실체화된 XML 뷰☆
관계형 데이터베이스의 삭제에 따른 XML 뷰 파일의 갱신

Materialized XML View
XML View File Update according to the Deletion of Relational Databases

김 미 수* 나 영 국**
Gim, Misu Ra, Younggook

요 약

이차원 테이블 구조의 관계형 데이터베이스를 계층구조의 XML 문서로 표현하기 위한 방안으로 관계형 데이터베이스의 외래키 연관관계를 적용시킨다. 즉 한 투플의 외래키가 참조하는 다른 테이블의 주키를 외래키의 부모노드로 매핑 시키면 순서를 갖는 XML 계층구조를 표현할 수 있다. 그리고 이렇게 관계형 데이터베이스의 외래키 연관관계에 의하여 생성된 XML 뷰 파일은 기본이 되는 관계형 데이터베이스에 갱신이 발생하였을 경우 외래키 참조 무결성 조건을 만족하기 위하여 부작용이 나타나게 된다. 이는 생성 당시의 외래키 연관관계를 역으로 유추해 보면 어느 곳에서 부작용이 발생하게 될지 예측 가능하게 된다. 그러므로 이를 토대로 XML 뷰 파일에 미리 부작용까지 예상하여 XML 실체 뷰 파일을 최신의 것으로 유지한다.

Abstract

For the mapping of two dimensional table structures of relational databases to the hierarchical XML documents, we apply the foreign key relationships of relational databases. In other words, the primary key of another table referenced by a foreign key of one tuple is mapped to the parent of the foreign key. Then XML hierarchies with order are expressed. In addition, the XML view file generated by the foreign key mappings shows side effects in the case of the relational database update for the satisfaction of referential integrity constraint in the foreign key. Thus, by inferring the foreign key roles at the generation of XML hierarchies, we can anticipate where the side effects occur in the hierarchies. We keep the XML view files up-to-date by reflecting the side effects to the XML files at the update of the underlying relational databases.

☞ Key words : XML view, update, deletion, materialization, side effect, XML 뷰, 갱신, 삭제, 실체화, 부작용

1. 서 론

본 논문은 외래키 연관관계를 통해 기존 관계형 데이터베이스의 내용을 XML(eXtensible Markup Language) 실체 뷰 파일(materialized view file)로 저장한 후 이 XML 파일에 XQuery를 이용하여 질의하는 방법에 대한 연구이다. 이 방법은 관계형 데이터베이스에 XML 뷰를 정의하고 이 XML 뷰에 적용되는 XQuery를 XML 뷰 정의를 이용하여 SQL로 변환하는 기존의 연구[1]와 차별성을 갖는다. 첫째, 이 논문에서 제시하는 XML 파일은 뷰가 아니라 실제 데이터를 포함하는 파일이기 때문에 XML 뷰보다 성능에서 우수성을 갖는다. 둘째, 실제 파일이기 때문에 인터넷으로 파일전송이 가능하다. 셋째, 파일 전송이 가능하므로 클라이언트 측에서 쿼리(XQuery)가 가능하므로 서버의 부하를 줄일 수 있다. 그리하여 오랜 기간 동안 방대한 자

* 정 회 원 : 서울시립대학교 대학원 전자전기컴퓨터공학부 박사수료 miso4u@uos.ac.kr
** 정 회 원 : 서울시립대학교, 전자전기컴퓨터공학부 교수 ygra123@gmail.com
[2008/06/10 투고 - 2008/06/15 심사 - 2008/11/12 심사완료]
☆ 이 논문은 2005년도 서울시립대학교 학술연구조성비에 의하여 연구되었음

료가 축적된 관계형 데이터베이스의 자료를 그대로 이용하면서도 웹 상의 콘텐츠 교환에 탁월한 능력을 발휘하는 XML 문서에 대한 XQuery 질의어를 이용하여 인터넷을 기반으로 정보를 교환, 공유할 수 있게 된다.

많은 사람이 공유하며 삽입, 삭제, 수정 등의 끊임없는 갱신을 반영해야 하는 데이터베이스의 특성상 XML 실체 뷰 파일 역시 항상 최신의 상태로 유지해야 하는 것 또한 매우 중요한 사항이다. 기본이 되는(Base) 관계형 데이터베이스에 부분적인 갱신이 발생하였을 경우, 관계형 데이터베이스까지 접근 하지 않고 갱신에 대한 모든 영향을 예측하여 효율적으로 XML 실체 뷰를 유지할 수 있도록 해야 한다.[2] 이를 위하여 이차원 테이블 구조를 갖고 있는 기본이 되는 관계형 데이터베이스를 계층형 구조의 XML 문서로 매핑하여 두 구조 사이에 일관성을 갖도록 해야 한다.

기본이 되는 관계형 데이터베이스에 갱신이 발생했을 때 부작용이 발생하는 것은 관계형 데이터베이스와 XML의 표현 차이로 인한 특성상 피해가기 어려운 부분이라 하겠다. 즉 각 테이블의 속성을 최소화하며 외래키 참조에 의해 필요한 테이블을 조인하면서 서로의 관계로서 표현하는 것이 관계형 데이터베이스이기 때문이다. 기본이 되는 관계형 데이터베이스에서 하나의 투플을 삭제함에 따라 이와 매핑 하는 XML 엘리먼트(element)만 삭제하려 했는데 참조 무결성 제약조건에 의해 원하지 않았던 엘리먼트가 삭제되는 부작용이 발생할 가능성이 매우 높게 된다. 이 경우 부작용까지 반영하여 XML 실체 뷰 파일을 최신의 것으로 유지한다.

이를 위하여 본 논문에서는 기본이 되는 관계형 데이터베이스의 갱신에 의해 발생하는 모든 부작용을 찾아내는 알고리즘을 정의한다. 기본이 되는 관계형 데이터베이스에 갱신이 발생했을 때, XML 스키마 트리 상에서 부작용이 발생한 모든 곳에 부작용을 반영한다. 그리하여 XML 실체뷰 파일과 관계형 데이터베이스의 내용이 동기화된다.[3] [4]

본 논문의 장점은 부작용을 거부하고 무시하기보다는 오히려 부작용이 발생하는 곳을 미리 조사하여 기본이 되는 관계형 데이터베이스에 갱신이 발생했을 때 부작용까지도 반영하여 정확하게 XML 실체 뷰 파일로 갱신할 수 있게 한다.

2. 배 경

방대한 자료가 축적된 관계형 데이터베이스의 장점을 그대로 이용하면서도 웹상의 콘텐츠 교환에 탁월한 능력을 발휘하는 XML 문서에 대한 XQuery 질의어를 이용할 수 있도록 관계형 데이터베이스에 저장되어 있는 데이터를 XML 형태로 문서화하는 방법에 대한 연구가 있다. 본 연구와 유사하게 XML 뷰 기반으로 질의처리가 이루어지는 시스템의 대표적인 것으로 Silk Route 시스템과 XPERANTO 시스템, XTABLE 시스템이 있다. 이는 관계형 데이터베이스에 대한 XML 뷰를 정의하여 이를 기반으로 XQuery 질의를 수행하여 결과를 생성하는 방법에 관한 것이다.

Silk Route시스템은 관계형 데이터베이스 시스템과 XML 문서 사이에서, 순환함수와 순서에 의존적인 XML 데이터의 특징을 제외하고 XQuery 표현을 SQL로 변환하여 질의를 처리하는 미들웨어이다. 데이터베이스 관리자는 데이터베이스에 대한 가상 뷰를 정의하기 위해 자체 질의어인 RXL(Relational to XML)을 사용한다. RXL로 작성된 질의를 뷰 질의라 하고, 이 질의는 관계형 데이터를 XML 뷰로 변환한다. 이 방법을 사용하면 관계형 데이터 모델이 속성 집합의 값을 지원하지 않으므로 단일 XML 문서의 속성 값을 여러 개의 테이블에 나누어 저장하므로 많은 조인이 발생한다. 또한 질의 결과를 XML 문서로 변환하는 알고리즘의 구현이 복잡하다는 단점이 있다.[5]

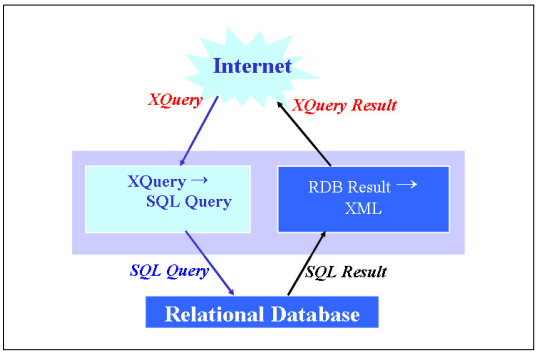
XPERANTO 시스템은 사용자 질의가 들어오면 먼저 파싱하여 XQGM(XML Query Graph Model)이라 불리는 중간 질의 표현으로 변환된다. 이는 기존의 관계형 데이터베이스가 아직 XQuery를 지원

