XML DTD를 위한 확장 SQL DDL의 설계

오준환 °, 이병욱
경원대학교 대학원 전자계산학과

Design of Extended SQL DDL for XML DTD

Jun-Hwan Oh °, Byung-Wook Lee
Dept of Computer Science, Kyungwon University

요 약

최근 XML문서를 저장 및 검색하기 위한 연구가 활발히 진행되고 있다. DTD문서를 효율적으로 저장 관리하는 것도 중요하지만 이들을 위한 질의문에 대한 연구도 중요하다. 기존의 질의문들은 XML 사용자들만을 위한 것이었고 데이터베이스와의 연동을 위한 것이 아니었다. 데이터베이스에 저장된 문서를 저장 관리하기 위해서는 기존의 SQL 질의를 확장할 필요가 있다.

본 논문에서는 RDBMS에 DTD를 저장하기 위해 SQL의 DDL을 확장하였다. 확장을 할 때 기존의 SQL 사용자들도 쉽게 접근하기 쉽게 하기 위해 각 DTD 인스턴스들을 SQL의 데이터 형의 형태를 가지도록 하였고 XML 사용자들도 접근을 쉽게 하기 위해 선언 한는 방법을 기존의 DTD 선언하는 방법과 비슷하게 설계하였다.

1. 서론

최근 인터넷의 발전으로 정보의 표현이 HTML 만으로는 부족하여 새로운 표현 방법이 1997년 W3C(World Wide Web Consortium)에서 제시되었다. 제시된 구조 정보표현 방법은 HTML(Hyper Text Markup Language)의 편리성과 SGML(Standardized Generalized Markup Language)의 확장성을 조합한 형태를 가지고 있는 XML(eXtensible Markup Language)이다[1][2].

XML을 이용한 정보표현이 많아짐으로 인해 구조적 문서인 XML 문서를 데이터베이스에 저장 관리하는 연구가 이루어지고 있다. 객체지향 데이터베이스로의 저장과 관계형 데이터베이스로의 저장으로 나누어진다[3]. 저장방식으로의 분류는 가상 분할 방식과 분할 방식으로 나누어진다.

위에서 알아본 저장에 관한 연구는 XML 실제 문서 저장에 관한 연구이다. XML 문서자체의 저장뿐만 아니라. 저장된 문서들의 관리도 필요하고 또 그들을 정의해주기 위한 질의 문도 필요하다.

문서 저장 방식만 제안할 뿐 사용자들의 위해 SQL를 확장하지 않아 실제 시스템을 사용할 때 많은 어려움을 주었다.

본 논문에서는 RDBMS에 DTD 문서를 저장 관리하기 위한 질의문으로 SQL의 DDL을 확장할 것이다. 기존의 SQL의 형태와 XML DTD 문서의 형태와 비슷하게 설계하여 기존의 SQL사용자와 XML 사용자들의 접근을 쉽게 하였다. 제2장에서는 SQL의 확장을 제안하고 제안한 설계에 대한 실례를 알아볼 것이다. 제3장에서는 결론 및 향후 연구 방향을 제시한다.

2. SQL의 확장

기존의 XML문서를 위한 SQL의 확장은 SQL의 문법과 약간 다르고 키워드를 중복해서 사용하는 단점이 있다. 본 논문에서는 기존의 확장을 보완하여 제안하는 데이터 모델에 적용할 수 있게 하였다.

기존의 XML 문서를 위한 SQL의 확장은 SQL의 문법과 약간 다르고 키워드를 중복해서 사용하는 단

점이 있다. 본 논문에서는 기존의 확장을 보완하여 제안하는 데이터 모델에 적용할 수 있게 하였다.

다음 그림들은 확장된 CREATE 문의 BNF를 나타낸다[11]. CREATE문 중 DTD에 해당하는 부분만을 표현한 것이다. 표현 방식은 기존의 SQL의 형태와 XML DTD의 형태와 유사하게 설계하여 기존의 SQL사용자와 XML 사용자 모두에게 사용의 편리성을 제공하고자 하였다.

```
CREATE DTD <DTD name> '('
( <Element expression> | <Entity expression>  | <Attlist expression
> [,...] ) | *
')'
```

