XQL-SQL 질의 변환을 통한 XQL 질의 처리 시스템의 설계 및 구현

김 천 식[†]·김 경 원[††]·이 지 훈[††]·장 복 선[††]·손 기 락[†††]

요 약

XML이 웹 문서의 표준이며, 문서교환용 언어로서 사용되고있다. 상업용 데이터는 대부분 관계형 데이터베이스에 저장되어 있고 이들 문서를 교환용 문서로 만들어서 문서교환에 이용하거나 관계형 데이터베이스에 저장된 XML데이터에 XQL로 질의하여 질의결과를 효율적으로 획득하는 것은 매우 중요하다. 따라서, 향후 많은 XML데이터의 보관 및 관리 그리고 XML데이터를 위한 질의어 처리는 필수적이다. 지금까지, XML데이터의 저장 및 검색과 관련한 연구 및 제품개발이 여러 업체에 의해 있어왔고, 지금도 연구 및 개발이 진행되고 있다. 하지만, 효율적인 XML데이터의 저장 및 검색을 위한 시스템은 아직까지 많지 않다. 따라서 본 논문에서는 효율적인 경로 질의를 위한 DFS-Numbering 방식을 사용하며, 효율적인 데이터 저장을 위해서 XML 데이터 저장을 위한 스키마를 설계하였다. 또한, 전통적인 관계형 데이터베이스 엔진을 이용한 효율적인 XQL 질의수행 방법을 설계 및 구현하였다. 즉, 사용자가 시스템에 XQL로 질의를 하면 XQL 처리기에 의해서 XQL이 SQL로 변환되고, SQL로 관계형 데이터베이스에 질의를 수행하면, 결과로 레코드를 반환한다. 이때 XML 생성기에 의해서 사용자에게 XML문서를 반환한다.

Design and Implementation of XQL Query Processing System Using XQL-SQL Query Translation

Chunsig Kim[†]·Kyungwon Kim[††]·Jihun Lee[††]
Boksun Jang[††]·Kirack Sohn[†††]

ABSTRACT

XML is a standard format of web data and is currently used as a prevailing language for exchanging data. Most of the commercial data are stored in a relational database. It is quite important to convert these conventionally stored data into those for exchange and use them in data exchange, or to get the query results effectively by utilizing XQL on XML data which are store in a relational database. Thus, it is absolutely required to have a proper query processing mechanism for XML data and to maintain many XML data properly. Up to now, many cases of researches on the storage and retrieval of XML data have been carried out and under study. But, effective retrieval and storage system for path queries like XQL has yet to be contrived. Thus, in this paper, a schema to store XML data is designed, in which DFS-Numbegering method is used to store data effectively. And an effective path query processing method is also designed and implemented, in which a traditional relational database engine is used. That is, XQL is converted into SQL with a XQL processor if a user makes query XQL in a system. A database system executes SQL, and a XML generator uses a generated record and makes a XML document.

키워드 : Query, XQL, XML, DFS-numbering, 관계형 데이터베이스(Relational Database)

1. 서 론

XML[1]은 인터넷 기반의 상업 어플리케이션을 위한 표준 데이터 포맷으로 등장하였다. 대부분의 상업적인 데이터는 전통적인 관계형 데이터베이스에 일반적인 저장형식으로 저장되었다. 이들은 상업적인 데이터를 문서교환에 활용하기 위해서는 XML로의 변환이 필수적이다. 하지만 관계형 데이터베이스와 XML문서는 형식에 있어서 관계형 데이터베이스는 테이블 형태의 데이터라면 XML문서는 계층적인 트리형태이거나 아니면 그래프형태의 데이터로 존재한다. 이런 데이터 모델 차이를 해결하기 위해서 SilkRoute[2], XPERANTO[3] 등의 미들웨어 시스템이 제안되었다.

이들 시스템은 전통적인 관계형 데이터베이스에 이미 저장된 데이터를 활용하자는 측면에서 매우 가치가 있지만, 이미 만들어진 XML문서가 있고 이들 문서가 복잡한 그래프 구조로 존재한다면 관계형 데이터베이스에 XML문서의 저장은 간단한 문제가 아니다.

† 준 회 원 : 경동대학교 정보통신공학부 교수
†† 정 회 원 : 한국외국어대학교 대학원 컴퓨터 및 정보통신공학과
††† 정 회 원 : 한국외국어대학교 컴퓨터 및 정보통신공학부 교수
논문접수 : 2002년 4월 18일, 심사완료 : 2002년 7월 12일

이와 같은 XML문서의 저장에 관하여 *DFS-Numbering*[4] 방법이 제안되었다. 이 방법은 XML문서가 트리 형태일 경우에 관계형 데이터베이스에 문서를 저장할 때 XML문서의 각 요소에 번호를 부여하여 트리 형태로 저장하는 것이다. 저장된 문서를 추출할 때 DFS 순행을 통하여 원문을 그대로 복구할 수 있다는 장점을 갖고있는 방법이다.

또 다른 문서방법으로는 XML문서 전체를 데이터베이스의 필드에 BLOB형태로 저장하는 방법과 XML문서의 데이터 구조인 DTD[1]를 관계형 데이터베이스와 일대일로 맵핑하여 저장하는 방법 등이 지금까지 제안된 방법이다.

본 논문에서는 임의의 XML문서를 *DFS-Numbering* 방법을 이용하여 관계형 데이터베이스에 저장하고 저장된 문서를 XQL[5]로 질의하여 질의결과를 획득하는 시스템을 제안한다. 본 논문에서 제안한 시스템은 어떠한 관계형 데이터베이스 상에서도 작동가능 하도록 개발하였다.

XQL을 질의 언어로 선택한 이유는 XQuery[7]가 SQL과 같이 섬세한 질의를 처리하도록 개발된 질의언어라는 점에서 이 언어를 질의언어로 선택하는 것이 당연할 것이다. 또한 이 언어가 표현할 수 있는 범주는 매우 크며 또한 가치가 있다. 게다가, XQuery는 경로질의 처리를 지원한다. 하지만, 구현과 질의 사용에 있어서 XPATH[6]보다는 복잡하다. 따라서, 경로 정보의 추출 및 정보 필터링에 XPATH를 사용하는 연구가 증가하고 있는 만큼 XPATH의 모체인 XQL을 질의언어로서 사용하는 것은 매우 의미가 있을 것으로 판단한다.

XPATH를 사용하는 경우에 대한 예로, XML 문서 필터링을 위해서 Continuous Query(CQ)를 이용한 시스템에서 XPATH 질의를 이용하여 필터링을 수행하고 있다.

본 연구에서 XQL을 이용한 질의방법은 시스템에 XQL이 입력으로 들어오면 SQL로 변환하여 SQL로 관계형 데이터베이스에 질의하여 질의 결과를 사용자에게 돌려주는 방식이다. [9]에서는 질의처리를 위한 인덱싱과 정규 경로표현을 위한 질의 알고리즘에 대한 측면이라면 본 연구에서는 상업적인 목적을 위해 전통적인 관계형 데이터베이스의 검증된 DBMS를 이용함으로써 실행 성능을 높이고, 사용자가 이해하고 배우기 쉬운 XQL로 효과적인 질의를 하도록 시스템을 설계 및 구현한다.

2. 관련 연구

관계형 데이터베이스에 XML 문서를 저장하고, 저장된 문서에 경로표현을 제공하는 질의언어로서, 질의를 하게되면 만족하는 경로를 표현하고 수행하기 위해서는 문서 트리의 길이만큼을 조인해야 하는 단점을 갖고 있다. 이 방법은 트리의 길이에 비례하여 성능이 떨어진다는 것을 나타내는 것이다.

본 논문에서와 같이 트리를 이용하는 방법으로서 과거에 제안된 [8]에서는 정규경로 표현을 위해서 성능이 좋지 않았다. 왜냐하면 경로의 길이가 길어지거나 복잡한 질의에 있어서는 트리 운행에 많은 시간이 소요된다. [4]에서는 *DFS-Numbering* 방식을 이용하도록 저장되고, 검색하는 방법으로서 기본적인 질의만 지원되었다. [9]에서는 [4]의 단점인 기본적인 질의처리를 개선하여 복잡한 질의 처리를 위한 여러 조인알고리즘을 제안하였다. 또한 [8]에서는 XML 및 반 구조적인 데이터 처리를 위한 OEM(Object Exchange Model)을 이용한 XML 및 반 구조적인 데이터를 저장하기 위한 전용 데이터베이스를 개발하였다. [8]의 연구는 그래프 형태의 데이터를 복잡한 구조적인 데이터를 저장하기에 적합하지만 복잡한 그래프 구조에서 전통적인 트리 운행방법을 사용한 접근법을 사용함으로써 경로 길이가 길어질 경우에 정규 경로표현을 통한 실행 성능은 비효율적이다. [9]에서는 [4, 8]의 단점인 트리 검색성능과 XQuery와 같은 복잡한 질의를 지원하기 위한 정규 경로 표현을 제공한다.

