엘리먼트의 중첩 문제를 해결한 Shared Inlining 저장 기법

(A Shared Inlining Method for Resolving the Overlapping Problem of Elements)

홍 은 지 † 이 영 호 ††
(Eunii Hong) (Young Ho Lee)

요 약 웹 환경에서 정보 표현과 교환을 위한 표준 방식으로 널리 사용되고 있는 XML 문서의 양은 급속히 증가하고 있으며, 대용량의 XML 문서도 많이 생산된다. 이러한 XML 문서들을 RDBMS에 효율적으로 저장하고 검색하는 연구가 활발히 진행되어 왔는데 이들 연구들 중에서 Shared Inlining 저장 방법이 높은 검색 효율을 보인다. Shared Inlining 방법은 DTD의 정보를 분석하여 XML 문서를 노드의 성분별로 분할하여 관계형 데이타베이스로 저장하는 기법이다. 본 논문은 기존의 Shared Inlining 방법에서 여러 하위 노드를 가지는 엘리먼트에서 발생되는 중첩 문제를 해결하기 위한 기법을 제안한다. 이 방법에서는 DTD 정의에 맞게 XML 문서를 Shared Inlining 구조로 저장하고 검색의 정확성을 높인다.

키워드 : XML, Shared Inlining, 중첩, 관계형 스키마, XML 스키마, 엘리먼트

Abstract The number of XML documents, which are widely used as a standard method for information expression and exchange in the web-based environment, increases rapidly along with the growing production of large XML documents. Many studies have been made to store and retrieve these XML documents on RDBMS, among which Shared Inlining storage method has a higher level of retrieval efficiency. The Shared Inlining method is the technique that analyzes the DTD information and stores the XML document in RDBMS by dividing for each node component. This study proposes the technique to resolve the overlapping problem that occurs in the element with several child nodes in the existing Shared Inlining method. The suggested method stores the XML document in the Shared Inlining structures appropriate to the DTD definition and enhances the accuracy of retrieval.

Key words : XML, Shared Inlining, overlapping, relational schema, XML schema, element

1. 서 론

웹 환경에서 정보 표현과 교환을 위한 표준 방식으로 널리 사용되고 있는 XML 문서의 양은 급속히 증가하고 있으며, 대용량의 XML 문서도 많이 생산된다. 이러한 XML 문서들을 효율적으로 저장하고 검색하는 연구가 활발히 진행되어 왔다. XML 문서를 효율적으로 관리하기 위해서는 XML과 데이타베이스 기술의 통합이 필요하며, 가장 많이 사용되고 안정적인 RDBMS(Relational Database Management System)에 XML 데이타를 저장하기 위한 다양한 방법들[1-8]이 연구 발표되었다. XML 문서를 검색하기 위한 방법으로는 XPath [9], XQuery[10]와 같은 질의 언어가 사용되는 반면에, RDBMS에서는 데이타 검색을 위해 SQL이 사용된다. 따라서 관계형 데이타베이스 모델로 저장된 XML 데이타를 검색하기 위해서는 XPath, XQuery를 SQL로 중간번역 하는 단계가 필요하다.

RDBMS에 저장된 XML 문서에 대한 질의 성능에

· 본 연구는 국토해양부 첨단도시기술개발사업 - 지능형국토정보기술혁신사업과제의 연구비지원(07국토정보C03)에 의해 수행되었습니다.

† 종신회원 : 성공회대학교 소프트웨어공학과 교수
hong@mail.skhu.ac.kr
†† 정 회 원 : (사)iCOOP생협연대 전산팀 주임
e0ho@naver.com

논문접수 : 2007년 12월 10일
심사완료 : 2008년 7월 8일

정보과학회논문지 : 데이타베이스 제35권 제5호(2008.10)

영향을 주는 주요 요인 중 하나는 XML 데이타를 저장하는 방식이다. 본 논문에서는 XML을 저장하기 위한 방식인 DTD 의존적인 방법(DTD dependent approach) 중 가장 효율적인 방식으로 밝혀진 Shared Inlining[8] 방식에 따라 관계형 스키마를 생성한다. 이때 #PCDATA를 가지는 엘리먼트가 자식 엘리먼트를 가질 경우 부모 엘리먼트로 정확히 연결되지 못하는 자식 엘리먼트가 발생하는 문제를 발견하였다. 본 논문은 Shared Inlining 저장 시 발생하는 이러한 문제점을 보완하여 저장과 검색의 정확성을 높인다.

본 논문은 다음과 같이 구성된다. 2장은 관련연구로 RDBMS을 기반으로 XML 문서를 저장하기 위한 접근 방법에 대해 알아본다. 3장에서는 DTD를 이용하여 관계형 스키마를 생성하는 기존의 Shared Inlining 방법을 설명하고 발생할 수 있는 문제점을 확인한다. 4장에서는 중첩 구조를 해결하기 위한 새로운 관계형 스키마의 생성 방법을 제시한다. 5장은 결론을 논한다.

2. 관련 연구

본 장에서는 XML 문서를 RDBMS에 저장하기 위한 기존연구에 대해 알아본다. RDBMS를 기반으로 XML 문서를 저장하는 방식은 DTD 독립적(DTD independent approach)[3] 방법과 DTD 의존적(DTD dependent approach)[8] 방법이 있다.

DTD 독립적 방법은 DTD 정보 없이 XML 데이타로부터 관계형 스키마를 생성하는 방법으로 Edge, Monet, XParent, XRel[4]이 있다.

DTD 의존적 방법은 DTD 기반으로 생성된 관계형 데이타베이스의 스키마에 따라 XML 문서를 저장한다. 이러한 방법은 특정 DTD들을 만족하는 XML 문서들이 많이 존재하거나 스키마 생성에 사용된 DTD의 수정이 발생하지 않는 경우에 적합하다. 인라인(Inlining) 기술은 DTD를 관계형 스키마(schema)로 매핑(mapping) 시키는 기술로서 가능한 많은 엘리먼트(element)의 자손을 하나의 관계(relation)로 불여서 표현하는 것이다.

