XML 실체뷰를 이용한 XML 질의 처리 : 경로 표현식의 분할 처리 및 결과 통합

문 찬 호[†] · 강 현 철[††]

요 약

웹 상에서 XML 문서의 수요가 증가함에 따라 XML 문서를 자원으로 갖는 웹 서비스 응용들이 증가하고 있다. 이들 웹 서비스 응용에서 질의 처리 시 성능 향상을 위해 XML 데이타에 대한 뷰 메커니즘을 고려할 수 있다. XML 질의 결과를 실체뷰로 유지하고, 하부 XML 문서에 대한 원래의 질의를 관련된 실체뷰에 대한 질의로 변환하여 수행한다면 질의 응답시간을 줄일 수 있을 것이다. 실체뷰를 이용한 질의 처리의 유형으로는 ① 실체뷰로부터 원하는 결과를 모두 얻을 수 있는 유형과 ② 질의 결과의 일부는 실체뷰에 존재하지만 일부는 하부 XML 문서로부터 검색해야 하는 유형이 있다. 본 논문에서는 두 번째 유형에 대하여 연구한다. 본 논문의 질의는 여러 XML 질의어들이 갖는 핵심적인 기능 중 하나인 XML 경로 표현식이다. 본 논문에서는 먼저, XML 저장소 내 하부 XML 문서들로부터 도출된 XML 실체뷰의 저장 구조를 제시한 후, 주어진 XML 질의를 실체뷰에 대한 질의와 하부 데이타에 대한 질의로 분할하는 알고리즘과 분할 질의의 결과를 통합하는 알고리즘을 제시한다. 그리고, 성능 평가를 통하여 실체뷰를 이용한 질의 분할 처리가 성능 향상을 가져오는 조건을 구한다.

Processing XML Queries Using XML Materialized Views : Decomposition of a Path Expression and Result Integration

ChanHo Moon[†] · Hyunchul Kang[††]

ABSTRACT

As demand of XML documents in the Web increases, Web service applications that manage XML documents as their resource are increasing. The view mechanism for XML data could be considered for effective XML query processing in these Web service applications. If the XML query results are maintained as XML materialized views and their relevant XML query is processed using them, the query response time could be reduced. There are two types of processing an XML path expression, which is one of the core features of XML query languages, using XML materialized views. One is the type where the complete query result is obtained from the materialized view, and the other is the type where some of the result is obtained from the materialized view and the rest is from the underlying XML documents. In this paper, we investigate the second type. An XML query in this paper is an XML path expression which is one of the core features of XML query languages. We first describe the storage structures of the XML materialized views derived from the underlying XML documents in the XML repository. Then, we propose the algorithms to decompose a given XML query into the subquery against the materialized view and the subquery against the underlying XML documents, and to integrate the results of these subqueries. Through performance evaluation, we figure out the condition under which our XML query decomposition using materialized views is more effective than the conventional processing.

키워드 : XML, 경로 표현식(Path Expression), 질의 분할(Query Decomposition), 결과 통합(Result Integration), 실체뷰(Materialized View)

1. 서 론

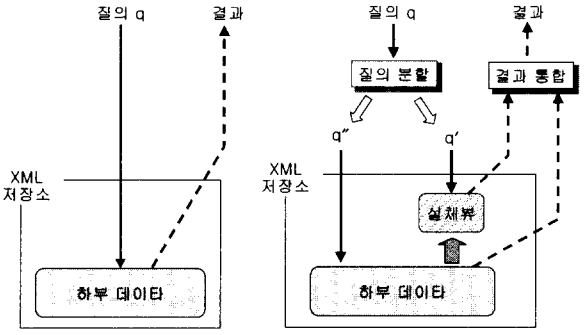
차세대 웹 문서를 위한 표준으로 XML이 등장함에 따라 향후 웹 상에는 수많은 XML 문서가 존재할 것이다. 그리고 이들은 디지털 도서관(digital library), 전자 상거래에서의 전자 카탈로그(e-catalog), 기업 내 업무를 통합 관리하는 전사적 자원 관리(ERP) 시스템 등 다양한 웹 서비스 응용에서 가장 중요한 웹 자원이 될 것이다. 이들 웹 서비스 응용에서는 질의 처리 시 성능 향상을 위해 XML 데이타에 대한 뷰 메커니즘의 도입을 고려할 수 있다.

뷰는 관계형 데이타베이스의 경우 테이블이나 다른 뷰에서 유도된 테이블로서 참조되어질 때마다 재생성되는 가상의 테이블이며, 일반적으로 다양한 데이타 저장소(repository)에 저장된 이질적인 데이타의 통합과 데이타 여과 기능을 제공한다. 뷰는 질의 처리 시 성능 향상을 위해 실체뷰(materialized view)로 유지할 수 있는데, 뷰가 실체뷰로 제공되면 관련된 질의의 효율적인 처리에 이용될 수도 있다[1]. 객체 지향형 데이타베이스 시스템에서도 뷰 개념에 관한 중요도를 인식하여, 객체 지향형 뷰에 대하여 많은 연구가 이루어져 왔다[2, 3]. 웹 데이타에서 많은 비중을 차지하고 있는 반구조적 데이타(semistructured data) 및 XML 데이

※ 본 연구는 한국과학재단 목적기초연구(R01-2000-000-00272-0)지원으로 수행되었음.
† 준 회 원 : 중앙대학교 대학원 컴퓨터공학과
†† 종신회원 : 중앙대학교 컴퓨터공학과 교수
논문접수 : 2002년 10월 19일, 심사완료 : 2003년 5월 6일

타에 대해서도 뷰는 유용한 개념이며[4,5] 실체뷰에 대한 연구도 수행되고 있다. [6-8]에서는 반구조적 데이타에 대한 실체뷰 및 그의 점진적 갱신에 관한 연구가 수행되었다. [9]에서는 관계형 데이타베이스의 XML 출판 결과를 실체화하는 연구가 수행되었다. [10]에서는 XML 질의 결과를 실체화하고 점진적 갱신으로 유지하는 XML 실체뷰 관리 프레임워크를 제안하였다.

(그림 1)은 다수의 XML 문서(하부 데이타)를 저장하고 있는 XML 저장소를 대상으로 질의 q의 처리 과정을 나타낸 것이다. (그림 1)(a)는 실체뷰를 제공하지 않는 기존의 XML 저장소를 대상으로 한 질의 처리를 나타낸 것이다. (그림 1)(b)는 XML 저장소 내 실체뷰를 이용한 질의 처리를 나타낸 것인데, 질의 q가 주어졌을 때, 질의 결과의 일부를 관련된 실체뷰로부터 얻을 수 있다면, **질의 분할(decomposition)**을 통해 질의 q를 실체뷰에 대한 질의 q′과 하부 데이타로부터 나머지 결과를 얻는 질의 q″으로 변환하여 수행한 후 이들 질의 **결과를 통합(result integration)**함으로써 방대한 양의 XML 저장소 내 문서들에 대한 검색을 피할 수 있어 질의 응답 시간을 줄일 수 있다[11].



(a) 실체뷰를 이용하지 않는 질의 처리 (b) 실체뷰를 이용한 질의 처리

(그림 1) XML 저장소를 대상으로 한 질의 처리

본 논문에서는 XML 실체뷰를 이용한 질의 처리에 관한 것으로, 주어진 질의를 실체뷰에 대한 질의와 하부 데이타에 대한 질의로 분할하여 처리한 후 이들 결과를 통합하는 기법에 대해 연구한다. 이를 위해 먼저 XML 저장소 내 하부 XML 데이타로부터 도출된 XML 실체뷰를 정의하고, 실체뷰를 지원하는 XML 저장소의 구조를 제시한다. 주어진 질의를 ① 실체뷰에 대한 질의와 ② 실체뷰에 포함되지 않은 내용을 직접 하부 데이타로부터 검색하는 질의로 분할한 후 이들을 처리하여 그 결과를 통합하는(이하, **실체뷰를 이용한 질의 처리 방법**) 질의 분할 알고리즘과 분할된 질의 결과 통합 알고리즘을 제시한다. 그리고 실체뷰를 이용한 질의 처리 방법과 **실체뷰를 이용하지 않는 질의 처리 방법**의 성능을 비교, 분석하여 실체뷰를 이용한 질의 처리 방법이 성능 향상을 가져오는 조건을 구한다. 이때 하부 데이타에 대한 질의 처리에서는 XML 인덱스의 사용을 고려하였다. 본 논문에서 사용된 XML 질의는 XQuery[12], XPath[13], XQL[14]과 같은 XML 질의어들이 갖는 핵심적인 기능 중 하나인 XML 경로 표현식이다.

본 논문의 구성은 다음과 같다. 2절에서는 관련 연구를 기술하며, 3절에서는 XML 실체뷰의 정의와 실체뷰를 지원하는 XML 저장소의 구조에 대해 기술한다. 4절에서는 질의 분할과 분할된 질의의 처리에 대해서 기술한다. 5절에서는 결과 통합에 대해 기술하고, 6절에서는 성능 평가 결과를 기술한다. 마지막으로 7절에서 결론을 맺는다.

2. 관련 연구

효율적인 XML 질의 처리를 위한 기법으로는 실체뷰를 이용한 것과 인덱싱을 이용하는 것으로 대별할 수 있다. XML 실체뷰를 이용한 질의 처리에 관한 연구로는, 실체뷰를 이용한 XQL 질의 변환 연구[11]가 있으나 다양한 질의 유형에 대한 변환을 다루지 않고 있다. 그 외 XML 실체뷰를 이용한 질의 처리와 관련된 연구로는 반구조적 데이타에 대한 뷰를 이용한 질의 변환(query rewriting)[15], 질의 응답[16], 그리고 질의 포함(query containment)[17] 문제에 관한 연구가 있으나, 이들 연구들은 모두 그래프 기반의 반구조적 데이타베이스로부터 생성된 뷰를 이용한 반구조적 질의 처리 문제를 다룬 것으로, 뷰의 소스로 XML 문서를 그리고 뷰의 정의 및 질의 표현을 위한 질의어로 XQL과 같은 XML 질의어를 대상으로 하지 않았다.

[11]에서는 XML 저장소에 저장된 하부 XML 문서들로부터 XQL을 통해 정의된 실체뷰가 지원될 경우 이를 이용한 XQL 질의 처리에 대해 기술하고 있다. 질의 처리시 실체뷰를 이용하기 위해, [11]에서는 먼저 주어진 질의에 대한 결과와 실체뷰 내용간의 포함 관계를 분류하고 있다. 이는 이들 둘 간의 포함 관계에 따라 질의 처리 방법이 달라질 수 있기 때문이다. [11]에서는 실체뷰와 질의 결과간의 포함 관계를 다섯 가지로 분류하고 있다. 이를 정리하면 다음과 같다.

i) 실체뷰와 질의 결과가 일치하는 경우
ii) 실체뷰가 질의 결과를 포함하는 경우
iii) 질의 결과가 실체뷰를 포함하는 경우
iv) 실체뷰와 질의 결과가 일부 중복되는 경우
v) 실체뷰와 질의 결과 사이에 포함 관계가 없는 경우

실체뷰와 질의 결과간의 포함 관계를 기반으로 하여, [11]에서는 질의 처리시 실체뷰의 활용 유형을 세 가지로 분류하고 있다. 이를 정리하면 다음과 같다.

- 유형 **MV**(Materialized View Only)
  : 주어진 질의의 결과를 실체뷰로부터 모두 얻는 경우
- 유형 **UD+MV**(Both Underlying Documents and Ma-terialized View)
  : 주어진 질의 결과의 일부는 실체뷰로부터 얻고 나머지 일부는 하부 데이타로부터 얻는 경우
- 유형 **UD**(Underlying Documents Only)

: 주어진 질의의 결과를 모두 하부 데이타로부터 얻는 경우

[11]에서는 실체뷰를 이용한 질의 처리 유형 중 유형 MV에 대해서만 주어진 질의를 실체뷰에 대한 질의로 변환하는 알고리즘을 제시였다. 유형 MV는 유형 UD+MV에 비해 훨씬 단순한 질의 변환을 요하며 질의 결과의 통합도 필요없다. 그리고, [11]에서는 실체뷰를 이용한 질의 변환에 대한 성능 분석이 제시되어 있지 않다.

한편, XML 문서에 대해 내용 기반 질의, 구조 기반 질의, 속성(attribute) 기반 질의가 가능함에 따라 이들을 신속히 처리하기 위한 XML 인덱싱에 관한 연구들이 활발히 진행중이다[18, 19]. 그 중 XML 문서의 검색을 위해 해당 문서의 구조적인 요약과 문서의 노드(엘리먼트) 인덱스를 모두 활용하는 방법은 [18]에서 제시하고 있다. 즉, 반구조적 데이타를 저장한 데이타베이스에서 노드들 간의 경로 표현식을 통한 인덱싱 방안을 제시하고 있다. 이를 위해 세 가지 인덱스 유형을 제시하고 있는데, 루트 노드로부터 특정 노드로의 전체 경로명을 활용한 검색 방법인 1-index, 데이타베이스 내 임의의 두 노드들간의 경로명을 활용한 검색 방법인 2-index, 그리고 여러 노드들간의 경로명을 활용한 검색 방법인 T-index가 이들이다.

3. XML 실체뷰를 지원하는 XML 저장소의 구조

본 절에서는 XML 저장소를 대상으로 한 XML 질의 모델을 제시하고, XML 질의를 통한 XML 실체뷰를 정의한다. 그리고, 실체뷰를 지원하는 XML 저장소의 구조에 대해 기술한다.