본 논문의 연구와 [9]의 연구를 비교하면 다음과 같다.

[9]에서의 연구와 본 연구에서의 유사점은 Numbering 기법을 사용한 다는 점이다. Numbering은 요소와 속성 그리고 그 밖의 노드에 대해서 번호 매김을 한다는 것이다. 본 논문에서는 (그림 5)와 같이 순서대로 번호 매김을 하고 [9]에서는 <order, size> 형태로 번호 매김을 한다. 본 연구에서는 부모-자식 관계를 알아내는 방법은 4.2절에서 설명하였다.

[9]에서의 연구와 본 논문의 연구방법의 차이점은 다음과 같다.

- [9]에서는 B+ 트리로 구현하였고, 본 연구에서는 상업적인 DBMS를 사용하였다.
- [9]에서는 Numbering 형태가 <order, size>이지만 본 연구에서는 부모 요소는 <시작태그 번호, 끝태그 번호>를 의미한다. 따라서 자식요소에 대한 경로 질의가 들어오면 시작태그와 끝 태그사이의 태그번호만 조회하면 쉽게 조회가 가능하다. [9]에서는 별도의 조회를 위한 조인 알고리즘을 사용한다.
- [9]에서는 질의를 위해 Path Join 알고리즘을 사용하며, 세부적으로는 요소와 속성간의 관계에서 결과를 추출하는 알고리즘, 요소와 요소간의 관계에서 결과를 추출하는 알고리즘, 정규 경로 표현을 위한 알고리즘으로 구성되며 된다. 즉, 하나의 질의 표현을 작은 단위로 나누어 중간결과를 임시로 보관하고 임시 보관된 자료간의 또 다른 표현을 처리하는 단계로 질의를 처리한다. 하지만 본 연구에서는 XQL질의를 SQL문법에 맞도록 모든 표현을 바꾸어 한번에 처리하므로 중간결과는 필요 없다.

본 논문에서 XQL언어를 질의어로 사용하는 만큼 XQL을 질의 특성별로 분류한 결과 다음과 같은 분류를 갖고 있음을 알 수 있었다.

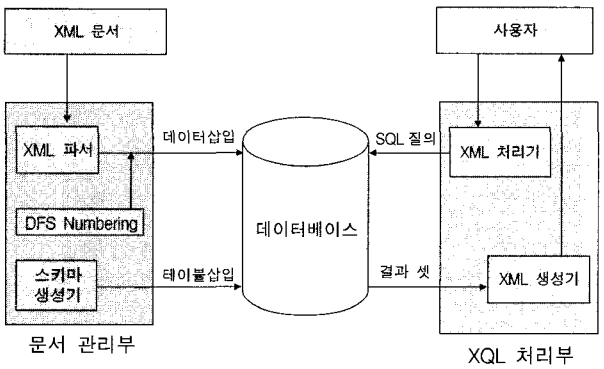
- Collection
- Selection children and descendants expression
- Filter expression
- Boolean expression
- Equivalence and comparison expression
- Comparisons and Literals
- Union and Intersection expression

본 논문에서 제안한 시스템에서는 XQL에서 제안한 질의 범주에 해당하는 질의 연산을 제공하고 있다. 본 논문의 기여는 제안된 알고리즘에 의해서 정규경로 표현을 XML 데이터에 대한 트리 운행(Navigation) 없이 처리할 수 있다는 점과 XQL에서 제안한 일부 함수를 제외한 대부분의 연산을 제공한다는 점과 XQL의 경로 질의를 위하여 XQL을 SQL로 변환하여 효율적인 질의가 가능하도록 알고리즘을 고안한 것이다.

3. XML문서 저장 시스템 설계

3.1 시스템의 전체 구조

본 시스템의 구조는 (그림 1)과 같다. (그림 1)은 크게 문서관리 부분과 XQL처리부분, 이렇게 두 가지 부분으로 나누어 볼 수 있는데, 문서관리부는 XML형태로 구성되어 있는 문서를 파싱후 *DFS-Numbering* 인덱스를 걸어서 데이터베이스에 저장하는 부분이고, XQL 처리부분은 사용자가 입력한 XQL형태의 질의를 SQL형태로 변환하여, 저장된 XML문서를 검색하는 부분이다.

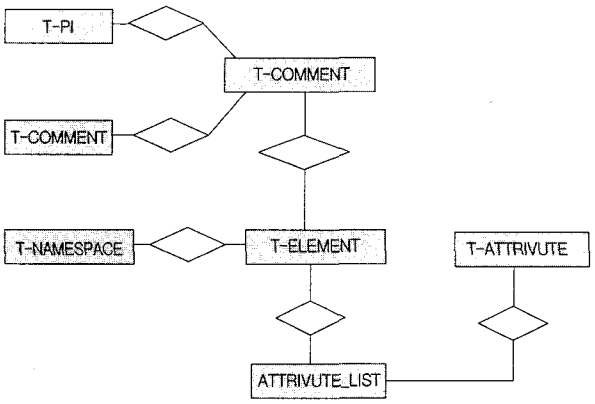


(그림 1) XQL 질의 시스템 구조

3.2 데이터베이스 저장스키마

(그림 2)는 XML문서 저장시스템을 위한 E-R 다이어그램으로 나타낸 것이다. T_DOCUMENT에 T_PI와 T_COMMENT, T_ELEMENT의 관계는 T_DOCUMENT에 연결된 각 객체가 T_DOCUMENT에 소속됨을 나타낸다. T_ATTRIBUTE는 T_ATTRIBUTE_LIST를 통해서 T_ELEMENT에 포함됨을 나타내는 관계를 보여주고 있다.

이렇게 총 7개의 테이블로 구성되어 있다. XML문서가 입력되면, 파싱을 한 다음 각각의 엘리먼트별로 *DFS-Numbering* 인덱스를 걸어서, 데이터베이스에 저장한다.



(그림 2) XQL스키마를 위한 E-R모델링

각 테이블별로 저장되는 필드의 내용에 대해서 살펴보면 다음과 같다.

<표 1>은 문서의 처리 명령(PI : Processing Instruction) 문을 저장하기 위한 테이블이다. doc_id는 T_DOCUMENT를 참조하여 Document의 PI를 나타내는데 이용된다. target는 <?target text ?>의 형식으로 존재하는 PI의 표적을 나타낸다. data는 <?target text?>의 형식으로 존재하는 PI의 텍스트를 나타낸다.

〈표 1〉 T_PI 테이블

이 름	타 입	참조 및 설명
doc_id	number	documentID, PK, reference T_DOCUMENT
target	varchar2(40)	PI target, PK
data	varchar2(2000)	PI data

<표 2>는 문서 내의 주석문을 저장하기 위한 테이블이다. doc_id는 T_DOCUMENT를 참조한다. content는 주석 내용을 저장하기 위한 필드이름이다.

〈표 2〉 T_COMMENT 테이블

이 름	타 입	참조 및 설명
id	number	commentID, PK
doc_id	number	documentID, PK, reference T_DOCUMENT
content	varchar2(2000)	Comment내용

<표 3> Document에 관한 정보들을 저장하기 위한 테이블이다. doc_type은 문서안에 DocumentType노드를 나타낸다. committed필드는 절 4.4에서 설명한다. id 필드는 문서를 유일하게 식별하는 식별자로 사용된다.

〈표 3〉 T_DOCUMENT 테이블

이 름	타 입	참조 및 설명
id	number	문서ID, pk
doc_type	varchar2(40)	문서의 document type
committed	char(1)	문서의 Commit여부

<표 4>는 요소(Element)의 기본적인 정보들과 *DFS-Numbering* 인덱스에 필요한 데이터들을 저장하기 위한 테이블이다. id는 요소(Element)의 식별자이고, doc_id는 T_DOCUMENT를 참조한다. tstart는 요소의 시작 태그번호이고 tend는 요소의 끝 태그번호이고, depth는 요소의 깊이를 나타내고, name은 요소의 이름을 나타내고, content는 요소의 내용을 나타내고, Content_long은 내용이 2Kbyte이상일 경우 저장하기 위한 필드이다. Prefix_id는 Namespace의 prefix로 T_DOCUMENT를 참조한다.

