방향 그래프 기반 XML 데이터 모델과 대수 구현

박 성 희† · 최 은 선† · 류 근 호††

요 약

데이터 교환을 위한 표준 형식으로 XML 활용 증가에 따라 데이터베이스 분야에서 XML 처리의 중요성이 증가하고있다. 현재까지 XML 데이터모델과 정규표현 질의 같은 복잡한 질의처리를 위한 XML대수에 관한 연구가 수행되고 있지만 미디에이터 시스템처럼 XML질의 처리 시 기능이 제한적이다. 따라서 이 논문에서는 반 구조데이터 모델을 확장한 방향 그래프 기반 XML 모델을 설계하고 XML 질의를 위한 XML 대수 연산을 정의하며 그 구현기법을 제시한다. XML 대수 연산 구현을 위해 물리적 저장소인 RDBMS를 접근하기 위한 접근 메소드와 패스 인덱스를 이용하여 알고리즘을 구현한다. 아울러 제안 알고리즘의 효율성을 보이기 위하여 반 구조 특성을 가지는 EST유전체 서열에 대한 XML 문서를 대상으로 성능을 평가한다.

Implementation of Algebra and Data Model based on a Directed Graph for XML

Sung Hee Park† · Eun Sun Choi† · Keun Ho Ryu††

ABSTRACT

As XML become more popular for encoding data and exchanging format on the web, recent work on processing XML Document in DBMS has been performed. However, there is no formal data model for XML, and there is lack of research on XML algebra for processing complex XML query and even the mediators have many restrictions. Therefore, this paper proposes formal data model and algebra based on directed edge labeled graph for XML query. To implement algebra, not only algorithms of operation for algebra are presented, but also they are implemented using access method and path index based on RDBMS or ORDBMS. In particular, experiments to show the effectiveness of the implemented algebra are performed on XML documents on EST data which are semistructured data.

키워드 : XML문서(XML), 데이터모델(Data Model), 대수(XML algebra)

1. 서 론

XML(Extensible Markup Language)[20]은 1998년 W3C (World Wide Web Consortium)에 의해 웹을 기반으로 하는 구조화된 문서를 기술하는 표준으로 지정되었다. 기존의 구조적인 데이터와는 다르게 스키마가 고정되지 않고 빠른 변경이 요구되는 웹 정보를 표현하는 XML은 반 구조적 데이터(SemiStructured Data : SSD)의 특징을 보여준다. 기존 구조 데이터를 관리하기 위한 데이터베이스 관리시스템에서는 XML같은 SSD를 잘 처리할 수 없으므로 이러한 SSD의 특징을 지원할 수 있는 데이터베이스 관리시스템의 추가적 기능이 필요하다.

데이터베이스에서 XML을 지원하기 위한 접근 방법으로는 시스템 구조 측면에서 서버 접근 방법과 미디에이터 접근방법으로 나누어지며 미디에이터 방법으로는 데이터 통합과 데이터 변환을 위한 것으로 나누어진다. 데이터 통합을 위한 미디에이터는 여러 개의 다른 소스 데이터에 대한 공통된 뷰를 제공하기 위해 서로 다른 저장소의 데이터를 통합한다. 데이터 변환을 위한 미디에이터는 이질적인 데이터 모델간의 데이터 변환을 목적으로 한다. 따라서 관계형 데이터베이스 기반의 미디에이터 시스템은 관계형 데이터를 XML형식으로 변환함으로써 관계형 데이터에 대한 XML뷰를 제공하고, 관계형과 XML간의 이질적인 데이터 모델을 매핑하고 XML뷰에 대한 질의를 SQL로 변환하여 관계형 데이터베이스에서 실행시킨다.

변환을 위한 이러한 미디에이터 시스템은 기존 데이터에 대한 XML 뷰를 제공한다는 장점이 있는 반면 다음과 같은 단점을 갖는다. 첫째, 다른 데이터 모델로 변환하기 위한 매핑 모델이 요구된다. 이때 XML과 같은 SSD가 가지는 특성

† 준 회 원 : 충북대학교 대학원 전자계산학과
†† 종신회원 : 충북대학교 컴퓨터과학과 교수
논문접수 : 2001년 4월 24일, 심사완료 : 2001년 9월 14일

인 스키마가 정형적이지 않으며 타입을 알 수 없다는 원천적인 특징을 지원하기가 쉽지 않다. 둘째로는 기존 데이터에 대한 스키마 정도를 유지하는 가상 뷰를 유지해야한다. 그러나 반 구조적 데이터는 빠르게 변화하기 때문에 이러한 변화가 일어날 때마다 자주 뷰를 갱신해야 한다는 단점이 존재한다. 마지막으로 질의처리의 경우 XML 질의를 SQL로 변환하기 위해 새로운 변환을 위한 언어가 필요하다. 또한 XML 질의언어가 가지는 다양한 연산을 SQL로 변환하기 어렵다. 따라서, 기존의 관계형 데이터베이스 시스템의 질의 최적화 기술을 XML분야에 적용하기 위해서는 우선적으로 XML에 대한 데이터 모델과 질의 처리를 위한 XML 대수 정의가 선행되어야 한다.

이 논문에서는 방향 그래프 기반 XML 데이터 모델을 설계하고 이 모델에 연산을 수행할 수 있도록 XML대수 연산을 정의하고 그 구현기법을 제시한다.

XML 모델 설계는 SSD모델인 방향그래프 모델을 확장한다. 또한 XML은 엘리먼트 태그, 애트리뷰트, 엘리먼트간의 참조 관계와 태그의 순서를 갖는 측면에서 SSD모델과 다르므로 이러한 XML의 고유한 특성을 반영하도록 설계한다. XML-QL질의를 물리적 연산자로 변환하기 위한 XML 대수 연산은 전통적인 관계형 대수에서 처리하기 어려운 정규 경로 표현식 및 질의 결과를 XML문서로 반환하기 위한 연산자로 구성된다. 더불어 제안된 XML 대수 연산을 기존의 RDBMS 또는 ORDBMS상에 구현하기 위한 기법 및 알고리즘을 제시한다. 이러한 XML 대수 연산은 질의 최적화에 이용할 수 있으며 질의 대수 연산을 사용하지 않는 미디에이터 시스템에서 발생되는 복잡하고 다양한 XML질의 처리 제약성을 해결할 수 있으며 기존 데이터베이스 시스템을 이용하여 정규 경로 표현식과 같은 복잡한 연산의 구현을 효율적으로 할 수 있다.

이 논문의 구성은 다음과 같다. 2장에서는 SSD 및 XML 관리시스템과 이러한 시스템에서 이용하는 데이터 모델과 대수에 대한 연구 동향을 살펴본다. 3장에서는 그래프기반 XML 데이터 모델을 기술하고 이러한 그래프 모델을 구성하는 정점과 간선, 이들의 속성을 상세히 설명한다. 4장에서는 XML 질의 대수 연산을 제시하고 5장에서는 이러한 질의를 이용하여 XML-QL 질의의 대수적 변환을 기술한다. 6장에서는 제시된 대수 연산을 구현하기 위한 기법 및 알고리즘을 살펴보고 제시된 알고리즘에 대한 성능 평가를 다룬다. 7장에서는 결론 및 향후연구를 기술한다.

2. 관련 연구

국내외에서 XML 및 반 구조적 데이터 관리를 위한 연구동향은 XML문서의 모델링과 저장[7, 9, 11, 13, 16, 19], 구조 추출 및 인덱스 방법[2, 12, 20], XML 질의 언어[6, 21]와 질의처리 및 최적화[2, 17]분야로 나눌 수 있다. 이러한 각 연구분야를 통합하여 XML문서 저장 및 질의를 위한 XML 관리 시스템에 대한 연구가 진행중이다.

XML관리시스템은 백엔드 저장 관리 시스템으로 파일시스템, 관계형[7, 8, 10], 객체지향[14]과 객체관계형[3] 또는 반 구조 데이터를 관리하기 위한 독자적인 저장 관리 시스템[16]을 이용한다.

LORE[16]는 반 구조 데이터를 위한 최초의 구현된 데이터베이스 시스템이다. ODMG에서 제안한 객체지향 표준 모델을 확장한 OEM모델과 OQL을 확장한 LOREL질의 언어를 이용한다. 질의 최적화[17]에서 기존의 비용기반 최적화 방법을 SSD에 적용하여 비용모델과 비용을 계산하기 위한 데이터베이스의 통계적 정보를 제시하였다. 기존의 구조적 데이터와 통합을 위한 별도의 시스템이 필요하며 SSD에 기반 하였기 때문에 XML을 지원하기 위한 시스템의 변경이 진행중이다.

관계형 데이터베이스 시스템을 물리적 저장소로 이용한 연구로는 Strudel[8], STORED[7], SilkRoute[10] 등이 있다. 이러한 연구에서는 XML 문서를 저장하기 위해 XML문서를 구성하는 엘리먼트 및 에트리뷰트를 관계형 테이블로 매핑한다. 이 방법은 RDBMS를 이용함으로써 기존의 응용프로그램에 XML 뷰를 제공할 수 있다. 반면, 관계형 모델이 속성의 셋 값을 지원하지 않기 때문에 여러 개의 테이블로 나누어 저장하게되므로 많은 조인이 발생한다. 또한 질의 결과를 복잡한 XML문서로 반환하기가 어렵다는 단점이 있다.

XML 데이터 모델에 관한 연구는 1995년 스탠포드대학의 TSIMMIS 프로젝트에서 처음으로 SSD를 위한 데이터 모델로써 OEM개발에서부터 시작되었다. LORE시스템에서 OEM[13, 16, 18]을 사용하였고 SSD를 위한 데이터 모델로서 제안되었다. 따라서 OEM 모델은 XML문서의 참조관계와 애트리뷰트와 같은 특성 처리가 미흡하여 OEM 모델에서 XML 지원을 위해 계속해서 변경하고 있다.

