 더 자연스러운 데이타 모델과 질의 언어를 제공할 수 있다. 두 번째 접근법은 XML 데이타와 질의를 관계 RDBMS에 의해 제공되는 구조에 매핑하는 것이다. XML 데이타는 릴레이션에 매핑되고, XML 데이타에 대한 질의는 SQL 질의로 변환되며, SQL 질의 결과는 사용자에게 반환되기 전에 XML 문서로 변환된다. 세 번째 접근법은 객체관계 DBMS를 사용하는 방법이다. 현재 첫 번째와 두 번째 방식에 대하여 연구가 주로 이루어지고 있다[3].

네이티브 XML 시스템을 이용하는 방식과 달리 관계 데이타베이스를 이용하는 방식은 XML 데이타를 관계 데이타로 매핑과정에서 많은 릴레이션을 가지게 되며 이는 질의 평가시 비용이 큰 조인을 잠재적으로 많이 유발한다. 이러한 단점에도 불구하고 XML 문서들을 저장하는 백엔드(back-end)로서 대량의 정보처리가 가능하고 회복, 동시성 제어 등의 성숙된 기술들을 활용할 수 있으며 기존 레거시 데이타베이스(legacy database)와의 통합 질의 처리가 가능한 장점으로 인하여 최근에 XML 문서들에 대한 질의 처리를 위해 의미있는 연구들이 진행되어 왔다

[2,4,5,6,7,8,9,10].

한편, XML 질의 언어로 XML-QL[12], XQL[13], Quilt[14] 등 다양한 XML 질의 언어들이 제안되어 왔으며 최근에 W3C에서 XQuery[15]를 표준언어로 채택하였다. 이들 질의 언어들은 계층적인 문서의 구조 검색을 위해 XPath[16]와 같은 경로식 형태의 질의어를 기본으로 삼고 있다[17]. 따라서 XML 데이타베이스에서 트리 패턴의 발생정보(occurrence)를 찾는 작업은 XML 질의 처리를 위한 핵심 연산이다[18].

XPath의 경로 질의 상에 부모-자식 관계성 '/'만을 가지는 질의를 전체 매치 질의(full match query)라고 부르고, 부모-자식 관계성 '/'외에 조상-자손 관계성 '//'를 가지는 질의를 부분 매치 질의(partial match query)라 한다. 이러한 부분 매치 질의는 서로 상이한 구조를 가지는 문서들이 존재하는 상황에서 스키마 정보를 부분적으로 알고 있을 때 질의 대상 XML 문서를 찾는 데 특히 유용하다[9].

XML 문서들에 대한 질의 처리를 위해 많은 연구들이 진행되어 왔으나 부분 매치 질의를 위해 적합하게 디자인되지 않았거나 제한적으로만 지원하고 있다. XRel[5], XParent[6,7]는 한 개의 대규모 XML 문서나 동일 구조를 가진 여러 XML 문서들에 대해 효율적인 경로 검색 지원을 목표로 하고 있다. EPIS[2]는 비용이 큰 세타 조인을 이용하여 질의를 처리하고 있으며, XIR-Branching[10]은 off-the-shelf DBMS가 아닌 정보검색(IR) 분야의 역 인덱스 기술을 활용한 확장 방식을 채택하고 있다.

본 논문은 off-the-shelf 관계 데이타베이스 위에 XML 데이타베이스를 구축하는 접근법을 사용하였으며 다음과 같은 목표 시스템을 기반으로 한다.

- 저장된 XML 문서에 제약이 없음: 유효하거나 well-formed 문서 저장
- XML 문서의 저장과 검색은 현재의 데이타베이스의 기능만을 이용: 데이타 모델, 질의 언어, 인덱스에 대한 확장은 없음
- 상이한 구조를 가지는 대량의 XML 문서 응용에의 적응성
- 부분 매치 질의의 효율적 처리
- XPath 질의 지원

이러한 목표 시스템을 미들웨어 형식으로 구현한 접근법은 상업용 데이타베이스

에 바로 적용 가능하다는 장점을 가지고 있다.

본 논문의 목적은 상이한 구조의 문서가 존재하는 상황에서 off-the-shelf 관계 데이타베이스를 기반으로 하여 부분 매치 질의를 효과적으로 처리할 수 있는 질의 처리 방법을 제안하는 것이다.

2. 연구 내용 및 논문 구성

본 논문은 off-the-shelf 관계 데이타베이스 기반으로 부분 매치 질의를 효과적으로 수행하기 위하여 다음과 같이 5단계로 나누어 연구를 수행하였다.

첫 번째, XML 데이타베이스에서 질의 처리를 위한 기법들에 대해 분석을 수행하였다. 스택 계열 조인 알고리즘들[18,19,20,21,22,23,24]은 XML 전용 시스템을 목표 시스템으로 하는 질의 처리 방법에 초점을 맞추고 있어서, 관계 데이타베이스에서 적용해온 질의 처리 방식 위주로 분석을 수행하였으며 이에 대한 내용은 관련 연구에서 언급한다.

두 번째, XML 경로 질의 처리 과정은 일반적으로 하나의 분기 경로 질의를 여러 개의 선형 경로 질의로 분할하고, 각각의 분할된 선형 경로 질의에 소속된 인스턴스(노드) 집합을 구한다. 그리고 주어진 분기 경로 질의의 최종 결과를 얻기 위하여 이들 선형 질의 처리 결과를 비교, 결합하는 방식을 따르고 있다.

따라서 본 논문에서는 빠른 선형 경로 질의 처리를 위해 XML 인덱싱 기법을 설계하고 제안한다. 기존의 연구에서 사용되고 있는 XML 인덱스에 비해 부분 매치 질의를 효과적으로 지원함을 실험을 통해 타당성을 검증해 보인다.

세 번째, XML 질의 처리는 최종 질의 결과를 얻기 위하여 분할된 질의 결과를 결합하는 과정에서 조인을 수반한다. 기본적으로 조인은 비용이 큰 연산이다. 본 논문에서는 기존 연구에서 수행하고 있는 조인 알고리즘과는 다른 새로운 조인 알고리즘을 제안하였다. 이 조인 알고리즘은 XRel[5]에서 사용된 BEL 값을 이용한 세타 조인이나 XParent[6,7]의 조상 테이블을 이용한 이퀴 조인과 달리 노드 식별자로 프리픽스 레이블링 방식을 사용하고 있다. 최근에 연구된 XIR-Branching[10]은 정보기술(IR) 분야의 역 인덱스를 이용하여 확장된 관계 데이타베이스에서 프리픽스 레이블링 방식을 이용한 조인 방식을 채택하고 있으며 이 방식은 앞선 두 방식에 비해 조인 횟수를 감소시켰다. 제안방식은 조인을 수행하기에 앞서 경로 질의

파싱 단계에서 추출한 정보와 저장된 레이블 경로의 레벨 정보를 가지고 분할된 선형 경로 질의에 해당하는 레이블 경로들 간의 결합 가능한 경로들을 식별한다. 이와 같은 유효 레이블 경로 연결 집합을 식별함으로써 선형 경로 질의 결과를 결합하는 과정에서 최종 결과에 포함되지 않는 불필요한 경로 결과들 간의 비교를 회피하는 방식을 채택하였다. 기존 조인 알고리즘들이 분할된 선형 경로 질의에 해당하는 레이블 경로 집합에 소속된 인스턴스들 전체에 대해 비교를 수행하는 반면 제안 방식은 분기 경로 질의의 문맥상 결합 가능한 레이블 경로 쌍을 인스턴스들 간 비교에 앞서 판정한다. 그리고 조인 과정에서 대응되는 레이블 경로에 소속된 인스턴스 들과의 비교만을 수행함으로써 분기 경로 질의 결과를 구하기 위해 발생하는 튜플 간의 비교 량을 감소시켰다.

네 번째, 제안된 XML 인덱스 구조와 조인 알고리즘의 지원을 위해 XML 문서 저장을 위한 시스템을 설계하였다. 제안된 시스템의 저장 구조는 XML 문서 검색 시 빠른 추출을 위해 통합된 구조를 채택하여 테이블 수를 줄였다.

마지막으로 기존 연구들과의 성능 평가를 통해 제안방식이 대규모의 상이한 XML 문서를 가지는 응용에서도 부분 매치 질의를 효과적으로 지원할 수 있음을 보였다.

XML 질의 처리 방법과 관련하여 본 논문에서 비교 연구를 수행한 범주를 그림 1에 나타내었다.

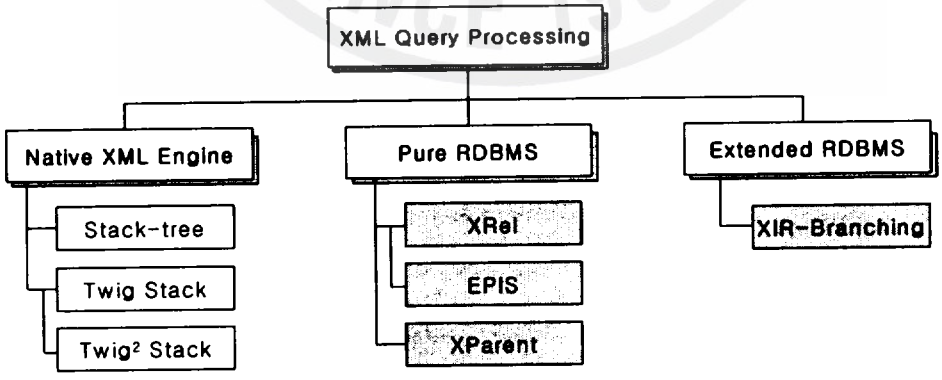


그림 1. XML 질의 처리 시스템 범주

Fig. 1. Category of the XML query processing system

본 논문의 구성은 다음과 같다.

제 2장은 XML 질의 처리 시스템 설계를 위한 관련 연구로서 먼저 XML 질의 처리를 위한 인덱싱 기법에 대해 살펴본다. 그리고 XML 데이타 노드를 식별하기 위한 노드 식별자 부여 기법인 노드 레이블링 기법(node labeling scheme)을 기술하고, XML 문서 저장을 위한 여러 데이타베이스 스키마를 고찰한다. 마지막으로 조인 알고리즘을 중심으로 XML 질의 처리 과정을 기술한다.

제 3장에서는 부분 매치 질의를 지원할 수 있는 XML 질의 처리 시스템을 제안한다. 먼저 제안 시스템에서 사용하는 문서 모델과 질의 모델을 소개한다. 그리고 효과적인 선형 경로 질의 처리를 위한 XML 인덱싱에 대해 제시하고, 끝으로 XML 문서를 저장하기 위한 시스템 구조 및 데이타베이스 스키마를 기술한다.

제 4장에서는 제안된 질의 처리 시스템의 XML 조인 알고리즘을 제시한다. 먼저 타 연구의 XML 조인 알고리즘을 분석하고 제안 방식의 기본 아이디어를 소개한다. 그리고 불필요한 데이타 경로들 간의 비교를 회피할 수 있는 조인 알고리즘을 설명한다.

끝으로 제안한 방법들을 이용하여 질의 처리하는 과정을 예를 들어 설명하고, 주어진 XML 질의를 수행하기 위한 변환된 SQL 문장을 기술한다.

제 5장에서는 제안 방식의 성능 평가를 위한 실험에 대해 설명하고 실험 결과를 제시한다. 먼저 XML 인덱싱의 성능 분석을 위해 제안된 방식과 XRel[5], XParent[6. 7], EPIS[2]의 인덱싱 기법을 비교하였으며, 다음으로 XRel[5], XIR-Branching[10]과의 조인 알고리즘 성능을 비교하였다.

끝으로 제 6장에서 결론과 앞으로의 연구 방향에 대해 기술한다.

II. 관련 연구

관계 데이타베이스를 이용한 XML 질의 처리 시스템에서의 질의 처리는 먼저 ① 분기 경로 질의를 분기 조건을 갖지 않는 여러 개의 선형 경로 질의로 분할한다. ② 그리고 나서 각각의 선형 경로 질의에 대해 질의 결과를 구한다. ③ 최종적으로 분기 경로 질의에 대한 결과를 구하기 위해 전 단계에서 구해진 질의 결과를 결합하는 과정을 수행한다[24]. 따라서 효율적인 XML 질의 처리를 위해 빠른 선형 경로 질의 처리방법에 관한 연구와 선형 경로 질의 결과를 효과적으로 결합하여 최종 질의 결과를 구하는 연구, 그리고 마지막으로 이런 과정에서 발생되는 중간 결과의 크기를 최소화하는 방안에 대한 연구가 이루어지고 있다[19]. 한편, 선형 경로 질의들 간의 효과적인 결합은 실질적으로 두 선형 경로 질의의 데이타 노드간 조상-자손 관계 식별 문제로 귀결되어진다. 따라서 데이타 노드간 조상-자손 관계 식별을 위해 기존의 연구들은 노드 식별자와 이를 이용한 조인 알고리즘을 사용한다. 본 장에서는 먼저 선형 경로 질의의 빠른 처리를 위해 사용되고 있는 XML 인덱싱 기법에 대해 살펴본다. 그리고 나서 선형 경로 질의들간 효과적인 결합을 위한 사용되는 방법의 하나인 노드 식별자를 부여하는 방법과 이를 이용한 조인 알고리즘들에 대하여 고찰한다. 이 장에서 다루는 내용은 3장과 4장에서 본 논문이 제안하는 질의 처리 시스템과 이를 기반으로 한 질의 처리 방법에 대한 기초 연구가 된다.

1. XML 인덱싱

선형 경로 질의 처리 기법은 크게 네이티브 XML 시스템에서 사용되는 방식과 관계 데이타베이스에서 사용하는 방식으로 나눌 수 있다. 네이티브 XML 시스템에서 사용하는 방식은 다시 항해 방식, 인덱스 기반 접근 방식, 정보검색(IR)의 역 인덱스 방식, 그리고 혼합 방식으로 나눌 수 있다. 관계 데이타베이스는 특별 목적의 인덱스 방식, 관계 테이블을 이용한 접근 방식, 그리고 정보검색 분야의 기술인 역

인덱스를 이용하는 방식으로 나눌 수 있다. XML 문서는 구조에 대한 정보와 노드 발생에 대한 값(인스턴스) 정보의 형태로 나누어 저장된다[10]. 네이티브 XML 엔진이나 관계형 데이타베이스는 효율적인 XML 질의 처리를 위해 XML 문서의 구조적 특성에 관한 정보를 이용하고 있으며 사용하는 정보의 레벨이 다르다. 네이티브 XML 엔진은 주로 질의 처리시 엘리먼트 태그나 노드 발생에 대한 값에 대한 인스턴스 정보-레벨만을 사용하여 질의를 처리하고 있으며, 관계형 데이타베이스는 질의 처리시 먼저 경로에 대한 정보-레벨을 사용하여 질의를 처리하고, 이후 값 인스턴스 정보-레벨을 사용하고 있다. 본 절에서는 네이티브 엔진에 사용되는 인덱스를 포함한 다른 경로 처리 기법을 간략히 설명한 후 제안방식의 기본 구조가 되는 관계형 테이블을 사용하는 인덱스 구조 중심으로 설명한다.

1) 네이티브 XML 시스템

네이티브 XML 시스템에서의 경로 질의 처리를 위한 방식에는 네 가지 다른 방법이 연구되었다. 첫째는 XML 트리 항해 방식으로 경로 질의를 유한 오토마타를 이용하여 상태 머신으로 변경하고, 상태 머신을 따라 XML 트리를 항해하는 방법으로 경로 질의를 처리한다[25,26,27]. 둘째는 XML 트리의 각 데이타 노드들을 위하여 태그-레벨 정보로 태그나 단말 값을 사용하는 방법으로 먼저 질의 처리 이전에 질의 노드에 해당하는 데이타 노드의 빠른 추출을 위해 태그 인덱스를 구축한다. 경로 질의를 이진 구조적 관계로 분할한 후 태그 인덱스를 이용하여 구조적 조인을 수행, 질의를 처리한다. 이 방식의 연구들로는 Path Join[4], Multiple Predicate Merge Join[11], Structural Join[19], Holistic Twig Join[18], Holistic Twig Join[20]의 변형, Twig^{2}Stack Join[24] 등이 있다. 이 방법들은 태그 인덱스를 사용함으로써 부분 매칭 질의를 처리할 수 있다는 장점을 가지고 있다. 그러나 이 방법은 경로 정보 대신 태그 정보만을 사용함으로써 경로 질의를 구성하는 레이블 노드에 대응되는 모든 데이타 노드를 추출하여 이들을 서로 조인한다. 따라서 실제 레이블 경로 상에 존재하지 않는 레이블 노드에 대응되는 많은 불필요한 데이타 노드들을 추출하게 되며, 이로 인해 조인 연산을 수행하는 과정에서 많은 비교 연산을 초래한다. 따라서 문서의 구조 정보를 매우 효과적으로 활용하여 불필요한 데이타 노드들을

거르는 경로-레벨 방법들에 비해 질의 처리 비용이 커지는 경향이 있다. 이들 중 Holistic Twig Join[18], Holistic Twig Join[20]의 변형, Twig^{2}Stack Join[24]은 스택을 사용하여 질의 처리시 발생하는 중간 결과를 줄였으며, 특히 트윅 질의를 구하기 위하여 사용되지 않는 유효하지 않은 데이타 경로들을 제거함으로써 비교 연산을 감소시킨다.

세 번째 방법은 정보 검색(IR) 기술의 역 인덱스를 사용하여 태그나 값에 대한 인덱스를 구축하여 사용하는 방법이다. 기존 연구로는 XQuery/IR[28], XRANK[29], Keyword Search on XML-QL[30] 등이 있다. 역 인덱스들은 태그-레벨 정보를 사용하여 생성되며 XQuery/IR, XRANK는 XML 문서 내에 존재하는 값들에 대해 생성된다. Keyword Search on XML-QL는 값들뿐만 아니라 태그(즉 엘리먼트와 속성)들에 생성된다.

네 번째 방법은 혼용하는 방법이다. 이에 대한 연구로 Mixed Mode[31], TJFast [22], Virtual Cursors[23] 등이 있다. Mixed Mode는 항해 모드와 정보 검색의 역 인덱스를 이용하여 값들뿐만 아니라 태그(즉 엘리먼트와 속성)들에 생성된 역 인덱스를 이용한다. TJFast는 두 번째 방법과 항해 모드를 혼용하는 방법으로 태그 인덱스에 의해 추출된 데이타 노드들의 확장된 Deway 값에 대해 오토마타를 사용하여 유효하지 않은 데이타 경로를 거른다. 그리고 Virtual Cursor는 태그 인덱스에 추출된 데이타 노드들(Deway id 사용)에 대해 경로 인덱스를 추가로 사용하여 유효하지 않은 데이타 경로를 거른다.

2) 관계 데이타베이스

관계 데이타베이스에서 사용된는 인덱스는 문서 구조에 대한 정보로 경로 정보-레벨을 사용하며, 레이블 경로 정보가 어디에 어떻게 저장되는 가에 따라 세 가지 방법으로 나뉜다. 첫 번째 방법은 특별한 목적의 인덱스를 사용하는 방법[32]으로 레이블 경로들은 질의 내에서 그 경로가 사용되는 시점에서 동적으로 저장된다. Index Fabric[32]은 질의에서 빈번하게 발생하는 질의들 내에서 발견되는 레이블 경로들과 값들을 인덱싱하기 위해서 패트리시아 트라이(Patricia trie)를 사용한다. 그러나 Index Fabric은 부분 매치 질의를 지원하지 못한다. 또한, 상이한 구조를 가지

는 대량의 문서가 존재하는 상황에 효과적으로 트리 질의를 지원하도록 설계되지 않아 사용하기 어렵다는 단점이 있다. 두 번째 방법은 순수 관계형 테이블을 사용하는 방법으로 문서내의 레이블 경로들이 질의 처리 이전에 관계형 DBMS 테이블에 저장된다. XRel[5], XParent[6,7], EPIS[2] 등에서 사용된 인덱스들이 여기에 해당된다. 세 번째 방법은 경로 정보를 저장하고 있는 관계 테이블에 대해 정보검색(IR) 기술의 역 인덱스를 이용, 구축하는 방법으로 XIR-Branching[10]이 해당한다.

(1) XRel

XRel[5]에서 XML 문서는 구조에 관한 정보와 노드 발생에 해당하는 값 정보로 나뉘어 관계 데이타베이스 테이블에 저장된다. 이 방식은 XML 트리에서 발생하는 모든 레이블 경로들을 경로 저장 테이블에 한 개의 레이블 경로당 한 개의 튜플 단위로 저장한다. 다음은 레이블 경로를 저장하는 테이블 구조이다.

Path(label_path, label_path_id)

label_path열이 기본키로 선언되어 B+-트리 군집 인덱스가 생성된다. 데이타 노드에 대한 정보를 저장하기 위한 나머지 테이블로는 엘리먼트 데이타 노드를 저장하는 Element 테이블, 속성 데이타 노드를 저장하는 Attribute 테이블, 그리고 마지막으로 텍스트 데이타 노드를 저장하는 Text 테이블이 있다. 문서 저장 구조에 대한 내용은 2장 3절에서 추가로 살펴본다.

이 인덱스를 이용한 XRel의 선형 경로 질의 처리는 먼저 Path 테이블에 대해 SQL의 스트링 매치 연산자인 LIKE 연산자를 사용하여 경로 질의에 부합되는 레이블 경로들을 찾는다. 전체 매치 질의인 경우와 '//'로 시작하는 형태의 부분 매치 질의인 경우에 레이블 경로를 찾기 위한 디스크 검색 양이 달라진다. 전체 매치 질의인 경우 선형 경로 질의 상에 있는 모든 레이블에 대하여 레이블 노드 단위가 아닌 레이블 경로 단위로 처리를 수행함으로써 태그 인덱스를 사용하는 경우보다 빠른 처리 성능을 보인다. 그러나 부분 매치 질의인 경우에는 SQL의 패턴 매칭 연산자 LIKE의 특성으로 인하여 인덱스 존재와 관계없이 Path 테이블 내에 있는 레이블 경로들을 모두 순차적으로 읽어야 한다. XRel은 선형 경로 질의의 최종 결과 노드들을 얻기 위하여 데이타 노드들을 저장하고 있는 나머지 테이블내의 label_path_id

컬럼을 경유하여, 레이블 경로들에 소속된 데이타 노드들을 찾는다. 다음 질의 Q1은 article 엘리먼트의 자손인 editor 엘리먼트들 중 자식으로 name 엘리먼트를 찾는 선형 경로 질의(LPQ:Linear Path Query)) 예이다.

Q1: //article//editor/name

그림 2는 XRel에서 LPQ Q1을 처리하기 위해 변환된 SQL문이다. XRel SQL은 먼저, Path 테이블에 저장된 레이블 경로 문자열들과 문자열 비교를 통해서 LPQ에 해당하는 레이블 경로들의 경로 식별자 집합을 구한다. LIKE절에서, '#' 심벌은 레이블 구분자이고, '%' 심벌은 0개 이상의 문자들을 일치시키기 위한 와일드 카드이다. 결과 노드들을 얻기 위해 Path 테이블에서 구한 경로 식별자를 가지고 Element 테이블과 조인을 수행한다.

```
SELECT   e1.document_id, e1.start_position, e1.end_position
FROM     Path p1, Element e1
WHERE    p1.label_path_id = e1.label_path_id
         and p1.label_path LIKE "#%/article#%/editor#/name";
```

그림 2. LPQ Q1를 위한 XRel SQL 문

Fig. 2. The SQL statement in XRel for LPQ Q1

(2) XParent

XParent[6,7]에서 사용되는 경로 인덱스는 XRel[5]의 인덱스와 유사하다. XParent는 레이블 경로를 저장할 때 레이블 경로의 길이를 나타내는 length 열을 추가 하였다. 이 열에 저장된 정보의 사용으로 "/DBGroup/*/Name" 같이 '*'가 포함된 경로 질의의 처리가 가능해져 처리 가능한 질의 범위가 확장되었다. 다음은 레이블 경로를 저장하는 테이블 구조이다.

LabelPath(label_path, label_path_id, length)

label_path열이 기본키로 선언되고 B+-트리 군집 인덱스가 생성되어 있다. 데이타 노드에 대한 정보를 저장하기 위한 나머지 테이블로는 엘리먼트 데이타 노드를 저장하는 Element 테이블, 속성 데이타 노드와 텍스트 데이타 노드를 저장하는 Data

테이블이 있다. 그리고 분기 경로 질의 처리시, XRel과는 다른 방법으로 조상-자손 관계를 식별하며 이를 위해 다른 스키마 구조를 갖는다. 데이타 노드들간의 조상-자손 관계 식별을 위해 XRel의 포함 관계 조인 대신(2개의 θ-조인으로 구현) 대신 이퀴 조인을 사용하며 이를 위해 Ancestor 테이블을 유지한다. 이와 선택적으로 사용할 수 있는 DataPath 테이블은 소위 Parent 테이블이라 부르며 이는 데이타 노드들 간의 부모-자식 관계성을 유지하는 데 사용한다. 따라서 선형 경로 질의 처리는 XRel과 동일한 과정을 수행하나 분기 경로 질의 처리는 다른 방식으로 처리된다.

그림 3은 XParent에서 LPQ Q1를 처리하기 위한 변환된 SQL문이다.

```
SELECT   e1.document_id, e1.node_id
FROM     LabelPath p1, Element e1
WHERE    p1.label_path_id = e1.label_path_id
         and p1.label_path LIKE ".%/article.%/editor./name" ;
```

그림 3. LPQ Q1를 위한 XParent SQL문

Fig. 3. The SQL statement in XParent for LPQ Q1

(3) EPIS

기존의 XRel[5], XParent[6,7] 시스템의 경로 인덱스들은 전체 매치 질의에 대해서는 효과적으로 처리를 수행하나 '//'로 시작하는 부분 매치 질의에 대해서는 전체 경로 테이블을 스캔함으로써 질의 처리 성능이 저하되었다. EPIS[2]는 XRel, XParent와는 다른 방식으로 레이블 경로 정보를 저장함으로써 이 문제를 개선하였다. 다음은 레이블 경로를 저장하는 테이블 구조이다.

UniqPathElmTbl(ename, pathid, plevel)

UniqPathElmTbl 테이블은 XML 문서들로부터 추출된 레이블 경로 정보를 저장하는 테이블이다. XRel[5], XParent[6,7]와는 달리 레이블 경로를 레이블 경로 단위로 저장하지 않고 레이블 경로를 레이블 노드 단위로 분할하여 저장한다.

XML 문서로 부터 구해진 각각의 레이블 경로는 각 경로를 구성하는 레이블 이름(ename), 그 레이블이 소속된 경로 식별자(pathid), 그리고 그 레이블 경로에 속하

는 단말 레이블의 레벨 값(plevel) 형태로 분할, 저장된다. 레벨 값 plevel은 0부터 시작하며 경로의 단말 레이블에는 '#'을 붙여서 해당 레이블 경로의 단말 레이블임을 표현한다. 그리고 레이블 이름을 저장한 열에 대하여 클러스터링 인덱스를 생성하여 데이타의 순서를 물리적으로 유지한다.

예를 들어 "/book/author/name"라는 레이블 경로가 경로 식별자 3을 가지고 있다고 가정하면, 이 경로는 <"book", 3, 0>, <"author", 3, 1>, <"name#", 3, 2>와 같이 세 개의 레이블 노드 단위로 분할되어 UniqPathElmTbl 테이블에 저장된다.

그 밖의 테이블로는 엘리먼트 데이타 노드의 발생에 대한 정보를 저장하는 PathOccurrenceTbl 테이블, XML 문서에 사용된 모든 용어(속성과 텍스트 노드를 용어라 지칭하고 있음)들을 저장하는 TermTbl 테이블, 그리고 각 용어들의 발생 정보를 저장하는 TermOccurrenceTbl 테이블이 있다.

EPIS[2]에서 선형 경로 질의 처리는 먼저 경로 질의에 나타난 레이블 단위로 분할하여, 각 레이블에 대해 인덱스를 탐색하여 해당 레이블을 포함하고 있는 후보 경로 식별자들을 찾는다. 그리고 후보 경로 식별자 집합 간에 셀프 조인을 수행한다. 이때 경로 질의의 레이블 순서 및 레이블의 레벨 값 차이를 만족하는 경로 식별자들을 구한다.