〈표 4〉 T_ELEMENT 테이블

이 름	타 입	참조 및 설명
id	number	elmentID, PK
doc_id	number	문서ID, reference T_DOCUMENT
tstart	number	element시작 tag number
tend	number	element끝 tag number
depth	number	element level
name	varchar2(40)	element이름
content	varchar2(2048)	element
Content_long	long	element내용(2 Kbyte 이상일 경우
Prefix_id	number	Namespace의 prefix, reference T_DOCUMENT
parent	number	T_ELEMENT간의 조인

<표 5>는 속성(Attribute)과 요소(Element)를 연결하기 위한 테이블이다. element_id는 T_ELEMENT를 참조하고, att_id는 T_ATTRIBUTE를 참조한다. 요소(Element)의 속성 값을 저장하기 위해서 attr_value필드를 사용한다.

〈표 5〉 ATTRIBUTE_LIST 테이블

이 름	타 입	참조 및 설명
elment_id	number	elementID, PK, reference T_ELEMENT
attr_id	number	attributeID, PK, reference T_ATTRIBUTE
attr_value	varchar2(2000)	attribute값
prepix_id	number	Namespace 21 prefix reference

<표 6>은 속성의 이름이 저장되는 테이블이다. att_id는 T_ATTRIBUTE의 식별자이다. attr_name은 요소(Element)의 이름을 위한 필드이다.

〈표 6〉 T_ATTRIBUTE 테이블

이 름	타 입	참조 및 설명
att_id	number	attributeID, PK
attr_name	varchar2(40)	attribute 내용

<표 7>은 Namespace를 저장하여, T_ATTRIBUTE와 연결하기 위한 테이블이다.

〈표 7〉 T_NAMESPACE 테이블

이 름	타 입	참조 및 설명
id	number	namespaceID, PK
prefix_name	varchar2(40)	namespace의 prefix name
url	varchar2(200)	namespace의 URL

4. 시스템 구현

4.1 XML문서의 삽입

XML문서를 SAX[10] 표준 파서를 이용하여 파싱한다. 요소별로 파싱한 결과를 관계형 데이터베이스에 저장하기 전에 *DFS-Numbering* 기법을 이용하여 요소(Element)의 태그별로 번호 매김(Numbering)한다. 이 기법은 Document의 모든 태그를 나오는 순서대로 번호 매김(Numbering)하여 요소별로 "시작태그(Start Tag)"와 "끝태그(End Tag)"의 번호와 Level을 번호로 표현하여 관계형 데이터베이스 테이블에 요소별로 삽입한다. 이 기법을 이용하면 요소들의 포함관계를 쉽게 파악할 수 있다.

본 논문에서 제안한 시스템은 XML문서의 구조적인 정보를 저장하기 위하여 *DFS-Numbering* 기법과 트리 레벨을 사용하였다. (그림 3)의 DTD에 부합하는 (그림 4)의 XML문서를 SAX파서로 읽어들이고 이벤트가 발생하는 순서대로 위에서 제안한 *DFS-Numbering*을 하게되면 (그림 5)와 같이 XML문서의 구조는 트리 구조를 형성하게 된다. 이러한 트리 구조를 표현하기 위한 방법으로 *DFS-Numbering*을 이용한 방문순서쌍을 이용한다.

요소 트리의 루트 노드로부터 DFS방식으로 각 노드를 방문하여 처음 방문할 때의 순서와 자신의 자식 노드의 방문이 모두 끝난 뒤에 노드의 방문순서의 1쌍으로 구성된다.

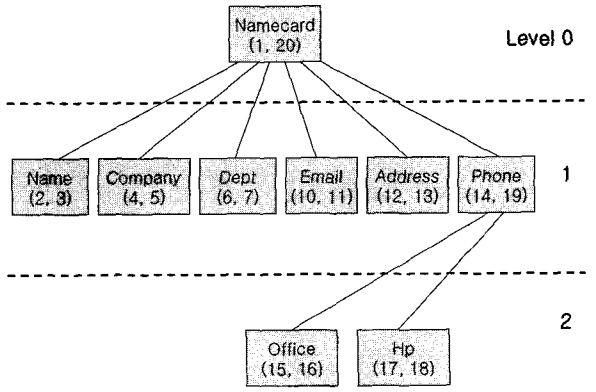
뿐만 아니라, 트리의 level정보를 저장함으로써 문서의 트리 구조상에서 특정 노드에 속하는 서브 트리에 대한 질의를 가능하게 한다.

```
<?xml version = "1.0" encoding = "iso-8859-1"?>
<!DOCTYPE namecard [
<!ELEMENT namecard ( name, company, dept, title, email,
                     address, phone) >
<!ELEMENT name ( #PCDATA ) >
<!ATTLIST name eng CDATA #REQUIRED >
<!ELEMENT company ( #PCDATA ) >
<!ELEMENT dept ( #PCDATA ) >
<!ELEMENT title ( #PCDATA ) >
<!ELEMENT email ( #PCDATA ) >
<!ELEMENT address ( #PCDATA ) >
<!ELEMENT phone ( office, hp ) >
<!ELEMENT office ( #PCDATA ) >
<!ELEMENT hp ( #PCDATA ) >
```

(그림 3) namecard.dtd

```
<namecard>
    <name eng = "Hong Gil-Dong"> 홍길동 </name>
    <company> XML회사 </company>
    <dept> 기술연구소 </dept>
    <title> 사원 </title>
    <email> hong@hufs.ac.kr </email>
    <address> 서울시 강남구 신사동 </address>
    <phone>
        <office> 02-3015-3771 </office>
        <hp> 016-353-7316 </hp>
    </phone>
</namecard>
```

(그림 4) namecard.xml



(그림 5) 문서에 대한 DFS-Numbering

4.2 XQL질의를 SQL질의로 변환 방법

XQL 표현(Expression)을 처리하기 위해서는 먼저 XQL 표현을 파싱 해주는 파서가 필요하다. 본 논문에서는 XQL 표준 BNF을 JavaCC라는 도구를 이용하여 파서의 틀을 구성하였다.

XQL 표현을 입력받으면 JavaCC로 구현된 파서가 XQL 표현을 파싱(Parsing)하여 토큰과 요소들을 구분해 낸다. 파싱된 XQL 표현을 스택을 이용하고, XQL 표현의 파싱이 모두 끝나면 스택에서 토큰으로 구성된 연산자에 따라서 스택의 내용을 pop하여 해당 연산자에 대하여 SQL 부분 문자열로 변환한 후, 다시 스택에 push한다. 이런 방법을 계속 반복하여 스택에 하나의 요소가 남을 때까지 반복하면 남은 요소는 모든 XQL 표현을 SQL 문자열로 변환된 형태로 가지게된다.

<표 8>은 스택에 저장될 데이터의 구조를 나타낸 것이다.

질의가 다음과 같을 때 SQL로 변환하는 일반적인 방법은 다음과 같다.

namecard/phone

위와 같이 namecard밑에 존재하는 phone이 있으면 phone의 하위에 존재하는 모든 엘리먼트 정보를 보이라는 질의이다. 이때 namecard과 phone에 관한 정보는 T_ELEMENT에 저장되어 있다. 따라서 Self-Join(id, doc_id)을 통해서 부모 자식관계를 파악한다. T_ELEMENT에는 tstart, tend 속성이 있는데 이를 이용하면, (그림 5)와 같이 name은 tstart가 1이고 tend가 20임을 알 수 있다. phone은 tstart가 14이고 tend가 19이므로 phone 요소의 하위에 존재하는 데이터는 tstart가 14보가 크고 tend가 19보다 작은 데이터가 name 밑에 phone데이터에 해당됨을 알 수 있다. 따라서 T_ ELEMENT를 Self-Join하고 phone의 tstart보다 크고 phone의 tend보다 작은 조건에 해당하는 데이터가 위의 질의에 해당하는 정보가 되는 것이다.

질의가 다음과 같을 때 SQL로 변환하는 일반적인 방법은 다음과 같다.

namecard/*/office

위와 같이 경로의 표현이 부모 자식관계가 아니라 부모 손자관계나 그 이상의 관계일 경우에는 T_ELEMENT에 있는 depth속성이 사용된다. (그림 5)에서 namecard는 level이 0이고 office는 level이 2이다. 따라서 경로를 모두 표현할 경우 SQL로 변환하여 자료를 탐색하는 시간이 증가할 것이다. 따라서 이 경우 depth정보를 이용하여 office가 level 2이고 depth가 3이므로 level 1단계를 탐색하지 않고 level 2를 탐색할 수 있으므로 검색의 효율성을 높일 수 있다.

