점진적 갱신에 기반을 둔 XML 형성뷰 관리 프레임워크

임 재 국[†]·강 현 철[††]·서 상 구[†††]

요 약

뷰는 이질적인 데이타의 통합 및 여과(filtering) 기능을 통해서 사용자가 요구하는 데이타를 제공한다. 뷰는 질의처리의 성능 향상을 위해 형성뷰(materialized view)로 유지될 수 있다. 형성뷰는 하부 데이타가 변경되었을 경우에 일관성을 유지해야 하는데, 그 기법으로는 뷰의 재생성(recomputation)과 변경 내용 중 뷰와 관련이 있는 것만 반영하는 점진적 갱신(incremental refresh)이 있다. XML은 문서의 구조정보를 나타낼 수 있으므로 XML 형성뷰에 대해서는 기존의 관계 형성뷰 등의 관리 기법과는 다른 관리 기법이 요구된다. 본 논문에서는 XML 문서들을 대상으로 생성된 XML 형성뷰를 지원하고, 하부 XML 문서가 변경되었을 경우에 형성뷰에 대해 점진적 갱신을 지원하는 XML 형성뷰 관리 프레임워크를 제안한다.

A Framework of XML Materialized Views Using Incremental Refresh

JaeGuk Lim[†]·Hyunchul Kang[††]·Sang-Koo Seo[†††]

ABSTRACT

The view can provide the user an appropriate portion of the database through data integration and filtering. Views can be materialized for query performance improvement, and in that case, their consistency needs to be maintained against the updates of the underlying data. They can be either recomputed or incrementally refreshed by reflecting the relevant updates. Since XML could represent the structural information of the documents, for the XML materialized views, new techniques that differ from the previous ones for incrementally refreshing the relational views are required. In this paper, we propose a framework of XML materialized view management where the XML views derived from the underlying XML documents are materialized and incrementally refreshed against the updates of the underlying documents.

키워드 : XML, 형성뷰(Materialized View), 점진적 갱신(Incremental Refresh)

1. 서 론

XML(eXtensible Markup Language)이 차세대 웹 문서 표준으로 제안되면서, E-Commerce 등과 같은 응용에서 XML 문서를 활용하려는 노력이 진행 중에 있다[1]. 이를 위해 XML 문서의 구조정보 표현 방법 및 저장 시스템에 대한 연구가 이루어지고 있으며[2-5], XQL(XML Query Language) 등과 같은 XML 표준 질의어에 대한 연구도 진행되고 있다[6, 7].

인터넷의 확산으로 현재 웹에서는 다양한 데이타 저장소(repository)에 많은 양의 데이타들이 산재되어 있다. 뷰는 이질적인 데이타의 통합과 데이타 여과(filtering) 기능을 통해서 사용자가 요구하는 데이타를 제공한다. XML 데이타에 대해서도 뷰는 유용한 개념이다[8].

뷰는 질의처리의 성능 향상을 위해 형성뷰(materialized view)[9]로 유지할 수 있는데 하부 데이타가 변경되었을 경우에 일관성을 유지해야 하는 오버헤드가 따른다. 일관성을 유지하는 방법으로는, 뷰를 하부 데이타로부터 재생성(recomputation)하는 것과 변경 내용만을 뷰에 반영하는 점진적 갱신(incremental refresh) 기법이 있다. 형성뷰의 점진적 갱신에 대해서는 주로 관계 데이타베이스 시스템에서 많은 연구가 수행되었다[10-12]. XML은 문서의 구조정보를 나타낼 수 있으므로 XML 형성뷰에 대해서는 기존의 관계 형성뷰 갱신 기법과는 다른 기법이 요구된다. 주요 연구 이슈로는 첫째, 형성뷰 및 그것의 점진적 갱신을 위한 XML 저장소 내의 자료구조, 둘째, 변경 내용이 해당 뷰와 연관성이 있는

※ 본 논문은 정통부의 대학정보통신연구센터 육성·지원 사업에 의한 것임.
† 준 회 원 : 동양시스템즈(주)
†† 정 회 원 : 중앙대학교 컴퓨터공학과 교수
††† 정 회 원 : 광운대학교 경영정보학과 교수
논문접수 : 2000년 11월 29일, 심사완료 : 2001년 6월 27일

지 여부를 점검하는 기법, 셋째, 점진적 갱신에 필요한 정보의 효율적 생성 기법, 넷째, 갱신 정보의 효율적 반영 등을 들 수 있다.

