XML DTD의 JDBC 기반 SQL 스키마로의 변환

Transforming XML DTD to SQL Schema based on JDBC

이 상 태* 주 경 수**
Sang-Tae Lee Kyung-Soo Joo

요 약

B2B 전자상거래와 같이 XML을 이용한 정보교환이 보편화되고 있으며, 이에 XML 메시지의 데이터베이스로의 저장을 위한 효율적인 방안이 요구되고 있다. 한편 Oracle8i와 9i 및 Informix 그리고 SQL2000 서버 등과 같이 멀티미디어 응용 등을 위하여 기존의 관계형 DBMS들은 객체-관계형 DBMS로 확장되고 있으며, 이에 따라 관계형 데이터베이스 표준안인 SQL2도 ORDB인 SQL3로 확대 개편되고 있다. 아울러 J2EE와 같이 JAVA를 기반으로 한 XML 응용이 확대됨에 따라 JDBC를 통한 XML 응용과 데이터베이스의 효율적인 연계방안이 요망된다.

본 논문에서는 XML DTD를 토대로 하여 SQL3 스키마로의 변환을 위한 방법을 제시한다. 이를 위하여 먼저 XML DTD를 UML 클래스 다이어그램인 객체모델로 변환시키기 위한 방안을 제안하였고, 변환된 객체모델을 SQL3 스키마로 모델링하기 위한 방법을 제시하였다.

본 논문에서 제안한 XML DTD를 토대로 한 SQL3 스키마로의 변환 방법은 JAVA를 기반으로 Oracle8i와 9i 및 Informix 그리고 SQL2000 서버 등과 같이 객체-관계형 데이터베이스를 토대로 XML 응용을 구축하기 위한 데이터베이스 설계 방안으로 활용될 수 있다.

Abstract

The information exchange on the using of XML such as B2B electronic commerce is common. So the efficient method to store XML message in database is needed. Because the RDBMS is extended to ORDBMS for supporting multimedia application such as Oracle8i, 9i, Informix and SQL2000 server, SQL2, the standard RDB is extended to SQL3 for ORDB. And the XML application based on java such as J2EE is extended. Therefor it is necessary for the efficient connection methods based on JDBC between XML application and database system.

In this paper, the methodology a transformation XML DTD to SQL3 schema is proposed. For the transformation, first the methods of transformation XML DTD to object model in UML class diagram are proposed. And then the methods of mapping transferred object models to SQL3 schema are proposed.

This approach for transformation XML DTD to SQL3 schema such as Oracle8i, 9i, Informix and SQL2000 server based on java is proposed in this paper, can be used in database design to build XML applications based on ORDB.

1. 서 론

XML은 웹에서 구조화된 정보나 반-구조화된 정보를 교환하기 위한 표준 마크업 언어로 채택되어 가고 있으며, B2B 전자상거래와 같이 XML을 이용한 정보교환이 보편화되고 있다[22,24]. 이에 XML 메시지의 데이터베이스로의 저장을 위한 효율적인 방안이 요구되고 있다. 한편 Oracle8i와 9i 및 Informix 그리고 SQL2000 서버 등과 같이 멀티미디어 응용 등을 위하여 기존의 관계형 DBMS들은 객체-관계형 DBMS로 확장되고 있으며, 이에 따라 관계형 데이터베이스 표준 안인 SQL2도 SQL3로 확대 개편되고 있다[21]. 아울러 J2EE와 같이 JAVA를 기반으로 한 XML 응용이 확대됨에

* 정 회 원 : 신성대학 컴퓨터응용계열 교수
stlee@shinsung.ac.kr
** 종신회원 : 순천향대학교 정보기술공학부 교수
gsoojoo@sch.ac.kr

따라 JDBC를 통한 XML 응용과 데이터베이스의 효율적인 연계방안이 요망된다.

본 논문에서는 XML DTD를 토대로 하여 SQL3 스키마로의 변환을 위한 방법을 제시하고자 하며, 먼저 XML DTD를 UML 클래스 다이어그램인 객체모델로 변환시키기 위한 방안을 제안하고, 변환된 객체모델을 SQL3 스키마로 모델링하기 위한 방법을 제시하고자 한다.

본 논문의 제2절에서는 기존의 데이터 변환 기술에 관련된 연구들에 대하여 설명하고, 제3절에서는 제안된 방법에 따라 XML DTD에서 객체모델로 변환하는 방법과 변환된 객체모델을 JDBC 기반의 SQL 스키마로 변환하는 방법을 다루며, 마지막으로 결론을 기술한다.

2. 관련 연구

XML은 1998년 2월에 인터넷 상에서 구조화된 문서를 표현하기 위한 W3C에 의해 발표되었다[5]. 또한 XML은 SGML에서 파생된 것으로써 SGML의 사용하기 어렵고 복잡한 기능을 축소하고 인터넷에서 사용하기 용이하도록 만든 언어이며 HTML과는 달리 사용자가 임의의 태그를 정의하여 사용할 수 있으며, 문서의 구조 정보를 표현할 수 있는 특징을 가진다. XML의 이러한 특징 때문에 다양한 분야에서 XML을 이용한 연구가 이루어지고 여러 시스템들이 개발되고 있다.

많은 논문들이 복잡한 XML 파일을 관계형 데이터베이스에 저장하는데 초점을 맞추어 데이터베이스와 XML 문서의 연결에 대하여 발표되었다[19]. 발표된 논문들의 내용을 분류하면 주어진 XML 파일을 관계형 데이터베이스에 어떻게 저장할 것인가 하는 제안들을 하고 있으며, 다른 한편으로는 주어진 XML 파일이나 DTD로부터 정보의 의미에 대하여 데이터베이스 제약사항들을 다루는 방법을 제안한다[6,11-13].