3.1 XML 질의 모델

본 논문의 XML 실체뷰 내용은 XML 질의 결과에 해당된다. 본 논문에서 사용된 XML 경로 표현식은 화일의 디렉토리 경로를 나타내는 것과 같이 **경로**로 질의 대상이 되는 XML 문서의 **목적 엘리먼트(target element)**를 표시하고, **조건**을 표시하는 필터 연산자, 비교 연산자 등을 이용하여 **조건 엘리먼트(condition element)**를 표현한다. 본 논문에서는 XML 문서 내 특정 엘리먼트의 경로 차이를 **레벨**로 정의하고, 루트 엘리먼트에서 최하위 엘리먼트 간의 레벨 차이를 **높이**라고 정의한다.

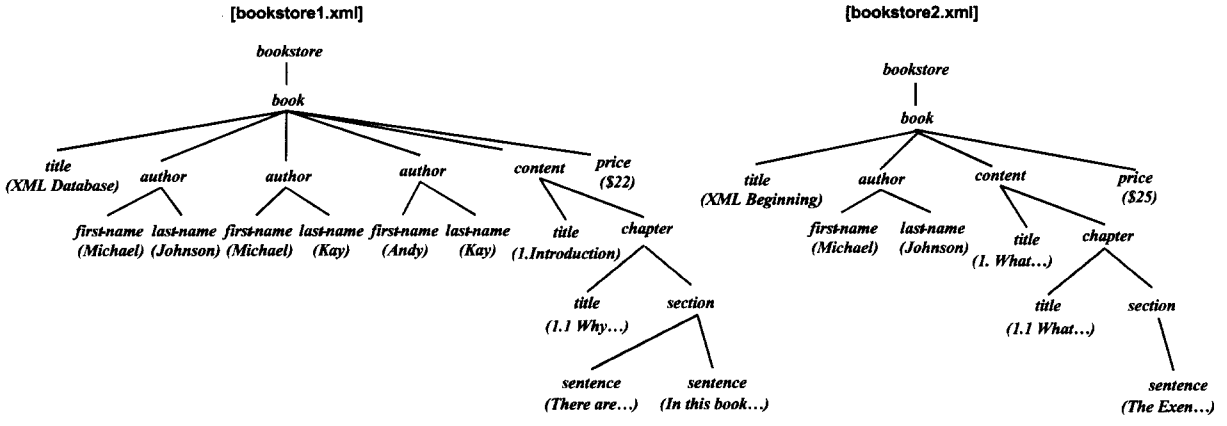
(그림 2)는 하부 데이타에 저장된 XML 문서를 나타낸 것으로, 하부 데이타에 bookstore1.xml과 bookstore2.xml이 저장되어 있다고 가정하자. XML 문서를 트리 구조로 표현하여 노드는 엘리먼트, 그리고 괄호 안의 내용은 값(content)을 나타낸다. 하부 데이타를 대상으로 XML 질의 "/bookstore/book/author[first-name="Michael"]"이 주어졌을 때, 경로는 "/bookstore/book/author"이며, 조건은 "first-name="Michael""이며, 결과 엘리먼트는 "author"이며, 조건 엘리먼트는 "first-name"이 되고, 결과 엘리먼트에서 조건 엘리먼트의 레벨은 1이 된다. (그림 2)의 bookstore1.xml 문서의 높이는 6이 된다.

결과 엘리먼트(result element)는 추출된 목적 엘리먼트와 그의 서브 트리(즉, 서브 엘리먼트들 모두)로 구성된다. 본 논문에서 고려한 XML 경로 표현식은 다음을 만족한다고 가정하였다.

- 조건을 명시하기 위한 필터 연산은 한번만 사용한다.
- 필터 연산자 내의 조건이 사용하는 비교 연산자('=', '!=', 'text()', 등)에는 제한이 없으나 불린 연산자에 대해서는 'and'로 제한한다(즉, 'or' 사용을 제한한다).
- 필터 연산 내에서 and 연산 con1 and con2가 주어졌을 경우(con1과 con2는 모두 비교 연산을 이용한 조건), con1의 결과와 con2의 결과는 겹치지 않는다.
- 결과 엘리먼트는 하나이다. 단, 그 결과의 서브 엘리먼트는 모두 함께 추출된다.

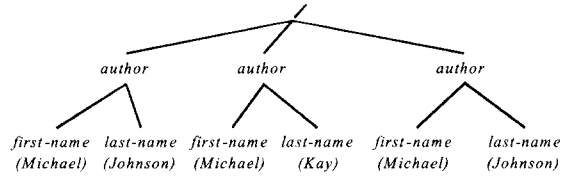
3.2 XML 실체뷰

XML 실체뷰는 뷰 정의(XML 질의의 결과)에 의해 하부 XML 문서로부터 검색된 결과로서, 검색된 결과 엘리먼트들이 XML 실체뷰를 구성한다. XML 실체뷰는 질의 조건



(그림 2) 하부 데이타에 저장된 XML 문서의 예

을 만족하는 각 XML 문서로부터 검색된 결과 엘리먼트들을 하나의 루트 엘리먼트로 묶은 XML 문서로 표현된다. 이 문서를 **XML 실체뷰 문서**라 한다. XML 실체뷰를 XML 실체뷰 문서로 쉽게 표현하기 위해서 XML 구조 형식을 갖는 템플릿을 활용한다. (그림 3)은 (그림 2)와 같은 하부 XML 데이타를 대상으로 *bookstore/book/author[first-name = "Michael"]*라는 경로 표현식을 처리한 결과를 뷰로 나타낸 것이다. (그림 3)(a)는 XML 실체뷰를 트리 형태로 표현한 것이고, (그림 3)(b)는 이를 XML 문서로 표현한 XML 실체뷰 문서이다.



(a) 트리 형태로 표현된 XML 실체뷰

```
<root>
    <author>
        <first-name> Michael </first-name>
        <last-name> Johnson </last-name>
    </author>
    <author>
        <first-name> Michael </first-name>
        <last-name> Kay </last-name>
    </author>
    <author>
        <first-name> Michael </first-name>
        <last-name> Johnson </last-name>
    </author>
</root>
```

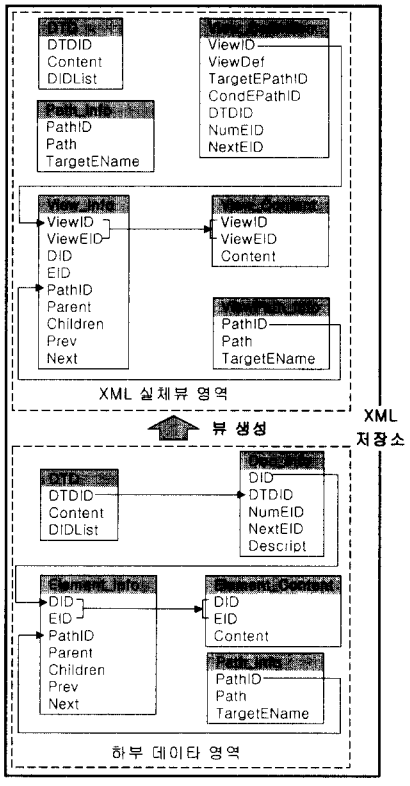
(b) XML 실체뷰 문서

(그림 3) XML 실체뷰의 예

XML 실체뷰는 그의 소스인 하부 XML 데이타와 마찬가지로 XML 저장소 내에 XML 문서로 저장된다. 사용자 또는 응용 프로그램은 반복적으로 자주 제기되는 경로 표현식 결과를 뷰로 정의하여 실체뷰로 유지할 수 있다. 실체뷰는 뷰의 내용을 실제로 저장하는 기법으로서, 사용자의 질의에 빠른 응답 시간을 제공한다. 그러나 실체뷰는 하부 데이타가 변경되었을 경우, 일관성을 유지해야 하는 오버헤드가 따른다. 일관성을 유지하는 방법으로는, 뷰를 하부 데이타로부터 재생성(recomputation)하는 기법과 변경 내용만을 뷰에 반영하는 점진적 갱신(incremental refresh) 기법이 있다. 이들 기법에 대한 자세한 사항은 본 논문의 범위 밖이므로 이하 더 다루지 않으며 XML 실체뷰의 갱신 기법[10]과 반구조적 데이타에 대한 실체뷰의 갱신 기법[6-8]을 참조한다.

3.3 XML 저장소 구조

(그림 4)는 XML 저장소의 구조를 테이블 별 스키마로 나타낸 것이다. 실선의 사각형은 테이블을, 음영 영역의 볼드체 활자는 테이블 명을, 바탕체 활자는 컬럼 명을 나타내고, 화살표(→)는 테이블의 컬럼 간 외래키의 참조 관계를 나타낸다. XML 저장소는 객체 관계형 데이타베이스를 기반으로 하여 (그림 4)의 점선 사각형으로 표시한 것과 같이 크게 **하부 데이타 영역**과 **XML 실체뷰 영역**으로 나누어진다. 하부 데이타 영역은 XML 문서들을 엘리먼트 단위로 나누어 XML 문서의 구조 정보와 내용 정보를 저장한 Element_Info 테이블과 Element_Content 테이블, XML 문서의 기타 정보를 저장한 Doc_Info 테이블, DTD 정보를 저장한 DTD 테이블과 경로 정보를 저장한 Path_Info 테이블, 그리고 XML 문서 검색을 위한 인덱스로 구성된다. XML 실체뷰 영역 또한 XML 실체뷰 정의로부터 얻어진 XML 실체뷰를 엘리먼트 단위로 나누어 뷰의 구조 정보와 내용 정보를 저장한 View_Info 테이블과 View_Content 테이블, 뷰 정의를 저장하는 View_Definition 테이블과 View의 경로 정보를 저장한 ViewPath_Info 테이블, 그리고 하부 데이타 영역의 Path_Info 테이블과 DTD 테이블로 구성된다(각 테이블의 컬럼에 대해서는 이후 각 테이블의 내용 및 기능 설명에서 기술한다).



(그림 4) XML 저장소의 테이블 스키마

이때, 하부 데이타 영역의 인덱스는 Element_Info 테이블 중 특정 경로의 일부 엘리먼트들의 위치에 대한 인덱싱 정보(본 논문에서는 [18]에서의 1-index 방법을 이용하여 하부 데이타의 인덱스를 표현함)만을 저장한다. 이는 하부 데이타 전체에 대한 인덱싱 정보를 유지하기 위한 저장 공간과 유지비용이 크기 때문이다.

3.3.1 하부 데이타 영역의 구조

XML 문서는 구조적 특성을 갖고 있기 때문에 파서를 이용하여 엘리먼트 단위로 나눈 뒤, 트리 형식으로 저장하는 것이 일반적이다. [20]에서는 구조적 문서인 XML 문서를 객체 관계형 데이타베이스에 저장하는 방법을 제시하고 있다. 저장된 XML 문서에 대한 엘리먼트 검색뿐만이 아니라 구조 정보를 이용한 항해(navigation)를 지원하기 위해 각 엘리먼트간의 구조 정보를 엘리먼트 내용과 분리하여 저장하고, 루트 엘리먼트에서 해당 엘리먼트까지의 경로를 따로 저장하여 구조 기반 질의를 가능하게 하고 있다. [21, 22]에서는 구조적 문서인 XML 문서를 관계형 데이타베이스에 저장하는 방법을 제시하고 있다. 이들은 XML 문서가 참조하는 DTD를 기준으로 DTD를 구성하는 엘리먼트들의 상하관계, 순차(,) 선택(|), 반복(+) 등을 고려하여 DTD 그래프를 작성한 후, DTD 그래프 내 엘리먼트들을 그룹화하여 이들을 각각 테이블에 저장하는 방법을 제시하고 있다. 본 논문에서는 하부 XML 문서를 저장하기 위해 [20]에서 제시한 내용 정보와 구조 정보를 분할하여 서로 다른 테이블에 저장하고 루트 엘리먼트에서 해당 엘리먼트까지의 경로를 따로 저장하는 구조를 선택하였다. 다음은 하부 데이타 영역을 구성하는 테이블에 대한 설명이다.

가) Element_Info 테이블과 Element_Content 테이블

본 논문의 하부 XML 문서들은 해당 DTD를 준수하는 유효(valid) XML 문서들이다. 이들을 저장하기 위해 먼저 엘리먼트 단위로 나누어 XML 문서의 구조 정보는 DID, EID, PathID, Parent, Children, Prev, Next 컬럼으로 구성된 Element_Info 테이블과 XML 문서의 내용 정보는 DID, EID, Content 컬럼으로 구성된 Element_Content 테이블에 각각 저장된다.DID에는 XML 문서의 ID가, EID에는 엘리먼트의 ID(저장되는 엘리먼트 순으로 번호 부여)가, PathID에는 엘리먼트의 경로 정보(이후 Path_Info 테이블에서 자세히 기술) ID가, Parent에는 부모 엘리먼트의 ID가, Children에는 자식 엘리먼트 ID(들)가, Prev/Next에는 같은 부모 엘리먼트를 갖는 이전/이후 형제 엘리먼트의 ID가, 그리고 Content에는 엘리먼트의 내용 정보가 저장된다. 이때, Children 컬럼은 객체 관계형 데이타베이스에서 제공하는 콜렉션(collection) 타입으로 정의된다.

