# XML Schema에 대한 관계형 스키마 자동 생성 시스템

## (An Automatic Relational Schema Generating System for an XML Schema)

김 정 섭 † 박 창 원 †† 정 진 완 †††
(Jung-Sup Kim) (Chang-Won Park) (Chin-Wan Chung)

**요 약** 점점 더 많은 문서들이 XML의 형태를 갖게 됨에 따라, XML문서들을 관계형 데이타베이스에 저장하기 위한 관계형 스키마의 생성이 더욱 중요해 지고 있다. 이 논문은 최근 W3C에 의해서 제안된 XML Schema로부터 관계형 스키마를 생성해 내는 기법 및 구현에 대해서 설명한다. 기존의 DTD기반 인라인 기법은 XML Schema에 적용될 수 없는데, 이유는 XML Schema에는 DTD에 존재하지 않는 새로운 특징들이 많이 있기 때문이다. 즉, 다양한 데이타 타입, 상속, 다형성과 같은 요소들이 추가되어, XML Schema는 보다 강력해진 반면, 이로부터 관계형 스키마를 생성하는 일은 더욱 어렵게 되었다. 본 논문에서는 기존의 인라인 기법을 기반으로 XML Schema 인라인 기법을 제시하였다. XML Schema 인라인 기법은 먼저 XML Schema의 다양한 데이타 타입들을 관계형 데이타베이스의 타입으로 매핑시키는 작업을 한 후, 타입과 엘리먼트 정보를 이용하여 스키마 그래프와 타입 그래프를 만들고, 이 그래프를 순회하면서 관계형 테이블들을 생성하게 된다. 그 외에도 xsi:type, 익명 타입들을 처리하기 위한 기법들을 소개하고 있으며, 시스템의 성능을 향상시키기 위한 몇몇 휴리스틱 기법에 대해서도 소개하였다. 마지막으로 이진 테이블 저장 방식과의 성능 비교 실험을 통하여 XML Schema 인라인 기법의 우수성을 보였다.

**키워드** : XML, XML Schema, relational schema

***Abstract*** As more and more documents are published in XML, generating relational schemas to store XML documents in a relational database is also getting important. This paper describes a technique as well as its implementation to produce a relational schema from the XML Schema, a standard recently recommended by W3C. The DTD-based inlining technique cannot be applied to the XML Schema, because the XML Schema has many new features, which don't exist in the DTD. Various built-in data types, inheritance, and polymorphism, for example, strengthen the XML Schema, but make the generation of a relational schema from an XML Schema more difficult. We propose an XML Schema Inlining Technique, based on the previous work. The technique first maps various data types in the XML Schema to those of the relational database. After that, it construct the schema graph and the type graph from types and elements defined in the XML Schema. The relational schema is generated while traversing the type graphs. Besides, we describe techniques for handling xsi:type, used for the polymorphism, and the anonymous type. We also propose a couple of heuristic methods for enhancing the performance of the system. Finally, we conducted experiments to show that our technique is better than the binary table approach.

**Key words** : XML, XML Schema, relational schema

· 본 연구는 대학 IT연구센터 육성 · 지원사업의 연구결과로 수행되었음
† 비 회 원 : LG전자 디지털미디어연구소 연구원
prnbada@lge.com
†† 비 회 원 : LG전자기술원 정보기술연구소 연구원
cwpark@lge.com
††† 종신회원 : 한국과학기술원 전산학과 교수
chungcw@cs.kaist.ac.kr
논문접수 : 2002년 11월 1일
심사완료 : 2004년 6월 3일

## 1. 서 론

HTML과 SGML의 단점을 보완하고자 제안된 XML(eXtendible Markup Language)은 매우 빠른 속도로 각종 애플리케이션에 사용되고 있다. 전자 상거래에서는 이미 활발히 사용되고 있으며, 각종 문서의 인코딩, 멀티미디어 데이타의 인코딩에도 XML이 사용되고 있다[1,10].

이와 함께, XML문서를 저장하고 검색하기 위한 노력이 경주되고 있다. XML문서의 저장은 오브젝트 저장소와 같은 특수한 목적의 소프트웨어를 사용하는 경우도 있으며, 객체 지향 데이타베이스 관리 시스템이나 관계형 데이타베이스 관리 시스템을 사용하는 경우도 있다. 한편, 관계형 데이타베이스 관리 시스템의 경우에는 이미 많은 사용자를 확보하고 있기 때문에 XML문서를 관계형 데이타베이스에 저장하기 위한 방법들에 대해서 보다 많은 관심을 받고 있다.

그러나, 계층적 구조를 가진 XML문서를 평평한 구조의 관계형 데이타베이스에 저장하는 작업은 쉽지 않다. 무엇보다도 관계형 테이블로는 계층 구조를 표현하기 어렵기 때문이다. 대표적으로 기존의 준구조적(semi-structured) 데이타를 저장하기 위해서 사용되던 간선(edge)방식의 저장 방법은 많은 조인 연산의 필요성으로 인해 성능이 저하되는 단점이 있다. 이에 비해, 일정한 속성을 공유하는 데이타(엘리먼트)들을 하나의 테이블에 "인라인" 시키는 방법은 간선 방식의 성능 저하 문제점들을 어느 정도 해결해주고 있다. 현재로서는 DTD[11]를 이용하는 인라인 기법이 제안되어 있으며, DTD를 대체하기 위해 W3C에 의해서 새로이 제정된 XML Schema[2,12]를 위한 인라인 기법에 대한 연구는 없었다.

본 논문은 XML Schema로부터 관계형 스키마를 자동으로 생성하는 방법 및 구현에 대하여 설명한다. XML Schema는 데이타 중심의 XML문서를 위하여 사용될 수 있도록 DTD와는 다른 많은 새로운 요소들을 포함하고 있다. 다양한 데이타 타입, 상속, 다형성 등과 같은 요소들은 XML Schema를 보다 강력한 스키마로서의 역할을 할 수 있도록 해 주지만, 한편으로는 DTD를 이용하는 인라인 기법이 더 이상 XML Schema에 적용될 수 없는 결과를 초래하였다. 본 논문의 공헌은 XML Schema의 다양한 각각의 요소들에 대해서 관계형 스키마를 어떻게 생성할 것인가를 제시하고 있으며, 다양한 휴리스틱 방법들을 통하여 보다 성능 향상된 XML Schema 인라인 기법을 제안하고 있다는 점이다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 먼저, XML문서를 관계형 데이타베이스에 저장하기 위한 방법들을 살펴보고, 그 다음으로 XML Schema를 DTD, 관계형 스키마와 비교하면서 설명한다. 3장에서는 본격적으로 시스템의 각 부분들에 대해서 상세히 설명한다. 4장에서는 구현된 시스템에 대한 실험결과를 소개하고, 마지막으로 5장에서는 결론에 대해서 기술한다.

## 2. 관련 연구

### 2.1 XML문서의 저장

XML문서의 저장은 크게 DTD와 같은 스키마 정보가 있는 경우와 그러지 않는 경우로 나누어 볼 수 있다. 먼저, 스키마 정보가 없는 경우에는 첫째, XML문서로부터 스키마 정보를 추출해 내어 관계형 테이블들을 생성하는 방법과 둘째, XML문서를 구성하는 요소들, 즉 노드와 간선(edge)들만을 저장하는 방법이 있다. 첫째의 경우는 STORED[4]에 해당하고, 둘째는 준구조적(Semi-structured) 데이타를 저장하기 위한 방법들[7,13]과 간선(edge) 방식[5,6] 등이 있다.

[4]의 경우, XML문서로부터 데이타 마이닝 기법을 사용해서 관계형 테이블들을 만드는데, 본 논문에서 사용하는 인라인 알고리즘과 많은 면에서 유사하다. 그러나, [4]의 경우, 모든 데이타를 관계형 데이타베이스에 저장하는 것이 아니며, 일부는 준구조적 데이타를 위한 전용 저장소에 저장한다는 점이 본 논문에서 제안하는 방법과 본질적으로 다르다.