또한 XML 문서가 객체-관계형 시스템으로 구조화된 데이터 타입을 저장하는 방법이 제안되고 있다[1,4,12]. 다른 한편으로는 객체-관계 데이터베이스로부터 XML로 변환하는 내용을 다루고 있으며, 그 내용은 어떻게 XML 파일을 객체-관계형 데이터베이스에 저장하여 빠르게 질의할 것인가를 다루고 있다[3,15]. 또한 객체-관계형 데이터베이스 시스템은 XML 문서들을 서로 연결하기 위한 것이며, XML의 네스티드 문서들을 객체-관계형 모델로 변환하는 내용을 다루고 있다.

객체지향 데이터베이스 시스템의 풍부한 데이터 모델링 때문에 전자문서의 저장이 잘 이루어진다[2,13]. 그러나 객체지향 데이터베이스 시스템은 효과적인 방법으로 대량의 데이터를 다루기에는 적합하지 않다[26].

이러한 시스템들이 증가함에 따라서 생성되는 XML 문서도 증가하고 있으며 XML 문서를 저장 관리하는 시스템의 중요성도 증가하고 있다. XML 문서를 저장하고 검색하는 시스템에 대한 연구는 효율적으로 XML 문서를 저장하고 검색할 수 있는 기능에 중점을 둔 경우가 많다. 또한 XML 문서를 저장하기 위해서 특별히 설계된 테이블이나 저장 구조를 가지고 있어서 XML 문서의 내용뿐만 아니라 문서의 구조 정보 등도 같이 저장할 수 있다. 그리고 이를 바탕으로 XML 문서의 내용 검색이나 구조 검색 등을 효율적으로 할 수 있도록 하고 있다. 그러나 XML 저장 시스템을 이용하여 새로이 시스템을 구축하는 경우가 아니라 기존에 저장되어 있는 데이터베이스를 그대로 활용하면서 XML 시스템을 구축하고자 하는 경우에는 데이터 변환 등의 많은 작업이 행해져야 한다[14]. XML 전용 저장 시스템에서는 XML 문서를 저장하기 위해서 특별히 설계된 테이블을 가지고 있지만, 기존의 데이터베이스에서는 이런 테이블이 존재하지 않고, 현존하는 테이블의 구조 또한 다르기 때문이다.

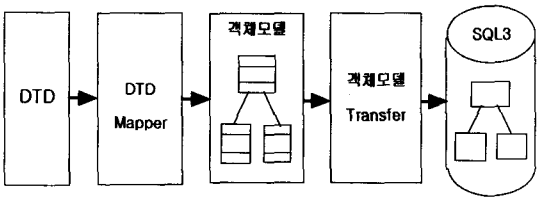
따라서 XML 응용에서 메시지들을 데이터베이스에 저장하고자 할 때 XML DTD를 데이터베이스 스키마로 변환하여야 하는데, XML DTD를 관계형 데이터베이스 스키마로 변환하는 연구는

XML-DBMS 등과 같이 많이 이루어져 있다[20,23]. 그러나 객체-관계형 데이터베이스 스키마로 변환하는 연구는 미미한 실정이다. 따라서 본 논문에서는 XML DTD를 객체-관계형 데이터베이스의 표준으로 되어가고 있는 SQL3 스키마로 변환하기 위하여 먼저 XML DTD를 객체모델로 변환하고, 변환된 객체모델을 SQL3 스키마로 변환한다. 제안된 변환 방법은 다른 시스템들이 구조의 변환에만 초점을 맞추고 있는데 반하여 내용까지도 SQL3 스키마로 변환이 가능하도록 하고 있다.

3. XML DTD의 SQL3 스키마로의 변환 방법

XML 응용에서 메시지들을 데이터베이스에 저장하고자 할 때 DTD를 데이터베이스 스키마로 변환하여야 하는데, 단순한 XML 문서는 직접 데이터베이스로 변환이 가능하나, 복잡한 XML 문서는 객체 기반 변환 방법으로 처리되어 효과적인 변환이 이루어진다[10,25].

따라서 그림 1에서와 같이 본 논문에서는 XML DTD를 객체모델로 변환하고, 변환된 객체모델을 SQL3 스키마로 변환한다.



(그림 1) DTD의 SQL3 스키마로의 변환

3.1 XML DTD의 객체 변환 방법

XML은 웹에서 구조화된 정보나 반-구조화된 정보를 교환하기 위한 표준 마크업 언어로 채택되어 가고 있으며, 시작 태그와 끝 태그가 컨테이너로 사용되고, 시작 태그의 내용과 끝 태그가 함께 하나의 요소(element)를 이룬다[8]. 요소는 하나의 루트 요소만 가져야 하며, 다른 태그는 모두 요소 안에 확실하게 중첩되어야 한다[9]. 이것은 한 요소가 다른 요소들을 포함할 경우, 그 요소들은 한 요소 안에 들어가야 한다는 뜻이다. 다음과 같이 요소 및 특성, 복합 내용 및 혼합 내용 모델 변환에 따라 XML DTD가 객체모델로 변환되는 것을 제안한다.

3.1.1 요소(element)

본 논문에서는 요소 타입을 두 개의 타입으로 분류하는데 PCDATA만 가진 요소의 타입을 단순 요소 타입이라고 하며, PCDATA을 제외한 모든 요소의 타입을 복합 요소 타입이라고 한다. 단순 요소 타입과 복합 요소 타입의 DTD가 객체모델로 변환되는 방법을 설명한다.

```
<!ELEMENT ProductCatalog (ProductName, DefaultLanguage,
                          DefaultCurrency)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
```

(그림 2) 단순 요소 타입

그림 2를 클래스 속성으로 변환하면 그림 3과 같다.

