한국지능시스템학회 논문지 2008, Vol. 18, No. 2, pp. 278-283

XQuery FLWOR 연산의 SQL 변환 기법 분석

Analysis of XQuery FLWOR expression to SQL translation

홍동권
Dong_Kweon Hong
계명대학교 컴퓨터공학과

요 약

인터넷의 활용이 웹 1.0, 웹 2.0으로 점점 더 활성화 되어감에 따라 XML의 사용도 점점 더 가속화 되고 있다. XML의 사용에 있어 효과적인 검색 기능은 필수적인 요소이며 XML 질의어의 사실상 표준인 W3C XQuery의 활용도 점점 늘어가는 추세이다. XQuery는 비정형적 문서와 데이터를 다루는 XML의 특성으로 인하여 질의어로서는 매우 복잡한 구조와 다양한 기능을 가지고 있다. XQuery의 가장 대표적인 구문은 for, let, where, order by, return의 기능을 나타내는 FLWOR 구문으로 XQuery 처리 시스템에서 가장 중요한 역할을 한다. 본 논문은 XQuery FLWOR 연산을 관계형 환경에서 처리하는 방법과 FLWOR 연산의 형식에 따라 그 방법들의 기능과 정확성을 증명 확인한다.

키워드 : XML, XQuery, FLWOR 연산, SQL 변환

Abstract

As the usefulness of internet is kept changing more productively with web 1.0, web 2.0 usage of XML is also increasing very rapidly. In XML environment the most critical function is the ability of effective retrieval of useful information from XML repository. That makes the W3C XQuery more popular. XQuery has very complicated structure as a query language due to the semi_structured nature of XML. FLOWOR , which stand for for, let. where, order by, return, is the most commonly used expression in XQuery. In this paper we suggest the methods to handle XQuery FLWOR on relational environments. We also analyze and evaluate our approach to prove its correctness.

Key Words : XML, XQuery, FLWOR expression, SQL translation

1. 서 론

웹 2.0의 등장으로 인터넷의 사용은 단순 정보의 검색에서 사용자들이 원하는 정보를 얼마나 빠르게 찾아서 실제로 사용할 수 있느냐 하는 사용자 중심으로 그 패러다임이 바뀌고 있다. 또 공급자 중심에서 사용자 중심으로 바뀌면서 다양한 형식의 데이터가 사용자로부터 제공됨에 따라 표준화된 검색 환경이 절실히 요구되고 있다. XML의 사용은 웹의 표준화된 데이터 형식으로 그 사용이 급격히 늘어나 XML 데이터의 양도 빠른 속도로 늘어나고 있다. 따라서 대용량의 XML 데이터에서 원하는 정보를 쉽게 찾아내기 위한 표준화 노력으로 XML의 질의어인 W3C XQuery가 2007년 사실상 표준으로 인정받게 됨에 따라 XQuery를 효과적으로 지원하기 위한 연구가 더 가속화 되고 있다. XQuery 1.0은 2007년 1월 23일에 W3C의 권고안(recommendation)으로 확정됨에 따라 사실상 그 표준안이 완성되었다 [1]. XQuery는 선언적 기능과 반복적 기능을 동시에 포함하고 있는데 선언적 기능은 XPath로 반복적 기능은 FLWOR 연산으로 나타낸다. FLWOR 연산은 for, let, where, order by, return으로 구성되어 있으며 꽃을 의미하는 영어 단어 flower로 발음한다.

FLWOR 연산은 비정형적인 XML의 특성으로 인하여 기존의 정형적인 관계형 테이블을 다루는 SQL보다 훨씬 더 복잡하고 많은 기능을 포함하고 있다. 특히 계층적 반복 기능과 return 구문의 엘리먼트 생성 기능은 기존의 관계형 SQL에 포함되어 있지 않는 연산으로 관계형 구현하기에 많은 어려움을 제공한다. 특히 엘리먼트의 생성 기능은 관계형으로 구성할 경우 중간 결과를 만드는 복잡함을 겪어야 한다. 본 논문의 구성은 다음과 같다. 2장에서는 관계 연구로 XQuery FLOWR 연산에 대해서 설명하며, 3장은 XQuery의 실행 환경에 대해 설명한다. 4장은 XQuery를 구문 분석한 후 AST(Abstract Syntax Tree)로 만들고, AST를 SQL로 변환하는 방법과 XQuery FLWOR 연산의 다양한 기능을 SQL로 변환하는 방법에 대한 분석 및 이론적인 평가를 제시하고, 마지막으로 5장에서는 결론 및 향후 연구 방향에 대해서 언급한다.