[5]에서는 기존의 준구조적 데이타의 저장 방법을 이용한 XML 문서의 저장 방법들을 소개하고 있다. 일반적으로 간선(Edge) 방식, 이진(Binary) 방식, 범용(Universal) 방식이 있는데, 이 중, 간선의 이름별로 별도의 이진 테이블을 만드는 이진(Binary) 방식이 가장 우수한 성능을 갖고 있는 것으로 실험결과가 나왔다. 그러나, 본 논문에서 수행한 실험 결과는 이진 방식 역시, 계층 구조내의 중간 간선 정보를 저장해야 하기 때문에 전체 테이블의 크기가 원래 XML문서보다 더 커지는 경향을 보였으며, 많은 횟수의 조인 연산으로 인해 성능이 저하됨을 보였다.

한편, XML문서의 스키마로부터 관계형 스키마를 생성하는 방법에는 [3][14]과 [8]이 있다. [3]의 방법에서는 DTD에서 표현될 수 있는 내용들 중에서 실제 관계형 테이블들을 만드는데 사용될 수 있는 정보들만을 이용하여 DTD Graph와 Element Graph를 만들고, Element Graph를 깊이 우선 순회하면서 방문하는 엘리먼트들을 하나의 관계형 테이블에 인라인 시키는 방법을 사용하고 있다. [3]에서는 소개하는 인라인 기법은 본 논문에서 제안하는 XML Schema 인라인 기법과 기본적으로 동일하므로 여기에서 보다 자세히 살펴보기로 한다. 아래는 DTD, DTD Graph, 그리고 Author 엘리먼트의 Element Graph를 보여 주고 있다.

Element Graph가 주어지면, 관계형 테이블들은 다음과 같은 방법으로 생성된다. 먼저, 그래프의 루트 엘리먼트에 해당하는 테이블을 생성한다. 그런 다음 루트 엘리먼트의 자손 노드들을 모두 이 테이블의 애트리뷰트로 인라인 시키게 되는데 다음과 같은 예외가 있다. 첫째는 * (그림 1에서 Book과 Author사이)를 만나는 경

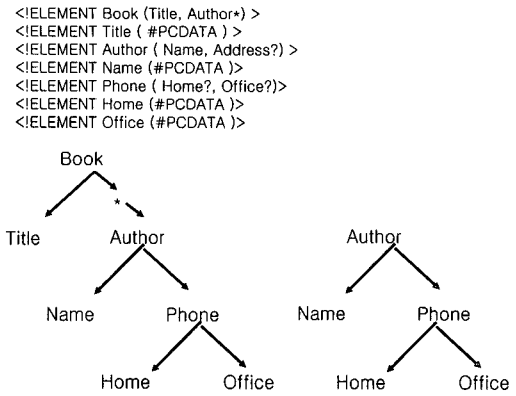



그림 1 DTD Graph와 Element Graph

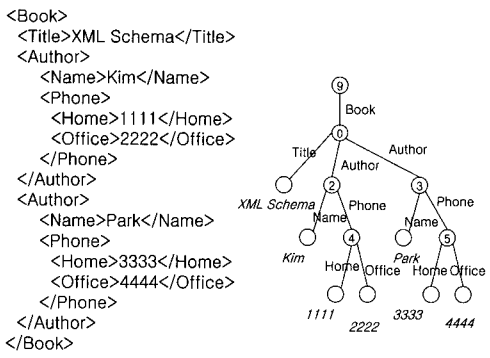



그림 2 XML문서와 구조

우이고, 둘째는 그래프 내에서 사이클을 만나는 경우이다. 이 경우에는 새로운 테이블을 생성하여 그 엘리먼트의 자손들을 인라인 시키게 된다. 그림 1의 그래프로부터 만들어지는 테이블들은 다음과 같다. ParentID에는 BookID의 값이 외래키로 저장되어 Book테이블과 Author테이블을 조인할 때 사용된다.

Book(BookID, Title)

Author(AuthorID, ParentID, Name, Phone.Home, Phone.Office,)

보다 명확한 이해를 위해서 위의 DTD를 이용해서 만들어진 XML 문서를 이진방식[5]과 인라인 방식[3]으로 각각 저장했을 때의 데이타베이스의 모습을 예로 나타내면 다음과 같다. 아래에는 위의 DTD를 이용해서 만들어진 XML문서와 함께, 이를 그래프를 이용해서 문서의 구조를 보여주고 있다. 오른쪽 그래프에서 원 안의 숫자는 객체의 ID를 나타낸다.

아래에는 이진 방식과 인라인 방식을 각각 사용했을 때의 테이블의 구조와 저장된 데이타를 보여주고 있다.

위에서 보는 것처럼 이진 방식은 그래프 상의 간선으로 테이블을 만들기 때문에 모두 7개의 테이블이 필요로 되어지는 반면, 인라인 방식은 2개의 테이블만이 필요로 되어진다. 한편, 이진 방식에서는 객체들의 ID가 테이블에 분산되어 저장되기 때문에, 원하는 정보를 얻

이진방식

Book

| source | target |
|---|---|
| 9 | 0 |

Title

| source | target |
|---|---|
| 0 | XML Schema |

Author

| source | target |
|---|---|
| 0 | 2 |
| 0 | 3 |

Name

| source | target |
|---|---|
| 2 | Kim |
| 3 | Park |

Phone

| source | target |
|---|---|
| 2 | 4 |
| 3 | 5 |

Home

| source | target |
|---|---|
| 4 | 1111 |
| 5 | 3333 |

Office

| source | target |
|---|---|
| 4 | 2222 |
| 5 | 4444 |

인라인방식

Book

| ID | Title |
|---|---|
| 0 | XML Schema |

Author

| ID | ParentID | Name | Phone.Home | Phone.Office |
|---|---|---|---|---|
| 2 | 0 | Kim | 1111 | 2222 |
| 3 | 0 | Park | 3333 | 4444 |

그림 3 이진 방식과 인라인 방식에서의 테이블

기 위해서는 관련된 테이블들을 조인하면 된다.

그러나, [3]에서 제안하는 방법들은 DTD를 위한 내용들이라서 XML Schema에는 적용되기 어렵다. 무엇보다 엘리먼트가 문서 구조의 주축을 이루는 DTD와는 달리, XML Schema는 타입과 엘리먼트라는 구성 요소가 주축을 이루고 있고 타입들은 상속과 다형성의 속성을 갖기 때문에 DTD Graph와 Element Graph를 그리는 것이 불가능하다.

### 2.2 XML Schema와 DTD

지금까지 DTD가 XML 문서의 구조를 정의하는 수단으로 사용되었다. 그러나, DTD는 여러 가지 문제점들을 내포하고 있는데, 첫째로 DTD는 XML과 다른 신택스(syntax)를 갖고 있다는 점이다. 이 점은 XML 개발자들에게 있어서는 DTD를 위한 별개의 파싱 프로그램이 존재해야 한다는 어려움을 야기한다. 둘째, DTD는 문자열 타입과 같이 매우 제한된 데이타 타입을 제공한다. 셋째, DTD는 엘리먼트들의 텍스트 컨텐트에 대해서 아무런 제약을 가할 수 없다는 문제점이 있다.

XML Schema는 DTD의 이러한 문제점들을 해결하기 위해 제안되었다. 첫째, XML Schema는 XML의 syntax를 따르고 있다. 즉, XML Schema자체가 하나의 XML 문서가 되며, 따라서 이미 존재하고 있는 XML문서 처리 기술을 이용해서 XML Schema를 처리할 수 있다.

둘째, XML Schema는 다양한 데이타 타입을 제공한다. string, byte, integer, date를 비롯한 40가지 이상의 Built-in 타입들을 제공하고 있으며(이들은 simple type이라고 불린다), 그 외에도 사용자는 자신이 원하는 타입들을 새로이 정의하여 사용할 수 있다.