기존 데이터베이스시스템의 질의처리에서는 고수준 질의언어를 대수로 변환하여 실행하였다. 이와 유사하게 SSD를 위한 질의처리를 위해 기존 관계형 대수를 SSD에 적용하기 위한 연구가 시도되었다. 이러한 연구에 대한 결과는 아래와 같이 요약된다.

Abiteboul[1]과 Christophidex[4]는 Clue[5]을 확장해서 GPE (General Path Expression)를 지원한다. Christophidex[4]는 GPE를 지원하기 위해 두개의 연산자 S_int와 D_inst를 추가하였다. Mecca[15]는 Araneus프로젝트의 일부로서 전통적인 SPJ대수를 웹 데이터를 항해할 수 있도록 확장하였으며 질의 실행 계획과 비용 기반 최적화를 위한 비용 모델을 제시하였다.

Mchugh[16, 17]는 SSD를 위해 전통적 DBMS의 질의 실행을 확장하여 질의 최적화 및 실행을 위한 질의 실행 모델

을 제시하였다. 논리적 대수를 물리적 대수로 변환하는 방법을 제시하였으나 SSD를 처리하기 위한 논리적 대수에 대한 고려가 없었으며 질의 결과를 XML로 반환할 수 있는 XML 질의처리 대수를 제시하지 않았다.

Fernandez[9]에서는 XML 질의에 대한 대수를 정의하였으나 질의 모델로서 문서간의 조인을 지원하지 않는 XSLT를 바탕으로 하기 때문에 조인을 지원할 수 있는 대수를 제시하지 않았다. 또한 애트리뷰트와 엘리먼트를 결합하여 이들간에 구별을 두지 않았으며 코멘트와 PI에 대한 처리를 배제하였다.

3. 방향그래프 기반 XML 데이터 모델

XML 문서는 방향 그래프(Directed Graph) G = (Vertex, Edge)로 표현된다. 정점집합 Vertex = {$V_{element}$, V_{text}}는 엘리먼트를 나타내는 엘리먼트 정점 $V_{element}$와 데이터 값을 포함하는 데이터 정점 V_{text}으로 구성된다. Edge는 간선 집합으로 Edge = {E, A, R}로 표현한다. E는 엘리먼트 간선집합, A는 애트리뷰트 간선집합, R은 참조간선집합을 나타낸다. (그림 3.1)은 EST 유전체 서열정보에 대한 XML 문서를 제시한 방향 그래프 기반 데이터 모델로 표현한 예이며 실선은 엘리먼트 간선, 데쉬는 참조 간선, 그리고 점선은 애트리뷰트 간선을 나타낸다.

엘리먼트 간선은 부모 엘리먼트와 자식 엘리먼트의 연관성을 나타내는 간선이다. 자식 엘리먼트는 데이터 값이거나 하위 엘리먼트가 된다. 애트리뷰트 간선은 엘리먼트와 애트리뷰트 값과의 관계를 나타낸다. 마지막으로 참조 간선은 엘리먼트가 다른 엘리먼트를 참조하는 참조 관계를 나타낸다.

문자 수준에서, IDREF는 문서 안에서 다른 객체의 ID에 대한 값을 참조하는 애트리뷰트 타입이다. 논리적 수준에서, IDREF는 그래프에서 두 노드사이의 참조관계를 나타내는 간선이다. 이러한 IDREF를 처리하기 위한 두 가지 방법이 존재한다. 만일 IDREF간선을 연결하면, XML 데이터 모델은 사이클이 존재하는 그래프 모델이 된다. 이러한 그래프 모델에서 질의는 보다 쉽게 할 수 있으나 질의엔진은 이러한 그래프구조를 처리하기 위해 더 어려운 과정을 처리한다.

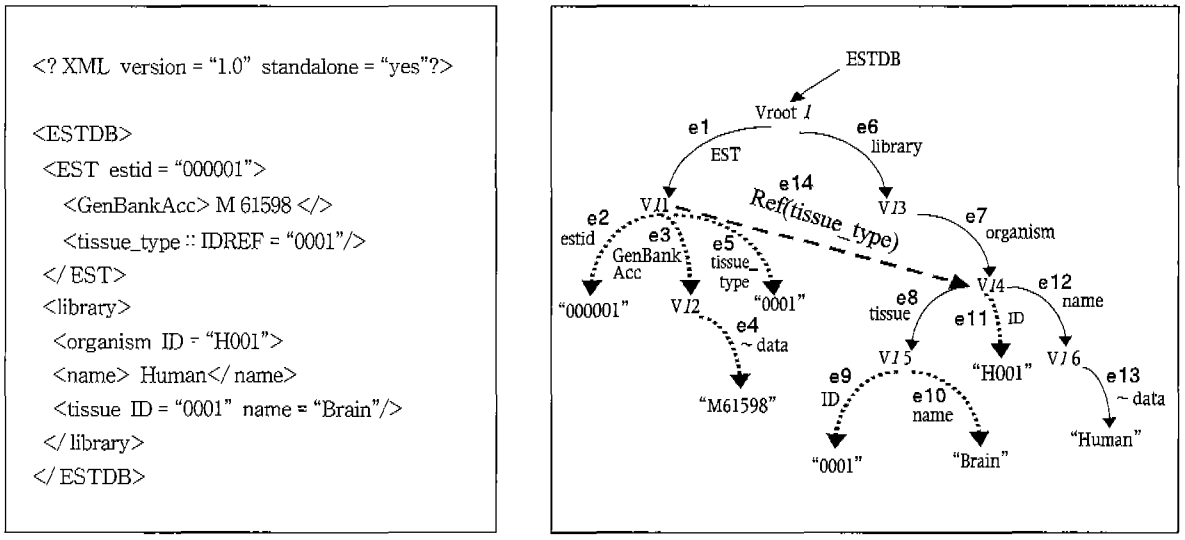
다른 방법으로 IDREF가 가리키는 객체에 대한 포인터를 나타내지 않는 트리 구조이다. 트리 구조에서는 객체에 대한 항해를 쉽게 할 수 있는 반면, IDREF 간선을 항해하기 위해 조인을 명시적으로 표현하여 질의가 복잡해진다. 이 논문에서는 IDREF를 간선으로 나타내는 그래프 구조로 나타낸다.

XML은 비 구조 데이터의 텍스트 형식이므로 생성될 때 순서를 갖는다. XML문서에서 엘리먼트의 순서가 중요시되는 문서와 그렇지 않은 경우가 발생한다. 따라서 사용자가 순서가 중요할 때와 중요하지 않을 때를 선택할 수 있어야 한다.

같은 부모 엘리먼트를 가지는 자식 엘리먼트들은 순서를 가진다. 엘리먼트 간선에 의해서 연결되는 자식 엘리먼트의 순서는 부모 엘리먼트에서 나타나는 자식의 순서로 나타낸다. 애트리뷰트 간선의 순서는 데이터모델에서 정의하지 않는다. 다수의 값을 참조하는 다수참조간선의 순서는 참조 순서 규칙에 의해서 참조간선의 순서가 정해진다.

3.1 정점 속성

XML엘리먼트의 태그는 엘리먼트 정점 v로 나타내며 이러한 엘리먼트 정점은 v∈$V_{element}$에 포함된다. (그림 3.1)에서 엘리먼트 정점은 {*V1*, *V2*, *V3*, *V4*}가 되고 *V1*과 같이 엘리먼트 정점 v는 유일하고 논리적이며 변경할 수 없는 엘



(그림 3.1) XML 문서와 데이터 그래프

리먼트정점 식별자를 갖는다. 데이터 모델에서는 식별자에 대한 형태를 제한하지 않으나 이 논문에서는 엘리먼트<X/>에 대한 엘리먼트 정점 식별자를 $V_{k/}$로 나타낸다. k는 문서에 할당된 식별자이고 J는 정점에 할당된 일련번호이다.

엘리먼트가 포함하는 내용을 나타내는 데이터정점은 $v \in V_{type(v)}$로 나타내며 데이터를 포함한 부모 엘리먼트 정점으로 구별할 수 있기 때문에 식별자를 갖지 않는다. 이 논문에서는 데이터 정점이 포함하는 데이터 타입을 문자형으로 다루기 때문에 데이터 정점은 V_{text}로 나타낸다.

함수 vertex(x)는 XML엘리먼트 또는 데이터 값에 해당하는 정점을 그래프에서 생성한다. X가 XML엘리먼트 태그인 경우, 이 함수는 엘리먼트 식별자와 타입 element를 반환한다. 그렇지 않은 경우에는 X의 데이터 값과 데이터 타입을 반환한다. 즉, 변환 함수 vertex는 아래와 같이 표시한다.

$$\text{If X is XML element, } vertex(X) \in V \equiv (get\_eid(x), element) \in V_{element}$$
$$\text{otherwise, } vertex(X) \in V \equiv (X, type(X)) \in V_{type(x)}$$

데이터 정점과 엘리먼트 정점 모두 type과 value의 기본적인 속성을 갖는다. 엘리먼트 정점의 type속성은 element값을 반환하며 value 속성은 엘리먼트 식별자를 반환한다. 데이터 정점에서 type 속성은 데이터 정점이 저장하는 데이터 타입을 나타내며 value는 저장되는 데이터 값을 가진다. 간선과 순서정보에 기초하여 다음과 같은 속성이 유도된다.

