XML뷰를 이용한 XQL질의처리 시스템

김천식[†] · 손기락[††]

요 약

XML은 웹 상에서 문서 데이터 교환을 위한 표준이 되었다. 현재 대부분의 상업적인 데이터가 관계형 데이터베이스 시스템에 저장되어있다. 관계형 데이터베이스에 저장되어 있는 문서를 B2B간에 교환하기 위해서는 XML형태의 문서로 변환하는 작업이 필요하다. 본 논문에서는 관계형 데이터베이스에 저장되어 있는 문서데이터로부터 필요한 정보를 쉽고 편리하게 질의하고 결과를 얻을 수 있도록 XML질의처리시스템에 대한 연구를 하였다. 그리고 XML 문서를 사용하는 사용자들이 관계형 데이터베이스에 저장되어 있는 문서를 XML문서로 간주하여 질의할 수 있도록 하기 위해서 R2X라 부르는 XML뷰를 설계하였다. 본 논문에서는 이 R2X뷰를 이용하여 XQL질의언어로 편리하게 질의할 수 있는 질의처리 시스템을 설계 및 구현하였다.

XQL Query Processing System using XML Views

Chun-sik Kim[†] and Kirack Sohn[††]

ABSTRACT

XML has become a standard for exchanging document data on the web. Currently, most of the commercial data are stored in the relational database system. It requires to converse the document into XML form to transfer the data to a relational database system. The purpose of this paper is to research on a query processing system which will make it easy and convenient to raise a query and derive results from the document data stored in a relational database.

We have designed a XML view called R2X(Relational To XML). With this R2X view, users can view a relational database as XML documents. Using R2X views, we design and implement a query processing system which will make it expedient to form a query with a XML query language called XQL.

Key words: XML, XML-QL, QUERY,

1. 서 론

XML[1]문서는 WWW(World Wide Web)상에서 XML문서 데이터 교환을 위한 표준이 되었다. XML의 포함과 자기서술구조라는 특징이 어플리케이션을 위한 어플리케이션간의 데이터교환을 용이하게 한다. XML은 XML문서 자신의 구조를 기술하기 위해서 DTD(Document Type Descrip- tion)를[1] 제공하고 있다.

DTD는 각종 출판물, 카탈로그 및 전자상거래를 위한 다양한 분야를 위해서 개발되었다. XML은 분명 편리한 구조를 갖고 있지만, 기업간에 문서 교환을 위해서 사용되는 XML문서의 수가 매우 많으므로 문서의 크기가 클 것이다. 이런 이유로, XML데이터는 파일 형태로 저장하거나 혹은 데이터베이스에 요소별로 저장시키거나 혹은 XML문서 전체를 저장시켜야 할 것이다. 파일 형태로 XML문서를 보관한다면, 여러 사람이 이 문서에 대한 공유문제는 풀기 어려운 문제가 될 것이다. 또한 이 문서에 대한 갱신문제, 정보검색 문제 등은 쉽지 않다.

따라서, XML문서를 데이터베이스에 저장하는 것은 당연하다. 만일 XML문서가 관계형 데이터베이스에 저장되어 있는 경우에는 당연히 SQL로 질의하면 되겠지만, 다음과 같은 질의를 SQL로 어떻게 표현할

[†] 경동대학교 정보통신공학부 교수
[††] 한국외국어대학교 컴퓨터 및 정보통신공학부 교수

수 있겠는가?

/A/B/C[.='XXX']

위의 질의는 A엘리먼트 밑에 B엘리먼트가 있고 다음으로 C엘리먼트가 나오며, C엘리먼트의 내용이 'XXX'인 C엘리먼트의 밑에존재하는 모든 엘리먼트를 보이라는 XQL질의이다. SQL은 분명 경로에 대한 개념이 없다. 이와 같은 경로를 표현하기 위해서 많은 SQL을 조합하거나 STORED PROCEDURE등을 이용해야 할 것이다. 이와 같은 형태의 질의 표현은 매우 복잡하고 어렵고 구현하기도 쉽지 않다. 이를 해결하는 방안으로 XML 뷰를 이용하는 방안을 제안한다.

XML데이터의 경우에는 질의가 복잡하다. 왜냐하면 질의가 정규 경로표현[2-5]를 포함하기 때문일 것이다. 게다가 구조가 비정규이므로 데이터를 순행하는데 유연성이 필요하다. 관계형 데이터베이스와 객체 지향적인 데이터베이스의 경우에 평가의 속도를 향상시키기 위해서 몇 개의 인덱스를 사용한다. 전통적인 관계형 데이터베이스의 경우에 특정 관계의 특정 속성에 인덱스를 하고, 반면에 객체지향 데이터베이스의 경우 객체지향 스키마의 특정 경로에 인덱스를 한다.

최근 작업은 반 구조적인 데이터베이스[6-9]에서 경로의 표현과 평가에 대한 문제를 제시하였다. 그러나 이들은 스키마 정보를 이용한 재 진술 질의(re-write query)를 사용하였다.

본 논문에서는 XML뷰(R2X)질의어를 정의하고, 사용자는 XQL로 질의한다. R2X와 XQL을 결합하여 새로운 R2X 질의를 생성한 하고, 새로 생성된 R2X를 SQL로 변환시킨 다음 관계형 데이터베이스에 질의하여 결과로 튜플을 얻어 사용자에게 제공하는 시스템의 설계 및 구현 알고리즘을 보이고자 한다.

2. 관련연구

대부분의 상업적인 데이터베이스 시스템은 관계형 데이터에 대한 형상화된 XML뷰를 만드는 방법을 제공한다[10-12].

하지만, 대부분의 시스템이 XML뷰를 통해서 질의하도록 지원하는 시스템은 없다[10,11]. ORACLE사의 XSQL툴은 각각의 릴레이션과 속성 이름을 XML태그와 튜플에 맵핑함으로써 관계형 데이터를 XML 문서로 규범적인 맵핑을 정의한다.

이와 같은 뷰는 가상적이지만 접근방법은 임의의 XML포맷으로의 맵핑을 지원하는데 충분히 일반적이지 않다.

하지만 최근 연구 중 하나로, SilkRoute[14,15]는 관계형 데이터에 대해 XML뷰를 이용한 질의를 지원하는 시스템으로 질의 언어로 XML-QL를 사용하였다. 본 논문은 XQL[16]을 질의어로 하는 XML 질의처리 시스템을 설계 및 개발하였다. XQL과 XML-QL[17]는 다음과 같은 언어적인 차이가 있다.

