객체-관계형 데이터베이스를 사용한 XML 문서 저장 기법

이월영, 용환승
이화여자대학교

요약

XML은 비정규적이고 불완전한 구조를 허용하는 반구조적인 데이터로서 인터넷 상에서 데이터를 교환하기 위한 사실상의 표준으로 자리잡아가고 있다. 따라서 이러한 데이터를 효율적으로 다루기 위해서는 어떤 장치에 어떤 저장 기법으로 저장하느냐가 중요한 요소가 된다. 이 논문에서는 객체-관계형 데이터베이스를 대상으로 DTD의 존재 여부에 관계없이 XML 질의 언어에서 필요로 하는 다양한 질의 종류를 지원할 수 있는 저장 기법을 개발하였다. 이 기법은 기존의 데이터와 자연스럽게 연계할 수 있고 XML 데이터 모델의 특징 때문에 발생될 수 있는 오버헤드를 최소화 하였다.

Abstract

XML is becoming the *de facto* standard for data exchange over the Internet as a semistructured data which properties are irregular and incomplete. Therefore, to handle these data efficiently, what we use storage devices and storage techniques are main factors. In this paper, we developed storage techniques, which take the virtues of an object-relational database and support various query types needed for XML query languages without regard to the DTD. The techniques are capable of connecting naturally with conventional data and reducing overheads caused by the characteristics of an XML data model.

Keyword: storage technique, XML, Schema mapping

1. Introduction

XML[1]은 구성 요소들이 서로 상하로 계층적인 관계를 가질 뿐 아니라 수평적으로는 순서를 가지고 있으며 구성 요소들 사이의 구조가 비정규적인 형태를 허용하는 데이터 모델이다. 이러한 XML 문서를 저장하는 장치는 크게 나누어 관계형 데이터베이스나 객체지향 데이터베이스와 같은 기존의 데이터베이스를 사용하는 경우와, XML 문서의 성격에 맞도록 특별히 디자인된 구조와 인덱싱 기법, 질의 최적화 기법 등을 사용하여 XML 문서를 효과적

으로 저장하고 검색할 수 있도록 새로운 XML 전용 서버를 구축하는 경우가 있다[2,3,4]. 그러나 이 중 XML 문서를 위한 전용 서버나 객체지향 데이터베이스 시스템을 이용한 방법은 관계형 데이터베이스 시스템에 비해 일반적으로 보편화되어 있지 못하기 때문에 XML 문서와 자연스럽게 연계시킬 수 있는 현존하고 있는 데이터의 양이 훨씬 적다는 점이다. 또한 관계형 데이터베이스의 오랜 기간 성숙해온 질의 속도, 데이터 확장 가능성, 에러 복구, 동시 접근 제어 능력과 같은 안정성 있는 성능은 보편적으로 인정 받고 있는 기술들이다. 요즈음은 순수한 관계형 데이터베이스보다 능력을 확장시켜 테이블의 필드에 객체를 포함시킬 수 있는 객체-관계형 데이터베이스로 변모되었다.

데이터베이스 회사[5,6,7]에서나 여러 연구가[8-15]들은 XML 문서를 다룰 때, 이러한 관계형 데이터베이스 시스템의 장점을 이용하고자 하는 노력들을 해 왔으나, 비정규적인 구조를 허용하는 XML 데이터 모델을 정형화된 형태를 띄고 있는 관계형 데이터 모델에 저장하기 때문에 아직도 여러 가지 문제점들을 가지고 있다. 이러한 문제점 중 대표적인 것으로는 첫째, XML이 가지고 있는 순서 정보와 같은 속성은 대개 간과되고 있어 XML 질의 언어[16-21]에서 요구하는 다양한 질의 종류들을 모두 지원하지 못하고 있다는 점이다. 둘째, 비정규적인 XML 데이터를 정형화된 데이터 모델에 저장했기 때문에 두 모델 사이의 차이를 메우고자 많은 오버헤드 데이터들을 포함하고 있어 기존 데이터와 자연스럽게 연계되어 사용하기가 어렵다는 것이다.

따라서 본 논문에서는 비정규적인 구조를 가지고 있는 XML 문서를 어떻게 처리하여 어떤 방식으로 스키마를 매핑한다면 기존 데이터베이스의 안정성 있는 성능과 좋은 기술을 그대로 활용하면서도 ① XML의 계층적인 구조나 순서 정보와 같은 고유의 속성을 잃어버리지 않고 다양한 질의를 할 수 있을 뿐 아니라 ②두 모델 사이의 차이를 메우고자 발생할 수 있는 오버헤드를 최소화할 수 있을까를 연구하였으며 다음과 같은 접근 방법을 사용하여 이 문제를 해결하였다.

첫째, 기존 연구가 XML 문서를 대상으로 스키마를 생성할 때 DTD가 있는 문서만을 대상으로 하거나, 아니면 인스턴스만을 대상으로 하였던 것을 본 논문에서는 DTD를 가지고 있는 유효 문서나 DTD가 없는 문서 모두를 대상으로 XML 문서의 계층적인 구조를 그대로 반영하는 구조 트리를 추출하였다.

둘째, XML 문서에서의 계층적인 정보나 순서 정보와 같은 고유 속성을 보존하여 질의할 수 있을 뿐 아니라 XML 질의 언어에서 필요한 질의 종류를 모두 지원할 수 있는 특별한 인덱스 컬럼을 객체-관계형 테이블에 할당하여, XML 데이터를 효과적으로 저장할 수 있는 세 종류의 저장 기법, 즉 구조-텍스트 혼합 저장 기법, 구조-텍스트 분리 저장 기법, 이름 기반 저장 기법을 개발하였다.

셋째, XML 문서로부터 추출한 구조 트리를 객체-관계형 릴레이션에 사상 시킬 때 객체-관계형 데이터 모델이 필드에 객체를 포함시킬 수 있는 기능과 다른 객체를 참조할 수 있는 속성을 이용하여 사상 규칙(mapping rule)을 세움으로써 오버헤드 데이터를 최소화 시켰다.

우리는 이러한 접근 방법이 앞으로 XML에서 DTD가 있는 문서나 없는 문서 모두에 대해 구조를 추출하여 객체-관계형 데이터베이스의 스키마를 생성할 수 있는 기반이 될 것을 기대한다. 또한 새로 개발한 이 저장 기법들은 객체-관계형 데이터베이스를 이용하여 XML이 가지고 있는 고유의 특징을 잘 살려 질의할 수 있도록 할 뿐 아니라, 기존의 저장 기법과는 다르게 XML의 특징이 관계형 데이터베이스의 정형적인 테이블 형태와 맞지 않기 때문에 발생되는 여러 가지 오버헤드를 최소화할 수 있는 방법이 될 것으로 기대한다.

본 논문의 구성은 서론에 이어 2장에서는 관련 연구를, 3장에서는 XML 문서에서 구조를 추출하고 4장에서는 객체-관계형 데이터베이스에 XML문서를 저장하여 5장에서는 각 저장 기법에 따르는 성능을 평가하고 6장에서 결론을 맺는다.

2. Related Work

지금까지 많은 연구가들은 XML 문서를 다룰 때 순수 관계형 데이터베이스나 객체-관계형 데이터베이스의 안정성 있는 성능을 이용하고 이미 존재하고 있는 데이터와 자연스럽게 연계하고자 연구를 활발히 진행해 왔다[5-9,13,14,22,23].

STORED[8]에서는 마이닝 기법을 이용하여 반구조적(semistructured) 데이터를 관계형 데이터베이스에 사상 시키기 위한 정형적인 구조를 추출하고, STORED라는 질의언어를 사용하여 데이터 저장/삽입/삭제와 같은 기능을 처리하고 있다. 그러나 이 저장 기법은 비정규적인 데이터 처리를 위해 별도의 저장 공간을 사용하여 데이터의 조각화 현상을 초래할 뿐 아니라, XML이 가지고 있는 고유 속성을 보존하지 못하는 단점을 가지고 있다.

