관계형 DBMS를 이용한 XML 질의 처리 시스템 XPERT의 개발

정 민 경† · 홍 동 권††

요 약

본 논문은 관계형 DBMS를 사용한 XML 질의 처리 시스템 XPERT(XML Query Processing Engine using Relational Technologies)의 개발 내용을 소개한다. 본 논문의 XPERT에서 제안하는 XML 저장 방식은 XML 문서를 여러 구성 성분별로 나누어 관계형 테이블에 저장하는 분할(decomposition 또는 shredded) 방식을 사용하고, 분할된 관계형 테이블을 바탕으로 XML 질의를 SQL로 변환하고, 관계형 DBMS에서 변환된 SQL을 실행하여 결과를 반환하는 방식을 사용한다. 제안한 XQuery 변환 방식은 먼저 XQuery의 구문 분석을 통하여 AST(Abstract Syntax Tree)를 생성하고, AST를 순회하면서 SQL 문장을 생성한다. 생성된 SQL 문장은 XML 문서의 경로를 사용함으로써 XQuery 연산의 조인 횟수를 감소시키며, 계층적 정보 검사나 문서에 내재된 결과의 순서를 지키기 위하여 사용하는 순서 정보는 Dewey 번호를 효과적으로 사용한다. 특히 XQuery의 XPath와 FLWOR 연산을 SQL로 변환하고 실행하는 효과적인 방법을 제시하고 제안된 XPERT 시스템의 프로토타입을 개발하여 그 기능을 평가한다.

키워드 : XML 질의, XQuery, 순서정보, Dewey 번호

XPERT : An XML Query Processing System using Relational Databases

Min-kyoung Jung† · Dong-kweon Hong††

ABSTRACT

This paper introduces the development XPERT(XML Query Processing Engine using Relational Technologies) which is based on relational model. In our system we have used a decomposed approach to store XML files in relational tables. XML queries are translated to SQLs according to the table schema, and then they are sent to the relational DBMS to get the results back. Our translation scheme produces AST(Abstract Syntax Tree) by analyzing XQuery expressions at first. And on traversing AST proper SQLs are generated. Translated SQLs can reduce the number of joins by using path information and utilize dewey number to preserve document originated orderings among compoments in XML. In addition we propose the efficient algorithms of XPath and XQuery translation. And finally we show the implementation of our prototype system for the functional evaluations.

Key Words : XML Query, XQuery, Document Order, Dewey Number

1. 서 론

인터넷에서 XML의 사용이 급격히 늘어남에 따라 XML 데이터의 양도 빠른 속도로 늘어나고 있다. 방대한 양의 XML 데이터를 효율적으로 관리하기 위해서는 XML과 데이터베이스 기술의 통합이 필수적이며, 이러한 추세에 맞추어 데이터베이스 분야에서는 XML 전용 데이터베이스(native XML database)와 기존 데이터 모델에 XML 기능이 부가된 확장 데이터베이스(XML enabled database) 연구가 활발히 진행되고 있다[1-7]. 특히 지금까지 우리 사회에서 기술적으로 가장 성숙되어 있으며 또 상업적으로 가장 성공한 데이터 모델인 관계형 DBMS 모델을 사용하여 기존에 널리 사용하고 있는 데이터베이스 운용 환경을 변화시키지 않으면서 XML 데이터를 수용하기 위한 노력으로 다양한 연구 결과가 발표되고 있다[1-4]. XML을 관계형 테이블에 저장하는 기법은 XML 문서 전체를 관계형 테이블의 특정 컬럼에 BLOB 자료형으로 저장하는 기법과 XML을 분해하여 테이블의 여러 컬럼에 나누어 저장하는 분할(decomposition 또는 shredded 기법) 기법으로 분류할 수 있다. 2가지 방법 모두 장. 단점이 있으나 데이터의 부분적인 변경 및 삭제와 같은 다양한 연산을 효과적으로 지원할 수 있는 방법으로 분할 기법이 많은 장점을 보이고 있다. 본 논문에서 제안하

※ 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10001-0) 지원으로 수행되었음.
† 준 회 원 : 계명대학교 컴퓨터공학과 박사과정
†† 종신회원 : 계명대학교 공학부 컴퓨터공학 전공 부교수
논문접수 : 2005년 9월 29일, 심사완료 : 2005년 12월 29일

는 방법도 XML을 저장하기 위하여 분할 방식을 사용하며, XML 질의어인 XQuery는 관계형 질의어인 SQL로 변환되어 사용된다.

XML 질의의 결과로 XML 문서의 일부분인 XML 부분(XML fragment)이 반환되게 될 때, 분할 방법에서는 XML의 내용이 관계형 테이블에 흩어져 있으므로 XML 질의의 조건을 만족하는 결과 엘리먼트들을 찾은 후 그 엘리먼트들을 루트로 하는 서브트리(subtree)를 XML 질의의 결과로 반환한다. 즉 XML이 분해되기 전의 XML 부분의 형태를 완전한 형태로 다시 복구하여 반환한다. 이때 XML 문서가 가지는 내재된 순서는 엘리먼트들 사이의 순서, 엘리먼트와 텍스트의 순서, 엘리먼트와 애트리뷰트의 순서들로 분류된다. 이러한 순서는 XML을 분해하여 그 내용을 테이블에 저장할 때 적절한 순서 정보와 같이 저장되어야 원래 XML이 가지는 순서를 효율적으로 완전히 복구할 수 있다. 본 논문은 XML 질의 처리 시스템을 개발하는 과정에서 XML을 관계형 테이블에 저장하는 분할 방법, XML이 가지고 있는 순서를 Dewey number로 인코딩하여 저장하는 방법, 저장된 순서 정보를 사용하여 원래의 XML을 복원하는 기법과 함께 XML 질의어인 XPath와 XQuery FLWOR 연산을 SQL로 변환하는 기법을 제시한다.

본 논문의 구성은 다음과 같다.

2장에서는 관계 연구에 대해서 설명하며, 3장은 제안하는 XML 질의 처리 시스템의 전체 구성과 각각의 구성 성분에 대한 구현 기법을 간략히 설명한다. 4장은 XPath를 구문 분석한 후 AST(Abstract Syntax Tree)로 만들고, AST를 SQL로 변환하는 방법을 제안한다. 5장은 XQuery FLWOR 연산을 SQL로 변환하는 방법을 제안하고, 6장은 본 논문에서 제안한 XQuery 처리 시스템의 시험제품 구현 환경과 실행 결과를 설명한다. 마지막으로 7장에서는 결론 및 향후 연구 방향에 대해서 언급한다.

2. 관계 연구

최근 몇 년간 관계형 DBMS를 이용하여 XML 데이터베이스를 구축하는 새로운 연구의 결과가 활발히 발표되고 있다. XML을 관계형 테이블의 여러 필드에 나누어 저장하는 분할 방법[1, 2, 4], XML 질의를 SQL로 변환하는 방법[4, 6], XML이 가지고 있는 순서 정보를 관계형 테이블에 표현하기 위한 넘버링 방법[9, 10] 등의 연구가 관계형 DBMS 분야에서 활발히 연구되고 있다. 이들 방법들은 비정형적인 XML 모델을 정형적인 관계형 모델로 표현하는 방법, 훨씬 더 다양한 기능을 가진 XQuery를 SQL로 변환하는 방법 등과 같이 어려운 문제를 가지고 있어서 아직 어느 특정 방법이 이 모든 것을 한꺼번에 해결하지는 못하고 있다.