질의가 다음과 같은 경우 SQL로 변환하는 방법은 다음과 같다.

namecard//office

위와 같이 경로의 표현에 //이 사용된 경우는 임의의 깊이를 갖는 office 요소를 찾아 하위에 존재하는 모든 데이터를 보이라는 질의이다. 이 경우는 임의의 깊이를 탐색해야 하므로 결국 namecard 요소로부터 office 요소에 이르는 모든 요소를 검색해야만 한다. 따라서 T_ELEMENT 간의 Self-Join과 tstart와 tend속성을 이용한 검색방법이 사용된다.

〈표 8〉 스택에 저장될 요소(Element)의 구조

m_Kind	요소 타입 (Table, Element, Attribute, Comparison)
m_Content	요소의 Content 내용
M_TableList	SQL문자열의 From절에 들어갈 테이블 명 리스트
M_Query	SQL 문자열의 Where절에 들어갈 연산 내용
M_Falias	SQL 문자열의 From절에 들어갈 테이블의 첫 Alias명 변환된 SQL 부분 문자열들을 연결하기 위해서 사용된다.
M_Lalias	SQL 문자열의 From절에 들어갈 테이블의 마지막 Alias명 변환된 SQL의 부분 문자열들을 연결하기 위해서 사용된다.
and_or	변환된 SQL 부분문자열에 and 나 or 명령의 사용여부
withUnion	변환된 SQL 부분문자열에 union이나 intersect명령의 사용여부
m_inFunction	method사용시의 method종류를 표시
not_in	변환된 SQL 부분 문자열에 not명령의 사용여부
m_MKind	Lvalue이 Any = 0, All = 1
m_CompareOP	비교에 필요한 연산자
m_LValue	비교의 Left Value
m_RValue	비교의 오른쪽 값

XQL 표현을 하나의 SQL 문자열로 변환하는 알고리즘(그림 6)은 다음과 같다.

1. XQL 표현(Expression)별로 파싱한다.
2. 파싱된 표현을 스택에 Push한다.
3. 1과 2를 XQL 표현이 끝날 때까지 반복한다.
4. 스택에서 저장된 표현들을 Pop한다.
5. 표현에 해당하는 SQL 부분 문자열로 변환·조합하여 스택에 Push한다.
6. 모든 XQL 표현이 SQL 문자열로 변환될 때까지 4, 5를 반복한다.
7. 최종적으로 변환·조합된 SQL 문자열을 실제 RDB에 질의한다.
8. 질의결과를 다시 XML 문서형태로 재 조합한다.

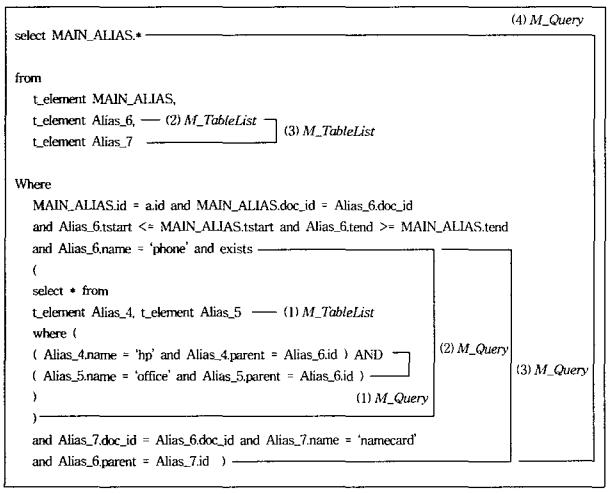
(그림 6) XQL을 SQL로 변환하기 위한 알고리즘

(그림 6)의 SQL을 XQL로 변환하기 위한 알고리즘을 예제 질의로서 설명하고자 한다.

질의 1 : namecard/phone[hp and office]

(질의 1)은 namecard밑에 phone이 존재하고 phone 밑에 hp와 office가 동시에 존재하는 phone요소의 트리구조를 추출하도록 하는 질의이다. (질의 1)를 (그림 7)의 SQL로 변환하는데 다음의 4단계의 과정의 알고리즘을 이용하여 만들어진다.

단계 1 : hp and office : 알고리즘 ASTAND(그림 8)
단계 2 : phone[hp and office] :
알고리즘 ASTSubscript(그림 9)
단계 3 : namecard/phone[hp and office] :
알고리즘 ASTRelativePath(그림 10)
단계 4 : SQL 부분문자열을 완전한 SQL문자열로 조합한다. : 알고리즘 ASTQuery(그림 11)



(그림 7) (질의 1)의 결과로 생성되는 SQL

(그림 7)은 단계별로 M_TableList<표 8>와 M_Query <표 8>에 들어가는 내용을 최종적으로 생성되는 SQL부분문자열로 표현한 것이다.

알고리즘 1 : ASTAND

```
01 : RNode = stack.pop ( ) ; // office정보
02 : Lnode = stack.pop ( ) ; // hp정보
03 : Switch ( Lnode.kind ( )) { // Lnode의 타입
04 :  Case : E_ELEMENT :
05 :   Switch ( Rnode.kind ( )) {
06 :    Case : E_ELEMENT :
07 :     New_alias1 = 새로운 Table_Alias 생성 ;
08 :     Lnode.setM_Falias ( New_alias1 ) ;
09 :     New_alias2 = 새로운 Table_Alias 생성 ;
10 :     Lnode.setM_Lalias ( New_alias2 ) ;
11 :    m_table = "t_element" + New_alias1 + ", t_element " + New_alias2 ;
12 :    m_query = " ( " + New_alias1 + ".name = " + Lnode.Content( )
                 + " and " New_alias1 + ".parent = PATH_ALIAS.id ) "
                 + " AND ( " + New_alias2 + ".name = "
                 + Rnode.Content ( ) + " and " New_alias2
                 + " .parent = PATH_ALIAS.id ) " ;
13 :    Lnode.setand_or ( true ) ;
14 :    Lnode.setM_TableList ( m_table ) ;
15 :    Lnode.setM_Query ( m_query ) ;
16 :    Case : E_COMPARISON : …
17 :    Case : E_ATTRIBUTE :
18 :    Case : E_TABLE :
19 : Case : E_COMPARISON :
20 : Case : E_ATTRIBUTE
21 : Case : E_TABLE :
22 : }
23 : Lnode.setKind ( E_TABLE ) ;
24 : stack.push ( Lnode ) ; // Lnode를 스택에 저장한다.
```

(그림 8) ASTAND

스택에는 (질의 1)에 XQL질의를 JavaCC로 파싱한 결과 각 요소가 스택에 name, phone, hp, office 순서대로 저장되어 있다. (그림 8)줄 1, 2의 Rnode, Lnode는 <표 8>의 구조로 정의된다.

(그림 8)의 줄 1의 Rnode는 스택의 pop연산을 수행한 후 *office* 요소정보가 할당된다. 줄 2의 Lnode는 스택의 pop연산을 수행한 후 *hp*요소정보가 할당된다. 줄 3은 Lnode의 타입 종류에 따라서 적절한 연산을 수행한다. Lnode의 타입은 E_ELEMENT(엘리먼트 타입), E_COMPARISON(비교연산자), E_ATTRIBUTE(속성 타입), E_TABLE(엘리먼트간의 결합 연산결과 타입)이 된다.

줄 3의 평가 결과 Lnode타입이 E_ELEMENT이므로 줄 5~줄 18까지를 수행한다. 줄 5의 Rnode의 타입이 E_ELEMENT이므로 줄 7~줄 12까지를 수행한다.

줄 7의 New_alias1에 SQL의 FROM절에 요소(Element) 테이블의 alias를 생성한다. 여기에서 *Alias_4*가 New_alias1에 할당된다. 줄 8은 선행참조를 위해서 setM_Falias를 이용하여 Lnode에 할당한다.

줄 9의 New_alias 2에는 새로운 Table_Alias인 *Alias_5*가 할당된다. 줄 10은 후행참조를 위해서 setM_Alias를 수행한다. 그 결과 Lnode에는 New_alias2의 값이 할당된다.

줄 11의 m_table은 SQL의 FROM절 이하에 오는 문자열을 구성하기 위한 구문이다. m_table은 "t_element", New_alias1, "t_element", New_alias2를 모두 더한다. 결과로 m_table에는 "t_element Alias_4, t_element Alias_5"가 할당된다.

줄 12의 m_query는 (그림 7)의 (1)을 구성하는 것으로서 SQL의 WHERE절을 구성한다. 줄12에 Lnode.Content()는 Lnode에 저장된 태그이름을 반환한다. Rnode.Content()는 Rnode에 저장된 태그이름을 반환한다. m_query에 저장된 내용은 결국 "(Alias_4.name = 'hp' and Alias_4.parent = PATH_ALIAS.id) AND) (Alias_5.name = 'office' and Alias_5.parent = PATH_ALIAS.id)"가 된다. 줄 12는 결국 관계형 데이터베이스에서 트리경로를 만들어 XQL질의를 처리하기 위해서 Self-Join 형태의 질의 구조를 만든다.