하지 못하기 때문이다. 따라서 XQGM 변환은 XQuery를 SQL로 변환할 수 있도록 하고 대용량 데이터의 효율적인 질의 처리를 위해 질의 최적화를 할 수 있도록 함으로써 XML 데이터에 대한 질의 처리 문제를 해결하는 개념이라 할 수 있다. XPERANTO 시스템은 본 논문과 비슷하게 관계형 데이터베이스를 XML 문서화 하는 방식을 택하고 있지만 본 연구와 비교하면, XPERANTO 시스템은 중간 과정을 만들지 않고, 파이프라인 방식으로 수행되어 필요한 자료만을 반환하므로 많은 자료 중 극히 일부분의 자료에 대한 질의가 들어올 경우는 매우 유용하다. 하지만 기본이 되는 데이터베이스에 부분적인 갱신이 발생할 경우에도 똑같은 방법을 매번 되풀이해야 하므로 수행시간이 많이 소요되어 비효율적이다.1)

XTABLE 시스템은 XPERANTO 시스템의 연구 버전으로 관계형 데이터베이스의 XML 뷰를 생성·질의하고, 관계형 데이터베이스 시스템을 사용하여 XML 문서를 저장·질의하는 일반적인 틀을 제공한다.[6]

관계형 데이터베이스를 XML로 매핑 하는 본 연구와는 달리 XML을 관계형 데이터베이스 릴레이션으로 매핑 하는 연구가 활발히 진행되고 있는데 이 중에는 함수적 종속성을 반영하여 XML 데이터의 의미까지 고려한 연구도 있다.[7]

그림1은 웹상의 XQuery 질의를 관계형 데이터베이스의 SQL 질의로 변환하여 관계형 데이터베이스에 접근하고 이의 결과를 다시 XML 문서로 변환한 후 XQuery 질의를 수행한 결과를 반환하는 과정이다. 이 과정을 통하여 XML의 계층적인 데이터를 관계형 데이터베이스의 이차원 구조로 표현한다. 하지만 이 과정에서 관계형 데이터베이스 모델이 속성의 집합 값(multi-value)을 지원하지 않으므로 단일 XML 문서의 속성 값을 여러 개의 테이블에 나누어 저장해야 하므로 많은 조인이 발생하고 널값을 가진 칼럼이 많아져 낭비되는 공간이 많거나 테이블의 개수가 많아져 비효율적이다.[8] [9] 일부 XML 문서는 관계형 데이터베이스에 저장되지 않고 XML 콘텐츠가 관계형 데이터베이스에 삽입되기 전에 조각으로 나뉘거나 분해되는 경우도 발생하고 일부 XML 문서는 분해되지 않은 채 본래의 XML 형식으로 저장되기도 한다. 무엇보다도 질의 결과를 XML 문서로 변환하는 알고리즘의 구현이 복잡하다는 단점이 있다.[10] [11]



(그림 1) 관계형 데이터베이스와 XML 문서의 변환 과정

또한 여러 개의 서로 다른 상용 관계형 데이터베이스 시스템은 각각 자체 내에서 독립적으로 고유한 방법에 의해 데이터를 처리하기 때문에 인터넷을 통해 서로 다른 컴퓨터 시스템에서 수행되는 전자상거래의 모든 요구를 처리할 수 없고 계층적이거나 트리 구조의 데이터인 XML 문서 특성 그 자체가 관계형 데이터베이스에 또 다른 문제를 야기하기도 한다.

그러므로 기존의 관계형 데이터베이스에 저장되어 있는 데이터를 정보의 손실이 없이 XML 문서화 할 수 있다면 질의 처리 기법, 인덱싱, 최적화, 보안, 암호화 기술 등을 새롭게 개발할 필요 없이 기존의 관계형 데이터베이스의 기능을 사용하면서도 사용자는 구조화된 XML 문서에 대한 복잡한 질의를 수행하는 XQuery를 이용하여 질의를 실행할 수 있다.

또한 기본이 되는 관계형 데이터베이스에 부분적인 갱신이 발생하였을 경우, 관계형 데이터베이스까지 접근 하지 않고 효율적으로 XML 실체 뷰를 유지할 수 있는 방안에 관한 연구도 있다. 본

논문과 유사하지만 반대 개념의 관련 연구는 주어진 관계형 데이터베이스와 관계형 데이터베이스 상의 XML 뷰에 대한 문제, 즉 어떤 뷰 엘리먼트의 삭제에 대한 관계형 데이터베이스 투플의 정확한 변환을 찾는 방법을 제기한 것이다. 세 단계의 과정을 거치며, 생성자(generator) 개념을 이용하여 부작용이 발생하므로 삭제할 수 없는 경우를 제외하면서 어떤 뷰 엘리먼트의 삭제에 대한 관계형 데이터베이스 투플의 정확한 변환을 찾는 방법이다. 하지만 이 경우에 외래키의 참조 무결성 제약조건에 의하여 실제로 변환할 수 있는 경우가 많지 않다.[12] [13]

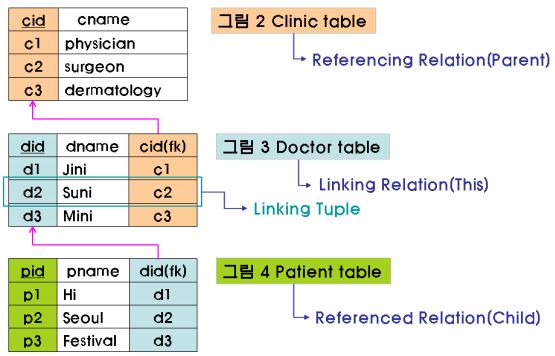
기본이 되는 관계형 데이터베이스에 갱신이 발생했을 때 부작용이 발생하는 것은 관계형 데이터베이스의 특성상 피해가기 어려운 부분이라 하겠다. 즉 각 테이블의 속성을 최소화하며 외래키 참조에 의해 필요한 테이블을 조인하면서 서로의 관계로서 표현하는 것이 관계형 데이터베이스이기 때문이다.[14] 기본이 되는 관계형 데이터베이스에서 하나의 투플을 삭제함에 따라 이와 매핑 하는 XML 엘리먼트만 삭제하려 했는데 참조 무결성 제약조건에 의해 원하지 않았던 엘리먼트가 삭제되는 부작용이 발생할 가능성이 매우 높게 된다. 이 경우 이 부작용까지 반영하여 XML 실체 뷰 파일을 최신의 것으로 유지하는 방안을 연구한다.

이차원 테이블 구조의 관계형 데이터베이스를 계층구조의 XML 문서로 표현하기 위한 방안으로 관계형 데이터베이스의 외래키 연관관계를 적용한다. 즉 한 투플의 외래키가 참조하는 다른 테이블의 주키를 외래키의 부모노드로 매핑시키면 순서를 갖는 XML 계층구조를 표현할 수 있다.13) 그리고 이렇게 관계형 데이터베이스의 외래키 연관관계에 의하여 생성된 XML 뷰 파일은 기본이 되는 관계형 데이터베이스에 갱신이 발생하였을 경우 외래키 참조 무결성 조건을 만족하기 위하여 부작용이 나타나게 된다. 이는 생성 당시의 외래키 연관관계를 역으로 유추해 보면 어느 곳에서 부작용이 발생하게 될지 예측 가능하게 된다. 그러므로 이를 토대로 미리 부작용까지 예상하여 XML 뷰 파일에 변환할 수 있다.

그러므로 본 연구의 장점은 부작용을 거부하고 무시하기보다는 오히려 부작용이 발생하는 곳을 미리 조사하여 기본이 되는 관계형 데이터베이스에 갱신이 발생했을 때 부작용까지도 반영하여 정확하게 변환할 수 있게 한다.

3. 정 의

그림5를 구성하는 그림2, 그림3, 그림4는 기본이 되는 관계형 데이터베이스 테이블이다. 이는 가상적 개념의 XML 실체 뷰를 생성하기 위하여 꼭 필요한 실제 테이블을 말한다.[15] 그림5에서 patient 테이블은 doctor 테이블을 참조하고, doctor 테이블은 clinic 테이블을 참조하는 외래키 참조관계를 갖고 있다. patient 테이블은 pid, pname, did인 세 개의 속성으로 구성되는데 이중 did는 외래키로써 그림3의 키인 did를 참조한다. doctor 테이블은 did, dname, cid인 세 개의 속성으로 구성되는데 이중 cid는 외래키로써 그림2의 키인 cid를 참조한다.



(그림 5) 각 테이블 간의 외래키 참조

각 테이블 간의 외래키 참조가 이와 같을 때, 그림3에서 Doctor.did=d2인 투플을 삭제하면 그림4에서 Patient.did=d2인 p2투플이 참조할 곳을 잃게 되므로 참조 무결성 제약조건을 위반하게 되고, 따라서 Patient.p2 투플까지 삭제해야 하는 부작용이

발생한다. 이처럼 외래키로 연관되어, 어떤 테이블의 임의의 투플 Ri.ti를 다른 테이블의 임의의 투플 Rj.tj가 참조할 때 Ri.ti를 Rj.tj의 생성자(Generater)라 하고 Ri의 키는 Rj의 외래키가 된다. 이 때 Ri는 Rj의 부모 릴레이션이 되고, 유사하게 Rj는 Ri의 자식 릴레이션이 되어 Ri.ti를 삭제할 경우 Rj.tj도 함께 삭제된다. 이는 다음과 같이 정의한다.