본 논문은 XML 저장소의 문서들로부터 생성된 XML 형성뷰에 관한 것으로, 점진적 갱신에 기반을 둔 XML 형성뷰 관리 프레임워크를 제안한다. 제안된 프레임워크는 XML 하부 문서 및 형성뷰를 저장하는 저장소의 구조, 하부 문서의 변경 관리, 변경과 형성뷰와의 연관성 검사 및 그에 따른 형성뷰의 점진적 갱신을 위한 정보 생성 알고리즘, 그리고 갱신 정보를 형성뷰에 반영하는 알고리즘 등으로 구성된다.

본 논문의 구성은 다음과 같다. 2절에서는 관련연구를 기술한다. 3절에서는 XML 문서 및 형성뷰의 저장구조, XML 문서의 변경 관리 기법, 그리고 XML 형성뷰의 점진적 갱신 알고리즘을 제안한다. 4절에서는 3절에서 제안된 내용을 예시한다. 마지막으로 5절에서 결론을 맺고 향후 연구내용을 기술한다.

2. 관련 연구

XML 형성뷰의 점진적 갱신에 관한 관련 연구로는, 반구조적(semistructured) 데이타에 대한 형성뷰의 점진적 갱신에 관한 연구가 있다[13-15]. 이들은 모두 그래프 기반의 반구조적 데이타베이스로부터 생성한 형성뷰의 갱신을 다루었고 XML 문서를 하부 데이타로 연구하지는 않았다. XML 뷰에 대해서는 최근에 연구가 시작되고 있으나[8, 16] XML 형성뷰에 관한 연구는 아직 없다.

[13]에서는 에지(edge)가 레이블을 갖는 루트가 있는 트리 구조[17]에 기반을 둔 반구조적 데이타베이스에 대한 형성뷰의 점진적 갱신에 대해 연구하였다. 뷰는 UnQL[17]로 정의한 것으로 조인이 없는 질의만을 대상으로 하였다. 하부 데이타베이스의 변경으로는 한 트리를 기존 노드의 서브트리로 삽입하는 것과 기존 서브트리를 새 트리로 대체(replace)하는 것을 고려하였다. 데이타 소스에 변경이 발생하면 그 소스로부터 생성된 뷰가 있는 사이트로 통지되고, 변경내용 △(즉, 새로 또는 기존 서브트리를 대체하기 위해 삽입된 서브트리)이 전송된다. 형성뷰가 있는 사이트는 전송된 △를 처리하여 형성뷰를 점진적으로 갱신한다. 이 기법은 형성뷰의 갱신을 위해 하부 데이타 소스에의 접근을 피하고 오직 변경내용 △만 접근하면 되지만, 조인 질의에 의한 형성뷰의 점진적 갱신을 지원할 수 없고 하부 데이타 소스의 내용 수정(modification)에 의한 형성뷰의 점진적 갱신을 수행하지 못한다.

[14]에서는 그래프 구조의 데이타베이스를 대상으로 생성된 형성뷰의 점진적 갱신을 연구하였다. 대상 데이타베이스의 예로는 링크로 연결된 웹 페이지(들)을 들 수 있다. 즉, 포인터(에지)를 갖는 객체(노드)들의 집합으로 모델링이 가능한 데이타베이스이다. 뷰는 OQL[18]의 확장 버전으로 정의한 것을 대상으로 하였고, 형성뷰는 뷰의 조건을 만족하는 객체들의 집합으로 표현하고 이들 간의 연결 즉, 에지는 취급하지 않았다. 데이타 소스의 변경으로는 기존의 두 객체 간에 새로운 에지의 삽입, 기존 에지의 삭제, 그리고 원자객체(atomic object) 값의 수정을 고려하였고, 객체의 삽입과 삭제는 대상으로 하지 않았다. 데이타 소스에 변경이 발생하면, 형성뷰에 영향을 미치는 객체들을 알아내기 위한 질의를 생성한 후 이를 데이타 소스에 대해 수행하여 검색된 객체를 형성뷰에/로부터 삽입/삭제한다.