- XQL은 쉽고 효율적이다.
- XQL은 필터로 엘리먼트, 애트리뷰트, 처리명령(PI), 주석(Comment), 개체참조(Entity Reference s)를 적용할 수 있다.
- XQL은 constructor 구문을 갖고 있지 않다. 대신에 Pattern표현은 질의에 대한 결과를 결정한다.
- XQL은 그룹 오퍼레이터를 지원하지 않는다.
- XQL은 인덱스에 의한 첨자를 사용한다. 첨자는 단일 숫자, 범위 혹은 숫자와 범위의 조합을 포함할 수 있다.
- XQL은 정규경로 표현을 지원하지 않는다. 하지만, 오퍼레이터 "/", "*" 와 "//"를 사용하여 자식과 손자 엘리먼트에 대한 접근을 지원한다. 따라서, 임의의 깊이를 갖는 엘리먼트에 대한 질의가 가능하다.

본 논문에서 제안한 시스템은 위의 특성 중에서 다음과 같은 특성은 제공하지 않는다.

- 애트리뷰트에 대한 질의는 지원하지 않는다.
- 인덱스에 의한 첨자를 사용한 질의표현은 지원하지 않는다.
- XML문서가 Recursion형태인 경우, 질의를 지원하지 않는다.

최근에 XML 질의언어로 표준화가 거의 확실시되는 XQuery를 질의언어로 선택하지 않고 XQL를 질의 언어로 선택하여 시스템을 설계 및 개발한 이유는 XQL로부터 XPATH가 유래하였고, XQuery는 SQL과 같이 섬세한 질의처리를 하도록 새로이 고안된 질의어이다. 따라서 두 질의어가 특성이 다르고 쓰임새도 다르다. XML에서 경로질의는 매우 중요한 질의처리 문제이며, XQuery도 경로질의 처리를 지원

하지만, 구현과 질의 사용에 있어서 XPATH보다는 복잡하다. 또한 XML 질의 처리에 있어서 XPATH를 축으로 하는 질의처리 유형과 XQuery를 축으로 하는 질의처리가 발전할 것으로 전망하고 있다. 예를 들어, 현재 학계의 연구에서 XML 문서 필터링을 위해서 Continuous Query(CQ)를 이용한 시스템에서 XPATH 질의를 이용하여 필터링을 수행하고 있다. 그러므로 본 논문에서는 XQL로 질의처리 시스템을 구현한 것이다.

따라서, 본 연구에서는 뷰(View)를 이용한 XQL 질의 처리시스템을 제안하고, 알고리즘을 통해서 구현방법을 보이고자한다.

3. XML질의를 위한 뷰 생성

XML은 웹 상에서 정보교환을 위한 계층적인 데이터 포맷이다. XML문서는 루트(Root) 엘리먼트로 시작되며 포함(Nested) 엘리먼트 구조로 구성된다. 엘리먼트 데이터는 속성 혹은 하위 엘리먼트의 형태가 될 수 있다.

(그림 1)은 monograph에 대한 정보를 기술하는 DTD이다. 먼저 (그림 1)에서 기술한 XML데이터가 관계형 데이터베이스에 저장된 것으로 가정한다.

본 논문에서는 XML문서를 관계형 데이터베이스에 어떻게 저장하는가에 대한 관심은 없으며 단지 전통적인 관계형 데이터베이스 시스템에 저장되어 있는 데이터에 대해서 SQL로 질의하는 것과 같이 자유롭고 유연한 질의를 하여 질의 결과를 얻도록 지원하고, 사용자는 단지 XML문서에서 원하는 정보를 얻는다는 생각만 갖도록 하는 편리한 질의지원이 목적이다.

```
<!DOCTYPE monograph [
<!ELEMENT monograph(title, author, editor) >
<!ELEMENT editor(monograph*)
<!ATTLIST  editor name CDATA #REQUIRED>
<!ELEMENT title(#PCDATA) >
<!ELEMENT author (name, address) >
<!ELEMENT author id ID #REQUIRED >
<!ELEMENT name (firstName?, lastName) >
<!ELEMENT firstName (#PCDATA) >
<!ELEMENT lastName (#PCDATA) >
<!ELEMENT address ANY >
]>
```

그림 1. monograph에 관한 DTD

(그림 1)의 DTD와 맵핑될 관계형 데이터베이스 스키마는 (그림 2)와 같다.

```
monograph(monographID, parentID, tile,
editorname, authorID)
author(authorID, parentID, firstName, lastName,
address)
```

그림 2. monograph 스키마

(그림 2)와 같은 구조는 데이터베이스 설계자의 관점에 따라서 다를 수 있을 것이다.

문제는 (그림 1)의 정의에 부합하는 뷰(그림 3)를 어떻게 효과적으로 설계하는가가 중요한 문제이다.

```
1: monograph {  |
2:    title ($mono:=MONOGRAPH) {
3:        mono/title
4:    }
5:    author ($author:=AUTHOR) {  |
6:        name {  |
7:            firstName {
8:              author[mono/authorid = author/authorid]/
                  firstName
9:            }
10:           lastName {
11:             author[mono/authorid = author/authorid]/
                    lastName
12:           }
13:       }
14:       address ($author:=AUTHOR){
15:         author[mono/authorid=author/authorid]/address
16:       }
17:   }
18:   editor ($mono:=MONOGRAPH) {
19:       mono/editor
20:   }
21: }
```

그림 3. XML 뷰

뷰의 구조를 설계하기 전에 뷰에 대한 설계 개념을 이해하는 것은 매우 중요할 것이다. 그러므로, 먼저 뷰의 설계 개념을 이해하고 이를 설계하는 방법을 설명하도록 하겠다.

(그림 3)의 뷰에서 줄1은 monogaph엘리먼트에 대한 정보를 설명하고 있다. monograph에 대한 규칙을 표시하는 곳은 "{"와 "}" 사이에 기록한다. mono-

graph은 PCDATA를 갖고 있지 않고 엘리먼트 정보를 갖고 있으므로 자식으로 엘리먼트를 갖는다는 의미에서 계층표현을 위해 "|"를 사용하였다. "("와 ")" 사이에는 변수를 사용한다. 변수는 "$"로 시작하는 알파벳문자이다. 변수는 스키마 이름을 저장하는데 이용한다. 일반적으로 스키마 이름의 문자열이 길 경우 효과적으로 표현하기 위해서 변수표현을 도입하였다. 변수 다음에 ":="의 오른쪽에는 스키마이름을 기술한다. 줄2는 title 엘리먼트에 대한 정보를 설명하는 것이다. "{"와 "}"사이에 mono/title가 있다. mono/title의 의미는 mono스키마 변수에 있는 title 속성 값을 보이라는 것이다. "("와 ")"사이에 있는 $title은 스키마 변수이고 TITLE은 스키마이다.