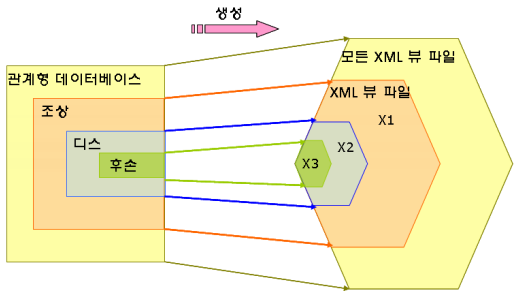
정의 1) Ri.k = Rj.fk이면 생성<Ri.ti> = Rj.tj이다.

정의 2) Ri.k = Rj.fk이고 Rj.k = Rn.fk이면 생성<Ri.ti> = Rn.tn이다.

기본이 되는 관계형 데이터베이스의 외래키 참조를 고려한 상황(그림5)에서 각 테이블에 있는 투플의 정의는 다음과 같다. 그림3의 Doctor.did=d2인 투플을 갱신하려고 할 때, 갱신하려는 투플을 링킹 투플(Linking Tuple) 또는 디스(this)라 하고, 디스가 속해 있는 릴레이션은 디스 릴레이션이라 한다. 이는 부모 릴레이션과 자식 릴레이션 사이의 연결고리 역할을 하고 있으므로 연결(linking) 릴레이션이라 하고 이의 한 투플이며 갱신하고자 하는 투플을 tlk라 한다. 이때 디스의 외래키가 참조하고 있는 디스의 부모 릴레이션을 참조(referencing) 릴레이션이라 하고 이의 한 투플이며 디스가 참조하는 투플을 trg라 한다. 유사하게 디스를 참조하는 디스의 자식 릴레이션은 참조되는(referenced) 릴레이션이라 하고 이의 한 투플이며 디스를 참조하는 투플을 trd라 한다. 이는 정의 2)에 의해 다음과 같이 표현할 수 있다.

생성<trg> = tlk이고, 생성<tlk> = trd이면, 생성<trg> = trd이다.

기본이 되는 관계형 데이터베이스는 외래키로 연관된 여러 개의 릴레이션으로 구성되어 있으며 이 중 하나 이상의 릴레이션이 결합하여 하나의 원하는 XML 뷰 파일을 생성할 수 있다. 그림6은 조상(ancestor), 디스(this), 후손(descendant)이 결합하여 하나의 XML 뷰 파일을 구성하는 예를 보이고 있다.



(그림 6) 관계형 데이터베이스로부터 XML 뷰 파일의 생성

x1은 조상 릴레이션으로부터, x2는 디스 릴레이션으로부터, x3는 후손 릴레이션으로부터 생성되는 XML 엘리먼트이다. 즉 디스 릴레이션의 외래키는 부모 릴레이션의 키가 된다. 마찬가지로 디스 릴레이션의 키가 자식 릴레이션의 외래키가 되는 것은 우리가 알고 있는 일반적인 의미와 같다. 관계형 데이터베이스는 평면구조로써 계층관계를 나타내지 못하지만 외래키-키 연관관계와 정의 1) 생성의 개념에서 생각하면 부모와 자식과의 순서를 정의할 수 있으므로 계층구조를 표현할 수 있고 이는 XML의 계층구조로 잘 매핑 된다.

관계형 데이터베이스 갱신에 따른 XML 뷰 파일의 갱신은 <P, △>로 표기한다. P는 XML 스키마 트리 상에서 갱신이 반영될 경로로써, 각 케이스 별로 그 위치가 이미 지정되어 있다. 때에 따라서는 관계형 데이터베이스의 갱신이 애초의 의도와 달리 원하지 않았던 부분을 갱신하는 부작용을 유발할 수도 있는데, 이 경우에는 부작용까지도 그대로 XML 뷰 파일에 반영한다. △는 관계형 데이터베이스가 갱신됨에 따라 XML 뷰 파일에 반영해야 할 XML 갱신이다. [16]

4. 부작용에 따른 케이스별 분류

4.1 분류 방법

그림7은 그림5에 대한 시스템 지정(default) XML 뷰로써 이는 단지 관계형 데이터베이스의 각 테이블에 대한 간단한 XML 표현, 즉 속성 이름과

속성 값을 나열할 뿐 XML이 갖고 있는 계층구조 정보를 표현하지 못한다. 하지만 관련연구인 XPERANTO 시스템을 이용하면 자동적으로 생성된 기본 XML 뷰를 이용하여 계층 구조를 나타내는 사용자 정의 뷰를 재정의하고 여기에 XQuery를 이용해 사용자 질의를 수행할 수 있다. 여기서 XML 뷰를 정의한 XQuery와 사용자 질의 XQuery는 각각 XQGM(XML Query Graph Model)이라는 중간 질의 표현으로 변환되는데 이는 XQuery를 SQL로 변환할 수 있도록 하고 대용량 데이터의 효율적인 질의 처리를 위해 질의 최적화를 할 수 있도록 함으로써 XML 데이터에 대한 질의 처리 문제를 해결하는 개념이라 할 수 있다. 결합된 XQGM은 반복적 순회 연산을 다른 연산과 결합시켜 불필요한 연산 작업을 최대한 줄여 중간 모델인 XQGM을 단순화 시키는 뷰 합성과정과 XML 단편을 생성하는 태그 정보에 의해 XML 문서를 생성하여 사용자에게 전달함으로써 XML의 계층 구조를 나타낸다. [17]

그림8(c)는 (b)의 스키마 트리를 XML 인스턴스로 표현한 것으로 그림5의 외래키 참조 흐름을 순리적으로 반영하고 있다. 경로는 루트(root)로부터 r/c1/d1/p1이다. 여기서 c1, d1, p1은 각각 Clinic.cid=c1, Doctor.did=d1, Patient.pid=p1인 투플로 외래키 참조에 의해 관계를 맺고 있다. 정의 2)에 의하면 c1 = trg, d1 = tlk, p1 = trd가 된다.

그러므로 그림8(c)에서 단말노드(leaf node) p1이나 p2의 삭제를 제외하고는, 삭제 시 후손 노드가 함께 삭제되므로 외래키 참조 무결성 제약조건을 검사해야 한다. 결과적으로 d1을 삭제할 경우 삭제하고자 하는 d1과 함께 원하지 않았던 p1도 삭제되고, c1을 삭제 시에는 c1의 모든 후손 노드인, d1과 p1까지 삭제되는 부작용이 발생하게 된다. 이처럼 관계형 데이터베이스에서 하나의 투플을 삭제할 경우에 참조 무결성 제약조건에 의해 다양한 부작용이 발생하듯이 관계형 데이터베이스를 기본으로 문서화한 XML 실체 뷰 파일에서도 원하지 않은 부작용이 발생한다. 특히 XML 문서는 각 엘리먼트 사이의 순서가 중요한 의미를 갖는 계층형 구조이므로 순서의 배열에 따른 다양한 경우의 부작용이 예상된다.

기본이 되는 관계형 데이터베이스의 한 투플이 삭제되었을 때 우리가 원하던 XML 뷰 엘리먼트의 삭제는 물론, 우리가 원하지 않았던 XML 뷰 엘리먼트까지 동시에 삭제하는 부작용이 발생한다. 본 논문은 tlk의 삭제가 XML 뷰 엘리먼트에 미치는 영향력을 분석하여, 부작용이 발생하는 케이스를 예를 들어 설명하고 삭제가 발생하는 모든 엘리먼트를 XML 실체 뷰 파일에 정확히 적용하는 방법을 연구한다.

연결 투플은 외래키 관계의 자식노드와 부모노드를 모두 갖는 투플이므로 tlk에서의 갱신에 대한 부작용을 탐지하면 외래키로 연관된 다른 여러 종류의 투플에 대한 갱신에도 적용할 수 있으므로 tlk를 삭제했을 경우, 각 케이스에 어떤 갱신이 발생하는지 조사한다. trg, tlk, trd의 순서나 위치에 대한 여러 케이스를 검토해 보면 디스의 조상 노드와 후손 노드에 대해서도 같은 케이스가 위나 아래로 파급적으로 전파되기 때문에 자식노드만 갖는 투플의 갱신이나 부모노드만 갖는 투플의 갱신에 대해서도 모두 적용할 수 있기 때문이다.