문서 구조가 상이한 문서의 개수가 증가되는 상황에서 XRel, XParent의 경로 저장 테이블은 크기가 점차 커지며 이에 비례하여 부분 매치 질의 처리를 위한 테이블 탐색 비용도 길어진다.

반면 EPIS의 경로 인덱스 방식은 근본적으로 경로 질의를 레이블 노드 단위로 분할 처리함으로써, '//'로 시작하는 부분 매치 질의 처리시 '//' 존재의 영향을 제거하였다.

따라서 EPIS는 항상 전체 테이블 검색 대신 인덱스를 이용하여 검색하기 때문에 상대적으로 적은 시간을 요구하며, 상이한 구조의 문서가 증가하는 상황에서도 잘 적응한다. 그러나 경로 질의를 레이블 노드 단위로 분할 처리하는 방식을 채택함으로써 길이가 긴 경로 질의를 처리하는 경우에 레이블 개수만큼의 조인이 발생하여 응답시간이 길어지는 단점이 있다. 그림 4는 EPIS[2]에서 LPQ Q1을 처리하기 위해 변환된 SQL문이다.

```
SELECT  e1.docid, e1.startpos, e1.endpos
FROM    UniqPathElmTbl p1, UniqPathElmTbl p2,
        UniqPathElmTbl p3, PathOccurrenceTbl e1
WHERE   p1.pathid = p2.pathid and p2.pahtid = p3.pathid
        and p1.plevel < p2.plevel and p2.plevel + 1 = p3.plevel
        and p3.pathid = e1.pathid and p1.ename = "article"
        and p2.ename = "editor" and p3.ename = "name#" ;
```

그림 4. LPQ Q1를 위한 EPIS SQL문

Fig. 4. The SQL statement in EPIS for LPQ Q1

(4) XIR-Branching

XIR-Branching[10]은 EPIS와 같이 부분 매치 질의를 효과적으로 처리하기 위하여 설계된 인덱스로서 정보 검색(IR) 분야에서 대규모 문서들을 빠르게 검색하기 위해 전통적으로 사용해 온 역 인덱스 기술을 적용한다. 이러한 목적을 위해 XIR-Branching은 먼저 XML 문서 내 존재하는 모든 레이블 경로들을 추출하여 중복을 제거하고, 이를 관계 테이블에 저장한다. 이 때 레이블 경로 하나 당 하나의 튜플 단위로 저장한다. 다음으로 레이블 경로 문자열이 저장된 테이블 열에 대해 정보검색 분야의 역 인덱스를 생성한다.

XIR-Branching에서 레이블 경로를 저장하는 LabelPath 테이블 구조는 다음과 같다.

LabelPath(pid, label_path)

Inverted Index on label_path of the table LabelPath

데이타 노드에 대한 정보를 저장하기 위한 나머지 테이블로 NodePath 테이블이 존재하며 이는 데이타 노드에 대한 모든 정보를 통합 저장한다.

한편, LabelPath 테이블은 전체 XML 문서들에 포함된 모든 발생 경로들로부터 추출된 서로 다른 레이블 경로들(label_path)과 그들의 경로 식별자들(pid)을 저장한다. 경로 레이블 저장시 각 레이블 경로의 시작 레이블과 종료 레이블을 표시하기 위해 프리픽스로써 '$'와 '&'를 덧붙여 저장하며 이것은 향후 선형 경로 질의를 처

리할 때 경로 질의의 루트 레이블과 단말 레이블에 일치시키기 위한 작업이다.

LabelPath의 역 인덱스 Inverted Index는 LabelPath 테이블의 label_path 열에 대해서 구축된다. 이 때, XIR-Branching은 각 레이블 경로를 텍스트 문서로 간주하고 이들 레이블 경로 안에 있는 레이블들을 키워드로 간주한다. 전통적인 역 인덱스와 같이, LabelPath 역 인덱스 구조는 키워드(즉 레이블)와 포스팅 리스트의 쌍들로 구성된다. 포스팅 리스트 내의 각 포스팅은 pid, occurrence_count, offsets, label_path_length와 같은 필드를 가진다. 여기서 pid는 레이블의 존재하는 레이블 경로의 식별자이고, occurrence_count는 레이블 경로 내에서 그 레이블이 나타난 개수이다. offsets는 레이블 경로의 시작 위치부터 해당 레이블들의 위치까지 존재하는 레이블 개수의 집합이자 옵셋 집합이며 레이블 하나를 1로 간주한다. label_path_length는 레이블 경로 내에 존재하는 레이블들의 개수이다.

예를 들어, 레이블 경로 $chapter.chapter.section.section.section.paragraph.&paragraph에서, section의 occurrence_ count, section의 offsets, label_path_length는 각각 3, {3, 4, 5}, 7이다.

XIR-Branching에서 선형 경로 질의 처리는 먼저 선형 경로 질의를 [9]에서 정의된 변환 규칙에 근거하여, 키워드 기반의 텍스트 검색 조건으로 변환한다. 이 때 near(w) 근접 연산자(최대 w개의 키워드들만큼 떨어진 두 개의 대상 키워드들을 가진 문서들을 검색하기 위해 사용되는 연산자)를 이용한 정보 검색 표현식이 만들어진다. 변환된 정보 검색식을 이용하여 기존의 문자열 비교 대신 키워드 기반의 텍스트 검색을 수행한다.

XIR-Branching에서 사용하는 인덱스 방식은 부분 매치 질의에 대해서 기존의 XRel[5], XParent[6,7]에 비해 개선된 경로 탐색 성능을 보였다. 그러나 이 방식은 근본적으로 정보검색 기술을 기반으로 하였기 때문에 off-the-shelf DBMS에 적용하기 어렵고 확장된 모듈을 적용하는 경우에도 near(w) 근접 연사자의 지원 시스템은 제한적이다.

그림 5는 XIR-Branching에서 LPQ Q1을 처리하기 위해 변환된 SQL문이다.

```
SELECT   n1.docid, n1.nodepath
FROM     LabelPath p1, NodePath n1
WHERE    p1.pid = n1.pid  and  MATCH(p1.label_path, "article"
         NEAR(MAXINT) "editor" NEAR(1) "name" NEAR(1) "&name") ;
```

그림 5. LPQ Q1을 위한 XIR-Branching SQL문

Fig. 5. The SQL statement in XIR-Branching for LPQ Q1

2. 노드 레이블 부여 기법

XML 데이타는 트리 구조로 표현되며 XML 질의 처리는 데이타 노드들 사이의 구조 관계에 대한 정보를 필요로 한다. 기본적인 구조 관계(structural relationship)는 조상-자손 관계(ancestor-descendant relationship)와 부모-자식 관계(parent-child relatio -nship)이다. XML 데이타에서 이런 기본적인 구조 관계의 어커런스들을 발견하는 작업은 XML 질의 처리를 위한 핵심 연산이다[33]. 이런 작업을 촉진하기 위하여 구조적 인덱스와 레이블링(넘버링) 기법(labeling scheme)이 있다. dataguide[34] 같은 구조적 인덱스 접근법은 XML의 계층 구조를 통하여 순회하는 것을 도와준다. 그러나 이런 순회 비용은 크다[35]. 레이블링 기법은 XML의 두 엘리먼트 사이에 존재하는 조상-자손 관계나 부모-자식 관계를 단지 레이블 값을 비교하는 것으로 결정할 수 있도록 지원한다[33]. 그 기법은 XML 문서를 레이블(노드 번호)가 붙여진 트리로 모델하며 모든 노드는 XML 문서를 순회하는 동안 노드의 순서나 문서내 위치 등에 기반한 유일한 식별자(즉 레이블)가 부여된다. 이런 레이블들은 두 데이타 노드들 사이에 조상-자손 관계나 부모-자식 관계의 존재 여부를 결정하기 위하여 사용되어진다[33]. 레이블링 기법은 크게 구간 기반 레이블링 기법(Interval-Based Labeling Scheme)[4,8,11,33,36], 프리픽스 레이블링 기법(Prefix Labeling Scheme) [8,35,37], 그리고 소수 레이블링 기법(Prime Number Labeling Scheme)[46] 등이 있다. 대표적인 노드 번호 부여 기법은 다음과 같다.

1) 구간-기반 레이블링 기법

구간-기반 레이블 기법(Interval-Based Labeling Scheme)[4,8,11,33,36]은 각각의 데이타 노드에 레이블로 시작 위치 정보와 끝 위치 정보를 할당하며, 각 노드의 레이블은 XML 트리를 깊이-우선 방식으로 순회하는 동안 양의 정수를 순차 할당함으로써 결정된다. 이 레이블링을 사용함으로써 두 노드 사이의 조상-자손 관계나 부모-자식 관계의 구조적 관계는 쉽게 결정된다. 구간-기반 레이블링의 특징은 노드의 시작 위치와 끝 위치가 노드의 구간을 나타내며 조상 노드의 구간은 자손 노드의 구간을 포함한다. 이런 특징은 구조적 조인을 효율적으로 처리할 수 있도록 만든다. 부모-자식 관계 식별을 위해 노드의 레벨 정보를 추가로 사용한다.

만약 N1 노드 레이블이 (D1, B1:E1, L1)이고, N2 레이블이 (D2, B2:E2, L2)일 때,

① 조상-자손 관계:

D1 = D2, B1 < B2 그리고 E2 < E1 이 성립된다면 노드 N1은 노드 N2의 조상이 된다. 그 역도 성립한다.

② 부모-자식 관계:

D1 = D2, B1 < B2, E2 < E1 그리고 L1 + 1 = L2 가 성립된다면 노드 N1은 노드 N2의 부모가 된다. 그 역도 성립한다.

그림 6은 구간-기반 레이블링 기법으로 레이블이 부여된 XML 데이타 트리이다.

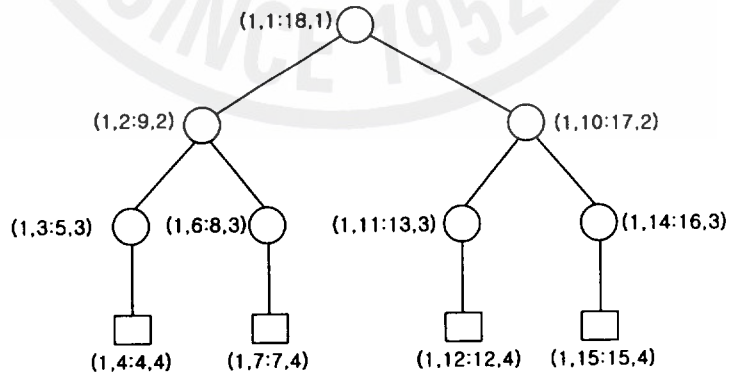


그림 6. IBL에 의해 레이블된 샘플 XML 트리

Fig. 6. The sample XML tree labeled by IBL Scheme

2) 프리픽스 레이블링 기법

프리픽스 레이블링 기법(Prefix Labeling Scheme)에서 한 노드의 레이블은 자신의 부모의 레이블에 자신의 레이블을 결합한 형태로 레이블을 할당한다. 프리픽스 레이블의 특징은 하나의 레이블이 다른 레이블의 프리픽스 여부에 따라 조상-자손의 구조적 관계가 결정되도록 레이블이 할당된다는 것이다. 따라서 임의의 노드 u의 레이블 label(u)가 노드 v 레이블 label(v)의 프리픽스이면 노드 u는 노드 v의 조상이다.

한편, 프리픽스 기법에는 정수 기반 프리픽스와 이진 문자열 기반 레이블링이 있다. Deway ID[8]는 정수 기반 프리픽스 기법으로 정수 n을 가지고 노드의 n번째 자식을 레이블로 부여한다. n번째 자식 노드는 완전한 레이블을 구성하기 위하여 자신의 정수 n을 부모의 레이블과 결합시킨다. 그림 7은 Deway ID 기법으로 레이블이 부여된 XML 데이타 트리이다.

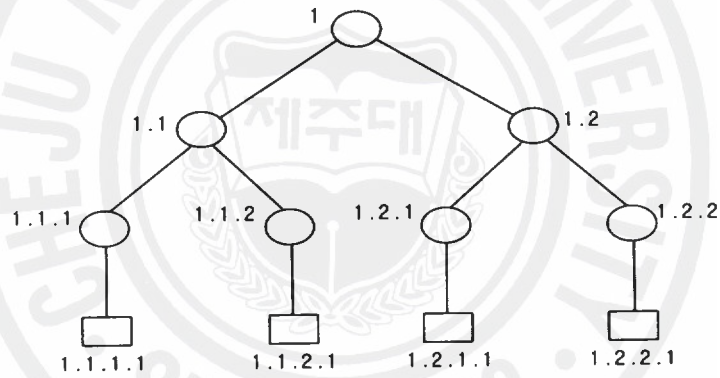


그림 7. 프리픽스 레이블링에 의해 레이블된 샘플 XML 트리

Fig. 7. The sample XML tree labeled by prefix labeling scheme

3) 소수 레이블링 기법

소수 레이블링 기법(Prime Number Labeling Scheme)은 소수의 특성에 기반하고 있다. 이 기법은 각 노드에 대해 셀프 레이블을 할당한다. 각 노드의 레이블은 부모 노드의 레이블과 그 노드의 셀프 레이블의 곱이다. 임의의 노드 u, v에 대하여 v의 레이블 label(v)를 u의 레이블 label(u)로 나누었을 때 나머지가 0이면 u는 v의 조상 노드이다.

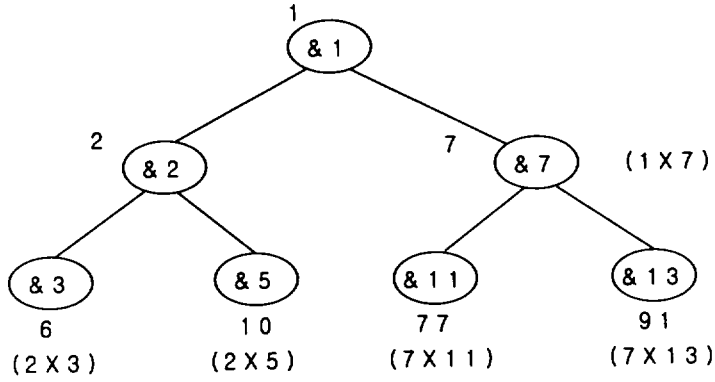


그림 8. 소수 레이블링에 의해 레이블된 샘플 XML 트리

Fig. 8. The sample XML tree labeled by prime number labeling Scheme

그림 8은 소수 레이블링 기법으로 레이블이 부여된 XML 데이타 트리이다.

레이블링 기법에는 이외에도 삽입 연산을 효과적으로 지원하기 위한 동적 레이블 기법들이 다수 연구되었다.

3. 데이타베이스 스키마

XML 문서는 기본적으로 분할되어(shred) 관계 테이블들에 저장된다[37]. 이 절에서는 앞 절에서 논의한 XML 인덱싱과 노드 식별자를 통해 XML 질의 처리를 수행하기 위한 기반이 되는 XML 문서 저장을 위한 데이타베이스 스키마에 대해 고찰한다. 먼저 관계 데이타베이스를 이용하는 XRel, XParent, EPIS에 대해 알아보고 마지막으로 확장된 관계 데이타베이스를 이용하는 XIR-Branching 방식을 살펴본다.

1) XRel

XRel[5]은 구간-기반 레이블링 기법을 이용하여 노드 식별자 값을 할당한다. XML 트리의 구조 정보는 다음과 같이 네 개의 테이블에 저장된다.

Path(label_path_id, label_path)

Element(document_id, label_path_id, start_position, end_position, sibling_order)

Text(document_id, label_path_id, start_position, end_position, value)

Attribute(document_id, label_path_id, start_position, end_position, value)

Path 테이블은 문서들 내에 있는 모든 레이블 경로들과 그들의 식별자들을 저장한다. Element 테이블은 엘리먼트 데이타 노드들에 대한 정보를 저장한다. 여기서, 각 노드에 대한 정보는 그 노드를 포함하고 있는 문서의 식별자, 그 노드에서 종료되는 레이블 경로의 식별자, 한 문서 안에 있는 그 노드의 시작과 끝 위치에 대한 옵셋(offset)들, 노드의 형제(sibling)들 가운데 해당 노드의 순번(order) 등으로 구성된다. 시작 위치와 끝 위치의 조합은 노드 식별자를 대신하여 노드를 고유하게 구분하기 위한 정보로 사용된다. Text 테이블은 텍스트 데이타 노드의 값들에 대한 정보를 저장한다. 여기서, value 컬럼은 텍스트 값들을 저장한다. Attribute 테이블은 value 컬럼에 텍스트 값 대신 애트리뷰트 값이 저장된다는 것 이외에는 Text 테이블과 같다. 이들 두 테이블의 정보는 구현에 따라 하나의 테이블에 저장될 수도 있다.

2) XParent

XParent[6,7]는 XRel과는 달리 에지-중심 접근법(edge-oriented approach)을 사용하여 식별자로 노드 식별자 대신 에지 값(단일 값)을 이용한다. 따라서 XParent의 스키마는 XRel과 유사하지만 다른 구조를 가진다. 조인 과정에서 세타 조인(θ-join)들을 사용하는 XRel과는 달리 이퀴조인(equi-join)을 사용한다. XParent의 스키마는 다음과 같다.

LabelPath(label_path_id, length, label_path)

Element(document_id, label_path_id, node_id, sibling_order)

Data(document_id, label_path_id, node_id, sibling_order, value)

Ancestor(node_id, ancestor_node_id, offset_to_ancestor)

DataPath(parent_node_id, child_node_id)

LabelPath 테이블은 XRel의 Path 테이블과 같다. Element와 Data 테이블은 영역 정보(즉, start_position, end_position) 대신, 에지 값으로 식별자를 사용한다는 점을 제외하면 XRel의 Element와 Text 테이블과 같다. Jiang 등[6,7]의 연구에서 Ancestor

테이블은 노드들 간의 조상-자손 관계성과 그들 간의 레벨 수를 유지하는데 사용하며, 이와 선택적으로 사용할 수 있는 DataPath 테이블(일명, Parent 테이블)은 노드들 간의 부모-자식 관계성을 유지하는데 사용한다. 그러나, 부분 매취 질의에 대해서는 DataPath 테이블의 사용이 적합하지 않으므로 Ancestor 테이블을 사용한다.

3) EPIS

EPIS[2]는 XRel와 동일하게 구간-기반 레이블링 기법을 이용하여 노드 식별자 값을 할당한다. 문서들의 저장을 위해 네 개의 테이블을 사용한다. 다음은 각 테이블의 구조를 보여준다.

UniqPathElmTbl(ename, pathid, plevel)

PathOccurrenceTbl(pathid, docid, startpos, endpos)

TermTbl(term, termid)

TermOccurrenceTbl(termid, docid, pathid, position)

UniqPathElmTbl 테이블은 전체 XML 문서들에 포함된 모든 데이타 경로들을 추출하여 중복을 배제한다. 이렇게 구성된 레이블 경로를 기반으로 하여, 각 경로를 구성하는 레이블 노드 단위로 분할, 저장한다. 저장되는 정보는 각 레이블의 이름(ename), 레이블이 속해 있는 경로 식별자(pathid), 레이블이 경로에 나타난 위치(plevel)이며 마지막 위치에 나타나는 레이블에는 포스트픽스로 '#'을 붙여서 경로의 마지막 레이블임을 표현한다.

PathOccurrenceTbl 테이블은 모든 엘리먼트 데이타 노드에 대한 정보를 저장하며 XRel의 Element 테이블과 유사하다. TermTbl 테이블은 XML 문서들에서 사용된 모든 용어들을 저장한 테이블로서, 용어(term)와 용어 식별자(termid)로 구성된다. TermTbl 테이블에는 서로 다른 용어들만이 저장된다. TermOccurrenceTbl 테이블은 각 용어들에 대한 발생 정보를 저장하는 테이블로서, 각 용어를 대표하는 용어 식별자(termid), 용어가 나타난 문서의 식별자(docid), 용어가 직접적으로 포함되는 경로의 식별자(pathid), 용어 발생 위치(position)으로 구성된다.

4) XIR-Branching

XML 문서는 구조 정보와 값 정보로 분리되어, 질의 처리에 필요한 정보의 형태로 관계형 데이타베이스 테이블에 저장된다. 이 때 XIR-Branching[10]은 노드 레이블링 기법으로 프리픽스 레이블 기법을 사용한다. 전체 노드에 대해 유일한 정수값을 가지고 프리픽스 레이블링 기법으로 노드 레이블을 할당한다. 문서의 레이블 경로를 LabelPath 테이블에 저장한다. 그리고 저장된 LabelPath 테이블 내 레이블 경로를 저장하고 있는 열에 대해 정보 검색(IR) 분야의 역 인덱스를 이용하여 인덱스를 구축한다. 데이타 노드를 저장하기 위하여 한 개의 테이블, NodePath 테이블만을 사용한다. 이 테이블에는 XML 트리 내에 있는 모든 인스턴스 정보를 레이블 경로와 매취되는 노드의 경로 형식으로 nodepath 컬럼에 저장한다. 다음은 XIR-Branching이 XML 문서 구조에 관한 정보를 저장하기 위하여 사용하는 두 테이블과 역 인덱스이다.

LabelPath(pid, label_path)

NodePath(pid, docid, nodepath, value)

Inverted Index : LabelPath 테이블의 label_path에 대한 역 인덱스.

인스턴스 정보는 XML 문서들 내에 있는 모든 데이타 노드들을 고유하게 식별할 필요가 있으며, 이러한 정보는 NodePath 테이블에 저장된다. NodePath 테이블은 XML 문서의 모든 노드 경로들을 nodepath 컬럼에 저장한다. 만약 노드 경로의 단말 노드가 값(value)을 가지고 있다면, 그 값은 value 컬럼에 저장한다. pid 컬럼은 동일한 레이블 경로에 속하는 모든 노드 경로들을 찾을 때, LabelPath 테이블과 조인하기 위해 사용될 레이블 경로 식별자들을 저장한다. docid 컬럼은 XML 문서 식별자들을 저장한다.

4. XML 질의 처리

XML 질의는 전형적으로 특정하게 지정된 트리 구조의 관계를 갖는 다중 엘리먼트들에 대한 선택 술어 패턴(pattern of selection predicates)을 지정한다. 기본적인 트리 구조의 관계는 부모-자식 관계(parent-child relationship)와 조상-자손 관계(ancestor-descendant relationship)이다. 그리고 XML 데이타베이스에서 이런 구조적 관계(structural relationship)에 해당하는 모든 어커런스들(occurrences)을 찾는 작업은 관계 데이타베이스를 이용한 구현이나 네이티브 XML 시스템 모두에서 XML 질의 처리를 위한 핵심 연산이다[19]. 따라서 구조적 관계에 해당하는 어커런들을 찾는 방법들은 관계 데이타베이스 시스템[38,39,40] 혹은 네이티브 XML 질의 엔진[41,42] 모두에서 많은 작업들이 이루어져 왔다.

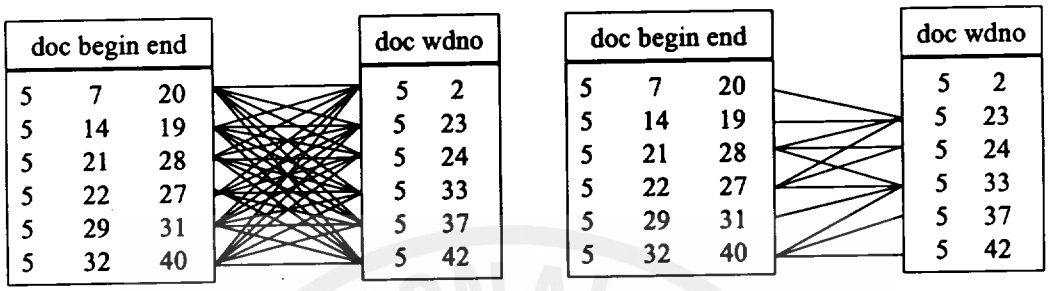
연구된 구조적 조인 알고리즘은 사용하는 노드 레이블링 기법(구간 기반 레이블 또는 프리픽스 레이블)과 조인 알고리즘의 실행 환경(관계 데이타베이스 또는 네이티브 XML 시스템)에 따라 달라진다. 먼저 네이티브 XML 시스템에서의 조인 알고리즘에 대해 간략히 살펴보고 관계 데이타베이스에서 사용되는 조인 알고리즘에 대해 자세히 고찰한다.

1) 네이티브 XML 시스템의 조인

(1) MPMGJN(Multi-Predicate Merge Join)

데이타 노드의 식별자가 IBL(시작, 끝, 레벨) 형태의 값을 가지는 경우에 대하여 두 데이타 노드간 포함관계(엘리먼트, 속성, 텍스트 노드 간)를 처리하기 위한 조인 알고리즘으로 기존의 관계 데이타베이스에서 사용해 온 표준 머지 조인(standard merge-join)을 확장하였다. 기존 표준 머지 조인은 조인 술어(join predicate)로 하나의 동등 조인 조건(equality join condition)만을 허용한다. MPMGJN[11]은 IBL에 기반하여 조상-자손 혹은 부모-자식 관계를 결정하기 위하여 다중 비동등 조인 조건(inequality join condition)을 허용한다. 이 조인 알고리즘은 포함 관계 질의를 처리함에 있어 RDBMS에 사용되고 있는 기존의 표준 머지 조인이나 인덱스된 다중 루프 조인(indexed nested-loop join) 알고리즘에 비해 우수한 성능을 보인다. 그림 9는

"t1.docno = t2.docno AND t1.begin < t2.wordno AND t2.wordno < t1.end" 라는 조인 술어 조건에 대해 기존의 표준 머지 조인과 MPMGJN이 조인을 수행하는 과정을 예시하는 그림으로 두 행을 연결하는 직선은 조인을 위해 비교를 수행하는 시도를 나타낸다.



(a) Standard merge join (b) MPMGJN

그림 9. 표준 머지 조인과 MPMGJN의 비교 과정

Fig. 9. Workout of standard merge join and MPMGJN

(2) Binary Structural Join

Stack-Tree Join[19]은 기존의 전통적인 관계 조인 알고리즘에 대응되는 형태(counterpart)가 없는 새로운 조인 알고리즘으로 스택을 이용하여 두 데이타 노드간 구조적 관계를 식별한다. 데이타 노드의 식별자가 (시작, 끝, 레벨) 형태의 값을 가지는 형태에 적용 가능하며 입력 전체를 읽는 순차 알고리즘 중에서 입출력과 CPU가 최적인 알고리즘이다. 기본적인 착상은 XML 트리를 트리 높이 크기의 스택을 가지고 트리를 깊이-우선 방식으로 순회를 하면, 순회 도중 트리내 모든 조상-자손 관계는 스택에서 조상 노드보다 상위에 나타나는 자손 노드들에 의하여 중첩된다는 관찰을 통해 개발되었다. 기존의 트리-머지 계열의 조인 알고리즘보다 최악의 경우에 더 좋은 성능을 보인다.

(3) Holistic Twig join

이 연구 이전의 작업들은 트리 패턴의 질의들을 먼저 각각의 이진 구조적 관계로 분할하여 결과를 구하고 난 후, 최종 결과를 얻기 위하여 각각의 매치 결과를

함께 붙인다[18]. 이진 구조적 관계로의 분할 처리 방법은 상당한 중간 결과를 만들어 내는 단점이 있다. 따라서 Twig-Stack[18], 그의 변형[20]은 Holistic Twig Join 계열 알고리즘으로 중간 결과를 줄이기 위하여 연결된 스택 체인을 사용하여 루트부터 단말 노드까지의 질의 경로를 압축되게 표현하였다. 그리고 Twig^{2}Stack[24]은 트윅 결과를 더욱 더 압축되게 표현하기 위하여 계층 스택을 사용한다. 이러한 작업들은 이진 구조적 관계로 분리하여 각각의 관계에 대해 조상-자손 관계를 식별하지 않고 선형 경로 전체적인 관점에서 질의 처리를 수행하여 이진 구조 관계 처리에 따르는 중간 결과 저장 및 다시 읽는 작업을 회피하였다.

2) 관계 데이타베이스에서의 조인

(1) XRel : BEL값을 이용한 세타 조인

XRel[5]은 구간-기반 레이블링을 통하여 노드 번호가 할당 되었다. 따라서 데이타 노드들 간의 조상-관계 식별은 자손 노드의 구간 값이 조상 노드 구간 값에 포함되는 지를 비교하여 결정된다. 관계 시스템에서의 포함 관계는 2개의 Θ-join을 이용하여 구현된다. (begin, end, level) 형태의 값을 BEL 값이라 부른다.