줄5는 author 엘리먼트에 대한 정보를 보인 것이다. "{"와 "}"사이에는 하위에 존재하는 엘리먼트에 대한 정보가 기술되어 있다. 왜냐하면 DTD상에 author 엘리먼트는 하위 엘리먼트로 name, address가 있기 때문이다. "("와 ")"사이에 $author는 스키마 변수 명이고, AUTHOR는 스키마이름이다.

줄7은 firstName엘리먼트에 대한 정보를 보인 것이다. "{"와 "}"사이에 있는 구문을 SQL로 바꾸면 다음과 같다.

```
SELECT firstName
FROM AUTHOR author, MONOGRAPH mono
WHERE mono.authorid = author.authorid
```

즉, 이 표현은 SQL 구문으로 바꾸어 보았을 때 테이블 조인(Join)에 해당하는 것이다.

줄8에서 author는 스키마 변수이고 "["와 "]" 사이에는 SQL에서 WHERE절에 해당하는 조건 구문이 나온다. 줄8의 의미는 스키마변수 mono에 속하는 authorid와 스키마변수 author에 속하는 authorid가 같은 lastName을 보이라는 질의이다.

줄10은 lastName에 대한 정보를 설명하는 것으로 줄7과 의미가 같다. 줄14는 address엘리먼트에 대한 정보를 보여주는 것으로서 줄7과 유사하다. 줄 18은 editor엘리먼트에 대한 정보를 보여주는 것이다. 줄 19는 mono스키마 변수에 속하는 editor를 보여주라는 것이다.

(그림 4)는 XML뷰를 저장하기 위한 일반적인 구조이다. (그림 4)는 (그림 3)의 XML뷰 중에서 firstName엘리먼트의 저장형태를 보인 것이다.

elementName	firstName
ruleType	r
textContent	""
Table	author
Filter	mono/authorid=author/authorid
Select	firstName
pathInfo	/monograph/author/name/ firstName

그림 4. XML뷰 정보를 저장하기 위한 구조

우선 Element Name은 "firstName"이 된다. 다음으로 Rule Type은 "r/t/x"로 표시할 수 있다. "r"은 해당 엘리먼트 내부에 XQL정보를 갖고 있다는 것이고 "t"는 해당 엘리먼트에 질의가 아닌 텍스트 형태의 값을 갖고 있다는 것이다.

텍스트 형태라면 "Clothing"과 같은 형태의 문자열을 의미한다. "x"를 사용하는 경우는 해당 엘리먼트에 아무런 정보가 없는 상태를 말한다. 지금은 firstName의 내부규칙이 있으므로 ruletype은 "r"이 된다. 다음으로 textContent가 나오는데, 이 필드에 정보가 저장되는 경우는 ruletype에 "t"가 입력된다. 다음으로 Table필드가 있다. 이 필드는 SQL의 FROM 다음에 나오는 정보, 즉 테이블 정보를 의미하는데, 이 곳에는 AUTHOR가 온다. 다음 필드는 Filter이다. 이 곳은 SQL의 WHERE절 다음에 오는 정보로서, 이 곳에는 "mono/authorid=author/authorid"가 온다. 다음으로 Select 필드가 있다. 이 필드는 SQL의 SELECT 다음에 나오는 정보가 이에 해당한다. 이 곳에서는 firstName이 해당된다. 다음 필드는 pathInfo이다. 이 필드는 XQL질의가 경로에 대한 정보를 요구하므로 이 경로에 대한 정보를 기록하는 것이다. 이 곳에서는 firstName엘리먼트가 monograph엘리먼트 밑에 author 엘리먼트 밑에 name 엘리먼트 밑에 존재함을 의미한다.

(그림 6)은 (그림 1)을 기반으로 XML뷰를 자동생성하기 위한 트리구조의 모습이다. (그림 6)의 왼쪽에 선분이 연결되지 않은 상자는 XML뷰의 구조이다. 이 구조를 표현하기 위한 구조는 (그림 5)와 같다.

구조(그림 5) 필드를 설명하면 다음과 같다.

- ID : 뷰의 엘리먼트를 식별하는 식별자로 이용한다.
- name : 뷰의 엘리먼트 이름이다.

```
struct Element {
    int ID;
    string name;
    int level;
    int parentID;
} ElementInfo[MAX];
```

그림 5. 엘리먼트 구조

• level :엘리먼트의 계층 관계를 표현하기 위한 필드이다.

• parentID : 특정 엘리먼트의 부모 엘리먼트가 누구인가를 지시하는 포인터로 사용하는 필드이다.

뷰 생성 알고리즘은 (그림 1)을 기반으로 (그림 6)과 같은 관계를 만들어내는 것이다.

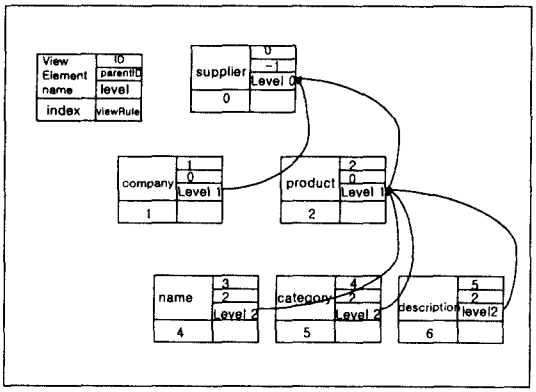

그림 6. DTD를 XML뷰로 변형시키기 위한 구조

(그림 7)은 뷰를 생성하는 알고리즘이다. 하지만 (그림 3)과 같은 뷰가 생성되지 않고 (그림 3)의 뷰(View) 구조만 만들어진다. 왜냐하면 뷰의 내부 구조는 관계형 데이터베이스의 뷰(View)와 같이 자신이 보고싶은 뷰의 모습은 결국 사용자가 정의하도록 하는 것이 당연하기 때문이다.

4. XML질의처리를 위한 구조

관계형 데이터베이스에 뷰 계층을 두고, 뷰를 통해서 관계형 데이터베이스를 본다면, 사용자는 관계형 데이터베이스를 XML 트리 구조로 볼 수 있기 때문에 질의에 도움을 주게된다.