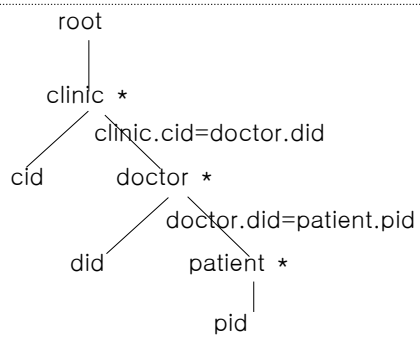
```
<DB>
  <clinic>
    <row>
      <cid>c1</cid>
       <cname>physician</cname>
    </row>
    <row>
      <cid>c2</cid>
       <cname>surgeon</cname>
    </row>
    <row>
      <cid>c3</cid>
       <cname>dermatology</cname>
    </row>
  </clinic>
  <doctor>
    <row>
      <did>d1</did>
      <dname>Jini</dname>
      <cid>c1</cid>
    </row>
```

```
      <row>
        <did>d2</did>
        <dname>Suni</dname>
        <cid>c2</cid>
      </row>
      <row>
        <did>d3</did>
        <dname>Mini</dname>
        <cid>c3</cid>
      </row>
    </doctor>
    <patient>
      <row>
        <pid>p1</pid>
        <pname>Hi</pname>
        <did>d1</did>
      </row>
      <row>
        <pid>p2</pid>
        <pname>Seoul</pname>
        <did>d2</did>
      </row>
      <row>
        <pid>p3</pid>
        <pname>Festival</pname>
        <did>d3</did>
      </row>
    </patient>
</DB>
```

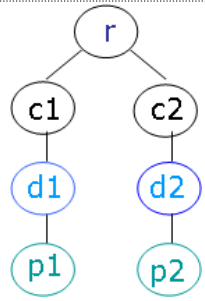
(그림 7) 그림5의 사용자지정 XML View

```
<root>
 FOR $c IN document( default.xml ) /clinic/row
 RETURN{
   <clinic>
     $c/cid,
     FOR $d IN document( default.xml )/doctor/row
     WHERE $c/cid=$d/did
     RETURN{
   <doctor>
      $d/did,
      FOR $p IN document( default.xml )/patient/row
      WHERE $p/did=$d/did
      RETURN{
     <patient>
       $p/pid
     </patient>}
   </doctor> }
   </clinic> }
</root>
```

(a) 그림5에 대한 XQuery



(b) 그림5에 대한 스키마 트리



(c) 그림5에 대한 XML 인스턴스 트리

(그림 8) XQuery/스키마 트리/XML 인스턴스 트리

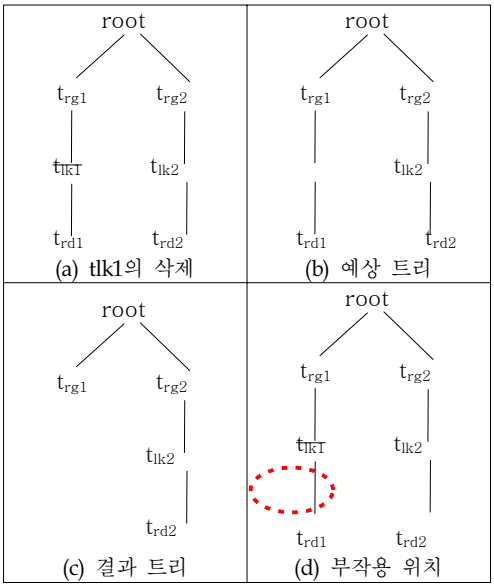
결과적으로 tlk의 자식노드가 있을 때, 그림5의 외래키 참조 단계인 trg/tlk/trd의 접근경로가 아니고 이의 순서가 바뀔 때와 tlk의 자식노드가 tlk의 형제노드나 또 다른 노드에 사용된 경우에 부작용이 발생한다.

4.2 케이스 별 분류 및 XML 뷰 갱신 언어

1) 케이스1

가장 일반적이고 빈번하게 발생할 수 있는 부작용으로 tlk를 참조하는 투플이 있는 경우이다. 그림9에서 trg1/ tlk1/ trd1의 순서로 구성되어, 기본이 되는 관계형 데이터베이스의 외래키 참조와 같은 구성을 갖는다. 즉, 삭제하고자 하는 tlk1은 trg1을 참조하고 trd1은 tlk1을 참조하는 방식으로 순리적인 짜임새를 갖는다.

tlk1을 삭제한 후의 결과로 그림9-(b)를 예상했는데 결과는 그림9-(c)가 되었다. 이는 tlk1을 참조하는 trd1이 존재하므로 삭제하고자 하는 tlk1과 함께 tlk1의 후손 노드가 함께 삭제되는 부작용이 발생하기 때문이다. 그러므로 XML 뷰 파일 갱신 시에는 이를 반영하여 tlk1과 함께 trd1도 삭제하여 그림9-(c)의 결과 트리로 갱신하는 것이 정확한 변환이다. tlk1은 tlk1이 삭제되었음을 의미한다.



(그림 9) 케이스1의 XML 인스턴스 트리

케이스1의 인스턴스 트리인 그림9-(a)에서 tlk1을 삭제할 때, XML 뷰 파일의 갱신은 detn<P, △>로 표기한다. n은 tlk1을 삭제함으로써 갱신해야 할 곳의 계수이다. tlk1을 삭제할 경우, P = root / trg1 / tlk1이고 △ = "" 로 하고 det1<P, △>로 표기한다. 그림9-(d)에서 보듯이 trd1은 삭제하고자 했던 엘리먼트가 아니지만 tlk1의 자식이기 때문에 함께 삭제되는 부작용이 발생한다. 여기서는 det2<P, △>로 표기하고 이것까지 삭제해야 정확한 변환이 되므로 케이스1은 두 곳의 갱신이 발생한다. det2<P, △>는 부작용이다.

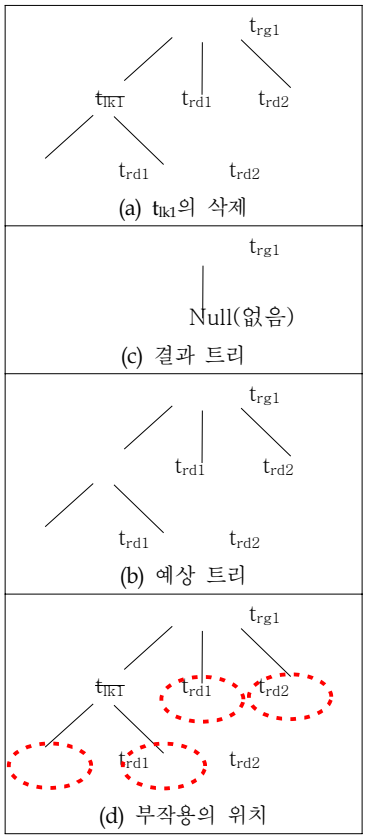
det1<P, △> = det1<root / trg1 / tlk1 , "">

det2<P, △> = det2<root / trg1 / tlk1 / trd1 , "">

2) 케이스2

trd1, trd2가 tlk1의 자식노드이면서 형제노드로도 사용된 경우이다.

tlk1을 삭제하면 tlk1을 참조하는 trd1, trd2 즉 tlk1의 자식노드가 함께 삭제되는 것은 물론이고, trd1, trd2가 tlk1의 형제노드로 사용되었으므로 이들 역시 삭제되어야 하고 이는 원하지 않았던 부작용이다. 우리는 tlk1을 삭제한 후의 결과로 그림10-(b)를 예상 했었는데 결과는 그림10-(c)가 되어 모든 노드가 삭제된다. 그러므로 기본이 되는 관계형 데이터베이스에서 tlk1을 삭제하면, tlk1의 자식노드인 trd1, trd2와 tlk1의 형제노드인 trd1, trd2도 동시에 삭제된다는 것을 XML 뷰 파일 갱신에 표현해야 한다.



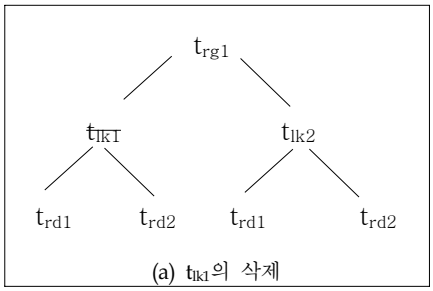
(그림 10) 케이스2의 XML 인스턴스 트리

그림10-(a)에서 tlk1을 삭제할 경우, 케이스1의 결과에 의해서 자식노드인 trd1과 trd2가 삭제되리라는 것은 이미 알고 있다. 하지만 trd1과 trd2가 tlk1의 형제노드로도 사용되므로 tlk1의 형제노드 역시 삭제되는 부작용이 발생한다. 우리가 원하지 않은 동작(event)이지만 케이스2에 대한 XML 뷰 파일의 갱신에 반영해야 한다. 그러므로 tlk1의 삭제로 인해 다섯 곳의 갱신이 발생하고, 이 중 det1<P, △>을 제외한 det2<P, △>, det3<P, △>, det4<P, △>, det5<P, △>는 부작용이다.

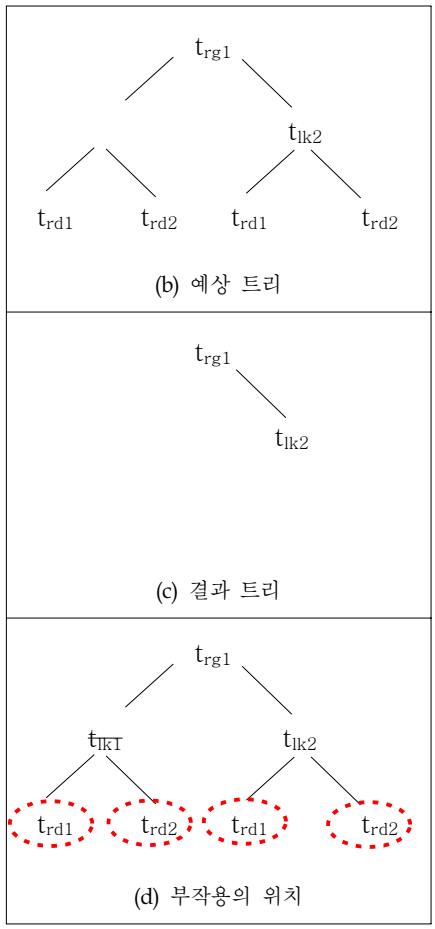
det1<P, △> = det1<trg1 / tlk1 , “”>
det2<P, △> = det2<trg1 / tlk1 / trd1 , “”>
det3<P, △> = det3<trg1 / tlk1 / trd2 , “”>
det4<P, △> = det4<trg1 / trd1 , “”>
det5<P, △> = det5<trg1 / trd2 , “”>

3) 케이스3

tlk와 이의 형제노드가 동일한 자식노드를 가질 때, tlk의 삭제는 형제노드의 자식노드까지 영향을 미친다. 그림11에서 trd1, trd2가 tlk1의 자식노드이면서 형제노드의 자식노드로 동시에 사용된 경우로써, tlk1을 삭제하면 tlk1을 참조하는 trd1, trd2, 즉 tlk1의 자식노드가 함께 삭제되는 것은 물론이고, 아울러 tlk1의 형제노드인 tlk2의 자식노드로 사용된 trd1, trd2도 역시 삭제되는 부작용이 발생한다. 그러므로 기본이 되는 관계형 데이터베이스에서 tlk1을 삭제하면, tlk1의 자식노드인 trd1, trd2와 형제노드의 자식노드인 trd1, trd2도 동시에 삭제된다는 것을 XML 뷰 파일 갱신에 적용해야 한다.