XML 문서가 가지고 있는 순서를 인코딩하는 방법은 저장하는 순서 정보의 수준에 따라 인터벌 인코딩(interval encoding) 방식의 전역 순서(global order), 부모와 자식 정보만을 사용하는 지역 순서(local order), dewey 번호를 엘리먼트의 ID로 사용하는 듀이 번호 순서(dewey number order)의 3가지 방법으로 분류 된다[1]. 3가지 방법의 성능 비교에서 XML 문서의 변경이 필요한 환경에서 듀이 번호 순서의 방법이 검색과 변경 모두를 지원하는 경우 많은 장점을 보이고 있다. 또 논문 [1]에서는 XQuery의 XPath를 SQL로 변환하는 방법을 보여주며 XML 질의의 위치 조건(positional predicate)을 지원하기 위하여 rank() 함수를 이용하여 효과적인 SQL로 변환하는 좋은 방법을 제안하고 있다.

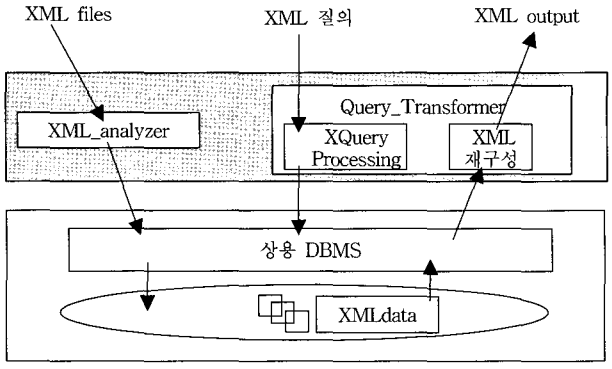
XRel[4] 방식은 XML을 저장하는 방법으로 분할 방식과 XML 전체를 저장하는 2가지 방식을 동시에 사용한다. 분할된 내용은 인덱스로만 사용되며 XML 질의를 효과적으로 지원하기 위하여 경로를 저장하는 방식을 제안하고 있다. 저장된 경로는 XPath를 SQL로 변환하는 알고리즘에서 효과적으로 사용되고 있으나 엘리먼트들의 계층 관계를 파악하는 부분은 XML 문서의 엘리먼트가 몇 번째 단어로 시작하고 끝나는지를 begin, end 값으로 표현하고 있다. 이 방법은 논문[1]의 순서를 저장하는 방법의 분류를 따르면 전역 순서에 해당되어 XML의 변경에 어려움이 있다. 또 질의의 결과를 관계형 테이블에 흩어져있는 내용을 읽어서 다시 복원하는 방식이 아니라 XML 문서의 원본을 BLOB에 저장한 후 begin, end로 표현되어 있는 인터벌 인코딩(Interval Encoding) 정보를 파일의 오프셋(Offset)으로 사용하여 결과 엘리먼트가 형성하는 서브트리를 읽어오는 방식을 사용한다. 이 방법은 검색 전용 환경에서는 사용 가능 하지만 XML의 변경이 가능한 일반적인 경우에는 BLOB의 내용과 테이블에 분산되어 있는 내용과의 동기화에 많은 어려움이 있는 방식이다.

기존의 방식들이 자식 또는 자손 축(axis)만을 고려한 연구인 것과는 달리 논문 [8]은 XPath의 13가지 축(axis)을 모두 고려하는 연구로 새로운 번호 부여 방식을 제안하였다. 각각의 엘리먼트 또는 에트리뷰트에 preorder, postorder number를 부여하여 다양한 축으로의 항해(navigation)를 쉽게 지원한다. 또 이 번호들을 이용하여 XPath의 13가지 축으로의 항해를 SQL로 표현하는 방법과 이들 번호를 XML 문서를 1번만 읽어서 계산하는 과정도 보여준다. 논문 [11]에서는 논문 [8]의 내용을 확장하여 XML 문서의 preorder, 노드의 서브트리에 있는 노드의 개수 size, 루트에서 노드까지의 경로 길이 level을 이용하여 13개의 축의 의미를 쉽게 표현하는 방법을 제안하고 있다. 그 밖에 듀이 번호의 개선 방법으로 XML 문서의 변경이 발생하여도 다른 부분의 번호가 전혀 변하지 않는 방법인 ORDPATH[10] 방법이 제안되었다. 이러한 새로운 번호 부여 기법에 따라 XML 질의의 계층 간 관계를 검사하는 부분의 방법이 달라지므로 XML 질의를 SQL로 변환하는 기법은 XML을 표현 방법에 따라 다양한 방법이 연구되어야 한다. XQuery FLWOR 연산의 SQL 변환은 반복, 재구성(Construction) 등의 기존 관계형 질의 언어인 SQL에서 제공하지 않는 복잡한 기능으로 인하여 매우 복잡한 과정이 필요하다 [6, 11]. 논문 [6]은 동적 인터벌 인코딩 기법과 여러 개의 뷰를 사용하는 방법으로

FLWOR 연산을 반복 및 재구성 기능을 SQL로 변환하는 방법을 소개하고 있다. 하지만 앞에서 언급한 것과 같이 XML을 관계형 테이블에 저장하는 방법에 따라 XQuery를 SQL로 변환하는 방법이 달라져야하므로 기존의 기법들을 그대로 적용할 수는 없다.

3. XML Query 처리 시스템 XPERT 설계

본 논문에서 제안하는 XQuery 처리 시스템 XPERT(XML Query Processing Engine using Relational Technologies)는 (그림 1)과 같이 관계형 DBMS의 상위 계층으로 만들어진다.



(그림 1) XML 질의처리 시스템 XPERT 구성도

(그림 1)에서 사용하는 XML 문서의 데이터베이스는 SiXDML [3]의 컬렉션과 같은 개념으로 각각의 컬렉션에 대해서 서로 다른 테이블 집합이 만들어지며 모든 연산은 현재 컬렉션(current collection)에 대해서 이루어진다. XQuery 처리 시스템의 사용자는 먼저 컬렉션을 선택하고, 현재 컬렉션에 XML 문서를 입력하면 XML_analyzer 모듈에 의해 XML 문서는 분할되어 적절한 테이블에 나누어져 저장된다. XML 컬렉션 또는 XML 문서에 대한 XML 질의는 Query_transformer 모듈에 의해 적절한 SQL로 변환되어 관계형 DBMS 시스템으로 전달되며, 관계형 DBMS는 그 실행 결과를 다시 상위 모듈인 XQuery 처리 시스템으로 반환한다. 구현 과정에서 XQuery 처리 계층에서 많은 SQL 구문을 관계형으로 전달하는 것을 최소화하기 위하여 XML 부분의 재구성과 같은 복잡한 연산은 관계형 DBMS 내에 저장 프로시저로 구현하였다.