알고리즘 1 : XML DTD를 이용하여 XML뷰 생성

```
뷰_생성()
{
  while(s = 한 줄을 성공적으로 읽는다.) {
    if (파일에서 1번째 줄을 읽을 경우) {
    • s에서 "(" 문자 앞의 element를 분리한다.
    • 분리된 엘리먼트에 ID값으로 유일한 값을 부여한다.
    • "("와 ")"사이의 엘리먼트를 추출한다.
    • 추출된 엘리먼트에 ID를 부여한다.
    • 추출된 엘리먼트에 name(엘리먼트이름)을 부여한다.
    • 추출된 엘리먼트의 parentID에는 이전의 엘리먼트의 id를 이용한다.
     }/* if */
     else {
    • s에서 "(" 문자 앞의 element를 분리한다.
    • 분리된 엘리먼트의 이름과 같은 이름이 이전에 만들어진 tree에 존재하는지 tree traverse 하여, 고유한 id를 알아낸다. 알아낸 id는 "("와 ")"다음의 엘리먼트의 parentID로 설정하는데 이용한다.
    • "("와 ")"사이의 엘리먼트를 추출한다.
    • 추출된 엘리먼트에 ID를 부여한다.
    • 추출된 엘리먼트에 name(엘리먼트이름)을 부여한다.
     }/* end else */
  }
  트리를 Inorder로 탐색하면서 "{"와 "}"를 만들어주면 뷰의 기본 구조를 만든다.
}
```

그림 7. DTD를 이용한 뷰 생성알고리즘

(그림 8)은 (그림 3)을 보고 질의하는 사용자 XQL 질의이다. 질의를 설명하면 다음과 같다. 즉, 루트(/) 패스에서 시작하여, monograph밑에 title엘리먼트의 내용이 "Subclass Cirripedia"인 editor 엘리먼트를 추출하라는 질의이다.

(그림 9)는 (그림 3)의 뷰와 (그림 8)의 질의를 결합한 결과 일치하는 부분만을 추출하여 재구성한 새로운 R2X질의이다.

```
results {  |
   result { |
    editor {
     /monograph/editor[/monograph/title="Subclass
      Cirripedia"]
    }
   }
}
```

그림 8. XQL질의

```
results { |
   result { |
      editor {
       mono[title='Subclass Cirripedia"]/editor
      }
   }
}
```

그림 9. XML뷰와 XQL질의가 결합된 새로운 XML뷰

(그림 10)의 질의 처리구조는 본 논문에서 R2X뷰를 이용한 질의에 대한 전반적인 구성을 나타낸 것이다. 입력으로 DTD가 들어오면, 이 DTD를 이용하여 본 논문에서 제안한 R2X뷰를 생성한다. 다음으로 XQL질의가 들어오면 파싱하고, R2X뷰와 XQL질의를 결합하여, 재 진술한 다음 새로운 R2X 뷰의 부분집합을 만든다. 새로 만들어진 부분집합 R2X뷰를 SQL로 변환한 다음 ODBC를 이용하여 관계형 데이터베이스에 질의하여 결과를 얻어온다. 얻어온 질의 결과물은 사용자에게 제공한다.

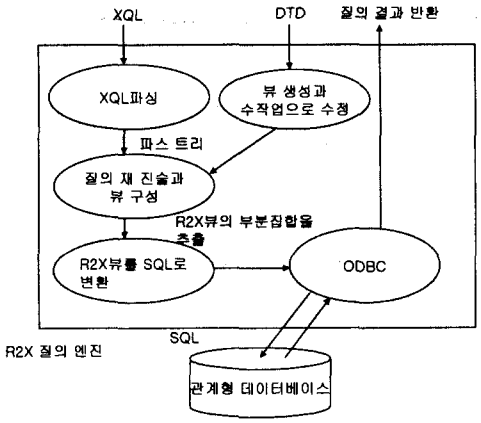

그림 10. Query처리 구조

(그림 9)를 SQL로 형태로 변환시켜보면, 다음과 같다.

```
SELECT  X.editor
FROM monograph X
WHERE X.title="Subclass Cirripedia"
```

위의 질의를 관계형 데이터베이스에 질의하여 결과를 얻어온 다음 사용자에게 결과를 반환한다.

5. 질의처리를 위한 결합 및 변환 알고리즘

본 논문에서 제안한 질의 처리를 하기 위해서는 (그림 7)에서 제안한 방법에 의해서 DTD를 입력인자로 하고, 결과로 XML뷰를 생성해야한다.

(그림 11)은 질의처리를 위한 알고리즘이다. 줄2는 (그림 3)의 뷰를 읽고 XML 저장구조(트리구조)에 저장한다. 줄3은 사용자부터 XQL 질의를 읽는다. 읽은 내용은 xqlClause에 보관한다.

줄5는 xqlClause를 읽고 각각 엘리먼트 이름과 ruleInfo로 분리한다. ruleInfo는 XQL질의어의 특성을 특성별로 분류하여 보관한다. XQL질의 정보에 대한 저장방법은 (그림 4)와 유사하며 저장 구조는 (그림 11)내부에 구조체 형태로 나타내었다.

줄6은 ruleInfo로부터 XQL질의 특성을 추출한 다음 rule에 보관한다. 줄7은 룰이 'select children' 이거나 'select descendants'인 경우에 대한 처리를 위한 것이다. 이 룰은 특정 엘리먼트의 상대 패스나 절대 패스에 존재하는 자식 엘리먼트나 손자 엘리먼트 또는 임의의 깊이를 갖는 엘리먼트를 선택하기 위한 질의로 주로 사용된다. 줄8은 줄7과 같은 조건인 경우, rewrite를 이용하여 (그림18)과 같은 트리를 만들고 트리에 임의의 노드에 필요한 정보를 저장한다. 질의가 (그림 8)과 같은 질의이면, 트리의 노드로 (그림 18)과 같이 results, result, editor가 노드가 되고 editor노드 내부에는 monograph, editor, title엘리먼트에 대한 정보를 R2X뷰에서 참조하여 (그림 18)의 구조를 만들고, 이 구조에 'title="Subclass Cirripedia"의 정보를 포함시킨다.

rewrite의 알고리즘에 대한 자세한 내용은 (그림 11)에서 다시 설명하도록 한다. 줄9는 사용자 XQL질의에 질의 특성이(rule type)이 'select children' or 'select descendants'인 경우 rewrite 모듈을 적용하기 전에 필요한 전처리로서 테이블이 하나이상이고 where절에 필요한 필터가 하나 이상인 경우의 처리이다. 줄9의 ScCompose알고리즘은 나중에 다시 설명하도록 한다.