2. 관련 연구

XQuery에 대한 다양한 문서와 표준, 구현 등의 관계 내용은 W3C에서 제공하는 http://www.w3.org/XML/Query에서 발견할 수 있으며 효과적인 XQuery의 구현에 많은

접수일자 : 2007년 9월 6일
완료일자 : 2008년 2월 4일

산업체가 참여하고 있는 것을 알 수 있다. 하지만 지금까지 구축되어 있던 기존의 관계형을 포기하지는 못하므로 XQuery를 지원하는 제품들에는 전용 데이터베이스를 위한 연구와 제품도 많이 있지만 기존의 관계형 데이터베이스를 이용한 제품과 연구도 활발히 진행되고 있다 [2, 3, 4, 5, 7, 8]. 특히 Ipedo 회사의 XIP는 SQL 엔진과 XQuery 엔진을 동시에 제공하는 듀얼 엔진 방법을 채택하고 있으며, IBM의 DB2 Viper는 XQuery를 지원하고 있다. 또 Abacus의 제품도 관계형 데이터와 XML을 동시에 지원하는 방법을 사용하고 있다 [1]. 이런 추세로 볼 때 관계형 데이터와 XML 데이터를 동시에 지원하는 것은 당분간 활발히 진행될 것으로 예측되며 관계형을 활용한 XQuery의 지원 방법도 활발히 연구될 것으로 예측된다.

XQuery FLWOR 연산의 SQL 변환은 반복, 재구성(Construction) 등의 기존 관계형 질의 언어인 SQL에서 제공하지 않는 복잡한 기능으로 인하여 매우 복잡한 과정이 필요하다 [3, 5, 6]. 논문 [3]은 동적 인터벌 인코딩 기법과 여러 개의 뷰를 사용하는 방법으로 FLWOR 연산의 반복 및 재구성 기능을 SQL로 변환하는 방법을 소개하고 있다. 하지만 앞에서 언급한 것과 같이 XML을 관계형 테이블에 저장하는 방법에 따라 XQuery를 SQL로 변환하는 방법은 전혀 다른 변환 방법을 사용하고 있으며 각 시스템이 XQuery를 지원하는 정도에 따라 테이블의 구성, 변환 방법이 많은 차이점을 보이고 있다.

3. XQuery 실행 환경

3.1 구성 환경

XQuery 실행 환경은 [그림 1]과 같이 관계형 데이터베이스의 상위 계층으로 만들어지는 환경을 사용한다[2].