그림 2.1 CREATE DTD의 BNF

그림 2.1은 CREATE DTD의 기본적인 문법을 BNF로 표현한 것이다. CREATE DTD에서 DTD의 구성 요소인 엘리먼트, 에트리뷰트, 엔티티를 모두 표현하였다. 이들은 "AS" 이후에 "ELEMENT", "ATTLIST", "ENTITY"를 사용함으로 자료형을 정의해준다. 사용법은 기존의 CREATE TABLE과 비슷하게 테이블의 이름을 선언해준 후 "()"로 구분하여 사이에 구성 요소들을 선언해 주면 된다. 물론 자료형은 DTD의 구성요소인 "ELEMENT", "ATTLIST", "ENTITY"만을 사용할 수 있다.

```
<Element expression> ::= <ELEMENT name> AS ELEMENT ( '(' <
NonMix Model> | <Mix Model> | <Content> ')')
<NonMix Model> ::= ( <choice> | <seq> ) ( '?' | '*' | '+' )?
<choice> ::= '(' <space>? <cp> ( <space>? '|' <space>? <cp> )* <s
pace>? ')'
<seq> ::= '(' <space>? <cp> ( <space>? ',' <space>? <cp> )* <spac
e>? ')'
<Mix Model> ::= ( <Mix choice> | <Mixseq> ) ( '?' | '*' | '+' )?
<Mix choice> ::= '(' <space>? <Content> ( <space>? '|' <space>? <
cp> )* <space>? ')'
<Mix seq> ::= '(' <space>? <Content> ( <space>? ',' <space>? <cp>
)* <space>? ')'
<cp> ::= ( <Element name> | <choice> | <seq> ( '?' | '+' | '*' )?
<Content> ::= '#PCDATA' | "EMPTY" | "ANY"
```

그림 2.2 CREATE DTD의 BNF

그림 2.2는 CREATE DTD의 엘리먼트 선언부를 BNF로 표현한 것이다. 엘리먼트를 정의해 준 후 엘리먼트의 형태인 EMPTY, ANY, 내용모델을 정의해 준다. 내용모델의 경우 Mixed, nonMixed, 자료형을 각각 표현해 준다.

또 부모 엘리먼트의 경우는 하위 엘리먼트의 이름들을 반복하면서 사용하여 표현해 준다. 이 때 표현 방식은 기존의 XML DTD 문서 형식인 "(ElementA | ElementB)"와 같은 형식을 그대로 사용한다. Mixed 엘리먼트의 경우는 데이터형인 PCDATA도 가지고 자식 엘리먼트도 가지게 된다. 따라서 예를 들어 "(#PCDATA | ElementA | ElementB)"와 같이 표현된다. 하지만 nonMixed 엘리먼트는 자신의 하위 엘리먼트의 정보만을 가지므로 보편적인 예인 "(ElementA | ElementB)"의 형태를 가진다.

또 EMPTY, ANY, 내용 엘리먼트는 간단히 PCDATA, EMPTY, ANY의 형태를 가진다. ANY는 하위 엘리먼트의 제약이 없어 다른 어떤 엘리먼트도 하위 엘리먼트를 가질 수 있기 때문에 특별히 하위 엘리먼트를 선언해 주는 문법을 추가해 줄 필요가 없다. EMPTY 엘리먼트도 마찬가지이다. 엘리먼트는 CREATE DTD 자료형 중 "ELEMENT"이다.

```
<Attlist expression> ::= <ATTLIST name> <Attist Type> [ <de
fault> ] "AS ATTLIST IN" <ELEMENT name>
<Attlist Type> ::= <Enumerate Type> | ( <ENTITY name> )+
| <Cdata> | <xml Extend type> | <Tokenized type>
<default> ::= ( NOT NULL | NULL | DEFAULT <string> )
<Enumerate Type> ::= <Enumerate> | <Notation>
<Enumerate> ::= '(' <string> ( '|' <string>) * ')' DEFAULT <
string>
<Notation> ::= 'NOTATION' <space> '(' <string> ( '|' <string
> )* ')' DEFAULT <string>
<Cdata> ::= 'CDATA'
<Tokenized type> ::= 'ID' | 'IDREF' | ' IDREFS' | 'ENTITY'
| 'ENTITIES' | 'NMTOKEN' | 'NMTOKENS'
```

그림 2.3 CREATE DTD의 BNF

그림 2.3은 CREATE DTD의 에트리뷰트 선언부를 BNF로 표현한 것이다. 에트리뷰트를 선언할 때는 에트리뷰트를 포함하는 엘리먼트의 이름을 "IN" 키워드를 이용해 선언해준다. 또 "#REQUIRES", "#IMPLIED", "#FIXED"를 "NOT NULL", "NULL", "DEFAULT"로 변경하여 기존의 SQL에서 NULL의 의미와 동일하게 설계하였다.