줄10은 rule이 'filters'인 경우이다. 이 것은 book[excerpt]와 같은 형태의 질의로서 book 엘리먼트 밑에 excerpt 엘리먼트의 인스턴스가 존재하는가를 묻는 것이다. 이 경우의 질의 처리는 줄10의 rewrite알고리즘의 처리로 뷰와 사용자 질의가 결합된다.

알고리즘 2 : 결합(Compose) 알고리즘{R2X 뷰와 XQL 질의 결합}

```
typedef struct {
   string elementName // xql엘리먼트 이름보관
   string ruleType // t<- 텍스트, r<- table,filer,select
   string elementContent // 엘리먼트의 내용이
         텍스트인 경우에 보관
   string tableName[3] // 테이블 이름보관
   string filterInfo[3] // 필터링 정보를 보관
   string selectInfo[3] // select 정보를 보관
   string pathInfo // 패스정보를 보관
   string sqlUserQuery // xql을 sql로 변환한 정보를 보
관
   string ruleTypeName // xql질의 형태정보를 보관
} ruleInfo[MAX];
1: xqlProcess()
2: {   R2X뷰파일(그림 3)을 읽고 의 XML저장구조
       (그림 17)에 저장
3:     xqlClause = 사용자로부터 XQL질의를 읽는다.
4:      while(xqlClause 가 널스트링이 아니면) {
5:       xqlClause로부터 TElementName과 ruleInfo로
         분리한다.;
6:       rule = ruleInfo의 타입을 추출한다.;
7:       if (rule = 'select children' or 'select
          descendants')
8:       { rewrite(ruleInfo);
9:            ScCompose();
10:      }else if (rule = 'filters') { rewrite(ruleInfo);
11:      }else if(rule = 'Union and Intersection') {
12:           rewrite(ruleInfo);
13:      }else if(rule = 'Equivalence') {
              rewrite(ruleInfo);
14:          Compose(r2xInfo,"Equivalence", ruleInfo);
15:      }else if(rule = 'Boolean Expression') {
16:          rewrite(ruleInfo);
17:          Compose(r2xInfo,"Equivalence", ruleInfo);
18:      }
19:   } /* end while */
20:   generateSQL()
21: }
```

그림 11. 질의 처리알고리즘

줄11은 rule이 'Union'이거나 'Intersection' 일 경우에 대한 질의 처리로서, $Union$인 경우는 여러 엘리먼트를 돌려주도록 하는 연산자로서 순서와 중복이 없이 결과를 돌려준다. $Intersection$연산자인 경우는 두 엘리먼트의 인스턴스에 공통되는 내용의 집합을 돌려준다. 이 경우에도 줄12와 같이 rewrite 처리만으로 질의 결합이 가능하다. 줄13은 rule이 'Equivalence'인 경우이다.

이것은 author[last-name='Bob']과 같은 것으로서, author엘리먼트 밑에 last-name 엘리먼트가 Bob인 것이 있으면 author를 보이라는 것이다. 이 타입의 질의에는 "=", "!="와 같은 조건이 해당하지만 본 논문에서는 대소비교와 연산까지를 포함시켰다. 이 질의는 rewrite를 수행한다. rewrite수행만으로는 질의가 결합되지 않으므로 Compose함수를 이용하여 질의 결합을 완성한다. 줄15는 rule이 'Boolean Expression'인 경우이다.

이것은 author[degree and award]와 같은 형태로서 author엘리먼트 밑에 degree와 award 엘리먼트의 인스턴스가 동시에 적어도 하나이상 존재하는가를 묻는 질의이다. 이 경우도 줄13과 같은 처리를 수행한다. 줄20은 SQL을 생성하도록 모듈을 호출하여 실행하는 부분이다.

(그림 12)는 질의가 (그림 8)과 같은 질의가 들어올 경우 "{"기호 전에 나오는 엘리먼트에 대해서는 XQL질의를 위해서 (그림 18)과 같은 트리를 만드는

알고리즘 3 : rewrite는 사용자 XQL질의와 xml뷰의 일치하는 부분에 대한 정보를 저장

```
typedef struct {
  string elementName // 엘리먼트 이름
  string ruleType // t<-텍스트, r <-table, filter,
     select
  string textContent // 룰 입이 텍스트인 경우
  string tableName // 테이블 이름
  string filterName //sql에서 where에 해당하는 정보
   를 갖는다.
  string selectName // SQL에서 select 다음에 나오
   는 프로젝션 정보
  string pathInfo // 루트로부터의 경로정보를 포함
   한다.
}rxinfoValue[MAX];
1: rewrite(ruleInfo)
2: {
3:  for(i = 0; i<엘리먼트카운트(ruleInfo); i++) {
4:    EachElementName=
        ruleInfo[i]로부터 엘리먼트 이름을 추출한다.
5:    LocationFromView
        =R2X뷰에서 EachElementName에 저장된 엘리
먼트와 일치하는 엘리먼트를 발견하면 발견된 인덱스를
반환한다.
6:     r2xInfo[i].selectInfo[0]=
7:       rxinfoValue[LocationFromView].selectName
8:     r2xInfo[i].tableName[elemIndex]=
```

```
10:     r2xInfo[i].filterInfo[elemIndex]=
11:         rxinfoValue[LocationFromView].filterName
12:     r2xInfo[i].pathInfo[elemIndex]=
13:         rxinfoValue[LocationFromView].pathinfo
14:     if (r2xInfo[i].elementContent 가 아무내용도 없
        으면)
15:         r2xInfo[i].ruleType = "r"
16:     else
17:         r2xInfo[i].ruleType = "t"
18:         elemIndex++
19:   } /* end for */
20:}
```

그림 12. rewrite 알고리즘

데 이용하고, "editor{" 다음에 나오는 질의에서 엘리먼트 이름을 추출하여 R2X뷰에서 참조하여 R2X에서 참조하는 엘리먼트 이름이 존재하면 일치하는 부분에 관한 정보를 가져와서 (그림 18)과 같이 구성한다.

(그림 13)은 사용자의 XQL질의 특성이(rule type)이 'select children' or 'select descendants'인 경우 rewrite 모듈을 적용하기 전에 필요한 전처리로서 테이블이 하나이상이고 where절에 필요한 필터가 하나 이상인 경우의 질의 결합처리를 위해서 만들어놓은 모듈이다.