XRel은 하나의 분기경로 질의(BPQ: Branching Path Query)를 여러 개의 선형 경로 질의(LPQ: Linear Path Query)로 분할한다. 각 LPQ는 루트 노드로부터 분기 노드까지의 LPQ와 루트 노드로부터 단말 노드까지의 LPQ들로 구성된다. 예를 들어, BPQ $/l_1[/l_2/l_3=v_2]/l_4$는 다음 세 개의 LPQ 즉 P_1:$/l_1$, P_2:$/l_1/l_2/l_3$, P_3:$/l_1/l_4$로 분해되며 조인 처리 과정은 다음과 같다. 여기서 P_3은 결과 노드를 포함하고 있는 LPQ를 의미한다.

① 각 LPQ에 대해서, 2장 1절 XML 인덱싱에서 설명한 것과 같은 방법으로 스트링 매치를 통하여 LPQ P_1, P_2, P_3에 해당하는 데이타 노드들의 집합을 각각 찾는다.

② 그리고, 선택조건의 있는 경우 선택 조건의 값을 저장하고 있는 테이블들(Element, Text, Attribute)중 하나의 테이블과 조인(label_path_id의 조인 열 이용)하여, 선택조건을 만족하는 데이타 노드들로 데이타 노드 집합을 줄인다. 이렇게 구해진 LPQ P_1, P_2, P_3에 해당하는 데이타 노드들의 집합을 각각

$NodeSet_1$, $NodeSet_2$, $NodeSet_3$이라 하자.

③ 분기 노드 $/l_1$에서 끝나는 LPQ P_1을 처리하여 얻은 노드 집합 $NodeSet_1$과 단말 노드 $/l_1/l_2/l_3$에서 끝나는 LPQ P_2로부터 얻은 노드 집합 $NodeSet_2$를 비교하여 $NodeSet_2$의 노드들을 포함하는 $NodeSet_1$ 노드들의 집합, $BNodeSet_1$(조상노드들)을 구한다.

④ 노드들의 집합 $BNodeSet_1$과 결과 노드 $/l_4$를 포함하는 LPQ P_3을 처리하여 얻은 노드 집합 $NodeSet_3$을 비교하여 $BNodeSet_1$과의 노드 값에 포함되는 노드들로 $NodeSet_3$의 노드 집합을 줄여 BPQ 처리를 완료한다.

예제 1. 다음과 같은 BPQ를 생각해 보자.

Q2: //journal[/keyword = "DB"]//author[/last="Lee"]/first

그림 10은 BPQ Q2의 질의 패턴을 보여준다. 여기서는, 다음과 같이 다섯 개의 LPQ들이 생성된다. LPQ P_1: //journal, P_2: //journal/keyword, P_3: //journal//author, P_4: //journal//author/last, P_5: //journal//author/first

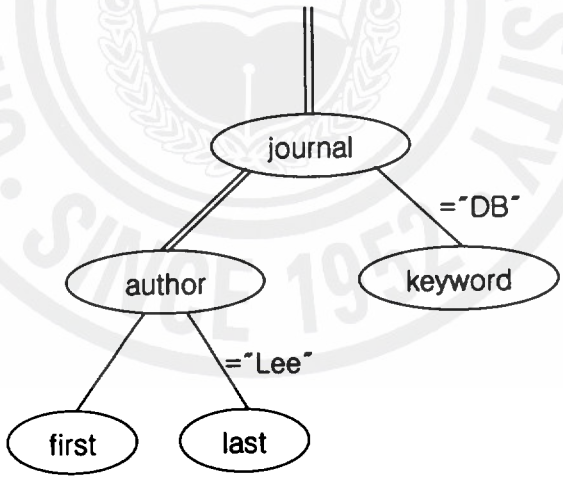


그림 10. BPQ Q2의 질의 패턴

Fig. 10. The query pattern of BPQ Q2

그림 11은 BPQ Q2를 처리하기 위해 변환된 SQL문이다. 그림 11의 XRel SQL은

① 각 LPQ에 대해서, 스트링 매치를 통하여 LPQ P_1, P_2, P_3, P_4, P_5에 해당하는 데

이타 노드들의 집합을 각각 찾는다.

② LPQ P_2와 P_4에 대한 ①의 처리 결과에 선택 조건의 값을 저장하고 있는 Text 테이블과 조인하여, 각각 데이타 노드 집합 $NodeSet_2$, $NodeSet_4$를 구한다.

③ ①에서 구한 $NodeSet_1$과 $NodeSet_2$를 비교하여 $NodeSet_2$의 노드들을 포함하는 $NodeSet_1$ 노드들의 집합, $BNodeSet_1$(조상노드들)을 구한다.

④ ①에서 구한 $NodeSet_3$과 $NodeSet_4$를 비교하여 $NodeSet_4$의 노드들을 포함하는 $NodeSet_3$ 노드들의 집합, $BNodeSet_3$(조상노드들)을 구한다.

⑤ 노드들의 집합 $BNodeSet_3$와 결과 노드를 포함하는 LPQ P_5를 처리하여 얻은 노드 집합 $NodeSet_5$를 비교하여 $BNodeSet_3$의 노드 값에 포함되는 노드들로 $NodeSet_5$의 노드 집합을 줄인다.

⑥ ⑤에서 구한 $NodeSet_5$와 $BNodeSet_1$을 비교하여 $BNodeSet_1$의 노드 값에 포함되는 노드들로 $NodeSet_5$의 노드 집합을 줄여 BPQ 처리를 완료한다. 두 개의 분기하는 노드가 있기 때문에 총 네 번의 포함 관계 조인이 수행되며, 따라서 총 8번의 세타조인(Θ-join)을 수행한다.

```
SELECT DISTINCT e5.document_id, e5.start_position, e5.end_position
FROM    Path p1, Path p2, Path p3, Path p4, Path p5,
        Element e1, Element e3, Element e5, Text t2, Text t4
WHERE   p1.label_path LIKE "#%/journal#"
AND     p2.label_path LIKE "#%/journal#/keyword#"
AND     p3.label_path LIKE "#%/journal#%/author#"
AND     p4.label_path LIKE "#%/journale#%/author#/last#"
AND     p5.label_path LIKE "#%/journal#%/author#/first#"
AND     e1.label_path_id = p1.label_path_id
AND     t2.label_path_id = p2.label_path_id
AND     e3.label_path_id = p3.label_path_id
AND     t4.label_path_id = p4.label_path_id
AND     e5.label_path_id = p5.label_path_id
AND     e1.start_position < t2.start_position
```

```
AND     e1.end_position > t2.end_position
AND     e1.document_id = t2.document_id
AND     t2.value = "DB"
AND     e3.start_position < t4.start_position
AND     e3.end_position > t4.end_position
AND     e3.document_id = t4.document_id
AND     t4.value = "Lee"
AND     e3.start_position < e5.start_position
AND     e3.end_position > e5.end_position
AND     e3.document_id = e5.document_id
AND     e1.start_position < e5.start_position
AND     e1.end_position > e5.end_position
AND     e1.document_id = e5.document_id;
```

그림 11. BPQ Q2를 위한 XRel SQL 문

Fig. 11. The SQL statement in XRel for BPQ Q2

(2) XParent : 조상 테이블을 이용한 이퀴 조인

XParent[6,7]는 XRel[5]에서 사용하는 구간-기반 레이블 대신 에지-중심 접근법(edge-oriented approach)을 이용하여 에지 당 1개의 값만을 가진다. 따라서 데이타 노드들 간의 조상-관계 식별은 자손 노드의 조상 정보 테이블인 Ancestor 테이블을 이용하여 조상 노드를 구하여 비교하고자 하는 조상 노드의 식별자 값과 동일한지를 비교하여 판정한다. XParent에서의 LPQ 질의 처리는 XRel과 동일한 방법으로 수행되지만 BPQ는 다르게 처리된다.

XParent는 질의 패턴의 루트 노드로부터 각 단말 노드까지의 LPQ들 형태로 분할하기 때문에 XRel 보다 더 작은 개수의 LPQ들이 만들어 진다. 예를 들어, BPQ $/l_1[/l_2=v_2]/l_3[l_4]/l_5$는 다음 세 개의 LPQ P_1:$//l_1/l_2$, P_2:$//l_1/l_3/l_4$, P_3:$//l_1/l_3/l_5$로 분해되며 조인 처리 과정은 다음과 같다. 여기서 LPQ P_3는 결과 노드를 포함하고 있는 LPQ를 의미한다.

① XRel과 같은 방법으로 LPQ를 처리하여 노드 집합 $NodeSet_1$, $NodeSet_2$, $NodeSet_3$을 구한다.

② 노드 집합 $NodeSet_1$, $NodeSet_2$, $NodeSet_3$과 각각 Ancestor 테이블과의 조인을 통하여 각 LPQ P_i(i=1,2,3)의 분기 노드에 해당하는 노드집합($NodeSet_i$의 조상 노드들의 집합, (i=1,2,3)) 즉 $BNodeSet_1$, $BNodeSet_2$, $BNodeSet_3$을 구한다.

③ ②에서 구한 노드들의 집합 $BNodeSet_2$와 LPQ P_3을 처리하여 얻은 노드 집합 $NodeSet_3$을 비교하여 $BNodeSet_2$의 노드 값과 동일한 값을 가지는 노드(공통 조상)들로 $NodeSet_3$의 노드 집합을 줄인다.

④ ②에서 구한 노드들의 집합 $BNodeSet_1$과 ③에서 구한 노드 집합 $NodeSet_3$을 비교하여 $BNodeSet_1$의 노드 값과 동일한 값을 가지는 노드(공통 조상)들로 $NodeSet_3$의 노드 집합을 줄여 BPQ 처리를 완료한다.

예제 2. 예제 1 안에 있는 BPQ Q2를 생각해 보자. 분할되는 LPQ는 다음과 같이 세 개의 LPQ들이 생성된다.

P_1: //journal/keyword, P_2: //journal//author/last, P_3: //journal//author/first

그림 12는 BPQ Q2를 처리하기 위하여 변환된 SQL문을 보여준다. 각 LPQ들로부터 얻어진 노드 집합들은 공통의 조상 노드들을 공유하는 노드들을 찾기 위하여 Ancestor 테이블과 조인된다. 그 밖의 다른 SQL 문은 XRel의 문과 매우 유사하다. 두 개의 분기하는 노드가 있기 때문에 총 5번의 이퀴 조인(=-join)을 수행한다.

```
SELECT DISTINCT e3.document_id, e3.node_id
FROM   LabelPath lp1, LabelPath lp2, LabelPath lp3,
       Data d1, Data d2, Element e3,
       Ancestor an1, Ancestor an2, Ancestor an3
WHERE  lp1.label_path LIKE ".%/journal./keyword."
AND    lp2.label_path LIKE ".%journal.%/author./last."
AND    lp3.label_path LIKE ".%/journal.%/author./first."
AND    d1.label_path_id = lp1.label_path_id
AND    d2.label_path_id = lp2.label_path_id
```

```
AND     e3.label_path_id = lp3.label_path_id
AND     d1.value = "DB"
AND     d2.value = "Lee"
AND     d1.node_id = an1.node_id
AND     d2.node_id = an2.node_id
AND     e3.node_id = an3.node_id
AND     an1.offset_to_ancestor = 1
AND     an2.offset_to_ancestor = 1
AND     an1.ancestor_node_id = an3.ancestor_node_id
AND     an2.ancestor_node_id = an3.ancestor_node_id;
```

그림 12. BPQ Q2를 위한 XParent SQL 문

Fig. 12. The SQL statement in XParent for BPQ Q2

(3) EPIS : BEL값 이용한 세타 조인

EPIS는 XRel에서 사용하는 구간-기반 레이블을 사용한다. 데이타 노드들간의 조상-관계 식별은 XRel에서와 같이 BEL 값을 이용한 포함관계로 판정된다. 2개의 세타 조인을 통하여 포함 관계가 구현된다. EPIS는 XRel과 다른 방식으로 LPQ 질의를 처리한다. 그러나 각 LPQ 처리 결과를 조인하는 과정은 XRel과 동일하다.

EPIS는 하나의 BPQ를 여러 개의 LPQ로 분할한다. 각 LPQ는 루트 노드로부터 분기 노드까지의 LPQ와 루트 노드로부터 단말 노드까지의 LPQ들로 구성된다. 예를 들어, BPQ $/l_1[/l_2/l_3=v_2]/l_4$는 다음 세 개의 LPQ P_1:$/l_1$, P_2:$/l_1/l_2/l_3$, P_3: $/l_1/l_4$로 분해되며 조인 처리 과정은 다음과 같다. 여기서 LPQ P_3은 결과 노드를 포함하고 있는 LPQ를 의미한다.

① 각 LPQ에 대해서, 2장 1절 XML 인덱싱에서 설명한 것과 같은 방법으로 각 LPQ내의 경로 질의를 레이블 노드 단위로 분할하고, 각각의 레이블에 대해 인덱스를 탐색하여 후보 경로 식별자를 구한다.

② ①에서 구한 각 LPQ 소속 후보 경로 식별자 집합 간에 경로 질의 상에 나타난 레이블 순서대로 셀프조인을 수행하여 경로 식별자들의 집합을 구한다.

③ ②에서 구한 각 LPQ의 경로 식별자들의 집합과 PathOccurrenceTbl을 조인하여 LPQ P_1, P_2, P_3에 해당하는 데이타 노드들의 집합을 각각 찾는다.

④ 그리고, 선택조건의 있는 경우 Term 테이블과 TermOccurrenceTbl을 이용하여 선택조건을 만족하는 데이타 노드들로 데이타 노드 집합을 줄인다. 이렇게 구해진 LPQ P_1, P_2, P_3에 해당하는 데이타 노드들의 집합을 각각 $NodeSet_1$, $NodeSet_2$, $NodeSet_3$이라 하자.

⑤ 분기 노드 $/l_1$에서 끝나는 LPQ P_1을 처리하여 얻은 노드 집합 $NodeSet_1$과 단말 노드 $/l_1/l_2/l_3$에서 끝나는 LPQ P_2로부터 얻은 노드 집합 $NodeSet_2$를 비교하여 $NodeSet_2$의 노드들을 포함하는 $NodeSet_1$ 노드들의 집합 $BNodeSet_1$(조상노드들)을 구한다.

⑥ 노드들의 집합 $BNodeSet_1$과 결과 노드 $/l_4$를 포함하는 LPQ P_3을 처리하여 얻은 노드 집합 $NodeSet_3$을 비교하여 $BNodeSet_1$의 노드 값에 포함되는 노드들로 $NodeSet_3$의 노드 집합을 줄여 BPQ 처리를 완료한다.

예제 3. 예제 1의 BPQ Q2를 고려해 보자. BPQ Q2는 다음과 같이 다섯 개의 LPQ들로 분할된다. LPQ P_1: //journal, P_2: //journal/keyword, P_3: //journal//author, P_4: //journal//author/last, P_5: //journal//author/first

그림 13은 BPQ Q2를 처리하기 위하여 변환된 SQL문을 보여준다. 각 LPQ들을 레이블 단위로 분할하여 후보 경로 식별자의 집합을 구하고 나서, 후보 경로 식별자 간의 셀프조인을 수행하여 최종 경로 식별자를 구하는 과정에서 5번의 셀프 조인이 발생한다. 두 개의 분기하는 노드가 있기 때문에 총 4번의 포함 관계 조인이 수행되며, 따라서 8번의 세타 조인을 수행한다.

```
SELECT DISTINCT e5.docid, e5.startpos, e5.endpos
FROM   UniqPathElmTbl p1, UniqPathElmTbl p21, UniqPathElmTbl p22,
       UniqPathElmTbl p31, UniqPathElmTbl p32,
       UniqPathElmTbl p41, UniqPathElmTbl p42, UniqPathElmTbl p43,
       UniqPathElmTbl p51, UniqPathElmTbl p52, UniqPathElmTbl p53,
       PathOccurrenceTbl e1, PathOccurrenceTbl e3, PathOccurrenceTbl e5,
```

```
        TermTbl tm2, TermTbl tm4,
        TermOccurrenceTbl t2, TermOccurrenceTbl t4
WHERE   p1.ename = "journal#"
AND     e1.pathid = p1.pathid
AND     p21.ename = "journal"
AND     p22.ename = "keyword#"
AND     p21.pathid = p22.pathid
AND     p22.pathid = t2.pathid
AND     p21.plevel = p22.plevel -1
AND     tm2.term  = "DB"
AND     tm2.termid = t2.termid
AND     e1.startpos < t2.position
AND     e1.endpos  > t2.position
AND     e1.docid  = t2.docid
AND     p31.ename = "journal"
AND     p32.ename = "author#"
AND     p31.pathid = p32.pathid
AND     p32.pathid = e3.pathid
AND     p31.plevel < p32.plevel
AND     p41.ename = "journal"
AND     p42.ename = "author"
AND     p43.ename = "last#"
AND     p41.pathid = p42.pathid
AND     p42.pathid = p43.pathid
AND     p43.pathid = t4.pathid
AND     p41.plevel < p42.plevel
AND     p42.plevel = p43.plevel - 1
AND     tm4.term  = "Lee"
```

```
AND     tm4.termid = t4.termid
AND     e3.startpos < t4.position
AND     e3.endpos  > t4.position
AND     e3.docid   = t4.docid
AND     e3.startpos < e5.startpos
AND     e3.endpos  > e5.startpos
AND     e3.docid   = e5.docid
AND     p51.ename = "journal"
AND     p52.ename = "author"
AND     p53.ename = "first#"
AND     p51.pathid = p52.pathid
AND     p52.pathid = p53.pathid
AND     p53.pathid = e5.pathid
AND     p51.plevel < p52.plevel
AND     p52.plevel = p53.plevel - 1
AND     e1.startpos < e5.startpos
AND     e1.endpos  > e5.startpos
AND     e1.docid   = e5.docid
```

그림 13. BPQ Q2를 위한 EPIS SQL 문

Fig. 13. The SQL statement in EPIS for BPQ Q2

(4) XIR-Branching : 데이타 경로의 접두어 매치를 통한 이퀴 조인

XIB-Branching은 프리픽스 레이블링 기법을 이용하여 노드 식별자 값을 할당하며 문서 내 전체 노드에 대해 유일한 정수 값을 사용한다.

XIB-Branching은 질의 패턴의 루트 노드로부터 각 단말 노드까지의 경로를 하나의 LPQ로 분할하기 때문에 XRel보다 더 작은 수의 LPQ들이 만들어 진다. 예를 들어, BPQ $/l_1[/l_2=v_2]/l_3[l_4]/l_5$는 다음 세 개의 LPQ $P_1:/l_1/l_2$, $P_2:/l_1/l_3/l_4$, $P_3:/l_1/l_3/l_5$로 분해되며 조인 처리 과정은 다음과 같다.

① 2장 1절에서 XML 인덱싱에서 설명한 바와 같이 정보검색(IR)의 역 인덱스를 이용하여 각 LPQ에 해당하는 노드 경로들의 집합 (pid와 labelpath)을 각각 찾는다. 이렇게 구해진 레이블 경로 식별자와 레이블 경로 문자열로 이루어진 노드 경로들의 집합을 $LPSet_1$, $LPSet_2$, $LPSet_3$이라 하자. 이 집합들은은 XRel, XParent, EPIS와는 달리 각 LPQ에 매치되는 데이타 노드의 식별자뿐만 아니라 레이블 경로 자체도 포함한다.

② 그리고, 선택조건의 있는 경우 $LPSet_1$, $LPSet_2$, $LPSet_3$ 각각을 선택 조건의 값을 저장하고 있는 NodePath 테이블과 조인하여, 선택조건을 만족하는 데이타 노드들로 데이타 노드 집합을 줄인다. 이렇게 구해진 LPQ P_1, P_2, P_3에 해당하는 데이타 노드들의 집합을 각각 $NPSet_1$, $NPSet_2$, $NPSet_3$이라 한다. ①과 ② 단계의 연산은 식(1)에서와 같은 방법으로 완료된다.

$$\Pi_{nodepath}((\sigma_{MATCH(labelpath,LPQ)}(\text{LabelPath})) \bowtie_{pid=pid} (\sigma_{value\ \theta\ v}(\text{NodePath}))) \quad \text{식(1)}$$

여기서, "*value* θ *v*" 는 처리될 LPQ 내에 포함된 분기 조건 표현식의 단말 레이블에 대한 선택 조건(selection condition)이다.

③ 노드들의 집합 $NPSet_2$와 결과 노드 $//l_5$를 포함하는 LPQ P_3을 처리하여 얻은 노드 집합 $NPSet_3$간에 프리픽스 매치를 통한 이퀴 조인(XIB-Branching에서 프리픽스 매치 조인이라 부름)을 수행하여 $NPSet_3$의 노드 집합을 줄인다.

④ ③에서 구한 $NPSet_3$의 노드 집합과 $NPSet_1$ 노드 집합 간에 프리픽스 매치를 통한 이퀴 조인을 수행하여 $NodeSet_3$의 노드 집합을 줄여 BPQ 처리를 완료한다.

한편, XIB-Branching에서 주어진 두 LPQ들로부터 얻는 두 개의 NPSet들 간의 프리픽스 매취 조인은 다음과 같이 수행된다. (1) 양쪽 LPQ들에 대해 루트부터 분기 레이블까지의 가장 긴 공통의 프리픽스 레이블 서브-경로 $l_1l_2 \ldots l_c$를 찾는다. (2) 그 레이블 서브-경로 $l_1l_2 \ldots l_c$에 속하는 공통의 프리픽스 노드 서브-경로들 $\{n_1n_2 \ldots n_c\}$의 집합을 찾는다. 그리고, (3) 그 집합 안의 각 노드 서브-경로 $n_1n_2 \ldots n_c$에 대해서, 공통의 서브-경로를 가진 모든 노드 경로들을 선택한다.

예제 4. 예제 1에서 사용된 BPQ Q2를 생각해 보자. 분할되는 세 개의 LPQ는 P_1:

//journal/keyword, P_2: //journal//author/last, P_3: //journal//author/first이다.

그림 14는 BPQ Q2를 처리하기 위하여 변환된 SQL문을 보여준다. 그것은 각 노드 집합들간 튜플 단위로 프리픽스 매취 조인을 수행한다. Prefix_matching 함수는 처음 두 입력 인수들(예, p1.nodepath와 p2.nodepath)로서 제공된 두 노드 경로들 간에 getCommonPrefixLength 함수로부터 반환된 길이만큼의 프리픽스를 비교함으로써 프리픽스 매취를 수행한다. getCommonPrefixLength 함수는 두 개의 LPQ와, 각각에 해당하는 두 레이블 경로들을 입력으로 받아 다음과 같은 과정을 거쳐 프리픽스의 길이를 계산한다. 그 과정은 ① 두 개의 입력 LPQ가 가장 긴 공통의 서브-표현식을 밝힌다. 예를 들어, //journal/keyword와 //journal//author/last의 공통은 //journal이다. ② 두 개의 입력 레이블 경로 즉, p1.label_epath와 p2.label_path 각각에 대해 공통의 프리픽스 서브-표현식에 해당하는 프리픽스 레이블들의 개수를 센다. ③ 만약, 두 숫자가 같으면 count를 반환하고, 그렇지 않으면 -1을 반환한다.

```
SELECT DISTINCT n3.doc_id, n3.nodepath
FROM    LabelPath p1, LabelPath p2, LabelPath p3,
        NodePath n1, NodePath n2, NodePath n3
WHERE MATCH(p1.labelpath, "journal" NEAR(1) "keyword" NEAR(1) "&keyword")
AND     MATCH(p2.labelpath, "journal" NEAR(MAXINT) "author" NEAR(1) "last"
                NEAR(1) "&last")
AND     MATCH(p3.labelpath, "journal" NEAR(MAXINT) "author" NEAR(1) "first"
                NEAR(1) "&first")
AND      n1.pid = p1.pid
AND      n2.pid = p2.pid
AND      n3.pid = p3.pid
AND      n1.value = "DB"
AND      n2.value = "Lee"
AND      Prefix_matching(p1.nodepath, p2.nodepath, getCommonPrefixLength(p1.label
          _path, p2.label_path, "//journal/keyward", "//journal//author/last")) > 0
AND      Prefix_matching(p2.nodepath, p3.nodepath, getCommonPrefixLength
```

```
(p2.label_path, p3.label_path, "//journal//author/last",
                                /journal//author/ first") > 0;
```

그림 14. BPQ Q2를 위한 XIR-Branching SQL 문

Fig. 14. The SQL statement in XIR-Branching for BPQ Q2

III. 제안 XML 인덱싱

본 장에서는 XML 질의 처리를 위한 시스템 설계를 위해 먼저 대상이 되는 XML 문서 모델을 정의하고 이에 대하여 지원하는 질의 모델, 질의 패턴에 대해 기술한다. 그리고 나서 효과적인 분기 경로 질의 처리를 위해 본 논문에서 제안하는 XML 인덱싱 방법에 대해 설명하고 마지막으로 제안 시스템의 구성과 저장 구조인 데이타베이스 스키마에 대해 상세히 기술한다.

1. XML 문서 모델

본 논문에서 사용하는 XML 문서 모델은 Bruno[18], XIR-Branching[10]에서 제안한 모델을 기반으로 하고 있다. 이 모델에서, XML 문서는 루트가 있고, 순서가 있으며, 레이블이 기록된 트리(rooted, ordered, labeled tree)로 표현된다. 이 트리의 노드(node)는 엘리먼트(element), 애트리뷰트(attribute), 값(value) 중 하나를 나타낸다. 또한, 이 트리의 에지(edge)는 직접적인 엘리먼트-서브엘리먼트 관계성(element-subelement relationship), 엘리먼트-애트리뷰트(element-attribute relationship) 관계성, 엘리먼트-값(element-value relationship) 관계성, 애트리뷰트-값(attribute-value relationship) 관계성 중 하나를 나타낸다. 엘리먼트와 애트리뷰트 노드는 XML 문서의 구조를 결정하는 요소로서, 각 노드에게는 레이블(label, 즉 name)과 고유 식별자(unique identifier)를 부여한다.

그림 15는 XML 문서의 XML 트리 예를 보인다. 이 그림에서 애트리뷰트 값(attribute value) "3"을 나타내는 노드 번호 4을 제외한, 모든 단말 노드(leaf node)들은 엘리먼트 값(element value)들을 나타내고, 애트리뷰트 @vol을 나타내는 노드 번호 3을 제외한, 단말 노드가 아닌 모든 노드들은 엘리먼트를 나타낸다. 이때, 애트리뷰트는 그 레이블 앞에 프리픽스 '@'를 사용함으로써 엘리먼트와 구분한다.

본 논문에서 사용하는 모델은 노드를 값(value)이 아닌 엘리먼트나 애트리뷰트를

표현하도록 하였고, 특히 엘리먼트 노드가 혼합 컨텐츠(mixed content)를 가질 수 있는 혼합 데이타 모델을 지원하기 위하여 참고 문헌[8,10]의 모델을 변경하였다. 그리고 변경된 문서 모델에 따라 레이블 경로의 개념을 가지도록 정의 1, 정의2와 같이 수정, 정의하였다.

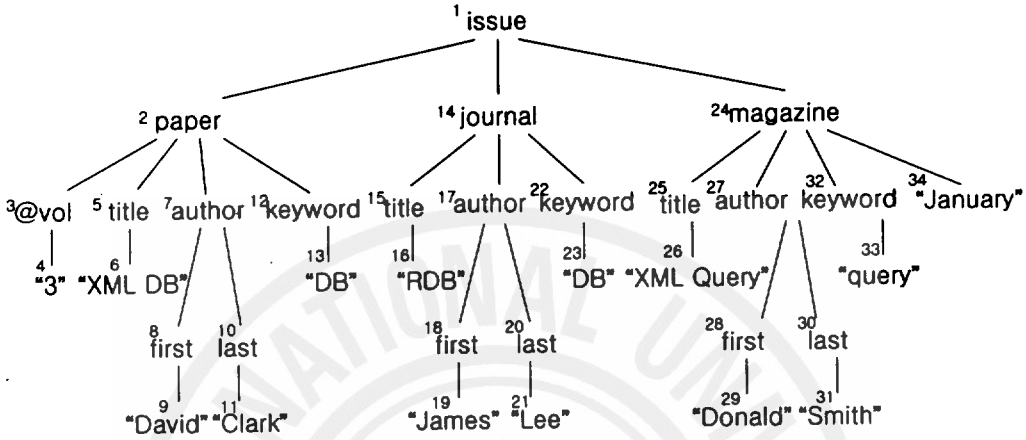


그림 15. XML 문서 트리

Fig. 15. An XML document tree

정의 1. 순방향 레이블 경로(forward label path)

XML 트리 내의 **순방향 레이블 경로**는 그 트리의 루트 노드로부터 특정 노드 p까지의 노드 레이블 l_1, l_2, ..., $l_p(p \geq 1)$들의 연속(sequence)으로 정의되고, l_1, l_2, ..., l_p로 표현된다.