[15]에서는 그래프 기반의 객체 교환 모델(OEM)[19]을 따르는 반구조적 데이타에 대한 형성뷰의 점진적 갱신에 대해 연구하였다. 뷰는 반구조적 데이타베이스 관리 시스템인 Lore[20]의 질의어인 Lorel[21]의 확장 버전을 사용하여 정의된 것을 대상으로 하였고 [14]와 달리 형성뷰에도 객체 간의 에지를 포함시켰다. 고려된 데이타 소스의 변경은 [14]에서와 같은 세가지이고 형성뷰의 점진적 갱신 방식도 동일하다. 즉, 변경 각각에 대해 뷰 유지 문장(view maintenance statement)들을 생성해내고 이들을 하부 데이타 소스에 대해 수행하여 뷰의 갱신에 필요한 객체들을 검색해낸 후 이들을 형성뷰에 반영한다.

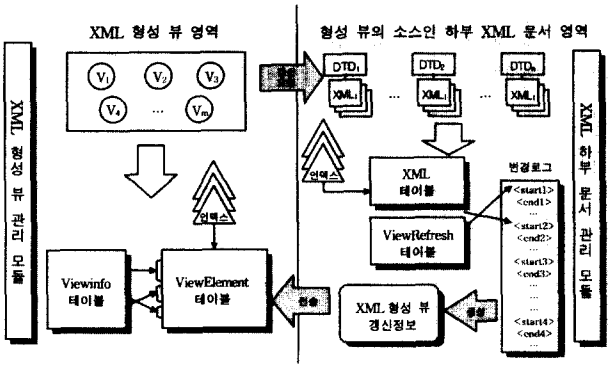
이들 연구들은 모두 그래프 기반의 반구조적 데이타로부터 생성된 형성뷰의 점진적 갱신을 다룬 것으로, 고려된 데이타 소스의 변경 유형이 제한적이며, 변경 유형이 아주 제한적인 [13] 외에는 형성뷰의 점진적 갱신을 위해 모든 유형의 변경에 대해 하부 데이타 소스의 검색을 필요로 한다. 또한 가장 복잡도가 높은 형성뷰를 지원하는 [15]의 경우에는 뷰와 연관성이 없는 변경 중 일부에 대해서도 뷰 유지 문장을 생성해야 한다. 이들은 점진적 갱신의 효율성을 저하시키는 요인이 된다.

3. XML 형성뷰 관리 프레임워크

본 절에서는 점진적 갱신에 기반을 둔 XML 형성뷰 관리 프레임워크를 제안한다. 제안하는 프레임워크의 구성요소들을 그림으로 나타내면 (그림 1)과 같다. XML 문서의 저장소는 XML 형성뷰의 소스인 **하부 XML 문서 영역(이하, 하부 영역)**과 **XML 형성뷰 영역(이하, 뷰 영역)**으로 구분되고, 이들 영역은 XML 하부 문서 관리 모듈과 XML 형성뷰 관리 모듈에 의해 각각 관리된다. 하부 영역에는 다수의 DTD와 XML 문서, XML 문서 검색을 위한 인덱스, XML 형성뷰의 점진적 갱신을 지원하기 위해 XML 문서의 변경을 기록하는 변경로그(update log), 각 XML 형성뷰를 점진적으로 갱신하기 위해 필요한 정보가 저장된다. 뷰 영역에는 각 XML 형성뷰에 대한 정보, 엘리먼트 ID 사상(mapping)정보, 형성뷰의 내용, 그리고 형성뷰 검색을 위한 인덱스 등이 저장된다.

