경로 포함 관계를 이용한 효율적인 XML 질의 처리 기법

(An Efficient XML Query Processing Method using Path Containment Relationships)

민 경 섭 † 김 형 주 ††
(Kyung-Sub Min) (Hyoung-Joo Kim)

요 약 XML 이 명실상부한 데이타 교환 표준 언어로서 자리 잡아 감에 따라, 효율적인 XML 질의 처리 지원에 대한 많은 연구가 진행되어 왔다. XML 질의 처리에서 가장 중요한 고려 사항은 '어떻게 XML 경로식을 효율적으로 처리할 것인가' 하는 것이다. 기존의 몇몇 연구들은 질의를 구성하는 각 라벨들에 대응되는 모든 데이타에 대해 일련의 조인 연산을 수행하는 방식으로 결과를 생성하였다. 다른 몇몇 연구들은 질의에 포함된 경로를 찾기 위해 RDBMS의 문자열 비교 연산자를 사용하고 찾은 경로들에 대응되는 레코드를 추출하여 결과를 생성하였다.

본 연구에서 우리는 경로 포함 관계를 기반으로 한 새로운 질의 계획 생성 방식과 이를 지원하기 위한 두 개의 새로운 조인 연산자들을 제안하였다. 제안한 조인 연산자들은 질의에 속한 경로들과 관련된 데이타만을 입력 데이타로 사용하며, 매우 적은 비교 연산만을 수행한다. 그리고 파이프라인 기법을 적용하여 결과를 생성한다. 우리는 제안한 기법이 다른 이전 기법들에 비해 높은 성능을 보임을 분석과 실험을 통해 증명하였다.

키워드 : XML, 경로 포함 관계, 질의 처리

Abstract As XML is a de facto standard for a data exchange language, there have been several researches on efficient processing XML queries. The most important thing to consider when processing XML queries is how efficiently we can process path expressions in queries. Some previous works make results by performing a sequence of join operations on all records corresponding to labels in the path expression. Others works check the existence of paths in the query using an RDBMS's string comparison operator and make results by extracting the records corresponding to the paths.

In this paper we suggested a new query planning algorithm based on path containment relationships and two join operators supporting the planning algorithm. The join operators use only the records related to the paths in a query as input data, scan them only once, and generate result data using a pipelining mechanism. By analysis and experiments, we confirmed that our techniques(a new query planning algorithm and two join operators) achieved significantly higher performance than other previous works.

Key words : XML, Path Containment Relationship, Query processing

1. 서 론

XML[1] 이 데이타 교환과 문서 표현을 위한 실질적인 표준 언어로서 자리잡아감에 따라, XML 데이타 질의 처리에 대한 관심이 점점 증가하고 있다. 다양한 XML 질의 언어들, XML-QL[2], XQL[3], Quilt[4] 등이 제안되어 왔으며, 최근에는 W3C에서 XQuery[5]를 표준 언어로 제안하였다. 이들 언어들은 대부분, 가장 기본적인 연산으로, 특정 경로 패턴에 대응되는 데이타를 추출하는 경로식 처리 연산을 필요로 하며, 한 질의는 여러 경로식을 포함할 수 있어, 이들에 대한 효율적인 처리는 전체 질의 성능에 매우 핵심적인 부분을 차지한다.

경로식은 하나 이상의 라벨들로 구성되며, 경로식 내

† 비 회 원 : 서울대학교 전기컴퓨터공학부
ksmin@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2003년 6월 17일
심사완료 : 2004년 1월 26일

의 인접한 두 라벨 간에는 구조적으로 부모-자식 관계나 조상-자손 관계가 있다. 또한, 경로식은 둘 이상의 경로식을 함께 표현할 수 있는 데, 이러한 경로식을 분기 경로식(branching path expression)이라고 한다. 분기 경로식은 추출 대상을 표현한 기본 경로식(primary path)과 추출 대상에 대한 선택적 조건을 표현한 조건 경로식(predicate path)들로 구성된다. 한 개의 경로식만을 가진 경로식은 분기 경로식의 특별한 형태, 즉 기본 경로식만을 가진 분기 경로식으로 볼 수 있다. 분기 경로식은 여러 조건 경로식을 포함할 수 있으며 한 조건 경로식 내에 다른 조건 경로식들을 포함할 수 있으므로, 복잡하고 정교한 질의를 표현할 수 있다. 예를 들어, Wisconsin의 Xml DataSet[6] 중 Nasa XML 문서들에 대해 '식별자가 'I_113A.xml'이고, 개정 년도가 불분명한 특정 조직에 가입된 개정자들을 추출하라'는 질의를 던진다고 하자. 이 질의는 XPath 로 표현하면,

Q1 : //dataset[identifier='I_113A.xml']//revision[date/year='UNKNOWN']//creator[affiliation]과 같이 표현된다. 이 분기 경로식은 추출 대상인 creator까지의 기본 경로식, //dataset//revision//creator과 이에 대한 선택 제약을 표현한 조건 경로식들, //dataset/identifier='I_113A.xml', //dataset//revision/date/year= 'UNKNOWN',//dataset//revision//creator/affiliation로 구성된다. 이처럼 분기 경로식은 사용자 질의를 구성하는 여러 부분 경로식들과 이에 대한 조건 경로식들을 한 개의 경로식에 모두 표현할 수 있다.

경로식을 처리하기 위한 연구들은 활발히 진행되어 왔다. 이들 중 상당한 연구들[7,8,9]은 경로식을 구성하는 각 라벨들에 대응되는 모든 데이타를 추출하고, 이들을 서로 조인 연산함으로써 결과 데이타를 생성하는 방식을 제안하였다. 하지만, 이들은 경로식을 구성하는 라벨들의 경로 문맥을 고려하지 않아 각 라벨에 대응되는 매우 많은 불필요한 데이타를 추출하게 되며, 이로 인해 조인 연산을 수행하는 과정에서 많은 비교 연산을 초래한다. 한편, 경로를 기반으로 한 몇몇 질의 처리 연구들[10,11,12]은 한 개 이상의 조건을 포함한 분기 경로식에 대해 효율적인 처리 방법을 제공하지 못하고 있다.

이에, 본 연구에서는 경로를 기반으로 한 새로운 질의 계획 생성 방식과 이를 지원하기 위한 조인 연산자들을 제안하였다. 제안한 조인 연산자는 경로식을 처리하는 과정에서 중간 결과 데이타를 생성하지 않으며, 질의 계획에 따라 여러 조건 경로식을 함께 처리하는 것이 가능하도록 구현되었다. 또한, 질의를 구성하는 경로들을 중심으로 질의 계획을 생성함으로써, 수행되어야 하는 조인 횟수를 줄이고 질의 경로와 무관한 데이타가 조인 연산에 참여하지 못하도록 보장하였다. 우리는 이러한 질의 처리 기법을 기존 연구들을 보완하고 확장하는 방식으로 구현함으로써, 기존의 연구 결과를 활용할 수 있도록 하였다. 우리는 분석과 실험을 통해 기존의 방식에 비해 월등히 높은 성능을 보임을 확인하였다.

논문의 전체 구성은 다음과 같다. 먼저, 2장에서는 본 논문과 관련된 선행 연구들에 대해 살펴본다. 3장에서는 경로식 처리 방식을 위한 XML 데이타 모델, 데이타 저장 방식, 그리고 경로 기반 역-인덱스에 대해 간략히 소개한다. 4장에서는 우선 제안하는 물리 조인 연산자들에 대해 소개하고, 이를 이용한 새로운 질의 계획 생성 알고리즘과 전반적인 질의 처리 과정에 대해 설명한다. 5장에서는 제안한 기법의 효율성을 분석과 실험을 통해 보인다. 마지막으로 6장에서 결론과 향후 연구 방향에 대해 제시한다.

2. 관련 연구

경로식 형태의 질의를 처리하기 위한 연구는 활발히 진행되어 왔다.

반구조화된 데이타를 처리하기 위한 전용 시스템인 Lore[13]는 여러 인덱스들(특히 경로 정보를 가진 Data Guide[14])과 조인 연산들을 이용한 질의 최적화 방식[15]을 제안하였다. 이후, 데이타베이스 시스템을 이용한 경로식 처리 기법들이 꾸준히 제안되었다.