줄 13은 and나 or연산자가 사용된 경우에 Lnode를 true로 설정한다. 줄 14는 setM_TableList를 이용하여 m_table에 저장된 문자열을 Lnode에 설정한다. 줄 15는 setM_Query를 이용하여 m_query를 Lnode에 설정한다.

줄 23의 Lnode의 타입을 E_TABLE로 설정한다. 줄 24는 Lnode를 스택에 push한다. 지금까지 변환된 모습을 (그림 7)의 (1)에서 확인할 수 있다.

알고리즘 2 : ASTSubscript

```
01 : Rnode = stack.pop ( ) ;
02 : Lnode = stack.pop ( ) ;
03 : Switch ( Rnode.kind ( )) {
04 :  Case : E_ELEMENT : …
05 :  Case : E_COMPARISON : …
06 :  Case : E_ATTRIBUTE :
07 :  Case : E_TABLE :
08 :    tmp_alias = Rnode.getM_Lalias ( ) ;
09 :    new_alias = 새로운 Table_Alias 생성 ;
10 :    Lnode.setM_Falias ( new_alias ) ;
11 :    Lnode.setM_Lalias ( new_alias ) ;
12 :    M_TableList = " t_element " + new_alias ;
13 :    M_Query = new_alias + ".name = " + Lnode.Content ( ) + " " ;
14 :    if ( Rnode.NumNot_in ( ) != 0 )
15 :      if ( Rnode.NumNot_in ( ) > 1 )
16 :        M_Query = M_Query + " and ( " + Rnode.getM_Query ( ) + " ) " ;
17 :      else
18 :        M_Query = M_Query + " and " + Rnode.getM_Query ( ) ;
19 :    else if ( Rnode.beand_or ( ))
20 :      M_Query = M_Query + " and exists ( select * from "
                     + Rnode.getM_TableList ( ) + " where ( "
                     + Rnode.getM_Query ( ) + " ) ) " ;
21 :    else {
22 :      M_Query = M_Query + " , " + Rnode.getM_TableList ( ) ;
23 :      M_Query = M_Query + " and " + tmp_alias + ".doc_id = "
                     + new_alias + ".doc_id and " + Rnode.getM_Query ( )
                     + " and " + tmp_alias + " .parent = " + new_alias
                     + " .id " ;
24 : }
25 : M_Query에 있는 PATH_ALIAS를 new_alias로 대체한다.
26 : Lnode.setM_TableList ( M_TableList ) ;
27 : Lnode.setM_Query ( M_Query ) ;
28 : }
29 : Lnode.setKind ( E_TABLE ) ;
30 : stack.push ( Lnode ) ;
```

(그림 9) ASTSubscript(Filter)

(그림 9)의 줄 1은 스택의 pop연산을 수행한다. 그결과 "hp and office"의 처리결과 정보가 Rnode에 할당된다. 줄 2는 pop연산을 수행한 후 phone정보를 Lnode에 할당한다. 줄 3은 Rnode의 타입을 평가한 후 Switch구문을 수행한다. Rnode의 타입은 (그림 8)의 줄 23에 의해서 타입이 E_TABLE임을 알 수 있다. 줄 23의 결과 줄 8에서 줄 26까지 수행한다.

줄 8에서 getM_Lalias에 의해서 (그림 8)의 줄 10에 저장된 alias를 추출한 다음 tmp_alias에 할당한다. 줄 9에서는 새로운 테이블 alias를 생성한 후 new_alias에 할당한다. new_alias에는 *Alias_6*이 할당된다. 줄 10은 setM_Falias에 의해서 new_alias를 Lnode에 할당한다. 줄 11은 setM_Lalias에 의해서 new_alias를 Lnode에 할당한다. 줄 12는 M_TableList에는 SQL의 FROM절에 해당하는 문자열을 구성하게 된다. 그결과 M_TableList에는 "t_element Alias_6"이 할당된다. 줄 13의 M_Query는 SQL의 WHERE절에 해당하는 문자열을 구성한다. 그 결과 M_Query에는 "Alias_6.name = 'phone'" 가 할당된다.

줄 14는 NOT 연산자가 사용되었는지를 판단하는 구문이다. 줄 15는 NOT 연산자가 하나 이상인지를 판단하는 구문이다. 이 구문이 참이면 줄 16을 수행하고 그렇지 않으면 줄 18을 수행한다. 줄 19는 and나 or 연산자가 사용되었는지를 판단하는 구문이다. (질의 1)에서 and 연산자를 사용했으므로 줄 19가 참이 되어서 줄 20을 수행한다. 줄 20에서 getM_TableList는 (그림 8)의 줄 14에 의해서 저장된 정보를 추출한다. 그 결과 M_Query에는 "Alias_6.name = 'phone' and exists (select * from t_element Alias_4, t_element Alias_5 where (Alias_4.name = 'hp' and Alias_4.parent = PATH_ALIAS.id) AND (Alias_5.name = 'office' and Alias_5.parent = PATH_ALIAS.id))"가 할당된다.

줄 21이하는 줄 14와 줄 19에 해당하지 않는 경우에 수행된다.

줄 25에서는 줄 20에서 생성된 M_Query에 존재하는 PATH_ALIAS를 new_alias로 대체한다.

줄 26는 setM_TableList에 의해서 M_TableList를 Lnode에 할당한다. 줄 27은 setM_Query에 의해서 M_Query를 Lnode에 할당한다. 줄 29은 setKind에 의해서 Lnode의 타입을 E_TABLE로 만든다. 줄 30는 스택에 Lnode를 push한다. 지금까지 변환된 모습을 (그림 7)의 (2)에서 확인할 수 있다.

알고리즘 3 : ASTRelativePath

```
01 : Rnode = stack.pop ( ) ;
02 : Lnode = stack.pop ( ) ;
03 : If ( Token.image == "/" )
04 :  Switch ( Rnode.kind ( )) {
05 :    Case : E_ELEMENT : …
06 :    Case : E_COMPARISON : …
07 :    Case : E_ATTRIBUTE : …
08 :    Case : E_TABLE : …
09 :     tmp_alias = Rnode.getM_Lalias ( ) ;
10 :     new_alias = 새로운 Table_Alias 생성 ;
11 :     Lnode.setM_Falias ( Rnode.getM_Falias ( )) ;
12 :     Lnode.setM_Lalias ( new_alias ) ;
13 :     level = Rnode.NumWildCard ( ) + 1 ;
14 :     M_TableList = Rnode.getM_TableList ( ) + ", t_element "
                       + new_alias ;
```

```
15 :    if ( Rnode.NumWildCard ( ) == 0 ){
16 :    M_Query = Rnode.getM_Query ( ) + " and " + new_alias
                  + ".doc_id = " + tmp_alias + ".doc_id and "
                  + new_alias + ".name = " + Lnode.Content ( )
                  + " and " + tmp_alias + ".parent = "
                  + new_alias + ".id " ;
17 :  } else {
18 :    M_Query = Rnode.getM_Query ( ) + " and " + new_alias
                  + ".doc_id = " + tmp_alias + ".doc_id and "
                  + new_alias + ".name = " + Lnode.Content()
                  + " and " + new_alias + ".tstart < "
                  + tmp_alias + ".tstart and " + new_alias
                  + ".tend > " + tmp_alias + ".tend and "
                  + new_alias + ".depth = " + tmp_alias
                  + ".depth - " + Integer.toString ( level ) ;
19 :  }
20 : Lnode.setM_TableList ( M_TableList ) ;
21 : Lnode.setM_Query ( M_Query ) ;
22 : }else if ( Token.image == // ) {
23 : …
24 : }
25 : Lnode.setKind ( E_TABLE ) ;
26 : stack.push ( Lnode ) ;
```

(그림 10) ASTRelativePath

(그림 10)의 줄1는 pop연산에 의해서 Rnode에 "phone [hp and office]" 연산의 결과를 추출하여 Rnode에 할당한다. 줄 2는 pop연산에 의해서 Lnode에 "namecard" 정보를 할당한다. 줄 3의 Token이 "/" 이면 줄 4에서 줄 21까지를 수행하고 줄 22의 Token이 "//"이면 줄 23을 수행한다.

줄 4는 Rnode의 타입을 평가한다. 평가 결과 Rnode의 타입이 E_TABLE이므로 줄 9에서 줄 21까지 수행한다. 줄 9의 tmp_alias는 (그림 9)의 줄 11에 의해서 저장된 정보가 할당된다. 그 결과 tmp_alias에는 *Alias_6*이 할당된다. 줄 10의 new_alias에 새로운 테이블 alias가 생성할당 된다. 그 결과 new_alias에는 *Alias_7*이 할당된다.