XML 형성뷰의 점진적 갱신은 지연 수행(deferred refresh)

된다. 즉, 하부 XML 문서의 변경이 발생할 때 즉각적으로 관련 뷰에 반영(immediate refresh)되는 것이 아니라 변경 내용이 변경 로그에만 기록되었다가 사용자로부터 뷰가 요청되었을 때 갱신된다. 따라서 XML 형성뷰 관리 모듈은 사용자로부터 뷰가 요청되면 해당 형성뷰의 점진적 갱신에 필요한 **XML 형성뷰 갱신정보(이하, 갱신정보)**를 XML 하부 문서 관리 모듈에게 요청한다. XML 하부 문서 관리 모듈은 변경로그를 검색해서 해당 뷰에 대한 XML 형성뷰 갱신정보를 생성하고, 이것을 XML 형성뷰 관리 모듈에게 전송한다. 갱신정보를 전송받은 XML 형성뷰 관리 모듈은 이를 이용해서 해당 XML 형성뷰에 대한 점진적 갱신을 수행한 후 사용자에게 뷰를 제공한다.



(그림 1) XML 형성뷰 관리 프레임워크

이후 본절의 구성은 다음과 같다. 3.1절에서는 하부영역의 XML 문서 저장 구조 및 뷰 영역의 형성뷰 저장 구조를 기술한다. 3.2절에서는 하부영역의 XML 문서 변경에 대해 기술하고, 3.3절에서는 형성뷰의 점진적 갱신에 대해 기술한다.

3.1 XML 문서 및 형성뷰 저장 구조

하부 영역에는 다수의 DTD와 XML 문서들이 저장된다. XML 문서의 저장 방법은 크게 분할 방법과 비분할 방법으로 나눌 수 있다[5]. 분할 방법은 XML 문서를 엘리먼트 단위로 분할해서 분할하는 기법이고, 비분할 방식은 문서 전체를 BLOB(Binary Large OBject) 혹은 CLOB(Character Large OBject)의 형태로 저장하는 기법이다. 본 논문에서는 XML 문서를 분할 방법에 의해 현재 가장 널리 사용되고 있는 관계 DBMS에 저장하는 경우를 연구 대상으로 하였다.

XML 문서들은 해당 DTD를 준수하는(valid) 것으로서 트리 구조를 취하며, DID, DTDID, EID, Ename, Content 컬럼으로 구성된 XML 테이블에 저장된다. DID에는 XML 문서의 ID가, DTDID에는 XML 문서가 참조하는 DTD의 ID가, EID에는 엘리먼트의 ID가, Ename에는 엘리먼트 명이, 그리고 Content에는 실제 XML 문서의 데이타가 저장된다. 엘리먼트 ID는 [3]에서 기술한 경로 엘리먼트 ID를 사용하였다. 경로 엘리먼트 ID는 엘리먼트의 ID를 할당함에 있어 자신이 부모 엘리먼트의 몇 번째 자식 엘리먼트인가를 두자리 수로 나타내고 이를 부모 엘리먼트의 ID 끝에 추가한 것이다. 예를 들어 조상 엘리먼트의 ID가 '0101'인 경우, 첫 번째와 두 번째 자식 엘리먼트의 ID는 각각 '010101'과 '010102'가 된다.

```
<!ELEMENT 논문 (제목, 요약, 절*, 결론, 참고문헌)>
<!ELEMENT 제목 (#PCDATA)>
<!ELEMENT 요약 (#PCDATA)>
<!ELEMENT 절 (#PCDATA, 구*)>
<!ELEMENT 구 (#PCDATA)>
<!ELEMENT 결론 (#PCDATA)>
<!ELEMENT 참고문헌 (#PCDATA)>
```

(그림 2) 논문 DTD

(그림 2)와 같은 논문 DTD가 있다고 하자. 논문 DTD는 '제목', '요약', '절', '결론', '참고문헌' 엘리먼트로 구성되고 '절'은 자식 엘리먼트 '구'를 가진다. '제목', '요약', '결론', '참고문헌'은 단 한번만 나올 수 있고, '절'과 '구'는 반복해서 여러 번 나올 수 있다. (그림 3)은, 논문 DTD를 따른 XML 문서가 3개 있다고 할 때 이들을 XML 테이블에 저장한 모습이다.