[9]에서는 애트리뷰트와 그 값을 어떻게 표현할 것인가에 초점을 맞추어 XML 문서를 관계형 데이터베이스에 사상하는 8가지 스킴을 제시하였다. 이 스킴들은 XML 문서에서 직관적으로 파악되는 구조를 모두 무시하고 엘리먼트 이름이나 애트리뷰트 이름에 따라 각각의 테이블에 이들이 가지고 있는 계층적인 정보를 함께 저장한다. 따라서 XML 문서의 계층 구조를 요구하는 질의에는 그 의미를 보존하여 질의할 수 있으나 테이블의 심각한 조각화 현상과 너무 많은 오버헤드 데이터를 포함하고 있다.

[13]은 XML 문서를 관계형 데이터베이스에 저장할 때 스키마를 추출하기 위하여 XML DTD를 대상으로 엘리먼트 그래프를 생성한다. 이것은 XML 문서를 관계형 데이터베이스에 완벽하게 사상 시킴으로써 XML 문서의 계층적인 정보를 유지시킬 수 있다는 장점을 가지고 있으나, 테이블에 저장된 엘리먼트들이 XML 문서에서 가지고 있던 계층적인 정보를 보존하기 위해 여러 테이블에 걸쳐 중복하여 저장된다. 또한 DTD가 없는 문서에 대해서는 스키마를 생성할 수 없다는 점이 문제이다.

대개의 이러한 저장 기법[8,9,13,23]들은 무엇보다도 XML의 고유 속성 중 하나인 순서 정보를 보존하지 못하고 있기 때문에 XML 문서상에서의 위치를 요구하는 질의문에는 응답할 수 없다는 것이다.

그러나 [14]는 순서 정보가 없는 관계형 데이터베이스에 XML 문서를 저장하여 순서 정보를 잃지 않고 질의할 수 있도록 하는 세가지 스킴을 개발하여 이들 각각에 대한 성능 평가를 수행하였다. 그러나 이것은 DTD가 없는 문서에 대해서는 엘리먼트나 애트리뷰트의 이름에 따라 서로 다른 테이블에 저장하는 기법이기 때문에 질의에 따라 테이블 간의 많은 조인을 필요로 하게 된다.

XML DBMS[26]는 모든 테이블에 기본키와 외래키 컬럼을 두고 XML 문서에 나타나는 엘리먼트와 애트리뷰트에 대해 순서 정보를 위한 컬럼을 할당함으로써 XML의 속성은 잘 유지할 수 있지만 오버헤드 데이터량이 실제 데이터량의 2배를 넘게 차지하고 있다.

또한 [27]은 DTD를 가진 문서에서만 가능한 기법으로서, DTD를 기준으로 [13]에서 제안하는 방법으로 스키마를 추출하여 XML 문서의 조각에 새로운 XADT라는 새로운 데이터 타입을 부여하여 ORDBMS에 저장하는 기법을 개발하였다.

상업적인 제품으로서 Oracle XML DB[5]나 SQL 2000 서버[6]는 XML 출판 기능이 최근 추가되었으나 XML 문서를 저장하기 위해서는 사용자가 일일이 수동으로 데이터를 저장해야 하고, SQL의 확장된 구문을 사용하여 질의하고 있기 때문에 XML 문서를 모두 분석할 줄 알고 관계형 데이터베이스의 특성을 잘 알고 있어서 테이블의 구조를 어떻게 디자인해야 효과적일지에 대한 지식이 있는 사용자라야만 이용이 가능하다.

따라서 DTD의 존재 여부에 상관없이 안정성 있는 관계형 데이터베이스의 기술을 이용하고 XML의 특징을 살릴 수 있는 다양한 질의를 지원하기 위해서는 효과적으로 두 데이터 모델간의 차이를 극복할 수 있는 방법 개발이 필요하다.

3. Structure Extraction for an XML Document

본 논문에서는 XML 문서를 객체-관계형 데이터베이스에 저장하고 이를 질의하기 위해서 3단계로 나누어 처리한다. 첫째, DTD가 없는 문서나 유효 문서 모두를 대상으로 XML 문서의 계층적인 구조를 추출한다. 둘째, 추출한 구조를 객체-관계형 데이터베이스의 스키마로

사상 시킬 수 있는 규칙을 세운다. 셋째, XML 문서에서 각각의 데이터에 대해 구성 요소들이 가진 속성을 보존할 수 있는 고유 식별자를 부여하여 테이블에 저장한다.

3.1 Structure Tree

XML 문서에서 계층적인 구조를 추출하여 구조 트리라는 것을 생성하는데 구조 트리의 각 요소에 대한 의미를 명확하게 하기 위하여 다음과 같이 정의한다.

정의 3.1 구조 트리 T = (L, V, E, O)는 간선에 레이블되는 트리로서 각 요소는 다음과 같이 정의된다.

- **L**은 사각형으로 표현된 레코드 형태로서 엘리먼트 이름과 애트리뷰트 이름들을 필드로 하고 이 레코드의 첫 번째 필드는 엘리먼트 이름이고 나머지 필드는 임의 개수를 갖는 애트리뷰트 이름들을 나타낸다.
- **V**는 엘리먼트 내용이 값인 것과 애트리뷰트 값의 타입을 나타내는 것으로서 원으로 표시한다.
- **E**는 간선을 나타내는 것으로서 엘리먼트의 구조 심볼(structure symbol)을 레이블로 가지고 있고 이것은 XML DTD에 있는 구조 심볼과 똑같은 의미를 갖는 것으로서 XML 인스턴스에서 나타나는 하나의 엘리먼트가 또 다른 엘리먼트로 나가는 간선의 수를 나타낸다. 이러한 구조 심볼로는 *, +, ?가 있고 *는 0개 이상을 뜻하고 +는 하나 이상을, 그리고 ?는 있거나 또는 없거나를 뜻한다. 또한 간선에 레이블을 가지고 있지 않은 경우는 이 간선을 통해 나가는 엘리먼트는 단지 하나임을 뜻한다.
- **O**는 엘리먼트들 사이의 순서를 정의하는 것으로 애트리뷰트 내의 순서는 무시하고 엘리먼트 사이의 순서는 어떤 엘리먼트가 서브엘리먼트로 가지고 있는 엘리먼트의 순서에 따른다.

객체-관계형 데이터베이스의 스키마를 생성하기 위하여 먼저 XML 문서로부터 구조를 추출하는데 DTD를 가지고 있는 유효 문서인 경우는 DTD를 기반으로 구조 트리를 구축하고, DTD가 없는 문서인 경우는 인스턴스로부터 구조를 추출하여 구조 트리를 생성한다.

3.2 Structure Tree for Valid-Documents

DTD가 있는 문서로부터 정의 3.1에서 정의한 구조 트리를 생성하기 위해서 다음과 같은 방법으로 DTD의 각 요소를 구조 트리에 사상한다.

Rule 3.1 DTD에 있는 엘리먼트 이름은 사각형으로 표현되는 L의 첫번째 필드에 사상하고 애트리뷰트 이름은 L의 두 번째 필드 이후부터 차례로 사상한다.

Rule 3.2 원으로 표현되는 V에는 엘리먼트의 값과 애트리뷰트 값이 가지고 있는 데이터 타입을 사상한다. 즉, 엘리먼트의 값이 가지는 데이터 타입인 PCDATA, CDATA, EMPTY, ANY와 애트리뷰트 값이 가질 수 있는 데이터 타입인 CDATA, ID, IDREF, NMTOKEN, enumerated, IDREFS, NMTOKENS을 그대로 원에 입력한다.

Rule 3.3 간선에 해당되는 E에는 DTD에 나타나는 엘리먼트에 대한 구조 심볼을 그대로 사상한다. 즉, 하나의 엘리먼트로부터 이름이 같은 또 다른 엘리먼트로 나가는 간선의 수에 따라 아무 표시도 없는 것과, *, +, ?를 레이블링한다.

Rule 3.4 DTD에 나타나는 엘리먼트의 순서는 그 자체가 순서 정보를 포함하고 있는 것이기 때문에 구조 트리에서도 이러한 순서를 유지하기 위해 순서 정보 O를 그대로 사상한다.

이 외에 다음과 같은 보조규칙을 정한다.

Sub-Rule 3.1 XML DTD에서 ENTITY, ENTITIES 타입으로 나타나는 것은 이 타입이 참조하고 있는 데이터를 값으로서 대입하고 난 것을 대상으로 구조 트리를 완성한다.