3.1 컬렉션 생성 및 삭제

컬렉션 생성 연산[3]은 관계형 DBMS에 다음과 같은 테이블 스키마를 생성한다. 전체 시스템에는 Collections 테이블이 1개 존재하며 각각의 컬렉션에는 Cname_xml, Cname_location, Cname_element, Cname_attribute, Cname_word 테이블이 존재한다.

본 논문에서 사용하는 테이블 스키마는 XFTS 방법 [2]의 확장으로 (그림 2)와 같은 구조를 가지며 밑줄 친 테이블 애

```
Collections(collection_name, Cname)
Cname_xml(doc_id, doc_name)
Cname_location(doc_id, path_id, path, depth, path_cnt)
Cname_element(doc_id, e_id, name, sibord, path_id, key_count, p_id,
value, info, dewey_n)
Cname_attribute(doc_id, a_id, e_id, a_name, a_value)
Cname_word(word, doc_id, e_id, path_id, position, depth)
```

(그림 2) XPERT 시스템의 데이터베이스 스키마

트리뷰트는 테이블의 주 키를 의미한다. 그리고 Cname은 컬렉션이름을 나타내며 컬렉션에 대한 연산으로는 현재 컬렉션으로 선택, 컬렉션 생성, 컬렉션 삭제를 지원한다.

3.2 XML 문서의 입력

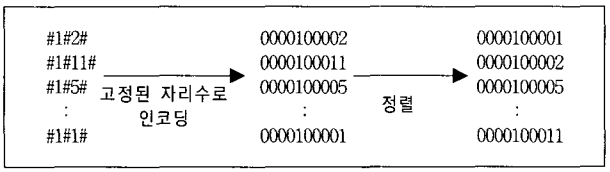
컬렉션을 선택하고 XML 문서를 입력하면 XML 문서는 (그림 1)의 XML_analyzer 모듈에 의해 분석, 분할되어 테이블에 저장된다. XML_analyzer의 초기 모델은 DOM만을 사용하여 구현하였으나 대형의 XML 문서를 처리하는 기능을 추가하기 위하여 SAX를 사용하는 방식도 추가하였다. SAX와 DOM으로 구현한 XML_analyzer는 엘리먼트에 대해서 이름, 경로, 형제간의 순서, 문서에서의 깊이를 추출하고, 애트리뷰트에 대해서 이름, 값, 부모 엘리먼트를 찾아내며, 텍스트 노드의 경우 부모 엘리먼트, 자식 노드들 간의 순서 등을 찾아낸다. 특히 XQuery의 전문 검색 기능(full-text retrieval)을 위해서 텍스트 노드에서 키워드를 추출하는 기능도 수행한다. XML_analyzer는 분석 시간을 최소화하기 위하여 XML 문서를 한번만 읽어 처리하는 단일 패스(single pass) 방식으로 만들어지며, 추출된 내용을 테이블에 저장하는 것도 상위 계층에서 반복적으로 SQL의 "insert into" 구문을 호출하는 것이 아니라 계층 간의 데이터 전달 오버헤드를 최소화하기 위해 데이터베이스 서버에서 일괄적으로 로드하는 방식을 사용하였다.

3.3 관계형 테이블에서 XML 재구성 방법

XML을 분할하여 테이블에 저장하는 방식에서는 XML 질의의 결과로 XML 부분을 재구성(XML fragment reconstruction)하는 과정이 필요하다. 특히 XQuery FLWOR 연산의 경우 XML 생성(construction) 구문을 포함하고 있어서 XPath와는 서로 다른 방법을 적용한다.

XPath의 경우 XML 검색 조건을 만족하는 엘리먼트를 먼저 찾은 후 결과 엘리먼트의 서브트리에 있는 구성 성분들을 문서에 내재된 순서로 읽어오는 과정을 거친다. 이때 엘리먼트를 읽어서 XML을 구성하는 것을 XQuery 처리 계층에서 실행할 경우 관계형 시스템으로 SQL을 여러 번 호출하여 결과를 얻어야 하는 비효율성이 있다. 따라서 본 논문의 XPERT에서 사용하는 방법은 저장 프로시저(오라클의 경우 PL/SQL)를 사용하여 결과를 데이터베이스 서버 쪽에서 필요한 구성 성분들을 순서에 맞게 읽어오고 또 XML로 구성하기 위하여 엘리먼트의 시작 태그와 종료태그를 적절한 위치에 삽입한 후 그 XML 결과를 CLOB 타입의 스트링

으로 만들어 상위 계층인 XQuery 처리 계층으로 보낸다. XQuery 처리 계층에서는 적절한 들여 쓰기와 줄바꿈 등을 사용하여 사용자에게 보기 좋은 형식의 XML을 되돌려준다. 이와는 달리 XQuery FLWOR식의 경우 저장 프로시저를 사용하여 데이터베이스 서버 쪽에서 필요한 결과 값들을 로드하는 것은 XPath와 동일하나 최종 결과값들이 뷰 형태로 존재하므로 기본 색인 테이블이 아닌 최종 뷰를 질의하여 서버로부터 데이터들을 읽어온다. 또한 XML문서에 내재된 순서대로 데이터를 읽어 오기 위해서는 특정 컬럼을 기준으로 결과 뷰에 존재하는 데이터들을 정렬하여야 한다. 이때 e_id 경우 문서의 변경에 상관없이 고정된 값을 가지므로 부정확한 결과가 발생할 수 있으며 만약 dewey order 값을 기준으로 정렬할 경우 이를 사전식 순서대로 정렬하여 데이터를 로드하므로 역시 부정확한 값이 출력된다. 따라서 본 연구에서는 dewey_order 인코딩 기법(본 연구에서는 이를 Dewey_n 인코딩 기법이라 한다.)을 정의하고 이를 적용한다. 즉 각 노드들마다 부여된 dewey_n컬럼값을 0을 포함한 특정 자리수로 변환함으로써 서버로부터 결과 값을 로드할 때는 이를 기준으로 정렬하여 본래 XML문서 내에 존재하는 순서대로 데이터를 읽어온다. 이 또한 저장 프로시저(오라클의 경우 PL/SQL)로 dewey_n컬럼값을 입력으로 하여 고정된 0을 포함한 자리수로 인코딩하여 반환하는 아주 간단한 알고리즘을 가진다.



(그림 3) Dewey_n 인코딩 기법

만약 XQuery FLWOR식에서 for문이 서로 중첩되어 있다면 최종 결과뷰에는 동일한 dewey_n컬럼값을 가진 행들이 다수 존재하게 된다. 이 경우 서브트리별로 차례대로 부여된 최종 결과 뷰의 row_id 컬럼과 Dewey_n 인코딩 기법으로 변환된 dewey_n컬럼들을 기준으로 정렬함으로써 본래의 순서에 맞게 서버로부터 데이터를 읽어 온다. 마지막으로 construction 엘리먼트(XQuery FLWOR식중 return절에 명시된 삽입할 새로운 노드)의 문자열을 이와 결합된 노드들의 name으로부터 분리하고 모든 노드의 name에 시작 태그와 끝 태그를 추가하는 부가적인 작업을 거침으로써 완전한 XML 형태로 사용자에게 보여준다.