셋째, XML Schema는 namespace를 지원한다. namespace를 이용하면 이미 만들어져 있는 다른 XML Schema로부터 타입과 전역 엘리먼트들을 재사용할 수 있는 장점이 있다. 또한, 여러 개의 XML Schema를 정의하여 하나의 XML문서를 작성하거나 문서의 타당성(validity)를 검증할 수도 있다.

넷째, XML Schema는 기본적으로 타입들로 구성되는데, 타입은 다시 simple type과 complex type으로 구분된다. Complex type은 애트리뷰트 또는 엘리먼트들을 이용하여 정의가 된다. 이러한 타입들은 상속이 가능하며, 추상 타입을 정의할 수 있는 점 등은 객체 지향 프로그래밍 언어와 같다.

아래에서는 간단한 예를 통해서 DTD와 XML Schema를 비교해 보기로 한다. 아래의 DTD와 XML Schema는 모두 ROOT라는 엘리먼트를 정의하고 있으며, 서브 엘리먼트로 A,B,C를 갖고 있고, A의 발생횟수(cardinality)는 0또는 1이고, B는 1번 이상, C는 0번 또는 1번 이상임을 나타내고 있다. 한편, A는 서브 엘리먼트로 E와 F를 갖고 있다.

```
<!ELEMENT ROOT ( A?, B+, C*)>
<!ELEMENT A(E,F)>
```

```
<element name="ROOT">
 <complexType>
  <sequence>
   <element name="A" minOccurs="0"/>
    <complexType>
     <sequence>
      <element name="E" type="string"/>
      <element name="F" type="string"/>
     </sequence>
    </complexType>
   </element>
   <element name="B" type="string" maxOccurs=
   "unbounded"/>
   <element name="C" type="string" minOccurs=
   "0" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

XML Schema의 경우, maxOccurs가 나타나지 않을 때에는 기본값이 1로 설정이 된다. 따라서 element A의 경우, 보다 완전한 형태로 표현하면 <element name ="A" type="string" minOccurs="0" maxOccurs="1"/> 이 된다. 한편, 위의 XML Schema에 나타난 complexType은 name애트리뷰트를 갖고 있지 않으므로 익명 타입이라고 부른다. 일반적으로 complexType은 name애트리뷰트를 가질 수 있는데, 이를 이용하여 위의 XML Schema를 다시 쓰면 아래와 같다.