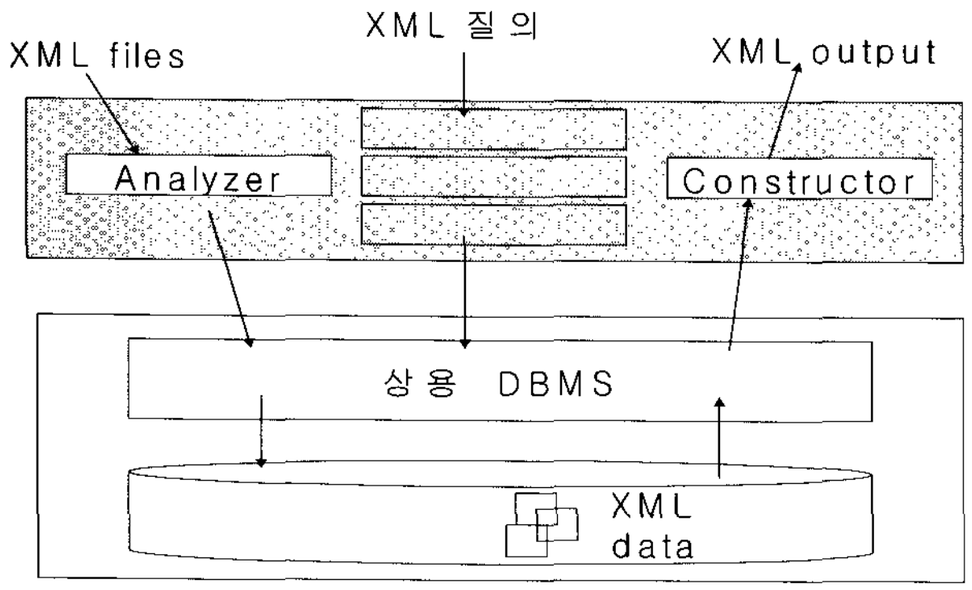

그림 1. XQuery 질의 처리 시스템 구성도
Fig. 1.XQuery query processing system

3.2. 테이블 구성

XML 저장 테이블의 구성은 [그림 2]와 같다. 전체 시스템에는 Collections 테이블이 1개 존재하며, 각각의 컬렉션에는 Cname_xml, Cname_location, Cname_element, Cname_attribute, Cname_word 테이블이 존재한다 [2]. 테이블 구성에서 Cname은 컬렉션 이름을 나타내며 각각의 데이터들은 논리적으로 컬렉션으로 나누어지게 된다.

Collections (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. XML 저장 테이블 데이터베이스 스키마
Fig. 2. Database schema for XML table

4. XQuery의 SQL 변환 알고리즘

4.1 XPath의 SQL 변환

사용자가 XQuery의 XPath를 입력할 경우 먼저 이를 파싱하고 구문을 분석하여 AST(Abstract Syntax Tree)를 만든다. AST 트리를 구성하는 모든 노드들은 각각 자신들만의 타입을 가지며 입력되는 XQuery의 유형에 따라 트리의 형태가 조금씩 달라진다. SQL로의 변환 방식은 AST를 깊이우선 이동(depth first traverse) 방식을 사용하면서 SQL변환 알고리즘을 각 노드 타입에 맞게 적용하여 최종 SQL문을 생성한다 [2]. 변환 알고리즘을 적용하여 생성된 최종 SQL문은 다음의 [그림 3]과 같다.

질문: Information Technology분야의 서적 중 XQuery와 관련된 책들의 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)))

```
SELECT E2.docid, E2.dewey_n
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.dewey_n like E0.dewey_n || '%'
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.dewey_n like E0.dewey_n || '%'
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
```

그림 3. XPath의 SQL 변환 결과
Fig. 3. Results of XPath to SQL translation

XPath 연산이 엘리먼트와 경로를 사용하므로 [그림 3]의 SQL에서는 element, location 2개의 테이블과 그리고 contains() 함수의 사용을 지원하기 위하여 word 테이블을 사용하였다. element 테이블의 doc_id, e_id, 그리고 location 테이블의 doc_id, path_id 결합 인덱스는 조인에 사용되므로 B-tree 인덱스를 사용하며, 경로를 확인하는 path와 dewey_n 필드는 like 연산을 사용한 selection 연산을 사용하고 있다. [그림 2]의 테이블은 XQuery에서 사용하는 13개의 축 연산을 지원하는데 사용된다[1, 4]. 그 중에서 애트리뷰트 축은 따로 attribute 테이블을 사용하여 지원하며 나머지 연산들은 SQL의 와일드 문자를 이용한 스트링 매칭 기능으로 지원한다.

정리 1: location 테이블의 path 정보는 XQuery의 축(Axis) 연산중에서 반드시 지원되기를 요구하는 자식(child), 부모(parent), 자손(descendants) 연산을 지원한다.

증명: 현재 내용 노드(current context node)를 A라고 가정하면 SQL의 스트링 패턴 매칭에서 와일드 문자를 이용하여 축 연산을 지원한다. 여기서 '~' 기호는 자식을 나타내기 위하여 경로를 저장할 때 미리 저장해 놓은 특수 문자이다. 각각의 노드는 다음의 SQL 연산에 의해서 구해진다.