몇몇 연구들은 경로식 질의를 구성하는 모든 라벨들을 일련의 이항 관계로 재구성하고, 일련의 조인 연산들을 수행하는 방식을 사용하였다. 이들이 제안한 조인 알고리즘들은 공통적으로 각 라벨에 대응되는 XML 데이타베이스 내의 모든 데이타를 조인 연산의 입력으로 사용한다. Chun Zhang[7]은 여러 술어식을 함께 비교할 수 있는 새로운 조인 알고리즘인 MPMGJN과 캐쉬 정책을 제안하였다. Al-Khalifa, et al.[8]는 스택 기반의 조인 알고리즘을 제안하여 MPMGJN을 개선하였으며, Shu-Yao Chien, et al.[9]는 조인 시에 불필요한 데이타에 대한 접근을 줄이기 위해 B-Tree 를 이용한 조인 알고리즘을 제안하고, 동일한 라벨을 가진 노드를 가리키는 형제 포인터(sibling pointer)를 추가함으로써 성능을 향상시켰다. BongKi[16]는 세 개의 조인, EE-Join, EA-Join, KC-Join, 알고리즘들과 엘리먼트,애트리뷰트, 문서구조,값 등에 대한 인덱스들을 이용하여 실행 계획을 세우고 이를 수행함으로써 경로식을 처리하였다. 이러한 연구들은 모두 조인 연산의 입력으로 사용되는 각 라벨의 데이타에 질의 문맥과 무관한 많은 데이타를 포함하고 있어, 비교 연산의 횟수가 매우 많다는 단점을 가지고 있다. 또한, 각 조인 연산의 결과로 발생되는 중간 결과 데이타를 관리해야 하는 문제도 발생된다. Nicolas Bruno[17]는 중간 결과를 제거하기 위한 Twig

Stack 이라는 스택 기반의 조인 알고리즘을 제안하였으나, 조인 연산에 참여하는 불필요한 데이타에 대한 고려가 없었으며, 조상-자손 관계를 처리하는 데 있어서 최적화된 알고리즘을 제안하지 못하였다.

이와 달리, 본 연구와 유사한 접근 방식을 사용한 Yoshikawa[10], ChiYoung[11], EPIS[12]는 XML 문서의 구조를 표현하는 모든 경로들을 문자열로 표현하여 테이블에 저장하고, 각 경로에 대응되는 인스턴스들을 해당 경로의 식별자와 함께 다른 테이블에 저장하는 방식을 사용하였다. 그리고 경로식을 처리하기 위해 경로식을 SQL 질의로 변환하는 알고리즘을 제안하였다. 변환된 SQL 질의는 RDBMS 의 질의처리기를 이용하여 수행되므로 추가적인 구현을 필요로 하지 않으며, 기본 경로식만을 가진 경로식 질의에 대해서 매우 빠르게 결과를 생성한다. 하지만, 처리하고자 하는 경로식이 분기 경로식인 경우, 여러 SQL 문장들을 생성하며, 이들에 대한 통합된 질의 최적화를 하기 어렵다. 그리고 각 SQL 문장의 수행 결과에 따른 중간 결과 데이타에 대한 입출력 비용 또한 발생한다. 더욱 중요한 부분은 데이타베이스에서 사용하는 일반적인 조인 알고리즘과 질의 계획을 사용하므로, 경로식 질의 처리에 부적합한, 즉 불필요한 비교 연산들과 작업 단계를 거칠 수 있다는 것이다. 이는 결과적으로 전체적인 질의 성능을 크게 저하시킨다.

이에, 본 연구에서는 경로 기반 역-인덱스를 제안한 방법들과 유사한 방식으로 한 개의 경로식을 처리하고, 분기 경로식에 대해서는 새로운 실행 계획과 물리 연산자를 이용하도록 함으로써 효율적인 처리가 되도록 보장하였다.

3. 선행 작업

본 장에서, 우리는 제안하는 질의 처리 기법의 기반이 되는 XML 데이타 모델, 번호 매김 방식, 그리고 XML 저장과 인덱싱 방식에 대해 소개한다.

3.1 XML 데이타 모델과 번호 매김 방식

XML 문서는 유효한(valid) 문서이며, 순서가 있는(ordered) 트리로 가정한다. XML 문서는 문서의 구조를 나타내는 태그인 엘리먼트(element)와 엘리먼트의 특징을 기술하는 애트리뷰트(attribute), 엘리먼트나 애트리뷰트에 포함된 값(또는 문자열)인 용어(term)로 구성된다. XML에서 두 엘리먼트 간의 참조 관계를 표현하는 애트리뷰트인 ID와 IDREF 는 XML 문서를 그래프로 표현할 수 있도록 해준다. 본 연구에서는 XML 문서를 트리로 가정하므로, 두 애트리뷰트는 단순히 값을 포함하는 애트리뷰트로 취급한다. 또한, 모든 애트리뷰트는 '@' 문자를 이름 앞에 붙여 엘리먼트와 구별하며, 트리 모델 내에서는 동일하게 취급한다. 한편, 애트리뷰트는 엘리먼트와 같이 시작 태그와 종료 태그를 갖지 않으므로, 애트리뷰트 이름을 시작 태그로, 애트리뷰트의 값이 나온 뒤 바로 가상의 종료 태그가 있다고 간주한다.

트리로 표현된 XML 문서의 각 구성 요소(엘리먼트, 애트리뷰트,용어)들은 깊이우선순위(depth-first traversal) 방식으로 방문되며, 방문될 때마다 번호를 부여받는다. 엘리먼트와 애트리뷰트는 두 번씩 방문되며, 이들의 범위를 표현한다. 그러므로 두 노드간의 구조적인 포함 관계는 노드간의 범위 포함 관계로 결정할 수 있다.

그림 1은 Wisconsin의 Xml DataSet[6] 중 Nasa XML 문서의 예이다. dataset과 revision과 같은 엘리먼트들과 subject와 같은 애트리뷰트들은 모두 자신의 범위 값을 가지며, 모든 용어들은 발생 위치 정보를 가진다. subject와 같은 애트리뷰트는 @를 이름 앞에 표시하여 속성임을 표현한다.

3.2 문서 저장과 인덱스 지원 방식

3.1절의 XML 데이타 모델과 함께, 우리는 하위 저장 모델과 인덱스로 경로 기반 역-인덱스 구조를 사용한다. 경로 기반 역-인덱스는 데이타베이스에 저장된 XML 데이타에서 특정 경로에 속하는 데이타를 빠르게 추출해 준다. 우리는 이러한 역-인덱스들 중 EPIS[12]를 문서 저장과 인덱싱에 사용한다. 본 연구와 관련하여 EPIS는 기존의 다른 방식들에 비해 더 나은 공간 효율성을 보인다는 점에서 채택하였을 뿐, 다른 경로 기반 역-인덱스를 사용해도 무방하다.

EPIS를 이용한 저장과 인덱싱 방식은 다음과 같다. 먼저 우리는 위의 XML 데이타 모델에 따라 재구성된 XML 문서에 대해 깊이-우선 탐색 방식에 따라 이동하면서, 문서를 구성하는 경로를 식별한다. 그리고 식별된 경로에 대응되는 엘리먼트(혹은 애트리뷰트)나 용어들에 대한 발생 정보를 각 경로의 식별자와 함께 저장한다. 경로에 대응되는 엘리먼트(혹은 애트리뷰트) 발생 정보를 표현한 테이블은 '문서번호/경로식별자/시작위치/종료위치' 형식으로 구성되며, 용어 발생 정보를 표현한 테이블은 '용어식별자/문서번호/경로식별자/발생위치' 형식으로 구성된다. 여기서 용어식별자는 각 용어에 부여된 식별자로서, 용어들에 대한 정보는 용어 테이블에 따로 구성된다. 이와 같은 방식으로 문서 순회를 마치면, 우리는 해당 문서에 대한 모든 발생 정보와 경로들을 가지게 된다. 이 때, 구성된 경로들을 라벨[1]별로 분리하여 라벨 테이블을 생성한다. 라벨 테이블의 구조는 '라벨이

1) 경로를 구성하는 엘리먼트나 애트리뷰트를 말함. /A/B/@C와 같은 경로에서 A, B, @C를 경로의 라벨이라 함.

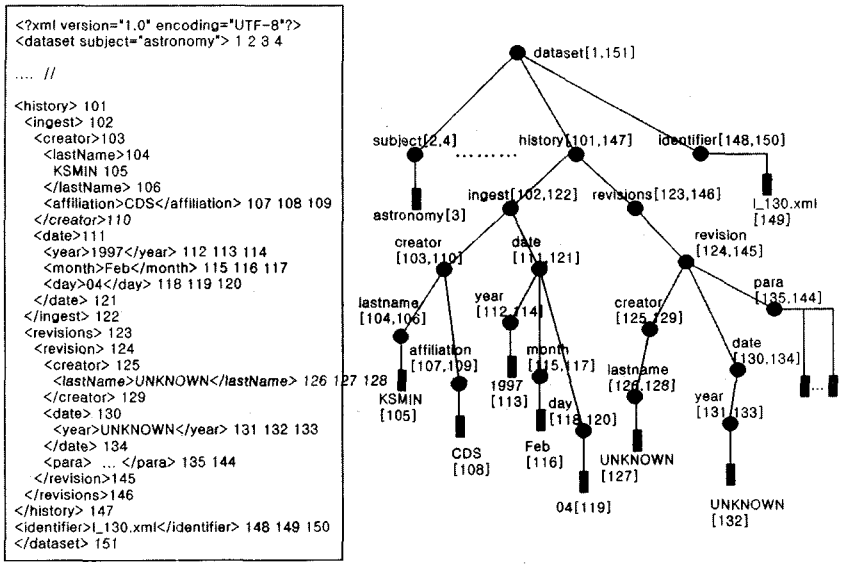


그림 1 XML 문서와 데이타 모델 예

름/경로식별자/경로라벨' 형식으로 표현된다. 경로라벨은 경로내에서 해당 라벨이 몇 번째 위치인가를 표현한다. 각 테이블에 대해서는 앞서 보인 구조의 모든 속성들에 대한 클러스터화된 인덱스(clustered index)를 유지하도록 하여, 데이타의 순서를 물리적으로 유지한다. EPIS로부터 특정 경로에 해당되는 데이타를 추출하기 위해서 우리는 경로식을 변환한 SQL 문장을 수행한다. 그림 2는 그림 1의 XML 문서를 저장한 역-인덱스 구조를 보여준다.