4. XPath의 SQL 변환 알고리즘

본 논문에서 제안하는 XQuery 처리 시스템 XPERT에서 지원하는 쿼리 유형은 text검색, 특정 키워드 검색, 현재 오라클에서는 지원하지 않는 여러 조건들의 and, or 연산, i번째 항목을 검사하는 인덱스 검색등과 같은 다양한 검색을 지원하며 이를 BNF 표기법으로 나타내자면 다음과 같다. 즉 본 논문에서 제안하는 XQuery 처리 시스템은 아래의 XPath grammar를 모두 지원한다.

```
XPath ::= Absolute_Expression
                                    Node_Test ::= QName | '@' QName
        | Relative_Expression       Predicate ::= [Complex_Type]
        | Complex_Expression
Complex_Type ::= Index_Number_Type
               | Comparison_Type
Index_Number_Type ::= Number
Equal_Op ::= '='
Relation_Op ::= 'or' | 'and'
Keyword_F :: = contains()
Document_F ::= doc()
Comparison_Type ::= Predicate
                  | Keyword_F 'Literal_String'
                  | Node_Test Equal_Op 'Literal_String'
                  | Comparison_Type Relation_Op
                    Comparison_Type
Absolute_Expression ::= Document_F '/' Node_Test
                      | Absolute_Expression '/' Node_Test
Relative_Expression ::= Document_F '//' Node_Test
                      | Absolute_Expression '//' Node_Test
                      | Relative_Expression '//' Node_Test
Complex_Expression ::= Absolute_Expression Predicate
                      | Relative_Expression Predicate
                      | Complex_Expression Absolute_Expression
                      | Complex_Expression Relative_Expression
```

(그림 4) XPath grammar

본 연구에서는 주어진 XPath를 관계형 언어로 처리하기 전에 먼저 다음과 같은 AST(Abstract Syntax Tree)트리로 변환한다. 이는 자식과 형제 노드만을 가지는 트리로 이들의 형태를 LISP 언어의 형태와 유사하게 표시할 수 있다. 또한 트리를 구성하는 모든 노드들은 각각 자신들만의 타입을 가지며 입력되는 XPath식의 유형에 따라 트리의 모양이 조금씩 달라진다.

(그림 5)와 같이 변환된 AST를 깊이우선탐색 하면서 본 연구에서 제안하는 아래의 알고리즘들을 거치게 되며 이러한 작업이 완료되면 최종 SQL문이 생성된다.

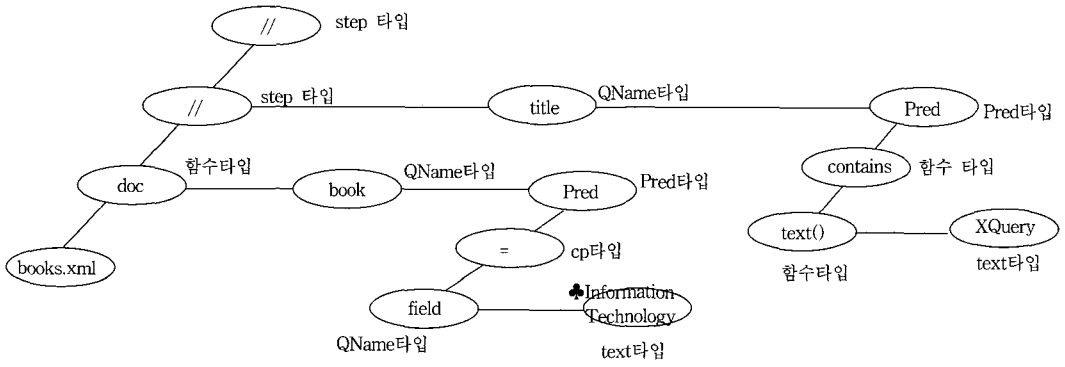
(1) OutputNodeSeek(AST Node)

XPath식으로부터 SQL문을 생성할 때 가장 먼저 적용되는 단계로 변환된 AST 트리에서 XPath 식의 최종 결과 엘리먼트 즉 결과노드(output노드)를 검색한다. 즉 AST 트리를 부모-우측자식-좌측자식 방향으로 운행하면서 가장 처음으로 QName 타입(엘리먼트 타입)의 노드를 만날 때 해당 노드가 가지는 문자열을 저장한다. 위의 (그림 5)의 AST 에서는 title노드가 출력한 결과 노드가 되며 이때 'title'이라는 노드명을 저장한다.

(2) SQL_Creator(Collection_Name, AST node)

두 번째 단계는 사용자가 선택한 컬렉션(Cname)과 AST의 루트 노드 그리고 (1)단계에서 검색한 output 노드의 노드명(본 예시에서는 'title')을 입력받아 AST를 깊이 우선 탐색하

*Infromation Technology분야의 서적 중 XQuery와 관련된 책 제목을 출력하라.
*XPath: doc('books.xml')//book[field='Information Technology']
//title[contains(text(), "XQuery")]
*AST: (// (// (doc books.xml) book (Pred (= field Information Technology))) title (Pred (contains text XQuery)))



(그림 5) AST 트리

면서 본 연구에서 정의하는 SQL_Creator 알고리즘을 각 노드타입별로 적용한다. 이때 엘리먼트들의 XML 트리상의 경로와 (그림 6)과 같은 기본적인 SQL문이 생성되지만 방문하는 AST의 각 노드들의 타입에 따라 조금씩 수정될 수 있다. (그림 6)은 AST 트리를 순회하면서 생성한 경로를 Cname_location 테이블로부터 질의하고 이와 동일한 pathid, docid 컬럼 값을 가진 엘리먼트들을 Cname_element 테이블로부터 검색하는 SQL구문이다. 단 (그림 6)의 'alias'는 AST를 순회하면서 QName 타입의 노드를 만날 때 마다 부여되는 정수값으로 FROM절에 나타날 각 테이블의 별칭을 의미한다.

```
FROM절: Cname_element E || alias, Cname_location L || alias
WHERE절: L || alias .path like 'path'
         and E || alias.docid = L || alias.docid
         and E || alias.pathid = L || alias.pathid
```

(그림 6) 기본적인 FROM절과 WHERE절

다음의 내용은 FROM절과 WHERE절을 생성하는 SQL_Creator 알고리즘은 아주 간략하게 표현한 것이며 자세한 내용은 부록 1에 별도로 작성되어 있다.

A. step 타입 : step 연산자를 Stack에 삽입 한다.

B. Qname 타입 : Stack을 삭제하여 Qname과 삭제한 스텝 연산자를 연결하여 하나의 경로를 형성하고 각 노드마다 1씩 증가한 정수값을 부여한다. 그리고 결과노드와 자신을 비교하여 같으면 그림 6의 기본적인 from절과 where절을 생성한다. 단 이때 부여된 정수값은 각 테이블의 별칭(alias)을 가리키기 위한 것으로 Concatenation 단계에서 SELECT문을 생성하는데 이용한다.

C. Pred 타입 : Qname 타입을 거치면서 생성된 Path로 from절과 Dewey 순서값, 경로를 비교하는 where절을 생성한다. 만약 해당 노드의 자식으로 숫자타입의 노드가 오면 I_TH_CHILD 프로시저를 호출하는 구문을 생성한다. 이는 특정 엘리먼트의 i번째 자식노드를 검색하는 프로시저로 book/title[2]와 같은 식을 처리한다.