알고리즘 4:사용자 XQL질의에 질의 특성이(rule type)이 'select children' or 'select descendents'인 경우 rewrite 모듈을 적용하기 전에 필요한 전처리로서 테이블이 하나 이상이고 where절에 필요한 필터가 하나 이상인 경우의 처리이다.

```
ScCompose()
{
  if (r2xInfo[i].ruleType 이 "t"와 같으면) { }
  else if (r2xInfo[i].ruleType 이 "r"과 같으면)
     uniqueTable[j++] =
        (r2xinfo[i].tableName[j++]) 로부터 유일한 테
이블 이름만 추출한다.

   if (테이블이 한 개 이상인 경우 )
      질의를 결합한다.
   else {
     if (r2xInfo[i].filterInfo 가 테이블 조인이면)
       r2xinfo[i].filterinfo = ""
   }/* end else*/
}
```

그림 13. 'select children' or 'select descendants' 처리를 위한 알고리즘

(그림 14)는 사용자 XQL의 질의 특성이 'Equivalence'인 경우에 질의 합성을 위한 것이다. 예를 들어 질의가 (그림 8)과 같다고 가정하면, editor엘리먼트 내부에 있는 다음과 같은

"/monograph/editor[/monograph/title="Subclass Cirripedia"]

XQL 질의를 R2X뷰와 합성할 경우, monograph, editor, title은 모두 엘리먼트이므로 질의 결합을 위해서 R2X뷰에서 참조한다. 참조된 엘리먼트의 필터정보는 모두 합성해야한다. 위의 질의는 title이 "Subclass Cirripedia"인 monograph editor를 보이라는 질의이다. 이와 같은 질의내용을 SQL로 변환하기 위해서는 질의합성이 필요하다.

monograph을 뷰에서 참조할 경우, 내부에 룰(질의)을 포함하고 있지 않다. editor를 뷰에서 참조하면 "mono/editor"가 된다. title을 뷰에서 참조하면

알고리즘 5: 사용자 XQL의 질의 특성이(RULE TYPE)이 Equivalence 인 경우에 질의 결합을 위한 전처리 모듈

```
1: Compose(r2xInfo,"Equivalence", ruleInfo)
2: {
3:   token = ""
4:   findIndex = 0
5:   filterInfo = ruleInfo에서 필터정보를 추출한다.
6:   tokenInfo[] = filterInfo에서 토큰을 추출
       // 토큰은 "문자열", ".", "/" 이다.
7:   for(i = 0; i<UBound(tokenInfo);i++) {
8:     token = Mid(tokenInfo[i], 1, 1)
9:     if (token 이 소문자이면 ) Or
10:          token 이 "." Or token 이 "/"이면 ) {
11:       findIndex = FindSubElement(tokenInfo[i])
12:       if (findIndex >= 1){
13:         comfilter = comfilter &
14:           rxinfoValue[findIndex].tableName
15:         comfilter = comfilter & "/" &
16:           rxinfoValue[findIndex].selectName
17:       } /* end if */
18:     }else {
19:       comfilter = comfilter & tokenInfo(i)
20:     }/* end if */
20:  } /* end for */
21:  r2xinfo[i].filter[0] =  r2xinfo[i].filter[0] & " and "
22:    & comfilter
23:}
```

그림 14. 결합 알고리즘

"mono/title"가 된다. 그리고 위의 질의에서 조건인 title이 "Subclass Cirripedia"이므로, 이 조건을 결합한 모습이 (그림 9)의 형태가 되고 내부적으로는 (그림 18)과 같은 형태가 된다. (그림 9)와 같은 형태로 질의를 합성하는데 필요한 처리를 하는 것이 (그림 14)이다. (그림 14)는 사용자 XQL질의에 존재하는 엘리먼트를 R2X뷰에서 참조할 경우 참조된 엘리먼트가 존재하는지를 찾고 존재하면 참조하며, 특히 엘리먼트가 필터정보를 갖고있으면 필터를 결합하여 (그림 18)의 필터 필드와 같이 구성하는 역할을 수행한다. 이 필터 필드는 나중에 SQL의 where절로 변환하는데 이용한다.

(그림 15)의 알고리즘은 (그림 18)과 같이 질의가 합성이 완료된 후에 합성된 질의를 SQL로 변환시키는데 이용하는 알고리즘이다. 이 알고리즘은 XQL규약을 분석한 결과 질의에 일정한 패턴이 있음을 알 수 있었다.

XQL은 질의 유형에 따라 분류할 수 있고 이 분류된 질의를 가지고 SQL로 변환하기 위한 방법을 다음과 같이 제안한다.

유형에 따라서 다음과 같이 처리한다.

• "Select Children" 혹은 "Select Descendant"일 경우

"select ~ from ~ where ~ "형태로 변환시킬 수 있다.

• "filter"일 경우

특정 엘리먼트 하부에 존재하는 엘리먼트의 인스턴스가 적어도 하나이상 존재하느냐는 질의이므로 이는 SQL로 "select ~ from ~ where ~ and exists (select ~ from ~ where ~)의 형태로 변환시킬 수 있다.

• "Equivalence"일 경우

모든 필터들을 "and"로 연결하는 형태로서 SQL로 "select ~ from ~ where ~ and ~ and ~ ...~ "로 변환시킬 수 있다.

• "Union and Intersection"의 경우

두 개 이상의 SQL을 Union으로 결합하거나 Intersection으로 결합하는 형태의 질의가 된다.

• "Boolean Expression"의 경우

특정한 엘리먼트들이 모두 또는 일부분 존재하는지 묻는 질의이다. 이것의 SQL로 "select ~ from ~ where ~ exists(select ~ from ~ where ~) and exists (select ~ from ~ where ~) . . ." 형태로 변환시킬 수 있다.

지금까지 설명한 내용이 (그림 15)의 알고리즘에 대한 설명이다.