- Name : $V \rightarrow T_{eid}$는 엘리먼트 태그이름을 반환하며 아래와 같이 표현된다.
  $Name(v) = \{get\_name(e) \mid \exists e \in E : child(e) = v\}$
- parent : $V \rightarrow V_{element}$ 부모 엘리먼트 정점을 반환한다.
  $parent(V) = \{parent(e) \mid \exists e \in E : child(e) = v\}$
- Referredby : $V \rightarrow \{V_{element}\}$은 정점을 참조하는 엘리먼트 정점집합을 반환한다.
  $referredby(v) = \{parent(e) \mid \exists e \in R : child(e) = v\}$
- Childelements : $V \rightarrow \{E\}$는 정점에서 나가는 엘리먼트 간선집합을 반환한다.
  $childelements(v) \equiv \{e \mid \exists e \in E : parent(e) = v\}$
- Attributes : $V \rightarrow \{E\}$는 정점이 포함하는 애트리뷰트 간선집합을 반환한다.
  $Attribute(v) \equiv \{e \mid \exists e \in A : parent(e) = v\}$
- References : $V \rightarrow \{R\}$는 정점에서 나가는 참조 간선집합을 반환한다.
  $references(v) \equiv \{e \mid \exists e \in R : parent(e) = v\}$

3.2 간선 속성

3.2.1 엘리먼트 간선

엘리먼트 간선(Element Edges : E)은 부모 엘리먼트와 자식 엘리먼트의 연관성을 나타내는 간선이다. 자식 엘리먼트는 데이터 값을 나타내는 데이터 정점이나 하위 엘리먼트 정점이 될 수 있다. 각각의 간선 $e \in E$는 ($name \in T_{name}$, $parent \in V_{element}$, $child \in V_{value}$)와 같이 나타낸다. name은 엘리먼트 간선의 이름을 나타내고 parent는 간선의 부모 정점을 나타낸다. ~data, ~comment, ~PI 간선의 경우는 child는 데이터 정점이 되고 이런 경우를 제외하고 child는 간선의 하위 엘리먼트 정점이 된다.

XML 엘리먼트 x가 들어오는 간선 e를 갖는다면, $e \in E$이고 $e = (name(x), vertex(parent(x)), vertex(x))$와 같이 표현된다. name(e)는 엘리먼트 x의 이름을 반환하고, parent(x)는 엘리먼트x를 포함하는 엘리먼트를 반환한다. vertex(x)는 엘리먼트 x를 정점 $v \in V$로 매핑하기 위한 함수이다. 따라서, $vertex(x) = v$이고 엘리먼트 정점 v는 부모 정점으로부터 정점 v까지를 연결하는 엘리먼트 간선 e를 갖는다.

<표 3.1>은 (그림 3.1)의 XML 문서와 그래프에서 엘리먼트 간선 속성을 나타낸 예이다. 엘리먼트 간선은 name, parent, child처럼 기초 속성과 간선의 순서정보에 대한 속성을 갖는다. 순서정보 속성은 동일한 부모 정점의 자식 엘리먼트 중 다음에 연속되는 간선을 반환하는 succ와 현재 엘리먼트 간선 순서를 나타내는 eo속성을 갖는다. null은 값이 존재하지 않음을 나타낸다.

엘리먼트 간선 중 간선 이름이 ~data인 경우에는 데이터를 포함하는 데이터 정점과 엘리먼트 정점을 연결하는 간선이다. (그림 3.1)에서 ~data간선은 name 엘리먼트의 내용인 "Human"을 저장하는 데이터정점과 name엘리먼트 정점을 연결한다. ~comment와 ~PI간선은 ~data 간선과 유사하게 XML 문서에 대한 코멘트와 처리명령들을 포함한 정점을 포함하는 간선이다.

〈표 3.1〉 엘리먼트 간선

Element Containment Edge					
Edge	name	parent	child	Order	
				eo	succ
e1	"EST"	Vroot*1*	V*J1*	1	e6
e3	"GenBankAcc"	V*J1*	V*J2*	2	e5
e4	"~data"	V*J2*	"M61598"	3	null
e6	"library"	Vroot*1*	V*J3*	4	null
e7	"organism"	V*J3*	V*J4*	5	null
e8	"tissue"	V*J4*	V*J5*	6	e11
e12	"name"	V*J4*	V*J6*	7	null
e13	"~data"	V*J6*	"Human"	8	null

〈표 3.2〉 애트리뷰트 간선의 예

Attribute Edge				
Edge	name	parent	child	type
e2	"estid"	V*J1*	"000001"	text
e5	"tissue_type"	V*J1*	"0001"	IDREF
e9	"ID"	V*J5*	"0001"	ID
e10	"name"	V*J5*	"Brain"	text
e11	"ID"	V*J4*	"H00"	ID

3.2.2 애트리뷰트 간선

애트리뷰트 간선(Attribute Edge : A)은 엘리먼트와 애트리뷰트의 관계를 나타내는 간선이다. 애트리뷰트 간선은 엘리먼트 간선과 같이 parent, value, name, type같은 속성을 갖는다. XML 엘리먼트 <X/>가 애트리뷰트 a를 가진다면, 애트리뷰트 a는 name, parent, value와 같은 기본속성을 갖는다. name은 에트리뷰트의 이름을 나타내고 parent는 애트리뷰트를 갖는 엘리먼트이고 value는 애트리뷰트 값을 나타낸다. 애트리뷰트 간선 e는 $e \in A$이고 e = (name(a), vertex(x), value(a))와 같이 정의할 수 있다. <표 3.2>는 (그림 3.1)의 XML문서와 그래프에서 에트리뷰트 간선을 나타낸 예이다.

3.2.3 참조 간선

참조 간선(Reference Edge : R)은 엘리먼트 사이의 참조 관계를 나타낸다. 애트리뷰트 속성 중 IDREF, IDREFS, foreign key, XLink, URL와 같은 참조타입으로 식별되는 경우 그래프모델에서 참조간선집합 R에 속한다. 이러한 R에 속하는 참조간선 r_i는 다음과 같이 정의한다.

$$r_i = (parent(e), \{e\}, refattr(child_i(e)))$$

e는 참조 타입을 자식 정점으로 갖는 애트리뷰트 간선이다. $child_i(e)$는 IDREFS타입의 애트리뷰트 중 하나이상의 참조를 가지는 것 중 i번째 참조되는 정점을 나타낸다.

참조 간선은 parent, child, type의 세 가지 기초적인 속성을 갖고 참조정보를 제공하는 애트리뷰트 또는 엘리먼트 간선에 대한 정보를 나타내는 refedge속성을 갖는다. 타입 속성은 간선이 참조간선임을 나타내고 XLink또는 URL같은 참조종류를 나타낸다. 즉, 다음과 같은 {Rid, Rkey, Rlink, Ruri, Rjoin} 참조 타입 중 하나 값을 가진다. Name 속성은 참조 간선의 이름을 나타낸다. 다음 <표 3.3>은 (그림 3.1)에서 나타나는 참조간선에 대한 속성을 보여준다.

〈표 3.3〉 참조 간선의 예

Reference Edge			
Edge	parent	refedges	child
c14	V*1*1	{e5}	V*1*4

4. 방향그래프 기반 XML 대수 연산

XML은 SSD에 속하며 구조가 약하고 구조의 변경이 빠르므로 예외상황이 발생할 수 있다. 따라서 XML 대수는 그래프 구조, 타입과 그래프구조의 이질성, 참조 관계, 순서와 같은 XML이 갖는 특성에 대한 연산을 처리할 수 있어야 한다. XML 대수는 기본적으로 항해(Φ), 셀렉션(σ), 조인(⋈)과 질의 결과를 XML문서로 생성하기 위한 결과 생성 대수들로 구성된다. 그 외에도 구조적인 순환과 정규표현을 연산하기 위한 구조적 순환(*) 연산으로 구성된다.

이 절에서는 XML 대수 연산을 O[f(X)](X : A)와 같은 연산형태로 정의하며 이 연산은 A에 포함된 각각의 엘리먼트는 X변수에 결합되고 O의 의미를 가지는 연산으로 f(x)가 적용된다. 이러한 XML 대수 연산의 예문은 (그림 3.1)의 EST(Expressed Sequence Tags) 정보에 대한 XML문서에 바탕을 두었다.

4.1 항 해

경로 항해(Φ) 연산은 그래프에서 주어진 정점 집합에서 시작해서 조건에 맞는 이름과 타입에 해당하는 간선을 탐색할 수 있도록 한다.

$$\Phi[edgetype, name](vertex)$$

vertex는 탐색을 시작할 정점집합이며 edgetype은 탐색할 간선에 대한 타입으로써 E, A, R 중 하나가 된다. name은 간선이름을 나타내고 정규 경로 표현식(RPE : regular path expression)이 가능하다. 이러한 항해 연산의 결과는 간선 집합이다. 따라서 결과로 얻어진 간선으로부터 도달할 수 있는 정점을 얻기 위해 항해 연산은 자식(child)연산과 함께 쓰인다.

[예 4.1] EST 라이브러리에 있는 생명체의 이름을 검색하라.

$$\Phi[E,name](child(\Phi[E,organism](child(\Phi[E,library](Root)))))$$

예 4.1의 질의는 library.organism.name 경로를 root 정점 집합부터 시작해서 항해 연산과 child 연산을 함께 사용해 엘리먼트 간선 library, organism, name을 항해한다. 이 연산의 결과로 엘리먼트 간선{e12}에 해당되는 <name = "Human"/>을 반환한다.

XML-QL에서 선택(|)과 연속(.)같은 정규 경로 표현식을 지원하며 항해연산에서 간선의 이름에 정규경로 표현식이 쓰일 경우 중간 경로에 대한 정보는 유지되지 않는다. 따라서 중간 항해 결과를 얻기 위해서는 각 중간단계의 경로 탐색 결과를 변수에 결합시켜 유지한다.

a: = Φ[E,library](root) ; b: = Φ[E,organism](child(a)) ;
c = Φ[E,name](child(b)) ;

변수 a는 {e6}이 할당되고 child(a)는 a에 할당된 간선집합의 하위 정점 집합이 할당되므로 {V*1*3}를 갖는다. 따라서 변수 b에는 {e7}이 변수 c는 {e12}이 할당된다. 최종적으로 연산 결과 값은 변수 c에 할당된 간선이 데이터 정점을 포함하면 데이터 정점의 값을 반환하고 그렇지 않은 경우에는 {e12} 간선에 대한 정보를 반환한다. 따라서, 연산 결과로 "Human"을 반환한다.