줄 11은 Rnode.getM_Falias()의 수행결과 *Alias_6*을 가져와서 Lnode에 할당한다. getM_Falias는 선행참조를 위한 오퍼레이션이다. 줄 12는 줄 10에서 생성된 테이블 alias를 후행참조를 위해서 Lnode에 new_alias를 할당한다.

줄 13의 level은 와일드카드의 개수에 1을 더한 숫자를 저장한다. 와일드카드만큼 탐색의 깊이를 만드는 연산이다. 줄 14의 M_TableList는 "t_element Alias_6, t_element Alias_7"이 할당된다. 줄 15는 와일드카드의 개수가 0과 같은지 판단하고 참이면 줄 16을 수행하고 그렇지 않으면 줄 18을 수행한다. 여기에서는 와일드카드가 없으므로 줄 16을 수행한다.

줄 16의 M_Query에는 "Alias_6.name = 'phone' and exists (select * from t_element Alias_4, t_element Alias_5 where (Alias_4.name = 'hp' and Alias_4.parent = Alias_6.id) AND (Alias_5.name = 'office' and Alias_5.parent = Alias_6.id)) and Alias_7.doc_id = Alias_6.doc_id and Alias_7.name = 'name-card' and Alias_6.parent = Alias_7.id"가 할당된다.

줄 20은 줄 14에서 생성된 정보가 Lnode에 할당된다. 줄 21은 줄 16에서 생성된 정보가 할당된다. 줄 25에서 Lnode의 타입을 E_TABLE로 설정하고 줄 26에서 스택에 Lnode를 push한다.

알고리즘 4 : ASTQuery

```
01 : node = Stack.pop ( ) ;
02 : M_Query = node.getM_Query ( ) ;
03 : M_TableList = node.getM_TableList ( ) ;
04 : if ( node.beUnion ( ))
05 :    s = " SELECT a.doc_id, a.tstart, a.tend, a.content, a.name, 's'
          AS tag, att.attr_name, " + " attl.attr_value, e.prefix_name,
          at. prefix_ name " + " FROM t_element a, " + " t_attribute
          att, attribute_list attl, t_namespace e, t_namespace at "
          + " where a.id = attl. element_id (+) and attl.attr_id = att.
          attr_id (+) " + " and a.prefix_id = e.id (+) and attl.prefix_id
          = at.id (+) and " + " exists (select " + node.getM_Falias( )
          + " .* from t_element " + node.getM_Falias ( ) + " where
          " + node.getM_Falias ( ) + " .id = a.id and exists ( " +
          M_Query + " ) ) " ;
06 : else
07 :    s = " SELECT a.doc_id, a.tstart, a.tend, a.content, a.name, 's'
          AS tag, att.attr_name, " + " attl.attr_value, e.prefix_name,
          at.prefix_ name " + " FROM t_element a, " + " t_attribute
          att, attribute_ list attl, t_namespace e, t_namespace at
          " + " where a.id = attl. element_id (+) and attl.attr_id =
          att.attr_id (+) " + " and a.prefix_ id = e.id (+) and attl.
          prefix_id = at.id (+) and " + " exists (select " + " MAIN_
          ALIAS. * from t_element MAIN_ALIAS, " + M_TableList
          + " where MAIN_ALIAS.id " + " = a.id and MAIN_ALIAS.
          doc_id = " + node.getM_Falias ( ) + ".doc_id and " + node.
          getM_Falias ( ) + " .tstart <= MAIN_ALIAS.tstart and
          " + node.getM_Falias ( ) + " .tend >= MAIN_ALIAS.tend
          and " + M_Query + " ) " ;
08 : ReconstructXML (s) ;
```

(그림 11) ASTQuery

(그림 11)의 줄 1에서 pop 연산을 수행 후 스택에 저장된 연산결과 정보를 node에 할당한다. 줄 2에서 node에 저장된 M_Query정보를 가져와서 M_Query에 할당한다.

줄 3에서 M_TableList에 node에 보관된 테이블 리스트를 가져와서 M_TableLIst에 할당한다.

줄 4에서 node.beUnion()은 node에 Union이 사용되었는지를 판단하는 구문이다. 여기에서는 사용되지 않았으므로 else구문을 수행한다. M_Query가 Union 형태의 질의를 포함하고 있는 경우 s에 할당될 질의에 이미 MAIN_ALIAS를 포함할 것이기 때문이다. else 문장에서의 s에 할당될 문자열에서는 MAIN_ALIAS를 포함한다.

(그림 11)에서는 최종적인 SQL문자열을 만들기 위해 이전에 조합했던 부분문자열을 조합하고, XML 문서형태로 출력하기 위한 문자열을 추가로 삽입하여 완전한 SQL문자열을 만든다.

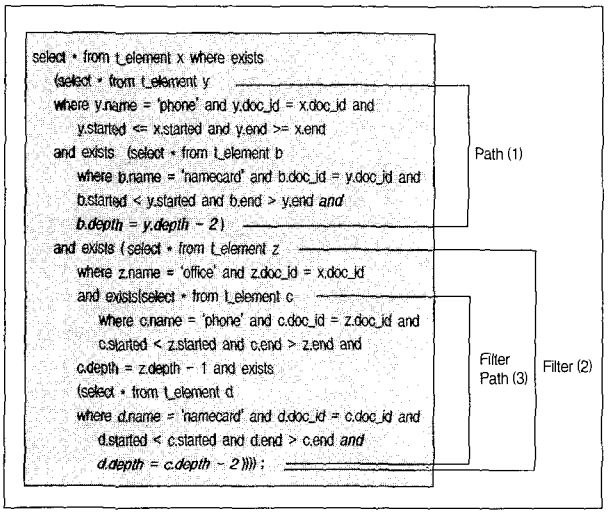
줄 7의 구문에서 s에 포함될 문자열은 엘리먼트, 애트리뷰트, 애트리뷰트 리스트, namespaces 간의 테이블을 조인하며, WHERE절에서 엘리먼트의 ID와 애트리뷰트 리스트의 ID를 OUTER 조인함으로서 엘리먼트에 존재하는 애트

리뷰트를 결합하고 애트리뷰트를 포함하지 않는 엘리먼트 정보를 획득할 수 있다. namespaces를 위한 prefix의 경우에도 이와같이 outer 조인을 한다.

질의 2 : *namecard/*/phone[office]*

위의 XQL은 *namecard* 요소로부터 손자에 해당하는 경로에 존재하는 *phone*요소를 추출하라는 것으로, 특히 *phone* 요소가 *office*요소를 포함하는 *phone*요소를 추출하라는 의미이다.

(질의 2)는 XQL을 SQL로 변환에서 XQL경로의 표현이 SQL로의 경로표현에 어떻게 표현되는지 의미는 어떠한지를 (그림 12)의 SQL로의 변환된 구문을 통해서 설명하고자 한다.



(그림 12) XQL에 해당하는 생성된 SQL

(그림 12)의 Path(1)가 설명하는 것은 "*namecard/*/phone*"에 해당하는 부분의 SQL 질의문자열 이다. 이 부분은 질의가 "*namecard/phone*"이나 "*namecard//phone*"일 경우에도 같다. 다른 부분은 이탤릭글꼴로 되어 있는 부분으로, 이 부분은 두 요소들의 연결관계(*Level* 차이)를 표현하는 부분으로 위의 예제에서는 "/*/"로 연결되어 있으므로 "*b.depth = y.depth - 2*"로 표현되어 있다.

(그림 12)의 Filter(2)가 설명하는 것은 "*[office]*"에 해당하는 부분의 SQL 질의문자열이다. Filter부에 해당하는 요소는 필터([ ])표현 앞부분에 나와 있는 경로 정보를 모두 포함하고 있어야만 한다. 따라서 필터부는 앞부분에서 표현된 경로정보를 모두 가지고 있어야만 한다.

(그림 12)의 Filter Path(3)가 설명하는 것은 Path(1)을 기반으로 생성되기 때문에 Path(1)이 바뀌면 이 부분도 바뀌어야 한다. 따라서, 이탤릭체로 되어 있는 부분이 위와 같이 변경되어야 한다.

4.3 SQL을 이용한 결과 추출과 XML문서 재조합 알고리즘

SQL의 부분 문자열은 결과를 doc_id, start로 정렬시킨 형태이다. 정렬시키는 이유는 추출된 doc_id별로 XML문서 형태로 재 조합하기 위해서이다. 위의 SQL질의 결과를 한 레코드씩 가져와서 태그를 붙여주면 추출된 결과를 XML문서 형태로 출력할 수 있게된다.