정의 2. 역방향 레이블 경로(backward label path)

XML 트리 내의 **역방향 레이블 경로**는 그 트리내의 특정 노드 p로부터 루트 노드까지의 노드 레이블 l_p, ..., l_2, $l_1(p \geq 1)$들의 연속(sequence)으로 정의되고, l_p, ..., l_2, l_1로 표현된다.

정의 3. 데이타 경로(data path)

XML 트리 내의 **데이타 경로**는 그 트리 내의 루트 노드로부터 특정 노드 p까지의 노드 식별자 n_1, n_2, ..., $n_p(p \geq 1)$들의 연속(sequence)으로 정의되고, n_1, n_2, ...,

n_p로 표현된다.

레이블 경로는 XML 문서 구조를 표현하며, 이를 스키마-레벨 정보(schema-level information)라 하고, 이에 반하여, 데이타 경로는 XML 문서 인스턴스를 표현하며, 이를 인스턴스-레벨 정보(instance-level information)라 한다. 본 논문에서는, 레이블 경로는 경로 질의와 "매칭(match)된다" 또는 "해당된다". 데이타 경로는 레이블 경로에 "속한다", 데이타 경로는 경로 질의로부터 "구해진다" 고 말한다. 예를 들어, 그림 15에서 *issue.journal.author.first*는 순방향 경로 질의 *//journal//first*와 매칭하는 순방향 레이블 경로이고, *first.author.journal.issue*는 역방향 경로 질의 *first\\journal*와 매칭하는 역방향 레이블 경로이다. 1.14.17.18은 순방향 또는 역방향 레이블 경로에 "속"하는 데이타 경로들이다. 레이블 경로와 경로 질의와는 달리 데이타 경로는 순방향만 존재한다. 단지 데이타 경로들을 찾기 위한 표현 방식의 차이일 뿐이며 순방향 경로 질의는 사용자가 사용하며 역 방향 경로 질의는 질의 처리를 위해 내부적으로 사용한다. 이때, 같은 구조를 가지는 하나 이상의 인스턴스들이 있을 수 있기 때문에, 같은 레이블 경로에 속하는 하나 이상의 데이타 경로들이 있을 수 있다.

기존 연구에는 레이블 경로에 대해 본 논문과는 다른 정의들이 있다. 그 예는 DataGuides[34]나 XParent[6,7], XIR-Branching[10] 등에서 찾아볼 수 있다. DataGuides는 반드시 루트부터 노드일 필요가 없는, 어떤 내부 노드(internal node)에서 어떤 단말 노드(leaf node)까지의 연속으로서 경로를 정의하였고, XParent는 노드 대신 에지의 연속으로서 경로를 정의하였다. 정의 1과 정의 3은 DataGuides의 정의와 개념적으로 다르고 XParent와 XIR-Branching의 정의와는 개념적으로 동일하나 XML 문서 모델이 다르다. 그리고 정의 2는 본 논문에서 추가적으로 제시하는 레이블 정의이다.

2. XML 질의 모델

XPath나 XQuery 등과 같은 주요한 XML 질의 언어들은 어떤 XML 트리에 대한

질의를 명시하기 위하여 경로 표현식을 사용한다. 본 논문에서 사용하는 질의 언어는 트리 패턴 질의(tree pattern query, TPQ) 클래스[19]에 속한다. 그들은 선형 경로 질의(linear path query, LPQ)와 분기 경로 질의(branching path query, BPQ)이다. 이들은 각각 순방향 선형 경로 질의(forward linear path query, FLPQ)와 역방향 선형 경로 질의(backward linear path query, BLPQ), 순방향 분기 경로 질의(forward branching path query, FBPQ)와 역방향 분기 경로 질의(backward branching path query, BBPQ)으로 표현된다.

정의 4, 정의 5에서 정의된 바와 같이, FLPQ는 '/'나 '//'로 연결된 레이블(label)들의 연속으로 표현되고, BLPQ는 '\'나 '\\'로 연결된 레이블(label)들의 연속으로 표현된다.

정의 4. 순방향 선형 경로 질의(forward linear path query)

순방향 선형 경로 질의는 $l_0\ o_1\ l_1\ o_2\ l_2\ \ldots o_n\ l_n$로서 정의된다. 이때 $l_i(i=1,2,\ldots,n)$은 그 경로 내에 존재하는 i 번째 레이블이고, $o_j(j=1,2,\ldots n)$는 '/'나 '//' 중 하나이며, 이들 각각은 l_{j-1}와 l_j 간의 부모-자식 관계성(parent-child relationship)이나 조상-자손 관계성(ancestor-descendant relationship)을 나타낸다. 여기서, l_0는 모든 XML 문서들의 집합인 XML 트리들을 모두 하나로 연결하는 가상의 루트 노드이고 생략될 수 있다.

정의 5. 역방향 선형 경로 질의(backward linear path query)

역방향 선형 경로 질의는 $l_n\ o_n\ l_{n-1}\ o_{n-1}\ \ldots\ l_1\ o_1\ l_0$로서 정의된다. 이때 $l_i(i=1,2,\ldots,n)$은 그 경로 내에 존재하는 i 번째 레이블이고, $o_j(j=1,2,\ldots n)$는 '\'나 '\\' 중 하나이며, 이들 각각은 l_j와 l_{j-1}간의 자식-부모 관계성(child-parent relationship)이나 자손-조상 관계성(descendant-ancestor relationship)을 나타낸다. 여기서, l_0는 모든 XML 문서들의 집합인 XML 트리들을 모두 하나로 연결하는 가상의 루트 노드이고 생략될 수 있다.

정의 7, 8에서 정의한 바와 같이 FBPQ나 BBPQ는 어떤 레이블 $l_i(i \in \{1,2,\ldots n\})$에 대해서, '분기 조건 표현식'[$C_i$]가 추가된 FLPQ나 BLPQ로서 표현된다. 다른 문헌[43]의 연구와 같이, 본 논문에서는 정의 6의 정의대로 분기 조건 표현식에 단순 선

택 조건(simple selection predicate) 만을 다룬다.

정의 6. 분기 조건 표현식(branch predicate expression) C_i는 각기 불리언 표현식(Boolean expression)으로 정의된다. 이들 불리언 항(Boolean term)들은 단일 레이블 l이거나, 조건(predicate) $l\ \theta\ v$ 중 하나이다. 여기서, v는 상수이고, θ는 비교 연산자($\theta \in \{=, \neq, >, \geq, <, \leq\}$)이다. 단일 레이블 l은 그 레이블을 가진 엘리먼트나 애트리뷰트의 존재 유무를 명시하고, $l\ \theta\ v$ 는 선택 조건(selection condition)을 만족하는 레이블들을 명시한다.

분기 경로 질의(branching path query)는 선형 경로 질의(linear path query) 내의 임의의 레이블에 조건을 부여할 수 있는 형태로 정의할 수 있다. 분기 경로 질의(branching path query)에 관한 연구[44]에서는 XML 문서를 그래프로 모델하고 그래프에 대한 분기 경로 질의를 정의하였다. 그러나, XML 문서는 트리로 표현하는 것이 자연스럽기 때문에, 본 연구에서는 다른 연구[5,6,7,10,20]에서와 같이 XML 문서를 트리로 표현한다. 본 연구에서는 기존 연구[44]와 유사한 내용으로 분기 경로 질의를 정의할 수 있으며, 기존 연구[44]에서 XML 문서를 트리로 모델하고 싸이클 표현을 제거하면, 정의 7, 정의8과 같이 새롭게 정의할 수 있다.

정의 7. 순방향 분기 경로 질의(forward branching path query)

순방향 분기 경로 질의는 $l_0\ o_1\ l_1[C_1]\ o_2\ l_2[C_2]\ \ldots\ o_n\ l_n[C_n]$로서 정의된다. 이때, ① $l_0\ o_1\ l_1\ o_2\ l_2\ \ldots\ o_n\ l_n$는 정의 3에서 정의된 FLPQ이고, ② FLPQ에 존재하는 레이블에 명시될 수 있는 $[C_1]...[C_n]$는 해당 레이블에 속한 노드 인스턴스에 대한 필터링 조건을 명시한 것이다. 이때, $[C_k](k=1,2,\ \ldots,n)$는 정의 6에서 정의된 분기 조건 표현식이다, 이들 중(전체가 아닌) 일부가 생략될 수 있다.

정의 8. 역방향 분기 경로 질의(backward branching path query)

역방향 분기 경로 질의는 $[C_n]l_n\ o_n\ [C_{n-1}]l_{n-1}\ o_{n-1}\ \ldots\ [C_2]\ l_2\ o_2\ [C_1]\ l_1\ o_1\ l_0$로서 정의된다. 이때, ① $l_n\ o_n\ l_{n-1}\ o_{n-1}\ \ldots\ l_2\ o_2\ l_1\ o_1\ l_0$는 정의 5에서 정의된 BLPQ이고, ② BLPQ에 존재하는 레이블에 명시될 수 있는 $[C_1]...[C_n]$는 해당 레이블에 속한 노드 인스턴스에 대한 필터링 조건을 명시한 것이다. 이때, $[C_k](k=1,2,\ \ldots,n)$는 정

의 6에서 정의된 분기 조건 표현식이다, 이들 중(전체가 아닌) 일부가 생략될 수 있다.

다음 질의는 "Lee"라는 값을 가진 last 엘리먼트를 자식으로 가진 author 엘리먼트가 후손이 되는 issue 엘리먼트들 중, 후손으로 적어도 하나의 keyword 서브엘리먼트를 포함하고 있는 article 엘리먼트들을 선택한 후, 이들 중 자식이 title 인 엘리먼트들을 찾는 BPQ 예이다.

Q3: //issue[//author/last = "Lee"]//article[/keyword]/title

Q3은 FLPQ //issue//author/last내의 레이블인 last에 대해서 선택 조건을 가지며, FLPQ //issue//article/keyword 내의 레이블인 keyword에 대해 존재여부 확인조건(existential condition)을 가지는 FBPQ이다.

Q4: title\[keyword\]article\\[last = "Lee"\author\\]issue\\

그리고 질의 Q4는 BLPQ last/author//issue//내의 레이블인 last에 대해서 선택 조건을 가지며, BLPQ keyword/article//issue//내의 레이블인 keyword에 대해 존재여부 확인조건을 가지는 BBPQ이다.

3. XML 질의 패턴

본 논문 정의 9에서 정의된 질의 패턴(query pattern)을 가지고 경로 질의를 모델한다. 본 논문에서는 사용된 개념들을 정형적으로 표현하기 위하여 Holistic Twig Join[18]에서 처음 사용된 트윅 패턴(twig pattern) 정의를 약간 확장하였다. 이 정의는 정의 7에서 정의한 FBPQ에 근거하며 순방향 분기 경로 질의에 대한 질의 패턴만을 정의하여 사용한다.

정의 9. 질의 패턴(query pattern)

정의 7의 정의 중, l_0가 생략된 분기 경로 질의 $l_0\ o_1\ l_1[C_1]\ o_2\ l_2[C_2]\ \ldots\ o_n\ l_n[C_n]$은 이진 트리(binary tree)와 그 트리의 최상위 루트 노드에 연결된 허상 에지(dangling edge)로 구성되어 있는 질의 패턴으로 표현한다. 이 질의 패턴은 다음의 성질들을

가진다.

- 에지는 경로 질의 내의 $o_j(j \in 1, 2, ..., n)$를 나타낸다. 에지는 경로 질의 내의 o_j가 '/'일 경우는 단일 실선으로, '//'일 경우는 이중 실선으로 표현된다. 질의 패턴의 루트 노드에 연결된 매달린 에지는 o_1을 나타낸다.
- 노드는 경로 질의 내의 레이블 $l_k(k \in 1, 2, ..., n)$를 나타낸다. 루트 노드는 경로 질의 내의 레이블 l_1을 나타낸다.
- 경로 질의 내의 레이블 $o_k(k \in 1, 2, ..., n-1)$를 나타내는 노드의 좌측 자식 노드는 l_{k+1}를 나타낸다.
- 경로 질의 내의 레이블 $l_k(k \in 1,2,...n)$를 나타내는 노드의 우측 서브 트리는 분기 조건 표현식 $C_k \equiv o_{k1}\ l_{k1}\ o_{k2}\ l_{k2}\ ...\ o_{kp}\ l_{kp}\ (p \geq 1)$를 나타내는 질의 패턴이다.
- 만약, 노드의 레이블이 노드에 대한 선택 조건("Θ v")을 가진다면, 그 노드는 "Θ v" 라는 윗 첨자를 가진다.

본 논문에서는 질의 패턴과 관련하여 다음과 같은 용어를 사용한다.

- 질의 패턴의 질의 노드들 중 하나는 질의 결과로서 검색된다. 이 노드는 정의 4, 5에서 정의된 LPQ나 정의 7에서 정의된 BPQ내의 레이블 l_n에 대응된다. 이 노드를 **결과 (질의)** 노드(result query node)라 부르고, 결과 질의 노드를 다른 노드들과 구분하기 위하여 회색톤으로 표현한다. 그리고 결과 질의 노드에 속하는 데이타 노드를 **결과 데이타** 노드라 부른다.
- 질의 패턴 내에 있는 노드들 중 일부 노드는 우측 서브 트리를 가진다. 우측 서브 트리를 가지는 노드는 **분기 (질의)** 노드(branching query node)라 부른다. 정의 7에서 볼 수 있는 질의 중 $l_k[C_k]$에서, 레이블 l_k를 분기 질의 노드라고 한다. 이러한 분기 질의 노드는 레이블 l_k에 속하는 모든 노드들 중에서 조건 $[C_k]$를 만족하는 노드를 구하기 위해 사용된 표현이다. 그리고 분기 질의 노드에 속하는 데이타 노드를 **분기 데이타** 노드라 부른다.
- 질의 패턴내 결과 질의 노드를 포함하는 LPQ로서 정의 4, 5에서 정의된 LPQ나 정의 7에서 정의된 BPQ내의 $o_1\ l_1\ o_2\ l_2\ ...\ o_n\ l_n$에 대응되는 LPQ를 **결과 LPQ**(result LPQ)라 부른다.

그림 16은 3장 2절의 질의 Q3의 질의 패턴을 보이고 있다. title 노드는 결과 (질

의) 노드이고, issue와 article 노드는 분기 (질의) 노드들이다.

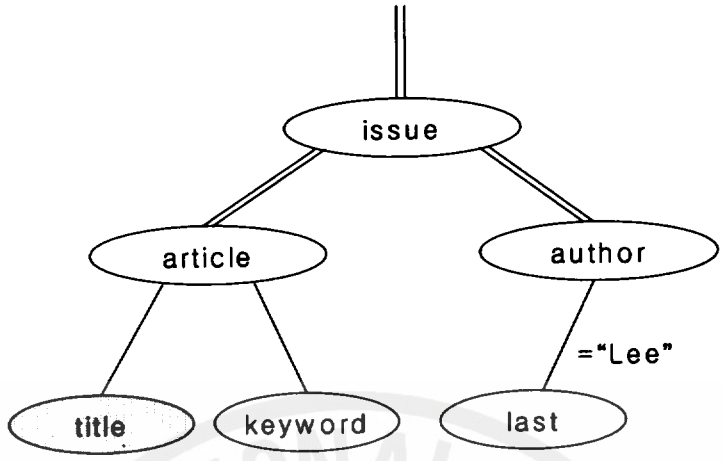


그림 16. 질의 Q3의 질의 패턴

Fig. 16. The query pattern of query Q3

선형 질의 패턴(linear query pattern)은 질의 패턴의 특별한 경우로서 다음과 같이 정의한다.

정의 10. 순방향 선형 경로 질의의 질의 패턴은 선형 질의 패턴(linear query pattern)이라 부른다. 정의 9에서 정의된 질의 패턴과 비교되는 것은, 선형 질의 패턴의 경우, 분기 질의 노드(branching query node)가 없다는 것이다.

그리고 정의 4에서 정의된 LPQ내 $o_1\ l_1\ o_2\ l_2\ \ldots\ o_i\ l_i\ o_{i+1}\ l_{i+1}\ \ldots\ o_n\ l_n$와 정의 5에서 정의된 $l_n\ o_n\ l_{n-1}\ o_{n-1}\ \ldots\ l_{i+1}\ o_{i+1}\ l_i\ o_i\ \ldots\ l_2\ o_2\ l_1\ o_1$와 정의 7에서 정의된 BPQ내의 $o_1\ l_1\ o_2\ l_2\ \ldots\ o_i\ l_i\ o_{i+1}\ l_{i+1}\ \ldots\ o_n\ l_n$에 대응되는 LPQ에서 각각 분기 질의 노드가 l_i라 할 때 $o_1\ l_1\ o_2\ l_2\ \ldots\ o_i\ l_i$ 부분을(순방향 일 때) 혹은 $l_i\ o_i\ \ldots\ l_2\ o_2\ l_1\ o_1$ 부분을(역방향 일 때) **전반 가지 패턴**(front twig pattern)이라 부른다. 한편 $o_{i+1}\ l_{i+1}\ \ldots\ o_n\ l_n$ 부분(순방향 일 때)이나 $l_n\ o_n\ l_{n-1}\ o_{n-1}\ \ldots\ l_{i+1}\ o_{i+1}$ 부분을 **후반 가지 패턴**(back twig pattern)이라 부른다. 그리고 두 개의 LPQ가 공통의 전반가지패턴을 가질 때 그 전반 가지 패턴을 **공통 전반 가지 패턴**(common front twig pattern)이라 한다.

추가적으로, 본 논문에서는 질의 패턴에 사용하는 루트 노드, 단말 노드, 결과 노드, 분기 질의 노드라는 용어와 경로 질의에서 사용하는 루트 레이블, 단말 레이블,

결과 레이블, 분기 레이블이라는 용어를 상호 혼용하여 사용한다. 예를 들어, 그림 16에서 //issue[//author/last = "Lee"]의 순방향 LPQ에서 "//issue"는 전반 가지 패턴이고 "//author/last"는 후반 가지 패턴이다. 한편 "issue"는 Q3L의 루트 레이블이다. title, keyword, last는 단말 레이블이다. title은 결과 레이블이다. 그리고 issue와 article는 분기 레이블들이다.

4. XML 인덱싱

현재 기존의 연구에서 사용되고 있는 관계형 데이타베이스에서 구축된 경로 인덱스는 부분 매치 질의를 지원하지 못하고 있다. 본 절에서는 관계형 데이타베이스에서 효과적으로 부분 매치 질의를 지원하기 위하여 레이블 경로에 대한 새로운 인덱스 구조를 제안한다.

1) XML 질의 특성 분석

먼저 XML 문서의 구조적 특성 및 경로 질의 특성을 분석하였다. 그리고 관계 데이타베이스에서 구축된 대표적인 경로 인덱스 XRel, XParent, EPIS 등의 인덱스 구조 및 이를 이용한 경로 질의 처리 과정을 분석하였다. 이의 분석결과로 다음과 같은 사실을 도출해 냈다.

사실 1. XML 문서 구조 분석 결과

XML 문서 내에 존재하는 모든 레이블 노드는 항상 루트로 시작한다. 따라서 특정 레이블 노드까지의 레이블 경로는 항상 동일한 루트 레이블로 시작하는 공통 조상의 프리픽스 문자열들을 가지고 있다.

사실 2. 경로 질의 분석 결과

XPath의 경로 질의는 경로 질의 선두나 혹은 중간에는 조상-자손 관계성 "//"이 존재하나 경로 질의 맨 끝부분에는 나타나지 않는다.

사실 3. 경로 인덱스 구조 분석 결과

레이블 경로 테이블에 저장된 모든 레이블 경로는 항상 길이만 다른 공통의 프리픽스 문자열들을 가지고 있다. 이런 특성으로 경로 질의에 해당하는 레이블 경로 문자열을 비교할 때 레이블 경로 문자열 후반부에서 일치 여부가 판정이 난다.

사실 4. 관계형 데이타베이스의 질의 처리 과정 분석 결과

관계 데이타베이스의 스트링 매치 연산자 LIKE는 찾고자 하는 문자열 패턴이 와일드 카드 "%" 문자로 시작하면 해당 열의 인덱스 존재와 관계없이 테이블 전체를 스캔한다.

본 논문에서는 이런 분석 결과를 토대로 기존의 레이블 경로(순방향 레이블 경로, 정의 1) 문자열 대신 본 논문의 3장 1절에서 정의한 역방향 레이블 경로(정의 2) 문자열을 사용하여 저장한다. 그리고 기존의 순방향 경로 질의(정의 4)는 경로 질의의 파싱단계에서 역방향 경로 질의(정의 5) 형태로 변환되어서, 경로 질의를 SQL 질의로 변환하는 단계에서 역방향 경로 질의 패턴을 이용한 문자열 패턴 매칭으로 변환된다. 순방향 레이블 경로 대신 역방향 레이블 경로를 저장하는 제안방식이 얻는 효과는 다음과 같다.

효과 1. 순방향 형태의 기존 부분 매치 질의 대신 역방향 부분 매치 질의를 이용함으로써 관계 데이타베이스의 문자열 패턴 매칭시 검색 문자열 패턴의 선두에 "%"가 나오는 것을 방지할 수 있다. 이는 현재 관계형 데이타베이스에 지원하는 B+-tree 인덱스를 클러스터링 인덱스로 구축, 사용이 가능하다.

효과 2. 현재 노드의 레이블부터 루트 레이블 순으로 저장하는 역방향 레이블 경로 문자열은 레이블 경로 상에 동일한 레이블이 존재하지 않을 경우 공통의 프리픽스 문자열을 가지고 있지 않으며, 심지어 동일한 레이블을 가지고 있는 경우도 순방향 레이블 경로를 저장한 방식보다 작은 개수의 공통의 프리픽스 문자열을 가져서 문자열 비교시 진위 판정이 빠르다. 즉 선택율이 높다.

2) XML 인덱스 구조

XML 문서는 스키마-레벨과 인스턴스-레벨 정보로 분리되어, 질의 처리에 필요한 정보의 형태로 관계 데이타베이스 테이블에 저장된다. 본 논문에서는 XML 문서에서 발생하는 레이블 경로들과 그 경로들의 레벨정보를 스키마-레벨 정보로 사용한다. 제안하는 레이블 경로 저장 구조는 다음과 같다.

LabelPath(labelpath, pathid, plevel)

LabelPath 테이블은 역방향 레이블 경로 정보를 저장하는 labelpath 열과 경로 식별자인 pathid, 그리고 현재 레이블 경로의 레벨을 저장하는 plevel로 구성되며 labelpath 열에 대하여 클러스터링 인덱스(B+-tree)를 생성하여 데이타의 순서를 물리적으로 유지한다. 기존의 XRel, XParent와 같이 pathid는 레이블 경로 구분을 위한 경로 식별자이다. plevel은 4장 질의 처리에서 사용되는 유용한 정보로 본 논문에서 BPQ 질의 처리를 효과적으로 하기 위하여 XML 문서로부터 필요로 하는 정보 중의 하나다. 이 값은 BPQ내 분기 질의 노드에 해당하는 분기 데이타 노드의 레벨을 찾고자 할 때 사용된다. plevel 정보의 활용은 4장 질의 처리에서 언급한다.

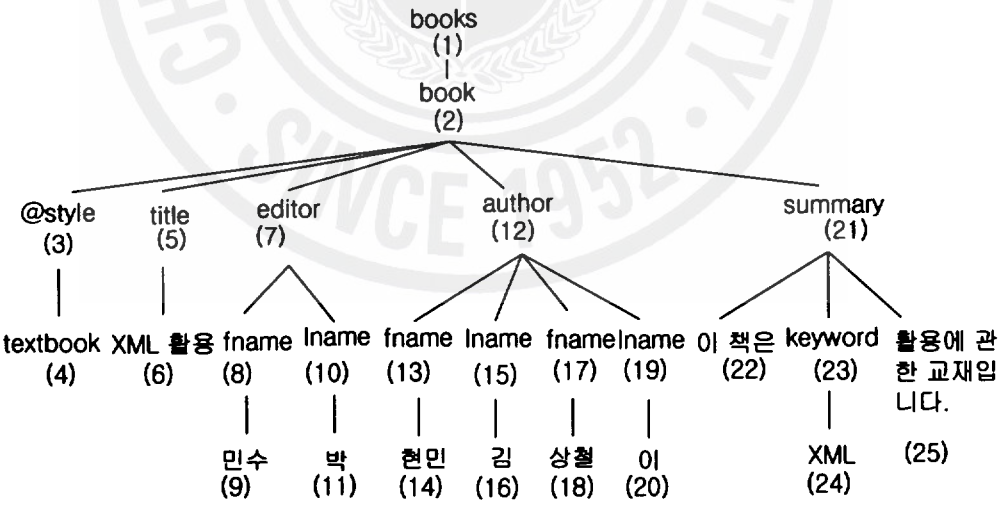


그림 17. XML 문서 트리

Fig. 17. An XML document tree

LabelPath 테이블은 먼저, XML 문서에 포함된 모든 레이블 경로(순방향 레이블 경로)들을 추출하여 이에 해당하는 역방향 레이블 경로들을 저장한다. 이 때 레이블의 구분자를 위해 "\" 대신 "#\"를 사용한다. 예를 들어 순방향 레이블 경로 "/books/book/editor"가 경로 식별자 5를 가지고 있다고 가정하면 이 경로는<"editor#\book#\books#\", 5, 3>와 같은 형태의 역 방향 레이블 경로가 저장된다. 그림 18은 그림 17의 XML 문서 트리내 레이블 경로 정보들을 제안방식의 LabelPath 테이블로 저장한 예를 보여주고 있다.

label_path	pathid	plevel
@vol#\paper#\issue#\	3	3
author#\journal#\issue#\	11	3
author#\magazine#\issue#\	17	3
author#\paper#\issue#\	5	3
first#\author#\journal#\issue#\	12	4
first#\author#\magazine#\issue#\	18	4
first#\author#\paper#\issue#\	6	4
issue#\	1	1
journal#\issue#\	9	2
keyword#\journal#\issue#\	14	3
keyword#\magazine#\issue#\	20	3
keyword#\paper#\issue#\	8	3
last#\author#\journal#\issue#\	13	4
last#\author#\magazine#\issue#\	19	4
last#\author#\paper#\issue#\	7	4
magazine#\issue#\	15	2
paper#\issue#\	2	2
title#\journal#\issue#\	9	3
title#\magazine#\issue#\	16	3
title#\paper#\issue#\	4	3

그림 18. 제안방식의 LabelPath 테이블

Fig. 18. The LabelPath table of the proposed method

그림 19는 2장 1절의 예제 질의 Q1에 대해 제안방식의 인덱스를 사용하여 다시 작성된 SQL문이다.

```
SELECT    e1.document_id, e1.start_position, e1.end_position
FROM      LabelPath p1, Element e1
WHERE     p1.pathid = e1.pathid
          and p1.labelpath LIKE "name#\editor#%\article#%";
```

그림 19. LPQ Q1를 위한 제안방식 SQL 문

Fig. 19. The SQL statement in the proposed method for LPQ Q1

표 1은 제안 방식과 기존의 XML 인덱싱의 성능을 비교 분석한 표이다. 제안 방식의 레이블 탐색 비용은 O(log n)이 소요된다.

표 1. 인덱싱 방법들의 성능 비교

Table 1. The performance comparison of indexing methods

성능관련 요소	XRel	XParent	EPIS	XIR-Branching	제안방식
레이블 경로 탐색방법	테이블 스캔	테이블 스캔	인덱스 스캔+조인	인덱스 스캔	인덱스 스캔
LPQ 처리 인덱스	B+-tree	B+-tree	B+-tree	IR 역 인덱스	B+-tree
레이블 경로 탐색 비용 (LPQ성능)	O(n)	O(n)	O(log(n x m))	O(log n)	O(log n)
조인 횟수	0	0	m -1	0	0

5. 제안 시스템 구조

본 절에서는 본 논문에서 부분 매치 질의를 효과적으로 처리하기 위하여 제안하는 XML 인덱싱과 조인 알고리즘 기법들을 적용하기 위한 목표 시스템을 설계하고 이에 따르는 저장 구조인 데이타베이스 스키마를 설계한다.