4.2 셀렉션

셀렉션 연산자 σ는 정점 집합을 입력받아서 조건을 만족하는 정점집합을 반환한다.

$$\sigma\,[\text{condition(e)}](\text{e} : \text{expression})$$

변수 e에는 expression에 나타난 엘리먼트 정점 집합이 결합되고 결과집합에는 Condition(e)을 참으로 만족하는 정점이 추가된다. 조건절을 만족하지 않는 정점은 결과에 추가하지 않는다. 결과 값은 공집합이 될 수 있으며 조건절 안에 동등 연산자뿐만 아니라 다음과 같은 비교연산자(<>,<,>, <=,> =)가 이용된다.

[예 4.2] EST 라이브러리에 있는 생명체중 인간에 대한 모든 조직을 검색하라.

A : = Φ[E, organism](child(Φ[E, library](x : root))) ;
B : = σ[value(child(Φ[E, ~data](child(Φ[E,name](e)))))
= "Human"](e : child(A)) ;
C : = child(Φ[E,tissue](B)) ;

root정점부터 organism간선을 항해한 결과 {e7}이 변수 A에 결합된다. 변수 B에는 A 변수에 결합된 간선 집합에 대한 정점 집합 {V*I4*}에 대해 하위 간선 name의 데이터 정점 값이 "Human"인 셀렉션 조건을 만족하는 정점 집합 {V*I4*}가 결합된다. 변수 C에는 셀렉션 조건을 만족하는 B에 결합된 정점 집합의 하위 간선 tissue에 대한 정점 집합 {V*I5*}가 결합된다. 따라서 이러한 질의 결과는 정점 V*I5*가 포함하는 엘리먼트에 대한 정보를 다음과 같이 반환한다.

<tissue ID = "0001" name = "Brain"/>

4.3 구조 순환

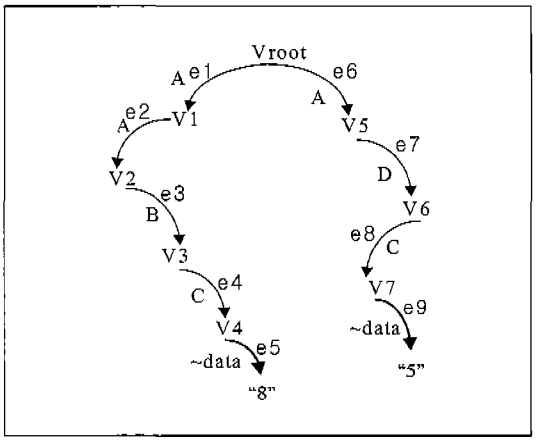
구조적 순환 연산자 *는 주어진 expression이 포함한 정점을 초기 점으로 시작해서 함수 f를 0부터 n번 반복한다. 반복할 때, 함수의 결과는 새로운 초기 값으로 이용되고, 초기 집합이 고정점에 도달할 때까지 함수는 반복해서 수행된다.

$$*\,[\text{f(x), n}](\text{x} : \text{expression})$$

*[pexpr (x)](x : source)는 pexpr (x)경로 표현식을 source에 포함된 정점집합에서 시작해서 고정점에 도달할 때까지 0부터 무한 번 반복한다. 따라서 XML-QL의 정규경로 표현식에서 쓰여지는 1또는 그 이상 반복(+)을 지원한다.

[예 4.3] root 정점에서 고정점에 도달할 때까지 A경로가 여러 번 반복되는 것을 항해하고 마지막 간선인 C에 도달하기 1레벨이전에 어떠한 경로가 존재할 수 있는 간선 C집합을 반환해라.

Φ[E,C](*[child(Φ[E, #](y),1](y : *[child(Φ[E,A](x)))](x : root))))



(그림 4.1) 구조 순환 연산 예

(그림 4.1)같은 데이터 그래프 모델에서 위와 같은 구조적 순환 연산을 수행하면 y 변수에는 A가 두번 반복되는 V2와 한번 반복되는 V5가 결합된다. 최종적인 질의 결과는 {e4, e8}이 포함하는 데이터 정점의 값 {"8","5"}를 반환한다.

위 연산은 Root/(A)+/#*1/C와 같은 XPath로 나타낼 수 있다.

4.4 조 인

조인 질의는 서로 다른 XML문서에 존재하는 정점집합에 대해 질의를 할 경우 이용되며 이러한 XML문서 사이에 조인(ⓧ)관계가 성립한다. 관계형 데이터베이스에서의 자연조인과 유사하며, 두개의 XML문서가 조인에 참여하는 이원조인을 고려한다.

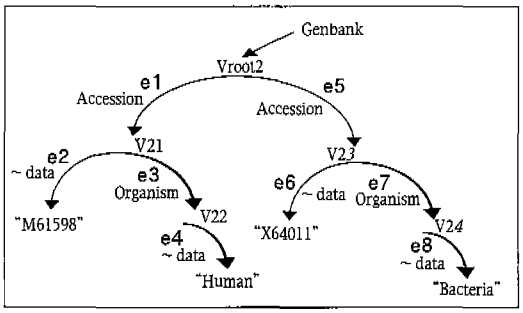
조인 연산자는 아래 식처럼 왼쪽과 오른쪽의 표현식에 언급된 두개의 정점 집합에 대해 정점의 카티션 프로덕트 집합을 만들고 조인조건을 평가한다. 두개 정점집합의 각 정점 (a, b)에 대해서 조인 조건 condition(a, b)을 만족할 때, 두개의 정점 a, b사이에 조인 정보를 포함한 가상 참조 간선이 생성된다. 이 가상 참조 간선은 XML문서 구조의 일부가 아니라 질의처리의 일부로서 추가되기 때문에 가상 참조 간선이라 부른다. 따라서 가상 참조 간선은 질의에서 유효하게 된다.

$$(\text{a} : \text{expression})\ \text{ⓧ}\ [\text{condition(a, b)}](\text{b} : \text{expression})$$

[예 4.4] (그림 3.1)의 EST에 대한 XML문서와 (그림 4.2)의 DNA에 대한 정보를 포함하는 GenBank에 대한 XML문서가 있다. GenBank의 DAN에서 만들어진 해당 EST서열정보를 검색해라.

<?XML version = "1.0" standalone = "yes" ?>

<GenBank>
  <Accession> M61598 </>
    <Organism> Human </>
  <Accession> X64011 </>
    <Organism> Bacteria </>
</ GenBak>



(그림 4.2) 조인 연산을 위한 GenBank XML문서와 그래프

(a : child(Φ [E,GenBankAcc](child(Φ [E,EST](root *1*))))
⊗ [value(child(Φ [E,~data](a))) = value(child(Φ [E,
~data](b)))] (b : *child*(Φ [E, *Accession*] (*root2*)))

두개의 XML 문서에 대해 수행되는 조인 질의는 EST에 대한 정보를 가지며 Vroot1을 루트로 가지는 XML문서1와 Vroot 2를 루트로 가지는 GenBank에 대한 XML문서2에 대해 수행된다. 두 XML 문서에서 조인조건으로 사용되는 공통된 엘리먼트는 account number이다. (그림 3.1)의 XML문서1에서 GenBankAcc 엘리먼트 정점 집합{V*12*}은 변수 a에 결합되고 (그림 4.2)의 XML문서2에서 Accession 엘리먼트를 항해한 정점 집합{V*21*, V*23*}은 변수 b에 결합된다. 이렇게 a와 b에 결합된 각각의 정점에 대해서 이 정점들이 가지는 데이터가 동일한 정점{V*12*, V*21*}이 결과로 반환된다. 조인연산을 이용하여 EST XML문서에 존재하는 Genbank DNA의 접근번호와 동일한 Genbank XML 문서의 DNA 접근번호를 찾아낼 수 있다. 조인 연산의 결과로써 각 DNA정점으로부터 만들어진 EST 정점을 연결하는 여러 개의 가상 참조 간선을 생성함으로써 두개의 문서가 연결된다.

4.5 결과 구조체

질의 결과로 반환해야 할 XML문서를 생성하기 위한 관련된 연산들이다.

4.5.1 정점생성

v [type] (value)

이 연산은 그래프 상에서 하나의 정점을 생성하고 간선 생성 연산에 대한 인자로서 이용된다. Type은 정점 타입을 나타내고 value는 정점 값을 나타낸다.

4.5.2 간선생성

edge [edgetype, name, child](parent)

edgetype은 애트리뷰트 간선 A 또는 엘리먼트 간선 E 중 하나가 될 수 있다. 간선의 name은 애트리뷰트 또는 엘리먼트의 이름 중 하나가 되고 child는 자식 정점이 되고 parent는 간선에 대한 부모 정점이다. 새롭게 추가되는 간선은 마지막부분에 추가된다.

[예 4.5] 엘리먼트 b의 값으로 "This is a test"라는 문자열을 추가해라.

edge[E, " ~data", v[string]("This is a test")](b)

4.5.3 Group By

Υ[grouping_expr(x)](x : expression)

그룹연산은 결과에서 유사한 정점과 간선 정보를 요약하거나 집계하기 위해서 이용된다. group-by 연산자는 grouping_expr값에 따라 x정점을 나누어 그룹을 형성한다. 각 그룹의 멤버는 동일한 grouping_expr값을 갖는다.

[예 4.6] 고용인 리스트에서 부서 번호에 따라 고용인들을 그룹화해라.

EmpList : = child(Φ[E, Employees](root))
EmpGroups : = [value(child(Φ[A, Dept_Number](a)))]
(a : EmpList)

4.5.4 맵

μ [f (x)](x : expression)

맵 연산자는 동일한 연산을 집합의 아이템에 적용할 때 이용한다. 즉, f (x)함수를 주어진 집합의 각 아이템에 반복해서 적용한다. *연산자와 다르게 이 연산은 수행된 연산의 결과가 반복 수행되는 다른 연산의 입력으로 포함되지 않는다.