자식 노드 '~%/A~%/'
자손 노드 '~%/A%/'
부모 노드 '~%/A'

4.2 XQuery FLWOR 연산의 SQL 변환

XQuery FLWOR 식을 SQL로 변환하는 방법은 앞의 방법과 같이 먼저 표현식을 AST 형태로 변환한 뒤 각 서브트리별로 XPath 변환 알고리즘을 적용하여 SQL문을 생성한다 [2]. 변환 방법에서 엘리먼트 구성이 있는 경우와 없는 경우의 처리 방법이 다르므로 2가지의 경우를 나누어서 고려한다.

4.2.1 엘리먼트 구성이 없는 변환

```
**가격이 39.95$인 책 제목이 'Web'이라는 단어를 포함 할 경우 그 책을 쓴 작가의 이메일 주소를 출력하라.
(Element construction이 없고 binding 변수가 2개)
for $p in doc('books.xml')//book[price = '39.95']
let $q := $p/author
where $p/title[contains(text(), 'Web')]
return $q//email
```

그림 4. 엘리먼트 구성이 없는 FLWOR 연산의 예제
Fig. 4. FLWOR expression without element construction

for, let, where의 각각의 절에 앞의 XPath 변환 알고리즘을 적용하여 만들어진 SQL 문장을 같이 결합하여 만든다. price의 값이 39.95인 모든 book 엘리먼트를 찾기 위하여 element 테이블을 사용하고, 경로를 확인하기 위하여 location 테이블을 사용한다. 그 다음 book 엘리먼트의 자식으로 존재하는 author 엘리먼트를 찾는다. where와 return은 다시 앞의 결과를 필터링하는 역할을 한다. 따라서 for, let, where, return에서 생성된 SQL 구문의 조건을 서로 단순 결합함으로써 변환 알고리즘에 의한 최종 SQL 구문이 [그림 5]와 같이 생성된다.

[그림 5]는 변환 알고리즘에 의해 자동 생성된 SQL 문장이다. XPath와 같은 방법으로 각각의 AST 서브트리에 대해서 SQL 조건을 독립적으로 만들어 결합하는 방법을 사용하여 변환 결과 SQL을 자세히 살펴보면 자체조인(self-join)이 많이 존재한다.

```
Select E5.docid, e5.eid, e5.name, e5.value
From ttt_element e5, ttt_location l5, ttt_element e4, ttt_location l4, ttt_element e0, ttt_location l0, ttt_element e1, ttt_location l1,ttt_element e2, ttt_location l2, ttt_element e3, ttt_location l3, ttt_word W3
Where e5.dewey_n like e4.dewey_n || '%'
and e5.docid = e4.docid
and e5.pathid = l5.pathid
and e5.docid = l5.docid
and e4.docid = e2.docid
and e4.dewey_n like e2.dewey_n || '%'
and l4.path like '~%/book~/author~%/email'
and e4.docid = l4.docid
and e4.pathid = l4.pathid
and l0.path like '~%/book'
and e0.pathid = l0.pathid
and e0.docid = l0.docid
and e0.docid in (Select id from ttt_xml_documents where docname = 'books.xml')
and trim(e1.value) ='39.95' and l1.path like '~%/book~/price'
and e1.dewey_n like e0.dewey_n || '%'
and e1.docid = l1.docid
and e1.pathid = l1.pathid
and e1.docid = e0.docid
and l2.path like '~%/book~/author'
and e2.pathid = l2.pathid
and e2.docid = l2.docid
and e2.dewey_n like e0.dewey_n ||'%'and e2.docid = e0.docid
and l3.path like '~%/book~/title'
and e3.pathid = l3.pathid
and e3.docid = l3.docid
and trim(W3.word)= 'Web'
and W3.eid = E3.eid
and W3.docid = E3.docid
```

그림 5 엘리먼트 구성이 없는 FLWOR 연산의 SQL 변환 결과
Fig. 5. Translation of FLWOR to SQL without element construction

정의 1: 각 엘리먼트가 가지는 dewey 번호는 각 문서에서 고유한 값을 가진다 [7].