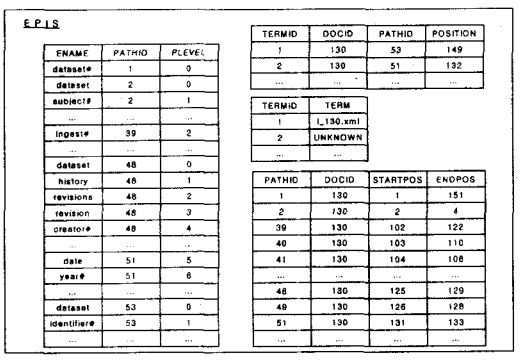


그림 2 EPIS 구성 예

4. 경로식 질의 처리 기법

본 장에서는 3장에서 소개한 XML 데이타 저장과 검색 기법을 기반으로 하여 임의의 복잡한 경로식을 효율적으로 처리할 수 있는 새로운 질의 계획 생성 방식과 물리 연산자들에 대해 소개한다.

4.1 물리 연산자

4.1.1 개요

일반적으로 경로식 처리에 있어 성능을 좌우하는 가장 중요한 물리 연산자(Physical Operator)는 인덱스와 조인 연산이다. 우리는 앞서 3장에서 EPIS를 기반으로 한 저장, 검색 모델을 채택하였으므로, 이 인덱스를 물리 연산자로 사용한다. 한편, EPIS를 포함한 모든 경로 기반 역-인덱스들을 이용한 질의 처리 방식은 여러 경로식을 포함하는 질의에 대해 비효율적인 조인 연산을 수행한다. 그러므로 우리는 각 경로식으로부터 추출된 데이타 간의 조인 연산을 효율적으로 수행하기 위한 새로운 물리 조인 연산자들인 B-Join 연산자와 T-Join 연산자를 제안한다. 질의 처리를 위한 다른 추가적인 물리 연산자들은 향후 추가로 지원할 것이다.

두 조인 연산자는 다음과 같은 공통된 특징을 가지고 있다.

첫째, 두 조인 연산자는 특정 순서로 정렬된 데이타에 대해 조인 연산을 수행한다. 즉, 입력 데이타가 특정 순서로 정렬되어 있어야 한다. 우리가 가정하는 역-인덱스들은 모두 이 특징을 만족하므로, 이는 매우 쉽게 적용 가능하다.

둘째, 두 조인 연산자는 두 입력(S,T라 하자) 중 한 쪽의 데이타만을 결과로 생성한다. 즉, 조인 조건을 만족하는 두 레코드(S에 속한 s, T에 속한 t)에 대해 기존의 라벨 기반 조인 방식[7,8]에서는 두 레코드를 하나로 결합하여 새로운 레코드를 생성하지만, 제안하는 두

연산자는 S나 T에 있는 레코드만을 결과로 생성한다. 이는 중간 데이타의 크기를 줄여준다.

셋째, 조인 연산의 결과로 생성되는 데이타는 중복된 레코드를 갖지 않는다. 제안하는 두 연산자는 입력 데이타(S라 하자)로부터 추출된 s라는 레코드가 조인 조건을 만족하는 경우 이를 출력하고, 다음 레코드에 대한 비교 연산을 수행한다. 그러므로 결과 데이타는 중복된 데이타를 갖지 않으며, 조인 조건을 만족하는 데이타로 필터링된 효과를 가진다.

넷째, 두 조인 연산자는 반복자(Iterator) 방식으로 동작한다. 즉, 조인 연산을 수행하기 위한 준비 작업을 open 연산을 통해 제공하고, 조인 조건을 만족하는 새로운 결과 데이타를 산출하는 next 연산을 제공하며, 조인 연산이 모두 종료되었을 때, 사용되었던 모든 리소스를 반환하는 close 연산을 제공한다. 반복자 방식을 사용함으로써 우리는 중간 결과를 관리해야 하는 부담을 제거할 수 있으며, 이에 따라 질의 성능이 향상되는 결과를 얻는다.

우리는 이러한 특징들을 가진 두 조인 알고리즘을 이용하여 향상된 질의 성능을 얻을 수 있다. 다음은 각 조인 연산자에 대한 동작 방식이다.

4.1.2 T-Join 연산자

이 조인 연산자는 MPMGJN[7] 과 유사한 조인 연산자로서, 경로식 상에 용어에 대한 비교를 포함하는 경로식에 대한 조인 연산을 수행하기 위한 연산자이다. 예를 들어, Q1 질의에서 조건 경로식에 해당되는 //dataset//revision/date/year='UNKNOWN'은 year가 'UNKNOWN'이라는 용어를 갖는 지를 검사하고, 대응되는 year 데이타를 추출한다. 이를 처리하기 위해, 기존의 경로 기반 역-인덱스들은 용어 발생 테이블과 경로 발생 테이블 간에 일반적인 조인 알고리즘(Nested Loop, Merge-Sort, etc)을 사용한다. 그러므로 두 입력 데이타간의 많은 상호 비교 연산이 필요하며, 불필요한 데이타를 생성해 낸다. T-Join 연산자는 이러한 문제점을 해소하기 위해, 4.1.1에서 언급한 특징을 가지는 새로운 조인 알고리즘을 제공한다. 그림 3의 a)는 T-Join 알고리즘이다. 이 조인 알고리즘은 Open 연산을 통해 용어가 나타나기 전까지의 경로식에 대한 대응되는 경로식별자들을 추출하고, 이 경로식별자들을 이용하여 경로 발생 테이블과 용어 발생 테이블에서 대응되는 데이타를 추출하기 위한 준비 작업을 수행한다. 다음으로, Next 연산에서는 두 입력 데이타로부터 문서 번호, 경로 번호, 범위가 조건에 대응되는 레코드를 찾기 위한 비교 연산을 수행한다. 만약 대응되는 레코드가 있으면, 이 레코드를 출력한다. 만약 대응되는 레코드가 없는 경우, 입력이 더 이상 없을 때까지 계속해서 비교 연산을 수행한다.

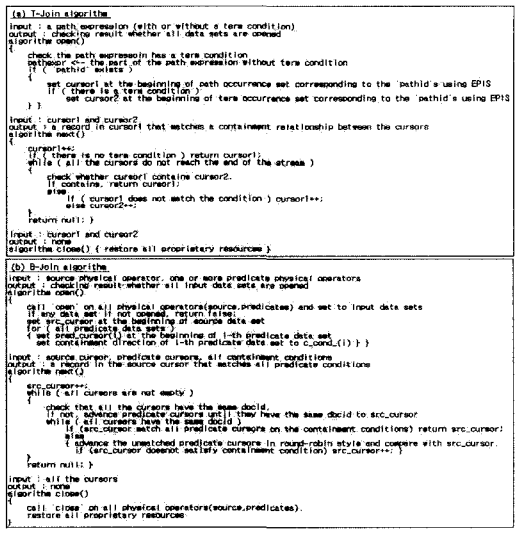


그림 3 물리 조인 연산자

이 때, 두 입력 데이타에 대해서는 한 번의 순차 접근만을 필요로 하므로, 조인 연산을 위한 최대 접근 횟수는 두 입력 데이타의 크기의 합이다. 즉 S의 크기를 n이라 하고, R의 크기를 m이라 할 때, n+m 번의 데이타 접근만을 필요로 한다.