(a) t_{lk1}의 삭제



(그림 11) 케이스3의 인스턴스 트리

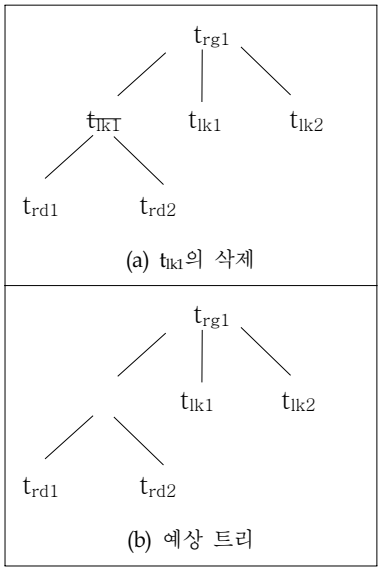
그림11-(a)에서 tlk1을 삭제할 경우, 실제 결과는 tlk1를 제외한 모든 노드가 삭제된 그림11-(c)가 된다. tlk1의 자식노드인 trd1과 trd2가 tlk1의 형제노드인 tlk2의 자식노드로도 사용되었기 때문에 tlk2의 자식노드 역시 삭제되는 부작용이 발생한 것이다. 우리가 원하던 바가 아니지만 케이스3에 대한 XML 뷰 파일의 갱신에 반영해야 한다. tlk1의 삭제로 인해 다섯 곳의 갱신이 발생하고 이 중 네 곳, 즉 det2<P, △>, det3<P, △>, det4<P, △>, det5<P, △>는 부작용이다.

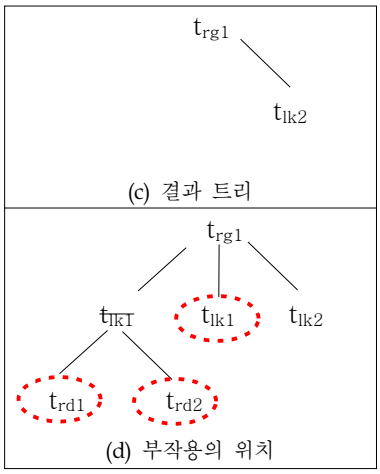
det1<P, △> = det1<trg1 / tlk1 , “”>
det2<P, △> = det2<trg1 / tlk1 / trd1 , “”>
det3<P, △> = det3<trg1 / tlk1 / trd2 , “”>
det4<P, △> = det4<trg1 / tlk2 / trd1 , “”>
det5<P, △> = det5<trg1 / tlk2 / trd2 , “”>

4) 케이스4

디스이며 동시에 형제노드인 케이스로 tlk가 tlk의 형제노드로 다시 사용되는 경우이다. 그림12의 XML 인스턴스 트리 모형에서 trd1, trd2라는 자식노드를 가진 tlk1과 자식노드가 없는 tlk1이 형제노드로 공존하는 경우이다.

tlk1을 삭제하면 tlk1을 참조하는 trd1, trd2, 즉 tlk1의 자식노드가 삭제됨과 동시에 형제노드인 tlk1도 역시 삭제된다. 이 경우 애초의 의도는 그림 12-(b)처럼 자식노드를 가진 tlk1만 삭제하고 형제노드인 tlk1은 삭제할 생각이 아니었는데 그림 12-(c)와 같이 형제노드인 tlk2만 남고 나머지 노드가 모두 삭제되는 부작용이 발생한다. 이 역시 XML 뷰 파일 갱신에 적용해야 한다.

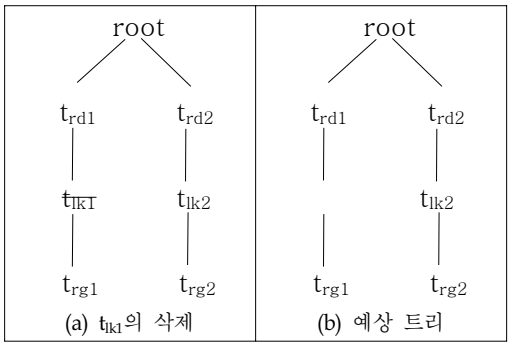


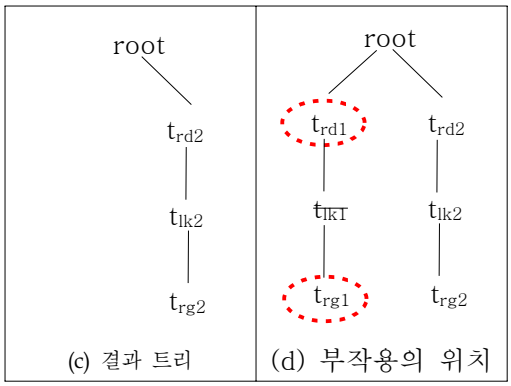


(그림 12) 케이스4의 인스턴스 트리

그림12-(a)에서 tlk1을 삭제할 경우, 자식노드인 trd1과 trd2가 삭제되는 것은 케이스2에서 이미 보았으므로 그림12-(b)의 예상트리에서 tlk1의 자식노드인 trd1과 trd2가 삭제될 것이라는 예상이 가능하다. 하지만 이 경우에는 tlk1이 형제노드로 다시 사용되고 있기 때문에 예상치 않은 한 곳의 부작용인 det4<P, △>도 반영하여야 한다. 케이스4의 부작용은 det2<P, △>, det3<P, △>, det4<P, △>로 표시하고 XML 뷰 파일의 갱신에 반영해야 한다.

det1<P, △> = det1<trg1 / tlk1 , “”>
det2<P, △> = det2<trg1 / tlk1 / trd1 , “”>
det3<P, △> = det3<trg1 / tlk1 / trd2 , “”>
det4<P, △> = det4<trg1 / tlk1 , “”>





(그림 13) 케이스5의 인스턴스 트리

5) 케이스5

그림13에서 보면, 그림5의 기본이 되는 관계형 데이터베이스 외래키 참조와 참조순서가 반대, 즉 자식노드인 trd가 tlk의 부모노드로 사용되어 참조의 주체와 객체가 뒤바뀐 형태이다. 이와 같이 기본 케이스와 반대인 trd1/ tlk1/ trg1의 순서로 구성되는 경우에는 많은 부작용이 발생한다.

케이스2, 케이스3, 케이스4의 경우에도 원하지 않았던 형제노드가 삭제되는 부작용이 있었지만 이들의 경우는 trg가 부모노드에 위치한다는 전제가 있었으므로 부모노드까지 부작용이 전파되지는 않았다. 하지만 이 경우는 trg1/ tlk1/ trd1의 순서로 구성된 기본이 되는 관계형 데이터베이스에서 tlk1이 삭제되면 함께 삭제될 자식노드인 trd1이 부모노드에 존재하므로 결과적으로 모든 노드가 삭제된다. 이를 다른 각도에서 보면 스키마 트리상에서 tlk1이 참조하고 있는 trg1이 자식노드에 위치하므로 trg1을 참조하는 노드, tlk1이 삭제된다는 의미이므로 같은 결과가 되어 사실상 모든 후손노드가 삭제되어 가장 많은 부작용이 예상된다.