알고리즘 6: 질의 결합알고리즘이 수행된 다음에는 결합된 내용이 자료구조에 저장되게 된다. 저장된 자료구조(트리구조)를 이용하고, 질의 특성을 이용하여, 자료구조(트리구조)에 저장된 새로운 R2X뷰를 SQL로 변환하는 알고리즘

```
generlateSQL()
{
   for (i=0; i<ubound(r2xinfo()) {
      if (rule = 'select children' or 'select descendants')
{
            r2xInfo(i).sqlUserQuery = select ~ from ~
where ~
      }else if (rule = 'filters') {
            r2xInfo(i).sqlUserQuery = select ~ from ~
where ~  and exists(select ~  from ~  where ~ )
        }else if(rule = 'Equivalence') {
            r2xInfo(i).sqlUserQuery = select ~ from ~
where ~ and ~  and ~
        }else if(rule = 'Union and Intersection') {
            r2xInfo(i).sqlUserQuery = (select ~ from ~
where ~ ) union (select ~ from ~ where ~)
        }else if(rule = 'Boolean Expression') {
             r2xInfo(i).sqlUserQuery = (select ~ from ~
where ~  exists(select ~ from ~ where ~) and exists
(select ~ from ~ where ~)  and exists(select ~ from ~
where ~)
        }
    }
}
```

그림 15. SQL생성 알고리즘.

6. 구현

(그림 16)은 R2X뷰를 위한 BNF 문법이다. 이를 바탕으로 R2X뷰에 대한 뷰 트리(그림 17)를 구성한다.

(그림 17)은 (그림 3)을 읽어서 (그림 4)와 같은 정보로 구성된 트리 구조를 나타낸 것이다.

```
ELEMENT_INFO ::= ELEMENT_NAME "(" VARIABLEDECL ")"
  "{" BODY "}" | "|" ELEMENT_NAME "(" VARIABLEDECL ")"
  "{" BODY "}"

VARIABLEDECL ::= VARIABLE ":=" TABLENAME

BODY ::= ELEMENTCONTENT BODY? | ELEMENTRULE BODY? |
         ELEMENT_INFO BODY | ELEMENT_INFO

ELEMENTCONTENT ::= TEXT

ELEMENTRULE ::= "$"VARIABLE [ FILTERINFO ] / SELECTINFO

FILTERINFO ::= ELEMENTNAME OP VALUE |
       "$"VARIABLE/SELECTINFO                    OP
"$"VARIABLE/SELECTINFO

SELECTINFO ::= TEXT

OP ::= "=" | "<" | ">" | "!="
```

그림 16. R2X뷰를 위한 문법

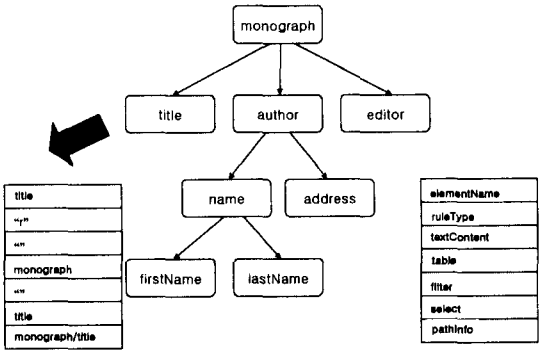

그림 17. XML뷰 저장구조

(그림 18)은 XML뷰에 사용자 XQL을 맵핑한 다음 질의를 재 진술(Rewrite)한 결과로 만들어진 새로운 XML뷰(R2X)이다.

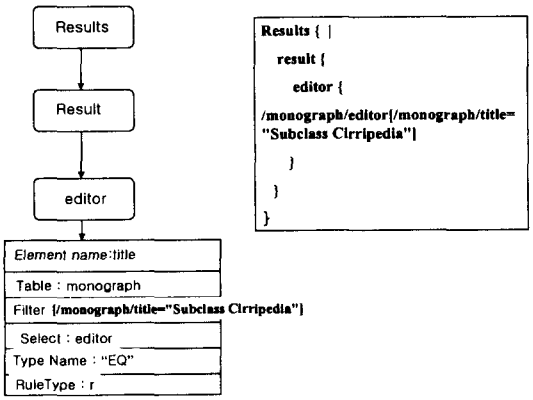

그림 18. R2X와 User XQL의 결합

(그림 8)의 monograph, editor, title 엘리먼트를 R2X뷰에서 찾아 일치하는 엘리먼트가 존재하면 룰을 (그림 18)과 같이 결합알고리즘을 이용하여 결합하고, (그림 18)과 같은 결과를 SQL변환 알고리즘을 이용하여 변환한 다음 ODBC를 이용하여 데이터베이스에 질의하고 질의 결과물을 사용자에게 제공할 수 있다.

본 논문에서 다룬 연구에 대한 실험을 위해서 본 시스템은 펜티움 500Mhz 128Mbyte 의 메인 메모리 시스템에 운영체제는 윈도우ME로 하였으며, Visual Basic 6.0을 이용하여 구현하였다.

(그림 19)는 질의 처리시스템에서 XML뷰를 선택하고 XQL질의를 입력한 화면이다.

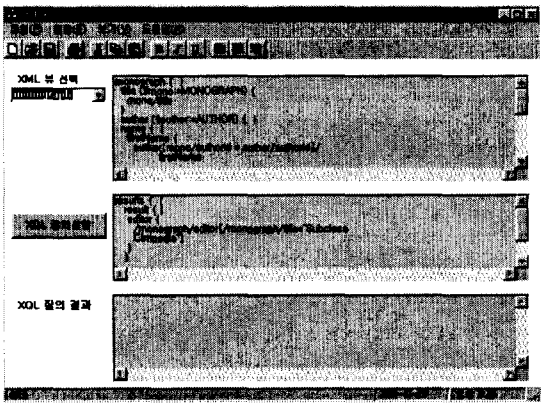

그림 19. 질의시스템에 XML뷰를 이용하여 XQL질의를 입력한 모습

(그림 20)은 질의시스템에서 XQL질의 실행한 결과화면을 보여주고 있다.

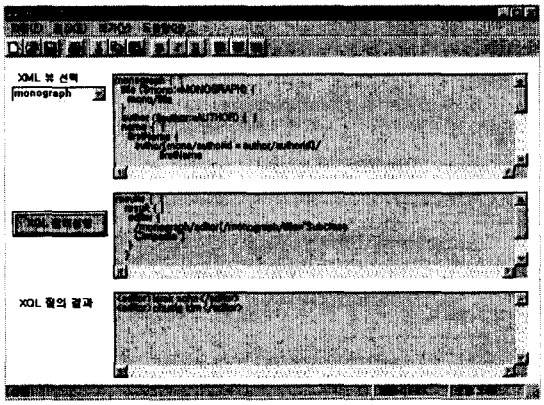

그림 20. 질의시스템에 XQL질의를 실행한 결과화면

(표 1)은 본 논문에서 제안한 시스템의 성능평가로서 단계별 실행시간 및 질의 유형별 총시간을 보인 것이다.