Basic Inlining에서 모든 엘리먼트는 자신에게 속한 모든 자손(descendant) 엘리먼트들을 인라인한다. 그리고 모든 엘리먼트들은 개별의 릴레이션으로 생성하며 어떠한 엘리먼트도 루트(root)가 될 수 있다.

Shared Inlining은, 그래프에서 2 이상의 진입차수(in-degree)를 가지는 노드(node)와 *에지로 연결이 되는 노드에 대한 릴레이션을 각각 생성한다. 그리고 진입차수가 1인 노드를 자신의 부모(parent) 노드에 인라인시킴으로써, 진입차수가 1인 노드에 대한 릴레이션을 따로 생성하지 않는다. Shared Inlining 기술은 한 엘리먼트 노드를 정확하게 한 릴레이션에만 표현한다. 만약 동일한 엘리먼트가 다른 릴레이션에 표현된다면, 해당 엘리먼트를 별도의 릴레이션으로 생성한다.

Hybrid Inlining은 Basic Inlining과 Shared Inlining을 결합한 방식으로 Shard Inlining에 진입차수가 2 이상인 노드가 *에지 없이 도달 가능하면 해당 노드를 인라인시킨다.

Basic Inlining은 질의 처리 시 특정 엘리먼트 노드에서 시작하여 조인(join)의 수를 줄이고 단편화를 해결하지만 많은 노드들이 중복 표현되는 단점이 있다. Hybrid Inlining은 조인의 수를 줄이지만 DTD의 구조에 따라서는 Shared보다 항상 좋은 성능을 나타내는 것은 아니었다. [8]의 결과에서는, Shared Inlining과 Hybrid Inlining이 비슷한 질의 처리 능력을 보여주고 있는데, Shared Inlining 방식이 노드들을 중복표현 하지 않기 때문에 저장 공간의 효율이 높다. 본 논문에서는 DTD 의존적인 방법 중에서 효율이 좋은 Shared Inlining 방식을 연구의 대상 모델로 한다.

3. DTD를 이용한 관계형 스키마 생성 방법과 문제점

본 장에서는 DTD를 기반으로 관계형 스키마를 생성하는 기존의 Shared Inlining 방법을 자세하게 소개하고 엘리먼트의 중첩 구조에서 발생할 수 있는 문제점을 논한다. DTD를 기반으로 관계형 스키마를 생성하는 순서는 복잡한 DTD 표현의 단순화, DTD 그래프 생성, 엘리먼트의 인라인 과정을 거치고 마지막으로 인라인된 노드들을 관계형 스키마로 생성한다.

3.1 DTD 단순화

일반적인 DTD 정의는 복잡해질 수 있으나 이를 저장하는 관계형 스키마가 복잡한 DTD구조를 그대로 표현한다는 것은 비효율적이다. 또한 생성된 관계형 스키마로부터 DTD를 생성할 필요는 없으므로, 관계형 스키마에 적합하도록 DTD를 단순화(simplify)한다. 복잡하게 정의된 표현식은 그림 1의 규칙으로 단순화한다. 그림 1과 같은 규칙이 적용된 DTD는 +, ?, |과 같은 연산자를 포함하지 않는다.

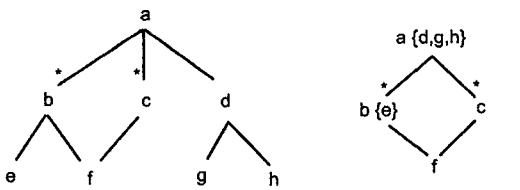
3.2 DTD 그래프 생성 및 인라인

DTD의 단순화 과정 이후에는 단순화된 DTD 스키마를 이용하여 DTD 그래프를 생성한다. 그리고 가능한 많은 자손 엘리먼트들을 다른 엘리먼트로 인라인시킨다. 엘리먼트 a와 b가 간선으로 연결되고 a가 b의 부모 노드일 때, 인라인 규칙은 다음과 같다.

- b가 a 이외의 아닌 다른 진입차수를 가지지 않는다면 b는 a로 인라인된다.
- b가 다른 진입차수를 가진다면 b는 공유 노드(shared node)로서 여러 부모 노드를 가진다.

1. $e^{+} \rightarrow e^{*}$
2. $e? \rightarrow e$
3. $(e_1|...|e_n) \rightarrow (e_1, ..., e_n)$
4. $(a)(e_1, ..., e_n)^{*} \rightarrow (e_1^{*}, ..., e_n^{*})$
   $(b)e^{**} \rightarrow e^{*}$
5. $(a)..., e, ..., e, ... \rightarrow ..., e^{*}, ...$
   $(b)..., e, ..., e^{*}, ... \rightarrow ..., e^{*}, ...$
   $(c)..., e^{*}, ..., e, ... \rightarrow ..., e^{*}, ...$
   $(d)..., e^{*}, ..., e^{*}, ... \rightarrow ..., e^{*}, ...$

그림 1 DTD 단순화 규칙



(a) DTD 그래프 (b) shared Inlining 적용

그림 2 Shared Inlining 적용 예

Schema Shared Inlining
a (aID, ParentID, a, a.d, a.d.g, a.d.h)
b (bID, ParentID, b, b.e)
c (cID, ParentID, c)
f (fID, ParentID, f)

그림 3 Shared Inlining 스키마의 예

• b가 *에지로 연결된다면, a는 자식 노드(child node) b를 여러 개 가진다. b는 a로 인라인될 수 없으므로, 공유 노드가 된다.