Sub-Rule 3.2 XML DTD에서 사용된 구조 심볼은 [13]에서 정의한 것처럼 원래의 DTD와 뜻이 같게 다음과 같이 단순화한다. ⓐ내포된 정의를 평평한 표현으로 변환하고, ⓑ여러 개의 연산자를 하나의 연산자로 줄이며, ⓒ같은 이름을 가진 서브 엘리먼트가 여러 번 나타나는 경우 이것을 그룹화하여 하나로 정리한다.

(a | b) → a?, b?	**a** → a***	**…, a*,…,a* → a***
(a, b)* → a*, b*	**a*? → a***	**…, a*,…,a? → a***
(a, b)? → a?, b?	**a?? → a?**	**…, a?,…,a? → a***
ⓐ	ⓑ	ⓒ

Sub-Rule 3.3 DTD 상에서 다른 엘리먼트를 공유함으로써 발생되는 사이클은 부모 엘리먼트와 같은 이름의 서브 엘리먼트를 명시적으로 표현하도록 함으로써 재귀적인 문제를 해결한다.

Example 3.1 다음은 XML DTD에서 엘리먼트가 가질 수 있는 여러 가지 핵심적인 타입과 애트리뷰트의 타입, 그리고 DTD의 구조 심볼들을 모두 포함하고 있는 하나의 XML DTD의 예이다.

```
<!ELEMENT MovieList (movie, music)*>
<!ELEMENT movie (title, genres+, leading_actor*, remark?)>
<!ATTLIST movie year ID #IMPLIED>
<!ATTLIST movie insertion_music IDREFS #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT genres (#PCDATA)>
<!ELEMENT leading_actor (name, artist_info?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT artist_info ANY>
<!ELEMENT remark (#PCDATA)>
<!ELEMENT music (title, genres+, (singer | vocalist))>
<!ATTLIST music music_no ID #REQUIRED>
<!ELEMENT singer (name, artist_info)>
<!ELEMENT vocalist EMPTY>
<!ATTLIST vocalist name CDATA #IMPLIED>
```

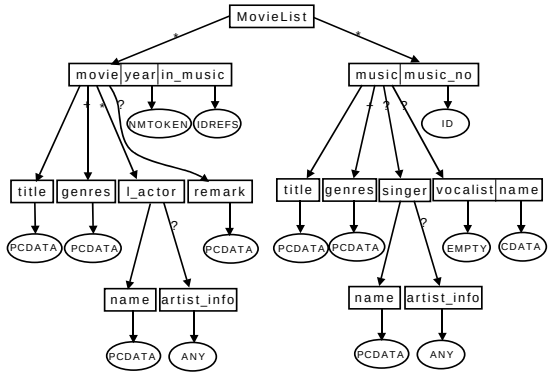

그림 3-1 XML DTD에 의한 구조 트리

이 예는 영화와 음악에 대한 XML 문서를 정의하는 DTD로서 영화에 삽입되었던 음악을 movie 엘리먼트가 참조하기 때문에 XML 문서 자체는 그래프의 형태를 띄도록 되어있다. 그러나 정의 3.1에 의하여 구조 트리를 완성하

면 그림 3-1과 같이 구조 트리에서는 애트리뷰트가 갖는 참조의 의미를 링크로 표현하지 않고 단지 애트리뷰트의 타입을 명시하기만 하기 때문에 트리 모습이 된다. 또한 singer와 vocalist는 원래 DTD에서는 선택적(|)이었는데 보조규칙 3.2에 의해 구조 트리에서는 ?이라는 구조 심볼로 변환되었다.

3.3 Structure Tree for Well-formed Documents

XML DTD가 제공되고 있지 않는 문서인 경우는 인스턴스만을 가지고 구조를 추출하여 객체-관계형 데이터베이스의 스키마를 생성할 때 이용할 수 있다. 본 논문에서는 XML 인스턴스를 직관적으로 이해하기 쉽게 하나의 루트를 가지고 있는 방향 그래프(directed graph) 형태로 표현한다. 엘리먼트는 사각형 모습의 레코드로 표현하여 첫번째 필드에 표시하고 두 번째 필드부터는 그 엘리먼트에 속하는 애트리뷰트를 표시한다. 엘리먼트 값이나 애트리뷰트 값은 원으로 표시하고 간선은 IDREF, IDREFS, XLink, URI등에 의해 애트리뷰트가 다른 엘리먼트에 속한 애트리뷰트를 참조하기 위해 것만 점선으로 표현하고 그 외는 실선을 사용한다.

다음은 DTD가 없는 XML 인스턴스로부터 구조 트리를 생성하기 위한 규칙이다.

Rule 3.5 XML 인스턴스 그래프에서 루트에 해당하는 정점을 *0* 레벨이라 하고 단말에 해당하는 정점을 *n*레벨이라 하자. 이 때 레벨 $i(0 \le i < n)$의 한 정점에서 같은 이름을 갖는 $i+1$ 레벨의 정점으로 나가는 간선들이 있을 때, 구조 트리에서는 이들을 그룹 지어 한 노드로 표현하고 애트리뷰트 필드들은 엘리먼트 필드가 그룹화 될 때 함께 이동한다. 또한 i 레벨에서 $i+1$ 레벨로 이르는 간선에는 몇 개의 노드가 그룹화 된 것이냐에 따라 구조 트리의 간선에 다음과 같이 레이블링한다.

- i 레벨의 한 노드에서 $i+1$ 레벨로 나가는 간선의 수를 0개나 또는 그 이상을 하나로 정리한 경우는 *로 한다.

- 1개나 또는 그 이상의 간선을 하나로 정리한 것이면 +로 한다.
- 동일한 이름의 노드가 없는 경우는 레이블링하지 않는다.
- 0 또는 1개가 존재하는 경우는 ?로 레이블링한다.

예제 3.2 XML 문서의 인스턴스가 그림 3-2와 같고 이것은 예제 3.1의 DTD를 따르지만 DTD가 알려져 있지 않다고 하자. 여기서 사각형으로 표시되는 첫 번째 필드는 엘리먼트를 뜻하고 나머지 필드들은 애트리뷰트를 뜻한다. 또한 원으로 표시되는 정점은 엘리먼트 값이나 애트리뷰트 값을 뜻한다. 이 때 루트에서 1 레벨로 나가는 간선에 매달린 정점들 중 같은 이름을 갖는 정점은 movie라는 이름의 A, B가 있고, music이라는 이름의 C, D, F가 존재하기 때문에 movie(A, B)와 music(C, D, F) 두 그룹으로 나누어진다. 이들에 대해서는 그룹 지어진 간선의 수가 모두 1개 또는 그 이상이기 때문에 +로 레이블링한다. 또한 music(C, D, F)에서는 singer로 나가는 간선이 존재하는데 자세히 보면 C와 F에서 각각 하나씩의 singer로 나가고 있지만 D에서는 singer라는 정점으로 나가는 간선이 존재하지 않는다. 따라서 ?로 레이블링한다. Vocalist도 역시 마찬가지이다. 또 movie(A, B)에서 title로는 각각 나가는 간선이 하나씩 존재하기 때문에 title이라는 정점으로 나가는 간선에는 아무런 심볼도 레이블링하지 않는다. 또한 l_actor는 A에서는 두 개, B에서는 한 개의 간선이 나가고 있기 때문에 +로 레이블링한다. 그림 3-2에 대해 규칙 3.5와 3.6을 적용하여 구조 트리를 완성하면 그림 3-3과 같이 된다.