```
<element name="ROOT" type="ROOTType"/>
<complexType name="ROOTType">
 <sequence>
  <element name="A" type="AType" minOccurs="0"/>
  <element name="B" type="string" maxOccurs=
  "unbounded"/>
  <element name="C" type="string" minOccurs=
  "0" maxOccurs="unbounded"/>
 </sequence>
</complexType>
<complexType name="AType">
 <sequence>
  <element name="E" type="string"/>
  <element name="F" type="string"/>
 </sequence>
</complexType>
```

### 2.3 XML Schema와 관계형 스키마

이 섹션에서는 본 논문에서 다루고자 하는 두 개의 스키마를 비교하면서, 이들 간의 차이점과 이로 인해 발생하는 변환에 있어서의 문제점들에 대해서 살펴 본다. 아래에는 XML Schema와 관계형 스키마를 비교하는 표가 나타나 있다.

표 1에서 보는 바와 같이 가장 먼저 해결해야 할 문제점은 시스템이 지원하는 데이타 타입의 상이이다. 관계형 스키마의 경우에는 DBMS의 제품에 따라서 지원되는 데이타 타입의 수에 차이가 있는데, 일반적으로 8~20개 정도의 타입을 지원한다. XML Schema의 경우에는 표에 나타난 시스템 지원 타입 이외에도 사용자가 임의로 데이타 타입을 정의하여 사용할 수 있으므로 이에 대한 고려도 필요하다. XML Schema를 관계형 스키마로 변환하는데 있어서 가장 큰 문제점은 구조의 차이에 있다. XML Schema의 경우에는 그림 1과 같은 계층 구조를 갖지만 관계형 스키마의 경우에는 테이블과 애트리뷰트로만 구성된 평평한 구조를 갖는다. 이를 해결하기 위해서 본 논문에서는 인라인 기법을 사용한다. 구조 변환 문제 이외에도 관계형 스키마에는 존재하지 않는 XML Schema의 객체 지향 개념들 - 상속, 다형성 - 을 처리해야 한다.

## 3. 관계형 스키마 자동 생성 시스템

그림 4는 시스템의 구성을 나타내고 있다. 우선 XML Schema가 시스템에 입력되면 타입 분리기가 XML Schema 내의 구성 요소들을 분류하여, 엘리먼트 처리기, 데이타 타입 변환기, 스키마 그래프 생성기 등을 거쳐서 최종적으로 관계형 스키마 생성기에 의하여 관계형 스키마를 생성하게 된다. 이 장에서는 각 부분들에 대해서 상세히 설명하도록 한다.

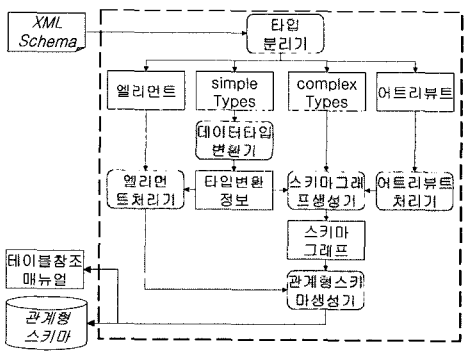



그림 4 시스템 구성도

### 3.1 타입 분리기

일반적으로 XML Schema의 구성은 전역 엘리먼트, 타입, 전역 애트리뷰트, 엘리먼트 그룹, 애트리뷰트 그룹으로 구성되어 있다. 타입 분리기가 하는 일은 이들을 구분하여 필요로 하는 다른 모듈에게 전달하는 것이다. 예를 들어, 전역 엘리먼트 선언을 만나게 되면, 이를 엘리먼트 처리기에게 전달한다. 타입의 경우에는 simple type과 complex type으로 구분하여 처리한다. 엘리먼트 그룹과 애트리뷰트 그룹은 각각 엘리먼트와 애트리뷰트로 처리한다.

### 3.2 데이타 타입 변환기

XML Schema는 string, byte, integer, date를 비롯한 40가지 이상의 다양한 Built-in 데이타 타입을 지원할 뿐 아니라, Built-in 데이타 타입으로부터 상속을 하여 새로운 데이타 타입을 만들어 낼 수 있다.

그러나, 일반적으로 관계형 데이타베이스시스템은 XML Schema에서 정의된 타입(simpleType) 들을 100% 지원하지 못한다. 예를 들어 오라클 DBMS의 경우, 문자열 타입을 위한 VARCHAR2, 또는 CHAR타입, 숫자타입을 위한 NUMBER타입, 그리고 DATE를 비롯한 몇몇 Built-in타입이 있을 뿐이다. 따라서, XML Schema를 관계형 스키마로 매핑시키기 위해서는 XML Schema에서 제공하는 타입들은 모두 관계형 데이타베이스 시스템에서 지원하는 타입들로 변환되어야 한다. 예를 들어, integer, float, double, unsigned 를 비롯한 모든 숫자와 관련된 XML Schema의 데이타들은 오라클 DBMS에서 지원하는 number타입으로 변환되어야 하며, 시간, URI 등을 위한 타입들은 VARCHAR2와 같은 문자열 타입으로 변환되어야 한다.

다음으로는 사용자가 XML Schema의 Built-in타입들을 상속 받아 새로이 정의한 simple Type들의 경우에는 XML Schema의 Built-in타입과 관계형 데이타베이스 시스템의 Built-in타입간의 매핑 정보, 그리고 사용자 정의 타입의 상속 정보를 이용하여, 관계형 데이타

표 1 XML Schema와 관계형 스키마

| | XML Schema | 관계형 스키마 |
|---|---|---|
| 시스템 제공 데이타 타입의 수 | 43 | 8~20 |
| 구조 | 계층 구조 | 평평한 구조 |
| 주요 구성 요소 | Type, Element, Attribute | Table, Attribute |
| Type/Table의 상속 관계 | 지원 함 | 지원하지 않음 |
| Type/Table의 다형성 | | |

베이스 시스템의 데이타 타입으로 매핑시켜야 한다. 예를 들어 XML Schema의 built-in타입 nonNegativeInteger로부터 상속을 받아서 만든 사용자 타입 unsigned1이 있고, nonNegativeInteger가 오라클 DBMS의 number타입에 매핑되었다면, unsigned1은 number 타입으로 매핑된다.

### 3.3 엘리먼트 처리기

엘리먼트 처리기로 전달되는 것은 크게 세 가지로 구분할 수 있다. 첫째는 XML문서의 루트 엘리먼트에 해당하는 엘리먼트이다. 이 엘리먼트는 나중에 관계형 스키마 생성기에 의하여 독립된 테이블을 생성하게 된다. 둘째는 전역 엘리먼트로서 complex type의 선언 때, 구성 요소로 참조된다. 셋째는 엘리먼트 그룹으로 둘째의 경우와 같이 complex type의 선언 때, 구성 요소로 참조된다. 한편, 엘리먼트가 XML문서의 루트 엘리먼트에 해당하는지 또는 단순한 전역 엘리먼트로만 사용되었는지의 여부는 스키마 그래프가 완성된 후에 파악이 가능하다.

### 3.4 스키마 그래프 생성기

XML Schema에는 실제 관계형 테이블을 생성하는데 도움을 주지 못하는 요소들이 있는데, 먼저 이들을 제거하는 스키마 단순화 작업을 진행하여야 한다. 대표적으로 <sequence>, <choice> 또는 minOccurs와 같은 요소들이다. 스키마 단순화 작업이란 이들을 스키마로부터 제거하기 위해 필요한 부가적인 작업과 함께 실제적으로 이들을 스키마 파일에서 제거하는 것을 말한다.

<sequence></sequence>는 이 태그 안에 열거되어 있는 엘리먼트들이 이 순서로 나타남을 의미한다. 하지만, 관계형 테이블에서는 이들이 칼럼 순서로 나타나기 때문에, 굳이 <sequence>라는 태그로 표시하지 않더라도 스키마에 나타난 순서대로 왼쪽 칼럼부터 오른쪽 칼럼으로 매핑이 되게 된다. 따라서 <sequence>와 </sequence>를 스키마 파일에서 제거하여도 된다. 다만, 이 때 주의 할 점은 이 태그의 애트리뷰트로 나타날 수 있는 maxOccurs의 값이다. 만약 이 값이 2이상이라면, 이 태그 안에 열거된 모든 엘리먼트들의 maxOccurs의 값에 위의 값을 곱하여 한다. 따라서, 비록 엘리먼트가 태그 안에서는 maxOccurs="1"일지라도 sequence를 없앰으로 인해, maxOccurs의 값이 변하게 된다.

<choice>의 경우도 <sequence>의 경우와 유사하다. 이들 역시 관계형 테이블의 형성에는 직접적인 영향을 미치지 못하기 때문에 스키마 파일에서 제거할 수 있다. 다만, 이 때에도 maxOccurs의 값이 2이상일 때에는 내부의 엘리먼트들의 maxOccurs값에 계산을 해 주어야 한다. minOccurs="0" 역시 관계형 테이블 형성에는 도움을 주지 않으므로 모두 제거할 수 있다

### 3.5 스키마 그래프

[3]에서는 관계형 테이블들을 생성하기 위해 DTD Graph와 Element Graph를 만들었다. DTD에는 문서의 구조를 나타내는 주된 요소가 엘리먼트이므로 위의 두 그래프들은 엘리먼트들간의 관계를 표시함으로써 문서의 구조를 파악할 수 있도록 해준다. 그러나, DTD Graph로는 XML Schema의 구조를 표현할 수 없다. 무엇보다 XML Schema에 나타나는 타입들간의 상속 관계를 전혀 나타낼 수 없으며, 추상 타입으로 선언된 엘리먼트의 구조 역시 표현할 수 없다. 또한, xsi:type으로 인해 발생하는 다형성은 DTD Graph를 그리는 것이 불가능하도록 만든다. 한편, XML Schema에서는 엘리먼트의 이름이 유일하지 않다. DTD에서는 하나의 엘리먼트 이름에 대응되는 엘리먼트의 구조가 유일하지만, XML Schema에서는 엘리먼트의 이름이 같더라도 선언된 타입이 다르면, 엘리먼트의 구조가 완전히 다를 수 있다. 따라서 XML Schema에서는 Element Graph가 생성될 수 없다. 따라서, 본 논문에서는 XML Schema의 구조를 그래프로 나타내기 위해 스키마 그래프와 타입 그래프를 제시한다. 이 그래프들의 가장 큰 특징은 기본 요소로 타입과 엘리먼트를 동시에 다루고 있다는 점이다.