기본이 되는 관계형 데이터베이스와 참조 순서가 반대로 된 예로써, 많은 부작용을 동반한다. 그림13-(a)에서 tlk1을 삭제할 경우, tlk1이 참조하는 trg1이 삭제되는데 trg1은 관계형 데이터베이스에서 생성<trg1> = tlk1이다. 그러므로 tlk1이 참조해야할 trg1투플이 삭제됨으로써 연속적으로 trg1을 참조하는 노드가 삭제되어야 한다. 그러므로 연속삭제가 발생하여 스키마 노드 상에서 trg1의 모든 후손노드가 삭제된다. 이는 또한, tlk1은 항상 중간에 위치하는 연결 투플이므로 trd1이 부모노드에 온다면 trg1은 자식이 되므로 생성<tlk1> = trd1인 trd1이 인스턴스 트리상의 부모노드에 위치하므로 모든 노드가 삭제됨을 알 수 있다. 그러므로 tlk1의 부모나 자식노드 중 어느 것을 체크해도 같은 결과가 된다. 여기서는 부모노드를 체크하여 반환한 결과가 다음과 같다. 이 중 det2<P, △>와 det3<P, △>는 부작용이다.

det1<P, △> = det1<root / trd1 / tlk1 , “”>
det2<P, △> = det2<root / trd1 / tlk1 / trg1 , “”>
det3<P, △> = det3<root / trd1 , “”>

5. 삭제 알고리즘 (Deletion Algorithm)

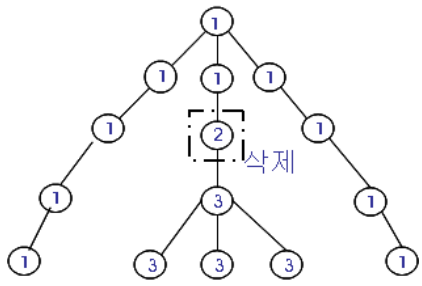
5.1 삭제 알고리즘

상위계층 릴레이션이 하위계층 릴레이션의 생성에 관여하므로 상위계층 릴레이션에 갱신이 발생한다면 하위계층 릴레이션으로 전파되게 된다. 즉, 조상 릴레이션을 삭제하였을 때, 조상 릴레이션의 키를 외래키로 하는 디스 릴레이션과 디스 릴레이션의 키를 외래키로 하는 후손 릴레이션이 같이 삭제된다. 단지 임의의 릴레이션의 한 투플을 삭제하려 했는데 이 투플과 외래키-키 연관관계가 있는 다른 릴레이션의 투플까지 삭제하는 결과가 된 것이다. 우리가 원하지 않았던 부작용이 발생한다고 해서 어느 한 투플을 삭제하지 못한다면, 정규화의 과정을 거치며 많은 릴레이션으로 분리되어 저장되고 이들이 외래키-키 연관관계로 다시 결합되며 새로운 관계를 갖게 되는 관계형 데이터베이스의 특성상 실제로 삭제할 수 있는 투플은 극히 제한적일 수밖에 없다.

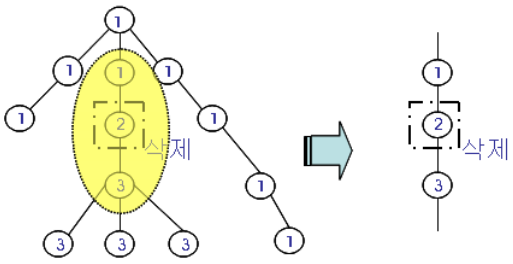
그러므로 처음부터 원했든, 원하지 않았든 하나

의 투플을 삭제함으로서 발생할 수 있는 모든 파급효과를 예측하여 발생할 수 있는 모든 갱신을 XML 뷰 파일에 반영하여 각 케이스 별로 정확한 갱신을 얻고자 함이 본 연구의 핵심이다. 사실상 디스가 최하위 레벨(terminal node)이 아닌 경우는 항상 부작용이 발생하게 된다. 케이스를 판별하는 것은 본 연구에 포함하지 않는다. 또한 각 케이스는 구조(syntax)만을 고려하고 의미(semantics)는 고려하지 않는다. 또한 관계형 데이터베이스의 삭제는 하나의 케이스에만 영향을 미친다.

그림14는 일반적인 스키마 트리를 보여준다. ②가 단말노드에 있는 경우는 ②의 삭제로 인한 부작용이 발생하지 않으므로 ②가 중간 레벨에 위치하는 경우에 대하여 검토한다. 이는 세 그룹으로 나눌 수 있는데 ①그룹은 스키마 노드가 디스의 후손이 아닌 노드이다. ②노드는 삭제하고자 하는 디스, 즉 tlk노드이다. ③그룹은 생성<디스> 또는 생성<디스>의 후손이다. 이 세 그룹은 어떤 겹침(Overlap)도 없이 모든 뷰 엘리먼트를 포함(cover)한다.



(그림 14) 일반 스키마 트리 구조



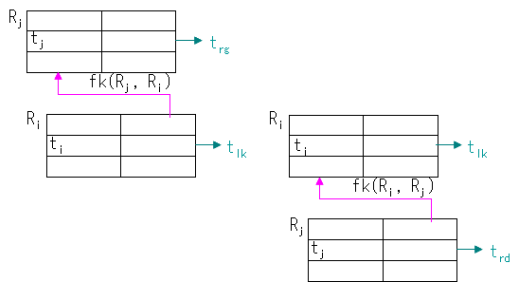
(그림 15) 스키마 트리의 삼원 연결 부분

그러므로 ②를 삭제했을 때 ①그룹과 ③그룹에 미치는 영향을 검사하면 모든 부작용을 탐지할 수 있다. ①그룹과 ③그룹은 ②의 부모와 자식이 계속적인 파급효과로서 성립되는 관계이므로 그림15 스키마 트리의 삼원 연결 부분에 대하여 연구한다. ②는 부모인 ①과 자식인 ③을 연결하는 부분이다. 이들 ①, ②, ③을 관계형 데이터베이스 내의 릴레이션이라 할 때 두 릴레이션을 서로 연결하는 특별한 목적의 속성을 관계속성이라 부르고 이를 fk로 표시한다. 두 개의 릴레이션 ①과 ②가 관계속성 fk로 연결되고 연결조건이 ①.k=②.fk 라면, 그 연결을 fk<①, ②>로 표시한다.[18] 여기서 ①.k는 ①의 키, ②.fk는 ②의 외래키이다. 마찬가지로 ①과 ②의 인스턴스 r1과 r2가 fk로 연결되고 연결조건이 r1.k=r2.fk 라면, 그 연결을 fk<r1, r2>로 표시한다. 이는 스키마 트리 구조상에서 r1이 r2의 부모가 됨을 의미하며 생성<r1>=r2이다. ②와 ③의 관계는 ①과 ②관계에서 상대적인 위치가 변경되었을 뿐 동일한 개념이므로 이를 그림15와 같은 삼원 연결부분으로 확장해보자.[19]

삼원 연결이란 ①이 ②의 외래키이고 ②가 ③의 외래키가 되어 그림16과 같은 구조를 가질 때이다. 기본이 되는 관계형 데이터베이스의 한 릴레이션을 Ri라 하고 이의 한 투플을 ti라 하자. 이때 어떤 릴레이션 Rj가 있는데 fk<Rj, Ri>의 관계가 있다면 Rj를 Rrg라 하고 fk<tj, ti>의 관계가 있는 투플 tj를 trg라 한다. 반대로 어떤 릴레이션 Rj가 fk<Ri, Rj>의 관계를 갖는다면 Rj를 Rrd라 하고 fk<ti, tj>의 관계가 있는 투플 tj를 trd라고 한다. 즉 Rrg와 Ri의 인스턴스 trg와 ti가 fk로 연결될 때 연결조건이 trg.k=ti.fk이고, Ri와 Rrd의 인스턴스 ti와 trd가 fk로 연결되고 그 연결조건이 ti.k=trd.fk 임을 만족하는 경우를 삼원 연결이라 한다. 삭제의 영향이 부모와 자식으로 전파될 수 있는 Ri의 한 투플을 삭제해야 하므로 Ri를 Rlk라 하고 삭제하려는 투플 ti는 tlk라고 하고 그 연결을 fk3<trg, tlk, trd>로 표시한다.[20]

인스턴스 트리 상에서 tlk의 자식노드는 tlk와 함

께 삭제되는 부작용이 발생함은 이미 알고 있다. 하지만 인스턴스 트리 상에서는 자식노드지만 스키마 트리 상에서는 tlk의 생성에 관여하는 노드 즉, fk3<trg, tlk, trd>일 경우에는 trg를 tlk로 하고 삭제리스트에 삽입하고 루트로 돌아가서 인스턴스 트리를 다시 순회하며 삭제 리스트에 있는 노드를 삭제하는 과정을 반복해야 한다. 하지만 모든 노드를 순회하며 fk3<trg, tlk, trd>를 체크하는 것은 매우 비효율적으로 소모적인 연산을 수반하게 된다. 그러므로 tlk가 있는 노드를 먼저 탐색한 후 이의 조상에 tlk의 자식인 trd가 있는지 탐색한 후 있다면, tlk의 부모와 후손을 삭제리스트에 넣고 나머지 노드를 순회하며 삭제리스트에 있는 노드인지 검사하면 불필요한 연산을 줄일 수 있다. 본 연구에서는 전체 트리를 삼원 연결 구조로 분할하여 전체 트리가 아닌 부모와 자식을 갖는 서브 트리로 분할하여 본 알고리즘을 적용하는 분할정복방법을 사용한다. 5가지 케이스에서 보듯이 tlk는 부모와 자식노드를 연결한다.



(그림 16) 각 투플의 관계속성도

이를 정리해 보면 삭제 과정은 부모와 디스와 자식노드를 체크하여 삭제노드 리스트와 삭제된 노드의 경로리스트를 만든다. 그리고 디스의 형제노드 등 나머지 노드를 순회하여 삭제노드 리스트에 속해 있는 노드가 발견되면 그 엘리먼트를 삭제한다. 이중, 디스인 하나의 tlk1을 제외한 나머지 삭제노드는 부작용이다.

노드는 깊이우선탐색(DFS: Depth First Search) 알고리즘으로 순회하고 기본이 되는 관계형 데이터베이스의 스키마 노드 상에서 삭제하려는 투플과 그의 부모와 자식노드 즉, fk3<trg1, tlk1, trd1>를 기억한다.