정의 2: 각 엘리먼트가 가지는 dewey 번호를 이용하면 각 엘리먼트의 부모, 자식, 자손 관계를 확인할 수 있다 [7].

정리 2: [그림 5]의 SQL 연산은 [그림 4]의 FLWOR 연산의 기능을 정확히 수행한다.

증명: [그림 4]의 FLWOR 연산은 XML 문서에서 [그림 6]의 구조와 값을 가진 부분들을 정확히 찾는 것을 보임으로써 그 정확성을 보인다.

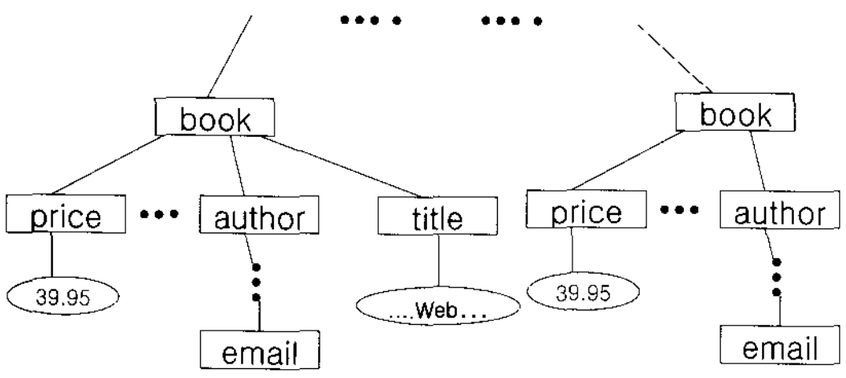

그림 6. XML 문서에서 질의 결과
Fig. 6 Results of XML query

질의 내용이 정확하기 위해서는 정확한 값을 가진 엘리먼트인지, 그 다음 정확한 경로에 존재하는지, 그리고 서로 연관된 경로인지를 확인한다. [그림 5]의 결과 SQL에는 2개의 값과 5개의 경로를 검사하는 부분이 포함되어 있다. 5개의 경로는 '~%/book', '~%/book/~price', '~%/book~/author', '~%/book~/title', '~%/book~/author~%/email' 스트링으로 표현되어 검사된다. 5개의 경로는 [그림 6]에 존재하는 모든 경로를 포함한다. 그 다음 각 경로들이 [그림 6]의 각각의 book 엘리먼트를 중심으로 서로 연결된 경로라는 것을 확인하여야 한다. 각 경로들이 문서의 다른 부분에 존재하는 것이 아니라는 것을 확인하기 위한 방법으로 경로 사이의 연관성을 확인한다. 연관성 확인은 위의 경로에 존재하는 엘리먼트를 구하고 정리 2와 정리 3의 내용을 활용하여 각 엘리먼트들의 dewey_n 값을 검사하여 경로에 존재하는 엘리먼트들이 자식, 부모, 자손 등으로 실제 연관된 엘리먼트라는 것을 확인한다. 따라서 [그림 5]에 생성된 SQL 구문은 [그림 4]의 FLWOR 연산을 정확히 실행하는 것임을 확인할 수 있다. □

즉 부모, 자식, 자손 등으로 이루어진 계층적인 구조와 값이 동시에 질의의 조건에 만족하는 데이터를 검색하는 이중의 어려움을 나타낸다. SQL이 기본적으로 계층적인 연산을 지원하지 않으므로 XML 문서가 가지는 계층적인 구조는 데이터에 인코딩 될 수밖에 없다.

4.2.2 엘리먼트 구성을 포함한 변환

엘리먼트 구성을 포함한 FLOWR 연산의 경우 단일 SQL 구문으로 처리하기는 어렵다. 몇 개의 임시 결과를 만들고, 그 결과들을 스트링 결합하여 엘리먼트 구성을 구현한다. 그 과정을 살펴보면 다음과 같다 [2].