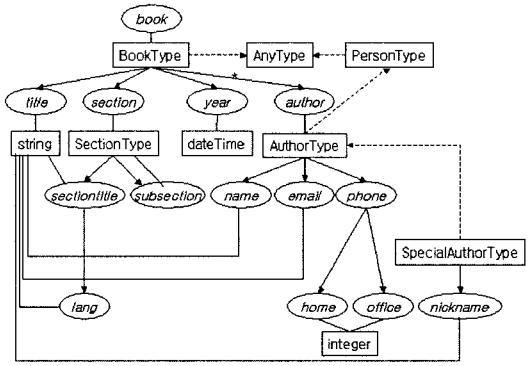



그림 5 스키마 그래프

위 스키마 그래프에는 여러 가지 형태의 데이타들이 나타나고 있는데 먼저, 사각형은 타입을 나타내며, 타원은 엘리먼트를 나타낸다. 실선으로 된 화살표는 엘리먼트의 내포관계를 나타내는데, BookType이라는 타입에는 title, section, year, author라는 엘리먼트들이 내포되어 있음을 보여주고 있다. 이 실선 위에는 발생 횟수(cardinality)를 나타내는 값이 올 수 있는데, *는 maxOccurs = "unbounded"로서 제한이 없음을 나타내며, 아무런 숫자도 표시되어 있지 않는 경우는 그 값이 1로

서 생략된 경우이다. 한편, 화살표가 없는 실선은 엘리먼트의 타입 정보를 나타낸다. 예를 들어, title이라는 엘리먼트의 타입은 string이며, section의 타입은 SectionType임을 위의 스키마 그래프에서 확인할 수 있다. 점선으로 표현된 화살표는 상속 관계를 나타내는 것으로 화살촉이 있는 쪽이 부모 타입이 되고, 반대편이 자식 타입이 된다. 예를 들어 AuthorType은 PersonType으로부터 상속을 받았으며, SpecialAuthorType은 AuthorType으로부터 상속받았음을 알 수 있다.

타입들 중에서 string, anyType, SectionType과 같이 상속 관계 트리의 루트에 있는 타입들을 **루트 타입**이라고 하고, NameType, SpecialAuthorType, SectionType과 같이 단말에 위치한 타입들을 **단말 타입**이라고 하며, 루트 타입을 포함하여 단말 타입이 아닌 타입들을 **비단말 타입**이라고 한다.

스키마 그래프가 만들어지면 [3]에서와 같이 특정한 타입에 대한 관계형 테이블 들을 만들기 위해서 타입 그래프를 만들게 된다. 먼저 타입 노드(AuthorType)에서 자식 쪽 상속 관계를 따라 가면서 자식 타입 노드(SpecialAuthorType)들과 그에 연결된 내포 관계를 타입 그래프에 표시 한다. 자식 타입 노드들은 다시 자신의 자식 타입 노드를 찾고 이를 그래프에 표시를 하며, 최종적으로 단말 타입 노드에 이를 때까지 반복한다. 이 작업이 끝나면, 부모쪽 상속 관계를 따라서 루트 타입(AnyType)에 이르기까지 방문하게 되는 타입 노드(PersonType)들과 그와 연결된 내포 관계들을 타입 그래프에 표시 한다. 위의 두 작업이 완료되면, 주어진 타입의 내포 관계들을 찾는다. 이 때 각각의 엘리먼트(name)들에 대해서 타입 관계를 통하여 타입 노드(string)를 찾고, 위의 방법을 반복한다. 다만, 위의 모든 경우에 이미 타입 그래프에 타입 노드가 나타나 있으면 관계만을 이어주고 더 이상 작업하지 않는다. 아래에는 AuthorType의 타입이 그려져 있다.

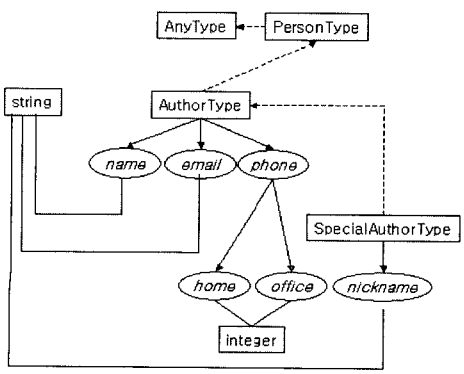



그림 6 타입 그래프

### 3.6 관계형 스키마 생성기

3.6.1 테이블 생성 기준

스키마 그래프를 순회하면서 관계형 스키마를 만들 때, 테이블을 생성하는 기준들은 다음과 같다. 첫번째는 [3]의 Element Graph에서 * 연산자를 만나는 것과 같은 경우이다. 즉, 주어진 타입에 내포되어 있는 엘리먼트의 maxOccurs의 값이 2이상 일 때 해당하는 엘리먼트는 독립적인 테이블로 만들게 된다.

둘째는 사이클이 형성되어 있는 경우에는 독립적인 테이블이 만들어지게 된다. 그림 [3]의 스키마에서는 SectionType을 독립된 테이블로 생성하게 된다.

셋째는 엘리먼트의 타입이 추상 타입으로 선언되어 있는 경우, 그 추상타입으로부터 상속된 구체 타입들은 독립된 테이블을 생성한다. 한편, 현재 생성중인 테이블에는 하나의 칼럼을 추가하여, 인스턴스 XML문서에서 어떤 구체 타입을 사용하는지, 즉 어떠한 테이블에 데이타가 저장되었는지를 표시한다. 이렇게 추상 타입으로 선언된 경우에 자식 타입들을 독립적인 테이블로 구성하는 이유는 XML문서를 보지 않은 이상, 서브 엘리먼트들의 구조를 알 수 없기 때문이다. 한편, 추상 타입으로 선언된 경우의 엘리먼트는 XML 문서 상에서 xsi:type이라는 애트리뷰트의 값을 통해서 서브 엘리먼트들의 구조를 지정하게 된다.

넷째는 루트 엘리먼트에 해당하는 경우에는 테이블을 생성한다.

다섯째 엘리먼트의 타입이 비단말 타입으로 선언된 경우에, 그 비단말 타입으로부터 상속된 자식 타입들은 독립된 테이블로 구성한다. 비단말 타입의 경우에도 xsi:type을 사용하여 XML문서에서 선언된 타입의 자식 타입을 사용할 수 있기 때문이다.

3.6.2 기본 인라인 알고리즘

지금부터는 위와 같은 테이블 생성 기준을 갖고, 타입 그래프를 순회하면서 관계형 스키마를 생성하는 알고리즘에 대해서 설명한다. 먼저, 사각형의 타입 노드를 위한 알고리즘을 살펴 본다. XML Schema는 상속이 가능한 타입을 기반으로 이루어져 있기 때문에, 이로부터 관계형 스키마를 생성하기 위해서는 관련된 모든 구조 정보를 파악해야 한다. 예를 들어, 상속이 가능한 시스템에서는 자식 타입의 구조를 정확히 알아내기 위해서 이들이 부모로부터 상속받은 엘리먼트, 애트리뷰트들도 모두 알아내야 한다. 그리고, 상속 관계 계층구조에서 루트를 포함하여 비단말에 해당하는 타입 노드들은 XML문서상에서 xsi:type이라는 애트리뷰트를 통해서 자신의 자식 타입을 사용할 수 있다. 이를 처리하기 위해서, 주어진 타입의 자식들을 순회하는 알고리즘도 필요로 된다. 이 정보들은 타입 그래프에서 얻을 수 있다.

타입 노드를 위한 알고리즘은 엘리먼트 노드를 위한 알고리즘과 긴밀히 연결되어 있다. 즉, 타입 노드에서 상속 관계를 따라 가면서 각각의 타입마다 내포된 엘리먼트 노드를 방문하게 된다. 엘리먼트 노드에서 해야 할 첫번째 일은 엘리먼트의 타입을 파악하는 일과 엘리먼트의 인라인 가능 여부를 파악하는 것이다. 즉, 인라인 될 수 있다면 현재 생성 중인 테이블에 이 엘리먼트를 인라인 시킬 것이며, 그렇지 않은 경우에는 새로운 테이블을 생성하게 된다. 테이블의 이름은 기본적으로 타입의 이름을 따르게 된다. 이점은 엘리먼트의 이름으로 테이블을 생성하는 [3]의 방식과는 상이한데, 이유는 다음과 같다. 즉, 다양한 엘리먼트들에 대해서 동일한 타입이 적용될 때, 이들은 가장 상위의 엘리먼트의 이름만 다를 뿐, 서브 엘리먼트들은 모두 동일하며, 관계형 테이블도 모두 같다. 따라서, 이들은 하나의 테이블로 관리할 수 있으며, 테이블의 이름은 타입의 이름을 따르게 된다. 물론, 이름이 동일한 엘리먼트에 다양한 타입이 선언되어 질 수 있지만, 이들은 모두 다른 관계형 테이블을 구성하게 되므로, 이 경우에도 엘리먼트의 이름으로 테이블을 생성할 수 없게 된다. 다만, 익명 타입으로 엘리먼트를 정의한 경우에는 엘리먼트의 이름으로 테이블을 생성하게 된다.

3.6.3 익명(Anonymous) 타입의 처리

익명 타입은, 앞에서도 잠깐 언급한 것과 같이, 타입에 이름이 부여되어 있지 않고, 하나의 특정한 엘리먼트의 선언을 위해서 사용되는 타입을 말한다. 익명 타입이 문제가 되는 경우는 이미 만들어진 타입으로부터 상속받는 경우이다. 비교적 간단한 처리 방법은 새로운 테이블 이름을 생성할 수 있다. 그러나, 이미 상속된 부모 타입에 해당하는 테이블이 존재하는 상황에서 거의 대부분의 칼럼이 동일한 새로운 테이블을 만드는 것은 바람직하지 않을 수 있다. 따라서, 기존의 테이블에 새로운 엘리먼트나 애트리뷰트들을 추가하는 방법도 고려해 볼 수 있으며, 새로운 엘리먼트가 없고 다만 애트리뷰트만 있는 경우에는 기존의 테이블에 추가하지만, 새로운 엘리먼트가 있는 경우에는 새로운 테이블을 만드는 혼합적인 방법을 사용할 수도 있다. 기존 테이블에 추가하는 경우에는, 불필요한 널(null) 값이 많아질 수 있다.

3.6.4 xsi:type의 처리문제

앞에서 설명한 것과 같이 xsi:type은 추상타입 뿐만 아니라, 구체 타입일지라도 비단말 타입 노드인 경우에 사용될 수 있다. 이처럼, 스키마에서 정의된 타입과는 다른 타입이 사용될 수 있다는 사실은 XML Schema만을 보고서는 정확히 어떠한 데이타가 어떠한 테이블에 저장되는지를 결정할 수 없음을 의미한다. 하지만, 다행히 xsi:type에 올 수 있는 타입은 스키마에서 정의된 타입으로부터 상속을 받은 타입들만 올 수 있다. 따라서, 우리는 스키마에서 선언된 타입의 자식 타입들을 위한 테이블들을 미리 생성해 둘 필요가 있게 된다.

이와 함께, 어떤 타입이 XML문서에서 사용되었는지를 저장해 두어야 하는데, 이 값은 xsi:type이라는 칼럼을 두어서 처리한다. 그런데, 이 칼럼을 어디에 두느냐는 경우에 따라서 조금 다르다. 만약 선언된 타입의 maxOccurs가 1이어서 인라인 되는 경우에는 인라인 되는 테이블에 xsi:type이라는 칼럼을 추가한다. 만약 maxOccurs가 unbounded이면, xsi:type칼럼은 새로이 생성되는 테이블에 추가가 된다. 아래에서 예를 들어 설명한다.

아래의 XML Schema는 PersonType과 이것으로부터 상속을 받은 EmployeeType과 CustomerType을 정의하고 있다.