ProductCatalog
ProductName DefaultLanguage DefaultCurrency

(그림 3) 단순 요소 타입을 객체로 변환

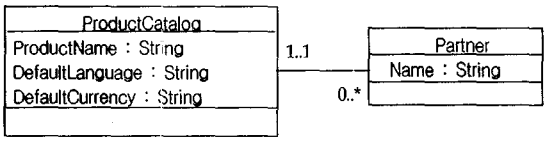
그림 3에서 보면 클래스 이름은 그림 2에 있는 요소가 포함하고 있는 부모 요소가 클래스 이름으로 지정되고, 데이터형은 작성자가 요소의 의미에 따라 정수형, 문자형 등으로 지정한다.

본 논문에서는 복합 요소 타입을 단순 요소 타입을 제외한 모든 요소를 말하는데 그림 4와 같다.

```
<!ELEMENT ProductCatalog (ProductName, DefaultLanguage,
 DefaultCurrency, Partner*)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Partner (Name)>
```

(그림 4) 복합 요소 타입

그림 4에서는 루트 요소가 ProductCatalog이고, 자식 요소는 ProductName, DefaultLanguage, DefaultCurrency 이다. 따라서 객체로 변환하면 그림 5와 같다.



(그림 5) 복합 요소 타입의 객체로 변환

그림 5는 객체로 변환된 클래스이다. 클래스 이름은 ProductCatalog이다. 여기서 데이터형은 그림 2에서 요소의 의미가 문자형으로 사용되므로 객체로 변환될 때 String형으로 변환된다.

3.1.2 특성(attribute)

모든 요소는 특성을 가질 수 있다. 특성은 이름 또는 값의 형태를 가진다. 특성은 요소에 포함되며, 특성에 부여된 값을 통해서 그 요소에 어떠한 특징을 제공하게 된다[3,18].

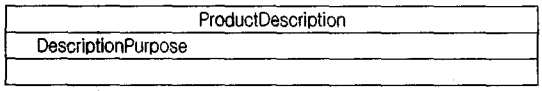
본 논문에서 특성은 XML 파서가 애플리케이션에게 보내는 값들을 뜻하는데 요소의 내용과는 구별되는 값이므로 요소와 마찬가지고 특성도 단일 값을 가진 특성과 다중 값을 가진 특성으로 분류된다.

단일 값을 가진 특성은 문자열 특성(CDATA), 토큰 특성(ID, IDREF, NMTOKEN, ENTITY, NOTATION), 열거형 특성이며, 객체로 변환될 때 클래스의 속성으로 변환된다.

그림 6에서 보면 클래스 이름이 ProductDescription이고 클래스의 속성은 DescriptionPurpose이다.

```
<!ELEMENT ProductDescription>
<!ATTLIST ProductDescription5 DescriptionPurpose CDATA
 #IMPLIED>
```

(그림 6) 단일 값을 가진 특성



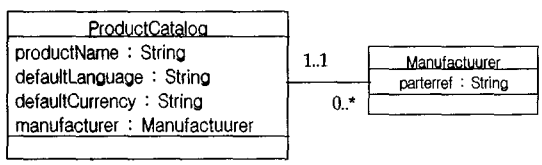
(그림 7) 단일 값을 가진 특성을 객체로 변환

다중 값을 가진 특성은 IDREFS, NMTOKENS, ENTITIES으로 선언된 것이다. 이것들을 객체로 변환할 때 값이 하나 이상으로 들어가기 때문에 별도의 객체로 만들 수 있다. 그림 8은 특성을 포함한 DTD를 보여준다.

```
<!ELEMENT ProductCatalog (ProductName, DefaultLanguage,
 DefaultCurrency, Manufacturer)>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Manufacturer >
<!ATTLIST Manufacturer PartnerRef IDREFS #IMPLIED>
```

(그림 8) 다중 값을 가진 특성

그림 8에서 보면 요소의 특성인 PartnerRef은 IDREFS로 선언되어 있다. IDREF는 단일 값을 가지고 있는 토큰 특성 ID와 의미는 같으나 ID는 요소에 대한 ID로 같은 문서에서 같은 ID 특성 값을 가진 두 개의 요소를 가질 수 없으며, IDREF는 ID를 가리키는 포인터이고, IDREFS는 하나 이상의 IDREF값으로 이루어진다. 따라서 객체로 만들 수 있다.



(그림 9) 복합 요소 타입의 객체로 변환

그림 9에서는 두 개의 클래스로 이루어지며, 한 클래스는 부모 클래스이고 다른 하나는 자식

클래스로 구분된다. 그림 8에서 부모 요소는 ProductCatalog이고 자식 요소는 Manufacturer이므로 객체로 변환하면 그림 9와 같다. 따라서 두 개의 클래스를 연결하려면 참조형으로 연결된다.

3.1.3 복합 내용 모델 변환

본 논문에서는 복합 내용 모델 변환을 위하여 아래와 같이 분류하고 있다.

(1) 순차

DTD에 있는 요소와 요소에 속하는 자식 요소들을 순차적으로 객체로 변환하면, 자식 클래스와 부모 클래스간의 연결을 참조형으로 변환할 수 있다. 테이블 스키마로 변환할 때, 그 테이블이 속하는 컬럼들은 반드시 NOT NULL 제약조건이 따른다.