다음의 그림 2, 그림 3은 인라인 방식의 예를 보여준다. 그림 2의 (a)는 인라인이 적용되기 전의 DTD 그래프이고 (b)는 인라인을 적용한 그래프의 모양이다. a 노드는 최상위 엘리먼트로 b, c, d 엘리먼트를 자식 노드로 가진다. b, c 노드는 a 노드에서 *에지로 연결된 자식노드로서 여러 개의 b, c 노드가 존재할 수 있다. f 노드는 진입차수가 2인 노드로 b, c 노드를 부모 노드로 가진다.

(a)에서 (b)로 인라인이 적용되는 과정은 다음과 같다. 진입차수가 1인 g, h, e는 자신의 부모 노드로 인라인되고, d 노드 역시 진입차수가 1이므로 자신에게 인라인된 노드들과 함께 부모 노드인 a 노드로 인라인된다. 인라인 과정이 끝나면 (b)와 같이 그래프는 공유 노드인 a, b, c, f가 남게 되어 그림 3과 같은 관계형 데이타베이스 릴레이션으로 생성된다.

3.3 실험

실험에 사용된 XML 파일은 XMark의 xmlgen[2,11] 프로그램을 이용하였다. XMark는 가장 대표적인 XML 벤치마크(benchmark)로서, 기존의 성능 평가 방법들의 문제점을 보완하고, XML의 반구조화 질의들의 유연성과 표현력을 제공할 수 있는 데이타 모델을 사용한다. 인자 값을 1로 입력하여 115,775KB 크기의 XML 문서를 생성하였으며, 문서의 텍스트 데이타는 자연어를 사용하기 위하여 셰익스피어 희곡의 문자들과 인터넷의 다양한 자료들을 이용하여 생성한다. 생성된 XML 파일은 인터넷 옥션 사이트를 적용한 모델로서 DTD 그래프는 그림 4와 같은 구조를 가진다.

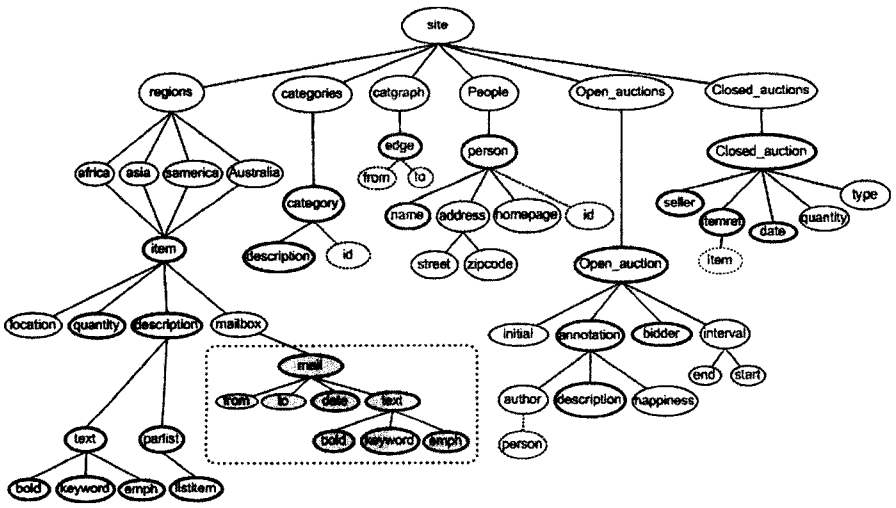


그림 4 auction.xml의 DTD 그래프

표 1 Shared Inlining 스키마 생성

테이블	레코드	테이블	레코드
amount	0	annotation	21,750
bidder	59,486	bold	71,804
category	1,000	closed_auction	9,750
date	90,182	description	44,500
edge	1,000	emph	69,536
incategory	82,151	interest	37,689
item	21,750	itemref	21,750
keyword	69,911	listitem	60,481
mail	20,946	name	48,250
open_auction	12,000	parlist	20,813
person	25,500	quantity	43,500
seller	21,750	site	1
status	0	text	267,690
type	21,750	watch	50,269

표 1은 auction.xml 파일을 Shared Inlining 방법으로 생성한 각 테이블과 저장된 레코드 수를 나타내며, 그림 5는 생성된 테이블 중에서 site, edge, item, status, text, type 테이블의 구조를 나타낸다.

3.4 중첩 구조의 문제점

그림 4와 같은 DTD 그래프로 표현된 XML 파일을 관계형 스키마로 생성하는 과정 중, 그래프의 일부분에서 모호한 Shared Inlining 그래프가 발견되었다. 그림 4의 전체 그래프 중에서 모호한 Shared Inlining 그래프를 생성하는 부분은 점선박스로 나타낸 mail 엘리먼트의 하위 엘리먼트들이다.

그림 6은 mail 엘리먼트와 그 하위 엘리먼트들의 DTD 정의다. mail, text, bold, keyword, emph 엘리먼트는 * 에지로 연결이 되어 별도의 테이블로 구성될 수 있는 공유 노드이다. date 엘리먼트는 그림 4의 그래프에서처럼 mail 엘리먼트와 closed_auction 엘리먼트의 자식 엘리먼트가 될 수 있기 때문에, 진입차수가 2 이상인 공유 노드가 되어 별도의 테이블로 구성된다. text 엘리먼트 또

```
site(SiteId, ParentId, Site.categories, Site,
     Site.regions, Site.regions.africa, Site.regions.asia,
     Site.regions.europe, Site.regions.namerica,
     Site.regions.samerica, Site.regions.australia,
     Site.catgraph, Site.people, Site.open_auctions,
     Site.closed_auctions )
                    ....생략...
edge(edgeid, parentid, edge, edge.to, edge.from)
item(itemid, parentid, item, item.mailbox,
     item.shipping, item.payment, item.location,
     item.featured, item.id)
status( statusID, parentID, status )
text (textID, parentID, text)
type (typeID, parentID, type)
```

그림 5 생성된 스키마 구조

```
<!ELEMENT mail (from, to, date, text)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT text (#PCDATA | bold | keyword | emph)*>
<!ELEMENT bold (#PCDATA | bold | keyword | emph)*>
<!ELEMENT keyword (#PCDATA | bold | keyword | emph)*>
<!ELEMENT emph (#PCDATA | bold | keyword | emph)*>
```