```
<complexType name="PersonType">
 <attribute name="id" type="ID"/>
</complexType>
<complexType name="EmployeeType">
 <complexContent>
  <extension base="PersonType">
   <element name="Salary" type="number"/>
  </extension>
 </complexContent>
</complexType>
<complexType name="CustomerType">
 <complexContent>
  <extension base="PersonType">
   <element name="Rating" type="int"/>
  </extension>
 </complexContent>
</complexType>
```

위의 XML Schema는 완전한 것이 아니며, 루트에 해당하는 엘리먼트가 정의되어야 한다. 아래에서는 두 가지 경우의 엘리먼트에 대해서 생각한다. 첫째는 maxOccurs="1"인 경우로 아래 그림의 왼쪽 상단에 정의되어 있고, 둘째는 maxOccurs="unbounded"인 경우로 아래 그림의 오른쪽 상단에 정의되어 있다.

그리고, 각각의 경우에 해당하는 XML문서가 그림의 중단에 나타나 있다. 먼저, 왼쪽 경우부터 살펴 보자. 위의 XML Schema로부터 관계형 테이블을 만들게 되면, Doc, CustomerType, EmployeeType이 만들어지지만, 주어진 XML문서의 데이타들을 저장하기 위해서는 위의 그림과 같이 Doc와 CustomerType 테이블만 필요하다. 먼저, Person이라는 엘리먼트는 단 1번만 나타나기 때문에 Doc이라는 테이블에 Person.xsi:type이라는

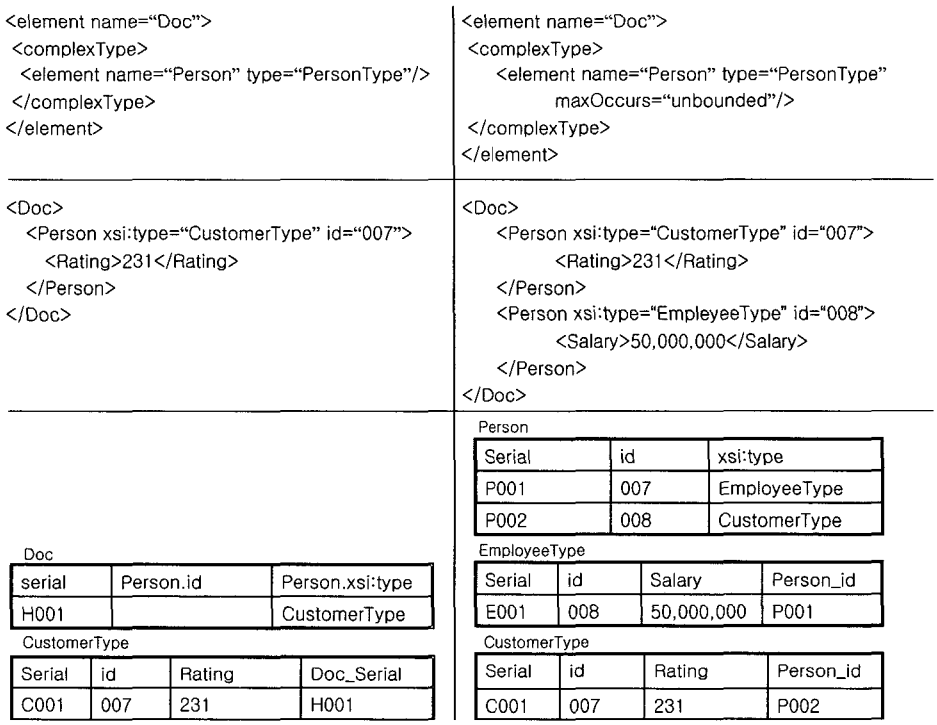



그림 7 xsi:type의 처리

칼럼을 추가하였다. 그리고, XML 문서에서는 CustomerType이 사용되었으므로 이 칼럼에는 "CustomerType"이 들어가게 된다. 그리고, CustomerType테이블에는 Doc_Serial에 Doc테이블의 키 값(H001)을 저장해 둠으로써, 조인 연산에서 사용할 수 있도록 되어 있다.

그림의 오른쪽 경우에는 Doc, PersonType, CustomerType, EmployeeType 네 개의 테이블이 생성되지만 실제 사용되는 테이블은 PersonType, CustomerType, EmployeeType와 같이 3개이다. 첫번째의 경우와는 달리 PersonType의 테이블에 xsi:type을 추가하고, 인스탄스의 값들을 저장하고 있다.

만약, PersonType이 추상 타입이라면 상황은 조금 달라진다. 즉 우리는 추상타입을 위한 테이블을 만들지 않기 때문에, xsi:type이라는 칼럼을 갖고 있어야 하는 새로운 테이블을 만들게 되며, 모든 추상 타입들은 이 테이블을 함께 사용한다. 테이블의 구조는 다음과 같다.

Abstract (ID, ParentTable, Abstract_type_name, xsi:type)

3.6.5 테이블의 생성

위에서 설명한 알고리즘에 의해서 관계형 스키마 생성기는 그림 5의 스키마 그래프로부터 아래와 같은 관계형 테이블들을 생성하게 된다.

Book ( BookID, title, sectiontitle, sectiontitle.lang, year)

SectionType ( SectionTypeID, sectiontitle, sectiontitle.lang, parentID, parentTable)

AuthorType ( AuthorTypeID, author.name, author.email, author.phone.home,

author.phone.office, parentID, parentTable)

SpecialAuthorType(SpecialAuthorTypeID, author.name, author.email, nickname,

author.phone.home, author.phone.office, parented, parentTable)

parentTable은 타입 그래프에서 내포 관계 링크의 부모쪽에 위치한 테이블의 이름이며, 이 칼럼이 필요한 이유는 여러 타입 노드에서 이 타입 노드로 연결이 가능하기 때문에 어느 테이블과 조인을 해야 할지를 알아내기 위해서 이다. ParentID는 parentTable의 키 값에 해당한다.

한편, 필요한 경우에는 테이블에 element_name이라는 칼럼들을 추가할 필요가 있다. 즉, 하나의 타입에 내포되어 있는 여러 개의 엘리먼트들이 동일한 타입을 사용하고 있는 경우에 ParentTable과 ParentID만으로는 필요한 정보를 얻지 못하는 경우가 있다. 예를 들어 아래의 타입 그래프를 보자.

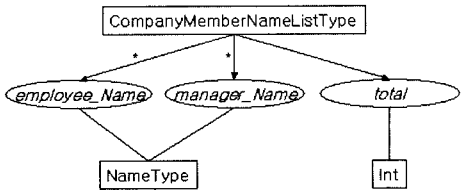



그림 8 CompanyMemberNameListType의 타입 그래프

위 스키마로부터 테이블을 생성하면 아래와 같은 테이블들이 만들어진다. 아래의 NameType테이블의 content_value에는 employee_name이나 manager_name엘리먼트의 텍스트 컨텐트가 저장된다.

CompanyMemberNameListType ( ID, total )

NameType ( NameTypeID, content_value, parentID, parentTable )

이 때, NameType테이블에서 employee_name만을 보고 싶을 때에는 현재의 테이블 구조에서는 불가능하다. 따라서, 위와 같은 경우를 위해서 element_name을 NameType에 추가하여 아래와 같은 테이블을 만들게 된다.

NameType ( NameTypeID, content_value, parentID, parentTable, element_Name )

3.6.6 휴리스틱을 이용한 기술들

지금까지 설명한 기본적인 XML Schema 인라인 기법은 기계적으로 작동하기 때문에 바람직하지 않은 결과를 만들어 내는 경우가 있다. 예를 들어, 테이블의 크기가 매우 커지는 것이 대표적인 예이다. MPEG7 그룹에서 작성한 MDS Schema로 관계형 테이블들을 만들어 본 결과 칼럼의 수가 500개 이상의 테이블들이 여러 개가 발견 되었다. 갯수 자체의 문제 뿐만 아니라, 동일한 칼럼 집합들이 반복해서 나타나는 경우도 나타났다. 또한, 비록 maxOccur가 1일지라도 서로 독립된 엔터티로 관리하는 것이 바람직한 경우에도 위 기법은 예외없이 하나의 관계형 테이블로 만들게 된다. 따라서 여기에서는 위와 같은 문제점들을 해결하기 위한 몇몇 휴리스틱 기법을 설명하고자 한다.