표 1. XQL 질의처리의 단계별 실행시간

구 분 (12만개의 Element를 대상으로 실험)	수행시간(ms)		
	질의 유형1	질의 유형2	질의 유형3
XQL 뷰(RXL) 생성	28ms	28ms	28ms
XQL 파싱	19ms	19ms	19ms
XQL-2-SQL 변환	1131ms	1089ms	1067ms
SQL 질의처리	1525ms	1433ms	1421ms
총 시간	2703ms	2569ms	2535ms

(표 1)에 있는 실험을 위한 질의 유형은 다음과 같다.

- "질의유형1" 의 질의형태
  /monograph/author/name/firstName
- "질의유형2" 의 질의형태
  /monograph/*/firstName
- "질의유형3" 의 질의형태
  //firstName

뷰를 이용하지 않은 즉, 관계형 데이터베이스에 XML데이터를 트리 형태로 삽입한 후, XQL로 질의하고 XQL을 SQL로 변형한 후 SQL로 결과를 추출하는 실험에서는 총 검색시간이 10분 이상이 소요되었다. 뷰를 사용하지 않는 시스템은 질의경로를 알아내기 위해서 Self-Join이 질의 경로만큼 이루어지므로 이 부분에서 많은 시간이 소요된다.

질의유형에 따른 질의 처리시간에서는 검색시간에서 큰 차이를 보이지 않았다.

본 논문에서 제안한 시스템은 데이터를 트리 형태로 저장하여 XQL로 질의하는 시스템이 아니므로 이 부분이 상대적으로 시간을 감소시키는 요인으로 평가된다.

7. 결 론

XML은 웹 상에서 상업 데이터 교환을 위한 표준이 되었다. 현재 대부분의 전자상거래에서 XML문서를 이용하여 기업간, 기업과 기관간에 기업과 개인간에 유용한 정보를 교환하기 위한 문서 포맷으로 XML을 이용하고 있다.

정보교환을 위해 과거에 사용하는 데이터는 대체로 데이터베이스에 저장된 데이터이거나 아니면 파일형태로 보관된 데이터일 것이다. 현재 정보교환을 위해서 새롭게 만든 데이터는 XML형태의 데이터일 것이다. XML형태의 데이터의 보관을 위해서나 혹은 문서에서 유용한 정보를 검색하거나 문서를 갱신하는 면에서 데이터베이스를 이용한 문서의 보관과 관리는 매우 중요하다.

또한 과거에 만들어진 데이터베이스에 보관된 유용한 데이터를 XML형태로 만들어서 문서 교환을 하는데 이용한다면 매우 가치가 있을 것이다.

본 논문에서 제안한 시스템은 전통적인 관계형 데이터베이스에 저장된 유용한 데이터를 XML형태로 보고 질의할 수 있는 RXL 뷰를 설계함으로써 관계형 데이터베이스에 XQL로 질의를 할 수 있고 질의 결과를 효율적으로 획득할 수 있는 시스템을 설계 및 구현하였다.

논문에서 제안한 시스템의 문제점은 XQL질의어는 XQL질의어를 5가지 유형으로 분류하여 유형에 맞는 질의가 들어오면 완벽한 질의처리가 가능 하지만 여러 유형이 혼합된 경우 질의처리 지원에 문제가 있다. 따라서, 향후 개선해야할 것으로 본다. 또한, XQL은 표준화 기구에서 XML을 위한 표준언어로 고려하고 있지 않은 언어이다. XQL을 위한 질의 처리시스템을 설계한 이유에 대해서는 관련연구에서 밝혔다. XQuery는 표준화가 될 것으로 기대되는 언어이므로 향후의 연구과제로서 XQuery를 지원하는 질의처리 시스템의 설계 및 개발은 중요한 이슈가 될 것으로 기대한다.

참 고 문 헌

[1] Extensible Markup Language (XML) 1.0 http://www.w3.org/TR/2000/REC-xml-20001006

[2] S.Abiteboul, D.Quass, J.McHugh, J.Widom, and Wiener. The Lore query language for semistructured data. International Journal On Digital Libraries. 1(1): 68-88, April 1977.

[3] Peter Buneman, Susan Davidson, Gerd Hil-

lebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In Proceedings of ACM SIGMOD International Conference on management of Data, 1996.

[4] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In Richard Snodgras and Marianne Winslett, editor, Proceeding of 1994 ACM SIGMOD International Conferences on Management of Data, Minneapolis, Minnesota, May 1994.

[5] A. Mendelzon, G.Mihaila, and T.Milo. Querying the world wide web. In Proceedings of the Fourth Conference on Parallel and Distributed Information System, Miami, Florida, December 1996.

[6] Serge Abiteboul. querying semi-structured data. In ICDT, 1997.

[7] S. Nestorov, J. Ullman, J.Wiener, and S. Chawathe, Representative objects: concise representation of semistructured, hierarchical data. In ICDE, 1997.

[8] S.Nestorov, S.Abiteboul, and R.Motowani. Inferring structure in semistructured data. In Proceedings of the Workshop on Managemen of Semi-structured Data. 1997.

[9] Roy goldman Jennifer Widom, DataGuides : enabling query formulation and optimization in semistructured databases. In VLDB, September 1997.

[10] S. Banerjee, et. al., "Oracle8i - The XML Enabled Data Management System", ICDE Conf., San Diego, March 2000, pp. 561-568.

[11] J.Cheng, J. Xu, "XML and DB2", ICE Conf., San Diego, March 2000, pp. 569-573.

[12] Microsoft Corp. http://www.microsoft. com/XML

[13] XML Query Requirements, http://www.w3.org/TR/xmlquery-req

[14] M. Fernandez, A. Morishima, D. Suciu, "Efficient Evaluation of XML Middleware Queries", SIGMOD Conf., Santa Barbara, May 2001, pp. 103-114.

[15] M. Fernandez., W. Tan, D. Suciu, "SilkRoute: Trading Betwen Reltions and XML", World Wide Web Conf., Toronto, Canada, May 1999.

[16] XML Query Language (XQL), http://www.w3.org/TandS/QL/QL98/pp/xql.html

[17] A.Deutsch, M.Fernandez, D.Florescu, A.Levy, and D.Suciu.XML-QL: A query language for XML, 1998. http://www.w3c.org/TR/NOTE-xml-ql/.

김 천 식

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

손 기 락

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
e-mail : ksohn@hufs.ac.kr