첫째, 부모노드를 체크하여 부모노드가 trg1일 경우는 갱신이 발생하지 않는다. 하지만 trd1일 경우, 이는 스키마 트리 상에서는 tlk1의 자식이므로 tlk1이 삭제되면 함께 삭제되는 부작용이 발생하므로 그 가지의 모든 노드를 삭제하고 삭제리스트에 삽입한다.

둘째, 디스노드를 체크한다. tlk1을 삭제한다.

셋째, 자식노드를 체크하여 삭제한다. 특히 tlk1의 자식노드가 trg1일 경우는 삼원 연결부분에 의해 부모노드에 trd1이 온 것과 같은 의미이므로 tlk1의 부모노드와 함께 그 가지의 모든 노드를 삭제하고 삭제리스트에 삽입한다.

넷째, 디스의 형제노드를 체크한다. 삭제노드 리스트에 속해 있는 노드가 발견되면 역시 이를 삭제노드 리스트와 경로리스트에 삽입하고 엘리먼트를 삭제한다.

이의 알고리즘을 의사코드(Pseudo Code)로 표현하면 다음과 같다.

```
/* (1) 기본이 되는 관계형 데이터베이스의 스키마 노드
   상에서 삭제하려는 투플(tlk1)과 그의 부모와 자식노
   드 즉, fk3<trg1, tlk1, trd1>를 기억한다.*/
fk[3]={"trg1", "tlk1", "trd1"}

// (2) XML instance tree를 깊이우선탐색으로 순회한다.
depth_first_search(XML_instance_tree)

while(all_node) do
  // (3) tlk1의 부모노드가 trd1이라면 케이스5가 되어
     모든 노드를 삭제하고, 삭제리스트와 경로리스트
     에 추가한 후 형제노드에 대해 깊이우선탐색으
     로 순회한다. 이 중 tlk1을 제외한 노드는 부작용
     이다.
  {if parent_node_of_tlk1 == trd1 then
     {deletion_list ← all_node_of_this_subtree
      path_list ← all_node_of_this_subtree
      //          sibling_node에          대해
   depth_first_search(XML_instance_tree)를 수행한다.
      call sibling_check
     }
```

```
        // (4) 방문노드가 tlk1이라면 tlk1을 삭제하고 삭제
            리스트와 경로리스트에 추가한다. 케이스1,
            2, 3, 4가 된다.
         elseif visiting_node == tlk1 then
              {deletion_list ← tlk1
               path_list ← tlk1
              }

              // (5)
              elseif visiting_node == child_node_of_tlk1
    then
                  {deletion_list ←
    all_node_of_this_subtree
                   path_list ← all_node_of_this_subtree
                   if child_node == trg1 then
                      {deletion_list ←
    parent_node_of_tlk1
                       path_list ← parent_node_of_tlk1
                      }

    call sibling_check      // (6)

    if other_node == item_of_deletion_list then    // (7)
        {deletion_list ← other_node
         path_list ← other_node
        }

    return (deletion_list, path_list)     // (8)

    // 형제노드가 삭제리스트에 존재한다면 케이스 2, 3, 4
       가 된다.
    sibling_check
        if sibling_node_of_tlk1 == item_of_deletion_list
then
        {deletion_list ← sibling_node
         path_list ← sibling_node
        }
    }
```

5.2 적용 사례

위의 알고리즘은 케이스 별로 정리되어야 한다. 즉, 케이스1에서 케이스5까지의 입력이 있을 경우, 4.2에 언급된 각 케이스 별 XML 뷰 갱신의 결과인 detn<P, △>를 출력하는 예를 보인다. 아래의 각 케이스에 있는 (3), (4) 등은 의사코드 내의 번호를 의미한다.

깊이 우선 탐색을 이용하고, 모든 XML 인스턴스 트리를 순회하며 스키마 트리상의 외래키-키 관계인 fk3=[trg1, tlk1, trd1]를 기억하고 있다는 것을 전제조건으로 한다.

1) 케이스1

(3) 부모노드가 trg1이므로 갱신 없이 (4)를 실행한다.
(4) tlk1이므로 삭제 표시를 하고 경로리스트에 저장한 후 다음 노드를 순회한다.
(5) tlk1의 자식이므로 삭제 표시를 하고 경로를 기억한 후 trg1인지 체크하였지만 trd1이므로 root의 오른쪽 가지를 탐색한다.
(6) trg2는 삭제리스트에 존재하지 않으므로 자식노드인 tlk2를 순회하지만 역시 삭제리스트에 존재하지 않으므로 갱신 없이 다음 노드인 trd2를 순회한다. 역시 삭제리스트에 없는 노드이므로 갱신이 없다.
(7) 이외의 다른 노드가 존재하지 않는다.
(8) 이의 결과로 다음의 삭제리스트가 출력된다.(표1)

(표 1) 케이스1의 삭제리스트 및 경로 리스트

순서	순회노드	삭제 여부	경로	부작용 여부
1	trg1	no		no
2	tlk1	yes	trg1/ tlk1	no
3	trd1	yes	trg1/ tlk1/ trd1	yes
4	trg2	no		no
5	tlk2	no		no
6	trd2	no		no

2) 케이스2

케이스1과 유사한 과정을 거쳐 다음의 삭제리스트가 출력된다.(표2)

(표 2) 케이스2의 삭제리스트 및 경로 리스트

순서	순회노드	삭제 여부	경로	부작용 여부
1	trg1	no		no
2	tlk1	yes	trg1/ tlk1	no
3	trd1	yes	trg1/ tlk1/ trd1	yes
4	trd2	yes	trg1/ tlk1/ trd2	yes
5	trd1	yes	trg1/ trd1	yes
6	trd2	yes	trg1/ trd2	yes

3) 케이스3

케이스1과 유사한 과정을 거쳐 다음의 삭제리스트가 출력된다.(표3)

(표 3) 케이스3의 삭제리스트 및 경로 리스트

순서	순회노드	삭제 여부	경로	부작용 여부
1	trg1	no		no
2	tlk1	yes	trg1/ tlk1	no
3	trd1	yes	trg1/ tlk1/ trd1	yes
4	trd2	yes	trg1/ tlk1/ trd2	yes
5	tlk2	no		no
6	trd1	yes	trg1/ trd1	yes
7	trd2	yes	trg1/ trd2	yes

4) 케이스4

케이스1과 유사한 과정을 거쳐 다음의 삭제리스트가 출력된다.(표4)

(표 4) 케이스4의 삭제리스트 및 경로 리스트

순서	순회노드	삭제 여부	경로	부작용 여부
1	trg1	no		no
2	tlk1	yes	trg1/ tlk1	no
3	trd1	yes	trg1/ tlk1/ trd1	yes
4	trd2	yes	trg1/ tlk1/ trd2	yes
5	tlk1	yes	trg1/ tlk1	yes
6	tlk2	no		no

5) 케이스5

⑶ 부모노드가 trd1이므로 이 가지의 모든 노드에 삭제 표시를 하고 경로를 기억하고 ⑹을 실행한다.

⑹ trd2는 삭제리스트에 존재하지 않으므로 갱신 없이 자식노드인 tlk2를 순회하지만 역시 삭제리스트에 존재하지 않으므로 갱신 없이 다음 노드인 trg2를 순회한다. 역시 삭제리스트에 없는 노드이므로 갱신이 없다.

⑺ 이외의 다른 노드가 존재하지 않는다.

⑻ 이의 결과로 다음의 삭제리스트가 출력된다.(표5)

(표 5) 케이스5의 삭제리스트 및 경로 리스트

순서	순회노드	삭제 여부	경로	부작용 여부
1	trd1	yes	trd1	yes
2	tlk1	yes	trd1/ tlk1	no
3	trg1	yes	trd1/ tlk1/ trg1	yes
4	trd2	no		no
5	tlk2	no		no
6	trg2	no		no

5.3 효율성 제고와 성능 비교

전체 트리를 부모, 디스, 자식으로 연결되는 삼원 연결 부분으로 나누어 그 부분을 집중적으로 연구하여 부작용을 여러 케이스로 분류하고 부작용이 발생하는 부분의 알고리즘을 제시하였다. 하지만 이보다 깊이가 더 깊은 트리인 경우에도 적용이 가능하다. 기본이 되는 관계형 데이터베이스의 외래키-키 결합이 많을수록 스키마 트리상의 깊이가 깊어지므로 그 깊이만큼의 외래키-키 관계를 파악하여 tlk의 생성에 관여하는 모든 조상과 생성<tlk>의 결과로 생성된 후손을 기억하고 있으면 위의 알고리즘을 확장할 수 있다. 즉, fk3<trg, tlk, trd>대신 fkn<t1, t2, … trg-1, trg, tlk, trd, trd+1, …, tn-1, tn>을 사용한다.

먼저 효율성을 높이기 위하여, tlk를 삭제할 경우 tlk의 부모노드에 tlk의 인스턴스 트리나 스키마 트리상의 후손노드가 위치하면 그 부모노드로부터 tlk를 포함한 모든 노드가 삭제됨을 이용한다. 즉 가장 많은 부작용이 발생하는 경우에 대하여 먼저 검토하여 노드마다 외래키-키 관계를 비교하

는 과정을 건너뛰어 효율성을 높인다. 이 경우 tlk가 속한 서브트리를 순회하며 모두 삭제리스트에 삽입한 후 나머지 노드를 순회하며 삭제리스트에 있는 엘리먼트인지 비교하므로 효율적이다. tlk의 자식노드나 형제노드를 순회할 때는 방문 노드가 삭제리스트에 있는지 확인해야 하므로 탐색시간이 소요되지만 전체트리는 한 번씩 순회한다.