(그림 5)와 같은 bookstore DTD가 있고, 이를 준수하는 XML 문서들((그림 2)의 bookstore1.xml과 bookstore2.xml)이 있다고 가정하자. (그림 6)은 이들 XML 문서를 Element_Info 테이블과 Element_Content 테이블에 저장한 모습이다.

```
<!ELEMENT bookstore      (book) >
<!ELEMENT book           (title, author +, content, price) >
<!ELEMENT title          (# PCDATA)>
<!ELEMENT author         (first-name, last-name)>
<!ELEMENT first-name     (# PCDATA)>
<!ELEMENT last-name      (# PCDATA)>
<!ELEMENT content        (title, chapter) +>
<!ELEMENT chapter        (title, section) +>
<!ELEMENT section        (sentence +) >
<!ELEMENT sentence       (# PCDATA)* >
<!ELEMENT price          (# PCDATA)* >
```

(그림 5) DTD의 예

DID	EID	PathID	Parent	Children	Prev	Next
1	1	1	NULL	2	NULL	NULL
1	2	2	1	3, 4, 7, 10, 13, 20	NULL	NULL
1	3	3	2	NULL	NULL	4
1	4	4	2	5, 6	3	7
1	5	5	4	NULL	NULL	6
1	6	6	4	NULL	5	NULL
1	7	4	2	8, 9	4	10
1	8	5	7	NULL	NULL	9
1	9	6	7	NULL	8	NULL
1	10	4	2	11, 12	7	13
1	11	5	10	NULL	NULL	12
1	12	6	10	NULL	11	NULL
1	13	7	2	14, 15	10	20
1	14	8	13	NULL	NULL	15
1	15	9	13	16, 17	14	NULL
1	16	10	15	NULL	NULL	17
1	17	11	15	18, 19	16	NULL
1	18	12	17	NULL	NULL	19
1	19	12	17	NULL	18	NULL
1	20	13	2	NULL	13	NULL
2	1	1	NULL	2	NULL	NULL
2	2	2	1	2, 4, 7, 13	NULL	NULL
2	3	3	2	NULL	NULL	4
2	4	4	2	5, 6	3	7
2	5	5	4	NULL	NULL	6
2	6	6	4	NULL	5	NULL
2	7	7	2	8, 9	4	13
2	8	8	7	NULL	NULL	9
2	9	9	7	10, 11	8	NULL
2	10	10	9	NULL	NULL	11
2	11	11	9	12	10	NULL
2	12	12	11	NULL	NULL	NULL
2	13	13	2	NULL	7	NULL

DID	EID	Content
1	1	/
1	2	-
1	3	XML Database
1	4	-
1	5	Michael
1	6	Johnson
1	7	-
1	8	Michael
1	9	Kay
1	10	-
1	11	Andy
1	12	Kay
1	13	-
1	14	1. Introduction
1	15	-
1	16	1.1 Why XML and Databases
1	17	-
1	18	There are may reasons why we might wish to ...
1	19	In this book, we'll see how XML may...
1	20	$22
2	1	/
2	2	-
2	3	XML Beginning
2	4	-
2	5	Michael
2	6	Johnson
2	7	-
2	8	1. What XML is?
2	9	-
2	10	1.1 What Does This Book Cover?
2	11	-
2	12	The Extensible Markup Language(XML) ...
2	13	$25

(a) Element_Info 테이블 (b) Element_Content 테이블

(그림 6) Element_Info 테이블과 Element_Content 테이블의 구조 및 예

나) Doc_Info 테이블

Doc_Info 테이블에는 해당 XML 문서가 참조하는 DTD의 ID, XML 문서를 구성하는 엘리먼트의 개수(NumEID), 엘리먼트 삽입시 할당될 EID 정보(NextEID), 그리고 XML 문서의 설명 정보(Descript)들이 저장된다. (그림 7)은 Doc_Info 테이블의 모습을 나타낸 것이다.

DID	DTDID	NumEID	NextEID	Descript
1	1	20	21	bookstore1.xml
2	1	13	14	bookstore2.xml

(그림 7) Doc_Info 테이블의 구조 및 예

다) DTD 테이블

DTD 테이블은 DTDID, Content, DIDList 컬럼으로 구성된다. Content에는 DTD 내용이, DIDList에는 DTD를 참조하는 XML 문서(들)의 DID가 저장된다. XML 문서들에 대한 구조 기반 질의를 지원하기 위해 본 논문에서는 DTD에 의해 생성될 수 있는 경로 정보를 Path_Info 테이블에 저장하여 질의 검색시 이용한다. Path_Info 테이블은 PathID와 Path, 그리고 TargetEName 컬럼으로 구성된다. Path에는 경로 정보가 저장되며, TargetEName에는 해당 경로의 목적 엘리먼트 명이 저장된다. (그림 8)은 DTD 테이블과 Path_Info 테이블의 모습을 나타낸 것이다.

DTDID	Content	DIDList
1	<!ELEMENT bookstore (book)> <!ELEMENT bookstore (title, ···	{1, 2}

(a) DTD 테이블

PathID	Path	TargetEName
1	/bookstore	bookstore
2	/bookstore/book	book
3	/bookstore/book/title	title
4	/bookstore/book/author	author
5	/bookstore/book/author/first-name	first-name
6	/bookstore/book/author/last-name	last-name
7	/bookstore/book/content	content
8	/bookstore/book/content/title	title
9	/bookstore/book/content/chapter	chapter
10	/bookstore/book/content/chapter/title	title
11	/bookstore/book/content/chapter/section	section
12	/bookstore/book/content/chapter/section/sentence	sentence
13	/bookstore/book/price	price

(b) Path_Info 테이블

(그림 8) DTD 테이블과 Path_Info 테이블의 구조 및 예

3.3.2 XML 실체뷰 영역의 구조

가) View_Definition 테이블

View_Definition 테이블은 ViewID, ViewDef, TargetEPathID, CondEPathID, DTDID, NumEID, 그리고 NextEID 컬럼으로 구성된다. ViewID에는 뷰의 ID가, ViewDef에는 뷰의 생성 조건이, TargetEPathID에는 목적 엘리먼트의 PathID가, CondEPathID에는 조건 엘리먼트의 PathID(들)가, DTDID에는 뷰가 참조하는 DTD의 ID가 저장된다. ViewDef의 내용은 XML 경로 표현식이 된다. (그림 9)는 View_Definition 테이블의 구조와 그 예를 나타낸 것으로 (그림 6)의 Element_Info 테이블과 Element_Content 테이블에 저장된 XML 문서를 대상으로 XML 질의 "/bookstore/book/author[first-name = "Michael"]"를 수행한 결과가 뷰로 정의되었음을 나타내고 있다.

나) View_Info 테이블, View_Content 테이블, 그리고 ViewPath_Info 테이블

ViewID	ViewEID	DID	EID	PathID	Parent	Children	Prev	Next
1	1	-	-	1	NULL	2, 5, 8	NULL	NULL
1	2	1	2	2	1	3, 4	NULL	5
1	3	1	3	3	2	NULL	NULL	4
1	4	1	4	4	3	NULL	3	NULL
1	5	1	5	2	1	6,7	2	8
1	6	1	6	3	5	NULL	NULL	7
1	7	1	7	4	5	NULL	6	NULL
1	8	2	8	2	1	9, 10	5	NULL
1	9	2	9	3	8	NULL	NULL	10
1	10	2	10	4	8	NULL	9	NULL

(a) View_Info 테이블

ViewID	ViewEID	Content
1	1	/
1	2	-
1	3	Michael
1	4	Johnson
1	5	
1	6	Michael
1	7	Kay
1	8	-
1	9	Michael
1	10	Johnson

(b) View_Content 테이블

PathID	Path	TargetEName
1	/root	root
2	/root/author	
3	/root/author/first-name	Michael
4	/root/author/last-name	Johnson

(c) ViewPath_Info 테이블

(그림 10) View_Info 테이블, View_Content 테이블, 그리고 ViewPath_Info 테이블의 구조 및 예

ViewID	ViewDef	TargetEPathID	CondEPathID	DTDID	NumEID	NextEID
1	/bookstore/book/anthor/[first-name = "Michael"]	4	{5}	1	10	11

(그림 9) View_Definition 테이블의 구조 및 예

(그림 9)의 뷰 정의에 의해 생성된 XML 실체뷰는 엘리먼트 단위로 나뉘어 View_Info 테이블과 View_Content 테이블에 저장된다. View_Info 테이블은 ViewID, ViewEID, DID, EID, PathID, Parent, Children, Prev, Next 컬럼으로 구성된다. ViewID에는 뷰의 ID가, ViewEID는 뷰의 엘리먼트의 ID가, DID와 EID에는 뷰의 엘리먼트와 매핑되는 하부 XML 문서 엘리먼트의 DID와 EID가 저장된다. View_Content 테이블은 ViewID, ViewEID, Content 컬럼으로 구성된다. 뷰의 구조 기반 검색을 지원하기 위한 ViewPath_Info 테이블은 하부 데이타 영역의 Path_Info 테이블과 같은 구조를 갖는다. (그림 10)(a)와 (그림 10)(b)는 (그림 9)의 뷰 정의에 의해 생성된 XML 실체뷰((그림 3)의 XML 실체뷰)를 각각 View_ Info 테이블과 View_Content 테이블에 저장한 모습이다. (그림 10)(c)는 ViewPath_Info 테이블의 모습을 나타낸 것이다.

4. 질의 분할 및 질의 처리

본 절에서는 XML 저장소 내에 XML 실체뷰가 있다고 가정할 때, [11]에서 제시한 실체뷰를 이용한 질의 처리 유형 중 XML 문서에 대한 질의 결과의 내용 중 일부분이 실체뷰에 존재할 경우(유형 UD+MV), 주어진 질의를 ① 실체뷰에 대한 질의와 ② 실체뷰에 포함되지 않은 내용을 직접 하부 데이타로부터 검색하는 질의로 분할한 후 이들을 처리하는 방법에 대해 기술한다. 먼저, [11]에서 제시한 실체뷰를 이용한 질의 처리의 유형 결정에 대해 기술한다. 그리고 이들 유형 중 실체뷰와 하부 데이타를 모두 이용하는 질의 유형의 다섯 가지 경우와 그 예를 기술한다. 그리고 XML 경로 표현식 분할 알고리즘을 제시하고, 질의 분할 알고리즘에 의해 분할된 질의의 처리 과정을 기술한다.

4.1 실체뷰를 이용한 질의 처리의 유형 결정

XML 질의 q와 실체뷰 v가 주어졌을 때, v를 이용한 q의 처리 가능 여부를 판별하고 그에 따라 해당 질의 처리 유형을 결정하는 것은, q와 v의 경로간의 비교 그리고 q와 v의 조건간의 비교를 통해 가능하다. [11]에서는 XML 실체뷰를 이용한 XML 질의 처리의 유형을 결정하는 방법을 <표 1>과 같이 제시하였다.

질의의 대상이 되는 엘리먼트를 목적 엘리먼트라 할 때, **q.path(v.path)**은 q(v)가 명시한 질의의 목적 엘리먼트까지의 경로를 나타내고 이를 q(v)의 **목적 엘리먼트 경로**라고 부른다. 예를 들어 *bookstore/book/author*[*first-name* = "*Michael*"]로 정의된 실체뷰 v가 있다고 할 때, *bookstore/book*[*author/first-name* = "*Michael*" *and author/last-name* = "*Kay*"]이라는 질의 q가 제기된 경우, v.path와 q.path는 각각 *bookstore/book/author*와 *bookstore/book*가 된다. 연산 $<_p$는 p1과 p2와의 경로 비교를 위해 사용된다 : p1이 p2의 prefix와 일치할 경우 p1 $<_p$ p2가 성립한다. 예를 들어 q.path = *bookstore/book/author*이고 v.path = *bookstore/book*이면, v.path $<_p$ q.path이다. <표 1>에서 **q.con(v.con)**은 목적 엘리먼트 또는 그 자손 엘리먼트에 대한 조건을 나타내고 이를 q(v)의 **목적 엘리먼트 조건**이라고 부른다. 이는, q와 v의 목적 엘리먼트 또는 그 자손 엘리먼트에 대한 조건 명시가 있을 경우 이들 조건을 비교하는데 이용한다. q.con(v.con)은 q(v)에서 목적 엘리먼트 또는 그 자손 엘리먼트에 대한 조건이 필터 연산자를 통해 conjunctive normal form으로 [p_1 and p_2 and ⋯ p_n]과 같이 주어졌을 때 이를 집합 {P_1, P_2, ⋯, P_n}으로 표현한 것으로 P_i는 p_i에서 조건이 주어진 대상 엘리먼트를 해당 문서 내의 완전 경로(full path)로 바꾸어 표현된다(i = 1, ⋯, n). 예를 들어 *bookstore/book/author*[*first-name* = "*Michael*"]로 정의된 실체뷰 v가 있다고 할 때, *bookstore/book*[*author/first-name* = "*Michael*" *and author/last-name* = "*Kay*"]이라는 질의 q가 제기된 경우, v.con과 q.con은 각각 {*bookstore/book/author/first-name* = "*Michael*"}과 {*bookstore/book/author/first-name* = "*Michael*", {*bookstore/book/author/last-name* = "*Kay*"}가 된다. 그리고 만약 q 또는 v에 조건이 명시되지 않았다면 q.con과 v.con은 공집합이 된다(자세한 내용은 [11]을 참조).