D. String 타입 : 현재까지 Qname 타입을 거치면서 생성된 경로로 from절과 where절을 생성하되 텍스트를 검색하는 구문을 추가한다.

E. contains 타입 : Collection_Word테이블의 word컬럼을 참조하는 SQL문을 생성한다.

F. doc 타입 : books.xml 문서의 식별자를 찾는 구문을 생성한다.

G. and, or등의 관계연산자 타입 : 관계 연산자 타입의 노드를 만난 시점부터 String 타입의 노드를 만난 시점까지 생성된 모든 where절을 '('로 묶은 뒤 SQL문에서도 이와 동일한 관계연산자로 처리한다.

(3) Concatenation(Output_node)

마지막으로 (2)단계에서 부여한 결과노드의 별칭으로 최종 결과 엘리먼트의 도큐먼트 식별자와 Dewey 순서값을 추출하는 SELECT절을 생성한 뒤 FROM절에 명시된 각 테이블의 docid 컬럼값을 조인하는 구문을 생성한다. 그리고 SELECT절과 FROM, WHERE절을 결합하여 완전한 하나의 SQL문을 생성한다. 아래의 output_alias는 결과 노드를 검색할 테이블의 별칭을 의미한다.

1) select := 'SELECT e || output_alias.docid, e || output_alias.numbering'
2) where := document id값을 조인하는 구문 생성
3) 최종 SQL := select || from || where

지금까지 살펴 본 알고리즘들을 통해 (그림 5)를 변환한 SQL문은 다음과 같다.

```
SELECT E2.docid, E2.numbering
FROM update_ELEMENT E0, update_LOCATION L0,
update_ELEMENT E1, update_LOCATION L1,
update_ELEMENT E2, update_LOCATION L2,
update_WORD W2
WHERE L0.path like '~%/book'
and E0.docid = L0.docid
and E0.pathid = L0.pathid
and trim(E1.value)='Information Technology'
and L1.path like '~%/book~/field'
and E1.numbering like E0.numbering || '%'
and E1.pathid = L1.pathid
and E1.docid = L1.docid
and L2.path like '~%/book~%/title'
and L2.pathid = E2.pathid
and L2.docid = E2.docid
and E2.docid in (Select id from update_XML_DOCUMENTS where
docname = 'books.xml')
and E2.numbering like E0.numbering || '%'
and trim(W2.word)= 'XQuery' and W2.eid = E2.eid
and W2.docid = E2.docid
and E2.docid = E1.docid
and E1.docid = E0.docid
```

5. XQuery FLWOR 연산의 변환 알고리즘

본 연구에서는 (그림 7)과 같이 for, let, where, return 절로 구성된 XQuery FLWOR식을 SQL문으로 처리할 수 있는 알고리즘을 개발함으로써 XPERT 시스템의 질의 처리 기능을 XPath에서 XQuery FLWOR 표현식까지 확장하였다.

XPERT 시스템에서 XQuery FLWOR식을 처리하는 과정은 단일 for문을 포함할 경우 6단계로 2중 for문을 포함할 경우 5단계로 새로운 엘리먼트 구성작업(element construc-

```
· Information Technology 분야의 'XQuery'와 관련된 책의 가격과
  작가에 대한 정보를 출력하라
for $p in doc('books.xml')//book[field = 'Information Technology']
let $q := $p/author
where $p/title[contains(text(), 'XQuery')]
return <item price = "{$p/price/text()}"> {$q/last} </item>
```

(그림 7) XQuery FLWOR 표현식

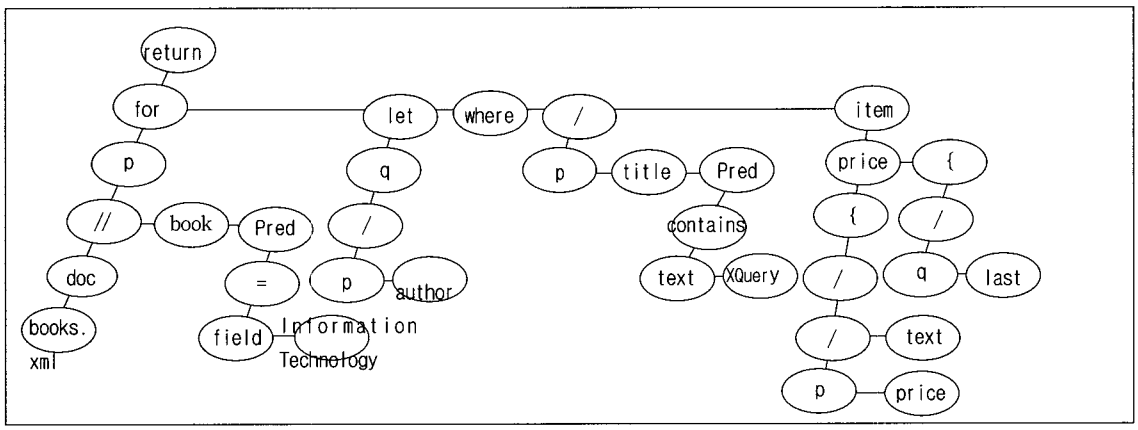
```
XQuery_Flower ::= For x in XPath_F
                | Let x := XPath_F
                | Where XPath_F
                | Return XML_Fragment_Construct
Axis ::= '/'  | '//'                    x ::= variables
XPath_F ::= XPath                  Function ::= count()
          | x 'Axis' XPath_F                | text()
          | XPath 'Function'                | contains()
                                            | exist()
                                            | empty()
XML_Fragment_Construct ::= Complex_Construct | '{' XPath_F '}'
Complex_Construct ::= <New_item> '{' XPath_F '}'
      | <New_item New_attr = "{XPath_F}"> '{' XPath_F '}'
```

(그림 8) XPERT 시스템에서 지원하는 XQuery grammar

tors)이 없는 구문인 경우 1단계로 (그림 8)의 문법에 포함되는 XQuery FLWOR식을 모두 처리한다.

XPath와 마찬가지로 사용자가 XQuery FLWOR식을 입력할 경우 먼저 이를 분석하여 AST형태로 변환한다. 그리고 트리를 깊이 우선 탐색하되 각 서브트리단위로 XPath 처리 알고리즘을 적용하여 SQL구문의 일부분을 생성한다. 이렇게 생성된 구문들은은 AST의 순회가 끝날 때 혹은 AST 상에서 삽입될 노드의 서브트리를 만날 때 동일한 변수에 바인딩 되는 엘리먼트들을 검색하는 구문끼리 결합됨으로써 완전한 SQL문이 생성된다. 예를 들면 다음의 (그림 9)는 (그림 7)의 XQuery FLWOR식을 AST 형태로 표현한 것이며 1개의 for문과 item'이라는 새로운 엘리먼트 구성작업을 포함하고 있으며 XPERT 시스템의 질의 처리 모듈에서 처리되는 과정은 다음과 같다.