그림 6 DTD 정의의 일부

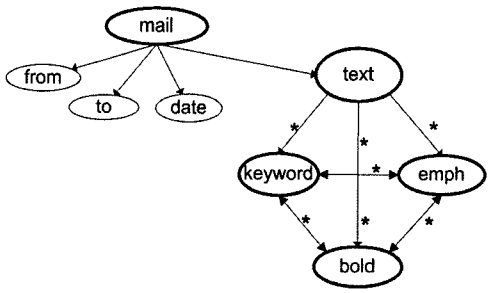


그림 7 Shared Inlining DTD 그래프의 일부

```
<mail>
 <from>Libero Rive mailto:Rive@hitachi.com</from>
 <to>Benedikte Glew mailto:Glew@sds.no</to>
 <date>07/05/2000</date>
 <text>
  virgin preventions half logotype weapons granted factious
  already carved fretted impress pestilent
  <keyword>girdles deserts flood george nobility
  reprieve</keyword>
  discomfort sinful conceiv corn preventions greatly suit
  observe sinews enforcement
  <emph>armed</emph>
  gold gazing set almost catesby turned servilius cook
  doublet preventions shrunk smooth great choice enemy
  disguis tender might deceit ros dreadful stabbing fold
  unjustly ruffian life hamlet containing leaves
 </text>
</mail>
```

그림 8 중첩 #PCDATA의 예

한 description, mail 엘리먼트의 하위 엘리먼트이므로 진입차수가 2 이상이 되어 공유 노드가 된다.

그림 7은 그림 6의 DTD 정의 중 mail 엘리먼트와 그 하위 엘리먼트들을 Shared Inlining DTD 그래프로 표현한 형태이다. keyword, bold, emph 엘리먼트는 서로를 재귀적인 형태로 중첩할 수 있고 #PCDATA 타입의 값을 가질 수도 있다.

그림 8은 auction.xml 파일의 일부로서 그림 6의 DTD 정의를 따르는 XML 문서이다. text 엘리먼트는 #PCDATA로 되어 있고 중간 중간에 keyword, emph 엘리먼트가 삽입되어 text 엘리먼트의 #PCDATA가 3개로

분리되어 있다.

기존에 제시된 방법으로 스키마를 생성하는 방법은 두 가지가 가능하다. 첫 번째 방법은 3개로 분리된 text 엘리먼트의 #PCDATA를 각각 3개의 노드로 구성하는 방법이고, 두 번째 방법은 3개로 분리된 #PCDATA를 하나의 text 엘리먼트로 표현하는 방법이다.

3.4.1 첫 번째 방법

기존에 제시된 방법으로 Shared Inlining 스키마를 생성했을 경우에 그림 6의 text, keyword, emph, bold 엘리먼트 정의처럼 *에지로 연결되는 #PCDATA 노드는 동일한 parentID 값을 가져야 한다.

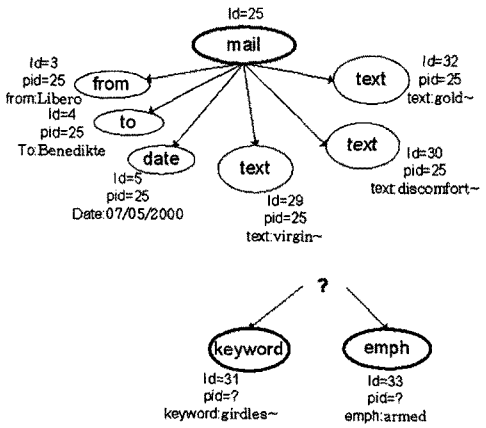


그림 9 모호한 그래프1

그림 9는 그림 8의 XML 문서 구조를 DTD 그래프로 표현한 것이다. text 엘리먼트의 #PCDATA는 중간에 삽입된 keyword, emph 엘리먼트에 의해서 3개로 나뉜다. 그림 8의 XML 문서 구조에 따라 text 엘리먼트는 3개로 분리되어 mail 엘리먼트를 부모 노드로 정확하게 연결된다. 하지만 mail 엘리먼트는 자식 노드로 하나가 아닌 여러 개의 text 엘리먼트를 가지게 되므로, 그림 6의 DTD 정의에 부합하지 않는다. 또한 id가 31과 33인 keyword나 emph와 같은 text 엘리먼트의 하위 엘리먼트들은 자신의 부모 노드가 무엇인지 정확하게 가리키지 못한다.

3.4.2 두 번째 방법

그림 10의 그래프는 그림 6의 DTD 정의에 부합된다. DTD 정의에 따라 mail 엘리먼트는 하나의 text 엘리먼트를 자식노드로 갖는다. keyword, emph 엘리먼트도 정확하게 text 엘리먼트를 부모 엘리먼트로 갖는다. 하지만, virgin~, discomfort~, gold~으로 시작하는 text 엘리먼트의 데이타들은 모두 함께 저장되어서, RDBMS에 저장된 데이타를 XML 구조로 재구성할 경우 keyword, emph 엘리먼트의 정확한 위치를 나타내지 못하는 문제가 발생한다.

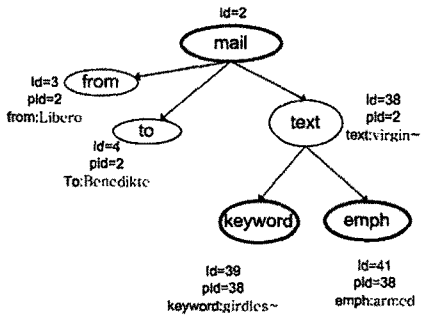


그림 10 모호한 그래프2

이 문제는 *에지를 가지는 #PCDATA가 text 엘리먼트로 인라인되면 동일한 textID에 동일한 ParentID를 가질 수 없기 때문에 발생한다. 본 논문에서는 *에지를 가지는 #PCDATA일 경우 #PCDATA를 공유 노드로 분리하고 별도의 테이블로 생성하여 이 문제를 해결한다. 이에 대해서는 다음 장에서 자세히 설명한다.