1) 시스템 구조

제안 시스템은 기본적으로 XRel[5], XParent[6,7]와 같이 off-the-shelf 관계 데이타베이스 위에 XML 데이타베이스를 구축하는 접근법을 사용하였으며 이 방식은 기존 시스템의 기능을 확장하지 않고 미들웨어 형식으로 구현하여 상업용 데이타베이스에 바로 적용 가능하다는 장점을 가지고 있다. 다음과 같은 설계 목표를 갖는다.

- 저장된 XML 문서에 제약이 없음
  - 유효하거나 well-formed 문서 저장, 혼합 컨텐츠 모델의 문서 저장
- XML 문서의 저장과 검색은 현재의 데이타베이스의 기능만을 이용
  - 데이타 모델, 질의 언어, 인덱스에 대한 확장은 없음
- 상이한 구조를 가지는 대량의 XML 문서 응용에의 적용
- XPath 질의를 사용하며 특히 부분 매치 질의의 효율적 처리를 지원
- 저장된 내부의 정보만으로 원본 XML 문서 재구성을 처리

그림 20은 본 논문에서 제안하는 XML 인덱싱과 조인 알고리즘을 사용하여 질의 처리를 하기 위한 목표 시스템의 구성도이다.

Repository generator를 이용하여 XML 문서의 저장소에 해당하는 데이타베이스를 생성하고 외부의 XML 문서들을 XML Doc. Loader를 이용하여 시스템내로 적재한다. 적재된 문서들에 대하여 XPath 질의가 주어지면 XML Query Processor의 일부분인 XML query parser는 이를 파싱하여 선형 경로 질의로 분할하고 또한 분기 유형별 최대 공통 분기 레벨을 결정하기 위하여 필요한 정보들을 구한다.

LPQ processor와 BPQ processor는 각각 선형 경로 질의와 분기 경로 질의 처리를

담당하는 모듈이며 XML Doc. composer는 처리된 질의 결과(노드 식별자들의 집합)를 가지고 최종 출력 결과물인 XML 문서나 XML fragment를 생성하는 모듈이다.

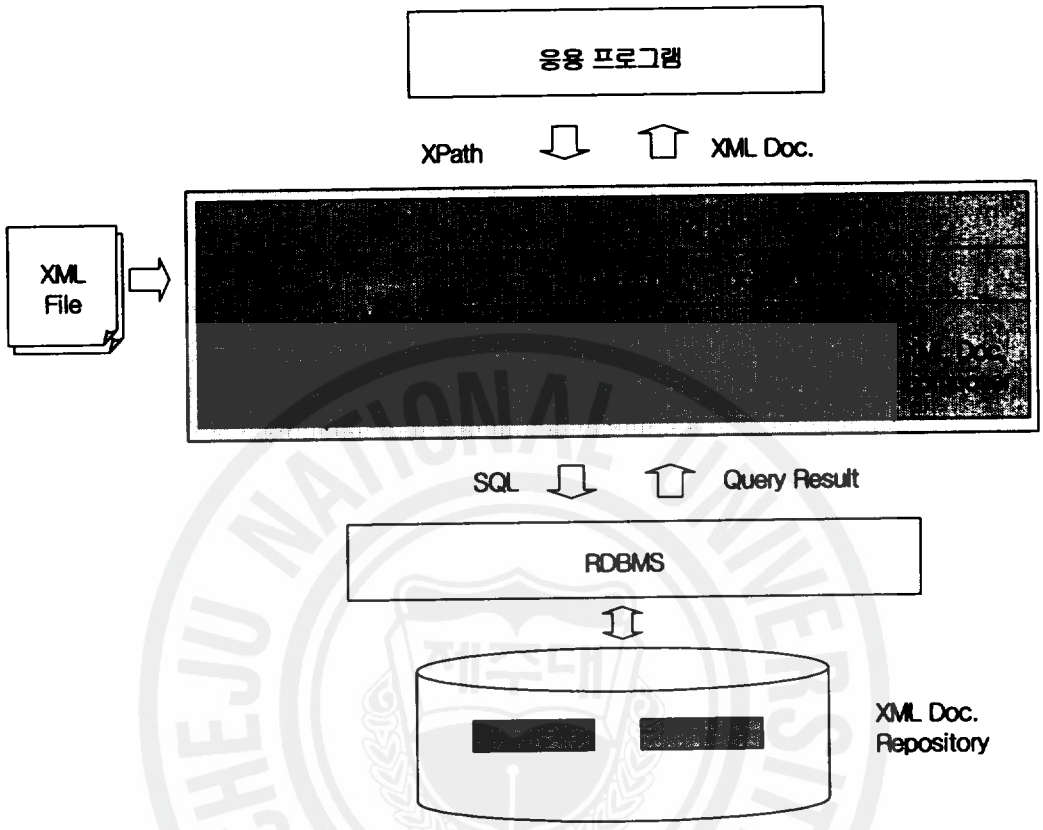


그림 20. 제안 시스템 구조

Fig. 20. The architecture of the proposed system

2) 데이타베이스 스키마

off-the-shelf 데이타베이스 관리 시스템에 XML 문서가 저장될 때 XML 문서를 위한 저장 모델 설계에 관한 문제는 데이타베이스 스키마 설계 문제가 된다. 데이타베이스 스키마는 두 가지로 분류된다. 구조-기반 접근법(structure-mapping approach)은 문서의 구조 정보를 나타내는 DTD를 이용하여 데이타베이스 스키마를 설계하고 모델-기반 접근법(model-mapping approach)은 DTD의 보조 없이 데이타베이스 스키마를 설계한다[6]. 제안 시스템은 DTD가 존재하지 않는 문서에 대해서도 저장을

지원하기 위하여 모델-기반 접근법을 사용한다.

XRel의 sibling, ssibling 열 같이 몇몇 필드는 검색 시 계산과정의 시간 절약을 이해 사용할 수 있으나 갱신에 따른 오버헤드가 커 질의처리에 큰 영향을 미치지 않은 경우 배제하고 계산방식을 선택하였다. 이는 텍스트 값에 대한 인덱싱은 데이타 중심 문서인 경우는 괜찮으나 문서 중심인 경우는 효용 가치에 비해 비용이 커 채택하지 않았다. XML 문서를 저장하기 위하여 본 논문에서는 2개의 테이블 LabelPath, DataNode를 사용한다. 각각의 테이블의 구조와 용도는 다음과 같다.

(1) LabelPath(labelpath, pathid, plevel)

LabelPath 테이블은 XML 문서의 역방향 레이블 경로 문자열을 저장하기 위한 테이블로 labelpath 열이 기본 키가 되고 이 열에 대해 클러스터링 인덱스가 생성된다. labelpath열은 3장 5절에서 언급한 바와 같이 레이블 경로를 저장하는 문자열이다. pathid는 경로를 식별하기 위한 식별자로 정수로 선언되고 plevel은 레이블 경로에서 단말 노드의 레벨을 나타낸다.

(2) DataNode(docid, nid, ntype, cmt, pathid, nlevel, lname, value)

DataNode 테이블은 문서내에 있는 모든 데이타 노드들을 저장한다. 이 테이블은 엘리먼트, 속성, 텍스트 노드에 관한 정보를 통합 저장한다. 기존의 질의 처리 시스템들이 질의 처리를 통한 결과 노드의 식별자를 가져오는 문제(select mode라 지칭)에 대해서만 초점을 두고 구해온 노드 식별자들의 식별자를 가지고 문서의 재구성(reconstruct mode라 지칭)에 대해서는 연구가 다소 부족했다.

몇몇 시스템들은 전체 XML 문서를 구성하는 부분에 대해 외부의 XML 파일을 이용하는 방식을 채택, 사용하고 질의 처리 시스템은 XML 질의에 해당하는 XML 문서 조각을 가져오는 데 사용하였다. 그러나 이러한 방식의 단점은 본질적으로 가져올 데이타 양이 문서 전체의 얼마가 될 것인지에 대한 답이 미정이고 심지어 XML 문서의 특성인 데이타 중심(data-centric)이냐 문서-중심(document-centric)인 가에 따라 좌우되지만 이것도 구분하기가 쉽지 않으며 대개의 문서는 두 개의 특성을 다 가지고 있다. 또한 외부 파일을 이용한 이중 관리 문제로 관계 데이타베이스를 이용한 질의 처리 시스템의 장점이 감소된다.

따라서 본 논문은 이를 해결하는 하나의 방안으로 데이타베이스내에서 전체 문서 혹은 일부 XML 문서 조각을 추출하는 XML 질의에 대해 효과적으로 대응하기 위하여 통합된 구조를 채택하였다. XML 문서에 대한 저장 및 검색 시스템을 위해 관계 데이타베이스를 이용하는 대부분의 시스템은 효율적으로 질의 처리를 수행하기 위하여 XML 문서를 조각내어(shred) 여러 개의 테이블에 저장하는 것이 보편적이다. 그러나 이 방식은 reconstruct mode에서 과도한 오버헤드를 유발한다. 따라서 본 논문에서는 기존의 시스템들에 비해 일정 부분의 중복을 허용하여 질의 처리 성능을 저하시키지 않고 reconstruct mode에서 문서의 재구성 문제를 회피하였다.

제안된 구조의 장점은 XML 문서의 태그가 나타나는 순서와 데이타 노드가 저장된 물리적 순서가 동일하다는 것이다. 이는 질의 결과 최종 구해진 노드 식별자들에 대해 해당 범위의 문서 조각을 추출할 때 단순히 해당 nid에 해당하는 데이타 노드들이 저장된 범위 즉 시작과 끝 위치에 존재하는 튜플들을 검색하여 약간의 처리(태그 명 생성 등)를 통해 원래의 문서를 재구성할 있다는 것이다. 출력을 위한 테이블간 결합이나 정렬이 필요하지 않다.

docid는 해당 노드를 포함하고 있는 문서 식별자이며, nid는 데이타 노드들을 위한 노드 식별자로서 동적 프리픽스 레이블링을 사용한다. 논문에서는 확장된 Deway 방식을 사용한다. ntype은 노드 유형을 구분하기 위하여 사용된다. 1이면 엘리먼트 노드, 2는 속성노드, 그리고 3은 텍스트 노드를 나타낸다. pathid는 해당 노드에서 종료되는 레이블 경로 식별자이며, cmt 열은 XML 문서의 데이타 노드가 텍스트 노드일 경우 효과적인 검색을 위해 사용된다.

앞에서 언급한 바와 같이 데이타-중심인 XML 문서는 텍스트 즉 문자열의 길이가 길지 않지만 문서-중심 XML 문서는 텍스트 길이의 범위가 아주 넓고 크다. 예를 들어 /books/book//[/title/text() = "Readings in Objected-oriented Database Systems"] 같은 질의의 빠른 처리를 위해 텍스트 노드의 value 값에 인덱스를 설정하는 것은 비용에 비해 효용가치가 적다. 그리고 빠른 문서의 재구성 및 질의 처리를 위해 클러스터링 인덱스를 데이타 노드의 식별자에 설정한 상황에서 보조 인덱스를 통한 텍스트 노드의 값 비교 역시 비용이 크다. 따라서 본 논문에서는 비용 대 효과 측면에서 cmt 열에 텍스트 노드의 텍스트 값을 압축한 문자열을 저장하여 향후 문자열 비교 검색시 이를 이용하여 1차적으로 관련없는 데이타를 필터링하여 디스크

접근 횟수를 줄이는 효과를 갖는다.

본 논문에서는 6바이트 이하의 문자열을 가질 경우에는 해당 문자열을 그대로 저장하고, 6바이트 이상의 문자열을 가지는 경우에는 문자열의 맨 앞 3바이트와 문자열의 맨 끝 3바이트를 결합하여 저장하는 방식으로 구현하였다. value열은 텍스트 노드나 속성 노드의 문자열 값을 저장하며 lname 열은 엘리먼트 명 또는 속성명을 저장한다.

DataNode의 기본 키는 (docid, nid)이지만 빠른 검색을 위해 ntype, cmt열을 포함하여 클러스터링 인덱스를 생성한다. 보조키로 (ntype, pathid, docid, nid, cmt) 열 목록에 생성한다. 데이타 노드 검색에 따른 키의 활용을 보면 다음과 같다. SQL 문장의 단순화를 위해 docid 값은 생략하며 PARENT(nid)는 주어진 노드 식별자의 부모 노드에 해당하는 노드 식별자를 반환하는 함수이다.

① 엘리먼트 노드 존재 검사 질의

- 질의 형태: //a//b/c
- SQL 문장: select distinct docid, nid from DataNode n, LabelPath p
  where ntype = 1 and n.pathid = p.pathid
  and p.labelpath LIKE 'c#\b#%\a#%';

② 텍스트 노드 존재 검사 질의

- 질의 형태: //a//b/c[/text()]
- SQL 문장: select distinct docid, PARENT(nid) from DataNode n, LabelPath p
  where ntype = 3 and n.pathid = p.pathid
  and p.labelpath LIKE 'c#\b#%\a#%';

③ 길이가 짧은 텍스트 노드 값 비교 검사 질의

- 질의 형태: //a//b/c[/text()= 'Apple']
- SQL 문장: select distinct docid, PARENT(nid) from DataNode n, LabelPath p
  where ntype = 3 and n.pathid = p.pathid and n.cmt = 'Apple'
  and value = 'Apple' and p.labelpath LIKE 'c#\b#%\a#%';

④ 길이가 긴 텍스트 노드 값 비교 검사 질의

- 질의 형태: //a//b/c[/text()= 'Readings in Objected-oriented Database Systems']
- SQL 문장: select distinct docid, PARENT(nid) from DataNode n, LabelPath p

where ntype = 3 and n.pathid = p.pathid and n.cmt = 'Reaems'
and value = 'Readings in Objected-oriented Database Systems'
and p.labelpath LIKE 'c#\b#%\a#%';

⑤ 속성 노드 존재 검사 질의

- 질의 형태: //a//b/c[/@name]
- SQL 문장: select distinct docid, PARENT(nid) from DataNode n, LabelPath p
where ntype = 2 and n.pathid = p.pathid and p.labelpath LIKE
'@name#\c#\b#%\a#%';

⑥ 속성 노드 값 비교 검사 질의

- 질의 형태: //a//b/c[/@name = 'R']
- SQL 문장: select distinct docid, PARENT(nid) from DataNode n, LabelPath p
where ntype =2 and n.pathid = p.pathid and n.value= 'R'
and p.labelpath LIKE '@name#\c#\b#%\a#%';

IV. 제안 XML 조인

1. XML 조인 알고리즘

본 장은 3장 5절에서 제안한 시스템 구조를 기반으로 분기 경로 질의를 처리하기 위한 알고리즘을 제안한다. XML 질의 처리의 일반적인 과정은 다음과 같다.

① 먼저 트리 패턴의 질의를 분기 조건을 가지는 않는 여러 개의 선형 경로 질의로 분할한다. ② 분할된 각각의 경로 질의에 속하는 데이타 노드들을 구한다. ③ 마지막으로 최종 경로 질의에 속하는 데이타 노드를 구하기 위해 ②에서 구한 각각의 경로 질의 결과를 조인을 이용하여 결합한다. 따라서 질의 처리에 대한 연구들은 선형 경로 질의를 효과적으로 처리하기 위한 방법과 중간 결과 크기를 줄이기 위한 방법으로 나누어지고 있다.

본 논문에서 제안하는 XML 질의 처리 과정은 다음과 같다. ① 우선 하나의 분기 경로 질의를 단말 레이블 개수만큼의 선형 경로 질의로 분할한다. ② 빠른 선형 경로 질의 처리를 위해 제안한 XML 인덱스를 이용하여, 분할된 각각의 선형 경로 질의에 대해 질의 결과를 구한다. ③ 최종 경로 질의 결과를 얻기 위하여 선형 경로 질의에 속하는 데이타 노드들을 결합한다. 이 때 불필요한 데이타 노드들 간의 비교 횟수를 줄여 조인 비용을 최소화한다.

본 논문에서는 선형 경로 질의 결과를 결합할 때 최종 결과에 포함될 수 없는 즉 데이타 노드간의 유효하지 않은 결합을 위한 비교를 회피하기 위하여 조인에 앞서 유효한 레이블 경로 결합을 식별한다. 다음으로 구해진 유효한 각 결합에 속하는 데이타 노드들끼리의 비교를 수행한다. 이를 위해 주어진 분기 경로 질의와 저장된 XML 문서의 구조적 정보를 활용한 새로운 XML 조인 방법을 제시한다. 먼저 기존의 조인 알고리즘을 분석하고 이를 개선한 조인 알고리즘을 제시한다.

1) 기존 조인 알고리즘의 분석

기존 조인 알고리즘들의 분기 경로 질의(BPQ) 처리 과정은 다음과 같다.

① 분기 경로 질의를 선형 경로 질의(LPQ)로 분할한다.

② 분할된 각각의 선형 경로 질의에 대해 해당하는 레이블 경로들을 구하고 이에 소속된 데이타 노드들의 집합을 구한다.

③ 결과 레이블을 포함하는 질의에 소속된 데이타 노드 집합을 나머지 선형 경로 질의에 소속된 데이타 노드 집합들과의 비교(포함관계, 프리픽스 매치)를 통해 결과 레이블 소속 데이타 노드 집합내 튜플들을 감축하면서 최종 질의 결과를 구한다.

표 2는 n개의 단말 레이블을 가지는 분기 경로 질의에 대해 XRel, EPIS, XParent, XIR-Branching에서의 조인 비용을 비교하여 요약한 표이다.

표 2. XRel, XParent, EPIS, XIR-Branching의 조인 비용 비교

Table 2. The comparison of join costs in XRel, XParent, EPIS, and XIR-Branching

	XRel	XParent	EPIS	XIR-Branching
분할 LPQ수	2n	n	2n	n
LPQ 처리 조인수	2n =-Joins	n =-Joins	2n =-Joins	n =-Joins
공통 조상 찾는 조인 수	2n Θ-Joins	n =-Joins	2n Θ-Joins	0
결과노드 찾는 조인 수	(n-1) x 2Θ-Joins	(n-1) =-Joins	(n-1) x 2Θ-Joins	(n-1) p=-Joins
총 조인 수	2n =-Joins + (4n-2) Θ-Joins	3n-1 =-Joins	2n =-Joins + (4n-2) Θ-Joins	n =-Joins + (n-1) p=-Joins

XRel과 EPIS는 구간-기반 레이블 기법을 사용하여 처리하며, 기본적으로 XParent나 XIR-Branching보다 분할되는 선형 경로 질의 개수가 많다. 따라서 분기 조건이 많은 BPQ일수록 BPQ 질의 처리를 위한 총 조인 횟수는 증가한다.

XParent나 XIR-Branching은 BPQ를 단말 레이블 개수만큼의 LPQ들로 분할하며

분할되는 LPQ의 개수는 동일하다. XParent는 조상 노드를 찾기 위하여 조상 테이블(Ancestor)과 분할된 LPQ 개수만큼 조인을 유발하나 XIR-Branching은 프리픽스 레이블 기법을 이용, 자체 노드 식별자로서 조상 노드들을 가지고 있어 추가 조인을 필요로 하지 않는다. 다만 XIR-Branching이 두 레이블 간에 공통의 프리픽스 문자열을 찾기 위한 추가적인 작업이 필요하다. 여기서는 최대 공통 분기 레벨까지의 프리픽스 공통 문자열을 찾기 위한 이퀴조인을 p=-join으로 표기하였다.

최근에 연구된 XIR-Branching(2장 4절 2항)의 조인 처리 알고리즘의 처리 절차는 다음과 같다. ① 하나의 BPQ를 여러 개의 선형 경로 질의(LPQ)로 분할한다. ② 각 LPQ에 속하는 데이타 노드들(NPSet)을 구한다. 이 때 데이타 노드들에 해당하는 레이블 경로도 함께 포함시킨다. ③ 각 LPQ의 질의 결과를 결합하기 위하여 먼저 각 NPSet내에 있는 레이블 경로를 이용, 두 레이블 경로의 프리픽스를 비교하면서 공통 가지 패턴에 해당하는 두 레이블 경로간의 최대 공통 분기 레벨을 구한다. 다음으로 최대 공통 분기 레벨이 존재하면 레이블 경로에 속하는 데이타 노드 경로 즉 노드식별자(nodepath 열)에 대해 루트부터 최대 공통 분기 레벨까지의 서브 문자열을 각각 구한다. 구해진 프리픽스 문자열은 분기 데이타 노드의 노드 식별자가 되며 이것들을 비교하여 일치하면 BPQ의 최종 결과에 포함시킨다.

2) 제안 조인 알고리즘의 기본 아이디어

본 논문은 새로운 조인 알고리즘을 제안하기에 앞서 기존의 조인 알고리즘 중에서 분기 경로 질의 처리시 조인 비용이 가장 작은 XIR-Branching의 조인 과정을 분석한다. 다음은 XIR-Branching의 각 LPQ 결과 간에 결합하는 단계를 세부적으로 분석한 내용이다. BPQ가 전체 매치 질의일 경우와 부분 매치일 경우로 나누어 분석한다.

(1) 전체 매치 질의 처리

BPQ가 "/a/b[/c]/d/e"라 할 때 2 개의 LPQ P_1: /a/b/c, P_2: /a/b/d/e가 생성된다. 최종 결과를 구하기 위한 결합은 P_1에 속하는 선형 경로 질의 결과와 P_2에 속하는 선형 경로 질의 결과 간의 공통 프리픽스 문자열을 찾는 조인을 이용하게 되며 이 경우

최대 공통 분기 프리픽스는 "/a/b"가 된다.

여기서 결합에 참가하는 LPQ P_1에 해당하는 레이블 경로의 집합은 "a/b/c"로 레이블 경로의 개수는 1개이며, P_2에 해당하는 레이블 경로의 집합은 "a/b/d/e"로 역시 레이블 경로의 개수는 1개이다. 따라서 결합을 통하여 레이블 경로 사이에 만들어지는 경로 연결의 개수는 1개이며 최대 공통 분기 프리픽스 "/a/b"를 만족한다. 이는 유효한 레이블 경로간의 결합이며 이를 **유효한 레이블 경로 연결**(valid label path connection)이라 부른다. 그러나 부분 매치 질의인 경우는 유효하지 않는 레이블 경로 연결이 다수 존재한다.

(2) 부분 매치 질의 처리

부분 매치 질의는 다음과 같이 4 가지 결합 유형으로 나누어 살펴볼 수 있다.

- 결합 유형 1: 분기 레이블 앞뒤에 '//'가 존재하지 않는 경우, ex) /a/b[//c]/d/e
- 결합 유형 2: 분기 레이블 앞에 '//'가 존재하는 경우, ex) /a//b[/c]/d/e
- 결합 유형 3: 분기 레이블 뒤에 '//'가 존재하는 경우, ex) /a/b[/c]//d/e
- 결합 유형 4: 분기 레이블 앞뒤에 '//'가 존재하는 경우, ex) /a//b[/c]//d/e

결합 유형 2의 BPQ "/a//b[/c]/d/e"를 고려 해 보자. BPQ "/a//b[/c]/d/e"는 LPQ P_1: /a//b/c, P_2: /a//b/d/e/로 분할되며, 공통 전반 가지 패턴은 "/a//b"가 된다. 논의를 간단히 하기 위하여 LPQ P_1에 해당하는 레이블 경로의 집합은 그림 21의 ①, ②, ③, ④, LPQ P_2에 해당하는 레이블 경로의 집합은 ⓐ, ⓑ, ⓒ, ⓓ, ⓔ, ⓕ 로 구성된다고 가정하자. 이 때 $[/X]^*$는 0 개 이상의 레이블 경로를 의미하며 각 레이블 경로의 밑줄 부분은 각 레이블 경로의 분기 프리픽스 부분을 나타낸다. 화살표는 유효한 레이블 경로 연결을 나타낸다.

①번 레이블 경로는 최대 공통 전반 가지 패턴 "/a//b"에 해당하는 분기 프리픽스로 "/a/b"를 가지므로 LPQ P_2에 해당하는 레이블 경로의 집합 원소 중 ⓐ번 형태의 레이블 경로와의 결합만이 유효한 레이블 경로 간 결합이다. 최대 공통 분기 프리픽스로 ②번 레이블 경로는 "/a/d/b"를, ③번 레이블 경로는 "/a/a/b"를 가진다. 따라서 ②번 레이블 경로는 ⓒ와 ③번 레이블 경로는 ⓔ번 형태의 레이블 경로와의 결합만이 유효한 레이블 경로 연결이다. 그럼에도 불구하고 기존 조인은 P_1에 해당하

는 ①, ②, ③, ④ 레이블 경로가 P_2에 해당하는 ⓐ, ⓑ, ⓒ, ⓓ, ⓔ, ⓕ 레이블 경로와 연결되고 ①, ②, ③, ④ 레이블 경로에 속하는 모든 데이타 경로 노드들이 ⓐ, ⓑ, ⓒ, ⓓ, ⓔ, ⓕ형태의 레이블 경로에 속하는 모든 데이타 경로 노드들과 비교된다. [/X]*가 단지 0 개의 레이블을 가진다고 가정하면 4 x 6 = 24개의 레이블 경로 연결 중 유효한 레이블 경로 연결은 3개뿐이고 21개는 유효하지 않은 결합이다. 이는 각 레이블 경로에 속하는 데이타 경로 수가 하나 이상인 점을 감안할 때, 유효하지 않은 즉 최종 결과에 포함될 수 없는 데이타 경로들 사이에 비교를 과다하게 수행한다는 것을 말한다.

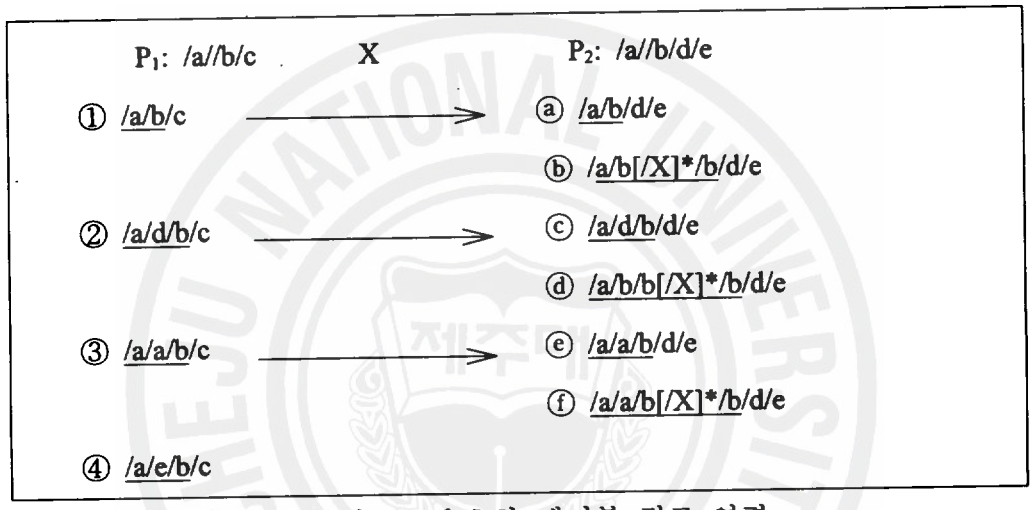


그림 21. 유효한 레이블 경로 연결

Fig. 21. The connections of valid label paths

다른 경우에 대해서도 분석을 수행한 결과 다음과 같은 사실을 얻었다.

- 결합 유형 1: /a/b[//c]/d/e, 모든 조합이 유효한 레이블 경로 연결을 생성
- 결합 유형 2: /a//b[/c]/d/e, 유효하지 않은 레이블 경로 연결이 존재
- 결합 유형 3: /a/b[/c]//d/e, 모든 조합이 유효한 레이블 경로 연결을 생성
- 결합 유형 4: /a//b[/c]//d/e, 유효하지 않은 레이블 경로 연결이 존재

따라서 기존의 조인 알고리즘들은 분기 경로 질의를 처리하는 과정에서, 특히 LPQ 질의 처리 결과들을 결합하는 단계에서 모든 데이타 경로들 간에 결합을 시도(비교)하여, 결과적으로 불필요한 즉 동일 레이블 경로 상에 존재하지도 않는 데이

타 경로 노드들을 비교하였다.