태킹(Tagging) 알고리즘은 가져온 레코드의 t_start와 t_end 값을 비교하면서 스택을 이용하여 순서에 맞게 태그를 붙여주게 된다. 알고리즘은 (그림 13)과 같다.

```
01 : while ( 질의결과 데이터가 존재하면 ) {
02 :     rec = 결과셋에서 한 엘리먼트를 가져온다.
03 :     if ( rec.문서_id == 이전문서_id ) {
04 :         if ( 스택이 널이 아니면 ) {
05 :             tmp = m_stack.pop ( ) ;
06 :             if ( tmp.getEnd ( ) > rec.getStart ( ))
07 :                 m_stack.push ( tmp ) ;
08 :             else end_tagging ( tmp.getEnd ( ))
09 :         }
10 :         start_tagging ( rec.getStart ( )) ;
11 :         content 및 attribute출력
12 :         if ( rec.getEnd ( ) - rec.getStart ( ) == 1)
13 :             end_tagging ( rec.getEnd ( )) ;
14 :         else m_stack_push ( rec) ;
15 :     }else {
16 :         while ( 스택이 널이 아니면 )
17 :         end_tagging ( m_stack.pop ( ).getEnd ( )) ;
18 :         이전문서_id=rec.getDoc_id ( ) ;
19 :     }
20 : }
```

(그림 13) 태깅(Tagging) 알고리즘

줄 01에서 SQL질의 결과 데이터가 존재하면 while루프를 수행한다. 줄 02에서 변수 rec에 결과셋에서 한 요소(Element)를 가져온 다음 rec에 할당한다. 줄 03에서 결과셋에서 가져온 문서_id와 이전문서_id가 같으면(문서 id가 바뀐 경우를 체크) 줄 04~줄 14를 실행한다. 줄 04에서 스택이 널이 아니면, 줄 05~줄 08을 수행한다. 줄 05에서 tmp에 m_stack에서 요소하나를 pop 값을 할당한다. 줄 06에서 변수 tmp의 끝 요소번호가 변수 tmp의 시작 요소번호보다 크면 줄 07을 실행한다(줄 6이 거짓이면 끝 태그를 출력한다.) 줄 10에서 rec의 시작태그를 출력한다. 줄 11에서 요소의 내용과 속성을 출력한다.

줄 12에서 rec의 끝 요소번호에서 rec의 시작 요소번호의 차이가 1과 같으면 하위 요소(Element)를 갖고 있지 않은 경우이다. 줄 *12*가 참이면 줄 *13*을 실행한다. 줄 *13*에서는 *rec*의 끝 태그를 출력한다. 줄 12가 참이 아니면 줄 14에서 m_stack에 rec을 push한다.

줄 07이 참이 아니면 줄 16~줄 19(현재 문서_id와 이전문서_id가 다른 경우)를 실행한다. 줄 20~줄 22에서는 스택에 보관된 모든 요소를 pop하고 끝 태그를 출력한다.

4.4 문서의 트랜잭션

본 시스템은 XML문서의 Commit여부를 나타내기 위하여

T_DOCUMENT 테이블에 COMMITTED 필드를 가지고 있다. 문서를 삽입하기 시작할 때 이 필드를 "N"으로 지정한 후, 문서의 모든 요소(Element)들이 무사히 삽입을 마치게 되면 이 Field를 "Y"로 변환해준다.

COMMITTED 필드가 "N"으로 지정되어 있는 문서는 XQL 문자열 처리작업 시에 처리 데이터에서 제외된다. 또, "Power Fault"등의 갑작스런 문제발생으로 인하여 문서가 완전 하게 삽입되지 못한 경우에는 별도의 방법을 이용하여 COMMITTED 필드가 "N"으로 지정되어 있는 레코드를 제거시킬 수 있다.

대용량의 XML문서 처리를 위하여 한 문서의 요소(Element) 삽입 시 요소 100000개 당 한번의 Commit을 수행한다. 모든 요소가 삽입 작업을 무사히 마치게 되면, COMMITTED 필드를 "Y" 변환해준다.

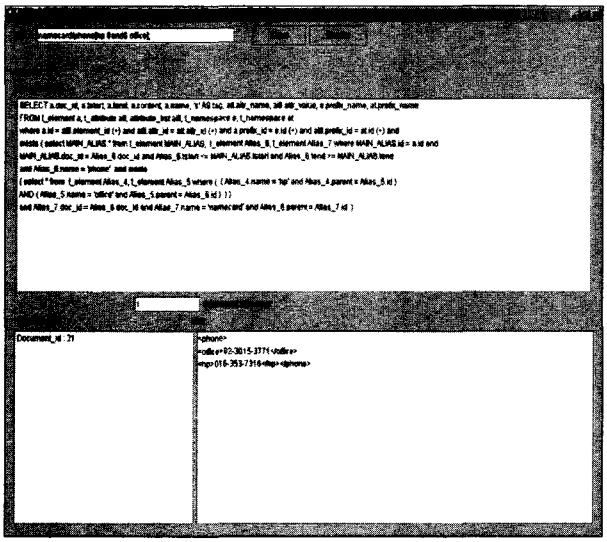
4.5 시스템 구현환경 및 실행화면

시스템을 위한 구현환경은 다음과 같다.

- 데이터베이스 : Oracle 8.16
- 운영체제 : Windows 2000
- 프로그래밍 언어 : JDK 1.2.2
- XML파서 : Xerces Java Parser 1.4.4
- XQL BNF 작성 : JavaCC

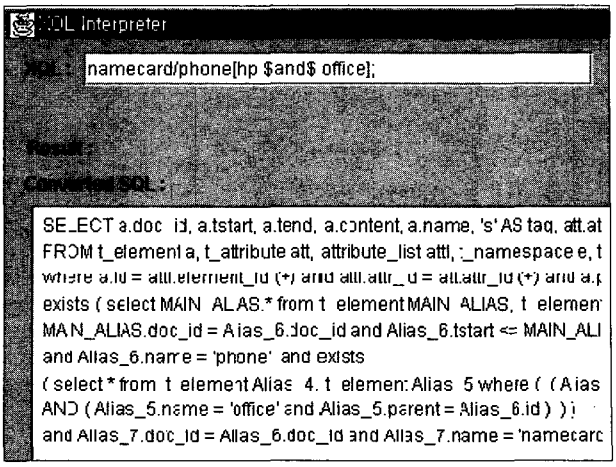
다음으로 구현결과에 대한 실행화면은 (그림 14)과 같다. XQL질의가 다음과 같을 때 (그림 14)는 실행결과에 대한 전체화면이다.

namecard/phone[hp and office]



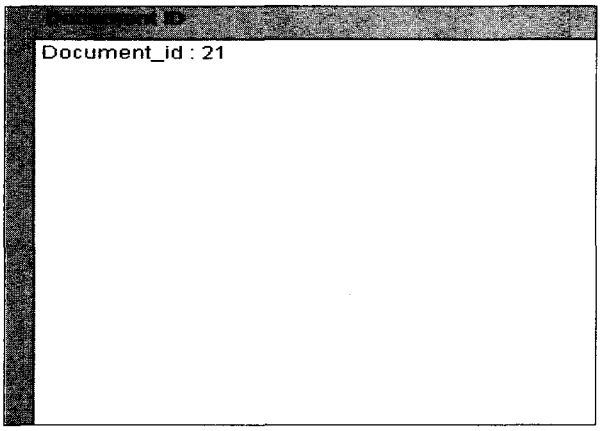
(그림 14) 자바 어플리케이션 실행화면

(그림 15)은 (그림 14)에서 XQL입력 창에 XQL을 SQL로 변환 한 결과 화면을 보인 것이다.



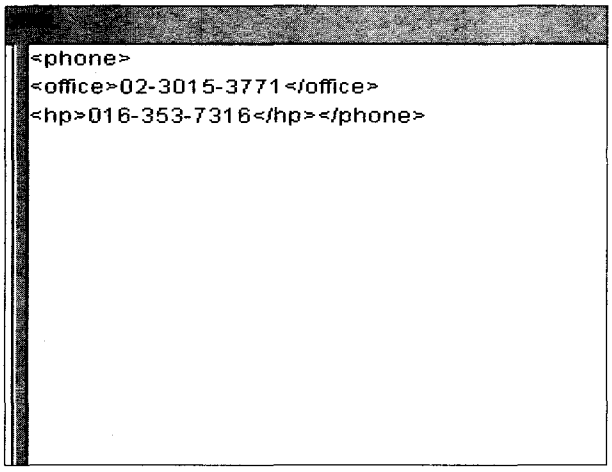
(그림 15) (그림 14)에서 XQL 입력 창

(그림 16)는 (그림 14)에서 문서의 검색결과 Document ID만을 확대한 화면이다. Document ID는 질의에 검색대상으로 결정된 Document ID만을 보인 것이다.