4.2 유형 UD+MV의 다섯 경우와 그 예

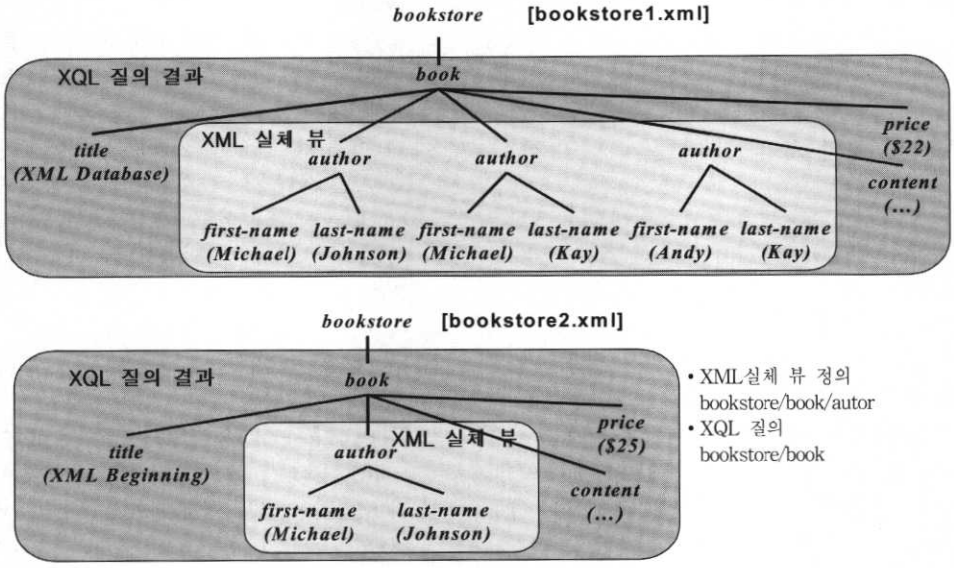
[11]에서는 2절에서 기술한 질의 처리 유형 중 유형 MV에 대하여 주어진 질의 q와 관련 실체뷰 v에 대해 q와 동일한 결과를 얻는 v에 대한 질의 q′를 생성하는 질의 변환 알고리즘을 제시하였다. 본 논문에서는 [11]에서 기술한 질의 처리 유형 중 유형 UD + MV의 질의 q를 관련 실체뷰 v에

<표 1> XML 실체뷰를 이용한 XML 질의 처리의 유형 결정

조건 \ 경로		v.path = q.path	v.path $<_p$ q.path	q.path $<_p$ v.path	otherwise
조건이 명시되지 않은 경우	-	MV	MV	UD + MV case - 1	UD
조건이 명시된 경우	v.con = q.con	MV	MV	UD + MV case - 2	UD
	v.con ⊂ q.con	MV	MV	UD + MV case - 3	UD
	v.con ⊃ q.con	UD + MV case - 4	UD + MV case - 5	UD	UD
	otherwise	UD	UD	UD	UD

〈표 2〉 유형 UD+MV의 다섯 가지 경우의 예

구분 / 유형	q	v
UD + MV case - 1	*bookstore/book*	*bookstore/book/author*
UD + MV case - 2	*bookstore/book* [*author/first-name* = "*Michael*"]	*bookstore/book/author* [*first-name* = "*Michael*"]
UD + MV case - 3	*bookstore/book* [*author/first-name* = "*Michael*" and *author/last-name* = "*Kay*"]	*bookstore/book/author* [*first-name* = "*Michael*"]
UD + MV case - 4	*bookstore/book/author* [*first-name* = "*Michael*"]	*bookstore/book/author* [*first-name* = "*Michael*" and *last-name* = "*Kay*"]
UD + MV case - 5	*bookstore/book/author* [*first-name* = "*Michael*"]	*bookstore/book* [*author/first-name* = "*Michael*" and *author/last-name* = "*Kay*"]

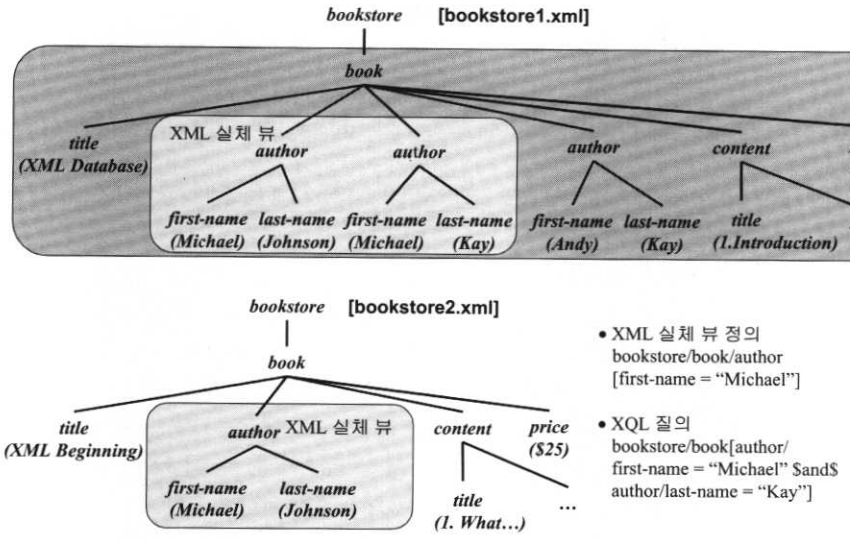


(그림 11) 유형 UD+MV case-1의 예

대한 질의 q'과 하부 데이타에 대한 질의 q"으로 분할하는 알고리즘을 제시한다. 이를 위해 <표 1>에서 나타낸 바와 같이 음영 영역으로 표시된 유형 UD+MV를 경로와 조건의 구분에 따라 다섯 가지의 경우로 나누었다. 이는 질의 변환 과정 시 다섯 가지 경우가 서로 다른 형태의 질의 변환 절차를 필요로 하기 때문이다. <표 2>는 유형 UD+MV의 다섯 가지 경우에 해당되는 예를 경로 질의 q와 실체뷰 v로 구분하여 나타낸 것이다.

(그림 11)은 유형 UD+MV의 다섯 가지 경우 중 case-1의 예를 그림으로 나타낸 것이다. *bookstore/book/author*로 정의된 실체뷰 v가 있다고 할 때, *bookstore/book*이라는 질의 q가 제기된 경우로서 질의와 실체뷰 둘 다에 조건이 명시되지 않으며 q.path $<_p$ v.path가 만족된다. 즉, q의 결과가 v의 내용을 완전히 포함함으로 질의 처리 유형 UD+MV로 처리 가능하다. v는 *bookstore* 엘리먼트의 자식 엘리먼트인 *book* 엘리먼트의 자식 엘리먼트 중 *author* 엘리먼트를 모두 검색하는 것으로 그 결과에 해당되는 서브 트리를 (그림 11)에 옅은 음영 영역으로 표시하였다. 한편, q는 *bookstore* 엘리먼트의 자식 엘리먼트 중 *book* 엘리먼트를 모두 검색하는 것으로 그 결과에 해당하는 서브 트리를 (그림 11)에 진한 음영 영역으로 표시하였다.

(그림 12)는 유형 UD+MV의 다섯 가지 경우 중 case-3의 예를 그림으로 나타낸 것이다. *bookstore/book/author* [*first-name* = "*Michael*"]로 정의된 실체뷰 v가 있다고 할 때, *bookstore/book*[*author/first-name* = "*Michael*" and *author/last-name* = "*Kay*"]이라는 질의 q가 제기된 경우로서 v.con $\subset$ q.con과 q.path $<_p$ v.path가 만족된다. 즉, q의 결과 중 일부가 v의 일부 내용과 일치함으로 질의 처리 유형 UD+MV로 처리 가능하다. v는 *bookstore* 엘리먼트의 자식인 book 엘리먼트의 자식 중 *author* 엘리먼트를 모두 검색하되 자식인 *first-name* 엘리먼트 값이 "*Michael*"인 것만 검색하는 것으로 그 결과에 해당되는 서브 트리를 (그림 12)에 옅은 음영 영역으로 표시하였다. 한편, q는 *bookstore* 엘리먼트의 자식 엘리먼트 중 *book* 엘리먼트를 모두 검색하되 자식인 *author* 엘리먼트의 자식인 *first-name* 엘리먼트 값이 "*Michael*"이고 자식인 *last-name* 엘리먼트 값이 "*Kay*"인 것만 검색하는 것으로 그 결과에 해당하는 서브 트리를 (그림 12)에 진한 음영 영역으로 표시하였다.



(그림 12) 유형 UD+MV case-3의 예

4.3 질의 분할 알고리즘

<표 1>에서 기술한 바와 같이 주어진 XML 경로 표현식 q와 그에 연관된 XML 실체뷰 v가 주어지면 v를 이용한 q의 처리 유형을 결정할 수 있다. 본 절에서는 이들 질의 처리 유형 중 유형 UD+MV를 위한 질의 분할을 수행하는 알고리즘 Decompose_Path_Expression( )을 제안한다(그림 13).

```
Decompose_Path_Expression(v, q, q', q") {
/*   v  : XML 실체뷰 정의
     q  : XML 경로 표현식
     q' : v에 대한 XML 경로 표현식
     q" : 하부 XML 문서에 대한 XML 경로 표현식
*/
/*  UD + MV case -1, 2, 3 */
    if ( (q.path <p v.path) AND (v.con ⊆ q.con)) {
       q' := "/ " + v.elem + filter(and(q.con - v.con))
       q" := q + "\ " + d_path(v, q.level).path + filter (and (q.con))
     }
/*  UD + MV case-4 */
     else if ((q.path = v.path) AND (q.con ⊂ v.con)) {
       q' := " / " + v.elem
       q" := q.path + filter(and(q.con ∪ {not(and(v.con-q.con))}))
}
/*  UD + MV case - 5 */
    else if ((v.path < p q.path) AND (q.con ⊂ v.con)) {
       q' := "/ " + v.elem + d_path (q, v.level).path
             + filter (and (q.con))
       q" := v.path + filter(not(and(v.con))) + d_path (q, v.level)
    }
    q' := refine_query (q')
    q" := refine_query (q")
}
```

(그림 13) XML 경로 표현식 알고리즘

Decompose_Path_Expression()은 각각 질의와 실체뷰를 정의하는 경로 표현식 q와 v를 입력받아 질의 분할을 수행하여, v에 대한 경로 표현식 q'과 하부 데이타에 대한 경로 표현식 q"을 반환한다. 알고리즘에서 사용된 표기(notation)는 <표 3>과 같다[11].

<표 3>의 표기 중 q.elem과 q.level은 다음과 같이 정의된다.

- **q.elem(v.elem)** : 경로 q의 목적 엘리먼트
- **q.level(v.level)** : 경로 q의 레벨 값으로 루트 엘리먼트로부터 목적 엘리먼트까지의 해당 문서 트리의 높이

예를 들어, q가 *bookstore/book/author*[*first-name="Michael"*]일 경우, q.elem은 *author*이고 q.level은 3이다. 또한, a_path(*p*, *l*)과 d_path(*p*, *l*)은 경로 *p*를 레벨 값 *l*을 기준으로 분할하였을 때 생기는 조상 경로와 자손 경로를 각각 나타낸다. 예를 들어, 경로 *bookstore/book/author*[*first-name = "Michael"*]을 레벨 2에서 분할하면 조상 경로는 *bookstore/book*이 되고, 자손 경로는 */author*[*first-name = "Michael"*]가 된다. 레벨 3에서 분할하면 조상 경로는 원래의 경로 그대로 *bookstore/book/author*[*first-name = "Michael"*]이 되고, 자손 경로는 NULL이 된다.

본 논문에서는 XML 질의 변환을 위해 제외 연산자('\')를 XML 경로 표현식의 연산자로 추가하였다. 제외 연산자는 제외 연산자 이후의 엘리먼트를 포함하는 서브 트리를 검색에서 제외하라는 의미를 갖는다. 예를 들어, bookstore/book\author라는 경로 표현식은 bookstore 엘리먼트의 자식인 book 엘리먼트를 검색하되 그 자식인 author 엘리먼트는 검색 결과에서 제외함을 의미한다.

<표 3>의 표기 중 refine_path(q)는 경로 표현식 q에서의 목적 엘리먼트 경로와 목적 엘리먼트 조건의 경로 중 중복되는 부분을 제거한다. 예를 들어 q가 *bookstore/book/author*[*bookstore/book/author/first-name = "Michael"*]이면, refine_path(q)는 *bookstore/book/author*[*first-name = "Michael"*]을 반환한다(v.con과 q.con을 얻는 과정에서 필터 연산자의 경로 표현식은 완전 경로로 바뀌기 때문에 이 작업이 반드시 필요하다). 또한 refine_path(q)는 경로 표현식 q중에 '\'와 '/'가 연속해서 명시되어 있을 경우(즉, '\/') '\'로 변환한

〈표 3〉 표기(notation)

표 기	설 명
not(con)	conjunctive normal form으로 나타낸 조건 cnf가 NULL이면 NULL을 반환하고, 아닐 경우 "{not (cnf)}"를 반환한다.
and(con)	목적 엘리먼트 조건 con이 공집합이면 NULL을 반환하고, {P}이면 P를 반환하고, {P1, …, Pn}일 경우 "P1 and… and Pn"을 반환한다.
filter(cnf)	conjunctive normal form으로 나타낸 조건 cnf가 NULL이면 NULL을 반환하고, 아닐 경우 "[cnf]"를 반환한다.
q.path	경로 표현식 q의 목적 엘리먼트 경로
q.con	경로 표현식 q의 목적 엘리먼트 조건
q.level	경로 표현식 q의 레벨
q.elem	경로 표현식 q의 목적 엘리먼트
a_path (p, l)	경로 p를 레벨 값 l을 기준으로 분할시 조상 경로
d_path (p, l)	경로 p를 레벨 값 l을 기준으로 분할시 자손 경로
refine_query (q)	경로 표현식 q에서 중복 요소 등을 제거

다. 예를 들어 q가 *bookstore/book\/author*이면, refine_path (q)는 *bookstore/book\author*를 반환한다(경로 분할에 의해 구해진 자손 경로는 '/'로 시작하기 때문에 이 작업이 필요하다).

Decompose_Path_Expression( )는 다음과 같이 작동한다. 먼저, DTD를 참조하여 q와 v로부터 q.path, q.con, v.path, v.con을 구한 후, 이들 간의 비교를 통해 <표 1>에서 정리한 질의 처리 유형 중 유형 UD + MV의 다섯 가지 경우 각각에 부합되는 질의 변환을 수행하여 q′과 q″을 생성한다. 이때 생성된 q′과 q″을 대상으로 refine_query( )는 중첩되는 요소를 제거하여 최종 리턴될 q′과 q″을 생성한다. 즉, 조건의 대상 엘리먼트를 나타내는 완전 경로(이들은 q.con과 v.con으로부터 도출된 것임)를 q′과 q″의 목적 엘리먼트 경로 이후의 자손 경로로 변환하고, 제외 연산자와 경로 연산자가 연속해서 명시되어 있을 경우(즉, '\/') 제외 연산자('\')로 변환한다.