4. 제안 기법

본 장에서는 중첩 구조에서의 문제점을 해결하고 기존 알고리즘을 보완하는 방법을 제시한다. 하위에 *에지로 연결되는 #PCDATA 타입을 가질 경우 별도의 노드로 구성하여 중첩 구조의 저장문제를 해결한다. 제안 기법은 DTD 단순화, 그래프 생성, 그래프 최적화, 노드 인라인, 스키마 생성의 순서로 진행한다.

4.1 DTD 단순화

DTD를 기반으로 관계형 스키마를 생성하기 위해서는 먼저 DTD 파일을 읽어 정의된 형식을 분석한다. DTD 파일을 읽어 그래프 생성 시에 다음의 규칙을 적용하여 단순화한다.

그림 11의 규칙은 그림 1의 규칙에 6, 7을 추가하였다. 이 규칙을 적용하여 *, (,) 연산자와 노드만 표현되도록 단순화시킨다. 이때 남은 노드에는 *에지로 연결되는 #PCDATA 노드도 포함된다.

4.2 그래프 생성

Shared Inlining DTD 그래프를 생성하기 위한 각 노드의 정보는 그림 12와 같이 정의한다.

Vertex는 DTD에 정의된 서브그래프들의 각 정점을 나타낸다. Vertex는 하위그래프에 속하는 노드들의 공통정보를 저장한다. visited는 그래프 탐색시의 Vertex의 방문여부를 나타내고, sharedroot는 해당 Vertex의 하위그래프가 공유 가능한 그래프로 분리될 수 있는지를 나타낸다. *incoming*은 해당 노드의 진입차수이다. 진

1. $e^{+} \rightarrow e^{*}$
2. $e? \rightarrow e$
3. $(e_1|...|e_n) \rightarrow (e_1, ..., e_n)$
4. $(a)(e_1, ..., e_n)^{*} \rightarrow (e_1^{*}, ..., e_n^{*})$
   $(b) e^{**} \rightarrow e^{*}$
5. $(a) ..., e, ..., e, ... \rightarrow ..., e^{*}, ...$
   $(b) ..., e, ..., e^{*}, ... \rightarrow ..., e^{*}, ...$
   $(c) ..., e^{*}, ..., e, ... \rightarrow ..., e^{*}, ...$
   $(d) ..., e^{*}, ..., e^{*}, ... \rightarrow ..., e^{*}, ...$
6. 에트리뷰트의 #REQUIRED, #IMPLIED, CDATA, ID, IDREF, IDREFS EMPTY, NMTOKEN, NMTOKENS 속성 타입 제거
7. *에지로 연결되는 #PCDATA의 노드 생성

그림 11 DTD 단순화 규칙

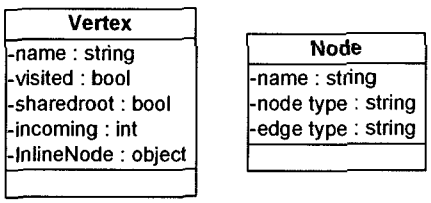


그림 12 각 정점과 노드 정의

입차수가 2 이상이면 해당 Vertex의 하위그래프는 별도의 그래프로 분리된다. InlineNode는 Vertex에서 연결되는 인라인 가능한 Node들의 정보를 리스트로 저장하여 관계형 스키마를 생성한다.

Node는 그래프에 존재하는 모든 노드의 개별 정보를 나타낸다. 노드명인 name, 노드가 ELEMENT인지 ATTLIST인지 구분하는 node_type, *에지인지 단순에지인지 나타내는 edge_type으로 구성된다.

그래프는 DTD의 단순화 규칙을 적용하여 생성한다. 그림 13은 단순화된 그래프의 노드를 생성하는 알고리즘이다.

4.3 그래프 최적화

단순화 과정을 거쳐 생성된 그래프는 최적화 단계를 거쳐야 한다. 현재 단계까지 생성된 형태에서는 아직 공유 가능한 노드와 인라인 가능한 노드가 구분 되지 않았다. 공유 노드와 인라인 노드의 규칙은 다음과 같다.

- 공유 노드
  - 진입차수가 2 이상인 엘리먼트
  - 에지가 "*" 인 엘리먼트
  - 에지가 "*" 인 #PCDATA
- 인라인 노드
  - 진입차수가 1인 엘리먼트
  - 에트리뷰트 노드

```
/*
DTD파일을 읽어 한 라인씩 파싱할 때마다 그래프의 노드를
생성하는 MakeNode를 호출함
*/

// 그래프의 vertex와 연결되는 node 정보를 함께 인자로 받음
MakeNode(string vertex_name, string node_name, string
        node_type, string edge_type)
BEGIN
  // 해당 vertex를 확인 후에 없다면 vertex를 먼저 생성함
  IF (vertex[vertex_name] == null) THEN
  Begin
    vertex.add(vertex_name, new MyNodes(vertex_name));
  END IF

  // 해당 node가 vertex정보에 없다면 node에 관련된 vertex 생성
  IF (vertex[node_name]==null) THEN
  Begin
    vertex.add(node_name, new MyNodes(node_name));
  END IF

  // vertex에 연결되는 node 생성
  vertex[vertex_name].Add(new MyNode(node_name, node_type,
  edge_type));
END Procedure
```

그림 13 노드 생성

- 단순 에지인 엘리먼트
- 단순 에지인 #PCDATA

그림 14는 공유 노드와 인라인 노드 조건을 검사하는 알고리즘이다.