5. XML 대수 연산자 이용 질의 변환

이 절에서는 XML 질의 언어 중 XML-QL 질의를 질의

대수로 변환하는 예를 보여준다. 선택된 질의 유형은 XML-QL[2, 7]을 참조하였으며 질의는 EST(Expressed Sequence Tag) 유전체 데이터를 대상으로 한다. XML-QL 질의는 제시된 대수를 이용하여 대수적 표현이 가능하지만 이 절에서는 셀렉션(σ), 조인(⊗)과 RPE(Regular Path Expression)질의에 한정하여 기술한다. 상세한 내용은 [24]를 참조한다.

5.1 셀렉션 질의

다음은 셀렉션 질의에 대한 변환 예로써 "Genomics" 저널에 발표된 논문의 저자와 제목을 검색한다. 질의를 XML-QL로 표현하면 아래와 같다.

```
where <publication>
          <journal>Genomics</journal>
            <title>$T</title>
            <author>$A</author>
       </publication> in "Est.xml"
construct <result>
            <title>$T</title>
            <author>$A</author>
          </result>
```

XML-QL로 표현된 위와 같은 질의를 XML 대수로 변환하면 아래와 같다.

- Publication : = child(Φ[E, publication](Root))
- Journal : = σ[value(child(Φ[E, ~data](child(Φ[E, journal](p)))))
  = "Genomics"](p : Publication)))
- T : = child(Φ[E,~data](child(Φ[E, title](j : Journal))))
- A : = child(Φ[E,~data](child(Φ[E, author](j : Journal))))
- R : = v[E, "result", v[element]()](null)
- Const : = μ[v[E, "title", parent(v[E,~data,v[string]
  (value(t : T))](v[element]()))](r:R),
  v[E, "author", parent(v[E, ~data, v[string]
  (value(a : A))] (v[element]()))](r : R)](p : Pub)

셀렉션 질의 변환에서는 항해연산을 이용해서 루트부터 "Genomics" 저널을 검색하고 셀렉션을 이용하여 저널에 발표된 논문의 제목과 저자를 결과로서 돌려준다. 따라서 Journal 변수는 Genomics 저널을 나타내는 정점 집합과 결합되고 A와 T에는 각각 저자와 제목에 대한 정점이 할당된다. 이렇게 검색된 값을 반환하기 위해 R은 result 정점을 생성하고 Const는 검색된 제목과 저자에 대한 정점들을 새로운 결과 정점의 하위 그래프로 생성한다.

5.2 조인 질의

두개의 XML 문서에 대해 수행되는 조인질의는 두개의 다른 정점 집합에 대해 조인조건을 만족하는 정점 집합을 반환한다. [예 4.4]의 조인 질의를 XML-QL 질의 언어로 나타내면 아래와 같다.

```
where <ESTDB> <EST> <GenkBankAcc> $C </></></>
          content_as $B1 in"EST.xml"
           <GenBank> <Accession> $A </></> content_as $B2
           in "Genbank.xml"
           B1 = B2
construct<result>$C</result>
```

XML-QL로 표현된 위의 질의를 XML 조인 대수 연산으로 변환하면 아래와 같다. 대수에 대한 상세한 설명은 [예 4.4]를 참조한다.

```
(a : child(Φ[E,GenBankAcc](child(Φ[E,EST](root1)))))
   ⊗ [value(child(Φ[E,~data](a))) = value(child(Φ[E, ~data](b)))]
(b : child(Φ[E, Accession](root2)))
```

5.3 RPE 질의

다음과 같은 DTD에서 엘리먼트 part는 순환적인 엘리먼트로 정의된다. 따라서 [예 5.1]과 같이 RPE를 이용한 질의를 할 수 있다.

```
<!ELEMENT part (name, brand, part*)>
<!ELEMENT name (PCDATA)>
<!ELEMENT brand (PCDATA)>
```

[예 5.1] 브랜드가 "Ford"인 모든 part의 이름을 검색하라.

```
where <part*><name> $R </name></part>
      <brand> Ford </brand> in "www.a.b.c/bib.xml"
Construct <result> $R </result>
```

XML-QL 질의에 쓰인 part*는 RPE이고 part를 포함한 모든 간선을 검색한다. <part*><name> $R </name></part> 패턴은 무한히 연속되는 패턴과 동일하며 구조적 순환을 지원하는 *대수를 이용하여 다음과 같이 변환될 수 있다.

```
S : = σ[value(child(Φ [E,~data](child(Φ [E, brand](y)))))
   = "Ford"](y : *[child(Φ [E,part](x))](x : Root))
R :  = child(Φ[E, ~data](child(Φ [E, name](s : S))))
Result : = μ[edge[E, "Result", parent(edge[E, ~data, v[string]
          (value(r)](v[element]()))](null)](r : R)
```

아래의 질의는 좀더 복잡한 RPE 연산의 예이다. 이 XML-QL질의는 순환(*), 선택(|)과 연속(.)연산자를 이용하

였고 아래와 같이 XML대수의 맵 연산자를 이용하여 변환된다.

```
where <part*>.(subpart|component.piece)> $r </>in "www.
a.b.c/part.xml"
construct <result> $r </>

P : = *[child(Φ[E, part](x))](x : Root)
R : = μ[child(Φ[E,~data](child(Φ[E, subpart](p)))), child(Φ[E, ~data]
      (child(Φ[E,piece](child(Φ[E, component](p))))))](p : P)
Result : = μ[edge[E, "Result", parent(edge[E, ~data, v[string]
(value(r))](v[element]()))](null)](r : R)
```

6. XML 대수 연산 구현 및 실험

XML 대수를 구현하기 위해서 물리적인 저장구조와 인덱스를 6.1절과 같이 구성하고 RDBMS를 물리적 저장소로 이용하여 항해, 순환연산에 대한 알고리즘을 제시한다. 셀렉션 및 조인 연산에 대한 알고리즘은 [24]을 참조한다.

6.1 저장구조 및 경로인덱스

XML문서를 RDBMS에 저장하기 위해서 그래프 모델로 표현된 XML문서를 간선을 중심으로 관계형 테이블로 매핑 한다. XML문서 정보를 저장하는 테이블은 (그림 6.1)과 같다.

```
XMLDocument(DocID, RootID, DocFileName, Location)
Element(DocID, RootID, EdgeID, ParentEID, ChildEID, EdgeName,
  LinkT, Order, Succ)
Attribute(DocID, RootID, EdgeID, ParentEID, Attribute_Name,
  <Value>, type)
Reference(DocID, RootID, EdgeID, ParentEID, refEdgeID, ChildEID)
TextData(DocID, RootID, ParentEID, Value)
PathIndex(DocID, RootID, PathID, TargetID, PathExp1, PathExpr2)
```

(그림 6.1) XML문서 저장 테이블

저장된 XML 데이터에 빠른 접근을 위해 DocID를 클러스터 인덱스로 이용하고 XML문서별로 클러스터링하기 위해 모든 테이블은 DocID 속성을 갖는다.

XMLDocument 테이블은 XML문서의 파일이름 및 문서의 루트와 파일의 위치정보로 구성된다. Element 테이블은 하나의 간선에 대한 정보를 튜플로 구성한다. 문서별 클러스터링을 위해 XMLDocument 테이블에 존재하는 DocID와 RootID의 속성을 외래 키로 사용한다. (그림 6.2)는 Element 테이블의 속성에 대한 자세한 설명이다.

Attribute테이블은 애트리뷰트 간선에 대한 정보를 저장하고 reference 테이블은 참조간선에 대한 정보를 튜플로 저장한다. 엘리먼트사이의 참조관계를 나타내는 ID, IDREF 애트리뷰트를 일반적인 애트리뷰트와 구별하기 위해 type 속성을 attribute 테이블에 저장한다. 따라서 이러한 참조 정보는 attribute 테이블에도 저장되고 reference 테이블에서 이러한 속성을 참조한다. TextData는 실제 데이터 값을 소유한 정점의 식별자와 데이터 값을 저장한다. 모든 데이터 값은 문자로 취급한다. PathIndex 테이블은 경로 인덱스에 대한 정보를 저장한다. 경로 표현식을 처리하기 위해 경로의 길이가 2인 경로 인덱스 테이블을 만든다. (그림 6.3)은 경로 인덱스 테이블의 속성을 상세히 기술한다.