(2) 선택

DTD 요소 안에 자식 요소들이 있는데 그 중에 하나만 선택할 경우에도 객체로 변환될 수 있고, 변환된 객체는 다시 객체-관계형 데이터베이스 스키마 객체 모델로 변환된다.

(3) 반복

요소 안에 자식 요소가 중복된 경우 객체화로 변환할 때 요소 안에 중복된 같은 요소들이 클래스 속성으로 변환하는 경우 배열 형태로 변환된다. 이렇게 변환된 클래스 속성은 별도의 테이블로 변환된다. 하나 이상의 요소들이 존재한다는 의미로 클래스 속성의 데이터형을 String[]으로 변환한다. 즉 작성자가 알 수 없는 만큼 요소가 존재한다는 의미로 별도의 테이블로 변환된다.

(4) 서브그룹

<!ELEMENT A (B, (C | D))>와 같은 경우를 말하며, 이 요소는 <!ELEMENT A (B, C))>와 <!ELEMENT A (B, D))> 요소로 분류된다. 따라서 두 개의 객체로 변환할 수 있다.

3.1.4 혼합 내용 모델 변환

텍스트나 요소, 그 둘을 모두 포함할 수 있는 요소들을 혼합 내용 모델이라고 하며, XML 프로세서는 공백, 탭 등의 PCDATA와 요소 내용간의 구분이 어렵다. 왜냐하면 끝 태그와 다음의 시작 태그 사이에 공백이 있으면 불분명하게 된다. 본 논문에서는 혼합 내용 모델에서 선택이 가능한 그룹은 하나이고, #PCDATA로 시작하며, 그 뒤에는 혼합 내용의 연산자수를 나타내는 타입 순서로 되며, 각각은 한번만 선언된다[23]. #PCDATA만 유일한 옵션이며, "*"는 반드시 괄호를 닫은 바로 뒤에 와야 한다. 다음은 혼합 내용 모델의 DTD 예를 보여준다.

```
<!ELEMENT Pick (#PCDATA | eeney | meeney | miney | mo)* )>
<!ELEMENT eeney (#PCDATA)>
<!ELEMENT meeney (#PCDATA)>
<!ELEMENT miney (#PCDATA)>
<!ELEMENT mo (#PCDATA)>
```

(그림 10) 혼합 내용 모델 DTD

그림 10은 혼합 내용 모델의 DTD이다. Pick 요소는 루트 요소이고 자식 요소들은 반복적으로 이루어진다. 따라서 루트 요소는 클래스 이름으로 변환하고 자식 요소는 클래스의 속성으로 변환이 된다. 그리고 반복의 의미가 있어 데이터형은 배열로 선언된다. 이 DTD를 객체로 변환하면 그림 11과 같다.

DTD에서 "*" 연산자는 요소나 요소 그룹이 생략되거나 한 번 이상 나타날 수 있으므로 설계하는 작성자는 몇 개가 나타나는지 알 수 없기 때문에 String[] 형으로 선언한다.

Pick
pcdata : String[ ] eeney : String[ ] meeney: String[ ] miney : String[ ] mo : String[ ]

(그림 11) 혼합 내용 모델 DTD를 객체화로 변환

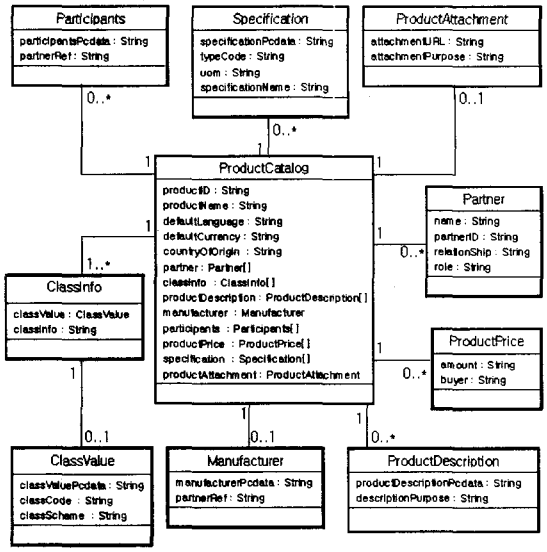
```
<?xml version="1.0" encoding="EUC-KR"?>
<!ELEMENT ProductCatalog (ProductName, DefaultLanguage,
                  DefaultCurrency, Partner*, ClassInfo+,
              ProductDescription*, CountryOfOrigin?,
                Manufacturer?, Participants*, ProductPrice*,
                  Specification*, ProductAttachment?)>
<!ATTLIST ProductCatalog  ProductID ID #REQUIRED>
<!ELEMENT ProductName (#PCDATA)>
<!ELEMENT DefaultLanguage (#PCDATA)>
<!ELEMENT DefaultCurrency (#PCDATA)>
<!ELEMENT Partner (Name)>
<!ATTLIST Partner
        PartnerID ID #IMPLIED
        RelationShip (Seller | Manufacturer | Intermediary)
                                              #IMPLIED
        Role CDATA #IMPLIED >
<!ELEMENT Name (#PCDATA)>
<!ELEMENT ClassInfo (ClassValue, ClassInfo?)>
<!ELEMENT ClassValue (#PCDATA)>
<!ATTLIST ClassValue
              ClassCode CDATA #IMPLIED
              ClassScheme CDATA #IMPLIED >
<!ELEMENT ProductDescription (#PCDATA)>
<!ATTLIST ProductDescription
           DescriptionPurpose CDATA #IMPLIED >
<!ELEMENT CountryOfOrigin (#PCDATA)>
<!ELEMENT Manufacturer (#PCDATA)>
<!ATTLIST Manufacturer
              PartnerRef IDREF #IMPLIED >
<!ELEMENT Participants (#PCDATA)>
<!ATTLIST Participants
              PartnerRef IDREF #IMPLIED >
<!ELEMENT ProductPrice (Amount, Buyer?)>
<!ELEMENT Amount (#PCDATA)>
<!ELEMENT Buyer (#PCDATA)>
<!ELEMENT Specification (#PCDATA)>
<!ATTLIST Specification
        TypeCode (Size | Weight | Ingredient |
              Color | Shape | Packing | Performance
                | Content | Others) #IMPLIED
        UOM CDATA #IMPLIED
        SpecificationName CDATA #REQUIRED >
<!ELEMENT ProductAttachment (AttachmentURL,
                        AttachmentPurpose?)>
<!ELEMENT AttachmentURL (#PCDATA)>
<!ELEMENT AttachmentPurpose (#PCDATA)>
```