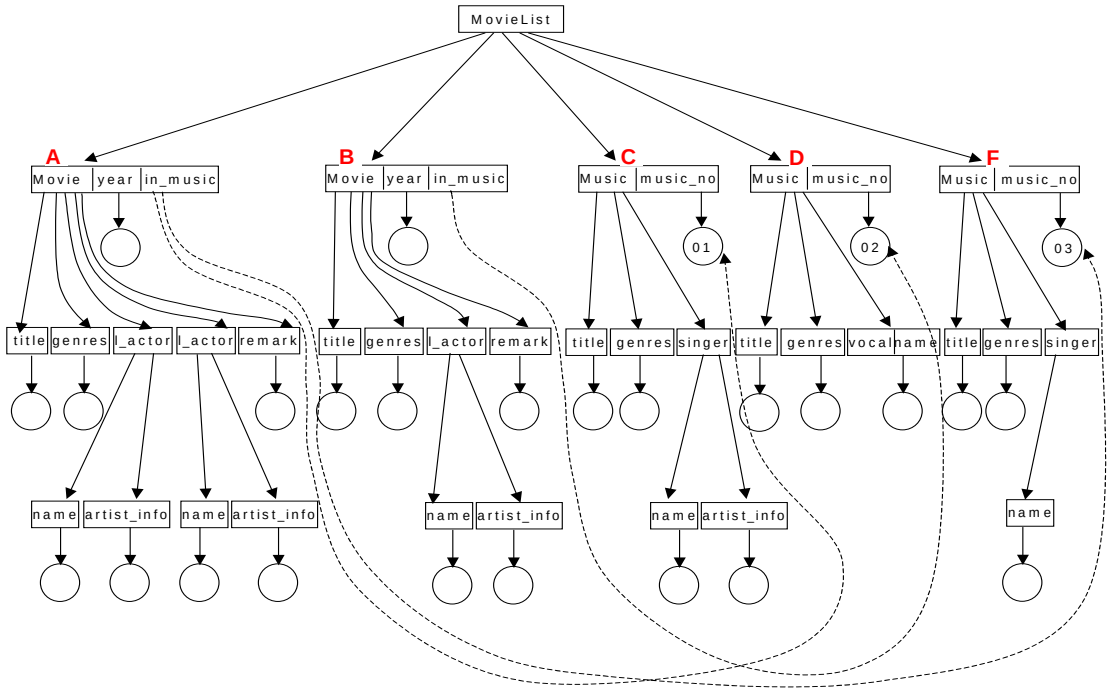

그림 3-2 XML 인스턴스를 위한 그래프

Rule 3.6 사각형으로 표현된 하나의 정점에서 하나의 간선이 실선으로 나가 원이 된 정점들은 모두 CDATA로 데이터 타입을 정의하여 다시 원으로 처리하고, 두개 이상의 실선이 나가는 경우는 오로지 애트리뷰트에서만 가능한데 NMTOKENS라 표시하여 원으로 처리한다.나가는 간선이 없는 경우는 EMPTY로 하여 원으로 표현한다. 점선으로 표시되는 참조를 뜻하는 간선은 나가는 간선이 하나인 경우는 CDATA로, 두개 이상인 경우만 IDREFS로 표시하여 원으로 처리한다.

Example 3.3 그림 3-2에서 원으로 표현되는 모든 정점은 CADTA로 처리하되, 참조를 뜻하는 점선으로 표시되는 in_music에서 나가는 간선만 두 개 이상인 경우가 존재하기 때문에 IDREFS로 표시하여 원으로 처리한다. 그리고 vocalist라는 정점은 사각형으로 표현되었음에도 아무런 나가는 간선이 없기 때문에 이런

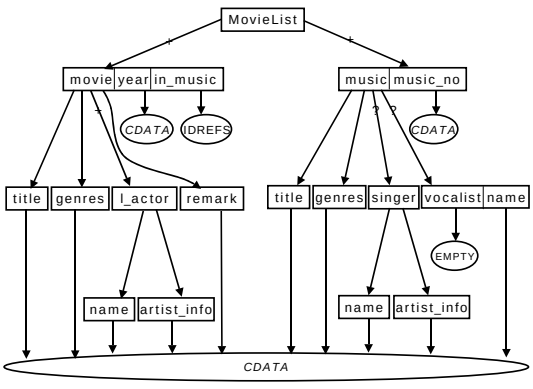

그림 3-3 그림 3-2에 대한 구조 트리

것은 내용이 비어있는 엘리먼트를 뜻한다. 따라서 구조 트리에서는 EMPTY로 표시하여 원으로 처리한다.

규칙 3.5과 3.6에 의해 그림 3-2의 XML 데이터 그래프에 대해 구조 트리를 생성하고 나면 그림 3-3과 같이 된다. 이것은 DTD로부터 구조 트리를 생성한 그림 3-1과 유사하다.

4. Storing XML Documents into an Object-Relational Database

객체-관계형 스키마는 ①구조 트리를 이용하여 릴레이션의 이름과 컬럼을 할당하고 데이터 타입을 정하는 단계, ②XML 문서에서 각 요소들이 가지고 있던 고유 속성들을 보존하여 질의할 뿐 아니라, 현재 질의 언어에서 필요로 하는 질의 유형들을 모두 지원할 수 있도록 인덱스 컬럼을 할당하기 위하여 XML 문서의 각 구성 요소에 고유 식별자를 부여하는 단계, ③고유 식별자들과 함께 실제 데이터를 저장하는 세가지 방법에 따라 완성된다.

4.1 Rules for Mapping an XML Structure Tree into an Object-Relational Database

구조 트리를 기반으로 객체-관계형 데이터베이스 테이블에 XML 데이터를 저장할 수 있도록 릴레이션의 이름과 컬럼을 할당하고 데이터 타입 결정을 위해 기본적인 규칙을 세운다.

Rule 4.1 기본적으로 객체-관계형 릴레이션의 스키마는 구조 트리에 대하여 너비 우선 탐색으로 순회하며 생성한다.

Rule 4.2 레벨별로 한 노드 A에서 나가는 간선에 매달리는 노드들은 모두 하나의 릴레이션으로 생성하며, 릴레이션 이름은 A 노드 이름과 같게 부여하고, 릴레이션 A의 컬럼 이름은 A 노드에서 나가는 간선에 달린 노드 이름을 부여한다. 단, 애트리뷰트 필드는 이것이 속한 엘리먼트의 서브 엘리먼트와 같이 취급한다. 이 때 간선에 레이블되는 구조 심볼이 * 나 +인 경우는 규칙 4.3을 따른다. 또한 간선에 레이블되는 구조 심볼이 ?인 경우는 규칙 4.4를 따른다.

Rule 4.3 상위 레벨의 노드에서 *나 +로 간선에 레이블링되어 생성된 노드는 자신의 하위 레벨에 두 형태의 서브 노드를 가질 수 있다. 즉, ① 단일 레벨로 끝나는 원으로 표시된 노드만을 갖는 경우 ②2 레벨 이상의 서브 노드를 갖는 경우로서 ①의 경우 내포된 릴레이션로 처리하고 ②의 경우 참조 타입으로 처리하여 여기서 참조하는 외부 릴레이션을 하나 더 생성한다.

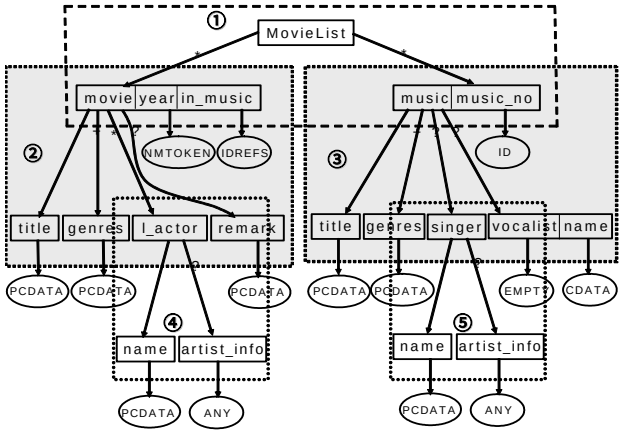

그림 4-2 구조 트리와 관계형 데이터베이스의 사상 관계

Rule 4.4 간선이 구조 심볼 ?로 레이블된 경우는 이 간선에 매달리는 노드가 하나 있거나 또는 없는 것을 뜻하기 때문에 문서상에서 일반적으로는 한번 이상 나타나게 되어 있다. 따라서 간선에 아무 것도 레이블되지 않은 노드와 똑같이 취급한다.

Rule 4.1 그림 3-1이나 3-2의 구조 트리를 대상으로 규칙4.1-4.4를 적용하여 객체-관계형 릴레이션이 생성될 것을 예측해 본다면 그림 4-2와 같다. 이 그림으로 본다면, 계층적인 구조와 텍스트를 모두 포함하는 릴레이션의 개수는 5개가 생성되며 릴레이션 이름은 상위의 노드 이름이 되고, 이 상위 노드에 매달린 노드들의 이름이 컬럼 이름이 된다. 즉, Movielist라는 릴레이션은 movie와 music을 컬럼으로 갖고, 다시 movie라는 릴레이션은 year, in_music, title, genres, l_actor, remark라는 컬럼을 갖는다.