- EdgeID : 간선 식별자
- ParentEID : 간선의 부모 엘리먼트 정점 식별자
- ChildEID : 간선의 자식 엘리먼트 정점 식별자
- EdgeName : 간선의 이름
- LinkType : 간선이 연결하는 자식 노드가 다른 문서에 위치하고 있는지를 나타냄
  - aggregation은 간선의 자식노드가 동일한 문서에 존재하는 경우
  - association은 자식 노드가 다른 문서에 존재하는 경우
- Order : 부모정점으로부터 나아가는 간선의 순서
- Succ : 같은 부모 정점으로부터 현재간선 다음에 연속되는 간선의 식별자

(그림 6.2) 엘리먼트 테이블의 속성

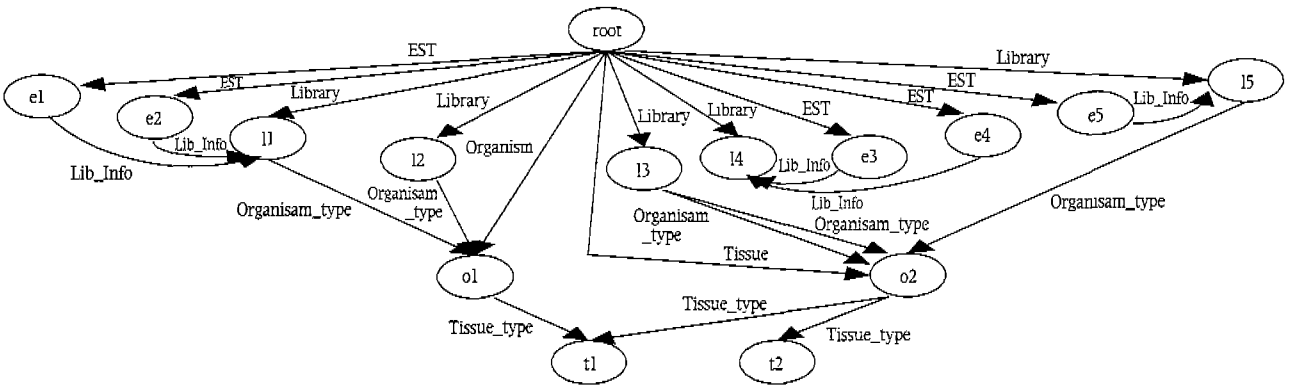
- PathID : 경로 식별자
- PathEID : 경로가 시작되는 엘리먼트 정점 식별자의 리스트
- TargetEID : 경로가 끝나는 엘리먼트 정점 식별자의 리스트
- pathexpr1 : 첫 번째 경로이름
- pathexpr2 : 두 번째 경로이름

(그림 6.3) 경로 인덱스 테이블의 속성

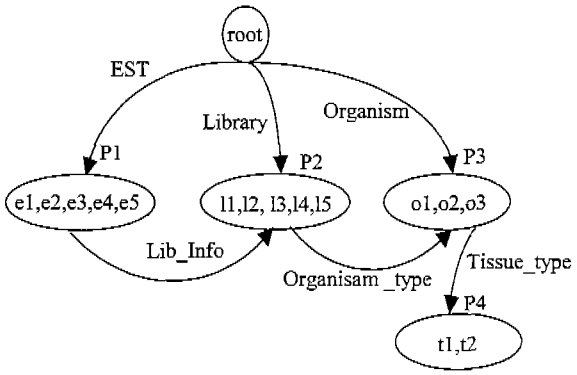
경로 인덱스그래프는 데이터그래프의 루트 노드로부터 모든 경로를 요약하는 감소된 그래프 형태를 갖는다. 인덱스 구조는 [2]에서 제시된 동치 클래스에 속하는 정점을 취하여 구성할 수 있다. (그림 6.4)의 데이터 그래프에서는 다음과 같은 동치 관계가 성립한다.

$e1 \equiv e2 \equiv e3 \equiv e4 \equiv e5$, $l1 \equiv l2 \equiv l3 \equiv l4$, $o1 \equiv o2 \equiv o3$, $t1 \equiv t2$

(그림 6.4) 데이터 그래프에 대한 인덱스 그래프 구조는 (그림 6.5)와 같다. 인덱스 그래프의 각 정점은 정점에 대한 라벨을 하나의 엔트리로 포함하는 해쉬 테이블이다. 인덱스의 각 정점은 동치클래스에 있는 데이터 정점에 대한 포인터 리스트를 갖는다. 예를 들어, P1는 [e1, e2, e3, e4, e5]리스트이다. 이러한 인덱스 그래프는 (그림 6.3)의 경로인덱스 테이블에 저장된다. 인덱스 구축시 많은 비용이 요구되므로 두 정점 x, y가 language-equivalent($x \equiv y$)는 bisimulation[2, 22] 알고리즘을 이용한다.



(그림 6.4) 데이터 그래프



(그림 6.5) 인덱스 그래프

6.2 접근 메소드

접근 메소드(access Methods)는 물리적인 저장소의 튜플을 프로그램상의 변수에 결합하기 위해 사용한다. 다음과 같은 다섯 가지의 접근 메소드를 이용한다.

- 확장 스캔(extent scan : ES(EdgeName en[ ], EdgeType et, Root r))

ES연산자는 간선이름을 입력받아서 간선으로 연결된 부모와 자식 정점을 돌려준다. 두개의 정점 x와 y 사이의 간선 en이 존재할 경우 x, y 정점을 포함한 튜플을 돌려준다. ES 메소드의 파라메터는 SQL의 조건절과 from절의 테이블 이름을 결정한다. 저장구조에서 간선 타입 별로 테이블로 매핑되므로 EdgeType에 대한 파라메터는 접근할 테이블 이름을 나타낸다.

```
select *
from $et(Element|Attribute)
where EdgeName = ($en0 OR $en1 OR…$enn) and RootID = $r
```

- 선행 스캔(forward scan : FS(Node n, EdgeName en[], EdgeType et, Root r))

FS메소드는 객체지향 데이터베이스 시스템의 포인터 따라가기(pointer chase)와 동일한 메소드이다. 선행스캔은 정점 n로부터 간선 en을 통해서 도달할 수 있는 모든 자식 정점에 대한 튜플을 반환한다. 이러한 FS메소드는 아래와 같은 SQL을 RDBMS에서 실행하고 그 결과를 반환한다.

```
select *
from $et(Element|Attribute)
where ParentEID = $n AND EdgeName = ($en0 OR $en1
      OR…Senn) AND RootID = $r
```

- 후행 스캔(backward scan : BS(Node n, EdgeName en[], EdgeType et, Root r))

BS 연산자는 역 포인터 따라가기 연산자이다. 간선 en과 정점 n를 입력받아 간선리스트 en를 통해서 도달할 수 있는 n의 모든 부모 정점을 반환하는 연산자이다.

- 데이터스캔(Data scan : DS(Operator op, Data v, Root r))

데이터스캔 함수는 데이터 값을 저장하고 있는 엘리먼트 정점을 반환하기 위한 메소드이다. textData 릴레이션에서 데이터 값이 주어진 조건에 만족하는 데이터 엘리먼트 정점을 반환한다. 데이터 정점의 값을 비교하기 위해 {=, >, < > =, <=}와 같은 비교연산자가 이용된다.

- PathIndex (Root r, PathExpression p, PathLength n)

하나 이상의 경로 이름과 길이를 인자로 받아들여 해당하는 결과를 반환하기 위한 메소드이다. PathExpression은 하나 이상의 경로를 (.)로 연결하여 나타낸다. 경로의 길이가 2이상인 경우에는 PathIndex 테이블과 Element 또는 Attribute 테이블을 조인하거나 PathIndex테이블을 셀프조인 한다.

6.3 알고리즘

6.3.1 항해 알고리즘

항해(Φ)연산은 정점 콜렉션에서 시작해서 간선의 이름과 타입을 만족하는 경로를 탐색할 수 있으며 간선의 이름에

따라 단순 경로 표현식과 RPE로 나눈다. 단일 경로를 나타내는 항해연산은 위에서 제시된 FS나 ES접근 메소드를 이용하며 질의 최적화 단계에서 최적의 접근 메소드를 선택하게 된다.

T : = child(Φ[E, tissue](child(Φ[E, library](Root)))))

항해 연산은 데이터 값에 도달하기 위해 빈번히 발생하는 연산이므로 위의 예제와 같이 연속된 항해연산은 질의 최적화 과정에서 한 번의 항해연산으로 수행된다. 이것은 FS 메소드의 EdgeName 파라메터에서 library.tissue로 경로 표현식을 실행한다.

RPE와 같은 복잡한 경로의 항해연산은 다음과 같은 단계로 나누어 처리한다. 첫째, 경로이름에 표현된 RPE 연산자를 파싱 및 식별한다. 둘째, 해당 연산자를 해결하기 위한 메소드를 호출한다. 메소드는 경로인덱스에 접근하기 위한 경로인덱스 메소드가 포함된다. 경로이름에 포함한 연산자에 대해 이러한 과정을 반복한다.

RPE에서는 선택(|), 연속(·), 반복(*), 와일드카드(#)가 사용될 수 있다. 구조적 순환(*)은 6.3.2에서 자세히 설명한다. 선택(|)연산자는 P1|P2…|Pn과 같이 다수의 경로 표현식으로 구성된다. 이것은 P1경로부터 Pn까지 각 경로에 해당되는 항해 연산 결과의 합집합으로 (|) = {Φ (P1) ∪ Φ (P2) ∪… Φ (Pn)} 같이 나타낸다. 단순 항해(Φ) 연산은 FS 메소드에 의해 실행되므로 선택(|) 연산자는 FS 메소드를 반복적으로 수행한다.