1) AST 트리를 순회하되 constructor 노드(item)의 attribute값으로 들어갈 엘리먼트를 만나면 현재까지 AST를 순회하면서 생성한 SQL문을 적절히 결합하여 attr_table 이라는 뷰를 생성한다.
   attr_table(docid, eid, name, dewey_n, value, info, path, pathid, sibord, row_id)
2) constructor 노드(item)의 text값으로 들어갈 엘리먼트를 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)과정을 수행했을 때는 AST 트리의 순회를 완료한 것이며 3)단계 부터는 AST의 트리와는 상관없이 단일 루프를 포함한 XQuery FLWOR 식이라면 모두 일률적으로 거치게 되는 단계이다. 2)단계에서 생성한 text_table 뷰를 for문에 출현하는 노드로 그룹핑하여 그중 최고 root노드만을 골라 group_view를 생성한다.
   group_view(child_n)
4) 3)과정에서 생성한 group_view에 rownum값과 인코딩된 Dewey_n값을 추출하여 group_seq라는 뷰를 생성한다. 이때 각각의 컬럼명을 row_id, child_n 이라한다.
   group_seq(child_n, row_id)
5) 4)과정에서 생성한 group_seq라는 뷰의 child_dewey_n 컬럼값과 동일한 값을 가진 엘리먼트들을 text_table에서 그리고 row_id컬럼과 동일한 값을 가진 엘리먼트를 attr_table에서 검색한 뒤 이들을 1:1 결합하여 concatenation_table이라는 뷰를 생성한다.
   concatenation_table(t.name, t.dewey_n, t.value,,,,,t.sibord, a.row_id)
6) concatenation_table과 text_table을 결합하고 중복된 값을 제거하여 최종 결과뷰를 생성한다. 여기서 중복된 값이란 concatenation_table과 동일한 eid를 가진 text_table의 행들이다.
   last_table(docid, eid, name, dewey_n, value, info)

위의 과정에 따라 [그림 7]의 FLWOR 연산을 SQL 구문으로 변환하는 과정은 [그림 8]과 [그림 12]에 나타나 있다. 변환 된 SQL 구문을 모의실험과 샘플을 이용한 확인은 이미 이루어졌다 [2]. 따라서 여기서는 이론적인 정확성을 확인한다.

```
**books.xml 문서중 'Web' 키워드를 포함하는 책의 출판사와 작가의 last 이름을 출력하라.
(Element construction이 있고 binding 변수가 2개)
for $p in doc('books.xml')//book
let $q := $p/publisher
where $p/title[contains(text(), 'Web')]
return <item public = "{$q/text()}"> {$p//last} </item>
```

그림 7. return에 엘리먼트 구성을 포함하고 있는 FLWOR 연산
Fig. 7 FLWOR return with element construction

```
create or replace view attr_table as
select e1.docid,e1.eid, e1.name, e1.dewey_n, e1.value, l1.path, rownum as row_id
from lib_element e0, lib_location l0, lib_element e1, lib_location l1
where l0.path like '~%/book'
and e0.docid = l0.docid
and e0.pathid = l0.pathid
and l1.path like '~%/book~/publisher'
and e1.docid = l1.docid
and e1.pathid = l1.pathid
and e0.docid = e1.docid
and e1.dewey_n like e0.dewey_n || '%'
and e1.docid in (Select id from lib_xml_documents where docname = 'books.xml')
order by Fixed_Digit(E1.dewey_n);
```

그림 8. 애트리뷰트로 삽입될 내용들
Fig. 8 Contents of attribute

```
create or replace view text_table as
select e3.docid, e3.value, e3.dewey_n, l3.path,
e3.name, e3.eid
from lib_element E1, lib_element E2, lib_element
E3,lib_element E0,
lib_location l1, lib_location l3, lib_location l2,
lib_location l0, lib_word w1
where l0.path like '~%/book'
and e0.pathid = l0.pathid
and e0.docid = l0.docid
and e1.dewey_n like e0.dewey_n || '%'
and l1.path like '~%/book~/title'
and e1.pathid = l1.pathid
and e1.docid = l1.docid
and trim(w1.word) = 'Web'
and w1.eid = e1.eid
and w1.docid = e1.docid
and l2.path like '~%/book~%/last'
and e2.pathid = l2.pathid
and e2.docid = l2.docid
and e0.docid = e1.docid
and e1.docid = e2.docid
and e3.docid = e2.docid
and e2.dewey like e0.dewey_n || '%'
and e3.dewey_n like e2.dewey_n || '%'
and e3.pathid = l3.pathid
and e3.docid = l3.docid
and e1.docid in (Select id from auc_xml_documents
where docname = 'books.xml');
```