한편, 뷰 영역에는 XML 형성뷰에 대한 정보와 데이타를 ViewInfo 테이블과 ViewElement 테이블에 저장한다. View Info 테이블에는 각 XML 형성뷰마다 한 개의 레코드가 존재하며 각 XML 형성뷰에 부여되는 뷰의 ID(ViewID), 뷰의 생성 조건을 정의한 뷰 정의(ViewDef), 뷰가 참조하는 DTD의 ID 등으로 구성된다. ViewElement 테이블은 ViewID, DID, BaseEID, ViewEID, Content 컬럼으로 구성된다. ViewID는 ViewInfo 테이블의 ViewID를 참조하는 외래 키이다. DID는 뷰를 생성한 하부 XML 문서의 ID를 나타낸다. ViewEID는 뷰를 구성하는 엘리먼트의 EID를 나타낸다. BaseEID는 DID가 가리키는 하부 XML 문서의 엘리먼트 ID를 나타낸다. Content에는 실제 XML 뷰를 구성하는 데이타가 저장된다.

본 논문에서는 뷰의 생성을 위한 XML 질의어를 특정 질의어로 한정하지 않는다. 단, 질의어의 구문으로 표현 가능한 뷰 중에서 단일 DTD를 대상으로 생성되는 XML 뷰만을 고려하였다. 즉, XML 저장소에 다수의 DTD가 저장되어 있고 각 DTD에 따른 다수의 XML 문서들이 저장되어 있을 때, XML 뷰는 하나의 DTD에 따라 저장된 XML 문서들을 대상으로 생성된 것이다. 또한 뷰의 생성 조건은 DTD를 구성하는 엘리먼트 중 한번만 나오는 엘리먼트 하나에 대해 명시된다고 가정하였다. 예를 들어 (그림 2)의 논문 DTD의 경우에 뷰의 생성 조건에 이용될 수 있는 엘리먼트는 '제목', '요약', '결론', '참고문헌' 중의 하나이다. (그림 4)는 ViewInfo 테이블과 ViewElement 테이블의 구조와 그 예를 나타낸 것으로, (그림 3)의 XML 테이블에 저장된 XML 문서를 대상으로 "제목에 'Refresh'라는 단어가 들어있는 논문의 제목, 요약, 참고문헌"으로 정의된 뷰 V_1을 생성하여 형성뷰로 저장한 모습이다.