다음은 데이터 타입을 결정하는 규칙이다.

Rule 4.5 구조 트리에서 원으로 표현되는 PCDATA, CDATA, NMTOKEN, ANY 등은 변화 가능한 크기의 문자형으로 정의하고 애트리뷰트를 위한 ID 타입과 IDREF 타입은 고정된 크기의 문자형으로 한다.

Rule 4.6 원으로 표현되는 엘리먼트 타입 중 EMPTY 타입은 컬럼을 할당하지 않는다.

Rule 4.7 원으로 표현되는 애트리뷰트 타입 중 IDREFS나 NMTOKENS는 여러 개의 값을

갖는 경우이기 때문에 규칙 4.3 ①의 경우와 같이 특별히 내포된 릴레이션으로 구성하여 해당 릴레이션의 내부에 포함시킨다. 데이터 타입은 변화 가능한 문자형으로 한다.

구조 트리에서 객체-관계형 스키마로 사상하기 위한 규칙들은 DTD가 없는 문서에서 구조를 추출한 그림 3-3과 같은 구조 트리에 대해서도 똑같이 적용될 수 있다.

4.2 Assigning Identifiers to XML Data

규칙 4.1-4.7에 따라 기본적인 스키마를 생성하고, XML 데이터의 각 요소에 대해 XML 문서에서 자신의 계층적인 상하관계나 순서 관계를 나타낼 수 있는 고유의 식별자를 규칙 4.8에 따라 부여하여 실제 데이터와 함께 저장될 수 있도록 고유 식별자 컬럼을 할당한다.

Rule 4.8 XML 인스턴스 그래프에 있는 각 정점에 대해 식별자 (P_a, P_b)를 부여한다. P_a, P_b는 {0,…,9}에 속하는 수들의 조합으로 이루어지고, P_a는 그래프에서 상하의 계층적인 관계 즉, 혈통관계를 나타내는 숫자이고, P_b는 수평 관계 즉, 형제관계를 뜻하는 숫자이다. 부모 노드 α의 식별자인 두 수 P_a, P_b를 연결한 $P_a + P_b$가 자식 노드 β의 P_a가 되고 자식 노드의 P_b는 β의 형제 사이의 순서 값을 부여 받는다. 이렇게 부여된 P_a와 P_b는 방향 그래프 내에서 그 정점을 결정지을 수 있는 고유의 패턴을 갖게 된다.

Example 4.2 그림 3-2에 대하여 고유 식별자를 부여하면 그림 4-1과 같이 된다. 각 정점 위에 있는 숫자가 인덱스를 위해 부여하는 고유 식별자 P_a와 P_b이다. 여기서 P_a와 P_b 값은 계층 기반 질의나 순서를 지정하는 질의를 할 때 그 속성을 보존하도록 하는 중요한 역할을 하는 값으로서 객체-관계형 테이블에 저장되는 모든 엘리먼트와 애트리뷰트에 대해 하나씩 할당하여 저장하고 이 컬럼을 인덱싱하여 XML 질의 언어에서 요구하는 모든 종류의 질의에 빠르게 응답할 수 있도록 한다[28].

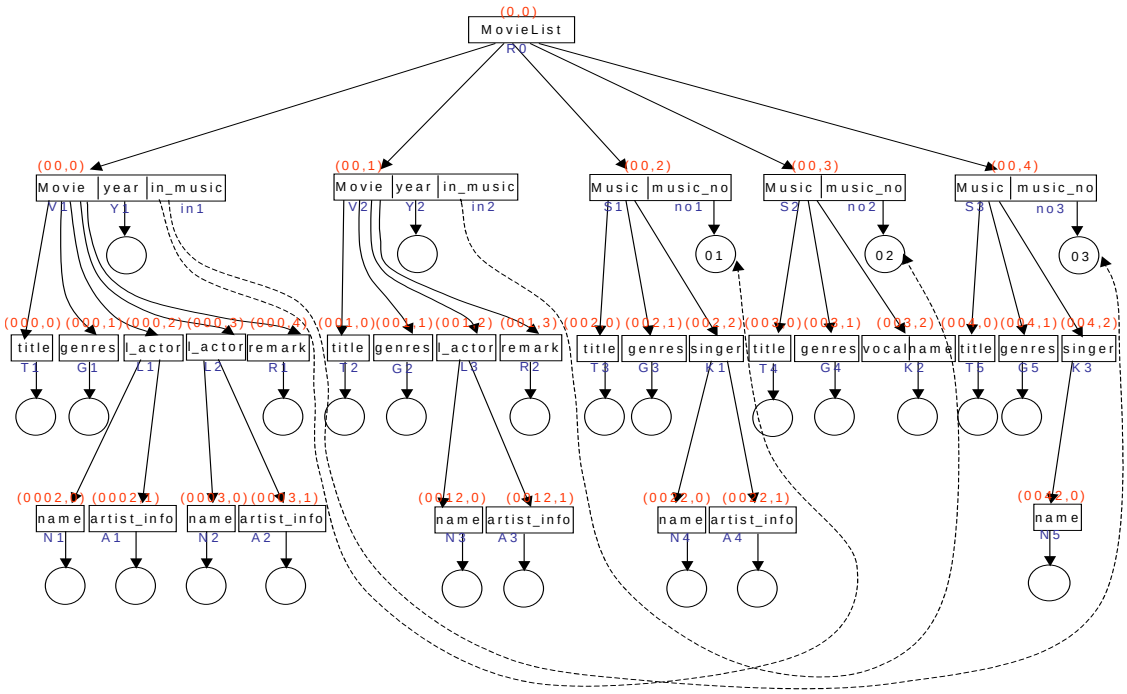

그림 4-1 XML 각 구성 요소에 패턴 부여

4.3 Storage Technique

이 절에서는 실제로 XML 데이터를 저장하는 방법에 따라 구조-텍스트 혼합 저장 기법, 구조-텍스트 분리 저장 기법, 이름 기반 저장 기법을 개발한다.

4.3.1 Structure-Text Mixed Storage Technique

XML 문서의 기본 골격을 이루는 엘리먼트는 엘리먼트 이름과 그 내부의 내용으로 이루어 진다. 그런데 이 내용은 또 다른 서브 엘리먼트를 갖는 엘리먼트이거나 아니면 단일 값일 수 있다. 우리가 여기서 제안하고 있는 두 가지 저장 기법은 바로 이러한 엘리먼트의 내용과 같은 모습으로 서브 엘리먼트와 단일 값을 혼합하여 한 릴레이션을 생성하는 방법과 모든 단일 값들에 대해서는 별도의 릴레이션을 생성하여 저장하는 방법으로 나누어진다.

우선 서브 엘리먼트와 단일 값을 혼합하여 저장하는 기법은 다음과 같은 규칙에 따라 데이터를 저장한다.

첫째, 모든 릴레이션에 대한 스키마에는 한 튜플 전체를 식별할 수 있는 ID(identification) 컬럼이 처음 컬럼으로 할당된다.

둘째, 구조 트리에서 사각형으로 표현되는 모든 노드들은 객체-관계형 스키마를 생성할 때 하나의 컬럼으로 사상된다. 또한 이렇게 사상된 컬럼을 질의할 때 XML 문서상에서 가지고 있던 계층적인 구조나 순서와 같은 의미 정보를 보존하여 질의할 수 있도록 4.2절에서 부여한 인덱스 정보를 저장하기 위한 컬럼을 확보한다. 따라서 구조 트리로부터 사상된 매 컬럼마다 이 컬럼을 식별할 수 있는 ID 컬럼과 짝이 되어 저장된다.

셋째, 규칙 4.3의 ②번의 경우에 의해 별도로 생성되는 릴레이션을 참조해야 하는 경우는 단지 참조할 값을 저장하기 위한 컬럼만을 할당한다.

넷째, 규칙 4.3의 ①번의 경우에 의해 내포된 릴레이션로 처리되는 컬럼에 대해서는 ID 컬럼을 함께 할당한다.

다섯째, 엘리먼트 중 구조 트리를 생성하는 과정에서 선택(|) 타입인 (a | b) 형태는 a?, b?로 처리하여 만든다. 이 때 두 노드는 각각 a 컬럼, b 컬럼으로 생성하지만 이들을 위한 ID 컬럼도 각각 별도로 할당한다. 그리고 a 필드가 값을 가질 땐 반드시 b 필드는 값을 갖지 않고, a 필드가 값을 갖지 않을 땐 b 필드는 값을 갖게 되는데, 이렇게 값을 갖지 않을 때는 실제 데이터 저장 시 널 값으로 필드를 채운다.