그래프는 루트로부터 깊이 우선 탐색을 진행하며 위와 같이 조건을 검사한다. 방문된 노드는 visited로 체

```
CheckSharedNode(MyNodes N)
BEGIN
/*
조건을 검사하여 공유노드에 해당된다면 노드정보의
sharedroot를 TRUE로 저장함
*/
  IF (N.node_type=="ELEMENT") THEN
  Begin
    IF (N.name != "#PCDATA") THEN
    Begin
      vertex[N.name].incoming++;
      IF (vertex[N.name].incoming>1) THEN //진입차수 2 이상
        vertex[N.name].sharedroot = true;
      END IF
    END IF
    IF (N.edge_type=="*") THEN // *에지로 연결될 경우
      vertex[N.node].sharedroot = true;
    END IF
  END IF
END Procedure
```

그림 14 공유 노드 검사

```
EliminateSharedNode(MyNodes N)
Begin
/*
각 그래프를 방문하여 연결된 자식 노드가 중간노드이고 다
른 그래프의 루트가 되면 해당 노드 삭제함
*/
 N.visited := TRUE  //노드 방문체크
 FOR each child C of N DO  //연결된 모든 자식노드 검사
   IF ((IsInternalNode(C))&&
       (vertex[C.name].sharedroot)) THEN
         N.Remove(C)
   END IF
 END FOR
END

IsInternalNode(Node N)  //중간노드여부 검사
Begin
//노드타입이 ELEMENT이고 #PCDATA이 아니면 TRUE 리턴
 IF ((N.nodetype=="ELEMENT")&&
     (N.name!="#PCDATA")) THEN
   RETURN TRUE
 ELSE
   RETURN FALSE
END Procedure
```

그림 15 그래프 최적화 알고리즘

크하고, incoming 값을 1 증가시킨다. 탐색은 노드마다 공유 노드인지 인라인 노드인지 조건을 검사하여 공유 노드의 조건과 부합된다면 sharedroot 값을 TRUE로 변경한다. 공유 노드는 독립적인 그래프로 분리될 수 있는 Vertex가 되며, 관계형 스키마에서 독립된 테이블로 생성된다.

그래프의 탐색을 마치면 각 그래프의 Vertex마다 연결된 하위 노드를 검사하여 공유 노드에 해당되는 노드를 삭제한다. 이러한 방법을 통해 각 노드는 전체 그래프에서 중복 존재하지 않게 된다. 그림 15는 그래프 내에서 공유 노드에 해당되는 중복 노드를 제거하는 알고리즘을 나타낸다.

그래프 최적화 단계가 종료된 후에는 각 Vertex에는 인라인 가능한 노드의 연결만 남게 되고, 제거된 노드들은 자신에게 인라인되는 하위 노드들과 별도의 그래프로 구성된다.

4.4 노드 인라인

노드 인라인 과정에서는 각 그래프의 정점을 탐색하여 하위로 연결된 인라인 가능한 노드들을 확인한다. 이후 관계형 스키마를 생성하기 위한 SQL을 생성한다. 해당 노드가 리프노드이면 text 타입, 중간 노드이면 자신의 하위 노드의 ParentID로 참조되므로 int 타입으로 생성한다. 노드 인라인에 사용되는 알고리즘은 그림 16에 표현하였다.

```
NodeInlining(string SharedRootName, MyNodes CurNode,string
path)
Begin
  CurNode.visited = true; //그래프 방문 체크
  FOR each child C of CurNode DO //모든 자식노드 검사
    string child = C.name;
    IF (IsInternalNode((CurNode)[i])) THEN
      //노드로 연결된 방문되지 않은 그래프 NodeInlining
      IF (!vertex[child].visited) THEN
        NodeInlining(SharedRootName,vertex[child],path+"."+
        child);
      END IF
    ELSE
     /*
     현재 노드명이 #PCDATA일 경우 그래프의 inlineNode
     에 SQL 형식로 추가함
     */
      IF ((CurNode)[i].name!="#PCDATA") THEN
        vertex[SharedRootName].inlineNode.Add("\""+path+".
        "+ CurNode[i].name + "\" text");
      END IF
    END IF
  END FOR
 /*
  관계형 테이블에서 생성 될 필드명과 필드타입을 현재 노
 드의 path와 노드의 형태에 따라서 SQL형식으로
 inlineNode에 추가함
 */
  IF (IsLeafNode(CurNode)) THEN  //리프노드일 경우
    vertex[SharedRootName].inlineNode.Add("\""+path+   "\"
    text");
   ELSE
    vertex[SharedRootName].inlineNode.Add("\""+path+   "\"
    int");
  END IF
 /*
 현재노드가 인라인되지 않는 각 그래프의 루트노드이면 개
 별로 생성되는 테이블이므로 parentid, id 필드가 추가된다.
 */
  IF (SharedRootName == CurNode.name) THEN
    vertex[SharedRootName].inlineNode.Add("parentid int");
    vertex[SharedRootName].inlineNode.Add(CurNode.name+"id
    int");
  END IF
END Procedure
```

그림 16 노드의 인라인

4.5 스키마 생성

알고리즘을 적용하여 제작한 실험 프로그램은 그림 17이다. "파일읽기"를 통해 DTD 파일을 분석하여 Shared Inlining 구조에 맞는 관계형 스키마를 출력하고 "스키마 생성"으로 데이타베이스에 관련된 스키마를 생성한다.

제시한 방법으로 생성한 테이블은 29개이고, 각 테이블의 구조는 표 2와 같은 형태로 표현된다.