(그림 12) XML DTD

앞에서 제안된 XML DTD의 객체 변환 방법에 따라 그림 12에 보여준 XML DTD는 그림 13과 같이 객체모델로 변환된다.

3.2 JDBC 기반의 SQL3 스키마로 변환 방법

J2EE와 같이 JAVA를 기반으로 한 XML 응용이



(그림 13) 객체모델

확대됨에 따라 JDBC를 통한 XML 응용과 데이터베이스의 효율적인 연계방안이 요망되는데, 먼저 XML DTD를 UML 클래스 다이어그램인 객체모델로 변환시키기 위한 방안을 제안하였으며, 본 절에서는 변환된 객체모델을 JDBC 기반의 SQL3 스키마로 변환하기 위한 방법을 제안한다.

3.2.1 변환 방법

다음은 XML DTD로부터 변환된 객체모델이 JDBC 기반의 SQL 스키마로 변환하는 과정이다.

① 타입, 테이블, 보조적인 객체를 생성하기 위해서는 객체-관계형 데이터베이스 시스템으로 확장되어 사용되며, 그러한 예로는 오라클8, Informix, SQL2000, DB2가 있다.
② 객체 클래스는 구조화된 객체 타입을 생성하며, 테이블과 연결하여 타입을 하나 또는 여러 개의 테이블에 객체의 원소가 되도록 한다.
③ 객체 타입은 사용자 정의 타입이거나, 속성을 가진 객체타입이다.
④ 클래스에서의 객체 속성은 객체 타입에서의 속성이다.

⑤ 타입이 테이블이나 객체 또는 특정 타입으로 변환할 때 클래스에서의 객체 속성 타입은 객체 타입에서의 속성 타입이다.
⑥ 객체 속성이 NULL 제약조건을 가지면 NOT NULL 제약조건도 가진다.
⑦ 객체 속성이 초기값을 가지면 컬럼에 DEFAULT 값을 더한다.
⑧ 독립적인 클래스나 함축적인 성질을 가진 클래들을 위해 기본키로 정수를 생성한다. {oid}를 위해 태그된 컬럼에 기본키 제약조건에 따라 {oid}를 더한다.
⑨ 부 클래스를 위해 기본키 제약조건과 외래키 제약조건에 따라 각각의 부모 클래스의 키를 더한다.
⑩ 클래스들의 결합을 위해 객체 타입을 생성하고 기본키 제약조건과 외래키 제약조건에 따른 역할 테이블로부터 기본키를 추가한다.
⑪ 교환 oid가 <n>인 경우 UNIQUE 제약조건에 따른 컬럼을 추가한다.
⑫ 각각의 확실한 제약조건을 위해 CHECK를 실시한다.
⑬ 결합하는데 있어서 각각의 0..1, 1..1 역할을 위한 참조 테이블에서 외래키 컬럼을 생성한다. 교대로 단일 객체들이나 컬럭션을 위한 그 자신 안에서 속성으로 객체를 선언하기 위하여 객체 타입에 참조를 사용하거나 또는 다중관계 객체를 위한 참조의 배열로 객체를 선언하기 위하여 객체 타입에 참조를 사용한다.
⑭ 집합 테이블에 외래키를 가진 다중 집합을 위해 기본키를 생성하고, 기본키를 위한 컬럼을 추가하며, 테이블 안에서 그 집합을 저장하기 위한 객체 타입을 사용한다. 또한 단일 객체를 위한 객체 속성 또는 컬렉션을 통하여 배열이나 네스티드 테이블을 사용한다.
⑮ 너무 많이 이동하게 되는 이진 결합 클래스들을 최적화 한다.
⑯ 외래키를 사용하여 결합이 안된 클래스와 함께 다대다 결합의 관계 테이블을 생성한다.
⑰ 다대다 결합에서 역할 테이블의 키로부터 기본키, 외래키의 제약조건을 생성한다.
⑱ 객체 클래스의 전달 작용을 위한 객체 타입에 메소드를 생성한다.

3.2.2 SQL3 데이터 타입 변환

SQL2에서는 데이터 타입 지원이 빈약하여 객체모델로부터 데이터베이스 스키마로의 변환이 어려웠으나, SQL3에서는 BLOB, CLOB, ARRAY, REF, STRUCT을 확장하여 사용함으로써 XML 메시지의 데이터베이스로의 저장을 위해 원활하게 변환할 수 있도록 지원하고 있다[7]. 표 1은 객체모델 데이터 타입이 SQL3 데이터 타입으로 변환되는 관계를 보여준다.