첫번째, 반 인라인 기법(counter inlining technique)으로 이름이 말하고 있듯이, 타입 그래프를 순회하면서 관계형 테이블을 만들어 나갈 때, 비록 엘리먼트의 maxOccur가 1일지라도 성능 향상의 목적상, 또는 관리의 목적상, 인라인 시키지 않는 것이 바람직하다고 볼 때에는 엘리먼트를 인라인 시키지 않고 독립된 테이블로 만드는 것이다. 대표적인 예가 <choice>로 묶여지는 엘리먼트들이다. XML Schema에서 choice는 여러 개의 엘리먼트 중에서 단 하나만을 선택해서 사용해야 하는 경우에 사용되는 태그이다. 만약 <choice> 태그에 의하여 묶이는 엘리먼트들의 구조가 간단한 경우에는 이들을 모두 인라인 시킬 수 있지만, 만약 구조가 매우 복잡하고 서브 엘리먼트들의 깊이가 매우 깊은 경우에는 이 엘리먼트들을 모두 인라인 시키게 되면 매우 많은 칼럼을 가진 테이블이 만들어지게 된다. 따라서 이러한 경우에는 각각에 대해서 독립된 테이블로 관리하는 것이 바람직하다.

둘째는, DTD의 경우 데이타의 발생 회수(cardinality)가 0또는 1, 0이상과 같이 매우 한정적이지만, XML Schema의 경우에는 구체적으로 몇 번이 발생하는지를 명시할 수 있다. 즉, 엘리먼트의 애트리뷰트 maxOccurs에 값을 할당하면 된다. 이 값을 이용해서 기준 값보다 작은 경우에만 인라인 시키고 기준 값보다 클 경우에는 독립적인 테이블로 만드는 방법이 있다.

### 3.7 테이블 참조 매뉴얼

시스템의 구현과 함께, 시스템에서 생성된 테이블들의 구조와 테이블들간의 관계를 쉽게 파악할 수 있도록 해주는 도구가 필요한데, 이러한 목적을 위하여 사용되어질 수 있는 테이블 참조 매뉴얼도 자동으로 생성된다. 테이블 참조 매뉴얼은 브라우징이 가능한 형태의 웹 문서로 되어 있다. 그림 9는 웹 브라우저를 통해서 본 테이블의 구조를 보여주고 있다. 웹 문서는 3개의 프레임으로 구성되어 있는데, 상단의 프레임에는 시스템을 통해서 생성된 모든 테이블들이 나타나 있다. 오른쪽 하단의 프레임에는 사용자에 의하여 선택이 된 테이블의 칼럼들이 나타나 있는데, 칼럼의 이름은 기본적으로 col이라는 영문자에 일련번호를 붙여서 생성하였고 그 칼럼이 나타내는 XML문서의 엘리먼트를 경로로서 나타내고 있다. 왼쪽 하단에는 선택된 테이블의 주키(primary key)를 외래키로 갖고 있는 테이블들의 목록이 나타나 있다.

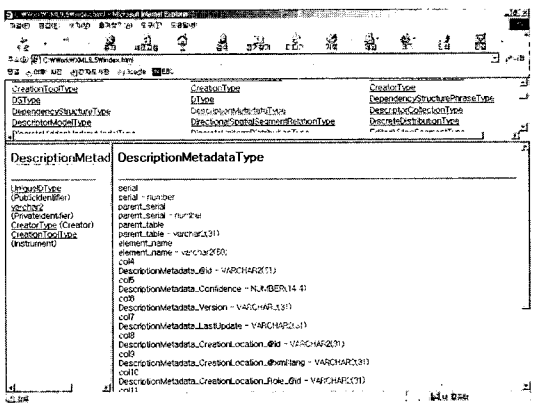



그림 9 테이블 참조 매뉴얼

## 4. 실 험

여기에서는 [5]에서 제시한 이진(Binary) 방식과 본 논문에서 제시한 XML Schema 인라인 기법에 대해서 성능 비교를 하고자 한다. 이진 방식의 경우에는 중간 노드들을 모두 저장하기 때문에, 검색 성능이 나빠지게 된다. 즉, 값이 들어 있는 단말 노드에 이르기 위해서는 인라인 기법의 경우에는 인라인 되지 않고 독립된 테이블의 개수만큼 조인을 하면 되며, 특히 단말 노드가 루트가 속해 있는 테이블에 인라인 되어 있는 경우라면, 조인을 사용할 필요가 없이 직접 가져올 수 있다. 그러나 이진 기법 경우에는 루트에서 단말노드에 이르는 길이만큼 조인을 해야 한다. 이러한 현상은 DTD에 대한 이진방식과 인라인 방식의 성능을 비교한 논문[5]에 잘 나타나 있다. 예를 들어, "2. 관련 연구"에서 사용한 예를 살펴보면 특정한 이름의 작가의 사무실 전화 번호를 알기 위해서는 최소한 Author, Name, Phone, Office 4개의 테이블을 조인하여야 한다. 반면, XML Schema 인라인의 경우에는 Author 테이블로 충분하며 작가에 관한 정보를 얻기 위해서는 별도의 조인이 필요 없다. 한편, 테이블 크기 측면에서도 중간 노드를 모두 저장해야 하는 이진 방식은 불가피하게 인라인 방식보다 테이블이 커지게 된다. 이와 같은 테이블 크기의 차이와 성능의 차이는 논문 [3][5]에서도 나타나 있다.

여기에서는 XML Schema 인라인 기법의 성능을 테스트하기 위해서, 실험 데이타로는 Mpeg7관련 표준을 만들고 있는 Multimedia Description Scheme Group에서 작성한 MDS Schema의 DescirptionMetadataType를 조금 변형한 것을 사용하였다. MDS Schema는 XML Schema로 정의된 현존하는 가장 복잡한 스키마 중의 하나이며, 성능 평가 대상으로 선택한 DescriptionMetadataType은 본 논문을 통해 설명된 XML Schema의 특성들을, 즉 다양한 데이타 타입, 추상 타입, 상속, 계층구조를 모두 포함하고 있다. 한편, 실험에 사용된 객체의 수는 90,000개이다. 아래에는 실험에 사용된 XML Schema의 스키마 그래프가 나타나 있다.

실험에 사용된 시스템 사양은 Pentium3로서 500MHz의 속도를 갖고 있으며, 메모리 크기는 128MB이다. 운영체제는 윈도우 NT이고, 관계형 데이타베이스는 오라클 DBMS로서 버전은 8.0.5이며, 사용된 프로그래밍 언어는 자바이고 관계형 데이타베이스로의 접근을 위해서 JDBC를 사용하였다.

먼저, 데이타베이스의 크기를 비교해 보면, 앞에서 설명한 바와 같이 이진 기법의 경우에는 계층 구조의 중간 간선 정보를 모두 저장하기 때문에 원래의 XML 문서의 크기보다 더 커지게 되었음을 알 수 있다. 반면, XML Schema 인라인 기법을 사용할 경우에는 원래의 XML문서, 혹은 이진 방식의 경우 보다 데이타베이스의 크기가 작아 졌음을 확인할 수 있다.

표 3 XML 문서와 테이블의 크기

| XML문서 | 이진 방식 | XML Schema 인라인 기법 |
|---|---|---|
| 106MB | 118MB | 50MB |

그 다음으로는 데이타베이스에 저장된 데이타에 대해서 SQL질의를 실행시켜서 결과를 얻기 까지 소요된 시간을 측정하는 실험을 하였다. 실험에 사용된 네 개의 SQL질의는 다음과 같다.

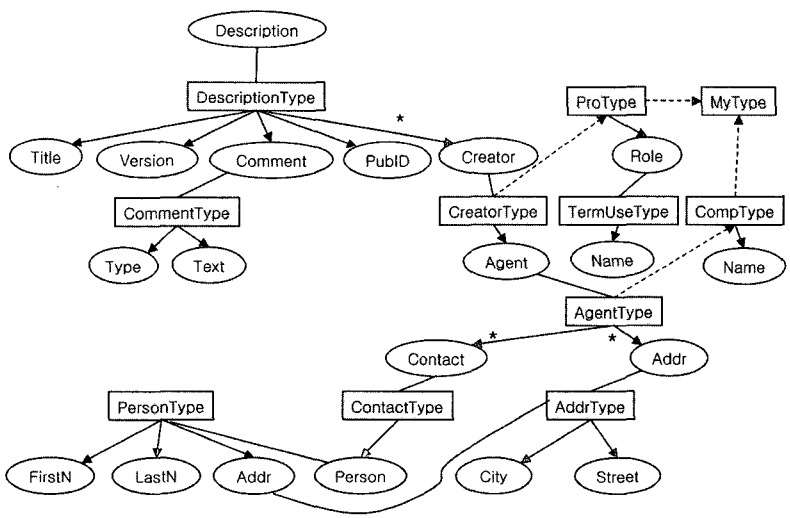



그림 10 실험에 사용된 Schema Graph

Q1. Object id를 주고서 그 객체의 모든 정보를 읽어 낸다.
Q2. Version의 값을 주고서 title을 읽어 낸다.
Q3. PubID를 주고서 Comment의 Type과 Text를 읽어 낸다.
Q4. Contact Person의 First Name을 주고서 Creator의 Role name을 읽어 낸다.