그림 9. 엘리먼트 구성에 텍스트로 삽입될 내용
Fig. 9. Text contents for element construction

```
create or replace view group_view as
select MIN(Fixed_Digit(e2.dewey_n)) as
child_dewey_n
from Text_table e2, lib_location l0, lib_element e0,
lib_element e1, lib_location l1, auc_word w1
where l0.path like '~%/book'
and l0.pathid = e0.pathid
and l0.docid = e0.docid
and e1.dewey_n like e0.dewey_n || '%'
and l1.path like '~%/book~/title'
and e1.pathid = l1.pathid
and e1.docid = l1.docid
and trim(w1.word) = 'Web'
and w1.eid = e1.eid
and w1.docid = e1.docid
and e2.dewey_n like e0.dewey_n || '%'
and e2.docid = e0.docid
and e1.docid = e0.docid
order by e0.dewey_n;
```

그림 10. group_view의 생성
Fig. 10. Creation of group_view

```
Create or Replace view Concatenation_table As
Select '<item public ="'|| E0.value ||'">' ||E1.name
as name, E1.dewey_n, E1.value, E1.eid,
E1.path, E0.row_id
FROM Text_table E1, Attr_table E0,
    Group_view E2
WHERE Fixed_Digit(E1.dewey_n) like
E2.child_dewey_n;
```

그림 11. concatenation_table의 생성
Fig. 11. Creation of concatenation_table

```
Create or replace view last_table As
Select E1.eid, E1.name, E1.dewey_n, E1.value,
E2.row_id
from Text_table E1, Concatenation_table E2,
lib_element E0, lib_location l0
where l0.path like '~%/book' and E1.eid !=
E2.eid
and l0.pathid = E0.pathid
and E1.dewey_n like E0.dewey_n || '%'
and E2.dewey_n like E0.dewey_n || '%'
and e1.docid = e0.docid
and e2.docid = e0.docid
UNION All
Select E3.eid, E3.name, E3.dewey_n, E3.value,
E3.row_id from Concatenation_table E3;
```

그림 12. last_table의 생성
Fig. 12. Creation of last_table

[그림 7]의 FLWOR 연산은 XML 문서에서 [그림 13]의 계층적인 구조와 값을 가진 부분을 찾아서 그 결과를 새로 구성해서 반환한다.