이와 같은 규칙들을 적용시켜 그림 3.1이나 그림 3.3의 구조 트리에 대하여 객체-관계형 스키마를 완성하여 이것을 릴레이션_이름(컬럼_이름$_1$: 컬럼_타입, ..., 컬럼_이름$_n$:컬럼_타입)의 형태로 표현하면 다음과 같이 된다.

Movilist (MovieList_ID:string, movie:reference, music:reference)
Movie (movie_ID:string, year_ID:string, year:string, in_music(in_music_ID:string, in_music:reference) title_ID:string, title:string, genres(genres_ID:string, genres:string), leading_actor_ID:string, leading_actor:reference, remark_ID:string, remark:string)
Music (music_ID:string, music_no_ID:string, title_ID:string, genres(genres_ID:string, genres:string), SingerVocalist_ID:string, singer:ref, vocalist_name:string)
L_actor (l_actorID:string, name_ID:string, actor_info_ID:string, actor_info:string)
Singer (singerID:string, name_ID:string, name:string, artist_info_ID:string, artist_info:string)

music

musicID	Music_noID	Music_no	TitleID	title	genres		singer	Vocal_noID	Vocal_no
					genreID	genres	singer		
00,2	00,2	01	002,0	Scabor ough fair	002,1	pop/balla d	Ref(K1)	Null	Null
00,3	00,3	02	003,0	Two of us	003,1	pop/soft rock	Null	003,2	The Beatles
00,4	00,4	03	004,0	The sound of silence	004,1	pop/folk	Ref(K3)	null	null

그림 4-2 구조-텍스트 혼합 저장기법

이러한 구조를 기반으로 XML 문서를 저장할 때는 그림 4-2와 같이 구조 트리에서 사각형으로 표현되는 노드가 서브 노드를 갖는 경우는 컬럼을 할당하여 서브 노드를 참조할 수 있도록 참조 값을 저장한다. 또한 서브 노드를 갖지 않는 경우는 그 노드가 갖는 값에 대한 데이터 타입이 표현되어 있기 때문에 그 노드의 실제 값을 저장한다. 또한 구조 트리에서 사각형 노드가 간선에 +나 *로 레이블링 되면서 단일 레벨의 원 노드만을 갖는 경우는 내포된 릴레이션로 처리하여 원 노드에 표현되어 있는 데이터 타입의 실제 값을 저장한다.

이 저장 기법은 XML 문서의 계층적인 구조를 순회하기 위해 참조 정보를 담고 있는 컬럼과 직접적인 데이터 값을 담고 있는 컬럼들이 혼합되어 하나의 릴레이션을 이루는 경우와 단말 노드의 값들 만을 담고 있는 릴레이션들로 이루어진다. 따라서 이 저장 기법은 구조 트리로 표현되는 형태와 같은 모습으로 저장되기 때문에 스키마를 생성하기도 쉽고 스키마를 예상하기도 쉬워서 사용자들이 직접 릴레이션을 대상으로 XML 질의를 수행하기가 쉽다.

4.3.2 Structure-Text Separated Storage Technique

이 방식은 구조 트리에서 볼 때 사각형으로 표현된 노드와 원으로 표현된 노드를 분리해서 릴레이션을 생성하는 것이다. 즉, 사각형으로 표현되는 노드에 대해서는 규칙 4.1에서 규칙 4.3까지를 적용하여 스키마를 생성하고 테이블에 저장하는 것은 실제 데이터를 담고 있는 테이블을 참조하는 정보만을 저장한다. 또한 원으로 표현되는 노드에 대해서는 이 노드의 부모 노드의 이름을 컬럼 이름으로 하고 테이블에는 원에 표현되어 있는 데이터 타입에 해당하는 실제 데이터를 저장한다. 따라서 사각형으로 표현되는 노드에 대한 릴레이션들은 XML 문서에서 내부 노드(internal node)들의 계층적인 구조와 같게 되고, 원으로 표현되는 노드를 저장한 릴레이션들은 단말 노드에 실제 데이터를 담고 있는 텍스트를 위한 값들의

릴레이션이 된다.

구조 트리에서 사각형으로 표현되는 내부 노드(internal node)들에 대한 릴레이션은 다음과 같다.

Movilist **movilistID:string, movie:reference, music:reference)**
Movie **(movieID:string, year:reference, in_music(in_music:reference), title:reference, genres(genres:reference), l_actor(l_actor:reference), remark:reference)**
Music **(musicID:string, music_no:reference, title:reference, genres(genres:reference), singer:reference, vocalist_name:reference)**
l-actor **(l_actorID:string, name:reference, artist_info:reference)**
singer **(singerID:string, name:reference, artist_info:reference)**

또한 텍스트 데이터를 저장하기 위한 릴레이션들의 스키마는 다음과 같다.

Year **(yearID:string, year:string)**
In_music **(in_musicID:string, in_music:string)**
Music_no **(music_noID:string, music_no:string)**
Vocalist_ name **(vocal_ nameID:string, name:string)**
Title **(titleID:string, title:string)**
Remark **(remarkID:string, remark:string)**
Genres **(genresID:string, genres:string)**
name **(nameID:string, name:string)**
Artist_info **(artist_info:string, artist_info:string)**

music

musicID	music_no	title	genres	singer	Vocal_no
			genres		
00,2	Ref(00,2)	Ref(003,0)	Ref(002,1)	Ref(002,2)	Null
00,3	Ref(00,3)	Ref(004,0)	Ref(003,1)	Null	Ref(003,2)
00,4	Ref(00,4)	Ref(005,0)	Ref(004,1)	Ref(004,2)	Null

music_no

music_noID	year
00,2	01
00,3	02
00,4	03

genres

genresID	genres
000,1	Drama
001,1	Drama
002,1	pop/ballad
003,1	pop/soft rock
004,1	pop/folk

title

titleID	title
000,0	The Graduate
001,0	I am Sam
002,0	Scaborough fair
003,0	Two of us
004,0	The sound of silence

vocal_no

vocalID	name
003,2	The Beatles

그림 4-3 구조-텍스트 분리 저장기법

이 저장기법은 구조-텍스트 혼합 저장 기법에 비해 관계형 데이터베이스를 이용하는 사용자들이 릴레이션 전체의 컬럼을 선택할 때 릴레이션에 저장된 컬럼들 하나 하나를 명시하지 않아도 된다는 장점을 가지고 있다. 즉, 그림 4-3처럼 텍스트 데이터를 저장하기 위한 테이블이 별도로 있기 때문에 genres에 들어 있는 모든 정보를 선택하기 위해 **select * from genres**하게 되면 바로 질의하여 결과를 받아볼 수 있다. 그러나 구조 트리에서 구조와 텍스트 데이터를 혼합하여 유지하고 있는 중간 노드들인 경우, 구조-텍스트 혼합 저장 기법보다 테이블들이 많이 나누어져 있기 때문에 결과를 반환할 때 보다 많은 순회가 필요하게 된다.

구조-텍스트 혼합 저장 기법이나 구조-텍스트 분리 저장 기법 모두 기존의 객체-관계형 데이터베이스의 데이터와 자연스럽게 연동하여 사용할 수 있는 장점이 있는 반면, XML 문

서의 형태로 결과를 반환하고자 할 때 관계형 데이터베이스 형태의 데이터를 XML 문서에서 필요한 형식으로 변환해야 하는 어려움이 있다. 또한 직접 테이블을 대상으로 질의를 하는 것이 아니라 XML 문서를 대상으로 질의한 경우, XML 문서에서처럼 서브 엘리먼트를 내포한 형태로 결과를 요구하기 때문에 내부 처리는 테이블 사이의 순회가 필요하게 된다. 따라서 이러한 부분을 최소화 할 수 있도록 하는 또 다른 저장 기법인 엘리먼트 이름과 애트리뷰트 이름을 기반으로 하는 저장 기법을 소개한다.