표 4 성능 비교(second)

| 질 의 | 이진 방식 | XML Schema인라인 |
|---|---|---|
| Q1 | 1.11 | 0.25 |
| Q2 | 0.968 | 4.016 |
| Q3 | 5.120 | 4.016 |
| Q4 | 67.578 | 4.297 |

실험 결과를 정리하면, 검색에 필요로 되어지는 엘리먼트의 개수가 적으면 이진방식의 경우에 보다 좋은 성능을 보여주고 있다. Q2의 경우에는 nesting의 깊이가 깊지 않고, where절에 포함되는 엘리먼트의 수가 2개에 불과하기 때문에, 비록 조인 연산이 필요하더라도 XML Schema 인라인 방법보다 빠른 성능을 보이게 된다. XML Schema 인라인의 경우, 테이블에 여러 엘리먼트들이 인라인 되어 있어서 조인 연산이 불필요하지만, 그 대신 큰 테이블 전체를 테이블 스캔 연산자를 이용해서 처리해야 하므로, 이진 방식보다 오히려 나쁜 성능을 보이고 있다. 그러나, 질의에 관련된 엘리먼트의 개수가 많아 질 수록, 성능 차이는 매우 커짐을 알 수 있다. 예를 들어, Q4의 경우에는 7개의 엘리먼트들이 질의에 필요하게 되며 여러 번의 조인 연산이 필요해지게 되므로 XML Schema 인라인 방식에 비하여 급격히 나빠진 성능을 보이게 된다.

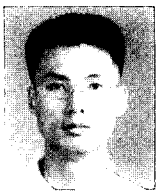
한편, 키 값을 이용해서 전체 XML문서의 데이타 값만을 찾아내는 Q2에서도 테이블 수가 적은 인라인의 경우에 성능이 좋았음을 알 수 있었다.

## 5. 결 론

본 논문에서는 첫째, XML Schema를 소개하면서 관계형 스키마와의 차이점과 이로 인해 발생하는 인라인 기법상의 문제점들을 제시하였다. 둘째, XML Schema로부터 관계형 테이블들을 생성하기 위해 필요한 정보들로 구성된 스키마 그래프와 타입 그래프를 제시하였으며, 이 그래프들을 순회하면서 관계형 테이블들을 생성해 낼 수 있는 알고리즘을 소개하였고, 기본적인 방법 이외에도 시스템의 성능을 향상시킬 수 있는 휴리스틱 기법들을 제시하였다. 셋째는 제시된 데이타 구조와 알고리즘을 시스템으로 실제 구현하였으며, XML문서를 관계형 데이타베이스의 저장하는 대표적인 방법인 이진 기법과의 성능 비교 실험을 통해서 본 논문에서 구현한 시스템의 성능을 입증하였다.

## 참 고 문 헌

[1] Jose M. Martinez, "Introduction to MPEG-7, version 2," ISO/IEC JTC1/SC29/WG11 N3751, 2000.
[2] XML Schema Part 0: Primer, http://www.w3.org/TR/2001/REC-xmlscheme-0-20010502, 2001.
[3] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, Jeffrey Naughton, "Relational Databases for Querying XML Documents : Limitations and Opportunities," VLDB 99, pp. 302-314, 1999.
[4] A. Deutsch, M. Fernandez, D. Suciu, "Storing semistructured data with STORED," SIGMOD 99, pp. 431-442, 1999.
[5] Danielo Florescu, Donald Kossmann, "Storing and Querying XML Data Using an RDBMS," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol. 22, pp. 27-34, 1999.
[6] Albrecht Schmidt, Martin Kersten, Menzo Windhouwer, Florian Waas, "Efficient Relational Storage and Retrieval of XML Documents," WebDB 00, pp. 47-52, 2000.
[7] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, "Lore: A database management system for semistructured data," ACM SIGMOD Record, Vol. 26, No. 3, pp. 54-66, 1997.
[8] Serge Abiteboul, Peter Buneman, Dan Suciu, "Data on the Web," Morgan Kaufmann, pp. 172-176, 2000.
[9] Gerti Kappel, et Al, "X-Ray - Towards Integrating XML and Relational Database Systems," ER 00, pp. 339-353, 2000.
[10] T.Bray, J.Paoli, C.M.Sperberg-McQueen, "Extensible Markup Language(XML) 1.0," http://www.w3.org/TR/REC-xml.
[11] J. Bosak, T.Bray, D.Connolly, E.Maler, G.Nicol, C.M.Sperberg-McQueen, L.Wood, J.Clark, "W3C XML Specification DTD," http://www.w3.org/XML/1998/xmlspec-report-19980910.htm.
[12] Microsoft Corporation, XML Schema, http://www.microsoft.com/xml/schema/reference/star.asp.
[13] C. Kanne, G. Moerkotte, "Efficient Storage of XML data," ICDE 00, p. 198, 2000.
[14] Dongwon Lee, Wesley W. Chu, "Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema," ER 00, pp. 323-338, 2000.



김 정 섭

1996년 2월 서울대학교 경영학과(학사) 2002년 2월 한국과학기술원 전산학과(석사). 현재 LG전자 디지털미디어 연구소 선임연구원. 관심분야는 XML, GIS

박 창 원

1995년 2월 서강대학교 전자계산학과(학사). 1997년 2월 한국과학기술원 전산학과(석사). 2002년 8월 한국과학기술원 전자전산학과 전산학전공(박사). 현재 LG전자기술원 정보기술연구소 선임연구원 관심분야는 XML, XQuery, XSLT, GML, LBS, GIS, Databases

정 진 완

정보과학회논문지 : 데이타베이스
제 31 권 제 1 호 참조