1) AST 트리를 깊이 우선탐색하면서 각 서브트리 단위로 XPath 알고리즘을 적용하여 SQL구문들을 생성한다. 그리고 item노드의 새로운 속성값으로 표현될 price노드('p'노드의 우측형제 노드)를 만날 경우 현재까지 생성된 SQL문들 중 변수 p에 바인딩 될 값을 질의하는 구문끼리 결합한다. 이렇게 결합된 SQL문은 item 노드의 속성값으로 표현될 price노드를 검색하여 attr_table뷰를 생성하게 되며 이때



(그림 9) AST 트리

관계형 DBMS 시스템에서 제공하는 rownum값도 함께 추출한다.

attr_table(docid, eid, name, dewey_n, value, info, path, pathid, sibord, row_id)

2) item의 text값으로 표현될 last노드를 만날 경우 현재까지 AST를 순회하면서 생성한 SQL문을 변수 q에 바인딩될 값을 질의하는 구문들 끼리 결합하여 text_table이라는 뷰를 생성한다. 만약 입력된 XQuery FLWOR식에서 새로운 엘리먼트 구성작업이 없을 경우 본 과정을 통해 생성된 text_table 뷰 하나만으로 모두 처리할 수 있다.

text_table(docid, eid, name, dewey_n, value, info, path, pathid, sibord)

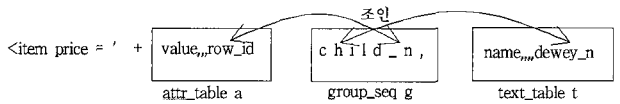
3) 본 과정부터는 AST를 순회하지 않고 새로운 엘리먼트 구성작업이 삽입된 구문일 경우에만 거치게 된다. 만약 새로운 엘리먼트 구성 작업이 없을 경우 본 과정을 모두 생략할 것이다. text_table 뷰의 행들을 book 노드의 Dewey 순서 값으로 그룹핑한 뒤 Dewey_n 인코딩 기법으로 각 그룹의 루트 노드(last엘리먼트)들을 찾고 그들의 인코딩된 Dewey_n값들을 추출하여 Group_view 뷰를 생성한다. 이는 item 노드와 문자열로 결합될 last노드들을 정확하게 검색하기 위함이다.

group_view(child_n)

4) Group_view 뷰의 값들을 정렬한 뒤 각 행의 rownum값을 추출하여 Group_seq라는 뷰를 생성한다. 이 뷰를 통해 다음 단계에서 item 노드의 속성으로 삽입될 값과 텍스트로 삽입될 값들을 정확하게 결합시킬 수 있다. 이때 각각의 컬럼명을 row_id, child_n 이라한다.

group_seq(child_n, row_id)

5) 전 단계에서 생성한 Group_seq 뷰의 child_n 컬럼값과 동일한 값 Dewey_n값을 가진 행들을 text_table 뷰에서 검색하고 row_id 컬럼값과 동일한 값을 가진 value 컬럼값들을 attr_table 뷰에서 검색한 뒤 각각 이들의 value, name 컬럼값을 문자열 결합하여 Concatenation_table 뷰를 생성한다. 이러한 기법으로 새로운 엘리먼트 구성작업을 처리할 경우 각 엘리먼트들의 순서정보가 변경되는 복잡한 갱신현상을 막을 수 있다.



=> conctenation_table(t.name, t.dewey_n, t.value,,,,,t.sibord, a.row_id)

6) Concatenation_table 뷰와 text_table 뷰를 결합하고 중복된 행을 제거함으로써 item 노드의 자식 노드로 위치할 값들을 표현한 뒤 최종 결과뷰를 생성한다. 여기서 중복된 값이란 concatenation_table과 동일한 eid를 가진 text_table의 행들이다.

last_table(docid, eid, name, dewey_n, value, info)

단일 for문을 포함하는 XQuery FLWOR식과는 달리 2개의 for문이 중첩되어 있을 경우 모두 5단계로 처리된다. 입력된 쿼리를 AST 형태로 변환하고 결과값을 검색하여 그룹핑하는 작업까지는 앞에서 살펴본 단일 for문을 포함하는 XQuery FLWOR식 처리 알고리즘과 동일하나 새로운 엘리먼트 구성작업을 처리할 때 attr_table 뷰와 text_table 뷰를 카티션 프로덕트 연산함으로써 2중 for문을 수행한 것과 동일한 개수의 서브트리를 생성하게 된다. 다음의 예시는 2개의 서로 다른 XML 문서로부터 데이터를 검색하는 구문으로 books_korean.xml은 한글이 포함된 문서이며 books.xml과 동일한 컬렉션에 존재한다.

```
1명이상의 작가들이 집필하였으며 제목이 'Data on the Web'인 책의
분류명을 books_korean.xml의 각 subject(도서항목)의 하위 항목으로
표현하여 출력하라.
for $p in doc("book_korean.xml")//book
 for $q in doc("books.xml")//book[title = 'Data on the Web']
 let $k := count($q/author)
 where $k > 1
 return <contents subject = "{$p//subjects/text()}'> {$q/field} </contents>
*본 예시의 AST tree는 많은 공간을 차지하므로 기술하지 않는다.
```

(그림 10) 중첩 for문을 포함하는 XQuery FLWOR 표현식

1) AST 트리를 순회하되 contents 노드의 attribute값으로 표현될 subjects노드를 만나면 현재까지 AST를 순회하면서 생성한 SQL문을 적절히 결합하여 attr_table이라는 뷰를 생성한다. 단 1중 for문의 처리와 유사하게 rownum값도 함께 검색한다.

attr_table(docid, eid, name, dewey_n, value, info, path, pathid, sibord, row_id)

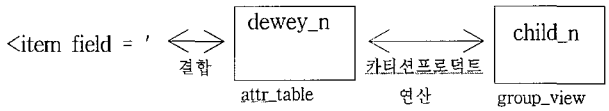
2) contents 노드의 text값으로 표현될 filed 노드를 만날 경우 현재까지 AST를 순회하면서 생성한 SQL문을 적절히 결합하여 text_table이라는 뷰를 생성한다.

text_table(docid, eid, name, dewey_n, value, info, path, pathid, sibord)

3) 2)단계에서 생성한 text_table 뷰를 for문에 출현하는 노드(book엘리먼트)로 그룹핑하여 그중 최고 root노드만을 추출하여 group_view를 생성한다. 이때 book노드는 books.xml에 존재하며 title이 'Data on the Web'인 text를 소유하고 있어야 되며 각 그룹의 root노드를 추출할 때 Dewey_n 인코딩 기법을 통해 인코딩된 dewey 순서값을 검색하여 child_n라는 컬럼명으로 뷰를 생성한다.

group_view(child_n)

4) group_view의 child_n값과 동일한 값을 가진 행을 text_table(item노드가 삽입될 위치)에서 검색한 뒤 attr_table의 행(item노드의 attribute로 들어갈 값)의 dewey_n컬럼 값을 고정된 자리수로 인코딩하고 그 값을 기준으로 카티션 프로덕트 연산을 수행한다. 이로써 2중 for문을 수행한 것과 동일한 개수의 contents노드를 생성한다.