4.3.3 Name-based Storage Technique

이 방법은 XML 문서에 대한 구조 트리를 기초로 객체-관계형 스키마를 생성할 때 모든 내부 노드에 대하여 릴레이션을 생성하는데 엘리먼트 이름이 같은 경우 같은 릴레이션에 저장한다. 따라서 간선에 레이블된 구조 심볼 같은 것은 중요하지가 않고 구조 트리에 나타나는 엘리먼트 이름과 애트리뷰트 이름이 중요하다. 그리고 엘리먼트의 타입이 EMPTY인 경우는 데이터가 없는 경우이기 때문에 객체-관계형 릴레이션의 컬럼을 생성하지 않는다. 이것도 앞의 두 기법처럼 질의를 위해 인덱스를 위한 컬럼을 매 릴레이션마다 할당한다. 따라서 이러한 방법으로 객체-관계형 릴레이션의 스키마를 생성하면 다음과 같이 정리가 된다.

MovieList	**(MovieListID:string, MovieList:string)**
Movie	**(movieID:string, movie:string)**
Music	**(musicID:string, music:string)**
Title	**(titleID:string, title:string)**
Genres	**(genresID:string, genres:string)**
leading_actor	**(l_actorID:string, l_actor:string)**
remark	**(remarkID:string, remark:string)**
name	**(nameID:string, name:string)**
artist_info	**(a_infoID:string, a_info:string)**
singer	**(singerID:string, singer:string)**
vocalist_name	**(v_nameID:string, v_name:string)**
year	**(yearID:string, year:string)**
in_music	**(in_musicID:string, in_music:string)**
music_no	**(music_noID:string, music_no:string)**

데이터를 저장할 때는 그림 4-4처럼 XML 인스턴스에 나타나는 모습 그대로 덩어리로 저장한다. 즉, 서브 엘리먼트가 내포되어 있는 경우 그대로 서브 엘리먼트를 포함시켜 저장한다.

이 저장 기법은 테이블 조인이나 순회, 그리고 응용에서 XML 문서 형태로 반환하기 위해 아무런 후처리 필요 없이 어떠한 질의에도 빠르게 응답할 수 있는 장점이 있는 반면 많은 데이터가 중복된다는 단점이 있다. 따라서 질의 속도에 민감하고 데이터의 양이 많이 중복된다 해도 크게 문제가 되지 않는 경우 유용한 저장 기법이다.

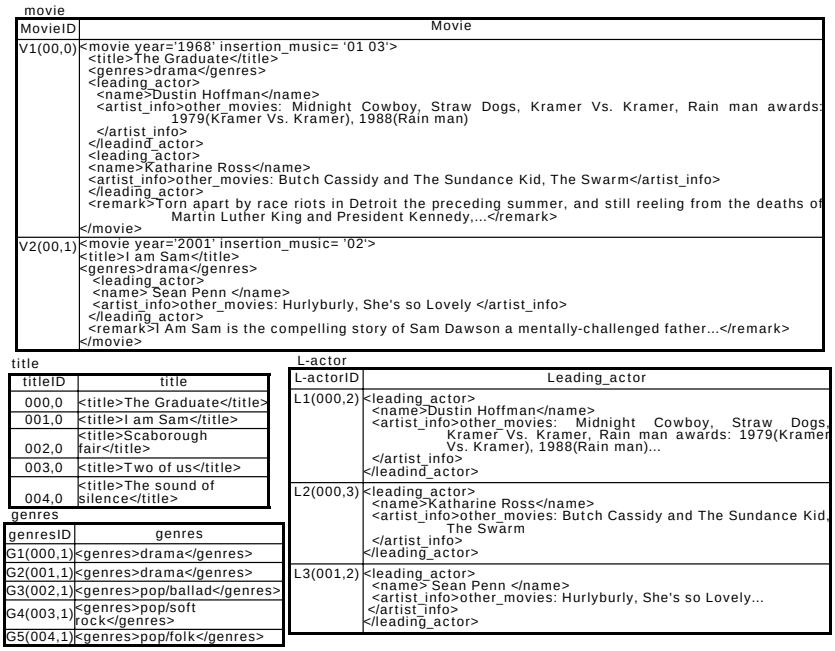

그림 4-4 이름 기반 저장기법

5. Performance Evaluation

우리는 먼저 XML 문서를 객체-관계형 데이터베이스에 저장하여 질의하기 위하여 필요한 비용 모델을 세웠고[29], 우리의 저장 기법이 지원할 수 있는 질의를 종류별로 분류하여 각 질의 마다 처리 속도를 기존 저장 기법과 비교 분석하였다[28]. 그러나 XML 데이터 모델과 기존 데이터 모델의 차이를 극복하기 위해서는 해결해야 할 여러 가지 문제점을 가지고 있기 때문에 질의 처리 속도만을 가지고 성능 평가를 하는 것은 부족하다고 판단하였다.

따라서 본 논문에서는 XML 문서를 기존 데이터 모델에 저장할 때 XML 문서의 고유 속성들을 충분히 지원할 수 있도록 저장한 것인지, XML 문서의 비정규적이고 불완전한 특징을 얼마나 잘 수용하는지, 저장된 데이터의 모습이 현존하는 데이터와 얼마나 자연스럽게

연계되어 사용될 수 있는지, 또한 XML의 속성을 살려 질의하기 위해 부가적인 데이터들이 얼마나 사용되었는지 등에 대해 총체적으로 정리하여 저장 기법에 대한 새로운 평가 기준을 마련하였다.

이러한 평가 기준을 토대로 반구조적 데이터를 관계형 데이터베이스에 저장하는 기법인 STORED[8], [9]에서 제안한 8가지 저장 기법 중 전반적으로 가장 성능이 좋다고 알려진 인라이닝을 가진 애트리뷰트 테이블(attribute table with inlining), XML-DBMS의 저장기법[26], [13]에서 제안한 세가지 저장 기법 중 조인을 필요로 하는 질의 중 전반적으로 성능이 좋다고 알려진 하이브리드 인라이닝(hybrid inlining), 그리고 본 논문에서 제안한 세 가지 저장 기법을 정성적으로 비교 분석하여 표 5-1과 같은 결과를 얻었다.

Table 5-1 Comparison of Storage Techniques

Consideration Items		STORED	Attribute table with inlining	XML-DBMS	Hybrid inlining	Structure-text Mixed ST	Structure-text separate ST	Name-based ST
Preserve XML properties?	Hierarchical information	×	○	△	△	○	○	○
	Order information	×	○	○	×	○	○	○
Process completely irregular and incomplete structures?	Optional elements	○	○	○	○	○	○	○
	*, ?, + type elements	△	○	△	○	○	○	○
	ANY type	△	○	△	○	○	○	○
	Multi-value attributes	△	○	△	×	○	○	○
Minimize data redundancies?		○	○	○	△	○	△	×
Minimize null values or overhead data?		△	×	×	×	△	○	○
Minimize table fragmentations?		○	×	○	○	○	×	×
Is reusable as relational databases seamlessly?		○	×	△	△	○	○	×
Minimize post-processing to return XML documents		△	×	△	△	△	△	○

이 표에서 보는 바와 같이 해당 사항이 얼마나 잘 지원되는지의 여부를 3등급으로 나누

어 전혀 지원되지 않는 경우는 ×로, 부분적으로 지원되는 경우는 △로, 전반적으로 잘 지원되고 있는 경우는 ○로 표현하였다.

XML 속성을 잘 유지하는가 여부는 관계형 테이블의 스키마에 계층적인 구조나 순서 정보를 직접적으로 포함하고 있지 않아 SQL 질의를 할 때 이러한 정보를 보존할 수 없는 경우를 ×로, 이러한 정보를 보존하기 위하여 기본키와 외래키를 이용하여 깊이가 깊은 질의의 경우 매 단계마다 순회나 조인을 해야 하는 경우 △로 표현하였다.

XML 구조가 비정구적이고 불완전한 성질을 갖는 것에 대한 처리 문제는 XML 인스턴스를 대상으로 직접 구조를 추출한다면 이러한 문제는 해결할 수 있다. 따라서 대개의 저장기법에서 이러한 것을 처리하기 위해 별도의 오버플로우 테이블을 사용하여 비효율적으로 테이블의 조각화를 야기시키는 경우는 △, 고려하지 않고 있는 경우는 ×로 표시하였다.