(표 1) 객체모델의 데이터 타입으로부터 SQL3 데이터 타입으로 변환

객체모델 데이터 타입	SQL3 Type
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
java.math.BigDecimal	NUMERIC
java.lang.String	VARCHAR or LONGVARCHAR
byte[]	BARBINARY or LONGBARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.sql.Blob	BLOB
java.sql.Clob	CLOB
java.sql.Array	ARRAY
java.sql.Ref	REF
java.sql.Struct	STRUCT

3.2.3 변환 사례

그림 12에서 보여준 XML DTD로부터 변환된 객체모델인 그림 13은 본 논문에서 제시하는 4.1 변환 과정의 내용에 따라 JDBC 기반의 SQL 스키마로 변환된다.

(1) Partner 객체는 ProductCatalog 테이블에서 위에

열거된 ⑭번의 성질에 따라 그림 15와 같이 JDBC 기반의 SQL 스키마로 변환되어 네스티드 테이블이 생성된다.

Partner
name : String partnerID : String relationShip : String role : String

(그림 14) Partner 객체

```
SQL> CREATE TYPE Partner_t AS OBJECT
(
  name            VARCHAR,
  partnerID       VARCHAR,
  relationShip    VARCHAR,
  role            VARCHAR
);
SQL> CREATE TYPE Partner_list AS TABLE
                          OF Partner_t;
```

(그림 15) Partner 테이블

(2) ClassInfo 객체는 ProductCatalog 테이블에서 부 클래스를 가지고 있으며, 그림 17과 같이 JDBC 기반의 SQL 스키마로 변환되어 네스티드 테이블이 생성된다.

ClassInfo
classValue : ClassValue classinfo : String

(그림 16) ClassInfo 객체

```
SQL> CREATE TYPE ClassInfo_t AS OBJECT
(
  classValue     ClassValue_t,
  classinfo      VARCHAR
);
SQL> CREATE TYPE ClassInfo_list AS
                      TABLE OF ClassInfo_t;
```

(그림 17) ClassInfo 테이블

그림 18은 부 클래스로서 ClassValue 객체가 생성되고 위에 설명된 ClassInfo 테이블의 속성이 되어 그림 19와 같이 ClassValue 타입이 된다.

ClassValue
classValuePcdata : String classCode : String classScheme : String

(그림 18) ClassValue 객체

```
SQL> CREATE TYPE ClassValue_t AS OBJECT
(
  classValuePcdata     VARCHAR,
  classCode            VARCHAR,
  classScheme          VARCHAR
);
SQL> CREATE TABLE ClassValue OF ClassValue_t;
```

(그림 19) ClassValue 타입

(3) ProductDescrition 객체는 ProductCatalog 테이블에서 위에 열거된 ⑭번의 성질에 따라 그림 21과 같이 JDBC 기반의 SQL 스키마로 변환되어 네스티드 테이블이 생성된다.

ProductDescrition
productDescritionPcdata : String descritionPurpose : String

(그림 20) ProductDescription 객체

```
SQL> CREATE TYPE ProductDescrition_t AS OBJECT
(
  productDescritionPcdata     VARCHAR,
  descritionPurpose           VARCHAR
);
SQL> CREATE TYPE ProductDescrition_list
        AS TABLE OF ProductDescrition_t;
```

(그림 21) ProductDescription 테이블

(4) Manufacturer 객체는 ProductCatalog 테이블에서 위에 열거된 ⑤번의 성질에 따라 테이블 속성에 포함되어 그림 23과 같이 변환된다.

Manufacture
manufacturePcdata : String partnerRef : String

(그림 22) Manufacture 객체

```
SQL> CREATE TYPE Manufacture_t AS  OBJECT
(
  manufacturePcdata     VARCHAR,
  partnerRef     VARCHAR
);
SQL> CREATE TABLE Manufacture OF Manufacture_t;
```

(그림 23) Manufacturer 테이블

(5) **Participants** 객체는 **ProductCatalog** 테이블에서 위에 열거된 ⑭번의 성질에 따라 그림 25와 같이 **JDBC** 기반의 **SQL** 스키마로 변환되어 네스티드 테이블이 생성된다.

Participants
participantsPcdata : String partnerRef : String

(그림 24) Participants 객체

```
SQL> CREATE TYPE Participants_t AS OBJECT
(
  participantsPcdata     VARCHAR,
  partnerRef     VARCHAR
);
SQL> CREATE TYPE Participants_list AS TABLE
                    OF Participants_t;
```

(그림 25) Participants 테이블

(6) **ProductPrice** 객체는 **ProductCatalog** 테이블에서 위에 열거된 ⑭번의 성질에 따라 그림 27과 같이 **JDBC** 기반의 **SQL** 스키마로 변환되어 네스티드 테이블이 생성된다.

ProductPrice
amount : String, buyer : String

(그림 26) ProductPrice 객체

```
SQL> CREATE TYPE ProductPrice_t AS OBJECT
(
  amount              VARCHAR,
  buyer               VARCHAR
);
SQL> CREATE TYPE ProductPrice_list AS
                    TABLE OF ProductPrice_t;
```

(그림 27) ProductPrice 데이블

(7) **Specification** 객체는 **ProductCatalog** 테이블에서 위에 열거된 ⑭번의 성질에 따라 그림 29와 같이 **JDBC** 기반의 **SQL** 스키마로 변환되어 네스티드 테이블이 생성된다.