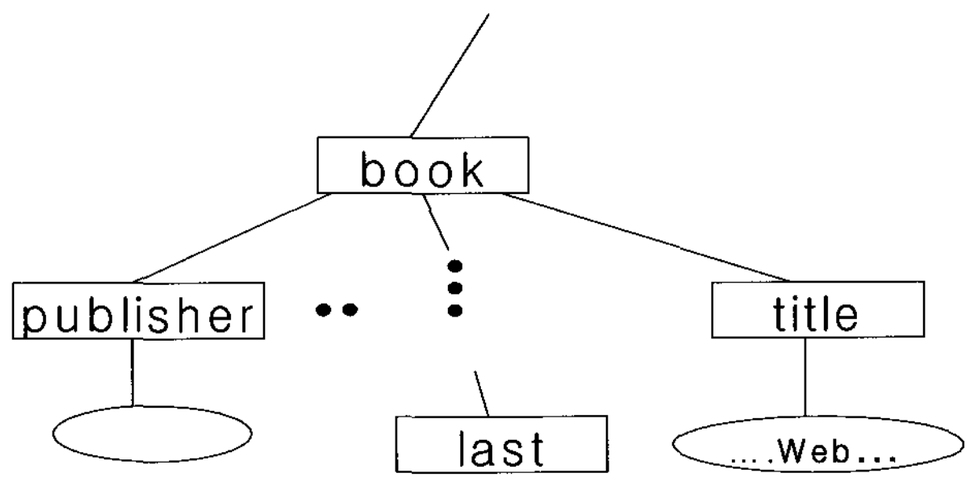

그림 13. FLWOR 질의에서 찾으려는 계층과 값
Fig. 13. Level and value for FLWOR expression

결과의 일부분은 애트리뷰트로 또 일부분은 엘리먼트의 텍스트로 구성해서 반환하므로 적어도 2개 이상의 SQL 구문이 필요하며 또 결과를 합치기 위한 방법이 필요하다.

증명: [그림 8]에서 [그림 12]의 SQL 연산들은 [그림 7]의 FLWOR 연산 기능을 수행한다. 애트리뷰트는 'publisher'의 내용이 필요하므로 [그림 8]에서는 '~%/book'과 '~%/book~/publisher' 경로와 dewey_n을 이용하여 애트리

뷰트에 삽입될 내용을 구한다. [그림 9]에서는 엘리먼트 구성에 텍스트가 사용될 경우 텍스트를 검색하기 위한 SQL 구문이다. 이때 검색된 내용에는 엘리먼트 중첩이 있을 수 있다. 즉 [그림 13]에서 검색된 'last'의 경우 하위 계층에 많은 내용이 존재할 수 있다. 이 문제를 해결하기 위하여 [그림 10]에서는 각 텍스트의 최상위 루트만 찾는 과정을 거친다. [그림 11]에서는 최상위 루트가 애트리뷰트의 내용과 1:1 연결이 되는 과정을 보여준다. 마지막으로 [그림 12]에서는 텍스트의 하위 계층이 연결되는 과정을 설명한다. □

5. 결론 및 향후 연구 방향

2007년에 XQuery의 최종 표준안(Recommendation)이 결정됨에 따라 XML Query의 형식이 결정되었다. XQuery의 질의 기능은 관계형 SQL의 기능을 훨씬 넘어선다. 특히 SQL의 'select from where'에 해당하는 FLWOR 연산의 기능은 일반 프로그램 코드 수십 줄을 대신하는 강력한 기능을 가지고 있다. 본 논문은 관계형 DBMS를 이용한 XQuery의 지원 방법으로 XQuery를 SQL로 변환 하는 방법의 정확성을 확인하였다. 먼저 XML을 분할 저장하는 테이블의 스키마가 XQuery를 지원하기에 충분한지를 확인하였으며, XPath 연산의 SQL 변환이 그 기능을 정확히 수행하는지 확인하였다. 마지막으로 XQuery의 FLWOR 연산을 SQL 구문으로 변환하는 과정을 검증하기 위하여 경로 정보와 dewey 번호를 이용하는 방법의 정당성을 증명하였다. 또 XQuery FLWOR 연산에서 엘리먼트 구성이 없는 경우와 엘리먼트 구성이 있는 경우 그 변환 과정이 서로 다른 이유를 설명하였다. XQuery의 가장 흔한 연산들이 관계형 DBMS의 기능을 사용하여 정확한 기능을 수행하는 SQL로 변환 가능함을 보임으로써 단순 가능성에서 벗어나 변환 과정이 구체적이며 정확함을 보여주었다.

참 고 문 헌

[1] www.w3c.org

[2] 홍동권, 정민경 "XSTAR: XML 질의의 SQL 변환 알고리즘" *퍼지 및 지능시스템 학회 논문지* Vol 17, No. 3, 2007.

[3] 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.

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

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

[6] 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.

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

[8] 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.

[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.

저 자 소 개

홍동권(Dong-Kweon Hong)

1985년 : 경북대학교 전자과 공학사.
1992년 : U. of Florida 컴퓨터공학 석사
1995년 : U. of Florida 컴퓨터공학 박사.
1997년 ~ 현재 : 계명대학교 컴퓨터공학과 교수

관심분야 : XML, 데이터베이스, 질의 최적화

Phone : 053-580-5281
Fax : 053-580-5165
E-mail : dkong@kmu.ac.kr