<표 2>에서 예시한 유형 UD + MV의 다섯 경우에 대해 Decompose_Path_Expression()을 적용하였을 경우 반환되는 q′과 q″은 <표 4>와 같다.

알고리즘 Decompose_Path_Expression( )의 성능은 주어진 질의 q가 접근하는 XML 문서와 관련 실체뷰 v를 도출한 XML 문서들이 준수하는 DTD 트리의 높이에 따라 결정된다. 즉, DTD 트리의 높이를 h라고 하면, q(v).path와 q(v).con의 계산, 이들 간의 비교, 그리고 q의 경로 분할 등을 통한 q′과 q″의 생성의 복잡도는 O(h)이다.

4.4 분할된 질의의 처리

질의 q를 4.3절에서 언급한 질의 q′과 q″으로 분할하였을 때, 이들을 처리하는 과정(즉, XML 저장소 내의 엘리먼트 접근 과정)은 다음과 같다.

4.4.1 하부 데이타에 대한 q″ 처리 과정

① 조건 엘리먼트 검색과 조건 만족 여부 비교

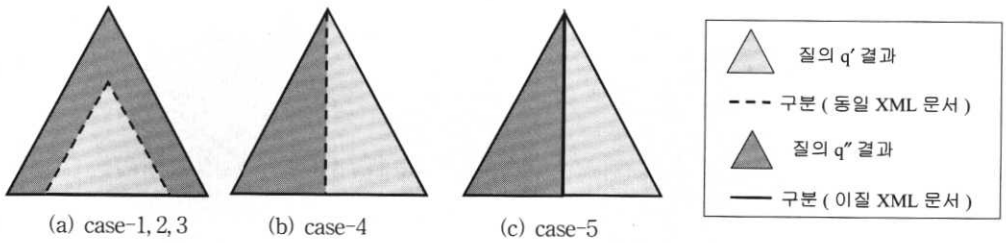
먼저, q″의 처리를 위해서는 조건 엘리먼트의 Element_Info 테이블 내 위치를 찾는다. 이때, 조건 엘리먼트에 대한 인덱스가 있을 때에는 인덱싱을 통해 위치를 직접 찾을 수 있다. 그러나 인덱스가 없을 때에는 하부 XML 문서 전체를 검색하여 조건 엘리먼트의 위치를 찾아야 한다. 즉, 하부 XML 문서 각각의 루트 엘리먼트로부터 연결 정보(즉, 3.3.1절에서 설명한 경로 정보 즉, Parent, Children, Next, Prev 등)를 이용하여 조건 엘리먼트를 찾는다. 검색된 조건 엘리먼트가 검색 조건에 부합되는 지를 판단하기 위해서는 Element_Content 테이블 내 해당 엘리먼트의 Content가 조건을 만족하는 지를 비교해야 한다. 검색 조건을 만족하는 조건 엘리먼트가 포함된 XML 문서가 검색 대상이 된다.

② 목적 엘리먼트 검색

목적 엘리먼트를 찾기 위해 상기 ①의 조건 엘리먼트 검색 과정과 마찬가지로, 목적 엘리먼트에 대한 인덱스가 있을 때에는 인덱싱을 통해 위치를 직접 찾는다. 그러나 인덱스가 없을 때에는 먼저, 조건 엘리먼트가 포함된 XML 문

〈표 4〉 〈표 2〉의 유형 UD+MV의 다섯 경우 예에 대한 질의 변환 결과

유 형 / 리 턴 값	q′	q″
UD + MV case-1	*/author*	*bookstore / book \ author*
UD + MV case-2	*/author*	*bookstore / book [author/ first-name = "Michael"] \ author [first-name = "Michael"]*
UD + MV case-3	*/author [last-name = "Kay"]*	*bookstore / book [author/ first-name = "Michael" and author/ last-name = "Kay"] \ author [first - name = "Michael" and last-name = "Kay"]*
UD + MV case-4	*/author*	*bookstore / book /author [first-name = "Michael" and not (last-name = "Kay")]*
UD + MV case-5	*/book/author [first-name = "Michael"]*	*bookstore / book [not(author/ first-name = "Michael" and author / last - name = "Kay")]/author[first-name = "Michael"]*



(그림 14) 유형 UD+MV의 다섯 가지 경우의 질의 결과 통합 유형

서의 루트 엘리먼트가 Element_Info 테이블 내 어디에 위치하는지를 찾는다. 그리고 검색된 루트 엘리먼트로부터 연결 정보를 이용하여 목적 엘리먼트를 찾는다. 상기 ①의 조건 엘리먼트 검색 과정에서 목적 엘리먼트가 될 수 있는 엘리먼트(목적 엘리먼트의 결정은 조건 엘리먼트가 검색 조건을 만족할 때에만 가능하기 때문)의 위치를 버퍼에 저장하면 하부 XML 문서에 대한 접근 수를 줄일 수 있다. 그러나 이는 조건 엘리먼트와 목적 엘리먼트를 검색하기 위한 구현상의 문제이기 때문에 본 논문에서는 조건 엘리먼트와 목적 엘리먼트의 검색 과정을 나누어 고려하였다.

③ 순회(traverse)를 통한 결과 엘리먼트 추출

Element_Info 테이블 내 목적 엘리먼트를 시작으로 연결 정보를 이용하여 순회하면서 결과 엘리먼트의 구조 정보는 Element_Info 테이블에서, 내용 정보는 Element_Content 테이블에서 추출한다.

4.4.2 XML 실체뷰에 대한 q′ 처리 과정

한편, q′의 처리를 위해서는 실체뷰를 접근해야 하는데, 이는 4.4.1절의 하부 데이타에 대한 질의 처리 과정과 동일하다. 단, XML 실체뷰 영역에는 뷰 정보에 대한 인덱스를 제공하지 않기 때문에 조건 엘리먼트와 목적 엘리먼트 검색시 루트 엘리먼트로부터의 연결 정보를 이용하여 찾는다는 것이 하부 데이타에서의 질의 처리 과정과 상이한 점이다. XML 실체뷰의 내용 전체 또는 일부를 주어진 질의 결과의 일부로 사용(즉, q′의 결과로 사용)하기 위해서는 실체뷰의 일관성이 유지되어야 한다. 본 논문에서는 [10]에서 제시된 하부 XML 문서의 변경을 XML 실체뷰에 점진적으로 지연(deferred) 반영하는 기법을 사용한다.

5. 질의 결과 통합

본 절에서는 4.3절에서 기술한 질의 분할 알고리즘에 의해 생성된 질의 q′과 q″을 처리한 후 이들의 결과를 통합하는 방법에 대해 기술한다. 먼저, 질의 결과 통합에 대한 유형 분류에 대해 기술하고, 질의 결과 통합에 대한 알고리즘을 기술한다.

5.1 질의 결과 통합의 유형 분류

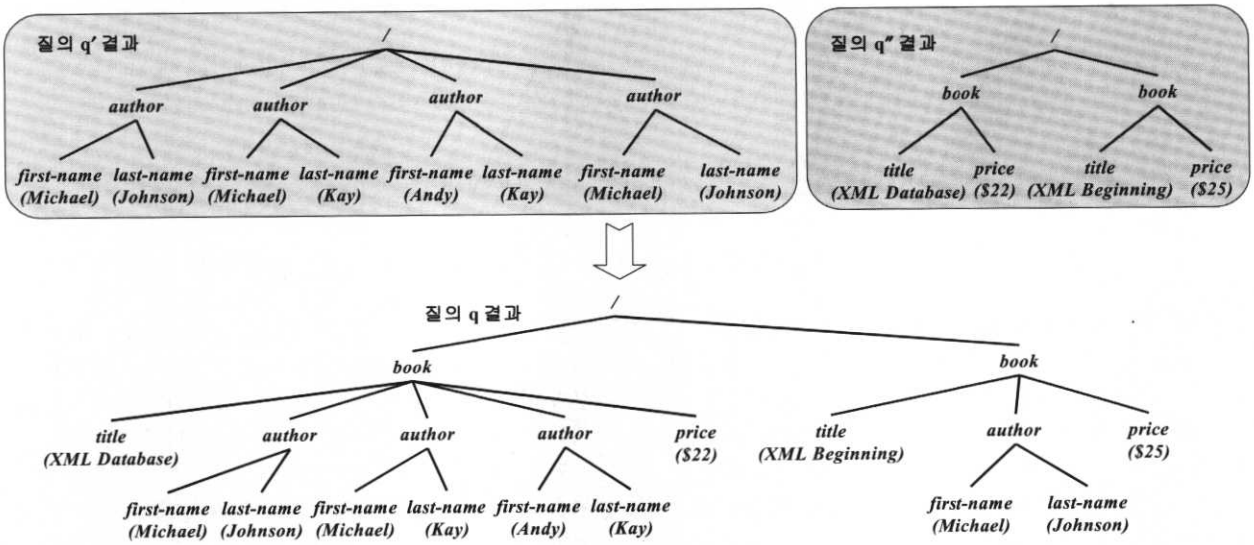
(그림 14)는 <표 1>의 XML 질의 처리 유형 중 유형 UD+MV의 다섯 가지 경우에 대한 질의 결과의 통합 유형을 나타낸 것이다. 질의 결과의 내용은 질의 결과로 검색된 서브 트리들을 루트 엘리먼트를 묶은 것으로 (그림 14)에서는 삼각형(△)으로 나타내었다. XML 경로 표현식의 결과는실체뷰에 대한 질의 q′ 결과와 하부 데이타에 대한 질의 q″ 결과의 통합으로 구성된다. (그림 14)에서는 질의 q′ 결과를 옅은 음영 영역으로 표현하였고, 질의 q″ 결과를 진한 음영 영역으로 표현하였다. 실체뷰는 하부 데이타에서 도출된 것이기 때문에 질의 q′ 결과와 질의 q″ 결과는 같은 하부 데이타의 XML 문서일 수도 있고 서로 다른 XML 문서일 수도 있다. (그림 14)에서 실선은 이질적인 XML 문서에 대한 경계선을 나타내며, 점선은 동일 XML 문서에 대한 경계선을 나타낸다. (그림 14)(a)는 질의 유형 UD+MV의 case-1, 2, 3에 해당하는 경우로서 질의 q′ 결과가 질의 q″ 결과의 서브 트리를 이루는 것을 나타낸 것이다. 이때, 질의 q′ 결과와 질의 q″ 결과는 동일한 XML 문서를 대상으로 한다. (그림 14)(b)는 질의 유형 UD+MV의 case-4에 해당하는 경우로서 질의 q′ 결과와 질의 q″ 결과는 동일한 XML 문서를 대상으로 하면서 질의 결과에 대한 루트 엘리먼트의 서브 트리가 된다. (그림 14)(c)는 질의 유형 UD+MV의 case-5에 해당하는 경우로서 질의 q′ 결과와 질의 q″ 결과는 서로 다른 XML 문서를 대상으로 하면서 질의 결과에 대한 루트 엘리먼트의 서브 트리가 된다.

(그림 15)는 (그림 11)의 유형 UD+MV case-1의 질의 예에 대해 4절에서 제시한 XML 경로 표현식의 분할 알고리즘을 적용하여 분할된 질의 q′ 결과와 질의 q″ 결과, 그리고 이들이 통합된 질의 q 결과를 그림으로 나타낸 것이다. 질의 결과 통합은 (그림 14)(a)와 같이 질의 q′ 결과의 루트 엘리먼트를 제외한 자식 엘리먼트들이 질의 q″ 결과의 book 엘리먼트의 서브 트리를 이루어 최종 질의 q 결과를 생성하는 과정으로 이루어진다.

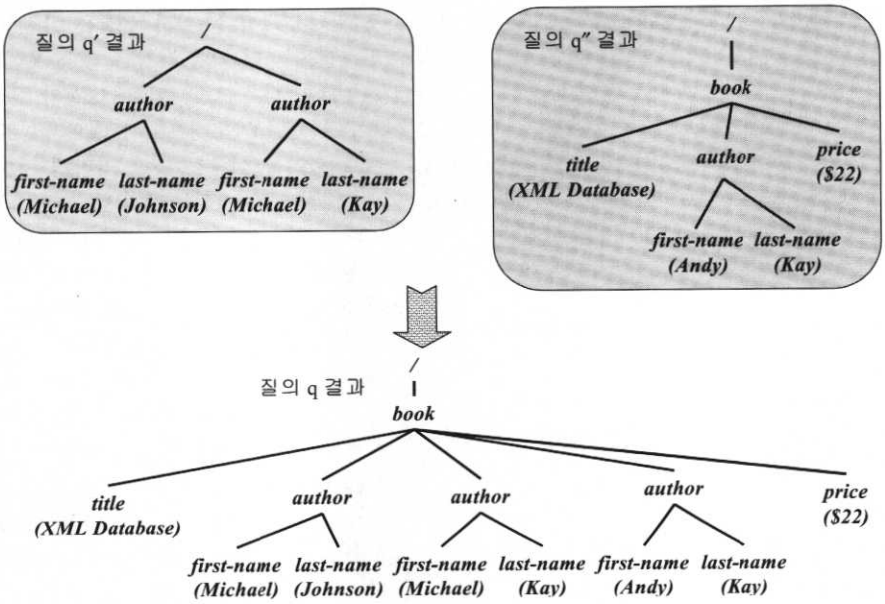
(그림 16)은 (그림 12)의 유형 UD+MV case-3의 질의 예에 대한 결과 통합을 그림으로 나타낸 것이다. 질의 결과 통합은 (그림 13)(a)와 같이 질의 q′ 결과의 루트 엘리먼트를 제외한 자식 엘리먼트들이 질의 q″ 결과의 book 엘리먼트의 서브 트리를 이루어 최종 질의 q 결과를 생성하는 과정으로 이루어진다.