제안하는 조인 알고리즘은 분기 경로 질의의 최종 결과를 구하기 위해 선형 질의 결과에 소속된 데이타 노드들끼리의 비교를 통해 결과 튜플을 구하는 과정에서, 최종 질의 결과에 포함되지 않는 데이타 노드들 간의 비교를 회피하는 방식을 채택함으로써 조인에 따르는 전체 비교 연산 횟수를 줄인다. 이를 위해 제안하는 알고리즘은 다음과 같은 단계로 질의를 처리한다. ① 먼저, 분기 경로 질의를 선형 경로 질의로 분할한다. ② 분할된 선형 경로 질의에 해당하는 레이블 경로 집합들을 구한다. ③ 각각의 선형 경로 질의에 해당하는 레이블 경로 집합들 간에 최대 분기 레벨 식별을 통해 유효 레이블 경로 연결을 식별한다. ④ 각 선형 경로 질의에 해당하는 레이블 경로에 소속된 데이타 노드 집합간의 튜플 비교시 식별된 유효 레이블 경로 연결에 포함되는 데이타 노드들끼리만 비교를 수행한다. 결과적으로 제안 방식은 최종 결과를 얻기 위하여 데이타 경로 간 결합은 유효한 레이블 경로 연결에 범위 내에서만 제한적으로 비교된다.

그림 22는 레이블 경로 연결과 최종 결과와의 관계를 보여 주는 그림이다. 유효한 레이블 경로 연결을 이용한 비교 연산이 그렇지 않은 경우에 비해 비교 범위가 작음을 알 수 있다. 반면 유효한 레이블 경로 연결에 속하는 데이타 경로들만으로 수행하는 비교과정에서도 일부 데이타 경로들은 유효한 레이블 경로 연결에 속하지만 최종 결과에 포함되지 않는 경우도 존재한다. 그러나 본 논문에서 제안하는 알고리즘은 비교 범위를 대폭 축소시켜 상당히 많은 비교횟수를 감소시켜 주는 장점을 가지고 있다.

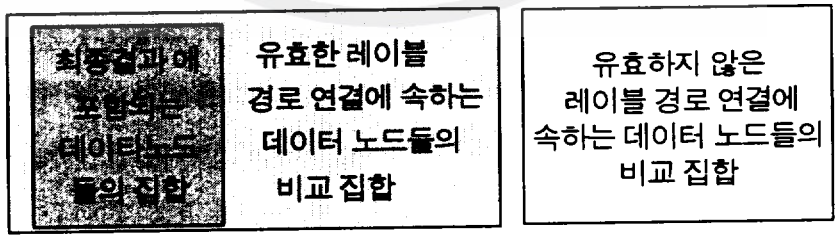


그림 22. 유효한 레이블 경로 연결과 최종 결과와의 관계

Fig 22. The relationship between valid label path connections and final result

유효한 레이블 경로 연결을 식별하는 문제는 최대 공통 가지 패턴에 해당하는 최대 공통 프리픽스를 구하는 문제가 되며 이는 결국 최대 공통 분기 레벨을 찾는 문제로 귀결된다. 최대 공통 분기 레벨 계산을 위해 본 논문에서는 분기 노드를 3가지 분기 유형으로 분류하여 다음과 같이 계산한다.

① 분기 노드 유형1: 분기 레이블 앞에 조상-자손 관계("//")가 비존재

ex) /a/b/c[/d/e], /a/b/c[//d/e]

② 분기 노드 유형2: 분기 레이블 앞에만 조상-자손 관계("//")가 존재

ex) //a/b/c[/d/e], //a/b//c[/d/e]

③ 분기 노드 유형3: 분기 레이블 앞, 뒤 조상-자손 관계("//")가 존재

ex) //a/b/c[//d/e], //a/b//c[//d//e]

분기 노드 유형 1인 경우는 경로 질의의 파싱 정보만으로 식별 가능하며 "/a/b/c[/d/e]"인 경우에 최대 공통 분기 레벨은 3이다. 분기 유형 2인 경우는 경로 질의의 파싱 정보와 단말 노드의 레벨 정보를 이용하여 계산한다. "//a/b/c[/d/e]"인 경우 최대 공통 분기 레벨은 먼저 "//a/b/c[/d/e]"의 단말 레이블인 "e"의 레벨을 LabelPath 테이블의 plevel 열에서 구한다. 다음으로 "//a/b/c[/d/e]" 질의에서 분기 조건내 레이블 개수 값을 구한다. 구하는 최대 공통 분기 레벨 값은 단말 레이블 레벨 값에서 분기 조건내 레이블 개수를 뺀 값으로 정의되며, 만일 단말 "e" 단말 레이블 레벨 값이 9라고 가정하면 9-2 = 7이 된다. 마지막으로 분기 노드 유형이 3인 경우의 최대 공통 분기 레벨 값 계산은 먼저 분할된 선형 질의에 해당하는 두 개의 레이블 경로 간에 최대 공통 전반 가지 패턴을 만족하는 분기 프리픽스 문자열을 찾는다. 그리고 해당하는 두 개의 레이블 경로 간 공통 분기 프리픽스가 2개 이상 존재할 때는 그 중에서 레이블 개수가 제일 큰 최대 공통 분기 프리픽스 문자열을 선택한다. 마지막으로 구해진 최대 공통 분기 프리픽스 문자열내의 레이블 개수를 센다.

설명의 편의를 위해 순방향 레이블 경로를 기반으로 논의를 하였으며 논문에서는 역방향 레이블 경로를 사용하며 이를 통한 최대 공통 분기 레벨을 계산한다. 분기 노드 유형이 3인 경우에 최대 공통 분기 레벨을 찾기 위한 내부 과정에서 분기 질의 노드에 해당하는 최대 공통 프리픽스 문자열 대신 최대 공통 포스트픽스 문자열을 구한다는 것 이외에는 동일하다.

3) 제안 조인 알고리즘

본 논문에서 제안하는 질의 처리 알고리즘 VPCJoin은 VPCJoin_PM과 VPCJoin_PP로 나누어진다. 질의 처리 알고리즘 VPCJoin은 먼저 하나의 분기 경로 질의를 역방향 선형 경로 질의 형태인 BLPQ을 사용하여, 루트 노드로부터 단말 노드까지에 해당하는 각각의 BLPQ로 분할한다. 다음 단계로 VPCJoin_PM 알고리즘을 이용하여 유효한 레이블 경로 연결들의 집합 PathJoin을 구한다. 마지막 단계로 VPCJoin_PP 알고리즘을 이용하여 결합된 유효한 레이블 경로 연결 집합내 각각의 경로와 이에 속하는 데이타 노드 집합간의 조인을 수행한다. BLPQ의 처리는 식(2)와 같다.

$$\Pi_{PJColumnlist_i}(\text{PathJoin}_i \bowtie_{\text{pathidi=pathid}} (\sigma_{value\ \theta\ v}(\text{DataNode}))) \quad (식2)$$

*PJColumnlisti*는 BLPQ P_i 질의에 대해 PathJoin_i와 조인을 수행한 결과 테이블의 열 목록으로 표 3과 같이 정의되어 있다. 여기서, "*value* θ *v*" 는 처리될 BLPQ내에 포함된 분기 조건 표현식의 단말 레이블에 대한 선택 조건(selection condition)이다.

그림 23 VPCJoin_PM 알고리즘은 유효한 레이블 경로 연결 집합을 구하기 위한 알고리즘으로 레이블 경로 결합을 위하여 파싱단계에서 얻은 분기 노드의 타입(bnodetype)에 따라 분할된 BLPQ P_1의 레이블 경로들의 집합 $LabelPathSet_1$을 구한다. 이후 (i-1)번째 BLPQ P_{i-1}와 이웃한 i번째 BLPQ P_i의 레이블 경로들 집합($LabelPathSet_{i-1}$, $LabelPathSet_i$) 간의 연결을 위해 분기 질의 노드에 해당하는 레이블 경로상의 분기 레이블 노드를 각각의 레이블 경로에 대하여 구하여, 두 레이블 경로에 대한 공통 분기 레이블 노드의 레벨 값을 찾는다. 값이 존재하면 이 값을 이용하여 루트로부터 분기 노드 레벨까지의 포스트픽스 레이블 서브-경로 문자열을 각각의 레이블 경로에 대하여 구하고 비교를 수행한다. 같은 서브-경로 문자열을 가지면 연결 가능하며 이 때 분기 레이블 노드의 레벨 값이 여러 개 존재할 시는 최대 공통 분기 레벨만을 선택한다. 다른 값들은 중복이다. 연결 후 포스트픽스 열은 불필요하다. 나머지 레이블 경로들의 집합에 대하여 반복 작업을 수행한다. 최종적으로 r개의 BLPQ의 레이블 경로들을 모두 연결한 *JoinedPath*, 즉 *PathJoin*을 반환한다.

Algorithm VPCJoin_PM

Input: branching path query P, LabelPath table

Output: set of valid connection of label paths, PathJoin matching P

begin

{ Assume that the path query P has r backward linear path querys P_1, P_2, ... , P_r

, that P_1 is the first path query

, that P_r is the last path query

, that the path query P_i has the branching node type $bnodetype_i$

, and that the path query P_r has the result query type $rqtype_r$. }

for each of the remaining path query $P_i(i=1,...,r)$ **do**

begin

case when i = 1 and r = 1 and $rqtype_i$ = "A" **then**

set $LPColumnList_i$ as the column, $pathid_i$;

when i = 1 and r = 1 and $rqtype_i$ = "B" **then**

set $LPColumnList_i$ as a set of the columns, $pathid_i$, $blevel_i$;

when i = 1 and r ≠ 1 and $bnodetype_i$ < 3 **then**

set $LPColumnList_i$ as a set of the columns, $pathid_i$, $blevel_i$, $postfix_i$;

when i = 1 and r ≠ 1 and $bnodetype_i$ = 3 **then**

set $LPColumnList_i$ as a set of the columns, $pathid_i$, $labelpath_i$;

when i > 1 and i <= r and $bnodetype_i$ < 3 **then**

set $LPColumnList_i$ as a set of the columns, $pathid_i$, $blevel_i$, $postfix_i$;

when i > 1 and i <= r and $bnodetype_i$ = 3 **then**

set $LPColumnList_i$ as a set of the columns, $pathid_i$, $labelpath_i$;

end // of case

obtain $LabelPathSet_i$ having $LPColumnList_i$ by matching with

the path query P_i;

if (i = r and $rqtype_i$ = "B") **then**

begin

set P_{r+1} as a FrontTwigPattern of path query P_i, FTP_i ;

add $labelpath_{r+1}$ to $LPColumnList_i$ by matching with the path query P_{r+1} ;

end

if (i = 1) **then** set $JoinedPath_1$ as $LabelPathSet_1$;

else

begin

case when i = 1 and r ≠ 1 and $bnodetype_i$ < 3 **then**

set $JColumnList_i$ as a set of the columns, $pathid_i$, $blevel_i$, $postfix_i$;

when i = 1 and r ≠ 1 and $bnodetype_i$ = 3 **then**

set $JColumnList_i$ as a set of the columns, $pathid_i$, $labelpath_i$;

when i > 1 and i < r and $bnodetype_i$ < 3 **then**

set $JColumnList_i$ as a set of the columns, ((for k=2, ..., i) $pathid_{k-1}$, $blevel_{k-1}$), $pathid_i$, $blevel_i$, $postfix_i$;

when i > 1 and i < r and $bnodetype_i$ = 3 **then**

set $JColumnList_i$ as a set of the columns, ((for k=2, ..., i) $pathid_{k-1}$, $blevel_{k-1}$), $pathid_i$, $labelpath_i$;

when i > 1 and i = r and $rqtype_i$ = "A" **then**

set $JColumnList_i$ as a set of the columns, ((for k=2, ..., i) $pathid_{k-1}$, $blevel_{k-1}$), $pathid$;

when i > 1 and i = r and $rqtype_i$ = "B" **then**

set $JColumnList_i$ as a set of the columns, ((for k=2, ..., i) $pathid_{k-1}$, $blevel_{k-1}$), $pathid_i$, $blevel_i$;

end // of case

set $JoinedPath_i$ as the table joined through a postfix match,

$\Pi_{JColumnList_i}$ (σ $postfix_{i-1}$ = $postfix_i$ ($JoinedPath_{i-1}$ X $LabelPathSet_i$));

```
    end //of for
    set PathJoin as the last joined path table, JoinedPath_r;
  return  PathJoin
end
```

그림 23. VPCJoin_PM 알고리즘

Fig. 23. Algorithm for VPCJoin_PM

한편, $LPColumnList_i$는 질의 P_i에 해당하는 레이블 경로 집합 $LabelPathSet_i$의 열 목록이며 $JColumnList_i$는 $JoinedPath_i$의 열 목록으로 표 3과 같다. 여기서, 분기 레벨 식별을 위해 사용하는 분기 노드 유형은 $bnodetype_i$이며 $pathid_i$는 경로질의 P_i에 매칭되는 LabelPath 테이블의 *pathid*이다. $labelpath_i$는 분할된 경로 질의 P_i에 매칭되는 LabelPath 테이블의 labelpath열 값이며, $blevel_i$는 BLPQ P_i, P_{i+1}의 최대 공통 분기 레이블의 레벨 값을 나타낸다. BLPQ P_i에 해당하는 레이블 경로들의 집합 $labelpath_i$의 루트부터 $blevel_i$ 값까지의 포스트픽스 서브 레이블 경로를 $postfix_i$로 표기한다. 결과 선형 질의 유형 *rqtype*은 결과 LPQ의 질의 결과 유형을 나타내며 결과 레이블 노드가 단말노드이면 "A"이고 단말노드가 아니면 "B"가 된다. "B" 타입의 질의에서 결과 레이블 노드는 분기 노드가 된다.

개념적으로 레이블 경로간의 결합은 Postfix_matching($labelpath_i$, $labelpath_{i+1}$, Front TwigPattern(P_i), BackTwigPattern(P_i), BackTwigPattern(P_{i+1})) > 0 형태가 되며, 본 논문에서는 유효한 레이블 경로 연결을 효과적으로 찾기 위한 방법을 제시한다. 이 방법의 핵심은 두 경로 질의에 해당하는 레이블 경로 간에 최대 공통 분기 레벨을 적은 비용으로 빨리 찾는 것이다. 이를 위해 본 논문에서는 분기 노드 유형을 3가지로 분류하여 파싱단계에서 최대한 정보를 획득하고, 이를 조인 단계에서 활용한다. 두 레이블 간의 비교를 통하여 최대 공통 분기 레벨에 해당하는 포스트픽스 매치 부분을 찾는 이퀴 조인 비용을 최대한 감소시키는 것이 목적이다.

그림 24는 P_i와 P_{i+1}의 최대 공통 분기 레벨 값 $blevel_i$를 계산하는 알고리즘이다. 분기 노드 유형이 1, 2이면 1개의 경로 질의만으로도 최대 공통 분기 레벨을 식별할 수 있으며 최대 공통 분기 레벨은 분기 노드 유형이 1이면 P_i의 분기레벨 값

FrontTwigPatternCnt(P_i)이 되며, 2이면 plevel$_i$ - BackTwigPatternCnt(P_i) 값이 된다. *bnodetype*$_i$이 3일 때는 질의 P_i에 해당하는 레이블 경로 집합 LabelPathSet$_i$와 질의 P_{i+1}에 해당하는 레이블 경로 집합 LabelPathSet$_{i+1}$과 결합하는 과정에서 계산된다.

Algorithm BLEVELi

Input: path queries, P_i and P_{i+1},
branching node types of path query P_i and P_{i+1}, *bnodetype*$_i$, *bnodetype*$_{i+1}$,
label paths of path queries P_i and P_{i+1}, *labelpath*$_i$, *labelpath*$_{i+1}$,
a path level of path query P_i, plevel$_i$,
current path query number i,
result path query number r

Output: maximum common branch level of P_i, *blevel*$_i$

begin

case when *bnodetype*$_i$ = 1 **then**
blevel$_i$ = FrontTwigPatternCnt(P_i);
when *bnodetype*$_i$ = 2 **then**
blevel$_i$ = plevel$_i$ - BackTwigPatternCnt(P_i);
when *bnodetype*$_i$ = 3 **then**
blevel$_i$ = MaxCommonBranchLevel(*labelpath*$_i$, *labelpath*$_{i+1}$,
CommonFrontTwigPattern(P_i,P_{i+1}), BackTwigPattern(P_i),
BackTwigPattern(P_{i+1}));

end // of case

end

그림 24. BLEVELi 알고리즘

Fig. 24. Algorithm for BLEVELi

Algorithm FrontTwigPatternCnt

Input: a i-th path query P_i,

Output: length of front twig pattern of path queries P_i

```
begin
  set t as the front twig pattern of path queries P_i obtained by definition 10;
  set n as the number of labels in pattern t;
  return n ;
end
```

그림 25. FrontTwigPatternCnt 알고리즘

Fig. 25. Algorithm for FrontTwigPatternCnt

표 3. $LabelPathSet_i$와 $JoinedPath_i$가 가지는 열 목록

Table 3. The column list of $LabelPathSet_i$ and $JoinedPath_i$

열 목록	bnodetype = 1	bnodetype = 2	bnodetype = 3
$LabelPathSet_1$ (i=1 ∧ r=1)	$pathid_1$(, if rqtype ="B" $blevel_1$ 추가)	$pathid_1$(, if rqtype ="B" $blevel_1$ 추가)	$pathid_1$(, if rqtype ="B" $blevel_1$ 추가)**
$LabelPathSet_1$ (=$JoinedPath_1$) (i=1 ∧ r≠1)	$pathid_1$, $blevel_1$, $postfix_1$	$pathid_1$, $blevel_1$, $postfix_1$	$pathid_1$, $labelpath_1$
$LabelPathSet_i$ (1< i≤ r)	$pathid_i$, $blevel_i$, $postfix_i$	$pathid_i$, $blevel_i$, $postfix_i$	$pathid_i$, $labelpath_i$,
$JoinedPath_i$ (=$JoinedPath_{i-1}$ X $LabelPathSet_i$) (1< i < r)	(for(k=2,...,i) $pathid_{k-1}$,$blevel_{k-1}$), $pathid_i$, $blevel_i$, $postfix_i$	(for(k=2,...,i) $pathid_{k-1}$,$blevel_{k-1}$), $pathid_i$, $blevel_i$, $postfix_i$	(for(k=2,...,i) $pathid_{k-1}$,$blevel_{k-1}$), $pathid_i$, $labelpath_i$
$JoinedPath_r$ (=PathJoin) (1 < i = r)	(for (k=2,...,i) $pathid_{k-1}$,$blevel_{k-1}$), $pathid_i$(, if rqtype ="B" $blevel_i$ 추가)	(for (k=2,...,i) $pathid_{k-1}$,$blevel_{k-1}$), $pathid_i$(, if rqtype ="B" $blevel_i$ 추가)	(for (k=2,...,i) $pathid_{k-1}$,$blevel_{k-1}$), $pathid_i$(, if rqtype ="B" $blevel_i$ 추가 **) "A"타입은 비존재

표 3은 분기 노드 유형별 $LabelPathSet_i$와 $JoinedPath_i$가 가지는 열 목록을 나타낸다. 열 목록은 질의가 결과 질의 또는 중간 질의 여부에 따라 달라지며 결과 질의

일 경우에는 결과 질의 노드가 단말 노드 혹은 분기 노드 여부에 따라 다르다. $LabelPathSet_i$는 질의 매칭을 통해 LabelPath 테이블에서 구해지는 열 목록을, $JoinedPath_i$는 질의 P_i부터 P_{i-1} 질의까지의 레이블 경로 결합 테이블인 $JoinedPath_{i-1}$와 $LabelPathSet_i$을 조인하여 구해지는 열 목록을 나타낸다.

Algorithm BackTwigPatternCnt

Input: a i-th path query P_i,

Output: length of back twig pattern of path queries P_i

begin

set t as the back twig pattern of path queries P_i obtained by definition 10;

set n as the number of labels in pattern t;

return n ;

end

그림 26. BackTwigPatternCnt 알고리즘

Fig. 26. Algorithm for BackTwigPatternCnt

Algorithm MaxCommonBranchLevel

Input: label paths of path queries P_i and P_{i+1}, $labelpath_i$, $labelpath_{i+1}$,

Common front twig pattern of path queries P_i and P_{i+1}, $CFTP_{i,i+1}$,

Back twig patterns of path queries P_i and P_{i+1}, BTP_i, BTP_{i+1}

Output: maximum common branch level of P_i, $blevel_i$

begin

/* i-th path의 branching node type이 3일 때, 최대 공통 분기 레벨 값을 반환*/

set $branchPostfix_i$ as the label path substring from the root to the branch label node matching with the branching node of the path query of P_i using $CFTP_{i,i+1}$, and BTP_i ;

set $branchPostfix_{i+1}$ as the label path substring from the root to branching label node matching with the branching node of the path query of P_{i+1} using $CFTP_{i,i+1}$ and BTP_{i+1} ;

```
set branchPostfixLen_i as the length of string branchPostfix_i ;
set branchPostfixLen_{i+1} as the length of string branchPostfix_{i+1} ;
blevel_i = 0 ;
while ( branchPostfixLen_i > 1 and branchPostfixLen_{i+1} > 1)
begin
  if ( branchPostfixLen_i = branchPostfixLen_{i+1}) then
  begin
    if (branchPostfix_1 = branchPostfix_2) then
    begin
      set blevel_i = the number of '/' in branchPostfixLen_i ;
      break;
    end
  end
  if ( branchPostfixLen_i < branchPostfixLen_{i+1}) then
    begin
      if ( branching node type of path query P_i, bnodetype_i = 2) then break;
      if ( branchPostfixLen_{i+1} = 0) then exit;
      set branchPostfix_{i+1} as the label path substring from the root to the next
          branching label node matching with the branching node of the path
          query of P_{i+1} using CFTP_{i,i+1} and BTP_{i+1} ;
      set branchPostfixLen_{i+1} as the length of string branchPostfix_{i+1} ;
    end
  else  /* (branchPostfixLen_i > branchPostfixLen_{i+1}) or
          ((branchPostfixLen_i = branchPostfixLen_{i+1}) and
          (branchPostfix_1 ≠ branchPostfix_2))                    */
      if ( branchPostfixLen_i = 0) then exit;
      set branchPostfix_i as the label path substring from the root to the next
          branching label node matching with the branching node of the path
```

```
            query of Pi using CFTPi,i+1 and BTPi ;
            set branchPostfixLeni as the length of string branchPostfixi ;
    end  -- of while
    return  blevelᵢ
end
```

그림 27. MaxCommonBranchLevel 알고리즘

Fig. 27. Algorithm for MaxCommonBranchLevel

$postfix_i$는 질의 P_i에 해당하는 레이블 경로들의 집합 $labelpath_i$에 대하여 루트 노드부터 최대 공통 분기 레벨 값 $blevel_i$까지의 부분 문자열 즉 포스트픽스 문자열을 나타내며 알고리즘은 그림 28과 같다.

```
Algorithm POSTFIX
Input: a label path of LabelPath table matching with Pi, labelpathi,
        a maximum_common_branch_level of path queries Pi and Pi+1 , blevelᵢ
Output: postfixi, maximum_common_branch postfix string of path queries Pi and Pi+1
begin
/* labelpathi에 대해 blevelᵢ 값 만큼 postfix 문자열 반환 */
      lp = '\' + labelpathi ;
      set lplen as the length of label path string, lp ;
      set lplevel as the number of '\' in label path string, lp ;
      targetlevel = lplevel - blevelᵢ ;
      find targetlevel-th '\' position in label path string, lp and set pos as it
      postfixi = substring(lp, pos + 1, lplen - pos) ;
      return postfixᵢ ;
end
```

그림 28. POSTFIX 알고리즘

Fig. 28. Algorithm for POSTFIX

그림 29는 VPCJoin_PP 알고리즘으로 유효한 레이블 경로 연결 집합에 대해 분할된 경로 질의에 속하는 데이타 노드들을 찾기 위한 작업을 진행한다.

Algorithm VPCJoin_PP

Input: BPQ P, PathJoin table, DataNode table

Output: set of data nodes *result_DNset* matching the tree query

begin

{ Assume the tree query T has r leaf labels $l_1, l_2, \ldots, l_r$

, that $lnodetype_i$ is a node type of i-th leaf label

, that l_1 is the leaf label of first path query

, and that nid_r is node id set through evaluating the path query P_r }

obtain $PathJoin_1$ by joining *PathJoin* table with the DataNode table through evaluating the path query P_1 according to leaf node type $lnodetype_1$;

for each of the remaining leaf nodes $l_i(i=2,\ldots,r)$ **do**

begin

obtain $PathJoin_i$, the result obtained by joining $PathJoin_{i-1}$ with the DataNode table through evaluating the path query P_i according to leaf node type $lnodetype_i$;

end

obtain *result_DNset* as the last projection of $PathJoin_r$ table,

$\Pi\ nid_r\ (PathJoin_r)$;

return *result_DNset*

end

그림 29. VPCJoin_PP 알고리즘

Fig. 29. Algorithm for VPCJoin_PP

그림 30은 표 3에서 레이블 경로간 연결이 완료되면 모든 BLPQ P_i에 해당하는

P_i와 $blevel_i$가 존재하게 되며 이는 향후 BLPQ 처리시 DataNode 테이블과 연결시 사용된다. 다만 결과 질의 노드를 포함하는 BLPQ P_r의 rqtype이 "A" (분기조건 비존재)일 때 분기 노드 유형이 1, 2이면 $blevel_k$(최종 결과 노드의 레벨인 $rlevel_k$에 해당)를 추가하며 분기 노드 유형 3은 존재하지 않는다.

Algorithm VPCJoin_QueryProcessing

Input: a branching path query P

Output: a set of node ids matching with the branching path query P

begin

Result = { };

decompose P into k linear path query patterns P_1, P_2, ..., P_k
by traversing the twig pattern P in the preorder;

obtain $LabelPathSet_1(pathid_1, blevel_1, postfix_1)$ or $LabelPathSet_1(pathid_1, blevel_1)$ by finding the set of label paths matched with the path query P_1 using the LabelPath and the parsing information of it;

$JoinedPath_1$ = $LabelPathSet_1$;

for (i=2; i <= k; i++) **do**

begin

obtain $LabelPathSet_i(pathid_i, blevel_i, postfix_i)$ or $LabelPathSet_i(pathid_i, blevel_i)$ by finding the set of label paths matched with the path query P_1 using the LabelPath and the parsing information of it;

if ($bnodetype_i$ = 3) **then**

$JoinedPath_i$ =

Postfix_matching($JoinedPath_{i-1}.labelpath_{i-1}$,$LabelPathSet_i.labelpath$,

MaximumCommonBranchLevel($JoinedPath_{i-1}.labelpath_{i-1}$,

$LabelPathSet_i.labelpath$, CommonFrontTwigPattern(P_i,P_{i+1}),

BackTwigPattern(P_i), BackTwigPattern(P_{i+1})));

else $JoinedPath_i$ = JOIN ($JoinedPath_{i-1}$, $LabelPathSet_i$);

end

```
    PathJoin1 = JOIN(JoinedPathk, DataNode) ;
    for (i=2; i <= k; i++) do
    begin
        PathJoini = JOIN (PathJoini-1, DataNode, nid = PREFIX(nidi-1, bleveli-1));
    end
    return Result
end
```

그림 30. VPCJoin_QueryProcessing 알고리즘

Fig. 30. Algorithm for VPCJoin_QueryProcessing

2. 질의 처리 알고리즘 적용 예

본 절에서는 3장에서 제안한 XML 인덱싱, 시스템 및 저장 구조에 기반하여 4장 1절에서 제안한 XML 조인 알고리즘의 처리과정을 설명한다.

예제 5. 그림 15와 같은 XML 문서에 대해 다음과 같은 BPQ /issue[/journal// keyword = "DB"]//author[/last = "Lee"]/first를 고려해보자.

제안하는 조인 알고리즘은 먼저 주어진 분기 경로 질의(BPQ)를 파싱하여 3개의 FLPQ /issue/journal//keyword, /issue//author/last, /issue//author/first 대신 다음과 같은 BLPQ P_1, P_2, P_3을 생성한다.