"*<Tokenized type>*" 타입을 이용하여 에트리뷰트의 여러 가지 형을 지원한다. "*<Enumerate Type>*" 타입에서는 Enumerate의 요소들 XML DTD의 문서 형식에 비슷하게 표현한다. 또 "DEFAULT"를 이용하여 기본

값을 선언해줄 수 있다. 에트리뷰트는 CREATE DTD 자료형 중 "ATTLIST"이다.

```
<Entity expression> :: <ENTITY name> '(' <Entity Definition> ')'
S? 'AS ENTITY'
<Entity Definition> :: <GEDecl | <PEDecl>
<GEDecl> :: <Name> <space> <EntityDef>
<PEDecl> :: <Name> <space> <PEDef> <space> "is Refer"
<EntityDef> :: <EntityValue> | (<ExternalID> |
<NDataDecl>?)
<ExternalID> :: <SystemLiteral> <space> 'SYSTEM' | '('
<PubidLiteral> ',' <SystemLiteral> ')' <space> 'PUBLIC'
<NDataDecl> :: <space> 'NDATA' <space> <Name>
```

그림 2.4 CREATE DTD의 BNF중 엔티티 선언부

그림 2.4는 CREATE DTD의 엔티티 선언부를 BNF로 표현한 것이다. 엔티티를 선언할 때는 XML DTD의 기본 문법과 비슷하지만 참조 객체일 때는 "is Refer"라는 키워드를 사용하여 표현한다. 또 외부 객체일 때 "*SYSTEM*"과 "*PUBLIC*"를 사용할 때 문장 맨 뒤에 사용하는 점의 차이만 있다.

```
<ENTITY name> :: <stinrg>
<Element name> :: <stinrg>
<EntityValue> :: <stinrg>
<ATTLIST name> :: <stinrg>
<stinrg> :: String
```

그림 2.5 CREATE DTD의 BNF중 기타 선언부

그림 2.5 CREATE DTD에서 사용되는 기본적인 선언부를 나타낸다. 빈 공간과 각 구성요소들의 이름을 표현한 것이다. 이들은 모두 "*String*"이므로 확장 SQL의 키워드 이외의 문자열이라면 어느 문자이든지 상관없이 올 수 있다.

다음 그림 2.6는 DTD를 삭제할 때 사용하는 DROP DTD에 대한 BNF이다. DTD를 삭제할 때 사용하는 것으로써 DTD 자체를 삭제할 때만 사용되고 엘리먼트, 에트리뷰트, 엔티티를 삭제할 때는 사용할 수 없다. 이들을 삭제할 때는 ALTER DTD를 이용한다.

```
DROP DTD <DTD name>
<DTD name> :: String
```

그림 2.6 DROP DTD의 BNF

다음 그림 2.7은 DTD의 내용을 변경할 때 사용하는 ALTER DTD의 BNF이다. CREATE DTD에서 사용하는 엘리먼트, 엔티티, 에트리뷰트의 BNF를 그대로 사용하고 뒤에 각각의 변경 내용(ADD, DROP, UPDATE)을 표시해준다. 또 변경해준 후 어떤 종류의 DTD 구성요소가 변경되었는지를 표현해준다.

```
ATLER DTD <DTD name> ( <New DTD name> | '('
<ELEMENT> "AS ELEMENT" | <ATTLIST> "AS ATTLIST" |
<ENTITY> "AS ENTITY"
')' )
<ELEMENT> :: <ELEMENT name> ( 'DROP' ) | ( <Element
expression> 'UPDATE' ) | ( <Element expression> 'ADD' )
<ATTLIST> :: <ATTLIST name> ( 'DROP' ) | ( <Attlist
expression> 'UPDATE' ) | ( <Attlist expression> 'ADD' )
<ENTITY> :: <ENTITY name> ( 'DROP' ) | ( <Entity
expression> 'UPDATE' ) | ( <Entity expression> 'ADD' )
```

그림 2.7 ALTER DTD의 BNF

위의 3가지 DDL를 확장함에 있어서 기존의 무법인 CREATE, DROP, ALTER와 유사하게 제안하여 기존의 SQL을 사용하던 이용자들도 쉽게 적용할 수 있게 하였다. 또 XML의 BNF를 참고하여 제안하였기 때문에 XML 사용자도 적용하기 쉽게 하였다.

설계한 확장 SQL의 실제 XML DTD에 적용된 예를 이번 절에서 알아본다. 예로 사용하게 될 DTD는 그림 2.8을 사용할 것이다.