4.1.3 B-Join 연산자

이 조인 연산자는 둘 이상의 임의의 수의 입력 데이타에 대해 조인 연산을 수행하기 위한 연산자로서, 한 개의 출력 대상 입력 데이타와 이에 대한 제약 조건에 대응되는 한 개 이상의 입력 데이타들로 구성된다. 이 연산자는 다른 B-Join 연산자나 T-Join 연산자의 결과 데이타를 입력으로 받는다. 이 연산자는 한 개 이상의 조건 입력들을 가지도록 함으로써 모든 조건을 만족하는 경우에만 새로운 결과 레코드를 생성하도록 한다. 만약 출력 대상 레코드와 어떤 조건 입력의 레코드가 조건을 만족하지 못하면, 그 상황에 따라 둘 중 하나의 커서를 증가시켜 다음 레코드와 비교한다. 이와 같은 작업이 모든 조건 레코드에 대해 수행되고 모두 조건을 만족하는 경우에만, 출력 대상 레코드가 결과 레코드로 생성된다. 만약 비교 작업을 수행하는 중에, 어느 한 입력의 레코드가 더 이상 없으면 다른 입력들의 레코드들이 얼마나 남아있던지 관계없이 조인 연산은 종료된다.

이처럼 각 조건 입력 데이타에 대해 함께 비교 연산을 수행함으로써, 각각을 따로 조인 연산했을 때 발생되는 Next 함수 수행의 부하를 제거할 수 있다. 이 조인 연산의 최대 레코드 접근 횟수는 각 입력 레코드 수의 합이다. 즉, 출력 대상 입력을 S, 레코드의 수를 n이라 하고, 각 조건 입력들을 R_1, R_2, ..., R_k, 레코드의 수를

r_1, r_2, r_k라 할 때, 최대 레코드 접근 비용은 $n+(r_1+r_2+...+r_k)$ 이다. 그림 3의 b)는 이 연산자의 알고리즘이다.

4.2 질의 실행 계획(Query Execution Plan)

일반적으로 질의 실행 계획을 생성하기 위해서는 사용자 질의를 파싱하고, 내부 형식으로 표현된 질의를 변환하여 논리 실행 계획을 생성한다. 그리고 시스템에서 제공하는 여러 물리 연산자들을 조합, 적용하여 최적화된 물리 실행 계획을 생성한다. XML 질의를 처리하기 위한 과정도 이와 동일하다. 우리는 이와 같은 일반적인 단계에 따라 질의 실행 계획을 생성한다. 하지만, 기존의 XML 질의 계획 생성 방식과는 근본적으로 다르다. 기존의 XML 질의 계획 생성 방식은 단순히 질의를 구성하는 각 라벨들이 데이타베이스에 존재하는 가에 의존한다. 하지만, 본 논문에서 제안하는 방식은 질의를 구성하는 각 라벨이 데이타베이스의 어떤 경로에 속해야 하는 가에 대한 정보를 이용하여 질의 계획을 생성한다. 이 장에서는 이와 같은 질의 수행 계획 생성 방식에 대해 단계별로 소개한다.

4.2.1 내부 질의 표현

사용자로부터 입력된 경로식은 하나 이상의 경로식을 포함할 수 있다. 그러므로 우리는 입력된 경로식은 트리 형태를 띤다. 그림 4는 도입부분에서 참조했던 Q1 질의를 트리 형식으로 표현한 것이다. 이 그림에서, 한 개의 질의 경로식은 한 개의 기본 경로식(Primary Path)과 여러 개의 조건 경로식(Predicate Path)들로 구성된다. 기본 경로식과 조건 경로식은 모두 부모-자식, 조상-자손의 구조적 포함 관계를 표현할 수 있으며, 조건 경로식은 기본 경로식이나 다른 조건 경로식에 속한 임의의 라벨에서 분기한다. 이 때 분기가 시작되는 라벨을 분기점(Branching Point)라고 부르며, 질의 트리에는 분기점의 수만큼의 조건 경로식이 존재한다. 위 질의에는 세 개의 분기점이 있으므로, 세 개의 조건 경로식을 가지게 된다.

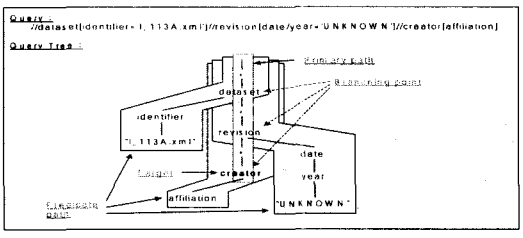


그림 4 내부 질의 트리 예

우리는 이러한 질의 트리의 구조를 기반으로 경로를 중심으로 한 새로운 내부 질의 트리를 생성한다. 그림 5는 Q1에 대한 경로 기반 내부 질의 트리이며, 그림 6은

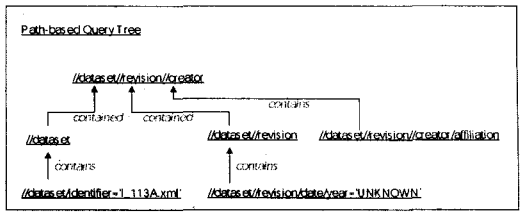


그림 5 경로식 기반 질의 트리 예

```
Algorithm makeQueryTree(P) { R = makeTreeNode(null,P); return R; } // R is the root node of the query tree.
Algorithm makeTreeNode(N, P)
/* N: a node of the query tree */
/* X,Y : a new node of path-based query tree. */
/* preds : an array which contains predicate path expressions surrounded by '[' and ']' */
/* prevPath : a path expression */
{if ( N == null ) { prevPath = null; }  // for root node.
 else { prevPath = N.pathexpr; }
 P' = prevPath + primaryPathExpr(P);  // remove all predicate path expressions which are surrounded by '[' and ']'
 X.pathexpr = P';
 if ( N != null ) { N.predNodes(0) = X; N.predRels(0) = the containment relationship between N and X. }
 while ( ! End of P )
  if ( found '[' symbol )
  { j = the position of the new '[' symbol.
    Y.pathexpr = the path expression except paths surrounded by '[' and ']' from the start of P to j;
    X.predNodes(i) = Y; X.predRels(i) = the containment relationship between X and Y.  // i is an index for predicates
    Q = a first path expression surrounded by '[' and ']' starting from j.  // Q is a path expression
    preds(i) = Q;  i++;
  }
for ( int k =0; k < i; k++ ) { makeTreeNode(X,predNodes(k), preds(k)); }
return X; }
```

그림 6 내부 질의 트리 변환 알고리즘

이 내부 질의 트리를 생성하기 위한 알고리즘이다. 그림 6의 알고리즘을 통해 생성되는 내부 질의 트리는 기본 경로식을 루트 노드에 배치하며, 기본 경로식에 있는 분기점까지의 경로식을 하위 노드로 한다. 그리고 그 경로식과 기본 경로식 간의 포함 관계를 표현한다. 다음으로, 기본 경로식 자체가 분기점이 되는 경우, 대응되는 조건 경로식을 기본 경로식의 하위 노드로 한다. 예를 들어, 그림 5의 //dataset//revision//creator/affiliation 경로가 이에 해당된다. 다음, 기본 경로식의 하위 노드로 조건 경로식을 설정한다. 예를 들어, 그림 5에서의 //dataset 노드의 하위 노드는 //dataset/identifier= 'I_113A.xml' 이 된다. 만약 조건 경로식 내에 추가적인 분기점이 있으면, 앞선 과정을 반복하여 트리를 확장한다.

4.2.2 논리 실행 계획

전 단계에서 생성된 경로식 기반 내부 질의 트리를 기반으로, 우리는 논리 실행 계획을 생성한다. 그림 7은 논리 실행 계획을 생성하기 위한 알고리즘이다.

논리 실행 계획은 두 개의 연산(Extract와 Join)들의 조합으로 구성된다. Extract는 한 개의 경로식에 대응되는 데이타를 데이타베이스로부터 추출하는 논리 연산자이며, Join은 추출된 두 데이타 셋에 대한 포함 관계 조인 연산자를 의미한다. 조인 연산의 수행 결과는 조인 연산의 두 입력 중 포함 조건을 만족하는 한쪽 입력의 데이타이다. 논리 실행 계획을 생성하는 과정은 간단하다. 먼저, 경로 기반 내부 질의 트리에 포함된 모든 경로식들에 대해 Extract 연산을 적용한다. 그리고 단말 노드에 있는 경로식과 해당 노드의 부모 노드에 있는

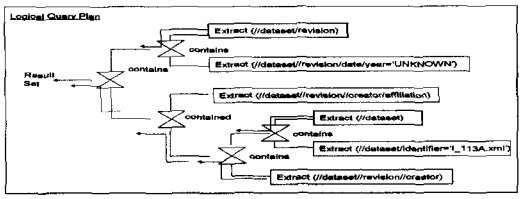


그림 7 논리 실행 계획 예

```
Algorithm makeLogicalQueryPlan(R)
/* R : the root node of the query tree, S : a stack holds Extract or Join operators */
{makeLQPNode(S,R); return S.pop(); }
Algorithm makeLQPNode(S, V)
/* V : a node of the query Tree, E : an extract operator, J : a join operator */
{make a new E; E.pathExpr = V.pathExpr; S.push(E);
 for ( int i=0; i < the number of children of V; i++ ) {makeLQPNode(S,V.predNodes(i)); make a new J;
  J.op2 = S.pop(); /* child node */  J.op1 = S.pop(); /* parent node */ J.rel = V.predRels(i); S.push(J); } }
```

그림 8 논리 실행 계획 생성 알고리즘

경로식에 대해 조인 연산을 적용한다. 이 때, 조인의 방향을 contains와 contained 관계를 이용하여 결정한다. 그리고 조인 연산에 대응되는 결과는 부모 노드에 해당되는 데이타를 출력하도록 구성한다. 만약 하나 이상의 노드들이 동일한 부모 노드와 구조적 포함 관계를 가진 경우, 각 노드에 대해 순차적인 조인 연산을 수행하도록 구성한다.