P_1: keyword\\journal\issue\, P_2: last\author\\issue\, P_3: first\author\\issue\

따라서 LPQ P_3은 결과 노드를 갖는 LPQ이다. 그리고 분기 질의 노드는 각각 issue와 author가 된다. 제안 알고리즘은 또한 파싱 과정에서 각 LPQ에 대해 분기 노드 유형 BNodeType 및 전반 가지 패턴 길이 FrontTwigPatternCnt, 후반 가지 패턴 길이 BackTwigPatternCnt 등의 정보를 추가로 얻는다.

- BNodeType(P_1): 1, BNodeType(P_2): 2, BNodeType(P_3): 2
- FrontTwigPatternCnt(P_1): 1, BackTwigPatternCnt(P_2): 1, BackTwigPatternCnt(P_3): 1

다음 단계는 레이블 경로들간의 결합을 통해 유효한 레이블 경로의 연결을 찾는 단계이다. LabelPath 테이블에서 LPQ P_1의 패턴과 일치하는 레이블 경로를 구하여 표 3에 제시된 열 목록으로 구성되는 레이블 경로들의 집합 LabelSet1(pathid1, blevel1, postfix1)테이블을 구한다. 여기서 P_1의 분기 질의 노드 유형 BNodeType(P_1)이 1이므로 issue에 해당하는 분기 레벨 blevel1은 FrontTwigPatternCnt(P_1) =1이고, postfix1은 P1에 해당하는 레이블 경로들 labelpath1에 대해 루트부터 분기레벨 bevel1까지의 서브-경로 문자열이다. 결과는 다음과 같다.

pathid1	blevel1	postfix1	*labelpath1*
13	1	issue#\	*keyword#\journal#\issue#*

동일한 방법 LabelSet2(pathid2, blevel2, postfix2)를 구한다. 여기서 P_2의 분기 노드 유형 BNodeType(P_2)이 2이므로 author에 해당하는 분기 레벨 blevel2는 plevel2 - BackTwigPattrernCnt(P_2)이며 4-1 = 3이다. 결과는 다음과 같다.

pathid2	blevel2	postfix2
12	3	author#\journal#\issue#\
18	3	author#\magazine#\issue#\
6	3	author#\paper#\issue#\

labelpath2	*plevel2*
last#\author#\journal#\issue#	*4*
last#\author#\magazine#\issue#	*4*
last#\author#\paper#\issue#	*4*

LabelSet1(pathid1, blevel1, postfix1)과 LabelSet2(pathid2, blevel2, postfix2)를 postfix1 = POSTFIX(postfix2, blevel1)의 조건으로 조인하여, 유효한 레이블 경로 연결 집합인 JoinedPath2(pathid1, blevel1, pathid2, blevel2, postfix2)를 구한다.

pathid1	blevel1	pathid2	blevel2	postfix2	*postfix1*
13	1	12	3	author#\journal#\issue#\	*issue#*
13	1	18	3	author#\magazine#\issue#\	*issue#*
13	1	6	3	author#\paper#\issue#\	*issue#*

P_3의 분기 노드 유형 BNodeType(P_3)이 2이고, 결과 레이블 first에 해당하는 분기 레벨 blevel3는 plevel3 - BackTwigPattrernCnt(P_3)이며 4-1 = 3이다. 결과 LPQ이므로 LabelSet3(pathid3, blevel3, postfix3)을 구한 결과는 다음과 같다.

pathid3	blevel3	postfix3
11	3	author#\journal#\issue#\
17	3	author#\magazine#\issue#\
5	3	author#\paper#\issue#\

labelpath3	*plevel3*
first#\author#\journal#\issue#	*4*
first#\author#\magazine#\issue#	*4*
first#\author#\paper#\issue#	*4*

JoinedPath2(pathid1, blevel1, pathid2, blevel2, postfix2)와 LabelSet3(pathid3, blevel3, postfix3)을 postfix2 = POSTFIX(postfix3, blevel2)의 조건으로 조인하여, 유효한 레이블 경로 연결 집합인 JoinedPath3를 구한다. P_3의 결과 LPQ 타입이 "A" 형태이므로 열 목록은 (pathid1, blevel1, pathid2, blevel2, pathid3)이고 결과는 다음과 같다.

pathid1	blevel1	pathid2	blevel2	pathid3	*postfix2*	*postfix3*
13	**1**	**12**	**3**	**11**	author#\journal#\issue#\	*first*#\author#\journal#\issue#\
13	*1*	*12*	*3*	*17*	*author#\journal#\issue#*	*first#\author#\magazine#\issue#*
13	*1*	*12*	*3*	*5*	*author#\journal#\issue#*	*first#\author#\paper#\issue#*
13	*1*	*18*	*3*	*11*	*author#\magazine#\issue#*	*first#\author#\journal#\issue#*
13	**1**	**18**	**3**	**17**	author#\magazine#\issue#\	*first*#\author#\magazine#\issue#\
13	*1*	*18*	*3*	*5*	*author#\magazine#\issue#*	*first#\author#\paper#\issue#*
13	*1*	*6*	*3*	*11*	*author#\paper#\issue#*	*first#\author#\journal#\issue#*
13	*1*	*6*	*3*	*17*	*author#\paper#\issue#*	*first#\author#\magazine#\issue#*
13	**1**	**6**	**3**	**5**	author#\paper#\issue#\	*first*#\author#\paper#\issue#\

총 9개의 연결 중 3개만 선택된다. 이렇게 구해진 유효한 레이블 경로들의 연결 집합 PathJoin(= JoindPath3)을 분할된 BLPQ P_1과 P_1의 분기 조건 표현식을 이용, pathid1 = pathid and ntype = 3 and cmt = "DB" and value = "DB" 조건으로 조인을 수행하여 PathJoin1(nid1, bevel1, pathid2, blevel2, pathid3)을 구한다.

nid1	blevel1	pathid2	blevel2	pathid3
1.14.22.23	1	12	3	11
1.2.12.13	1	6	3	5

pathid1	*labelpath1의 단말노드 값*
13	*"DB"*
13	*"DB"*

구해진 PathJoin1을 BLPQ P_2와 P_2의 분기 조건 표현식을 이용, ntype = 3 and pathid2 = pathid and nid LIKE PREFIX(nid1, blevel1) + '%' and cmt = "Lee" and value = "Lee" 조건으로 조인을 수행하여 PathJoin2(nid2, bevel2, pathid3)을 구한다. 여기서

cmt 열은 텍스트 노드 값에 대한 6 바이트 요약 값을 의미한다.

nid2	blevel2	pathid3
1.14.17.20.21	3	11

nid1	*blevel1*	*pathid2*	*pathid2에 해당하는 labelpath*
1.14.22.33	*1*	*12*	*last#\author#\journal#\issue#*

최종 단계로 구해진 PathJoin2를 BLPQ P_3을 이용, ntype = 1 and pathid3 = pathid and nid LIKE PREFIX(nid2, blevel2) + '%' 조건으로 조인을 수행하여 PathJoin3(nid3) 결과 집합을 구한다.

nid3
1.14.17.18.19

nid2	*blevel2*	*pathid3*	*pathid3에 해당하는 labelpath*
1.14.17.20.21	*3*	*11*	*first#\author#\paper#\issue#*

그림 31은 예제 5에 대해 제안된 방식에 의해 변환된 SQL 문이다. 편의상 문서 식별자 번호 docid는 생략하였으며 PREFIX(nid, blevel) 함수는 프리픽스 레이블 기반 노드 번호 nid에 대해 blevel 만큼의 프릭픽스 문자열을 반환하는 함수이다.

```
WITH JoinedPath1(pathid1, blevel1, postfix1)
AS
   ( SELECT p1.pathid,  1 as blevel1, POSTFIX(labelpath, blevel1) as postfix1
     FROM    LabelPath p1,
     WHERE p1.labelpath like "keyword#%\issue#\" )
,    JoinedPath2(pathid1, blevel1, pathid2, blevel2, postfix2)
AS
   ( SELECT pathid1, blevel1, pathid2, blevel2, postfix2
     FROM    JoinedPath1, (SELECT pathid as pathid2, plevel - 1 as blevel2,
                                  POSTFIX(labelpath, blevel2) as postfix2
                           FROM LabelPath p2
                           WHERE p2.labelpath like "last#\author#%\issue#\")
```

```
      WHERE postfix1 = POSTFIX(postfix2, blevel1)    )
,     JoinedPath3(pathid1, blevel1, pathid2, blevel2, pathid3)
AS
   ( SELECT pathid1, blevel1, pathid2, blevel2, pathid3
     FROM   JoinedPath2, (SELECT pathid as pathid3, plevel - 1 as blevel3,
                                 POSTFIX(labelpath, blevel3) as postfix3
                          FROM   LabelPath p3
                          WHERE  p3.labelpath like "first#\author#%\issue#\")
     WHERE postfix2 = POSTFIX(postfix3, blevel2) )
,     PathJoin1(nid1, blevel1, pathid2, blevel2, pathid3)
AS
   ( SELECT nid as nid1, blevel1, pathid2, blevel2, pathid3
     FROM   JoinedPath3, DataNode
     WHERE ntype = 3 and pathid = pathid1 and cmt = "DB" and value = "DB")
,     PathJoin2(nid2, blevel2, pathid3)
AS
   ( SELECT nid as nid2, blevel2, pathid3
     FROM   PathJoin1, DataNode
     WHERE ntype = 3 and pathid = pathid2 and nid LIKE (nid1, blevel1)
           + "%"  and cmt = "Lee" and value = "Lee")
     SELECT nid
     FROM   PathJoin2, DataNode
     WHERE ntype = 1 and pathid = pathid3 and nid LIKE PREFIX(nid2, blevel2)
           + "%" ;
```

그림 31. 예제 질의를 위한 제안 방식의 SQL 문

Fig. 31. SQL Statement in the proposed method for example query

변환되는 SQL 문은 사용 RDBMS에 따라 구문이 달라질 수 있으며 변환시 주안

점은 데이타 노드들간의 비교에 앞서 유효한 레이블 연결 식별을 위해 즉 LabelPath 테이블간의 조인을 먼저 수행하도록 하는 것이다. 일부 시스템에서는 임시 테이블 혹은 함수를 사용할 수도 있다.

V. 성능 평가

1. 실험 환경

본 논문에서는 제안 방식의 성능을 평가하기 위한 실험 환경을 다음과 같이 두 가지로 나누어서 수행한다. 먼저, 인덱싱 성능 평가를 위한 실험 환경에 대해 설명하고 다음으로 조인 알고리즘 성능 분석을 위한 환경을 설명한다.

1) 인덱스 실험 환경

먼저 인덱싱의 성능 실험을 위해 Pentium3 930MHz CPU에 256 메모리를 가진 PC와 MS SQL Server 2000을 사용하여 실험 환경을 구축하였으며, 제안한 방법의 실험을 위해 필요한 모든 것들은 Java와 SAX2.0을 이용하여 구현하였다. 실험에 대한 성능 평가 기준으로 질의 처리 비용을 위한 소요시간(Elapsed time)을 사용한다. 두 번째로 상이한 구조의 문서 증가에 따르는 확장성을 평가하기 위하여 레이블 경로의 개수 변화에 따른 질의 소요시간을 측정한다.

레이블 경로 개수의 증가에 따르는 확장성 실험을 위해서 충분한 정도의 레이블 경로가 필요하다. 웹에서 무작위로 수집된 XML 문서들을 분석한 결과 한 개의 XML 문서 당 평균 8개 이하의 상이한 레이블 경로를 가지고 있었으며 따라서 실험 환경 충족을 위한 상이한 구조의 XML 문서 개수는 대략 수십만 개에서 수백만 개에 이른다. 본 실험에서는 일부 수집된 문서와 자체적으로 생성한 XML 문서들을 가지고 실험을 수행하였다. 제안 인덱싱 기법의 성능 평가를 위해 XRel[5], EPIS[2]를 비교 대상으로 선정하였다. XParent[6,7]는 XRel과 동일한 방법으로 선형 경로 질의를 처리하기 때문에 비교 대상에서 제외하였으며 XIR-Branching[10]은 off-the-shelf RDMBS의 기능이 아닌 정보검색(IR)의 역 인덱스를 사용한 관계로 실험대상에서 제외하였다. 비교 대상방식의 레이블 경로를 저장하는 테이블들에 대해 완전 클러스터 인덱스(Full Clustered Index)를 제공하였으며 사용된 질의의 SQL 변환은 수작업을 통해 수행하였다.

표 4는 실험에서 사용된 질의이며 질의를 구성하는 경로의 길이, 조상-자손 관계

'//'의 개수를 고려하여 질의를 선정하였다.

표 4. 실험 질의들

Table 4. The experimental queries

질의 유형	XPath 질의
Q1	/PLAY/ACT
Q2	/PLAY/ACT/SCENE/SPEECH/LINE/STAGEDIR
Q3	/PLAY//SCENE/STAGEDIR
Q4	//ACT
Q5	//SCENE//LINE
Q6	//ACT//SPEECH//SPEAKER
Q7	//ACT/SCENE//SPEECH/LINE/STAGEDIR

2) 조인 알고리즘 실험 환경

제안 조인 알고리즘의 성능 실험을 위해 Pentium4 1.6 GHz CPU에 256 메모리를 가진 PC와 MS SQL Server 2000을 사용하여 실험 환경을 구축하였다. 실험을 위한 프로그램 구현은 Java와 SAX2.0, 그리고 Transact-SQL을 이용하여 구현하였다. 조인 알고리즘에 실험 성능 평가 기준으로 조인을 처리하기 위한 소요시간(Elapsed time)을 사용한다. 실험을 위한 XML 문서 집합은 세익스피어의 희곡들의 모음을 XML로 기술한 데이타로서 웹에 공개된 Shakespeare2.0[45]을 사용하였다. 실험은 우선 전체 문서를 로딩하고 주어진 질의에 대해 소요시간을 측정한다. 두 번째로 전체 문서를 4번 데이타베이스에 적재하여 문서 개수의 증가에 따르는 소요시간을 평가한다. 제안 조인 알고리즘의 성능 평가를 위해 BEL 조인 기법을 사용하는 XRel을 선정하였다. 그리고 최근에 연구된 XIR-Branching과의 비교를 수행하기 위하여 먼저 동일한 인덱싱 기법을 사용하도록 조정하였다. XRel과 XIR-Branching은 원 논문에서 사용한 것과 같은 데이타베이스 스키마와 인덱스들을 사용하였다. 그리고 사용된 XML 질의의 SQL 변환은 수작업을 통해 수행하였다. 표 5, 표 6는 실험에 사용된 XML 문서에 대한 상세 내용이다.

표 5. 사용된 XML 데이타 집합에 대한 상세 정보들

Table 5. Information of XML dataset

항목	값
문서의 총 크기(MB)	7.53M
문서 개수	37
문서 평균 크기(KB)	208.4
문서내 총 노드의 수	327131
엘리먼트 노드의 수	179689
텍스트 노드의 수	147442
경로의 개수	57
문서의 최대레벨	6
문서당 평균 엘리먼트 노드 수	4856
문서당 평균 텍스트 노드의 수	3989
텍스트 최대 길이	1015

표 6. 실험 문서 집합의 노드 개수와 분포 레벨

Table 6. The number and distributed level of node in experimental documents

엘리먼트	총 노드 수	문서당 평균개수	분포레벨
PLAY	37	1	1
TITLE	1031	27	2, 3, 4
ACT	185	5	2
SUBHEAD	18	0	4, 5
EPILOGUE	6	0	3
P	148	4	3
PGROUP	90	2	3
GRPDESCR	90	2	4
STAGEDIR	6258	169	3, 4, 5, 6
SCNDESCR	37	1	2
PROLOGUE	14	0	2,3
SPEAKER	31081	840	4, 5
SCENE	750	20	3
PERSONAE	37	1	2
FM	37	1	2
LINE	107833	2914	4, 5
INDUCT	2	0	2
PLAYSUBT	37	1	2
PERSONA	969	26	3, 4
SPEECH	31028	838	3, 4
SUBTITLE	1	0	4

2. 실험 결과 및 분석

1) 인덱스 실험

그림 32는 실험 질의들의 실행 결과 값을 보여주는 그래프이다. XRel은 EPIS보다 전체 매치 질의 Q1, Q2와 '//'로 시작하지 않는 부분 매치질의 Q3에 대해서 우수한 성능을 보였다. 반면 '//'로 시작하는 부분 매치 질의 Q4, Q5, Q6, Q7에 대해서는 항상 전체 경로 테이블을 읽기 때문에 EPIS보다 열악한 결과를 보여주고 있다. EPIS는 경로 식별자를 찾기 위해 먼저 레이블별로 인덱스 탐색을 통해 후보 경로 식별자들을 찾는다. 이후에 최종 경로 식별자를 찾기 위해 후보 경로 식별자간 조인을 수행한다. 따라서 EPIS는 XRel에 비해 효과적으로 부분 매치 질의 처리를 지원한다. 제안방식은 전체 매치 질의에 대해서는 XRel과 동일한 수준의 성능을 보이고 있으며 부분 매치 질의에 대해서는 EPIS보다도 더 우수한 성능을 보이고 있다. 이런 결과는 제안방식이 전체 매치 질의, 부분 매치 질의 즉 질의 유형과 관계없이 경로를 찾기 위하여 추가적인 조인연산 없이 항상 경로 테이블 탐색 대신 인덱스를 탐색하기 때문이다.

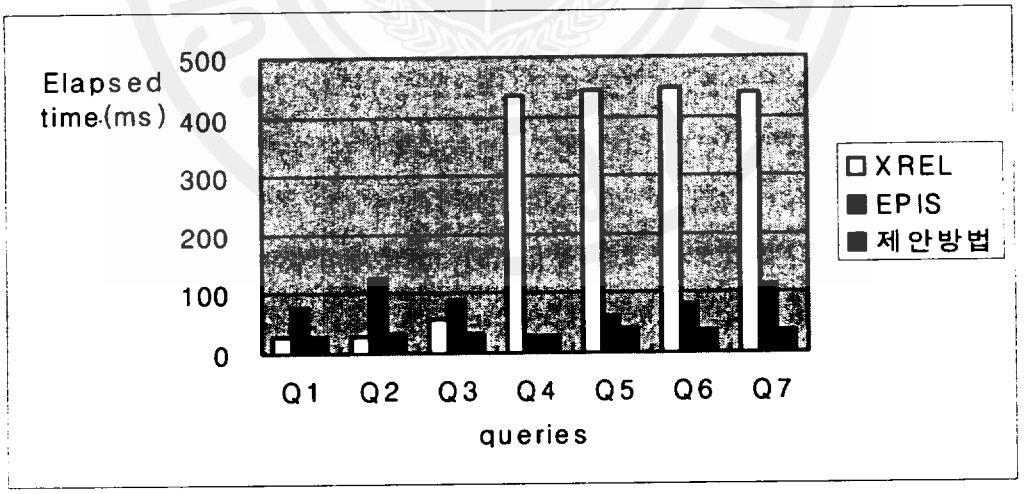


그림 32. 실험 질의 비용

Fig. 32. Costs of the experimental queries

한편, 상이한 구조를 갖는 문서 개수의 증가함에 따라 레이블 경로 개수도 증가된다. 그림 33, 그림 34는 이러한 상황에서도 제안방식이 우수한 성능을 갖는다는 것을 보이기 위한 확장성 검증 실험이다. 그림 33은 긴 경로식을 갖는 전체 매치 질의 Q2에 대한 질의 처리 비용 그래프이고, 그림 34는 '//'로 시작하는 부분 매치 질의 Q6에 대한 그래프이다.

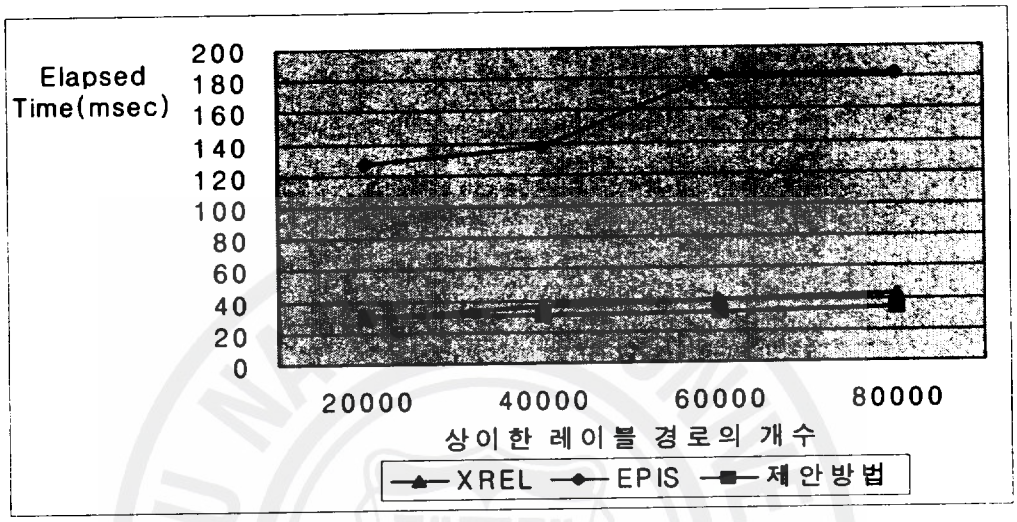


그림 33. 레이블 경로 개수 증가에 따른 Q2를 위한 질의 비용

Fig. 33. Query costs for Q2 by increasing number of label paths

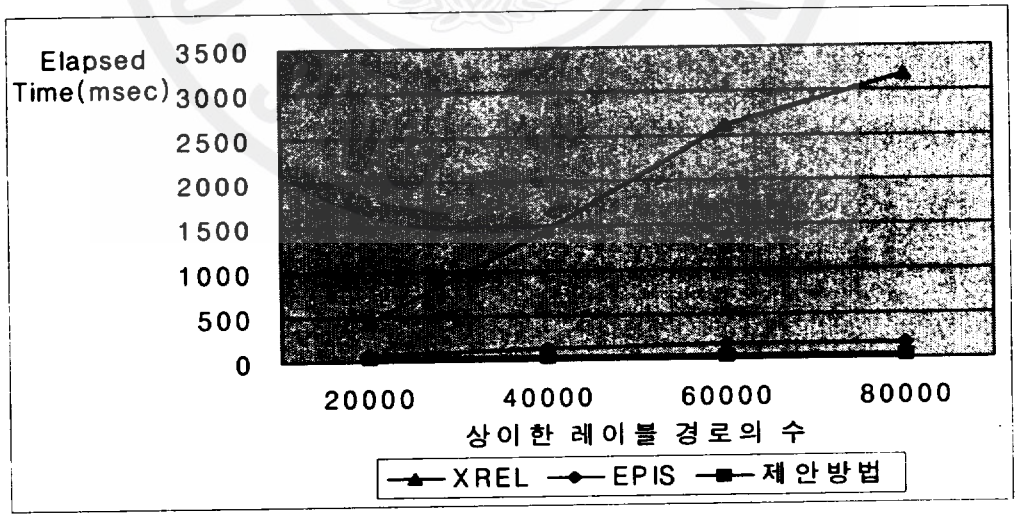


그림 34. 레이블 경로 개수 증가에 따른 Q6을 위한 질의 비용

Fig. 34. Query costs for Q6 by increasing number of label paths

제안방식은 레이블 경로의 개수가 커지는 상황에서도 질의 유형에 관계없이 질의 비용이 거의 상수적(nearly constant)으로 증가한다. 제안방식은 레이블 경로를 저장하는 경로 테이블 크기가 증가함에 따라 그 성능의 차이는 더욱 더 커짐을 보인다.

2) 조인 알고리즘 실험

조인 알고리즘 성능 평가를 위한 실험은 크게 선형 질의 결합 유형 1에 해당하는 질의와 결합 유형 3에 해당하는 질의에 대해 나누어 실험을 수행한다. 각 실험은 우선 전체 문서 1 셋(37개의 문서)을 데이타베이스에 적재하고 검색 대상의 문서 수를 증가시키면서 질의를 수행한다. 다음으로 전체문서를 4번 적재(총 148개의 문서)에 대해 검색 대상의 문서 개수를 증가시키면서 질의를 수행한다.

(1) 질의 결합 유형 3

BPQ1 : /PLAY[/ACT//TITLE]//LINE

- LPQ1: /PLAY/ACT//TITLE, LPQ2: /PLAY//LINE
- common twig pattern: /PLAY, back twig pattern: /ACT//TITLE, //LINE
- LPQ1 경로 번호 = {14, 16, 25, 38}, 경로 수: 4개
  LPQ2 경로 번호= {20, 28, 36, 42, 48, 55}, 경로 수: 6개
- 소요시간

문서갯수(2^N)	XRel	XRel+제안방법	XIR-Branching	제안방법
0	1513	1050	11696	970
1	3816	3273	7443	3486
2	3803	3323	13456	4756
3	9923	7903	16913	8060
4	17816	14190	23893	15293
5	36683	29623	32476	28830

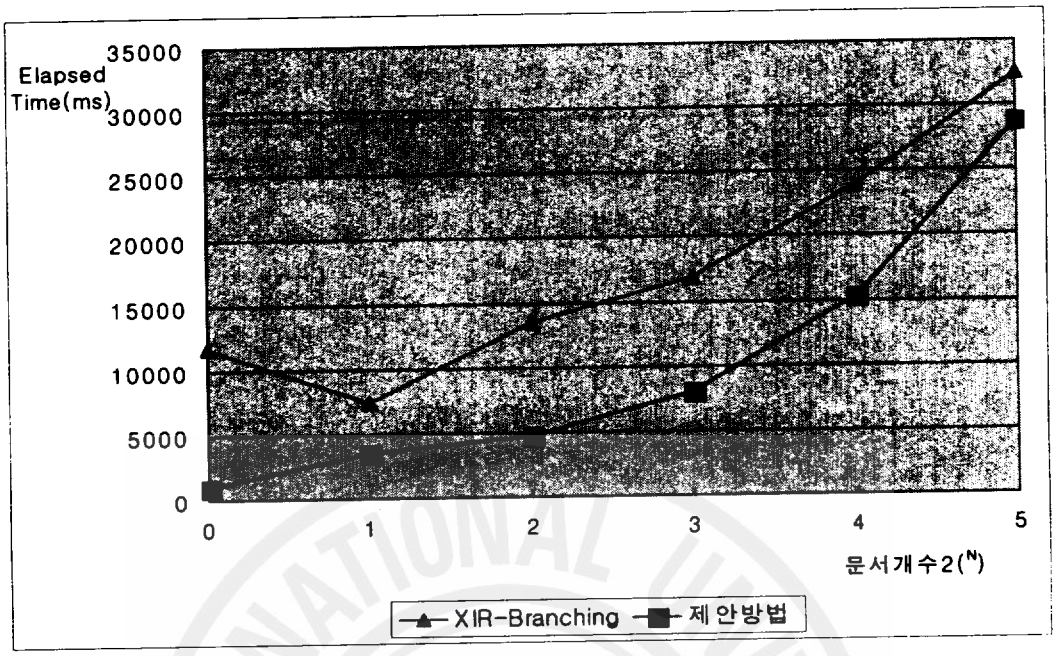


그림 35. XIR-Branching과 제안방법의 BPQ1 질의 처리 소요시간

Fig. 35. Elapsed time of XIR-Branching and proposed method for the query BPQ1

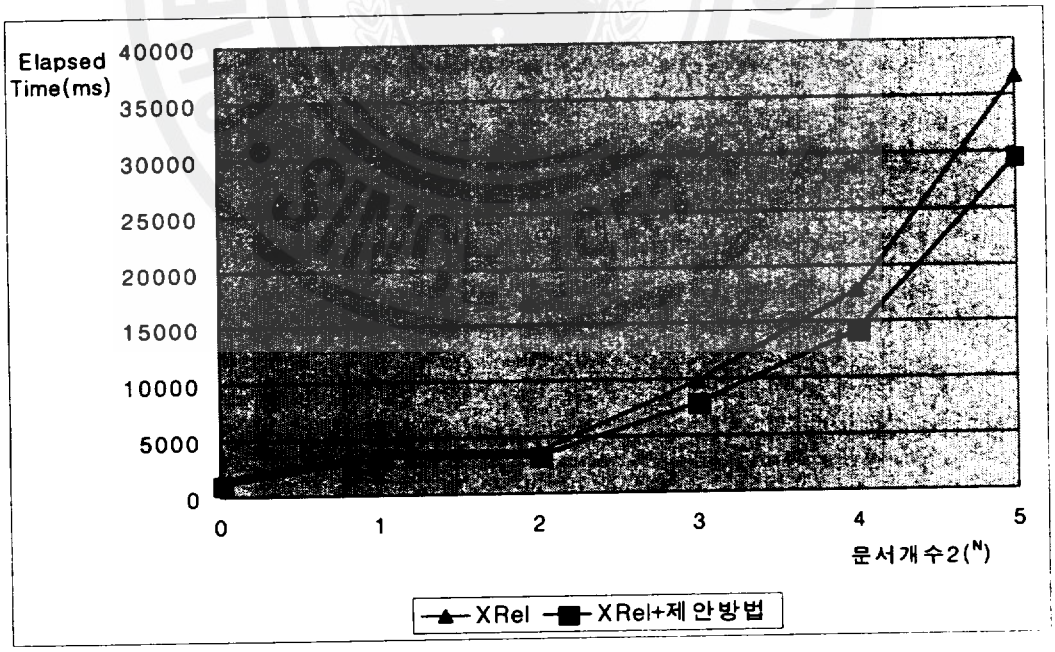


그림 36. XRel과 XRel+제안방법의 BPQ1 질의 처리 소요시간

Fig. 36. Elapsed time of XRel and XRel+proposed method for the query BPQ1

그림 35, 그림 36은 XML 전체 문서 1셋이 적재된 상태에서 검색 대상이 되는 문서 수를 증가시키면서 수행한 질의 결과를 보여 주고 있다. 이 질의인 경우에는 감축되는 레이블 경로 결합이 존재하지 않는다. 그러나 그림 35와 같이 제안한 방법이 XIR-Branching에 비해 좋은 결과를 보이는 것은 첫 번째 유효한 연결 식별을 위한 비교 횟수의 차이이다. 제안한 방법은 유효 레이블 경로 연결을 식별하기 위해 24번의 비교만으로 결정되나 XIR-Branching은 전체 문서일 경우 951개 (LPQ1에 소속된 노드 수) X 107833개(LPQ2에 소속된 노드 수) = 102549183번의 비교가 발생한다. 그리고 레이블 경로 연결 개수가 감축되지 않는 상황에서 XRel과 XRel+제안방법의 성능 차이는 차이가 없을 것으로 판단되었으나 실제로는 그림 36과 같은 결과를 보였다. 질의 실행 플랜을 분석한 결과 XRel은 기본적으로 세타 조인을 사용하며 RDBMS는 세타 조인에 대해 보편적으로 우수한 실행 플랜을 생성하지 못한다는 점이다. 반면 제안한 방법의 기능을 XRel과 결합한 XRel+제안방법은 경로 테이블간의 조인을 우선적으로 처리하여 좀 더 나은 실행 플랜을 생성하였다.