데이터의 중복 여부는 하나의 XML 문서를 저장하기 위해서 실제 데이터의 10%~30% 정도를 초과하는 경우 △, 그 이상을 초과하는 경우는 ×로 하였다.

널 값이나 오버헤드 데이터를 체크하는 항목은 XML이 가지고 있는 고유 속성 때문에 이를 위해 일반적인 관계형 데이터베이스에서는 잘 나타나지 않는 컬럼들이 부가적으로 사용되고 있는 정도와 이런 컬럼 때문에 발생되는 널 값의 양을 보는 것인데, 이러한 데이터 양이 전체 데이터의 양에서 20%~40%를 차지하고 있는 경우는 △, 그 이상 차지하고 있는 경우는 ×로 표현하였다.

테이블 조각화에 대한 사항은 XML 문서의 모습에 아주 많이 의존적인 것이라서 두 가지로만 분류하였다. 즉, XML 문서에 나타나는 엘리먼트 이름과 애트리뷰트 이름의 수만큼 테이블이 쪼개지는 현상이 생길 수 있는데 이러한 저장기법은 종류가 다른 항목에 대해서 모두 다른 엘리먼트 이름을 사용할 때 최악의 경우가 발생된다. 따라서 조각화를 발생시키는 저장 기법은 ×, 그렇지 않은 경우는 ○로 표현하였다.

현존하는 데이터와의 자연스런 연계가 가능한지를 보는 항목은 XML 문서가 가지고 있

는 고유 속성을 살려 질의하기 위하여 실제 데이터와 비교하여 얼마나 많은 부가적인 데이터를 포함하고 있는가를 보는 지표이다. 오버헤드 정보, 널 값, 테이블 조각화, 데이터 중복 등에서 2항목 이상 다른 것과 비교하여 효율성이 떨어지는 경우 ×, 이 중 어느 하나의 항목만 좋지 않은 경우는 △로 표시하였다.

마지막 항목은 질의의 결과를 XML 문서 형태로 반환하기 위해 테이블 사이의 컬럼들을 조합해야 하는 정도, 각 엘리먼트를 태깅하는 것, 애트리뷰트에 대해서는 원래의 문서가 가지고 있던 참조 정보나 엘리먼트에 내포되었던 모습을 복원해서 출력할 수 있는지의 여부를 보는 지표이다. 따라서 깊이가 깊게 내포된 엘리먼트를 출력하는 경우 그 깊이만큼 테이블 조인을 해야 하고, 한 엘리먼트에서 나가는 간선의 차수(degree)만큼 또 다시 조인해야 하는 경우를 ×, 이 두 가지 사항 중 어느 한가지만 해야 하는 경우는 △로 표시하였다. 특별히 이름 기반 저장 기법은 두 가지 모두 필요하지 않을 뿐 아니라 태깅이나 애트리뷰트에 대한 처리도 신경 쓸 필요가 없는 기법이다.

6. Conclusion

우리는 본 논문에서 XML 문서가 DTD를 가지고 있는 경우나 가지고 있지 않은 경우 모두에 대하여 구조를 추출하는 방법과 이것을 기반으로 객체-관계형 스키마에 사상할 수 있는 규칙을 정의하였다. 또한 XML 문서가 가지고 있는 고유의 속성인 계층적인 구조나 순서 정보와 같은 의미 기반의 정보를 보존하여 질의할 수 있고 계층 구조 기반 질의나 순서를 요구하는 질의 등, 다양한 종류의 질의를 지원할 수 있는 저장 기법을 개발하였다. 이들은 XML의 비정규적이고 불완전한 구조를 처리하는데 있어 기존의 저장 기법에 비해 효율적일 뿐 아니라, XML의 의미 정보를 보존하여 질의하기 위해 부가적으로 발생되는 오버헤드를 최소화할 수 있는 저장 기법이다.

References

[1] W3C Consortium, XML1.0.(Second Edition), W3C Recommendation, 6 Oct. 2000, available at http://www.w3.org.TR/2000/WD-xml-2e-20000814

[2] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, and F. Wattez, Querying XML Documents in Xyleme, SIGIR, 2000

[3] Z. G. Ives, A. Y. Levy, and D. S. Weld, Efficient Evaluation of Regular path Expressions on Streaming XML Data, Technical report, 2000

[4] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. SIGMOD Record, 26(3):54-66, September 1997

[5] S. Banerjee, Oracle XML DB, Oracle Corporation Technical White Paper Release 9.2, Jan. 2002

[6] S. Howlett and D. Jennings, SQL Server 2000 and XML: Developing XML-Enabled Data Solutions for the Web, MSDN magazine, Jan. 2002, available at http://msdn.microsoft.com/library/ default.asp?url= /msdnmag/issues/0800/sql2000/toc.asp

[7] IBM Corporation, DB2 XML Extender, IBM Corporation, 2000, available at http://www-4.ibm.com/

[8] A. Deutsch, M. Fernandez, and D. Suciu, Storing Semistructured Data with STORED, SIGMOD, Philadelphia, PN, 1999

[9] D. Florescu and D. Kossman, A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database, INRIA, Rocquencourt, France, 1999

[10] Q. Li and B.Moon, Indexing and Querying XML Data for Regular Path Expressions, VLDB, 2001

[11] F. Rizzolo and A. Mendlzon, Indexing XML Data with ToXin, 4th Int. Workshop on the Web and Database, 2001

[12] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. R. B. Lindsay, and H. Pirahesh, Efficiently publishing Relational Data as XML Documents, VLDB, 2000

[13] J. Shanmugasundaram, K. Tuffe, G. He, C. Zhang, D. DeWitt, and J. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities, VLDB, 1999

[14] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, Storing and Querying Ordered XML Using a Relational Database System, SIGMOD, 2002

[15] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, On Supporting Containment Queries in Relational Database Management Systems, SIGMOD, 2001

[16] S. Adler, A. Berglund, J. Caruso, S. Deach, T. Graham, P. Grosso, E. Gutentag, A. Milowski, S. Parnell, J. Richman, S. Zilles, Extensible Stylesheet Language (XSL) Version 1.0, W3C Proposed Recommendation Aug. 2001, available at http://www.w3.org/TR/xsl/

[17] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener, The Lorel Query Language for Semistructured Data, International Journal on Digital Libraries, Apr. 1997

[18] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, J. Siméon, XQuery 1.0: An XML Query Language, W3C Working Draft 16 Aug. 2002, available at http://www.w3.org/TR/xquery/

[19] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, XML-QL: A Query Language for XML, Submitted to the W3C 19 Aug. 1998, available at http://www.w3.org/TR/1998/NOTE-xml-ql-19980819

[20] J. Robie, XQL (XML Query Language), Aug. 1999, available at http://www.ibiblio.org/xql/xql-proposal.html

[21] W3C Consortium, XML Path Language (XPath) Version 1.0, W3C Recommendation 16 Nov. 1999, available at http://www.w3.org/TR/xpath.html

[22] M. Fernandez, W. -C. Tan, and D. Suciu, SilkRoute: Trading between relational and XML In Proc. of the WWW9, 2000

[23] T. Shimura, M. Yoshikawa, and S. Uemura, Storage and Retrieval of XML Documents Using Object-Relational Databases, DEXA, 1999.

[24] M. J. Carey, D. Florescu, Z. G. Lves, Y. Lu, and J. Shanmugasundaram, E. Shekita, and S. Subramannian, XPERANTO: Publishing Object-Relational Data as XML, In Proc. of the Int. Workshop on Web and Databases, 2000

[25] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld, Updating XML, SIGMOD, 2001

[26] R. Bourret, XML-DBMS: Middleware for Transferring Data between XML Documents and Relational Databases, available at http://www.rpbourret.com/xmldbms/

[27] K. Runapongsa and J. M. Patel, Storing and querying XML data in ORDBMSs, EDBT workshop, 2002

[28] 이월영, 용환승, 족보 기반 XML 문서 인덱싱 기법, 한국정보과학회 논문지, 심사중

[29] J. Kim, W. Lee, K. Lee. The Cost Model for XML Documents, In Proc. of ACS/IEEE International Conference on Computer Systems and Applications, Beirut, Lebanon, Jun. 2001.

[30] R.Murthy and S.Banerjee, XML Schema in Oracle XML DB, In Proc. Of the 29th VLDB Conference, Berlin, Germany, 2003