5.2 질의 결과 통합 알고리즘

(그림 17)은 질의 q′ 결과와 질의 q″ 결과의 통합 알고리즘을 나타내고 있다. 질의 q′ 결과를 $R_{q'}$, 질의 q″ 결과를 $R_{q''}$라고 할 때, 질의 처리 유형에 따라 integrate( ) 함수를



(그림 15) 유형 UD+MV case-1의 결과 통합 예



(그림 16) 유형 UD+MV case-3의 결과 통합 예

불러 통합 과정을 수행한다. 통합하고자 하는 결과 A와 B (이때, 결과 A와 B는 모두 하나의 루트 엘리먼트를 갖는 XML 문서임)가 있다고 할 때, integrate( )은 다음과 같이 구성된다.

integrate( )는 결과 A와 B를 (그림 14)의 결과 통합 유형에 따라 통합하는 알고리즘으로 결과 A에 결과 B의 내용을 추가하는 작업을 수행한다. integrate( )의 첫 번째 인자는 통합하고자 결과 A를 나타내고, 두 번째 인자는 결과 A와 통합하고자 하는 결과 B의 내용, 즉, 루트 엘리먼트를 제외한 서브트리를 나타낸다. 세 번째 인자는 A와 B가 통합된 결과 C를 나타낸다. 네 번째 인자는 결과 B의 내용을 추가하기 위한 결과 A의 위치를 나타내며, 다섯 번째 인자는 결과 A와 B의 통합시 A와 B가 동일 XML 문서인 지, 이질 XML 문서인 지를 표시하기 위한 구분자를 나타낸다. 동일 XML 문서의 경우 "SAME"을, 이질 XML 문서의 경우 "DIFF"을 사용한다. subtree()는 루트 엘리먼트를 제외한 서브 트리를 나타낸다. UD+MV case-4, 5의 경우에는 결과 B의 내용이 결과 A의 루트 엘리먼트 아래에 추가되지만, UD+MV case-1, 2, 3의 경우에는 결과 B의 내용이 결과 A의 q″.d_path(v, q.level-1).path에 해당되는 엘리먼트의 아래에 추가된다. (그림 15)의 경우 q″.d_path(v, q.level-1).path은 book이 되어 $R_{q''}$의 book 엘리먼트 아래에 루트 엘리먼트를 제외한 $R_{q'}$의 내용이 추가된다. 또한 UD+MV case-1, 2, 3, 4의 경우에는 "SAME"이라는 지시자를 부여하여 결과 통합시 같은 XML 문서에서 도출된 결과끼리 묶어서 최종결과를 구성한다.

```
Integrate_Result(Rq, Rq' ) {

/* Rq' : 실체뷰에  대한 질의 q'의 result set
   Rq'' : 하부 데이타에 대한 질의 q''의 result set
*/

      Rq' := init_resultset( )     /* 질의 q의 result set 초기화)

      if (UD + MV case-1, 2, 3)
           integrate (Rq', subtree(Rq', '/' ), Rq, Rq-d_path(v, q level-1),path,
                      SAME)

      if (UD + MV case-4)
           integrate(Rq, subtree(Rq', '/' ), Rq, '/', SAME)

      if (UD + MV case-5)
           integrate(Rq'', subtree(Rq', '/' ), Rq, '/', DIFF)

      Rq := refine_result(Rq)

      return(Rq)
}

integrate (A, B, C, Path, Flag)
/* A, B, C : result set의 포인터로 A와 B를 통합하여 결과 C를 완성
   Path : B를 A에 통합시키기 위한 A의 위치
   Flag : 통합시 A와 B가 동일 XML 문서인 지, 이질 XML 문서인 지를 표시)
*/
      i := j := k := 0

      Ra := fetch (A [i++])
      Rb := fetch (B [j++])

      if (Flag = DIFF) {          /* 이질 XML 문서인 경우 */

         do {                     /* A의 내용을 C에 모두 저장 */
               write (C[k++], Ra)
               Ra := fetch (A[i++])
         }while (!end_of A)

         do {                     /* B의 내용을 C에 모두 저장 */
               write (C[k++], Rb)
               Rb := fetch (B [j++])
         }while (!end_of B)

      }
      else {                      /* 동일 XML 문서인 경우 */

         do {
               if (Path_Info[Ra.PathID].Path = Path) /* 통합 위치 검사*/
                  merge(A, B, C, Ra, Rb, &i, &j, &k)   /* 통합 */

               write (C [k++], Ra)
               Ra := fetch (A [i++])
         }while (!end_of A)

         if (!end_of B) {          /* B의 나머지 내용을 B에 저장 */
               Rb := fetch (B [j++])
               do {
                    write (C [k++], Rb)
                    Rb := fetch (B [j++])
               }while (!end_of B)
            }

   }
}
```

(그림 17) XML 질의 결과 통합 알고리즘

6. 성능 분석

본 절에서는 XML 경로 표현식 처리에 있어 질의 분할에 의해 XML 실체뷰를 이용한 질의 처리와 실체뷰를 이용하지 않는 질의 처리간의 성능을 비교 분석한 후 실체뷰를 이용한 질의 처리가 성능 향상을 가져오는 조건을 구한다. 6.1절에서는 성능 분석의 개요에 대해 기술한다. 6.2절에서는 성능 분석에 사용되는 파라미터에 대해 설명한다. 6.3절에서는 성능 분석 결과를 기술한다.

6.1 성능 분석 개요

본 논문의 질의 처리 비용 모델은 다음과 같다. 질의 처리 비용은 XML 저장소를 대상으로 질의를 처리하는 과정에서 접근되는 테이블((그림 4)에서 언급한 하부 데이타 영역의 Element_Info 테이블, Element_Content 테이블, Doc_Info 테이블, DTD 테이블, Path_Info 테이블, 그리고 인덱스와 XML 실체뷰 영역의 View_Info 테이블, View_Content 테이블, View_Definition 테이블, ViewPath_Info 테이블, Path_Info 테이블과 DTD 테이블)의 레코드 수의 총합에 비례한다. 따라서 본 성능 분석에서는 질의 처리 과정에서 접근되는 레코드 수의 총합을 성능 척도로 하였다.

실체뷰를 이용한 질의 처리 방법과 실체뷰를 이용하지 않는 질의 처리 방법간의 성능을 분석하기 위해 본 논문에서는 발생 가능한 질의 유형을 다음과 같이 분류한다. 4.4절에서 기술한 조건 엘리먼트 검색과 목적 엘리먼트 검색에 대해 하부 데이타에 대한 인덱스의 제공 여부로 <표 5>와 같이 질의 유형을 분류할 수 있다. 본 논문에서는 하부 데이타에 대한 인덱스로 [18]에서의 1-index 방법을 사용하였다.

〈표 5〉 인덱스 제공 여부에 따른 질의 유형 분류

질의 유형 \ 질의 처리과정	조건 엘리먼트 검색	목적 엘리먼트 검색
유형 1	○	○
유형 2	○	×
유형 3	×	○
유형 4	×	×

○ : 인덱스 제공, × : 인덱스 제공하지 않음

(그림 5)의 bookstore DTD를 준수하는 하부 XML 문서들이 XML 저장소에 저장되어 있고, 이들 XML 문서를 대상으로 다음의 엘리먼트 경로에 대한 인덱스가 존재한다고 가정하자.

- /bookstore/book/author
- /bookstore/book/author/first-name
- /bookstore/book/content/title

(유형 1)은 조건 엘리먼트 검색과 목적 엘리먼트 검색 시 인덱스를 모두 활용할 수 있는 질의 유형으로 "/bookstore/book/author[first-name = "Michael"]"가 그 예이다. (유형 2)는 조건 엘리먼트 검색시 인덱스를 활용할 수 있으나, 목적 엘리먼트 검색시 인덱스를 활용할 수 없는 질의 유형으로 "/bookstore/book/author[last-name="Johnson"]"이 그 예이다. (유형 3)은 조건 엘리먼트 검색시 인덱스를 활용할 수 없으나, 목적 엘리먼트 검색시 인덱스를 활용할 수 있는 질

의 유형으로 "/bookstore/book/content[title = "1. Introducti on"]"이 그 예이다. 마지막으로 (유형 4)는 조건 엘리먼트 검색과 목적 엘리먼트 검색시 인덱스를 모두 활용할 수 없는 질의 유형으로 "/bookstore/book/content[chapter/title = "1.1 Why XML and Databases"]"가 그 예이다.

유형 분류에 의해 4가지 질의 유형으로 분류했으나(유형 2)는 (유형 1)과 같은 질의 처리 과정을, (유형 3)은 (유형 4)과 같은 질의 처리 과정을 수행한다. 엘리먼트 명에 대한 인덱싱은 조건 엘리먼트를 포함하는 XML 외 다수의 XML 문서의 엘리먼트를 지칭하고 있음으로 목적 엘리먼트를 찾기 위해 인덱스를 활용하는 것보다는 Element_Info 테이블 내 루트 엘리먼트로부터 연결 정보(Parent, Children, Next, Prev 등)를 이용하는 것이 유용하기 때문이다. 따라서 본 논문에서는 (유형 1)과 (유형 4)의 실체뷰를 이용한 질의 처리 방법과 실체뷰를 이용하지 않는 질의 처리 방법 간의 성능을 분석한다.

또한 본 논문에서는 성능 분석의 간편화를 위해 다음의 조건들을 가정하였다.

- 질의 분할에 의해 생성된 질의 q′의 결과는 실체뷰의 내용 전체가 되는 질의 처리만을 고려하였다.
- 실체뷰를 이용한 질의 처리 유형의 결정 및 질의 분할 시 참조되는 View_Definition과 조건 엘리먼트 검색 단계에서 필요한 DTD 테이블과 Path_Info 테이블에 대한 접근 비용은 무시한다(즉, 접근 레코드 수 = 0). 이는 XML 문서의 내용을 저장하고 있는 Element_Info 테이블(View_Info 테이블)과 Element_Content 테이블(View_Content 테이블)에 대한 접근 비용이 질의 처리 비용 중 대부분을 차지하기 때문에 세운 가정이다.

6.2 성능 파라미터

성능 분석에 사용되는 파라미터는 <표 6>과 같다. q는 XML 저장소를 대상으로 한 질의를 나타내는 것으로, 실체뷰에 대한 질의 q′과 하부 데이타에 대한 질의 q″으로 분할된다. $C_{refresh}$는 질의 처리에 사용되는 XML 실체뷰의 갱신을 위한 레코드 접근 수로서 u×x×t로 나타낼 수 있다. 여기서 u는 단위 시간 당 뷰의 하부 XML 문서에 대한 변경 회수를 나타내며, x는 하부 XML 문서에 대한 한번의 변경을 뷰 갱신에 반영하기 위한 엘리먼트 접근 수이다. 따라서 u×x는 단위 시간 당 뷰 갱신을 위한 레코드 접근 수가 된다. t는 해당 XML 실체뷰 접근 후 다시 접근할 때까지의 시간 간격을 나타낸다. 예를 들어 1초마다 뷰와 연관된 XML 문서 중 하나의 문서가 변경되고, 뷰 갱신과 연관된 엘리먼트의 레코드 접근수가 10이라고 하자. 이때 XML 실체뷰에 대한 재요청이 10초마다 발생한다고 하면, XML 실체뷰의 갱신 비용인 레코드 접근 회수는 1회/초×10×10초로 100회가 된다. 마찬가지로 1초마다 뷰와 연관된 XML 문서 중 다섯 문서가 변경되고, 뷰 갱신과 연관된 엘리먼트의 엘리먼트의 레코드 접근수가 20이며, XML 실체뷰에 대한 재요청이 20초마다 발생한다면 XML 실체뷰의 갱신 비용은 20000회가 된다.

6.3 성능 분석 결과

XML 경로 표현식 처리의 성능은, 먼저 XML 실체뷰를 이용한 질의 처리와 실체뷰를 이용하지 않는 질의 처리로 나누어 분석 후 이들을 서로 비교한다.

6.3.1 실체뷰를 이용하지 않는 질의 처리

4.4절에서 기술한 질의 처리 방법에 따라 조건 엘리먼트 검색 단계, 목적 엘리먼트 검색 단계, 그리고 결과 엘리먼트 추출 단계로 나누어 계산한다. 인덱스 이용 여부는 6.1절에서 기술한 바와 같이 조건 엘리먼트 검색 단계에 영향을 준다.