(그림 16) 문서 id

(그림 17)은 XQL질의 결과화면이다.



(그림 17) XQL질의 결과화면

4.6 성능분석

논문에서 제안한 시스템의 성능분석은 XQL Engine API를 이용하여 XQL Query에 대하여 첫째, Doc_id를 제공하는 메소드 둘째, 결과를 XML문서 형태로 제공하는 경우로 나누어 성능을 분석하였다.

실험을 위해 사용된 데이터는 프로그램을 이용하여 생성된 데이터를 각각 10000개 요소(Element), 50000개 요소, 100000개 요소를 XML저장소에 삽입한 후에 질의 결과를 출력하는 시간을 체크하였다.

실험을 위한 XQL Query는 6가지 질의를 사용한다. 또한 결과를 화면에 출력하는 시간은 고려하지 않는다. 즉, 순수하게 결과를 받아오는데 걸리는 시간만을 고려한다. 다음은 실험을 위한 질의를 보인 것이다.

- **XQL Query**

1. namecard ;
2. namecard/phone/hp ;
3. //office ;
4. namecard/name [@eng] ;
5. namecard/name [@eng eq 'tpfoj'] ;
6. namecard [address eq 'eindb'] ;

실험을 위한 시간단위는 SEC를 사용했고, 실험 장비의 다른 요인은 고려하지 않는다. <표 9>는 요소의 개수가 10000, 50000, 100000 개로 될 때, 위의 XQL(1~6)질의에 대해 해당하는 Doc_id(문서ID)를 추출하는데 걸리는 시간을 측정한 것이다.

〈표 9〉 Doc_id만을 추출하는 시간

XQL query	10000	50000	100000
1	0.5	3.37	6.91
2	0.54	3.31	6.42
3	0.7	3.14	6.24
4	0.82	3.49	6.71
5	0.86	3.35	6.64
6	1.04	4.07	8.41

<표 10>은 요소의 개수가 10000, 50000, 100000개 일 경우에, 위의 XQL(1~6)질의에 대해 XML문서를 생성하는데 요소 된 시간을 측정한 것이다.

〈표 10〉 XML형태로 질의결과를 생성하는 시간

XQL query	10000	50000	100000
1	1.04	8.89	18.84
2	1.20	5.08	9.71
3	1.14	4.89	9.50
4	1.16	5.27	9.84
5	1.28	5.27	9.87
6	1.62	6.59	13.09

[9]에서는 실험용 데이터를 *SIGMOD* 데이터와 *NITF100 (News Industry Text Format)* 데이터를 사용하여 성능분석을 하였고 *SIGMOD* 데이터의 요소를 100에서 1000개까지를 증가시키면서 시간을 측정한 결과는 요소의 크기에 좌우되지 않고 5초로 균일하였는데, 이는 실험에 사용한 요소의 개수가 작아서 실험결과를 잘 반영할 수 없었던 것으로 보인다.

NITF100을 이용한 실험에서는 1개의 요소에서 1000개이 요소사이에서 시간은 대략 3초로 균일하였다. NITF100의 경우는 프로그램으로 생성된 데이터라는 점에서 실제데이터와 약 2초 정도의 차이를 보이고 있다.

본 연구에서의 실험 결과는 요소의 개수를 10000개 50000개 10000개로 데이터의 크기가 [9]의 실험과는 차이가 있으나 요소의 개수가 10000개인 실험데이터의 결과가 대략 1.5초 정도로서 NITF100의 1000개 요소를 사용한 경우보다 3배가 빠름을 알 수 있다.

5. 결론 및 향후 연구

본 논문에서는 XML문서의 요소(Element), 속성(Attribute)의 특성 및 부모와 자식 요소사이의 관계를 유지하고, 관계형 데이터베이스에 XML문서를 저장하기 위한 트리 형태의 저장 방식인 "*DFS-Numbering*"방식을 사용하여 XML문서를 저장하였다. 또한, 관계형 데이터베이스에 저장된 XML문서에 XQL질의어로 질의하여 질의 결과로 XML문서를 획득하기 위한 시스템을 설계 및 구현하였다.

XML문서에서의 트리 구조상 요소(Element)의 레벨 값을 저장함으로써 효과적인 문서의 재구성과 구조적인 검색이 가능하도록 하였다. 그리고 데이터베이스 스키마를 이용하여 XQL를 최적의 SQL로 변환하여 효율적인 질의를 수행하기 위한 알고리즘을 제안하였다.

본 논문의 개선사항으로는 XQL질의어가 입력으로 들어왔을 때 SQL로 변환하는 시간과 변환된 SQL질의어가 데이터를 가져올 때 데이터베이스에 저장된 데이터가 트리 형태로 저장되어 있으므로 경로의 길이가 길어지면 수행시간이 떨어지는 단점을 향후 보완할 필요가 있다.

향후 계획으로는 본 논문에서 XQL[6]에서 제안한 질의 연산의 완벽한 지원과, 질의 처리성능을 개선하여 상업적으로 이용 가능한 시스템을 구현하는 것이다. 또한 XQuery가 XML표준언어로 확실시되므로 이를 지원하는 시스템도 개발할 것이다.

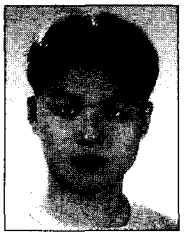
참 고 문 헌

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and EveMaler. Extensible Markup Language (XML) 1.0 second edition W3C recommendation. Technical Report REC-

xml-20001006, World Wide Web Consortium, October, 2000.
[2] M. Carey, et. al., "XPERANTO : Middleware for Publishing Object-Relational Data as XML Documents," WebDB Workshop, Dallas, May, 2000.
[3] M. Fernandez. W. Tan, D. Suciu, "SilkRoute : Trading Between Relations and XML," World Wide Web Conf., Toronto, Canada, May, 1999.
[4] 이용석, 손기락 "XML문서 저장을 시스템의 설계 및 구현", 정보과학회 학술발표논문집(I). 1998.
[5] Jonathan Robie, Joe Lapp, David Schach, XML Query Language (XQL), http://www.w3.org/TandS/QL/QL98/pp/xql.html #.
[6] James Clark, XPATH, http://www.w3.org/TR/1999/REC-xpath-19991116.
[7] Don Chamberlin, Daniela Florescu, Jonathan Robie, Jrme Simon, and Mugur Stefanescu. XQuery : A Query Language for XML W3C working draft. Technical Report WD-xquery-20010215, World Wide Web Consortium, February, 2001.
[8] Jason McHugh and Jennifer Widom. Query optimization for XML. In Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, pp.315-326, September, 1999.
[9] Quanzhong Li, Bongki Mon : Indexing and querying XML data for requlr path expressions, VLDB, 2001.
[10] David Megginson, http://www.megginson.com/SAX/.

김 천 식

e-mail : mipsan@kl.ac.kr
1995년 안양대학교 전자계산학과(공학사)
1997년 한국외국어대학교 컴퓨터 및 정보통신공학부(공학석사)
2001년 한국외국어대학교 컴퓨터 및 정보통신공학부(박사수료)
2000년~현재 경동대학교 정보통신공학부 교수
관심분야 : 데이터베이스, 멀티미디어 데이터베이스, XML



김 경 원

e-mail : kwstone@daps.hufs.ac.kr
2001년 한국외국어대학교 컴퓨터 및 정보통신 공학부(학사)
2001년~현재 한국외국어대학교 컴퓨터 및 정보통신공학과(석사과정)
관심분야 : 데이터베이스, 멀티미디어 데이터베이스, XML

이 지 훈

e-mail : airer@lycos.co.kr
2001년 한국외국어대학교 컴퓨터 및 정보통신 공학부(학사)
2001년~현재 한국외국어대학교 컴퓨터 및 정보통신공학과(석사과정)
관심분야 : 데이터베이스, 멀티미디어 데이터베이스, XML

장 복 선

e-mail : boksun77@argio.net
2001년 한국외국어대학교 컴퓨터 및 정보통신 공학부(학사)
2001년~현재 한국외국어대학교 컴퓨터 및 정보통신공학과(석사과정)
관심분야 : 데이터베이스, 멀티미디어 데이터베이스, XML

손 기 락

e-mail : ksohn@hufs.ac.kr
1984년 서울대학교 계산통계학과(이학사)
1986년 서울대학교 계산통계학과 전산학(이학석사)
1993년 미국 University of California, Santa Cruz,(전산학박사)
1996년~현재 한국외국어대학교 컴퓨터 및 정보통신공학부 교수
관심분야 : 데이터베이스, 멀티미디어, XML