Specification
specificationPcdata : String typeCode : String uom : String specificationName : String

(그림 28) Specification 객체

```
SQL> CREATE TYPE Specification_t AS OBJECT
(
  specificationPcdata       VARCHAR,
  typeCode                   VARCHAR,
  uom                       VARCHAR,
  specificationName        VARCHAR
);
SQL> CREATE TYPE Specification_list AS
                    TABLE OF Specification_t;
```

(그림 29) Specification 테이블

(8) **ProductAttachment** 객체는 **ProductCatalog** 테이블에서 위에 열거된 ⑤번의 성질에 따라 테이블 속성에 포함되어 그림 31과 같이 변환된다.

ProductAttachment
attachmentURL : String attachmentPurpose : String

(그림 30) ProductAttachment 객체

```
SQL> CREATE TYPE ProductAttachment_t AS OBJECT
(
  attachmentURL             VARCHAR,
  attachmentPurpose         VARCHAR
);
SQL> CREATE TABLE ProductAttachment OF
                          ProductAttachment_t;
```

(그림 31) ProductAttachment 테이블

위에서 변환된 각각의 객체들은 그림 32와 같이 객체타입이 생성되며, 그림 33과 같은 SQL3 스키마가 생성된다.

```
SQL> CREATE TYPE ProductCatalog_t AS OBJECT
(
  productID          VARCHAR,
  productName        VARCHAR,
  defaultLanguage    VARCHAR,
  defaultCurrency    VARCHAR,
  countryOfOrigin    VARCHAR,
  partner            Partner_list,
  classInfo          ClassInfo_list,
  productDescrition  ProductDescrition_list,
  manufacturer       Manufacturer_t,
  participants       Participants_list,
  productPrice       ProductPrice_list,
  specification      Specification_list,
  productAttachment  ProductAttachment_t
);
```

(그림 32) ProductCatalog 타입

```
SQL> CREATE TABLE ProductCatalog OF ProductCatalog_t
     NESTED TABLE partner STORE AS Partner_tbl;
     NESTED TABLE classInfo STORE AS ClassInfo_tbl;
     NESTED TABLE productDescrition STORE AS
                                  ProductDescrition_tbl;
     NESTED TABLE participants STORE AS Participants_tbl;
     NESTED TABLE productPrice STORE AS
                                  ProductPrice_tbl;
     NESTED TABLE specification STORE AS
Specification_tbl;
```

(그림 33) ProductCatalog SQL3 스키마

4. 결 론

XML은 웹에서 구조화된 정보나 반-구조화된 정보를 교환하기 위한 표준 마크업 언어로 채택되어 가고 있으며, B2B 전자상거래와 같이 XML을 이용한 정보교환이 보편화되고 있다. 이에 XML 메시지의 데이터베이스로의 저장을 위한 효율적인 방안이 요구되고 있다. 한편 Oracle8i와 9i 및 Informix, 그리고 SQL2000 서버 등과 같이 멀티미디어 응용 등을 위하여 기존의 관계형 DBMS들은 객체-관계형 DBMS로 확장되고 있으며, 이에 따라 관계형 데이터베이스 표준 안인 SQL2도 SQL3로 확대 개편되고 있다. 아울러 J2EE와 같이 JAVA를 기반으로 한 XML 응용이 확대됨에 따라 JDBC를 통한 XML 응용과 데이터베이스의 효율적인 연계 방안이 요망된다.

본 논문에서는 XML DTD를 토대로 하여 SQL3 스키마로의 변환을 위한 방법을 제시하였다. 이 변환을 위하여 먼저 XML DTD를 UML 클래스 다이어그램인 객체모델로 변환시키기 위한 방안을 제안하였으며, 변환된 객체모델을 SQL3 스키마로 모델링하기 위한 방법을 제시하였다.

본 논문에서 제안한 XML DTD를 토대로 한 SQL3 스키마로의 변환 방법은 JAVA를 기반으로 Oracle8i와 9i 및 Informix 그리고 SQL2000 서버 등과 같이 객체-관계형 데이터베이스를 토대로 XML 응용을 구축하기 위한 데이터베이스 설계 방안으로 활용될 수 있다.

향후 연구 방향으로 DTD로부터 객체모델로 매핑되는 관계를 기술하는 DTD-Mapper와 객체모델로부터 객체-관계 데이터베이스 표준으로 자리잡아가고 있는 SQL3 스키마로 변환하는 객체모델 Transfer에 대해 구현하고자 한다.

Acknowledgement

본 연구는 정보통신부의 ITRC 사업에 의해 수행된 것임

참 고 문 헌

[1] Banerjee, S., Kirshnamurthy, V., Kriishnaprasad, M., Murthy, R.,: Oracle8i-The XML Enabled Data Management System. In Sixteenth International Conference on Data Engineering(ICDE'00), 28 February-3 March 2000, San Diego, IEEE Computer Society Press, Los Alamitos, CA, 2000, 561-568.

[2] Boem, K., Aberer, K.: HyperStorM-Administering Structured Documents Using Object-Oriented Database Technology. Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada, June 1996.

[3] Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E., Subramanian, S.,: XPERANTO: Publishing Object Relational Data

as XML. In Suciu, D., Vossen(eds.), G., Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000, Dallas, Texas, USA, May 18-19, 2000, 105-110.

[4] Cheng, J., Xu, J.,: XML and DB2, In Sixteenth International Conference on Data Engineering (ICDE'00), 28 February-3 March 2000, San Diego, IEEE Computer Society Press, Los Alamitos, CA, 2000, 569-576.

[5] "Extensible Markup Language(XML)", http://www.w3.org/TR/PR-XML-971208.

[6] Florescu, D., Kossmann, D.: Storing and Querying XML Data using an RDBMS. Data Engineering 22:3 (1999), 27-34.