먼저 조건 엘리먼트 검색 단계에서 인덱스를 이용하지 않을 때의 레코드 접근 수 $C_{element_nonindex}$는 식 (1)과 같다. $C_{element_nonindex}$는 DTD를 참조하는 각 XML 문서들의 루트 엘리먼트로부터 조건 엘리먼트의 위치를 찾기 위한 레코드 접근 수와 검색된 조건 엘리먼트와 조건을 비교하기 위한 Element_Content 테이블로의 레코드 접근 수의 합으로 구

〈표 6〉 성능 파라미터 및 값 설정

파라미터 이름	내 용	값
DNxml	하부 데이타에 저장된 XML 문서 중에서 같은 DTD를 참조하는 XML 문서 수	1000
ENxml	같은 DTD를 참조하는 XML 문서의 엘리먼트 개수의 평균	100, 500, 1000, 2000, 3000, 5000
CN	한 XML 문서에서 조건 엘리먼트 경로를 갖는 엘리먼트 개수의 평균	5
Nview	q′ 결과를 위한 대상 XML 문서의 수	100
Nxml	q″ 결과를 위한 대상 XML 문서의 수	100
Rview	XML 실체뷰를 구성하는 엘리먼트의 개수 (즉, q′ 결과의 엘리먼트 개수)	0, 10, 50, 100, 200, 300, 800, 1800, 2500, 3800, 5800
Rxml	실체뷰를 이용한 질의 처리 시 하부 XML 문서에서 검색되는 엘리먼트의 개수 (즉, q″ 결과의 엘리먼트 개수)	10, 50, 100, 200, 300, 2500
Crefresh	XML 실체뷰 갱신 비용	100, 200, 1000, 2000, 5000, 10000
hxml	하부 XML 문서 접근을 위한 인덱스 접근 비용	2
Navg	6.1절에서 가정한 인덱스가 갖는 지시자 수의 평균	100
α	인덱스 접근 비용의 레코드 접근 회수로의 환산율	2
β	실체뷰 갱신 비용의 레코드 접근 회수로의 환산율	1

한다.

$$C_{element\_noindex} = DN_{xml} \times (EN_{xml} + CN) \quad (1)$$

조건 엘리먼트 검색 단계에서 인덱스를 이용할 때의 레코드 접근 수 $C_{element_index}$는 식 (2)와 같다. $C_{element_index}$는 인덱스를 이용한 레코드 접근 수와 Element_Info 및 Element_Content 테이블의 레코드 접근 수의 합으로 구한다.

$$C_{element\_index} = \alpha \times h_{xml} + N_{avg} \times 2 \quad (2)$$

다음은 목적 엘리먼트 검색 단계로서 레코드 접근 수 $T_{element}$는 식 (3)과 같다. 목적 엘리먼트를 포함하는 XML 문서 별로 루트 엘리먼트에서부터 XML 문서 전체를 검색하기 위한 Element_Info 테이블의 접근 수가 발생한다.

$$T_{element} = (N_{view} + N_{xml}) \times EN_{xml} \quad (3)$$

마지막은 결과 엘리먼트 추출 단계로서 추출에 따른 레코드 접근 수 $E_{element}$는 식 (4)와 같다. $E_{element}$는 Element_Info 테이블과 Element_Content 테이블에 대한 레코드 접근 수의 합으로 구한다.

$$E_{element} = 2 \times (R_{xml} + R_{view}) \quad (4)$$

6.3.2 실체뷰를 이용한 질의의 처리

먼저 XML 실체뷰를 이용한 질의 처리의 경우, XML 경로 표현식 처리비용 R_{UV}는 기본적으로 식 (5)에서 열거한 다섯 가지 항들의 합으로 구성된다.

실체뷰를 이용한 XML 경로 표현식의 처리를 위한 비용 R_{UV}
= 질의 분할 비용(A)
+ 실체뷰에 대한 질의 q′ 처리비용(B)
+ 하부 데이타에 대한 질의 q″ 처리비용(C)
+ 결과 통합 비용(D)
+ 실체뷰 갱신 비용(E) (5)

첫 번째 항(이하, A항)은 주어진 XML 경로 표현식의 질의 분할 비용이다. 두 번째 항(이하, B항)은 질의 분할에 의해 생성되는 q′을 처리하는 비용이다. 세 번째 항(이하, C항)은 질의 분할에 의해 생성되는 q″을 처리하는 비용이다. 네 번째 항(이하, D항)은 q′의 결과와 q″의 결과를 하나의 결과로 통합하는 데 소요되는 비용이다. 다섯 번째 항(이하, E항)은 해당 실체뷰를 도출한 하부 XML 문서에 발생한 변경을 실체뷰에 반영하는 데 드는 비용이다.

먼저, 질의 분할은 O(n)(n은 경로 표현식 내 XML 엘리먼트의 수)의 복잡도를 갖는 (그림 13)의 질의 분할 알고리즘에 의해 CPU 내 계산으로 수행되기 때문에 A항 즉, 질의 분할 비용은 전체 XML 경로 표현식 처리 비용 R_{UV}에 대비하여 미미하다. 또한 6.1절에서와 같이 질의 분할 시 참조되는 View_Definition 테이블에 대한 레코드 접근 수를 0으로 가정했기 때문에 본 논문에서는 실체뷰를 이용한 질의의 처리 과정 중에 발생하는 질의 분할 비용을 무시한다 (즉, A = 0).

식 (5)의 B항 즉, 실체뷰에 대한 질의 q′을 처리하는 비용 $R_{q'}$은 식 (6)과 같다. $R_{q'}$은 View_Info 및 View_Content 테이블의 레코드 접근 수의 합으로 구한다.

$$R_{q'} = 2 \times R_{view} \quad (6)$$

식 (5)의 C항 즉, 하부 데이타에 대한 질의 q″을 처리하는 비용 $R_{q''}$은 상기 (가) 실체뷰를 이용하지 않는 질의 처리 방법과 같이 3단계로 계산된다. 먼저 인덱스를 이용하지 않을 때의 $R_{q''}$은

$$R_{q''} = DN_{xml} \times (EN_{xml} + CN) + N_{xml} \times EN_{xml} + 2 \times R_{xml}$$

이고, 인덱스를 이용할 때의 $R_{q''}$은

$$R_{q''} = \alpha \times h_{xml} + N_{avg} \times 2 + N_{xml} \times EN_{xml} + 2 \times R_{xml}$$

이다.

식 (5)의 D항 즉, q′과 q″을 처리한 결과를 통합하는 비용 R_I는 식 (7)과 같다. 비용 R_I는 질의 q의 검색 결과 $R_{view} + R_{xml}$을 디스크에 저장하고, 이를 읽어 통합하는 레코드 접근 수의 합으로 구성된다.

$$R_I = 2 \times 2 \times (R_{view} + R_{xml}) \quad (7)$$

마지막으로 식 (5)의 E항은 $\beta \times C_{refresh}$로 나타낸다.

6.3.3 성능 분석

6.1절에서 기술한 질의 유형에 따라 실체뷰를 이용한 질의 처리와 실체뷰를 이용하지 않는 질의 처리간의 성능 분석 결과를 정리하면 다음과 같다.

- (유형 1)의 질의 처리(조건 엘리먼트 검색시 인덱스를 이용한 경우)
  - 실체뷰를 이용한 질의 처리 방법(P1) :

$$2 \times R_{view} + \alpha \times h_{xml} + 2 \times N_{avg} + N_{xml} \times EN_{xml} + 2 \times R_{xml} + 4 \times (R_{view} + R_{xml}) + \beta \times C_{refresh} \quad (8\text{-a})$$

  - 실체뷰를 이용하지 않는 질의 처리 방법(P2) :

$$\alpha \times h_{xml} + N_{agv} \times 2 + (N_{view} + N_{xml}) \times EN_{xml} + 2 \times (R_{xml} + R_{view}) \quad (8\text{-b})$$

- (유형 4)의 질의 처리(조건 엘리먼트 검색시 인덱스를 이용하지 않는 경우)
  - 실체뷰를 이용한 질의 처리 방법(P3) :

$$2 \times R_{view} + DN_{xml} \times (EN_{xml} + CN) + N_{xml} \times EN_{xml} + 2 \times R_{xml} + 4 \times (R_{view} + R_{xml}) + \beta \times C_{refresh} \quad (8\text{-c})$$

  - 실체뷰를 이용하지 않는 질의 처리 방법(P4) :

$$DN_{xml} \times (EN_{xml} + CN) + (N_{view} + N_{xml}) \times EN_{xml} + 2 \times (R_{xml} + R_{view}) \quad (8\text{-d})$$

XML 실체뷰를 이용한 질의 처리 방법과 실체뷰를 이용

하지 않는 질의 처리 방법의 레코드 접근 회수를 비교하기 위해 두 방법간의 차를 나타내면 다음과 같다.

$$P1-P2 = P3-P4 = 4\times(R_{view}+R_{xml}) + \beta\times C_{refresh} - N_{view}\times EN_{xml} \quad (9)$$

XML 실체뷰 이용한 질의 처리 방법의 성능이 실체뷰를 이용하지 않는 질의 처리 방법보다 더 우수하기 위해서는, (유형 1)의 질의에 대해서는 P1이 P2보다 작아야 하고, (유형 4)의 질의에 대해서는 P3이 P4보다 작아야 한다. 즉,

$$P1-P2 < 0 \ \ or \ \ P3-P4 < 0 \quad (10)$$

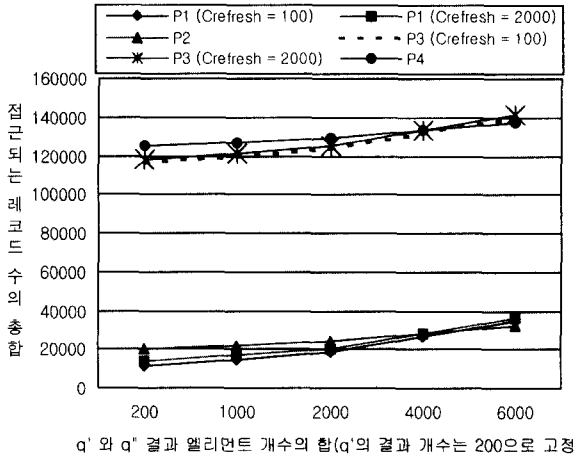
가 되어야 한다.

질의 q가 주어지면, 식 (9)에서의 R_{view}와 R_{xml}, 그리고, N_{view}와 EN_{xml}의 값들은 XML 저장소 내 메타 데이타로부터 추정할 수 있다. 따라서 이들 값들을 이용하여 질의 q에 대한 식 (10)의 성립 여부를 판단할 수 있다. 즉, 질의 분할이 가능한 XML 질의가 주어졌을 때, 질의 처리기는 질의 분할에 의한 실체뷰를 이용한 질의 처리 방법을 사용하는 것이 더 효율적인지 아니면 실체뷰를 이용하지 않는 질의 처리 방법을 사용하는 것이 더 효율적인지 판단할 수 있다.

식 (10)에서 하부 XML 문서의 질의 결과에 해당되는 엘리먼트의 수($R_{view}+R_{xml}$)가 일정하다면, 결국 XML 실체뷰 관리비용 $C_{refresh}$와 실체뷰를 구성하는 하부 XML 문서의 엘리먼트 개수($N_{view}\times EN_{xml}$)에 의해 XML 실체뷰를 이용한 질의 처리의 성능이 결정됨을 알 수 있다. 따라서, 실체뷰 관리비용 $C_{refresh}$를 줄일 수 있는 실체뷰의 효과적인 관리가 이루어지고, 다량의 엘리먼트들로 구성된 하부 XML 문서로부터 생성된 실체뷰를 대상으로 한 질의를 선택하여 처리한다면, XML 실체뷰의 사용은 XML 경로 표현식 처리에 있어 아주 좋은 성능을 나타낼 것이다.

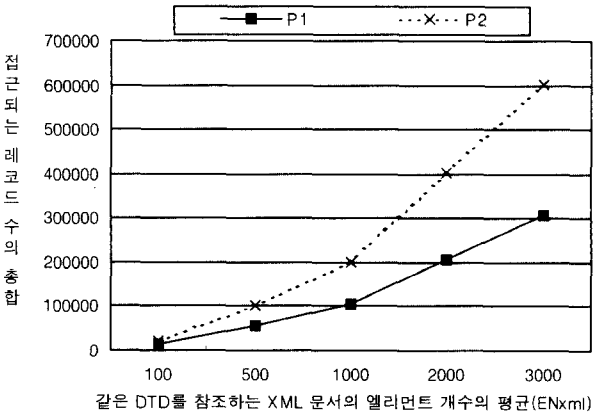
다음은 식 (8-a), 식 (8-b), 식 (8-c)와 식 (8-d)에 정리된 4가지 식에 <표 6>의 파라미터 값을 대입하여 구하여지는 성능 비교의 예를 기술한다.

먼저, R_{view} 즉, 실체뷰를 구성하는 엘리먼트의 개수를 200개로 고정하고 실체뷰 갱신 비용을 레코드 접근 수 100과 2000으로 주었을 때, 결과 엘리먼트를 구성하는 엘리먼트 수의 변화(200, 1000, 2000, 4000, 6000개)에 대한 질의 처리 상의 레코드 접근 수를 구하면 (그림 18)과 같다. (그림 18)에서 나타난 것과 같이 결과 엘리먼트의 개수가 적을 때에는 실체뷰를 이용한 질의 처리 방법(P1, P3)의 레코드 접근 수가 실체뷰를 이용하지 않는 질의 처리 방법(P2, P4)의 레코드 접근 수보다 적음을 알 수 있다. 그러나 결과 엘리먼트의 개수가 증가할수록 XML 실체뷰를 이용한 질의 처리 방법의 증가 비율이 실체뷰를 이용하지 않는 질의 처리 방법의 증가 비율보다 커서 결과 엘리먼트의 개수가 4000개 정도일 때 두 방법간의 레코드 접근 수가 같음을 알 수 있다. 이는 결과 엘리먼트의 개수가 증가할수록(즉, 질의 처리 비용 중 실체뷰에 대한 질의 q′ 처리 비용에 비해 하부 데이타에 대한 질의 q″ 처리 비용의 비중이 커질수록), 실체뷰를 이용하지 않는 질의 처리 방법에 비해 실체뷰를이용한 질의 처리 방법은 q′ 처리에 의한 비용 감소보다 결과 통합 비용이 커지기 때문이다. 또한 XML 실체뷰 갱신 비용이 많을수록 레코드 접근 수가 증가하였으나 레코드 접근 수에 크게 반영되지 않음을 나타내었다. 그리고 인덱스를 사용한 질의 처리 방법(P1, P2)이 인덱스를 사용하지 않는 질의 처리 방법(P3, P4)보다 매우 좋은 성능을 나타냈다. 향후 XML 문서의 특성을 이용한 인덱스가 제공되면 질의의 효율적인 처리가 이루어질 것이다.