그림 8은 그림 5에 대한 논리 실행 계획의 예이다. 이 그림에서, 각 Extract 연산은 해당 경로식에 대한 데이타를 추출하며, 추출된 데이타에 대해 부모-자식 노드에 대응되는 데이타 간에 포함 관계를 비교하는 조인 연산을 수행한다. 그림에서는 각 조인 연산의 수행 결과로 추출되는 데이타가 어떤 것인지를 화살표로 표시하였다. 단계별로 살펴보았을 때, 최종적으로 기본 경로식에 대응되는 데이타가 추출되는 것을 알 수 있다.

4.2.3 물리 실행 계획

생성된 논리 실행 계획에 대해 우리는 4.1에서 제안한 두 개의 물리 조인 연산자와 EPIS를 이용하여 물리 실행 계획을 생성한다. 그림 9는 물리 실행 계획을 생성하기 위한 알고리즘이다. 우리는 먼저 논리 실행 계획에 있는 모든 Extract 연산을 T-Join 물리 연산자로 매핑한다. T-Join 연산자는 Extract가 소유하던 경로식을 갖는다. 다음, 논리 실행 계획에 있는 일련의 조인 연산들에 대해, 이를 B-Join 물리 연산자로 매핑한다. 이 때, 경로 질의 트리에서 동일한 부모 노드를 가진 하위 노드들에 대응되는 모든 조인 연산들은 한 개의 B-Join 연산자에 포함된다. 즉, 어떤 경로에 대한 모든 조건 경로들은 모두 함께 조인 연산되도록 한 개의 B-Join 연산자의 입력으로 구성된다. 이와 같은 방식으로 논리 실행 계획의 모든 조인 연산들을 B-Join 연산자로 재구성하면, 최종적으로 물리 실행 계획이 완성된다. 그림 10은 그림 8의 논리 실행 계획을 이용하여 물리 실행 계획을 생성한 예이다. 각 Extract 연산은 T-Join 연산자

```
Algorithm makePhysicalQueryPlan(L)
/* L : the root node of the logical query plan, TopQP : the top-level B-Join operator, PQT : the path-based query tree */
{ TopOP = makePQPNode(L); return mergeBJoins(TopOP, the root of PQT); }

Algorithm makePQPNode(L)
/* T : a T-Join operator, B : a B-Join operator */
{if ( L is Extract operator ) { make a new T; T.pathExpr = L.pathExpr;
   if ( L.pathExpr has a term condition ) { T.hasTermCond = true; }
   else T.hasTermCond = false; return T; }
 else // L is Join operator
  {make a new B; B.op1 = makePQPNode(L.op1); B.predOps(0) = makePQPNode(L.op2); B.rel = L.rel; return B; } }

Algorithm mergeBJoins(TopOP, R) // R is the root of PQT
{ while ( there is a node to visit in R ) {
   N = a new visited node.
   if ( N has two or more predicates )
   { X = the B-Join node which contains a T-Join node holding the same path expression with N;
     for ( each of predicate nodes of N )
     { Y = the T-Join node holding the same path expression with current predicate node;
       Z = the B-Join operator containing Y.
       if ( Z's other operand node is T-Join operator )
       { cut Z from Z's parent B-Join Operator; X.predOps.insert(Z); } // register Z as a new predicate source
       else { cut Y from Z; X.predOps.insert(Y); } } }
  remove all B-Join nodes with one operand and if necessary move TopOP to the new top-level node.
  return TopOP. }
```

그림 9 물리 실행 계획 생성 알고리즘

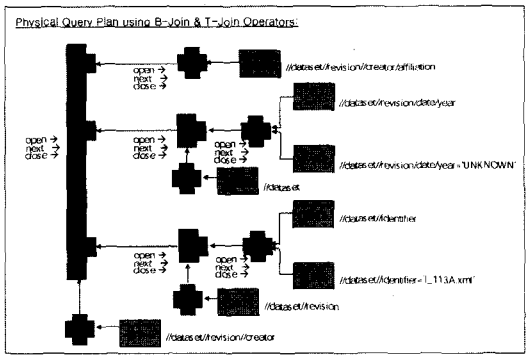


그림 10 물리 실행 계획 생성 예

와 T-Join 연산자에서 내부적으로 이용하는 EPIS에 의해 처리되도록 구성되었으며, 중간 단계의 조인 연산들은 B-Join 연산자로 처리되도록 구성되었다. 특히, 생성된 물리 실행 계획의 최종 레벨 B-Join 연산자는 논리 실행 계획의 세 조인 연산을 하나로 결합하여 처리하도록 구성되었다.

이렇게 생성된 최종 실행 계획은 라벨을 기반으로 한 실행 계획에 비해 더 적은 횟수의 조인 연산을 수행한다. 그림 10에서 용어를 포함한 T-Join 연산자들과 B-Join 연산자들이 수행하는 조인 횟수는 7번이다. 만약 라벨을 기반으로 조인 연산을 수행한다면, 8번의 조인 연산이 필요하다. 특히, 조건을 포함하지 않은 부분 경로들이 길수록, 조인 횟수는 더욱 큰 차이를 나타내게 된다. 그러므로 생성된 물리 실행 계획은 라벨 기반의 실행 계획에 비해 더 효율적인 수행이 가능하다.

4.2.4 질의 수행

생성된 물리 실행 계획은 파이프라인 방식을 통해 수행된다. 그러므로 중간 결과를 저장, 관리하지 않으며, 이와 관련된 어떠한 비용도 발생하지 않는다. 제안한 두 연산자는 이러한 방식을 위해 open, next, close 세 개의 인터페이스를 제공한다.

```
Algorithm QueryProc()
/* P : a path query */
/* PQT : a path-based query tree */
/* LQP : a logical query plan */
/* PQP : a physical query plan */
/* BOper : a B-join object */
{
 input a path query into P.
 PQT ← makeQueryTree(P); // make a path-based query tree
 LQP ← makeLogicalQueryPlan(PQT); // make a logical query plan. It get PQT's root node as its argument.
 BOper topOP= makePhysicalQueryPlan(LQP); // make a physical query plan. It gets LQP's root node as its argument.

 topOP.open(); // make all the preliminary work for processing
 While ( (resultRec = topOP.next()) ) /* repeat until there is a result record. */
 { insert resultRec into a result table or send it directly to external application object. }
 topOP.close(); // restore its temporary resources
}
```

그림 11 질의 수행 알고리즘

그림 11은 질의 수행 알고리즘이다. 질의 처리는 최종 레벨의 B-Join 연산자에 대해 open 명령을 전달하는 것으로 시작된다. 그러면 이와 관련된 모든 하위 B-Join 연산자의 open 연산들이 수행되며, 그 하위의 B-Join, 혹은 T-Join 연산자의 open 명령을 재귀적으로 호출한다. 만약 open 연산을 수행하는 과정에서 한 개라도 제대로 open 되지 않으면, 질의는 결과를 생성하지 않고 종료된다. 모든 연산자들이 정상적으로 open 되었으면, 새로운 결과 레코드를 추출하기 위해 최종 레벨의 B-Join 연산자의 next 연산을 수행한다. 그러면, 내부적으로 이 연산자의 하위 연산자들에 next 연산을 수행하여 데이타를 추출하고, 비교 연산을 수행하여 모든 조건을 만족하는 결과 레코드를 생성한다. 만약 next 연산을 수행하는 중에 어떠한 입력으로부터 더 이상 데이타를 추출할 수 없는 상황에 도달하면, next 연산은 결과 레코드 생성없이 작업을 종료한다. next 연산은 한 번 수행될 때 한 개의 레코드를 생성하며, 반복적으로 수행함으로써 질의에 대응되는 모든 레코드를 추출한다. 더 이상 레코드를 추출하지 못하게 되면, 우리는 close 연산을 호출하여 물리 실행 계획에서 임시로 유지하고 있는 모든 자원을 해제함으로써 질의 수행을 마친다. 이처럼 질의 수행은 최종 레벨의 open, next, close 연산을 실행하는 것으로 간단하게 진행된다.