표 2 관계형 스키마 구성 예

annotation	AnnotationId, ParentId, Annotation, Annotation.author, Annotation.author.person, Annotation.happiness
category	Categoryid, ParentId, Category, Category.id
closed_auction	Closed_auctionId, ParentId, Closed_auction, Closed_auction.buyer, Closed_auction.buyer.person, Closed_auction.price
description	DescriptionId, ParentId, Description
edge	EdgeId, ParentId, Edge, Edge.from, Edge.to
person	PersonId, ParentId, Person, Person.emailaddress, Person.phone, Person.address, Person.address.street, Person.address.city, Person.address.country, Person.address.province, Person.address.zipcode, Person.homepage, Person.creditcard, Person.profile, Person.profile.education, Person.profile.gender, Person.profile.business, Person.profile.age, Person.profile.income, Person.watches, Person.id
quantity	quantityID, parentID, quantity
site	SiteId, ParentId, Site.categories, Site, Site.regions, Site.regions.africa, Site.regions.asia, Site.regions.europe, Site.regions.namerica, Site.regions.samerica, Site.regions.australia, Site.catgraph, Site.people, Site.open_auctions, Site.closed_auctions
status	statusID, parentID, status
text	textID, parentID, text
type	typeID, parentID, type
....	....
_PCDATA	_PCDATAID, parentID, _PCDATA

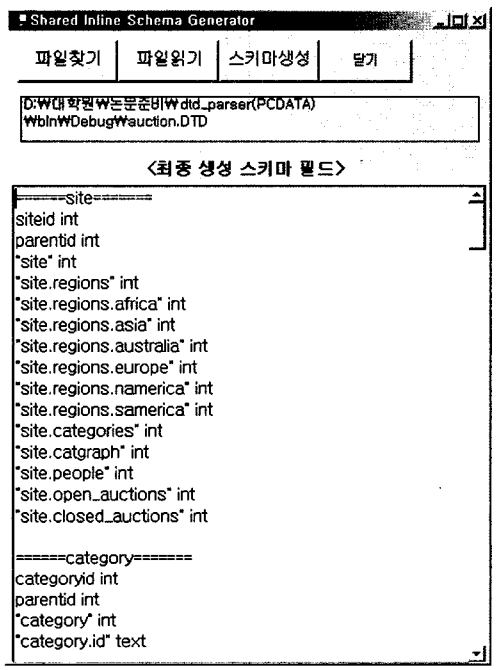


그림 17 Shared Inlining Schema Generator

표 3은 제안한 방법으로 생성한 각 테이블과 레코드 수를 나타낸다. 굵은 글씨체로 표시한 테이블들은 기존의 방식으로 생성한 표 1의 결과와 차이가 있는 테이블이다. 제시하는 방법에서는 _PCDATA라는 테이블이 추가되었으며, 모호한 그래프에 해당되던 bold, emph, keyword, text 테이블의 레코드 수가 변경되었다.

4.6 구현 결과

표 3 제안기법 적용 후 생성한 스키마

테이블	레코드	테이블	레코드
amount	0	annotation	21,750
bidder	59,486	**bold**	71,856
category	1,000	closed_auction	9,750
date	90,182	description	44,500
edge	1,000	**emph**	69,588
incategory	82,151	interest	37,689
item	21,750	itemref	21,750
keyword	69,969	listitem	60,481
mail	20,946	name	48,250
open_auction	12,000	parlist	20,813
person	25,500	quantity	43,500
seller	21,750	site	1
status	0	**text**	105,114
type	21,750	watch	50,269
_PCDATA	502,232		

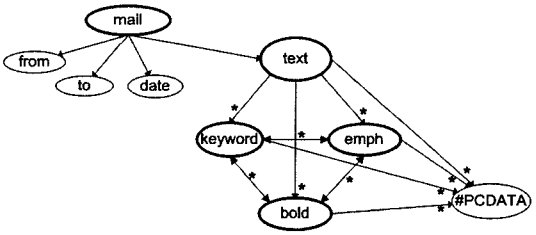


그림 18 개선된 Shared Inlining DTD 그래프

본 장에서 제시한 방식으로 생성한 그림 7의 개선된 그래프는 그림 18과 같다. DTD 정의에서 *에지를 가지는 #PCDATA를 별개의 노드로 추가하였다.

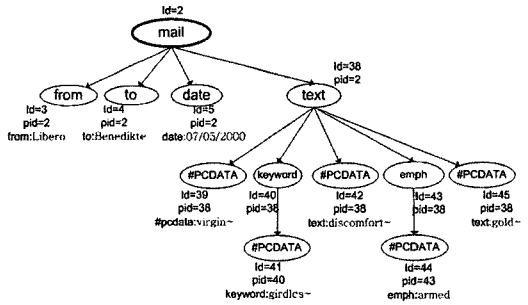


그림 19 개선된 그래프 표현

	textid	id	pcdata
1	[redacted]	39	virgin preventions half logotype weapons granted factious already carved fretted ...
2	38	41	girdles deserts flood george nobility reprieve
3	38	42	discomfort sinful conceiv corn preventions greatly suit observe sinews enforcement
4	38	44	armed
5	38	45	gold gazing set almost catesby turned servilius cook doublet preventions shrunk ...

그림 20 중첩 #PCDATA의 질의 결과

추가된 #PCDATA 엘리먼트는 공유 노드가 되며, * 에지로 정의된 #PCDATA의 값을 저장한다. 이는 DTD의 정의를 그대로 따르며 분할되어 나눠지는 #PCDATA의 경우 각 엘리먼트에 id를 부여하여 정확하게 노드들을 표현할 수 있다.

그림 9, 그림 10의 모호한 그래프 표현은 그림 19와 같이 개선된다. text, keyword, emph 엘리먼트 모두 * 에지의 #PCDATA 노드로 표현되었으며, 해당 노드는 #PCDATA 데이타를 가지는 공유 노드를 자식노드로 가진다.

id가 40인 keyword 엘리먼트와 id가 43인 emph 엘리먼트는 정확히 자신의 부모인 text 엘리먼트를 가리킨다. 또한 text 엘리먼트의 3개의 문자열 데이타도 동일한 부모 엘리먼트인 text 엘리먼트에 속한다. 결과적으로 text 엘리먼트는 DTD 정의에 따라 자식 엘리먼트로 #PCDATA, keyword, emph 엘리먼트를 가질 수 있으며 분할되는 #PCDATA 사이에서 각 엘리먼트는 정확하게 자신의 부모 엘리먼트와 연결될 수 있다. 이렇게 하여 3장의 중첩 구조에서 발생하는 엘리먼트의 모호한 연결의 문제점을 해결하고 정확하게 표현 가능하다.