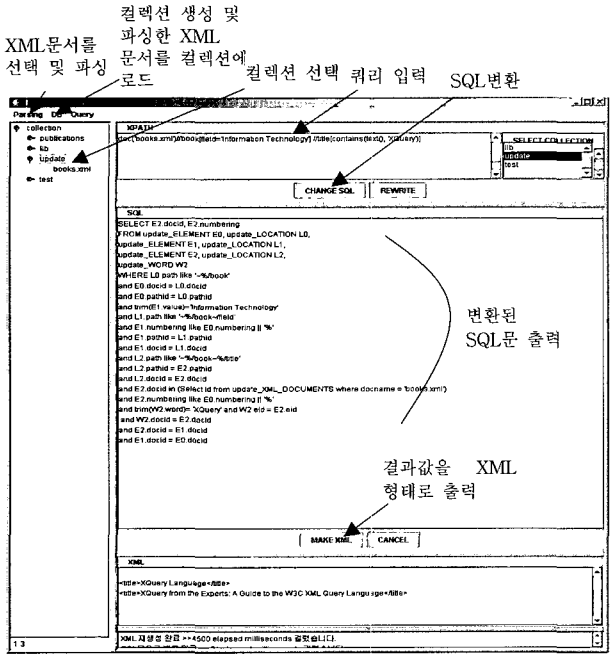
conctenation_table(t.name, t.dewey_n, t.value,,,,,,t.sibord, a.row_id)

5) item노드의 자식으로 출현할 서브트리들을 text_table로부터 검색하되 attr_table과 카티션 프로덕트 연산을 한다. 그리하여 4)과정에서 생성한 item노드와 동일한 개수의 서브트리를 생성하고 concatenation 테이블과 UNION All 연산을 한 뒤 최종 결과뷰를 생성한다. 단 아래의 최종 결과뷰는 서로 동일한 dewey_n컬럼값을 가지게 되는데 이때 row_id 컬럼과 함께 정렬함으로써 결과 뷰를 XML 형태로 재생성할 때 XML문서에 내재된 순서대로 데이터를 로드할 수 있다.

last_table(docid, eid, name, dewey_n, value, info, row_id)

6. 실행 및 분석

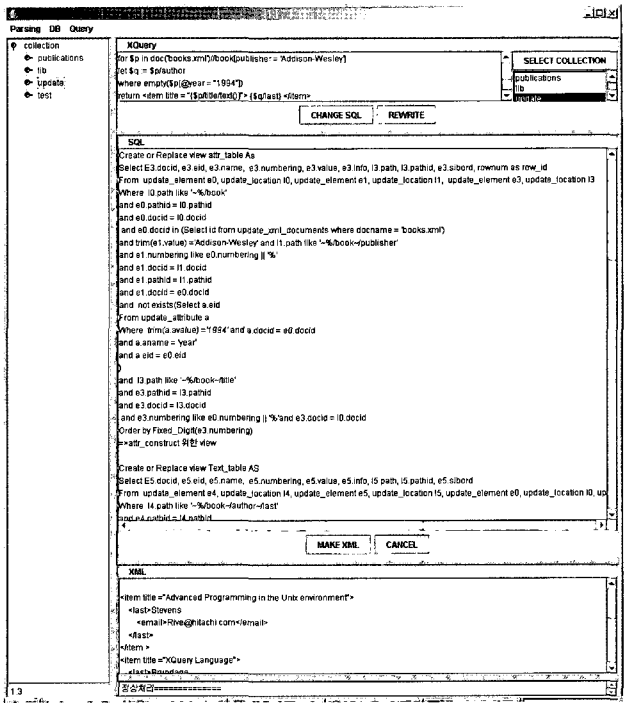
다음은 본 연구에서 제안하는 XPERT 시스템의 XPath 및 XQuery 실행화면이다.



(그림 11) XPath 실행화면

위의 XPath 실행 화면에서 볼 수 있듯이 컬렉션을 생성 및 XML문서를 파싱하여 로드할 수 있으며 이를 선택하여 질의를 입력하고 처리한다. 그리고 질의한 결과 값을 보기 좋게 XML 형태로 출력하여 사용자에게 반환한다.

아래의 화면은 XQuery FLWOR식을 실행한 화면이다.



(그림 12) XQuery 실행화면

7. 결론 및 향후 연구 방향

XML의 사용이 급격히 늘어남에 따라 XML을 효과적으로 관리하는 데이터베이스 기술의 요구가 늘어나게 되었다. 이런 산업체의 요구사항을 만족시키기 위하여 XML 전용 DBMS와 XML 처리 기능을 보완한 DBMS가 발표되었으며 XML을 처리하기 위한 많은 연구가 진행되고 있다. 특히 현재 구축된 데이터베이스를 그대로 활용할 수 있으며 또 기술적으로 가장 안정 되었고, 상업적으로 가장 성공한 관계형 DBMS를 이용하여 XML을 관리하는 많은 기술들이 연구되고 있다. 본 논문의 연구도 이러한 XML 관리 시루 추세에 따라 관계형 DBMS를 이용하여 XML 질의어 XQuery를 처리하는 XPERT 시트템을 실제 구현하면서 구현 과정의 처음부터 끝까지 과정의 다양한 기술들을 설명하고 있으며 특히 실제적인 문제들에 대한 해결 방법을 제공하고 있다.

XPERT 시스템은 논리적인 XML 파일의 모임을 컬렉션으로 선택하고 모든 연산들이 컬렉션 단위로 이루어지게 하고 있으며 스키마의 구성은 XML 성분에 따라 다른 테이블들을 구성하여 대부분의 XPath 연산과 XQuery의 전문 검색의 기본적인 기능까지 실행 가능하게 하였다. 또 효율적인 검색과 계층 정보의 비교를 위해 경로와 순서 정보 번호

를 적절하게 사용하였으며, XQuery를 SQL로 변환하는 방법도 먼저 XQuery를 AST로 구성하고, AST를 SQL로 변환하는 실제적인 형식으로 설계, 구현하였다. 그 외에 XML을 분석하는 방법도 DOM 방법과 SAX 방법을 같이 제공하여 다양한 크기의 XML 파일이 지원될 수 있게 했다.

본 논문에서 설명한 XPERT 시스템은 관계형 DBMS에 세 새로운 내용을 요구하는 것이 아니라 관계형 DBMS 위에 새로운 계층을 만들어 기존에 구축된 데이터를 변경 없이 사용할 수 있으므로 XML과 관계형 응용 모두에 적용될 수 있는 장점을 가지고 있다.