연속(.)연산자는 단일 경로가 (.)연산자로 연결된 연속 경로 표현식이다. (.)는 PathIndex()메소드를 이용하여 경로길이 3까지 한번에 항해한다. 아래는 이러한 RPE를 처리하기 위한 알고리즘이다.

```
1   Input : N : Collection of vertex
2     RPE : regular path expression(EName1, Op1, EName2,
            Op2, …, ENameN, OpN)
3     TPE : Edge type (E|A|R)
4     RPE : Root vertex
5   Output : result : Collection of Edge ( type(result) =
            {E,A,R})
6   Begin
7    Collection of Edge name S
8    Collection of vertex V1, V2, r
9     for each exp in RPE do
                    // RPE에서 RPE 연산자를 식별
10     if (exp ==“*”) then
11       SRecursion( N, Φ)
                    // 구조적 순환처리 함수 호출
12     else if ( exp ==“.”) then
                    // “.” 연산자 처리
13       S: = getEdgeName(RPE)
14       P: = PathIndex(R, S, pathLenth(S))
                    // PathIndex 메소드 이용
15       for each vertex n in N do
16         temp : = Compare(n, P)
17           r : = r + temp
18       end - for
19     end - if
20     else if (exp == “#”) then
                    // 와일드카드연산자 처리
21       r : = r + Match( N, R)
22       end - if
23     else if (op == “|”) then
                    // “|”연산자 처리
24       S1 : = getLeftEdgeName(RPE)
25       S2 : = getRightEdgeName(RPE)
26       if (Simple RPE(S1, S2)) then
                    // 단순경로에 대한 항해 연산 수행
27         V1 : = ES(S1, T, R)
28         V2 : = ES(S2, T, R)
29         V1 : = Union( V1, V2)
                    // 단순항해 연산결과의 합집합
30         for each vertex n in N do
31           for each vertex v1 in V1 do
                    // 결과 정점집합 얻기
32             r : = r + equivalence(v1, n)
33           end - for
34         end - for
35       end - if
36     end - for
37   end
```

(알고리즘 6.1) RPE를 위한 항해 알고리즘

6.3.2 구조적 순환 알고리즘

구조적 순환(*)연산자는 F(x)를 반복할 횟수 n이 결정된 경우에는 0부터 n번 반복하고 이것이 결정되지 않은 경우에는 고정점에 도달할 때까지 무한 번 반복한다. 이런 경우 IsFixPoint(F(x))는 F(x)가 고정점에 도달했는지를 검사하여 F(x)가 종료되는 시점을 결정한다. F(x)는 4절에 제시된 대수에 대한 연산이 이용된다.

```
1   Input  : X     : Collection of vertex
2            F(X) : function
3            n     : number of iteration
    Output : result : Collection of vertex
4   Begin
5            Collection of vertex r, temp
6     for each vertex x in X do
7      If (n == null) then              // 반복횟수가 정해지지
                                          않은 경우
8       While(IsFixPoint(F(X) == true) do // 고정점에 도달한때까지
                                          F (x)를 반복
9               temp : = F(X)
10              r : = r + TEMP
11              X : = CHILD(X)
12     end - while
13    end - if
14    else
15      for each 0 to n do             // 반복횟수가 정해진 경우
16        temp : = F(X)
17          r : = r + temp
18      end - for
19    end - if
20    result : = result + r
21  end - for
22  End - structural recursion
```

[알고리즘 6.2 구조적 순환 알고리즘]

6.4 구현 및 실험

구현된 XML 대수에 대한 성능평가 실험은 Sun Ultra-Sparc 기종에서 Solaris 2.5 운영체제를 사용하였다. 데이터베이스로는 Oracle 7.3.4를 이용하고 XML 대수는 Java언어를 사용하여 구현하였다.

실험에 쓰인 데이터집합은 200,400,840개 EST 서열을 XML로 변환하여 이용하였다. EST 유전체 데이터 그래프는 (그림 6.6)과 같고 실험 데이터 집합은 <표 6.1>처럼 구성된다.

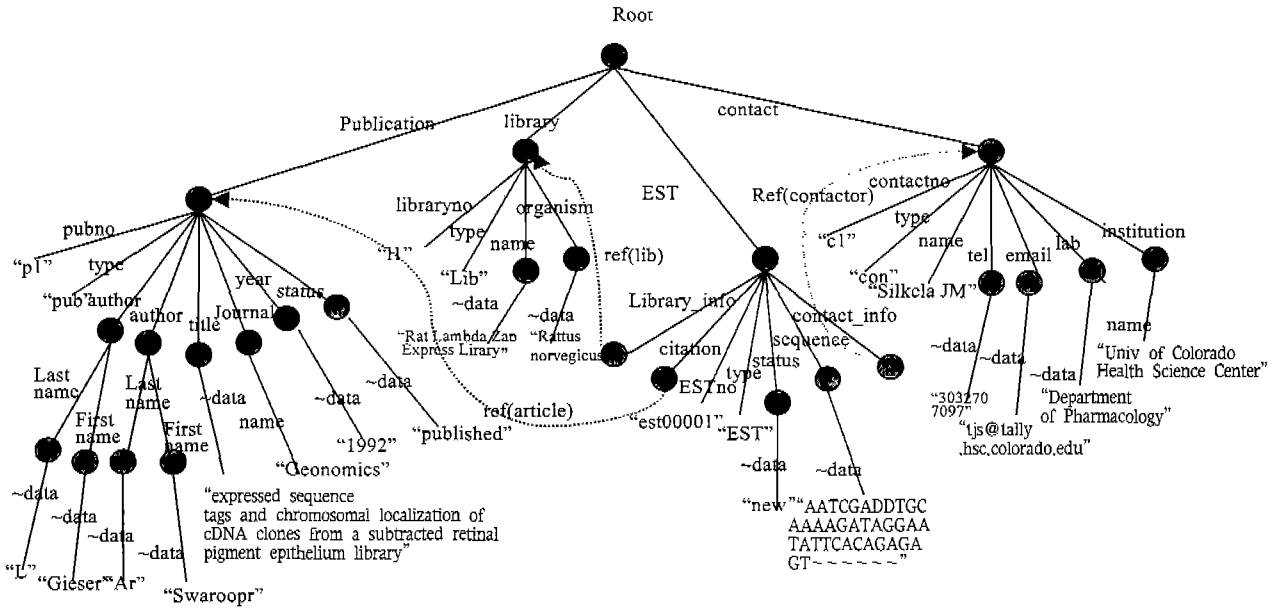
실험은 다른 연산의 단위 연산이 되는 항해 연산을 단순 항해와 RPE연산으로 나누고 각 연산의 실행시간을 측정하였다.

〈표 6.1〉 실험 데이터 집합

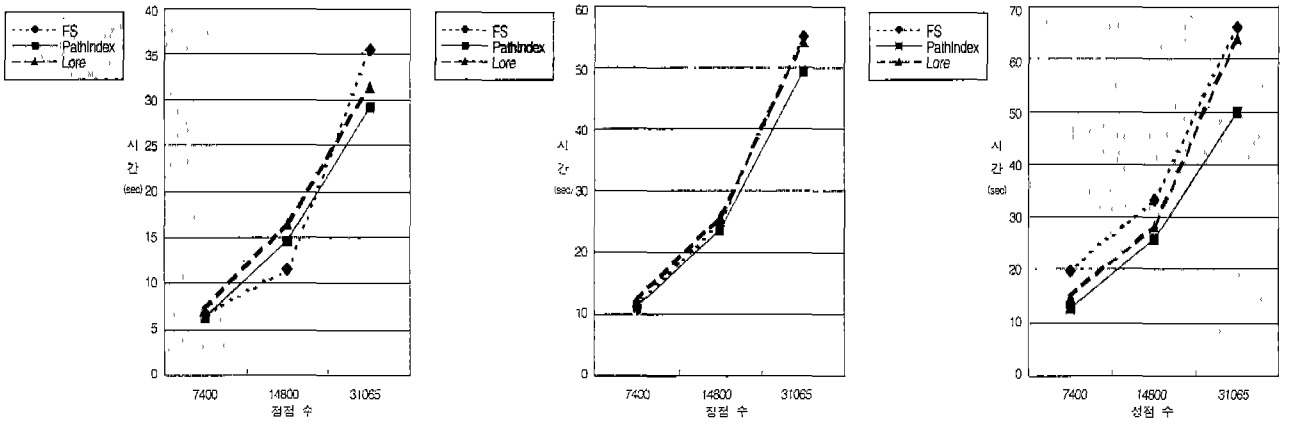
데이터 집합	엘리먼트 정점	데이터 정점	엘리먼트 간선	애트리뷰트 간선	참조 간선	총간선	총정점
DS1	4,600	2,800	7,200	2,600	800	10,600	7,400
DS2	9,200	5,600	14,400	5,200	1,600	21,200	14,800
DS3	19,320	11,760	30,240	10,920	3,360	44,520	31,065

[실험 1] "EST서열을 인용한 모든 출판물을 검색하라"

[실험 1] 질의는 child(Φ[E,publication](Root))같은 항해 연산자로 변환되고 FS와 BS 메소드에 의해서 실행될 수 있다. 엘리먼트, 애트리뷰트, 데이터 및 참조 정점에 대한 항해 시간을 서로 다른 접근 메소드를 이용하여 <표 6.2>의 측정결과를 얻었다.



(그림 6.6) EST 유전체 데이터 그래프



(i) 경로 길이가 2인 경우 (ii) 경로 길이가 3인 경우 (iii) 경로 길이가 4인 경우

(그림 6.7) 정규 경로 표현식 성능 평가

단순 항해 연산에서는 두개의 접근 메소드 FS와 BS사이의 실행시간에 대한 차이가 많이 나지 않는다. 이것은 FS와 BS 모두 동일한 테이블을 접근하기 때문이다. 또한 엘리먼트, 애트리뷰트와 참조 엘리먼트에 대한 항해 연산은 각각의 테이블을 접근하기 때문에 실행시간에 대한 차이가 많이 나지 않는다.

〈표 6.2〉 단순 항해 실행시간측정

(단위 : millisecond)

메소드 \ 정점	엘리먼트 정점	애트리뷰트 정점	참조 정점
FS	98,800	85,600	85,600
BS	94,000	88,000	90,000

[실험 2] 경로길이가 2이상인 RPE 질의에 대한 실험이다. (그림 6.7)은 경로길이가 2부터 4까지 증가할 때 PathIndex와 FS 두 가지 메소드의 RPE 처리 실행시간을 비교하고 DataGuide[12]를 이용하여 RPE 질의를 실행하는 LORE시스템과 성능비교를 나타낸다. 실험에 사용된 RPE는 경로 길이가 2인 경우는 publication.author를, 경로길이가 3인 경우는 와일드 카드를 이용한 publication.#.author를, 경로길이가 4인 경우는 publication.author.#.lastname에 대해서 실험하였다.

성능 평가에 따르면 경로 길이가 2이상인 RPE 질의는 FS 메소드보다 패스인덱스를 접근 메소드로 이용하는 것이 효율적이다. LORE 시스템은 경로 인덱스를 최대 경계 스키마 기법을 구현한 DataGuide를 이용하기 때문에 경로 인덱스에 중복되는 정점이 존재한다. 따라서 데이터가 많아질 수록 경로 인덱스의 검색 효율이 낮아진다. 반면 이 논문에서는 Suciu[2]가 제안한 그래프 시뮬레이션 이론을 적용한 최소경계 기법을 이용하였기 때문에 데이터의 양이 증가할수록 RPE 질의 처리에서 LORE 시스템보다 효율적으로 처리한다.