이에 따라 본 알고리즘의 시간 복잡도는 순차탐색의 비교 횟수와 트리순회 횟수를 더한 것이 된다. 모든 키가 탐색될 확률이 동일하다고 가정하면 순차탐색은 탐색에 성공할 경우 (n+1)/2번 비교하고 탐색에 실패할 경우 n번 비교하며, 트리 역시 n번 순회한다. 그러므로 총 수행횟수 = (n+1)/2 + n = (3n+1)/2 가 되어 시간 복잡도는 O(n)이 된다.

본 연구의 성능을 공식화하여 분석하기는 매우 어렵지만 관계형 데이터베이스의 삽입, 삭제, 갱신에 따른 XML 뷰 반영에 드는 비용보다는 실체화된 XML 파일에 직접 쿼리하는 비용이 훨씬 적으리라 예상된다. 특히 쿼리가 복잡할 경우 본 시스템의 효율성이 두드러질 것으로 보인다. 예를 들어 본 연구처럼 관계형 데이터베이스를 XML로 문서화 하는 XPERANTO 시스템과 비교하면, XPERANTO 시스템은 중간 과정을 만들지 않고, 파이프라인 방식으로 수행되어 필요한 자료만을 반환하므로 많은 자료 중 극히 일부분의 자료에 대한 질의가 들어올 경우는 매우 유용하다. 하지만 기본이 되는 데이터베이스에 부분적인 갱신이 발생할 경우에도 똑같은 방법을 매번 되풀이해야 하므로 수행시간이 많이 소요되어 비효율적이다.

(표 6) 다른 시스템과의 성능 비교

항목	본 시스템	참고문헌[13]	참고문헌[17]
기본개념	RDB의 부분적인 갱신에 대한 실체화 된 XML 뷰파일의 유지	뷰 엘리먼트의 삭제에 대한 RDB의 변환	RDB의 XML 뷰 상에 쿼리를 제공
수행결과	부작용까지 적용한 XML 뷰파일	부작용이 발생한 부분은 제외한 결과를	원하는 자료만을 검색

	생성	반환	
재질의에 대한 응답	XML 뷰파일에 직접 쿼리	모든 과정을 재실행	모든 과정을 재실행
부분적 갱신적용	RDB까지 가지 않고 XML 뷰파일에 직접적용	RDB에서 작업	RDB에서 작업

이에 반해 본 논문은 부분적인 갱신이 발생할 경우, 기본이 되는 관계형 데이터베이스까지 접근할 필요 없이 XML 실체 뷰에서 모든 최신의 자료를 유지하므로 XQuery 질의 수행 효율이 높아진다. 이를 정량적으로 표현하기는 매우 어렵지만, 항목별로 정리한 표6을 참조하면 RDB에 부분적인 갱신이 발생할 경우 부작용까지 적용한 XML 뷰파일에 갱신을 직접 적용할 수 있으므로 다른 시스템보다 빠른 응답시간을 갖게 되어 효율적이다.

6. 결 론

이차원 테이블 구조의 관계형 데이터베이스를 계층구조의 XML 문서로 표현하기 위한 방안으로 관계형 데이터베이스의 외래키 연관관계를 적용시킨다. 즉 한 투플의 외래키가 참조하는 다른 테이블의 주키를 외래키의 부모노드로 매핑 시키면 순서를 갖는 XML 계층구조를 표현할 수 있다. 그리고 이렇게 관계형 데이터베이스의 외래키 연관관계에 의하여 생성된 XML 뷰 파일은 기본이 되는 관계형 데이터베이스에 갱신이 발생하였을 경우 외래키 참조 무결성 조건을 만족하기 위하여 부작용이 나타나게 된다. 이는 생성 당시의 외래키 연관관계를 역으로 유추해 보면 어느 곳에서 부작용이 발생하게 될지 예측 가능하므로 이를 토대로 XML 뷰 파일에 예상되는 부작용까지 적용하여 변환할 수 있다.

그러므로 본 논문의 장점은 부작용을 거부하고 무시하기보다는 오히려 부작용이 발생하는 곳을 미리 조사하여 기본이 되는 관계형 데이터베이스에 갱신이 발생했을 때 부작용까지도 고려하여 이를 XML 뷰 파일에 정확히 반영할 수 있게 한다.

삭제로 인해 발생되는 부작용을 각 케이스별로 구분하고 부작용이 발생하는 노드와 그의 경로를 탐지하기 위하여 삼원 연결 구조를 이용한 삭제 알고리즘을 각 케이스에 적용한 결과 모든 케이스에서 삭제의 영향을 받는 엘리먼트를 정확히 탐지하는 것을 보았다. 후속 연구로서 삽입에 따른 변환도 연구하려 한다.

참 고 문 헌

[1] J. Shanmugasundaram et. al., “Efficiently Publishing Relational Data as XML Documents”, Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000

[2] Igor Tatarinov et. al., “Updating XML”, ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

[3] Philip Bahannon et. al., Incremental Evaluation of Schema-Directed XML Publishing, SIGMOD 2004, June 13-18, 2004, Paris, France.

[4] Jose A. et. al., Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates, Proceedings of the Twelfth International Con erence on Very large Data Bases Kyoto, August, 1986

[5] M. Fernandez et. al., ″SilkRoute: Trading Between Relations and XML,″ Proceedings of the 9th International World Wide Web Conference, Amsterdam, May 2000.

[6] J. E. Funderburk et. al., XTABLES: Bridging relational technology and XML, IBM SYSTEMS JOURNAL, VOL 41, NO 4, pp616-640, 2002

[7] 조정길, “함수적 종속성을 반영한 XML 문서의 관계형 스키마 매핑 기법”, 한국 인터넷 정보학회, (8권2호), pp 95-103, 2007.

[8] Vanessa P. Braganholo et. al., “From XML view updates to relational view updates: old solutions to a new problem”, Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

[9] Surajit Chaudhuri et. al., On Relational Support for XML Publishing: Beyond Sorting and Tagging, SIGMOD 2003, June 9-12, 2003, San Diego, CA.

[10] Igor Tatarinov et. al., “Storing and Querying ordered XML Using a Relational Database System”, ACM SIGMOD 2002, June 4-6, Madison, Wisconsin, USA.

[11] J. E. Funderburk et. al., XTABLES: Bridging relational technology and XML, IBM SYSTEMS JOURNAL, VOL 41, NO 4, pp616-640, 2002

[12] J. Shanmugasundaram et. al., “Relational Databases for Querying XML Documents: Limitations and Opportunities”, Proceedings of the 25th VLDB Conference,Edinburgh, Scotland, 1999.

[13] Ming Jiang, Ling Wang, Murali Mani and Elke A. Rundensteiner “Propagating single XML-view element deletion over relational databases”, INCRUIS, pp 1-8, Jan 2006

[14] 나영국, “객체 지향 뷰 기술을 이용한 투명한 스키마 진화”, 한국 정보과학회, pp1~14, 2001

[15] Silberschatz, ″Database System Concepts 4/E″, McGraw-Hill, pp113-117, 2002

[16] Vanessa P. Braganholo et. al., “Propagating XML View Updates to a Relational Database”, Technical Report Number RP-341 Universidade Federal do Rio Grande do Sul - UFRGS Instituto de Informática Porto Alegre - RS - Brazil February 2004

[17] Jayavel Shanmugasundaram Jerry Kiernan Eugene Shekita Catalina Fan John Funderburk, ″Querying XML Views of Relational Data″, Proceedings of the 27th VLDB Conference, Roma, Italy, 2001

[18] Y.G.Ra and E.A.Rundensteiner, ″A Transparents schema evolution system based on object-oriented view technology.″ IEEE Transaction on knowledge and Data Engeering, 1997

[19] Yi Chen, Susan B. Davidson, Yifeng Zheng, ″Constraint preserving XML Storage in Relations,″ Technical Report, MS-CIS-02-04, University of Pennsylvania, 2002

[20] Ashish Gupta et. al., ″Maintaining Views Incrementally″, SIGMOD May 1993, Washington, DC, USA

◐ 저 자 소 개 ◑

김 미 수

1983년 충남대학교 건축교육공학과 졸업(공학사)
2004년 한국방송통신대학교 컴퓨터과학과 졸업(이학사)
2006년 서울시립대학교 대학원 전자전기컴퓨터공학부 졸업(공학석사)
2008년 서울시립대학교 대학원 전자전기컴퓨터공학부 박사수료
관심분야 : 데이터베이스, XML
E-mail : miso4u@uos.ac.kr

나 영 국

1987년 서울대학교 전자공학과 졸업(학사)
1989년 펜실베니아주립대학교 대학원 컴퓨터공학과 졸업(석사)
1996년 미시간대학교 대학원 컴퓨터학과 졸업(박사)
1997～2002 삼성SDS, 국립한경대학교
2002～현재 서울시립대학교, 전자전기컴퓨터공학부 교수
관심분야 : 데이터베이스, 개발툴.
E-mail : ygra123@gmail.com