(그림 18) 결과 엘리먼트 개수의 변화에 따른 레코드 접근 회수

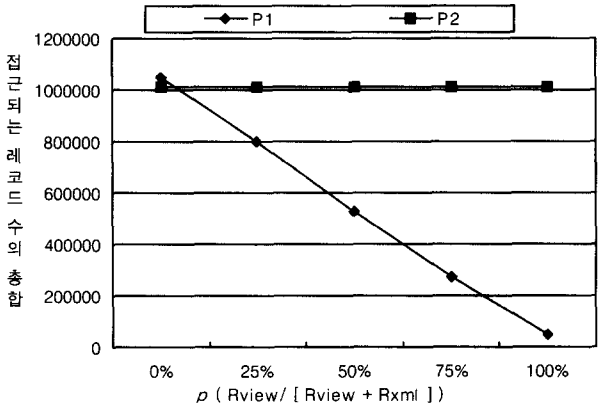
다음은 인덱스를 이용한 질의 처리 방법 중 같은 DTD를 참조하는 XML 문서의 엘리먼트 개수 EN_{xml}의 변화에 대한 질의 처리 상의 레코드 접근 수를 구하면 (그림 19)와 같다. (그림 19)에서는 EN_{xml}을 100, 500, 1000, 2000, 3000으로 하고 질의의 결과는 각각 EN_{xml}의 20%인 20, 100, 200, 400, 600으로 하였으며, 질의 결과 중 R_{view}와 R_{xml}의 비율을 1 : 1로 하였다(즉, 각각 R_{view}와 R_{xml}을 10, 50, 100, 200, 300으로 하였다). 실체뷰 갱신 비용을 레코드 접근 수 5000으로 주었을 때, XML 문서의 평균 엘리먼트 개수가 증가할수록 실체뷰를 이용한 질의 처리 방법과 실체뷰를 이



(그림 19) XML 문서의 엘리먼트 개수의 변화에 따른 레코드 접근 회수

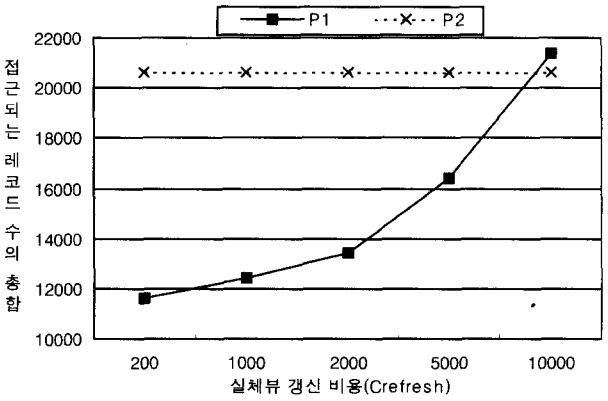
용하지 않는 질의 처리 방법 간의 성능 차가 크게 나타났다. 이는 XML 문서의 엘리먼트 수가 증가할수록 하부문서에서의 검색 비용이 증가하는 데, 실체뷰를 이용함으로써 검색 시간을 줄일 수 있기 때문이다.

다음은 인덱스를 이용한 질의 처리 방법 중 결과 엘리먼트의 개수가 5000개이고, 실체뷰 갱신 비용을 레코드 접근 수 2000으로 주었을 때, 결과 엘리먼트 중 실체뷰에 대한 질의 q′의 결과 엘리먼트 비중의 변화에 따른 레코드 접근 수를 구하면 (그림 20)과 같다. 비율 p는 질의 q의 결과 엘리먼트 중 실체뷰에 대한 질의 q′의 결과 엘리먼트가 얼마나 차지하는 지에 대한 비율을 의미한다. 예를 들어, 질의 q의 결과 엘리먼트를 하부 엘리먼트와 실체뷰에서 각각 반씩 얻을 때, 비율 p는 50%가 되며, 전부 하부 엘리먼트나 실체뷰로부터 얻을 때는 각각 0%와 100%가 된다. 비율 p가 0%일 때, 실체뷰를 이용하지 않는 질의 처리 방법의 성능이 실체뷰를 이용한 질의 처리 방법의 성능보다 좋게 나타난다. 이는 실체뷰를 이용한 질의 처리 방법의 경우, 관련 실체뷰에 대한 접근하는 데 필요한 레코드 수에 하부 데이타 접근에 필요한 레코드 수가 더해져서 실제 실체뷰에서 얻어지는 결과가 없더라도 접근 비용이 생기는 오버헤드가 발생하므로 실체뷰를 이용하지 않는 방법보다 좋지 않은 성능을 내게 되기 때문이다. 그러나 비율 p가 증가할수록 실체뷰를 이용한 질의 처리 방법의 성능이 실체뷰를 이용하지 않은 질의 처리 방법의 성능보다 좋게 나타난다.



(그림 20) 비율 p의 변화에 따른 레코드 접근 회수

성능 비교 예의 마지막으로 (그림 21)은 실체뷰 갱신 비용의 변화에 따른 질의 처리 상의 레코드 접근 수를 구한 것이다. (그림 21)에서는 EN_{xml}을 100으로 하고 질의의 결과는 200(R_{view}와 R_{xml}을 각각 100으로 둠)으로 하였으며, 실체뷰 갱신 비용을 200, 1000, 2000, 5000, 10000으로 변화시켰다. 실체뷰 갱신 비용이 증가할수록 실체뷰를 이용한 질의 처리 비용이 증가하였으며, 실체뷰 갱신 비용이 9000정도 일 때, 실체뷰를 이용한 질의 처리 방법과 실체뷰를(그림 21) 실체뷰 갱신 비용의 변화에 따른 레코드 접근 회수 이용하지 않는 질의 처리 방법의 레코드 접근 수가 같음을 알 수 있었다. 실체뷰를 이용한 질의 처리가 좋은 성능을 갖기 위해서는 실체뷰의 효과적인 관리가 이루어져야 한다.



(그림 21) 실체뷰 갱신 비용의 변화에 따른 레코드 접근 회수

7. 결 론

본 논문은 XML 저장소 내에 XML 실체뷰가 있다고 가정할 때, XML 문서 검색의 성능 향상을 위해 실체뷰를 이용한 XML 경로 표현식 처리 기법에 대해 연구하였다. 이를 위해 먼저 XML 저장소를 대상으로 한 XML 질의의 모델을 제시하고, XML 실체뷰를 정의하고, 실체뷰를 지원하는 XML 저장소의 구조를 제시하였다. 그리고 주요 연구 이슈인 주어진 XML 경로 표현식을 실체뷰에 대한 질의와 하부 데이타에 대한 질의로 분할하는 알고리즘과 분할 질의의 결과를 통합 알고리즘을 제시하였다. 그리고 ① 실체뷰에 대한 질의와 하부 데이타에 대한 질의로 분할한 후 이들을 처리하여 그 결과를 통합하는 방법과 ② 원래의 질의를 실체뷰를 이용하지 않고 처리하는 방법간의 성능을 비교 분석한 후, 실체뷰를 이용한 질의 처리가 성능 향상을 가져오는 조건을 구하였다.

XML 실체뷰는 뷰 정의에 의해 하부 데이타로부터 검색된 결과로서, 검색된 결과 엘리먼트들을 하나의 루트 엘리먼트로 묶은 XML 문서로 표현된다. XML 저장소는 객체관계형 데이타베이스를 기반으로 하여 크게 XML 문서를 저장하는 하부 데이타 영역과 실체뷰를 저장하는 XML 실체뷰 영역으로 나누어진다. 본 논문에서는 하부 XML 문서와 XML 실체뷰를 저장하기 위해 XML 문서의 구조 정보와 내용 정보를 분할하여 서로 다른 테이블에 저장하고 루트 엘리먼트에서 해당 엘리먼트까지의 경로를 따로 저장하는 구조를 선택하였다.

본 논문에서는 기존의 실체뷰를 이용한 질의 처리 유형 세 가지 중에서 질의 결과 일부를 실체뷰로부터 얻고 나머지 결과를 하부 XML 문서들로부터 얻는 유형 UD + MV에 대해, 주어진 질의 q와 관련 실체뷰 v에 대하여 v에 대한 경로 표현식 q′과 하부 데이타에 대한 경로 표현식 q″을 생성하는 질의 분할 알고리즘 Decompose_Path_Expression( )을 제시하였다. Decompose_Path_Expression( )은 유형 UD + MV 질의를 질의와 실체뷰의 경로와 조건의 구분에 따라

다섯 가지의 경우로 나누어 질의를 분할한다. 분할된 질의 처리를 위해 본 논문에서는 새로운 XML 경로 표현식의 연산자를 추가하였다. 그리고 분할 질의의 결과를 통합하여 원래 질의의 결과를 생성하는 질의 결과 통합 알고리즘 Integrate_Result()를 제시하였다.

유형 UD+MV 질의에 대해 XML 실체뷰를 이용한 질의 처리 방법과 실체뷰를 이용하지 않는 질의 처리 방법으로 나누어 성능을 평가하기 위해, 본 논문에서는 질의 처리 과정에서 접근되는 레코드 수의 총합을 성능 척도로 하였다. 성능 분석 결과, 실체뷰 관리비용이 적을수록, 그리고 같은 DTD를 참조하는 XML 문서의 엘리먼트의 개수가 많을수록, 실체뷰를 이용한 질의 처리 방법이 실체뷰를 이용하지 않는 방법보다 성능이 더 좋음을 확인할 수 있었다.

향후 연구 과제로는 실제 프로토타입 시스템의 구현을 통한 XML 실체뷰를 이용한 질의 처리 방법의 성능 평가가 필요하다. 즉, 제기된 XML 질의를 XML 저장소 내의 하부 XML 문서들에 대해 수행하는 경우에 비해, 실체뷰를 이용한 질의 처리 방법이 얼마나 질의 응답 시간의 단축을 가져오는 지를 확인하는 것이 필요하다.

참 고 문 헌

[1] A. Gupta and I. Mumick, "Materialized Views : Techniques, Implementations and Applications," MIT Press, 1999.

[2] S. Abiteboul and A. Bonner, "Objects and View," Proc. ACM SIGMOD Conf., pp.238-247, 1991.

[3] S. Heiler and S. Zdonik, "Object Views : Extending the Vision," Proc. IEEE Int'l Conf. on Data Engineering, pp.86-93, 1990.

[4] S. Abiteboul, "On Views and XML," Proc. ACM Symp. on Principles of Database System, pp.1-9, 1999.

[5] S. Cluet, et al., "Views in a Large Scale XML Repository," Proc. Very Large Data Bases (VLDB) Conf., pp.271-280, September, 2001.

[6] D. Suciu, "Query Decomposition and View Maintenance for Query Languages for Unstructured Data," Proc. Very Large Data Bases (VLDB) Conf., pp.227-238, 1996.

[7] Y. Zhuge and H. Garcia-Molina, "Graph Structured Views and Their Incremental Maintenance," Proc. Int'l Conf. on Data Eng., pp.116-125, 1998.

[8] S. Abiteboul et al., "Incremental Maintenance for Materialized Views over Semistructured Data," Proc. Very Large Data Bases (VLDB) Conf., pp.38-49, 1998.

[9] M. Fernandez et al., "Efficient Evaluation of XML Middleware Queries," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.103-114, 2001.

[10] 임재국 외, "점진적 갱신에 기반을 둔 XML 형성뷰 관리 프레임워크", 정보처리학회논문지D, 제8-D권 제4호, pp.327-338, 2001.

[11] 김수희 외, "XML 실체뷰를 이용한 XQL 질의 처리", 정보처리학회논문지D, 제8-D권 제5호, pp.461-472, 2001.

[12] S. Boag et al., "XQuery 1.0 : An XML Query Language," http://www.w3.org/TR/xqeury/, 2002.

[13] A. Berglund et al., "XML Path Language (XPath) 2.0," http://www.w3.org/TR/

[14] J. Robie et al., "XML Query Language (XQL)," http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[15] Y. Papakonstantinou and V. Vassalos, "Query Rewriting for Semistructured Data," Proc. ACM SIGMOD Int'l Conf. on Management of Data, pp.455-466, 1999.

[16] D. Calvanese et al., "Answering Regular Path Queries Using Views," Proc. Int'l Conf. on Data Eng., pp.389-398, 2000.

[17] D. Florescu et al., "Query Containment for Conjunctive Queries with Regular Expressions," Proc. PODS, pp.139-148, Jun., 1998.

[18] T. Milo and D. Suciu, "Index Structures for Path Expressions," Proc. Int'l Conf. on Database Theory (ICDT), pp.277-295, 1999.

[19] F. Rizzolo and A. Mendelzon, "Indexing XML Data with ToXin," Proc. 4th Int'l Workshop on the Web and Databases, Santa Barabra, pp.66-73, 2001.

[20] T. Shimura et al., "Storage and Retrieval of XML Documents Using Object-Relational Databases," Proc. Database and Expert Systems Applications (DEXA), pp.206-217, 1999.

[21] A. Deutsch et al., "Storing Semistructured Data with STORED," Proc. ACM SIGMOD Int'l on Management of Data, pp.431-442, 1999.

[22] J. Shanmugasundaram et al., "Relational Databases for Querying XML Documents : Limitations and Opportunities," Proc. Very Large Data Bases (VLDB) Conf., pp.302-314, 1999.

문 찬 호

e-mail : moonch@dblab.cse.cau.ac.kr

1997년 중앙대학교 컴퓨터공학과(공학사)

1999년 중앙대학교 컴퓨터공학과 대학원 (공학석사)

현재 중앙대학교 컴퓨터공학과 박사과정 재학중

관심분야 : 웹 데이타베이스, XML, 질의 변환

강 현 철

e-mail : hckang@cau.ac.kr

1983년 서울대학교 컴퓨터공학과(공학사)

1985년 U. of Maryland at College Park, Computer Science(M.S.)

1987년 U. of Maryland at College Park, Computer Science(Ph.D.)

1988년~현재 중앙대학교 컴퓨터공학과 교수

관심분야 : 이동 데이타베이스, 웹 데이타베이스, DBMS 저장 시스템