관계형 데이터베이스를 XML 저장 시스템으로 이용할 경우 경로 인덱스를 이용하여 n개의 정규 경로 표현식의 효율적 처리가 가능함을 실험을 통해서 알 수 있다. 관계형 데이터베이스를 XML 저장시스템으로 사용할 경우 경로가 2이상인 경우 경로 인덱스를 사용하지 않으면 엘리먼트 테이블을 셀프 조인해야하므로 처리비용이 증가한다. 이러한 실험 결과는 동일한 대수 연산일지라도 질의 실행계획에서 이용되는 접근 메소드와 구현 알고리즘에 따라 실행시간의 효율이 다름을 알 수 있다.

7. 결 론

웹 환경에서 XML의 사용증가에 따라 XML에 대한 정형화된 모델과 정규 경로 표현식 질의와 같은 복잡한 질의처리를 위한 XML 대수연산에 대한 연구가 진행되고 있다. 따라서 이 논문에서는 XML 질의를 위한 데이터모델을 설계하였고 방향그래프 기반 XML 대수 연산을 정의하였다. 그리고 이 XML 연산자를 이용하여 기존의 관계형 데이터베이스 기반 구현기법을 제시하였다.

XML은 반 구조 데이터와 다르게 엘리먼트 태그와 애트리뷰트로 구성되고 엘리먼트 사이의 참조관계 및 엘리먼트 간의 순서를 가진다. 따라서 제안한 XML 데이터 모델에서는 XML 문서를 이루는 이러한 구성요소를 빠짐없이 그래프의 정점으로 매핑하고 이들의 관계를 세 종류의 간선을 이용하여 나타내었다. XML 그래프 모델에서 정점은 엘리먼트와 데이터 정점으로 구성되고 간선은 엘리먼트 포함간선, 애트리뷰트 간선과 참조간선으로 이루어지며 이러한 정점과 간선은 기본속성과 유도된 속성을 갖도록 설계하였다.

XML 대수 연산은 정점 및 간선을 검색할 수 있는 단순 항해, 셀렉션, 조인, 정규경로 표현식을 처리할 수 있는 구조적 순환과 질의 결과를 XML로 반환하기 위한 결과 구조체 연산으로 구성된다. 질의 변환을 통해서 대부분의 XML-QL질의를 제안한 대수 연산으로 변환이 가능함을 보였다. 제안한 연산의 구현은 XML에 대한 물리적 저장소로 RDBMS를 기반 하였으며 다섯 가지 접근 메소드와 경로 표현 처리를 위한 경로 인덱스를 이용하였다. 이러한 RPE 질의에 대한 성능을 EST유전체 데이터를 대상으로 실험하여 그 효율성을 보였다.

이 논문에서는 미디에이터 시스템에서 처리할 수 없었던 RPE질의를 RDBMS를 XML 저장소로 이용하여 효율적으로 처리할 수 있음을 보여준다. 이렇게 함으로써 대부분의 기존 응용 프로그램에서 이용하는 RDBMS를 XML 저장소로 이용할 수 있으며 기존 RDBMS에서는 지원하지 않았던 정규경로 표현식 질의를 처리할 수 있다. 그러므로 RDBMS 기반 기존 응용프로그램에서 XML을 손쉽게 이용하고 기존 데이터와 함께 저장 및 질의를 할 수 있다.

이 연구는 XML에 대한 논리적 질의 대수를 기존의 데이터베이스 관리 시스템을 이용하여 구현함으로써 다양하고 복잡한 연산을 포함한 XML질의를 효율적으로 처리할 수 있음을 보여준다. 그러나 XML 대수의 구조적 순환 알고리즘 구현시 고정점을 찾기 위한 규칙이 필요하므로 이 규칙 정의가 추가로 수행되어야 한다.

참 고 문 헌

[1] Serge Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte,J. Simeon, "Querying Documents in object database," In Journal on Digital Libraries, 1996.

[2] Serge Abiteboul, Peter Buneman, Dan Suciu, "Data On the Web : From Relation to semistructured Data and XML," Morgan Kaufmann, 2000.

[3] M. J. Carey, D. Florescu, Z. G. Ives, Y. Lu, J. Shanmugasundaram, E. J. Shekita, S. N. Subramanian. "XPERANTO : Publishing Object-Relational Data as XML," In informal proc. of WebDB, pp.105-110, 2000.
[4] V. Christophidex, S. Cluet, G. Moerkotte, "Evaluating queries with generalized path expression," In proc, of SIGMOD. Int. Conf. on Management of Data, 1996.
[5] S.Clue, G. Moerkotte, "Nested queries in on object bases," In Proc. of Int. Workshop on Database Programming Languages, 1993.
[6] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu, "XML-QL : A Query Language for XML," 1998. Http : // www.w3c.org/TR/NOTE-xml-ql/.
[7] Alin Deutsch, M. Fernandez, D. Suciu, "Storing Semi-structured Data with STORED," Proc. of the ACM SIGMOD conf., 1999.
[8] M. Fernandez, D. Florescu, J. Kang, A. Y. Levy, D. Suciu, "STRUDEL : A Web-site Management System," SIGMOD Conf., 1997.
[9] M. Fernandez, J, Simeon, D. Suciu, P. Wadler, "Data Model and Algebra for XML Query," Technical Report, Unpublished Manuscript, 2000.
[10] M. Fernandez, Wang-Chiew Tan, D. Suciu, "SilkRoute : Trading between Relations and XML." In proc of Www9, 2000.
[11] Daniela Florescu, Donald Kossmann. "Storing and querying XML Data using an RDBMS," Bulletin of the Technical Committee on Data Engineering, Vol.22. No.3, 1999.
[12] R. Goldman and J. Widom, "Approximate DataGuides," In proc. of the Workshop on Query Processing for Semi-structured Data and Non-Standard Data Formats. 1999.
[13] Roy Goldman, Jason McHugh, Jennifer Widom, "From Semistructured Data to XML : Migrating the Lore Data Model and Query Language," WebDB, 1999.
[14] T. Lahiri, S. Abitcboul, J. Widom. "Ozone : Integrating Structured and Semistructued Data," In proc. of the 7th Int. Workshop on Database Programming Languages, 1999.
[15] G. Mecca, P.Merialdo, P.Atzeni, and V. Crescenzi, "The ARANEUS Guide to Web-Site Development," In WebDB, 1999.
[16] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, Jennifer Widom, "Lore : A Database Management System for Semistructured Data," SIGMOD Record 26(3) : 54-66, 1997.
[17] J. McHugh, J. Widom, "Query Optimization for XML," In Proc. of VLDB, pp.315-326, 1999.
[18] Y. Papakonstantinou, H. Garcia-Molina, J. Widom, "Object exchange across heterogeneous information sources," In Proc. of Int. Conf. on Data Enginnering, pp.251d-260, 1995.
[19] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, "Relational Databases for Querying XML Documents : Limitations and Opportunities," In proc. of VLDB. pp.302-314, 1999.
[20] W3C, "ExtensREC-xml-19980210," 1998.
[21] The World Wide Web Consortium (W3C), "XSLT (XSL Transformations)," November 16, 1999.
[22] 박경현, 김록원, 양은주, 최은선, 류근호, 반구조적 데이터의 효율적인 최소경계 스키마 추출 기법, 한국정보과학회 학술발표논문집, Vol.27, No.2, 2000.
[23] 박성희,박경현, 김록원, 남광우, 류근호, ORDBMS를 이용한 XML 문서의 저장 및 질의, 한국정보과학회 학술발표논문집, Vol.27, No.1, 2000.
[24] 박성희, 방향그레프기반 XML 데이터 모델, 충북대학교 전자계산학과 석사학위논문, 2001.
[25] 박성희, 김대중, 류근호, XML질의를 위한 정규 경로 표현 구현 기법, 한국정보과학회 학술발표논문집, Vol.28, No.1, 2001.

박 성 희

email : shpark@dblab.chungbuk.ac.kr
1996년 충북대학교 도시공학과 졸업(공학사)
1998년 한국전자통신 연구원 컴퓨터소프트웨어 연구소 위촉 연구원
2001년 충북대학교 대학원 전자계산학과 석사(이학석사)
2001년~현재 충북대학교 대학원 전자계산학과 박사과정
관심분야 : 반구조 및 XML 데이터베이스, 시공간데이터베이스, Bioinformatics 등

최 은 선

email : eschoi@dblab.chungbuk.ac.kr
1993년 충북대학교 전자계산학과 졸업(이학사)
1999년 충북대학교 교육대학원 전자계산교육학과 석사(교육학석사)
2000년~현재 충북대학교 대학원 전자계산학과 박사과정
관심분야 : XML 데이터베이스, 시공간데이터베이스, Bioinformatics 등

류 근 호

email : khryu@dblab.chungbuk.ac.kr
1976년 숭실대학교 전산학과 졸업(이학사)
1980년 연세대학교 산업대학원 전산전공(공학석사)
1988년 연세대학교 대학원 전산전공(공학박사)
1976년~1980년 육군군수 지원사전산실(ROTC장교)
1980년~1983년 한국전자통신연구소 연구원
1983년~1986년 한국 방송대학교 전산학과 조교수
1989년~1991년 University of Arizona, Research Staff(TempIS 연구원, Temporal DB)
1986년~현재 충북대학교 컴퓨터과학과 교수
관심분야 : 시간데이터베이스, 시공간 데이터베이스, Temporal GIS, 지식기반 정보검색, 데이터베이스 보안, 데이터 마이닝, Bioinformatics 등