[7] George Reese, Database Programming with JDBC and JAVA 2nd Edition, O'RELEY, 2000.

[8] Henri Jubin, Juergen Fridrichs, Enterprise JavaBeans, Addison Wesley, 2000.

[9] Hiroshi Matuyama, Kent Tamura, Naohiko Uramoto, XML and Java Developing Web Applications, Addison Wesley, 1999, pp. 20-21.

[10] Joo Kyung-Soo, "A Design of Middleware Components for the Connection between XML and RDB", 2001 IEEE International Symposium on Industrial Electronics Proceedings, Pusan, Korea, June 2001.

[11] Kanne, C.-C., Moerkotte, G.: Efficient Storage of XML Data. In Sixteenth International Conference on Data Engineering(ICDE'00), 28 February-3 March 2000, San Diego, IEEE Computer Society Press, Los Alamitos, CA, 2000, 198.

[12] Klettke, M., Meyer, H.,: XML and Object-Relational Database Systems-Enhancing Structural Mappings Based on Statistics. In Suciu, D., Vossen(eds.), G., Proceedings of the the Third International Workshop on the Web and Databases, WebDB 2000, Dallas, Texas, USA, May 18-19, 2000, 63-68.

[13] Lee, D., Chu, W. W.,: Constraints-Preserving Transformation from XML DTD to Relational Schema. In Laender, A., Liddle, S., Storey(eds.), V., Conceptual Modeling, Salt Lake City, Utah, USA, October, 9-12, 2000, LNCS 1920, Springer-Verlag, Berlin, 2000, 323-338.

[14] Lee Sang-Tae, Joo Kyung-Soo, "Transforming XML DTD to SQL Schema based on JDBC", Proceedings of International Conference on East-Asian Language Processing and Internet Information Technology 2002(EALPIIT2002), Hanoi, Vietnam, January 8-11, 2002.

[15] Shanmugasundaram, J., Shekita, E. J., Bar, R., Carey, M. J., Lindsay, B. G., Pirahesh, H., Reinwald, B.,: Efficiency Publishing Relational Data as XML Documents. In Abbadi, A. E., Brodie, M. L., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., Whang(eds.), K. Y., VLDB2000-Proceedings of the 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt, Morgan Kaufmann Publishers 2000.

[16] Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., De-Witt, D., NaughtonJ.: Relational Databases for Querying XML Documents: Limitations and Opportunities. In Atkinson, M., Orlowska(eds.), M., VLDB'99-Proceedings of the 25th International Conference on Very Large Data Bases, Ediburgh, Sep 7-10, 1999, Morgan Kaufmann Publishers San Francisco, 1999, 302-314.

[17] W3C-World-Wide-Web Concertium. http://www.w3.org, 2000.

[18] World Wide Wed Consortium, "The XML Data Model", http://www.w3.org/XML/Datamodel.html, 2000.

[19] 방승윤, 주경수. "XML 스키마를 관계형 스키마로의 변환에 관한 연구", 한국정보처리 지식 및 데이터 공학연구회 학술발표논문, 2001, 293-299.

[20] 이상태, 주경수. "제약조건 유지를 위한 XML DTD의 관계 스키마로 변환 방법", 한국 인터넷정보학회, 제1권, 제2호, pp. 189-196, 2000.

[21] 이상태, 주경수. "UML 객체모델의 ORDB 객체 매핑", 한국인터넷정보학회, 제2권, 제1호, pp. 187-191, 2001.

[22] 이상태, 주경수. "객체모델을 이용한 XML DTD의 ORDB 스키마로의 변환", 한국데이터베이스학회, 춘계 Conference, pp. 303-310, 2001

[23] 이상태, 이정수, 주경수, "객체모델을 기반으로 한 XML DTD의 RDB 스키마로의 변환 방법", 대한전자공학회 하계종합학술대회 논문집 III, pp. 113-116, 2001.

[24] 이상태, 이정수, 주경수, "XML DTD를 기반으로 한 RDB 스키마 설계를 위한 Component 구현", 한국정보처리학회 지식 및 데이터공학 연구회 제8회 학술발표대회 논문집, pp. 309-316, 2001.

[25] 이상태, 주경수, "객체모델을 이용한 XML DTD의 ORDB 스키마로의 변환", 한국데이터베이스학회 논문집, 정보기술과 데이터베이스저널, 제8권, 제1호, pp. 105-116, 2001.

[26] 최문영, 방승윤, 주경수, "객체모델을 이용한, XML DTD의 OODBMS 스키마로의 매핑", 한국정보처리 지식 및 데이터 공학연구회 학술발표논문, 2001, 3001-307.

◐ 저 자 소 개 ◑

이 상 태

1987년 독일 DORTMUND대학교 전산학과 졸업(학사)
1994년 독일 DORTMUND대학교 대학원 전산학과 졸업(석사)
2002년 순천향대학교 대학원 전산학과학과 졸업(박사)
1996~현재 : 신성대학 컴퓨터응용계열 교수
관심분야 : XML, 웹데이터베이스, 전자상거래, 인터넷 응용
E-mail : stlee@shinsung.ac.kr

주 경 수

1980년 고려대학교 이과대학 수학과 졸업(학사)
1982년 고려대학교 일반대학원 전산학과 졸업(석사)
1986년 고려대학교 일반대학원 전산학과 졸업(박사)
1998년 University of North Carolina
1999년 Visiting Professor
1986~현재 : 순천향대학교 정보기술공학부 교수
관심분야 : Database Systems, Semi-structured Data and XML, System Integration, Object-oriented Systems
E-mail : gsoojoo@sch.ac.kr