5. 성능 평가

이 장에서는 본 논문에서 제안한 질의 처리 기법이 기존의 다른 여러 질의 처리 기법들에 비해 효율적임을 증명한다. 우선, 5.1절에서는 MPMGJN[7]이나 구조적 조인(structural join)[8]과 같은 라벨 기반의 조인 알고리즘을 이용한 질의 처리 방식과 본 논문에서 제안한 질의 처리 방식간의 성능을 분석한다. 두 방식에 대해 우리는 질의 계획에서 생성한 초기 입력 데이타의 크기와 조인 연산 시 요구되는 비교 연산 횟수를 분석하여 성능을 비교한다. 한편, EPIS는 우리가 제안한 방식과 유사하게 경로 정보를 이용하지만, 경로 질의를 SQL 질의로 변환하고 모든 처리를 DBMS에 위임하는 방식을 사용한다. DBMS는 상황에 따라 서로 다른 질의 계획을 생성하므로, 둘 사이의 성능을 분석적으로 파악하기 어렵다. 이에 우리는 5.2절에서 둘 간의 성능 차이를 실험을 통해 분석하기로 결정하였다. 우리는 위의 두 가지 평가를 통해 제안한 방식이 기존의 다른 여러 방식들에 비해 효율적인 방식임을 입증하였다. 각 성능 평가에 대한 구체적인 내용은 다음과 같다.

5.1 라벨 기반 질의 처리 방식과의 비교

이 절에서는 MPMGJN과 구조적 조인(structural join)과 같은 라벨 기반 조인 알고리즘을 사용하는 질의 처리 방식(여기서는 간단히 LBQP라 부른다)과 본 논문에서 제안한 질의 처리 방식(여기서는 간단히 PBQP라 부른다)의 성능을 비교한다. 두 방식을 비교하기 위해 우리는 두 방식의 조인 횟수, 조인 연산을 수행하는 데 필요한 비용을 계산한다. 조인 연산을 수행하기 위해 필요한 비용은 입력 셋의 크기와 비교 연산 횟수로 결정된다. 전체 질의 처리 과정에서 두 방식 간에 서로 다른 부수적인 특징들(예:Pipelining의 적용 여부)도 질의 처리 성능에 영향을 끼치지만, 이들은 분석적으로 평가하기 힘든 부분이며 두 방식에 공통적으로 제공될 수 있는 부분이므로, 이들은 고려하지 않는다.

어떤 XML 문서 D에 대해 우리는 3.1절의 EPIS 데이타 모델을 적용한다. 이 모델에서 각 노드는 특정 라벨에 대응되며, 문서 내에서의 발생 위치 정보를 가진다. 또한 각 노드는 루트 노드로부터 해당 노드까지의 라벨들로 구성된 특정 경로 상에 위치한다. 예를 들어, 그림 1에서 'history[101,147]'는 history라는 라벨에 대응되는 노드이며 발생 정보[101,147]을 가진다. 그리고 이 노드는 루트에서 해당 노드까지의 경로, 즉 '/dataset/ history' 상에 존재한다. 이처럼 트리 상의 모든 노드들은 루트로부터의 어떤 경로 상에 반드시 나타난다. 한편, 어떤 라벨에 대응되는 노드들은 서로 다른 경로에서 나타날 수 있는 데, 예를 들어, 그림 1에서 date 라벨에 대응되는 두 노드 date[111,121]과 date[130,134]는 각각 서로 다른 두 경로, '/dataset/history/ingest/date'와 '/dataset/history/revisions/revision/date'에 위치한다. 하지만, 한 경로가 서로 다른 여러 라벨에 대응되지는 않는다. 그러므로 어떤 경로는 한 라벨에 속한다고 할 수 있다.

즉, 문서 D에 있는 각 라벨 L은 루트로부터 시작된 한 개 이상의 경로, P_0, ..., P_k, $k>0$를 통해 도달될 수 있으며, L에 대응되는 노드 수는 P_0, ..., P_k에 대응되는 노드 수 C_0, ..., C_k의 합, $C_0+C_1+ \dots + C_k$로 정의할 수 있다. 이와 같은 기본적인 특징으로부터 우리는 경로 질의에 대한 두 방식의 비용을 계산한다.

• 조인 횟수:

어떤 경로 질의가 주어졌을 때, LBQP는 질의를 구성

하는 각 라벨에 대응되는 모든 노드들을 질의 처리를 위한 입력 셋으로 사용한다. n 개의 라벨로 구성된 질의에 대해 LBQP는 n-1 번의 조인 연산을 필요로 하며, 각 라벨의 입력 셋의 크기는 위에 정의한 것처럼 해당 라벨에 대응되는 모든 경로들의 노드 수의 합이 된다. PBQP는 질의를 구성하는 경로들을 중심으로 질의 계획을 생성하므로, 질의를 구성하는 경로들의 수 p 만큼 입력 셋이 생성된다. 이 때 생성되는 경로의 수는 '분기점의 수*2'(마지막이 조건경로인 경우) 또는 '분기점의 수*2+1'(마지막이 조건경로가 아닌 경우)이며, 모든 경우에 대해 경로를 구성하는 라벨 수 n 보다 작거나 같다.

이는 다음과 같이 증명될 수 있다. 어떤 질의 q가 있다고 하자. q는 $l_1l_2..l_k[l_{k+1}...l_n]$ (k>0, n>k)과 같이 k 개의 라벨들로 구성된 경로와 k+1부터 n까지의 조건 경로로 구성된다고 하자. 이 때, PBQP는 k의 값이나 n 값과 무관하게 분기점까지의 경로, $l_1l_2..l_k$,와 분기점 내의 조건부를 포함한 전체 경로, $l_1l_2..l_kl_{k+1}...l_n$, 2개를 질의 처리를 위한 경로들로 생성한다. 경로가 가장 짧아지는 경우는 한 개의 라벨로만 구성되었을 때이므로, q의 가장 짧은 형태는 k=1, n=2 인 경우이다. 즉, 전체 라벨의 수가 2인 질의이다. 이 때, PBQP와 LBQP는 동일한 입력 셋의 수를 가진다. 하지만, 그 외의 경우, 즉 k>1이거나 n > k+1인 모든 경우에 대해 LBQP는 더 많은 입력 셋을 필요로 하게 된다. 그러므로 PBQP는 항상 LBQP의 최소 조인 횟수만큼만 조인 연산을 수행한다. 한편, 이처럼 어떤 질의가 조건 경로를 포함하는 경우, 그 질의는 $(Pa[Pb])^+$ Pc나 $(Pa[Pb])^+$ 패턴을 띠게 된다. 그러므로 질의를 구성하는 경로의 수는 '분기점의 수*2'(마지막이 조건경로인 경우) 또는 '분기점의 수*2+1'(마지막이 조건경로가 아닌 경우)이 된다.

• 입력 셋의 크기:

위의 조인 횟수 비교에서 두 방식이 동일한 입력 셋을 구성한다고 가정하자. 즉, q가 $l_1[l_2]$ 형식으로 구성되었다고 가정하자. 그리고 l_1에 대해 도달 가능한 경로들을 p_{11}, p_{12}, ..., p_{1r} (r>0), 각 경로들에 대응되는 노드 수를 c_{11}, c_{12}, ..., c_{1r}이라 하고, l_2에 대해 도달 가능한 경로들을 p_{21}, p_{22}, ..., p_{2s} (s>0), 각 경로들에 대응되는 노드 수를 c_{21}, c_{22}, ..., c_{2s}라 하자. 이 때, 두 방식은 서로 다른 입력 셋을 구성한다. LBQP는 l_1과 l_2 각각에 도달 가능한 모든 경로들에 대응되는 모든 노드들을 입력 셋에 포함시킨다. 즉, l_1에 대한 입력 셋의 크기는 $c_{11}+c_{12}+...+c_{1r}$이 되며, l_2에 대한 입력 셋의 크기는 $c_{21}+c_{22}+...+c_{2s}$이 된다. 한편, PBQP는 l_1과 l_1l_2 두 경로에 대한 노드들을 입력 셋으로 구성한다. 만약 l_1의 시작이 '/'이면 루트를 의미하므로, l_1에 대한 경로들 중 루트에 대응되는 한 개 이하의 경로만이 입력 셋의 대상 경로수가 된다. 그러므로 이 경우 l_1에 대한 입력 셋의 크기는 0 또는 c_{1f} (1 <= f <= r)이다. 만약 '//' 으로 시작한다면 이는 임의의 경로들을 의미하므로, LBQP와 동일한 입력 셋 크기를 갖는다. l_1l_2 경로에 대응되는 입력 셋은 l_2의 경로들 중 l_1을 선행 경로로 하는 경로들에 대응되는 노드들로 구성된다. 이러한 경로들은 l_2에 속하는 전체 경로들보다 항상 작거나 같다. 만약 이러한 경로들이 e (<= s) 개 있다면, 이 경로들에 대응되는 노드 수의 합이 입력 셋의 크기가 된다. 이처럼, PBQP에서 생성되는 입력 셋의 크기는 LBQP의 각 입력 셋의 크기에 비해 작거나 최악의 경우 같은 크기를 갖는다. 특히, 질의 내에 부모-자식 관계를 포함한 경우, PBQP는 더 많은 경로들을 입력 셋의 대상에서 제외하므로 더 적은 입력 셋 크기를 가지게 된다.