부록. SQL_Creator(Cname, AST node, output_node's name) 알고리즘

*alias := -1, path := '', from := 'FROM', where := 'WHERE', doc_SQL := ''
*output_name := output_node's name, output_num = -1
*basic_from := Cname_element e || alias, Cname_location l || alias
*basic_where := l || alias .path like 'path'
and e || alias.docid = l || alias.docid
and e || alias.pathid = l || alias.pathid

1) step 타입: Stack.push(노드명)

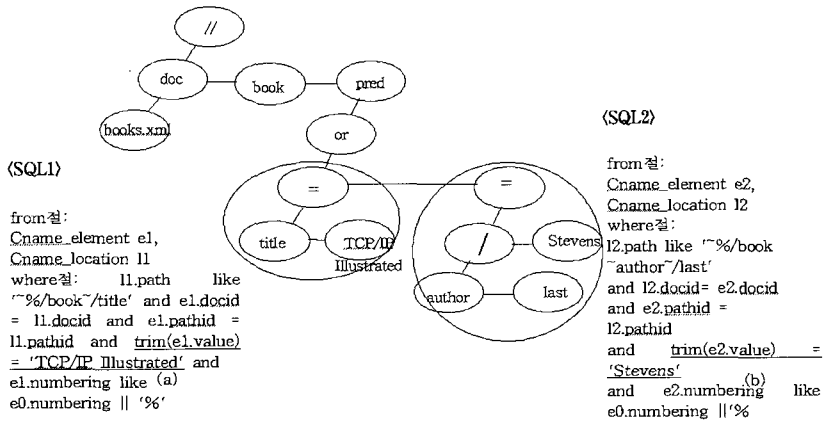
2) QName타입:
- alias++;
- path 생성
- if(stack.pop() == '/') path := '~/' || 노드명
- if(stack.pop() == '//') path := '~%/' || 노드명
- OutputNodeSeek알고리즘에서 찾은 output노드mink와 동일한 노드인지를 검사
- if(노드명 == output_name && stack.top == 0)
{ output_name := null
output_num := alias
from := basic_from
where := basic_where || 'e || alias.docid in'|| doc_SQL'}

3) Pred 타입
- if(output_name != null)
{from := basic_from, where := basic_where}
- 숫자타입의 자식노드를 가질 경우 예를 들어 book[2]와 같은 XPath식을 AST로 변환하면 Pred타입의 노드는 2라는 숫자타입의 노드를 자식으로 가진다. 이때 Dewey_n 인코딩 기법으로 book 노드들을 정렬한 뒤 인라인 뷰를 생성한다.
- from :='(select e || alias.eid, e || alias.docid, rank() over(order by fix_digits(e || alias.numbering)) rank, || from || where) inline_view'
- where := 'inline_view.rank = 자식 노드명'

4) text타입
- from := basic_from
- where:= basic_where || 'trim(e || alias.value) = 노드명'
- if(alias > 0)
- 만약 (그림 5)를 예로 들 경우 현재의 위치가 ♣에 있다면 이를 텍스트로 가지는 field 엘리먼트가 book의 자손인지를 검사하는 아래의 구문을 추가한다.
- where := 'e || alias.numbering like e || alias-1. numbering || '%"

5) 함수 타입
- if(노드명 == 'contains')
{from := 'word w || alias'
where := 'trim(w || alias.word) = text타입의 자손 노드명'
|| 'and w || alias.eid = e || alias.eid and w || alias.docid = e || alias.docid)}
- if(노드명 == 'doc')
- 사용자가 선택한 컬렉션에서 특정 XML문서를 검색한다
- doc_SQL :='(Select id from Cname_xml_documents where docname = 자식노드명)'

6) 논리 연산자
- and 연산자만 존재할 경우 이를 무시하고 1)~5) 과정을 그대로 수행
- or 연산자만 존재할 경우
- or 타입의 노드의 하위 노드들을 '=' 타입의 노드를 기준으로 서브트리로 분리한다. 그리고 이를 순회하면서 생성한 SQL문 중 텍스트를 검색하는 구문만을'('로 묶은 뒤 SQL의 or연산으로 처리한다. 예를 들면 그림 부록_1의 AST는 책 제목이 TCP/IP Illustrated 이거나 Stevens라는 last 이름을 가진 사람이 쓴 책을 검색하는 XPath를 표현한 것이며 <SQL1>과 <SQL2>는 or 타입의 노드의 각 서브트리를 순회하면서 생성한 SQL문을 가리킨다. 이중 텍스트를 검색하는 (a)와 (b)구문만을'('로 묶어 SQL의 or연산을 수행한다.



(그림-부록 1) XQuery의 'or', 'and'연산의 처리

- where := <SQL1>의 (a)구문 || 'or' || <SQL2>의 (b)구문
- 아래의 예시와 같이 and, or연산자가 모두 존재할 경우 위와 마찬가지로 각 서브트리를 순회하면서 SQL문 <SQL1>, <SQL2>, <SQL3>를 생성하고 이들을 '('로 묶어 차례대로 SQL의 and, or 연산으로 처리한다.

<SQL1>
ex3) doc('books.xml')//book[price='39.95' and
<SQL2> <SQL3>
author/last = 'Buneman' or author/first ='W.']

- where := <SQL1> || 'and' || <SQL2> || 'or' || <SQL3>

참 고 문 헌

[1] Igor Tatarinov, Stratis D. Viglas, Kevin Bayer, J. Shanmugasundaram, Eugene Shekita and C. Zhang, "Storing and Querying Ordered XML Using a relational database system" in ACM SIGMOD June, 2002.

[2] D. Hong, K. Kim, "Update conscious Inverted indexes for XML queries in relational databases" in Lecture Notes in Computer Science #3180 pp.263-272 Springer-Verlag, August, 2004.

[3] Dare Obasanjo, "A proposal for an XML Data Definition and Manipulation Language" in Lecture Notes in Computer Science #2590 Springer-Verlag, 2003.

[4] M Yoshikawa et al, "XRel: A Path-Based Approach to storage and retrieval of XML document using relational databases" in ACM Transactions on Internet Technology, August, 2001.

[5] J. Shanmugasumdaram et al, "Relational Databases for Querying XML document: Limitations and Opportunities" in Proceedings of the 25th VLDB Conference, 1999.

[6] D. Dehaan, D. Toman. M. Consens, and M. Tamer Ozsu, "A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding" in ACM SIGMOD, San Diego CA, June, 2003.

[7] W.Meier, "eXist: An Open Source Native XML Database," Web, Web Services, and Database System, Germany, Springer LNCS Series, 2002

[8] Torsten Grust, "Accelerating XPath Location Steps" in ACM SIGMOD, Wisconsin, June, 2002.

[9] C. Zhang, Jeffery Naughtom, D. DeWitt, Qiong Luo, and Guy Lohman, "On supporting Containment Queries in Relational Database Management Systems" in ACM SIGMOD, Santa Barbara May, 2001.

[10] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATH: Insert-Friendly XML Node Labels" in ACM SIGMOD, Paris, France 2004.

[11] Torsten Grust, Sherif Sakr, and Jens Teubner "XQuery on SQL Hosts" in Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004.

정 민 경

email : skallet@kmu.ac.kr
2003년 계명대학교 회계학과(학사)
2006년 계명대학교 컴퓨터공학과(석사)
관심분야 : RDBMS, XQuery, Multimedia Database

홍 동 권

email : dkhong@kmu.ac.kr
1985년 경북대학교 전자공학과(학사)
1992년 University of Florida 전자계산학과(석사)
1995년 University of Florida 전자계산학과(박사)
1985년~1990년 한국전자통신연구원
1996년~1997년 한국전자통신연구원
2004년 정보처리학회논문지D, 제11-D권 제 3호(6월)
1997년~현재 계명대학교 공학부 컴퓨터공학 전공 부교수
관심분야 : 능동 실시간 데이터베이스, 병렬 처리, 멀티미디어 처리, XML 데이터베이스