위의 XML DTD의 DTD파일 이름은 video이다. video라는 이름으로 DTD를 선언해 주면 된다. 또 각각의 엘리먼트와 에트리뷰트를 선언해 주고 각각 인스턴스의 종속 관계를 표현해준다.

```
<!ELEMENT video (title,company*,actor)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST title producer CDDATA #REQUIRED>
<!ELEMENT company (#PCDATA)>
<!ELEMENT actor (name | firstname ,lastname)>
<!ATTLIST actor cast CDDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

그림 2.8 비디오샵 video.dtd 예제

video 엘리먼트는 하위 엘리먼트로 title, company,

actor를 가지고 있다. 또 company 엘리먼트는 발생할 수도 있고 안 할 수도 있으며, 반복적으로 발생할 수도 있는 엘리먼트의 표현은 "*"으로 해준다. 또 actor 엘리먼트는 또다시 name, firstname, lastname 엘리먼트의 하위 엘리먼트를 가지고 있다.

producer 에트리뷰트는 title 엘리먼트에 속해 있고 cast 에트리뷰트는 actor 엘리먼트에 속해 있다. 이들 에트라뷰트는 단순한 속성인 CDATA형을 가진다. 이들은 "IN" 키워드를 이용해 속해 있는 엘리먼트들을 선언해 준다.

```
CREATE DTD video {
        video AS ELEMENT {
                (title,company*,actor) },
        title AS ELEMENT {
                #PCDATA },
                producer CDATA AS ATTLIST IN title,
        company AS ELEMENT {
                #PCDATA },
        actor AS ELEMENT {
                (name | firstname , lastname) },
                cast CDATA AS ATTLIST IN actor,
        name AS ELEMENT {
                #PCDATA },
        firstname AS ELEMENT {
                #PCDATA },
        lastname AS ELEMENT {
                #PCDATA } }
```

그림 2.9 CREATE DTD의 선언 예제

위에서 알아본 예제들은 XML DTD를 생성할 때 사용한 확장 SQL이었다. 다음으로 알아볼 예제들은 XML DTD를 수정하거나 삭제할 때 사용하는 키워드인 ALTER와 DROP에 관한 예제이다.

```
ALTER DTD video {
        producer DROP,
        company {
                (name , birth ) } UPDATE,
        name AS ELEMENT {
                #PCDATA } ADD,
        birth AS ELEMENT {
                #PCDATA } ADD }
```

그림 2.10 ALTER DTD의 선언 예제

```
DROP DTD video
```

그림 2.11 DROP DTD의 선언 예제

위에서 설계한 확장 SQL을 이용한다면 실례에서 알아본것과 같이 XML 사용자와 SQL사용자들에게 모두 사용하기 편한 질의문을 제공하여준다.

3. 결론 및 향후 연구 방향

XML 문서의 저장 뿐 만 아니라 XML 문서를 정의하는 부분인 XML DTD 문서의 저장관리도 필요하다. 마찬가지로 XML 문서 자체에 대한 질의문과 DTD 문서 자체를 위한 질의문의 필요성도 높아지고 있다.

따라서 본 논문에서는 사용자들의 편의성을 제공하고자 질의문을 설계하였다. XML 문서 자체를 위한 질의문은 기존에 많이 나와 있지만 데이터베이스에 저장하기 위한 질의문에 관한 연구는 많이 이루어져 있지 않다. 본 논문은 데이터베이스에 저장하기 위하여 SQL의 DDL 부분을 확장하였다. 특히 XML 문서의 생성을 위한 질의문이 아니라 DTD 문서의 저장을 위한 질의문을 제안하였다.

제안된 확장 SQL은 기존의 XML의 문법에 적용하여 설계하였고 특히 문서를 실제적으로 저장 관리하게 될 데이터베이스 관리자들을 위해 DTD 문서의 각 인스턴스들은 SQL의 데이터형으로 취급하여 하였고 또 선언 방식을 기존 SQL의 데이터형 선언과 비슷하게 하였기 때문에 사용이 편리해졌다.

향후 연구방향으로는 제안된 확장 SQL의 질의 처리를 위한 질의 처리기의 연구가 필요하다. 또 제안한 DDL의 확장 뿐만 아니라 검색에 사용하게될 DML의 확장도 필요하다.

[참고 문헌]

[1] ISO 8879, "Standard Generalized Markup Language(SGML)", Genervа Sritzerland, 1986

[2] eXtensible Markup Language(XML),"http://www.w3.org/TR/PR-xml-971208," 1997

[3] Susan Malaika, "Using XML in Relational Database Application," ICDE99, pp167, 1999

[4] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Complers - Principles, Techniques and Tools*, 1988

[5] Serge Abiteboul, Peter Buneman, Dan Suciu, *Data On the Web from Relations to Semistructuer Data and XML*, Moran Kaufmann Publishers, 2000