• 조인 시의 비교 횟수:

조인 연산의 두 입력 셋을 S와 T, 이들의 레코드 수를 m과 n이라 하자. 앞선 분석 결과로부터 우리는 LBQP의 입력 셋이 PBQP의 입력 셋에 비해 클 것이라는 것을 알 수 있다. 하지만, 여기서는 최악의 경우, 두 방식의 입력 셋의 크기가 동일하다고 가정한다.

LBQP와 PBQP의 비교 연산 방식은 노드의 발생 정보를 기반으로 하여 노드 간의 포함 관계를 파악한다는 점에서 동일하다. 하지만, 두 방식은 비교 대상 선정 방식에서 차이점을 가지고 있다. LBQP는 S의 어떤 레코드 s가 T의 어떤 레코드 t와 비교 연산을 했을 때, 비교 조건을 만족하면 두 레코드를 결합한 새로운 결과 레코드를 생성한다. 다음, 두 레코드의 발생 범위를 계산하여 작은 값을 가진 쪽의 다음 레코드(이를 T 쪽에 있는 t1 레코드라 하자)를 읽어온다. 그리고 s와 t1 간의 포함 관계를 계산하여 위와 동일한 과정을 반복한다. 이 방식은 조건을 만족한 레코드 중 한쪽 입력의 레코드가 항상 다음 입력과 비교 연산을 수행한다. 그러므로 두 입력에 대해 최대 m+n 번의 비교 연산을 필요로 한다. 한편, PBQP는 S 의 어떤 레코드 s가 T의 어떤 레코드 t와 비교 연산을 했을 때, 비교 조건을 만족하는 경우, 그 중 한쪽의 레코드가 출력 레코드가 되고, 다른 한 쪽의 레코드는 입력에서 제거된다. 다음 연산을 위해 PBQP는 각 입력 셋에서 다음 레코드들을 추출하고 동일한 방식으로 비교 작업을 수행한다. 그러므로 PBQP는 두 입력에 대해 최대 max{m,n} 회의 비교 연산만을 필요로 한다. 따라서 두 입력의 크기가 동일한 최악의 경우에도 PBQP는 LBQP에 비해 매우 빠른 조인 성능을 제공한다고 할 수 있다.

• 종합:

위의 여러 가지 결과들을 종합하여 보았을 때, PBQP는 질의처리를 위한 조인 횟수나 입력 셋의 크기, 조인

시의 비교 연산에 있어 LBQP에 비해 더 나은 결과를 제공함을 알 수 있다. 특히, 최악의 경우(조인 횟수와 입력 셋의 크기가 같은 경우)에도 비교 횟수에 있어 매우 큰 차이로 조인 연산을 수행하므로, PBQP는 항상 LBQP에 비해 월등히 높은 질의 처리 성능을 보일 것임을 알 수 있다.

5.2 EPIS 질의처리 방식과의 성능 비교

5.2.1 실험 구성

우리는 실험을 위해, Pentium4 1.2G CPU에 256M Bytes 메모리를 가진 PC와 상용 데이타베이스 시스템(MS-SQL Server 2000)을 사용하였다. 본 논문에서 제안한 질의 처리 기법과 관련된 모든 것들은 .NET 환경하에서 C# 언어와 기본 클래스 라이브러리를 이용하여 구현하였다. 또한 단일 경로에 대한 질의에 대해서는 EPIS를 이용하므로, 단일 경로 질의를 SQL 질의로 변환하는 함수를 구현하였다.

실험을 위한 데이타로 Wisconsin에서 제공하는 문서들[6]을 사용하였다. 이 문서들은 한 문단 정도의 텍스트와 간단한 데이타를 복합적으로 포함하고 있으며, 각각 10~40KBytes 정도의 크기를 가진다. 총 220 개 정도의 원본 문서를 사본을 만들어 실험에 사용하였다.

실험 비교 대상으로 선정한 EPIS 기반 질의 처리 기법은 질의에 속한 각 라벨들의 경로 정보를 이용한다. 이 방식은 여러 경로를 포함하는 질의에 대해 이들 각각을 내포 SQL(Nested SQL) 문장으로 변환하는 방식을 사용한다. 그리고 완성된 SQL 문장을 DBMS에 전달하고 이후 모든 처리는 DBMS의 질의 처리기에 위임한다. EPIS와 유사한 다른 질의 처리 기법들[10,11]은 실험 대상에서 제외 되었는데, 이는 EPIS 의 실험 결과에서 EPIS 방식이 다른 유사 기법들보다 높은 성능을 보임을 증명하였기 때문이다.

실험 대상인 두 방식은 한 개의 경로를 가진 질의에 대해서는 동일한 SQL 문장을 실행시키므로, 둘 간의 성능 차이는 거의 없다. 즉, 본 논문에서 제안한 방식은 EPIS에서 제공하는 변환 방식을 사용하여 선형 경로 질의를 처리하므로, 이에 대한 둘 간의 비교는 무의미하다. 둘 간의 성능 차이는 일반적인 트리 형태의 경로 질의에서 나타나게 되므로, 우리는 질의를 구성하는 경로의 수, 즉 조건 경로들의 수와 질의 경로의 길이, 그리고 조작되는 데이타의 양을 고려하여 질의들을 선정하였다. 표 1은 이러한 기준을 반영한 실험 질의들이다. Q5는 조건식이 없는 한 개의 경로만 가진 단순 선형 경로 질의이다. Q4와 Q1은 한 개의 조건 경로식을 가진 질의이며, Q4를 처리하기 위해 조작되는 데이타의 양은 Q1의 경우에 비해 상대적으로 많다. 또한, Q2와 Q3은 두 개의 조건 경로식을 가진 질의로서, Q3에 의해 조작되는 데이타의 양이 Q2에 비해 매우 크다. 우리는 이 질의들을 수행해 봄으로써 조건 경로들의 수가 증가함에 따라, 그리고 조작되는 데이타의 양에 따라 질의 성능이 어떻게 나타나는 지를 파악할 수 있다.

표 1 실험 질의 구성

Q1	//journal[author/lastname='fogh']//bibcode
Q2	//dataset[identifier='I_113A.xml']//revision[date/year='UNKNOWN']//creator
Q3	//fields/field[name]//footnote[footnoteid]/para
Q4	//field[definition]//units
Q5	//reference//journal/name

5.2.2 실험 결과

우리는 XML 문서들을 데이타베이스에 저장하였다. 저장된 XML 문서의 크기를 20M, 40M, 60M로 증가시키면서, 각 질의의 성능을 평가하였다. 각 질의에 대한 수행 시간은 동일한 질의가 데이타베이스 시스템에 수행되지 않는 상태, 즉 해당 질의에 대한 어떠한 캐싱이나 버퍼링이 되어 있지 않은 상태에서 10회 반복 실험한 결과의 평균을 계산하여 결정하였다. 그림 8은 각 데이타베이스 크기에 따라 표 1의 실험 질의들을 수행한 결과 그래프이다. 그림에서 EPIS라는 이름은 EPIS를 기반으로 한 경로 질의 처리 방식을 의미하며, BT는 B-Join과 T-Join을 이용하는 본 논문에서 제안한 경로 질의 처리 방식을 의미한다.

결과 그래프에서, 한 개의 경로식만을 가진 Q5의 경우, 두 방식이 비슷한 성능을 보인다. 그 중 EPIS가 좀 더 빠른 성능을 보이는 데, 그 이유는 다음과 같다. 두 방식이 모두 동일한 한 개의 경로식을 처리하지만, EPIS의 경우는 해당 경로식에 대한 모든 작업을 데이타베이스 시스템 내부에서 처리한다. 한편, BT는 질의 처리를 위한 준비 작업과 마무리 작업을 DBMS 외부에서 수행한다. 그러므로 DBMS와 외부 시스템 간의 연동으로 인한 비용이 추가로 필요하게 되어, 성능이 약간 떨어지게 된다. 하지만, 이 차이는 그리 크지 않으며, 만약 BT 의 질의 처리 부분을 DBMS 내부에 함께 구현한다면, 그 차이 또한 없어질 것이다.