(2) 질의 결합 유형 4

BPQ2: //SPEECH[/SPEAKER]//LINE

- LPQ1: //SPEECH/SPEAKER, LPQ2: //SPEECH//LINE
- common twig pattern: //SPEECH, back twig pattern: /SPEAKER, //LINE
- LPQ1 경로 번호 = {19, 27, 35, 41, 47, 54}, 경로 수: 6개, 총 노드 수: 31028
  LPQ2 경로 번호= {20, 28, 36, 42, 48, 55}, 경로 수: 6개, 총 노드 수: 107833
- 경로 연결 수 : 36개
- 유효 연결 수: 6개 {(19:20, (27:28), (35:36), (41:42), (47:48), (54:55)}
- 소요시간 : 1set 문서

문서갯수(2^N)	XRel	XRel+제안방법	XIR-Branching	제안방법
0	410	3266	503646	7760
1	16683	10583	11386	16443
2	20236	10936	141893	36603
3	42820	20560	171450	76757
4	89046	51593	326626	150456
5	217543	107423	676343	150636

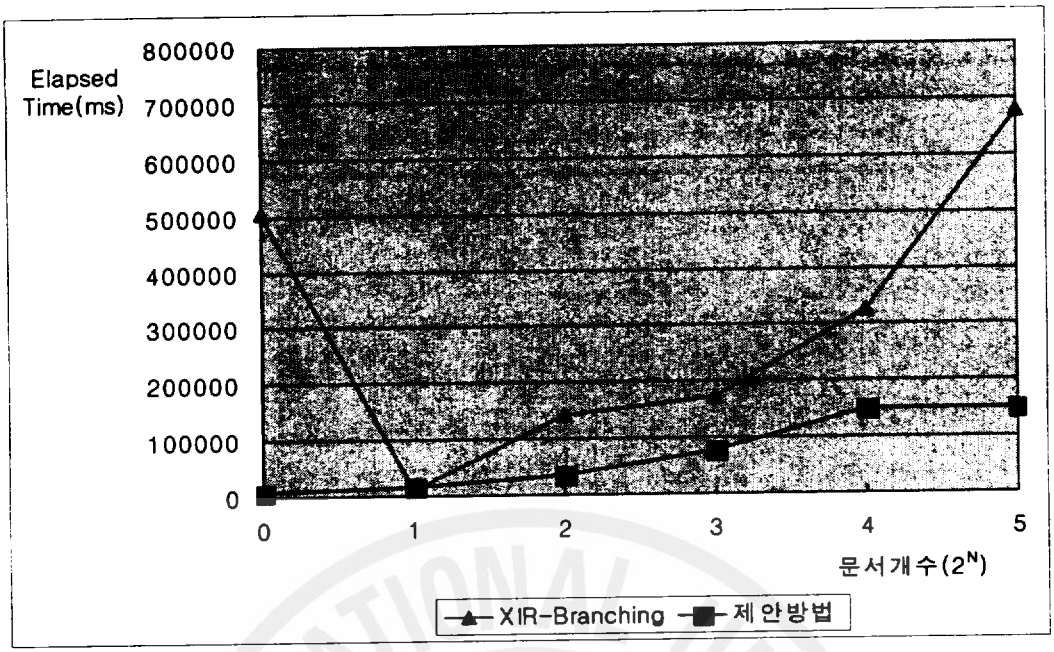


그림 37. XIR-Branching과 제안방법의 BPQ2 질의 처리 소요시간

Fig. 37. Elapsed time of XIR-Branching and proposed method for the query BPQ2

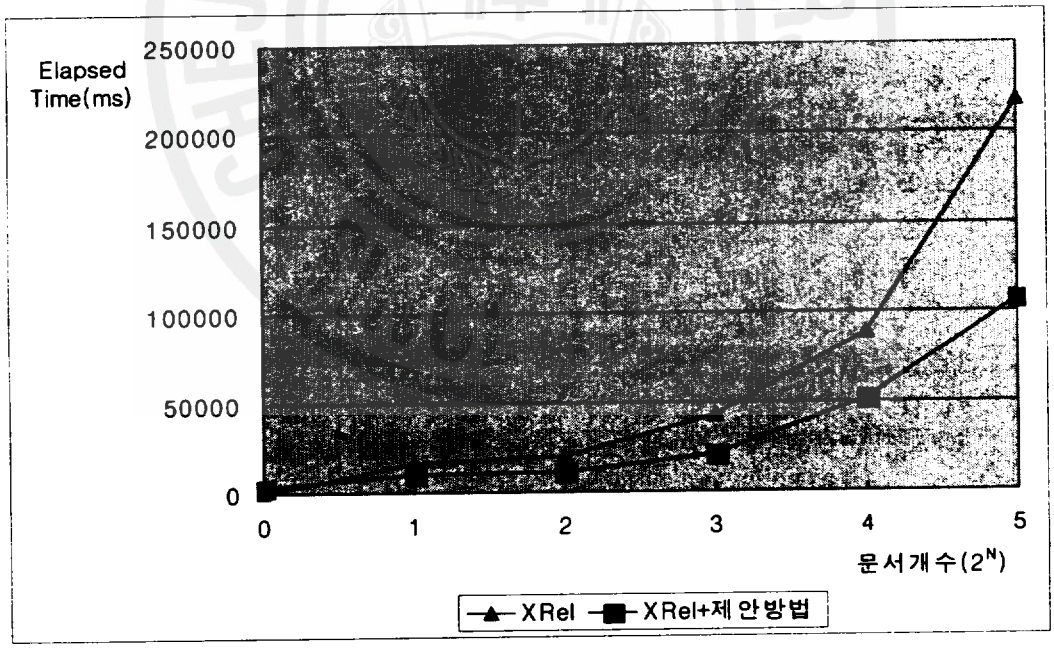


그림 38. XRel과 XRel+제안방법의 BPQ2 질의 처리 소요시간

Fig. 38. Elapsed time of XRel and XRel+proposed method for the query BPQ2

BPQ2는 질의 결합 유형이 4이고 경로 연결 개수가 36인데 유효 연결 결합 수는 6이다. 따라서 감축된 레이블 경로에 소속된 데이타 노드들 간의 비교가 회피된다. 이런 결과로 그림 37에서 보여주는 바와 같이 제안한 방법은 XIR-Branching에 대해서 그림 35에서 보여주는 성능 차이보다 더 큰 차이를 보여 주고 있다. 그림 38에 XRel+제안방법 또한 이러한 유효하지 않은 연결에 속한 데이타 경로들의 비교 비용이 절감되어 그림 36보다 좀 더 큰 차이를 보여 주고 있다.

문서갯수(2^N)	XRel	XRel+제안방법	XIR-Branching	제안방법
0	12540	4736	717733	11166
1	31143	30483	16023	24616
2	65376	28670	28616	28153
3	131360	50243	80141	70243
4	213696	67736	190220	143043
5	385833	133270	666130	230510
6	344193	212463	2048623	278256
7	662283	340440		308693

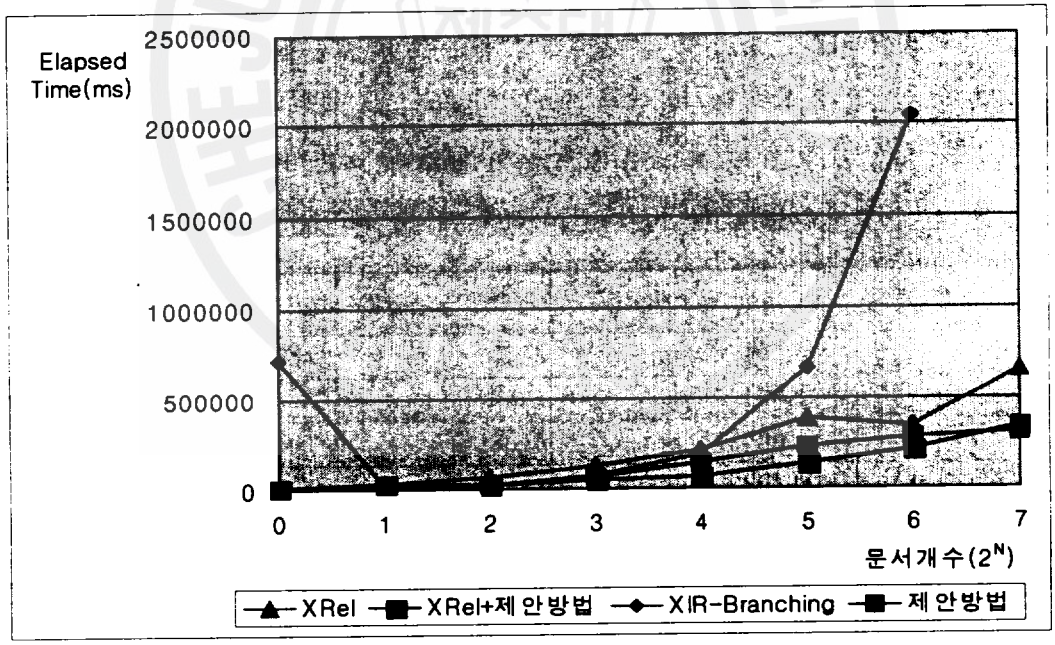


그림 39. XRel, XRel+제안방법, XIR-Branching, 제안방법의 BPQ2 질의 처리 소요시간

Fig. 39. Elapsed time of XRel, XRel+proposed method, XIR-Branching, and proposed method for the query BPQ2

그림 39와 그림 40은 XML 문서 전체 37개의 문서를 4번 적재하여 검색 대상의 문서 개수를 변화시켜 문서 증가에 따른 조인 알고리즘의 영향을 평가하기 위한 실험이다. 검색 대상의 문서 수가 작을 경우에는 XRel+제안방법(구간 기반 레이블링)이 좋은 성능을 보였으나 문서 개수가 증가함에 따라 제안방법(프리픽스 레이블링)의 성능이 좋아진다는 것을 알 수 있다.

VI. 결 론

웹 환경에서 상이한 문서 구조를 가지는 XML 형식의 문서들이 빠르게 증가하고 있고 따라서 이러한 문서들로부터 사용자가 원하는 정보들을 추출하는 질의 처리가 매우 중요해지고 있다. 본 논문에서는 이러한 상이한 구조의 문서가 존재하는 상황에서 off-the-shelf 관계 데이타베이스를 기반으로 하여 부분 매치 질의를 효과적으로 처리할 수 있는 방법을 제안하였다.

제안한 질의 처리 방법은 off-the-shelf 관계 데이타베이스 위에 XML 데이타베이스를 구축하는 접근법을 사용하였으며 다음과 같은 시스템 환경을 가지도록 설계하였다.

- 저장된 XML 문서에 제약이 없음: 유효하거나 well-formed 문서 저장
- XML 문서의 저장과 검색은 현재의 데이타베이스의 기능만을 이용 : 데이타 모델, 질의 언어, 인덱스에 대한 확장은 없음
- 상이한 구조를 가지는 대량의 XML 문서 응용에의 적응성
- 부분 매치 질의의 효율적 처리

본 논문에서는 제안한 시스템 환경을 기반으로 부분 매치 질의를 효율적으로 처리하기 위하여 새로운 XML 인덱스 저장 구조와 조인 알고리즘을 제안하였다.

제안한 XML 인덱스 저장구조는 XML 문서에서 발생하는 레이블 경로들을 경로 테이블에 저장하는 기존의 방법과는 달리 역방향 레이블 경로를 구해서 저장하는 방식을 사용한다. 기존의 인덱싱 방법들은 조상-자손 관계성 '//'이 선두에 존재하는 부분 매치 질의 처리시 생성된 인덱스들을 사용하지 못하고 테이블 전체를 탐색하는 한계를 가지고 있다. 반면 역방향 레이블 경로 저장 방식은 그러한 상황에서도 전체 테이블 탐색 대신 인덱스 탐색을 수행함으로써 대규모의 상이한 구조의 문서가 존재하는 상황에서 효과적으로 부분 매치 질의 처리를 지원한다. 기존 인덱싱 방법들과의 실험을 통해 제안된 인덱싱 방법의 성능을 비교 하였으며 제안 방법이 전체 매치 질의는 물론 부분 매치 질의 처리에 적합함을 보였다.

또한 선형 경로 질의 처리 결과 간의 효율적인 조인 처리를 위해 기존 연구에서

수행하고 있는 조인 알고리즘과는 다른 새로운 조인 알고리즘을 설계하였다. 기존 조인 알고리즘의 분기 경로 질의 처리 과정은 다음과 같다.

① 분기 경로 질의를 선형 경로 질의로 분할한다.

② 분할된 각각의 선형 경로 질의에 대해 해당하는 레이블 경로들을 구하고 이에 소속된 인스턴스들의 집합을 구한다.

③ 결과 질의 노드를 포함하는 결과 선형 경로 질의에 소속된 인스턴스 집합을 나머지 선형 경로 질의에 소속된 인스턴스 집합들과의 비교(포함 관계 조인, 프리픽스 매치 조인)를 통해 결과 선형 경로 질의에 소속된 인스턴스 집합내 튜플들을 감축하면서 최종 질의 결과를 구한다.

제안한 조인 알고리즘은 분기 경로 질의의 최종 결과를 구하기 위해 선형 질의 결과에 소속된 인스턴스 들끼리의 비교를 통해 결과 튜플을 구하는 과정에서 최종 질의 결과에 포함되지 않는 인스턴스들 간의 비교를 회피하는 방식을 채택함으로써 조인에 따르는 전체 비교 연산 횟수를 줄인다. 제안하는 알고리즘은 다음과 같은 단계로 질의를 처리한다.

① 먼저, 분기 경로 질의를 선형 경로 질의로 분할한다.

② 분할된 선형 경로 질의에 소속된 레이블 경로 집합들을 구한다.

③ 각각의 선형 경로 질의에 해당하는 레이블 경로 집합들 간에 최대 공통 분기 레벨 계산을 통해 각각의 레이블 경로에 대해 결합 가능한 레이블 경로들(일명 유효 레이블 경로 연결)을 찾는다.

④ 각 선형 경로 질의에 소속된 인스턴스 집합간의 튜플 비교시 식별된 유효 레이블 경로 연결에 포함되는 인스턴스 들끼리만 비교를 수행한다.

제안 알고리즘의 구현을 위해 본 논문에서는 먼저, 선형 질의 결과를 결합하는 단계에서 최종 결과에 포함되지 않는 즉 유효하지 않은 데이타 경로들간의 비교가 발생하는 경우를 분석하여 유형별로 제시하였다. 또한 제시된 유형별로 유효한 레이블 경로들 간의 결합을 식별하는 방법 및 알고리즘을 제시하였다. 유효 결합 식별을 위해 본 논문에서는 레이블 경로의 레벨 정보와 경로 질의 파싱 단계에서 추출한 정보를 가지고 결정하는 방식을 채택함으로써 결정 비용을 최소화 하였다.

실험을 위해 제안 알고리즘을 구간-기반 레이블링 방식을 사용하는 XRel 시스템과 프리픽스 레이블링 방식을 사용하는 XIR-Branching 시스템 환경에 구현하여 비

교 분석을 수행하였다.

제안한 조인 알고리즘은 XRel에서 사용된 BEL 값을 이용한 세타 조인에 비해 조인횟수와 조인시 수행하는 데이타 비교량을 감소시켜 질의 처리 시간을 개선시켰다. 또한 XIR-Branching과는 동일한 조인 횟수를 가지나 조인에 앞서 유효 레이블 경로 결합을 식별함으로써 조인 수행 과정에서 발생하는 튜플 간의 비교량을 감소시켰다. 그리고 제안 알고리즘은 기존의 조인 알고리즘과 결합하여 유효한 레이블 경로들 간의 식별을 수행하도록 함으로써 기존 조인 알고리즘의 성능을 향상시킬 수 있음을 실험을 통해 보였다.

본 연구와 관련하여 앞으로 진행할 연구 과제는 다음과 같다.

첫 째, 질의 처리 비용은 비교 횟수외에 다양한 요소가 고려되며 특히 질의 최적화기에 의한 조인 순서는 중요하다. 제안 알고리즘의 성능 향상을 위해 질의 최적화기에 의한 실행계획과 부합하는 지능적인 조인 알고리즘에 관한 연구를 수행할 예정이다.

두 번째, 전체 XML 문서 검색시 빠른 추출을 위해 XML 문서를 외부에 저장하는 방식을 사용하고 있으나 향후 자체에 포함하여 통합 관리하는 출판에 관한 연구를 수행할 것이다.

참고문헌

[1] eXtensible Markup Language(XML), http://www.w3.org/XML/.

[2] 민경섭, 김형주, "상이한 구조의 XML 문서들에서 경로 질의 처리를 위한 RDBMS 기반 역 인덱스 기법", 정보과학회논문지:데이타베이스, 제30권, 제4호, pp.420-428, 2003.

[3] K. Runapongsa, J. M. Patel, "Storing and Querying XML Data in Object -Relational DBMSs," In Proc. of EDBT Workshop, pp.266-285, 2002.

[4] Q. Li and B. Moon, "Indexing and Querying XML Data for Regular Path Expressions," In Proc. of VLDB, pp.361-370, 2001.

[5] M. Yoshikawa et al., "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," ACM Trans. on Internet Technology, Vol.1, No.1, pp.110-141, 2001.

[6] H. Jiang et al., "Path Materialization Revisited: An Efficient Storage Model for XML Data," In Proc. of ADC, pp.85-94, 2002.

[7] H. Jiang et al., "XParent: An Efficient RDBMS-Based XML Database System", In Proc. of IEEE ICDE, pp.335-336, 2002.

[8] I. Tatarinov, "Storing and Querying Ordered XML Using a Relational Database System," In Proc. of SIGMOD, pp.204-215, 2002.

[9] 박영호, 한욱신, 황규영, "정보 검색 기술을 이용한 대규모 이질적인 XML 문서에 대한 효율적인 선형 경로 질의", 정보과학회논문지:데이타베이스, 제31권, 제5호, pp.540-552, 2004.

[10] 박영호, 한욱신, 황규영, "프리픽스 매칭 조인을 이용한 XML 문서에 대한 분기 경로 질의 처리", 정보과학회논문지:데이타베이스, 제32권, 제4호, pp.452-472, 2005.

[11] C. Zhang et al., "On Supporting Containment Queries in Relational Databases Management Systems," In Proc. of SIGMOD, pp.425-436, 2001.

[12] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, D. Suciu, "XML-QL,"

QL, 1998.

[13] H. Ishikawa, K. Kubota, Y. Kanemasa, "XQL: A Query Language for XML Data," QL, 1998.

[14] D. D. Chamberlin, J. Robie, D. Florescu, "Qult: An XML Query Language for Heterogeneous Data Sources," In Proc. of WebDB, pp.53-62, 2000.

[15] XQuery 1.0: An XML Query Language, W3C Working Draft, http://www.w3.org/TR/xquery/, 2002.

[16] J. Clark and S. DeRose, XML Path Language(XPath), W3C Recommendation, http://www.w3.org/TR/xpath, 1999.

[17] 박충희, 구홍서, 이상준, "역방향 레이블 경로를 이용한 XML 문서의 선형 경로 질의 처리", 한국 퍼지 및 지능시스템학회 논문지, 제17권, 6호, 2007.

[18] N. Bruno, N. Koudas and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching," In Proc. of SIGMOD, pp.310-321, 2002.

[19] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava and Y. Wu., "Structural Joins: A Primitive for Efficient XML Query Pattern Matching," In Proc. of IEEE ICDE, pp.141-152, 2002.

[20] H. Jiang et al., "Holistic Twig Joins on Indexed XML Documents," In Proc. of VLDB, pp.273-284, 2003.

[21] H. Jiang et al., "XR-Tree: Indexing XML Data for Efficient Structural Joins," In Proc. of IEEE ICDE, pp.253-264, 2003.

[22] J. Lu, W. Ling, C. Y. Chan, T. Chen, "From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching," In Proc. of VLDB, pp.193-204, 2005.

[23] B. Yang, M. Fontoura, E. J. Shekita, S. Rajagopalan, K. S. Beyer, "Virtual cursors for XML Joins," In Proc. of CIKM, pp.523-532, 2004.

[24] S. Chen, H. Li, J. Tatemura, W. Hsiung, D. Agrawal, K. S. Candan, "Twig^{2}Stack: Bottom-up Processing of Generalized-Tree-Pattern Queries over XML Documents," In Proc. of VLDB, pp.283-294, 2006.

[25] M. Altinel, M. J. Franklin, "Efficient Filtering of XML Documents for Selective

Dissemination of Information," In Proc. of VLDB, pp.53-64, 2000.

[26] Z. Ives, A. Levy, D. Weld, "Efficient Evaluation of Regular Path Expressions on Streaming XML Data," Technical Report UW-CSE-2000-05-02, University of Washington, 2000.

[27] J. McHugh, J. widom, "Query Optimization for XML," In Proc. of VLDB, pp.315-326, 1999.

[28] J. Bremer, M. Gertz, "XQuery/IR: Integrating XML Document and Data Retrieval," In Proc. of WebDB, pp.1-6, 2002.

[29] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram, "XRANK: Ranked Keyword Search over XML Documents," In Proc. of SIGMOD, pp.16-27, 2003.

[30] D. Florescu, D. Kossmann, Ioana Manolescu, "Integrating Keyword Search into XML Query Processing," In Proc. of WWW, pp.119-135, 2000.

[31] A. Halverson, J. Burger, L. Galanis, A. Kini, R. Krishnamurthy, A. N. Rao, F. Tian, S. Viglas, Y. Wang, J. F. Naughton, D. J. Dewitt, "Mixed Mode XML Query Processing," In Proc. of VLDB, pp.225-236, 2003.

[32] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, M. Shadmon, "A Fast Index for Semistructured Data," In Proc. of VLDB, pp.341-350, 2001.

[33] J. Yun, C. Chung, "Dynamic Interval-based Labeling Scheme for Efficient XML Query and Update Processing," Journal of Systems and Software, Vol.81, No.1, pp.56-70, 2008.

[34] R. Goldman, J. Widom, "Data Guides: Enabling Query Formulation and Optimization in Semistructured Databases," In Proc. of VLDB, pp.436-445, 1997.

[35] C. Li, T. W. Ling, "An Improved prefix Labeling Scheme: A Binary String Approach for Dynamic Ordered XML," In Proc. of DASFAA, pp.125-137, 2005.

[36] 박충희, 구홍서, 이상준, "XML 문서 갱신을 위한 확장 가능한 노드 넘버링 구조", 멀티미디어 학회 논문지, 제8권, 5호, pp.606-617, 2005.

[37] P. E. ONeil et al. "ORDPATHs : Insert-friendly XML Node Labels," In Proc. of SIGMOD, pp.903-908, 2004.

[38] D. Florescu, D. Kossmann, "Storing and Querying XML Data Using an RDBMS," IEEE Data Eng. Bull., Vol.22, No.3, pp.27-34, 1999.

[39] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, J. F. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," In Proc. of VLDB, pp.302-314, 1999.

[40] J, Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," In Proc. of VLDB, pp.65-76, 2000.

[41] J. McHugh, J. Widom, "Query Optimization for XML," In Proc. of VLDB, pp.315-326, 1999.

[42] U. of Wisconsin, "The Niagara System", Available from http://www.cs.wisc.edu /niagara/.

[43] A. Aboulnaga, A. R. Alameldeen, J. Naughton, "Estimating the Selectivity of XML Path Expressions for Internet Scale Applications," In Proc. of VLDB, pp.591-600, 2001.

[44] R. Kaushik, P. Bohannon, J. F. Naughton, H. F. Korth, "Covering Indexes for branching Path Queries," In Proc. of SIGMOD, pp.133-144, 2002.

[45] http://www.cs.wisc.edu/niagara/data/shakes.

[46] X. Wu, M. Lee, W. Hsu, "A Prime Number Labeling Scheme for Dynamic Ordered XML Trees," In Proc. of IEEE ICDE, pp.66-78, 2004.

감사의 글

본 논문이 있기까지 헌신적인 지도와 자상한 가르침을 베풀어 주신 이상준 교수님과 학문 연구에 바쁘신 가운데도 심사위원을 맡아 더욱 좋은 결실을 맺도록 지도해 주신 곽호영 교수님, 김도현 교수님, 박경린 교수님, 김철민 교수님께 깊은 감사를 드립니다. 아울러 항상 부족한 점을 깨우쳐 주시고 학술적인 조언을 아끼지 않으신 김장형 교수님, 안기중 교수님, 변상용 교수님, 송왕철 교수님, 변영철 교수님께 진심으로 감사드립니다.

바쁜 업무와 어려운 상황에서도 본 논문이 완성될 수 있도록 배려해 주신 제주산업정보대학 고희준 교수님, 양창우 교수님, 김대영 교수님께도 감사를 드립니다.

대학원 생활을 하면서 많은 도움을 준 지능 시스템 연구실 김영민 선생님, 김휴찬 선생님, 이종헌 선생님, 그리고 과정을 마친 연구실내 식구들에게도 고마운 마음을 전합니다. 또한 지금까지 저를 지켜봐 주시고 도와주신 많은 분들께 감사드립니다.

오늘이 있기까지 한없는 사랑으로 지켜봐 주신 부모님과 힘들 때마다 항상 격려와 믿음으로 지켜보며 응원해 주신 형님과 형수님들께도 깊은 감사를 드립니다.

끝으로 논문을 쓴다는 핑계로 같이 놀아주지 못한 용준이와 은지에게도 미안한 마음을 전하며 특히 집안일에 소홀한 남편에게 믿음과 위로를 보내준 사랑하는 아내에게는 그저 고마울 뿐입니다.

2007년 12월

박 충 희