그림 20은 id가 38인 text 엘리먼트의 자식 노드를 표현한 질의 결과이다. 이 질의를 통해서 text 엘리먼트에서 연결되는 자식 엘리먼트들을 순서에 따라 정확하게 표현되었다는 것을 알 수 있다.

5. 결 론

본 논문에서는 Shared Inlining 저장 기법에서 발생하는 중첩 구조에서의 문제점을 보완한 방법을 제안하였다. 제시한 방법을 통해 Shared Inlining 스키마의 생성 방법을 보완하여 XML 문서의 구조를 관계형 데이타베이스로 정확하게 저장하고 검색할 수 있다.

표 4 제안기법 적용 후 질의응답 속도(sec)

	Shared Inlining	
질의	개선 전	개선 후
Q1	1.047	1.735
Q2	2.0	2.0
Q4	1.58	1.68
Q5	0.49	0.73
Q6	0.475	0.318
Q8	3.0	4.0
Q9	5.0	5.0
Q11	106	107
Q12	13	13.5
합계	132.592	135.963
평균	14.73	15.10

본 논문에 제안된 방법이 기존의 방법과 비교하여 질의처리 성능을 저하시키는지를 실험으로 확인하였다. 표 4는 Shared Inlining 방법의 개선 전과 개선 후의 질의 성능 비교이다. 실험에 사용된 질의는 XMark benchmark에서 제시한 20개 중에서 문자열 검색, 정규 경로 표현식, 참조, 조인의 성능을 나타내는 질의로서 응답시간이 긴 9개를 선택하여 수행 속도를 비교하였다.

실험 결과를 통해서 제안된 방법이 질의 처리 성능에 별다른 영향을 주지 않으면서 XML 데이타의 정확한 저장과 검색이 가능함을 확인하였다.

향후의 연구 과제로는 DTD 뿐만 아니라 XML 스키마를 이용하여 관계형 스키마를 생성하는 방법이 필요하다. XML 스키마는 한정된 DTD의 표현 방식보다는 복잡하고 많은 표현이 가능하다. 모든 XML 문서를 관계형 데이타베이스로 저장하는 여러 가지 연구들을 통해 효율적인 XML 저장 및 관리를 기대한다.

참 고 문 헌

[1] 이혜자, 정병수, 김대호, 이영구, "경로정보의 중복을 제거한 XML 문서의 저장 및 질의처리 기법", 한국정보처리학회 논문지, VOL.12-D, NO.05, pp. 0663-0672, 2005.

[2] Schmidt, A.R., Waas, F., Kersten, M.L., Florescu, D., Manolescu, I., Carey, M.J., Busse, R., The XML benchmark project, CWI Technical Report, 2001.

[3] HONGJUN LU, JEFFREY XU YU et al., "What Makes the Differences: Benchmarking XML Database Implementations," ACM Transactions on Internet Technology, Vol.5, No.1, pp. 154-194, February 2005.

[4] MASATOSHI YOSHIKAWA and TOSHIYUKI AMAGASA et al., "XRel: A Path-Based Approach

to Storage and Retrieval of XML Documents Using Relational Databases," ACM Transactions on Internet Technology, Vol. 1, No. 1, pp. 110-141, August 2001.
[5] Roy Goldman and Jennifer Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," Proceedings of 23rd International Conference on Very Large Data Bases, 1997.
[6] Kentarou Kido, Toshiyuki Amagasa, Hiroyuki Kitagawa, "Processing XPath Queries in PC-Clusters Using XML Data Partitioning," ICDE Workshops, pp. 114, 2006.
[7] Dao Dinh Kha, Masatoshi Yoshikawa, Shunsuke Uemura, "An XML Indexing Structure with Relative Region Coordinate," 17th International Conference on Data Engineering (ICDE'01), pp. 313-320, 2001.
[8] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. De-Witt, J. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," Proceedings of the 25th VLDB Conference, 1999.
[9] "XML Path Language (XPath) 2.0 W3C Recommendation 23 January 2007," http://www.w3.org/TR/xpath20/
[10] "XML Query (XQuery)," http://www.w3.org/XML/Query/
[11] "XMark," http://monetdb.cwi.nl/xml/.
[12] Shiyong Lu, Yezhou Sun, Mustafa Atay, Farshad Fotouhi, "A New Inlining Algorithm for Mapping XML DTDs to Relational Schemas," In Proceedings of the 1st International Workshop on XML Schema and Data Management, in Conjunction with the 22nd ACM International Conference on Conceptual Modeling, Vol. 2814 of Lecture Notes in Computer Science, pp. 366-377, 2003.
[13] Mustafa Atay, Artem Chebotko, Dapeng Liu, Shiyong Lu, Farshad Fotouhi, "Efficient Schema-Based XML-to-Relational Data Mapping," Information Systems (IS), Vol. 32, No. 3, pp. 458-476, 2007.

홍 은 지

1989년 2월 서울대학교 계산통학과 졸업(학사). 1991년 2월 서울대학교 전산과학과 졸업(석사). 1996년 8월 서울대학교 전산과학과 졸업(박사). 1997년부터 현재까지 성공회대학교 소프트웨어공학과 교수로 재직. 관심분야는 데이타베이스, 객체지향시스템, XML, GIS 등

이 영 호

2005년 2월 성공회대학교 정보통신학과 졸업(학사). 2007년 8월 성공회대학교 IT학과 졸업(석사). 2005년부터 현재까지 (사)iCOOP생협연대 전산팀 재직. 관심분야는 데이타베이스, XML 등