경로식에 포함된 조건의 수와 조작되는 데이타의 크기에 따른 성능 변화는 두 방식 모두 공통된 모습을 보임을 확인하였다. 즉, 조건 경로식을 한 개만 가지고 있는 질의들(Q1과 Q4)은 조건 경로식을 두 개 가지고 있는 질의들(Q2와 Q3)에 비해 더 빠른 성능을 보인다. 또한, 조작되는 데이타의 크기가 큰 질의들(Q4와 Q3)이 조작되는 데이타의 크기가 작은 질의들(Q1과 Q2)에 비해 더 많은 응답 시간을 필요로 한다. 이는 두 방식이 경로 간의 조인 연산을 수행함으로써 질의 처리를 수행

함에 따라, 조건식과 이에 대응되는 데이타의 크기가 커질수록 더 많은 조인과 비교 연산이 필요하기 때문이다.

한편, Q5를 제외한 모든 질의들에 대해서, EPIS를 기반으로 한 질의 처리 방식은 급격한 성능 저하(약 6배에서 290배 정도까지 느린 수행 시간)를 보였다. 특히, Q3의 경우는 EPIS를 기반으로 한 질의 수행을 진행할 수 없을 만큼 느리게 동작하였다(한 시간 이상 걸려도 처리가 되지 않음. 그래프에서는 -1로 표기함). 이는 EPIS 방식으로 수행하기 위해 변환된 SQL 질의가 많은 내포 SQL 문장들과 조인 연산들을 포함하기 때문이다. 또한, 데이타베이스 시스템에서 제공하는 질의 최적화 알고리즘은 범용적인 테이블 방식의 질의 처리에는 효율적이지만, 경로식을 처리하는 데 있어서는 나쁜 질의 실행 계획을 생성하기 때문이다.

반면, 본 연구에서 제안한 BT는 상대적으로 매우 적은 비용으로 질의를 처리하고 있음을 알 수 있다. 이는 본 연구에서 제안한 두 조인 알고리즘과 질의 계획 생성 알고리즘이 매우 효과적으로 질의를 처리함을 증명한다. 특히, 조인 연산의 수행 시간이 입력 데이타들의 크기의 합으로 최적화되어 있다는 점이 질의 수행에서의 성능 향상에 크게 기여했다고 할 수 있다.

6. 결론 및 향후 연구 계획

우리는 본 연구에서 임의의 경로식 질의를 효율적으로 처리하기 위한 새로운 질의 계획 생성 방식과 이에 적용할 수 있는 새로운 물리 조인 연산자들을 제안하였다. 이 방식은, 경로식을 구성하는 각 라벨을 중심으로 조인 연산을 수행하는 기존 방식이나 경로식을 기반으로 질의 처리를 했던 기존 방식들과는 달리, 질의를 구성하는 경로식들 간의 관계를 중심으로 실행 계획을 생성함으로써 질의 경로와 무관한 많은 데이타들을 조인 연산에서 배제시켰으며, 그 결과 조인 성능을 향상 시켰다. 또한, 새롭게 제안한 조인 연산자들은 입력으로 제공되는 데이타들에 대한 한 번의 순차 접근 연산만으로 조인 연산을 수행함으로써 최적화된 조인 성능을 제공하였다.

우리는 여러 분석과 실험을 통해 제안한 방식이 효율적임을 확인하였다. 특히, 관련된 데이타의 크기에 관계없이 복잡하게 구성된 경로식 질의에 대해서, 본 연구에서 제안한 방식은 조인 연산의 수를 줄이고, 효율적인 조인 연산을 지원함으로써 기존 방식에 비해 높은 성능을 보임을 입증하였으며, 향후 추가 작업을 통해 더욱 발전시킬 수 있음을 확인하였다.

다만, 하위 시스템으로 사용되는 경로 기반 역-인덱스의 크기가 매우 커질 수 있어, 이에 대한 질의 수행 가능한 압축 기법을 제공해야할 필요가 있음을 확인하였다. 이에, 우리는 향후에 확장된 질의 처리 기능과 공간 효율성을 제공하기 위한 연구를 진행할 계획이다.

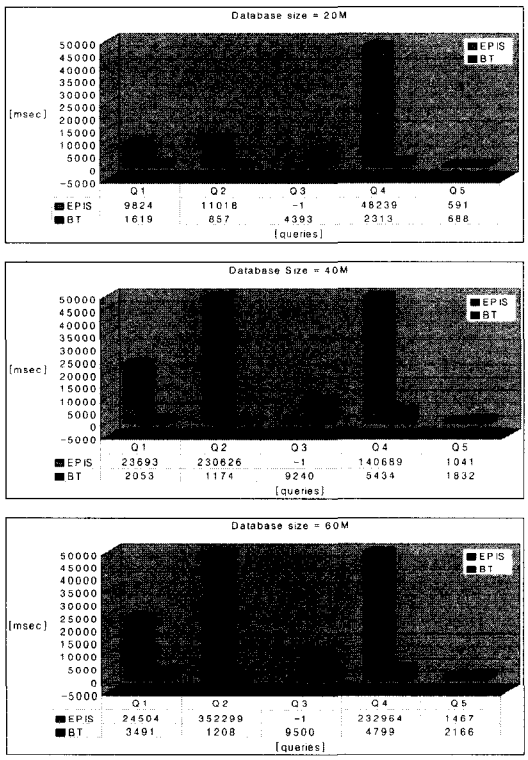


그림 12 실험 결과

참 고 문 헌

[1] Neil Bradley, The XML companion second edition, Addison Wesley, 2000.

[2] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, Dan Suciu, XML-QL, QL, 1998.

[3] Hiroshi Ishikawa, Kazumi Kubota, Yasuhiko Kanemasa, XQL: A Query Language for XML Data, QL, 1998.

[4] Donald D. Chamberlin, Jonathan Robie, Daniela Florescu, Quilt: An XML Query Language for Heterogeneous Data Sources, WebDB, pp. 1-25, 2000.

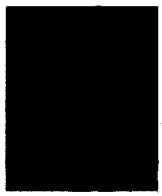
[5] XQuery 1.0: An XML Query Language W3C Working Draft, http://www.w3.org/TR/xquery/, 2002.

[6] Wisconsin XML Data Set, http://www.cs.wisc.edu/niagara/data.html

[7] Chun Zhang, Jeffery Nahgton, David DeWitt, Qiong Luo, and Guy Lohman, On Supporting Containment Queries in Relational Database Management Systems, SIGMOD, pp. 425-436, May 2001.

[8] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, and Yuqing Wu, Structural Joins : A Primitive for Efficient XML Query Pattern Matching, ICDE, pp. 141-153, February 2002.
[9] Shu-Yao Chien, Zografoula Vagena, Donghui Zhang, Vassilis J. Tsotras, and Carlo Zaniolo, Efficient Structural Joins on Indexed XML Documents, VLDB, pp. 263-274, August 2002.
[10] Masatoshi Yoshikawa and Toshiyuki Amagasa, XRel : A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases, ACM TOIT, Volume 1, Number 1, pp. 110-141, August 2001.
[11] Chiyoung Seo, Sang-won Lee, and Hyoung-Joo Kim, An Efficient Inverted Index Technique for XML Documents using RDBMS. Information and Software Technology (Elsevier Science), Volume 45, Issue 1, pp. 11-22, January 2003.
[12] Kyung-Sub Min, Hyoung-Joo Kim, An RDBMS-based Inverted Index Technique for Efficient Support of Processing Path Queries on XML Documents with Different Structures, JKISS, Volumn 30, Number 4, pp. 420-428, 2003.
[13] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, Jennifer Widom, Lore: A Database Management System for Semistructured Data, SIGMOD Record Volume 26, Number 3, pp. 54-66, 1997.
[14] Roy Goldman and Jennifer Widom, DataGuides : Enabling Query Formulation and Optimization in Semistructured Databases, VLDB, pp. 436-445, August 1997.
[15] J. McHugh and J. Widom, Query Optimization for XML, VLDB, September 1999.
[16] Quanzhong Li and Bongki Moon, Indexing and Querying XML Data for Regular Path Expression, VLDB, pp. 361-370, September 2001.
[17] Haifeng Jiang, Wei Wang, and Hongjun Lu, Holistic Twig Joins on Indexed XML Documents, VLDB, pp. 273-284, September, 2003.

민 경 섭
1995년 한국항공대학교 전자계산학과 학사 졸업. 1997년 서울대학교 컴퓨터공학과 석사 졸업. 1999년 서울대학교 인지과학과 박사 수료. 관심분야는 데이타베이스, XML, 지식 검색



김 형 주
1982년 서울대학교 전산학과(학사). 1985년 미국 텍사스 대학교 대학원 전산학(석사). 1988년 미국 텍사스 대학교 대학원 전산학(박사). 1988년 5월~1988년 9월 미국 텍사스 대학교 POST-DOC 1988년 9월~1990년 12월 미국 조지아 공과대학 조교수. 1991년~현재 서울대학교 컴퓨터공학부 교수