DID	DTDID	EID	Ename	Content
1	1	01	논문	/root
1	1	0101	제목	XML 문서의 검색 및 변경을 위한 저장관리기의 평가
1	1	0102	요약	XML는 차세대 인터넷 어플리케이션을 위한 문서 표준으로 …
1	1	0103	절	1. 서론
1	1	010301	구	1990년대 인터넷의 발전에 가장 큰 영향을 주었던 웹 …
1	1	010302	구	웹이 지금까지 발전하는 데에는 누구나 만들고 쉽게 …
1	1	010303	구	급속히 발전하고 있는 웹상에서 또는 컴퓨터상의 각각이 …
1	1	0104	절	2. XML 문서의 유형
1	1	010401	구	XML 문서는 DTD가 있는가 없는가에 따라 문서를 나누어 …
1	1	010402	구	데이터 중심 XML 문서는 매우 정형적인 구조를 가지며 …
1	1	0105	결론	본 논문은 객체 지향 데이터베이스 시스템으로 개발된 …
1	1	0106	참고문헌	[1] Document Object model(DOM) Level 1 Specification …
2	1	01	논문	/root
2	1	0101	제목	A Snapshot Differential Refresh Algorithm
2	1	0102	요약	This article presents an algorithm to refresh the contents of database …
2	1	0103	절	1. Introduction
2	1	010301	구	A DBMS provides a mechanism for maintaining, access, and updating …
2	1	010302	구	The notion of a database snapshot was introduced in [ADIBA80] …
2	1	0104	절	Snapshot Refresh Objectives
2	1	010401	구	Snapshot refresh should make the snapshot reflect the current, …
2	1	0105	절	2. Alternative Refresh Methods
2	1	010501	구	Several alternatives are available for implementing snapshot refresh …
2	1	010502	구	Another alternative is to buffer the changes to the base table and …
2	1	0106	결론	The differential snapshot agorithm has been implemented as …
2	1	0107	참고문헌	[ADIBA 80] M.E. Adiba and B.G. Lindsay, Database Snapshots, …
3	1	01	논문	/root
3	1	0101	제목	XML 저장소에서 문서 링크와 뷰 갱신
3	1	0102	요약	XML 문서의 폭발적인 확산에 따라 Web 상의 각 사이트는 …
3	1	0103	절	1. 서론
3	1	010301	구	현재의 컴퓨팅 환경에서는 Web의 발전과 더불어 전자 …
3	1	010302	구	다양한 정보 형태를 가진 전자 문서의 효과적인 관리를 …
3	1	010303	구	실제 Web 상에서는 HTML 문서와 같이 XML 문서들도 …
3	1	0104	절	2. 관련 연구
3	1	010401	구	XML 문서는 기존의 정보와는 달리 논리적 구조를 갖는다. …
3	1	010402	구	XML에서는 링크는 XLink[6]와 Xpointer[7]mf 사용하여 …
3	1	0105	결론	본 논문에서는, 실제 Web 상에 분산되어 저장된 전자 문서 …
3	1	0106	참고문헌	[1] T. Bray et al., "Extensible Markup Language (XML) …

(그림 3) XML 테이블의 구조 및 예

ViewInfo 테이블

ViewID	ViewDef	DTDID
V_1	제목에 "Refresh" 단어가 들어있는 논문의 제목, 요약, 참고문헌	1

ViewElement 테이블

ViewID	DID	BaseEID	ViewEID	Content
V_1	2	01	01	/root
V_1	2	0101	0101	A Snapshot Differential Refresh Algorithm
V_1	2	0102	0102	This article presents an algorithm to refresh the …
V_1	2	0107	0103	[ADIBA 80] M.E. Adiba and B.G. Lindsay, "Database …

(그림 4) ViewInfo 테이블과 ViewElement 테이블의 구조 및 예

3.2 하부 영역의 XML 문서 변경 관리

XML 문서는 문서 단위, 엘리먼트 단위, 속성 단위로 변경될 수 있는데, 본 논문에서는 문서 및 엘리먼트 단위의 변경을 고려하였다. 수정(modify)은 XML 문서 내에서 값을 갖는 엘리먼트에 대해 엘리먼트 단위로 수행되고, 삽입과 삭제는 문서 단위로 수행된다고 가정하였다.

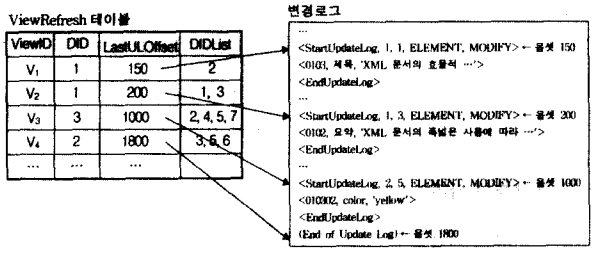
XML 하부 문서 관리 모듈은 XML 문서가 변경될 때마다 XML 형성뷰의 점진적 갱신을 지원하기 위해서 그 내용을 변경로그에 기록한다. 변경로그 레코드의 자료구조는 (그림 5)와 같다. 변경로그 레코드는 <StartUpdateLog> 필드로 시작해서 <EndUpdateLog> 필드로 끝나는 블록구조를 갖는다. 변경로그 레코드가 블록구조를 갖는 것은 XML 문서에 대한 일련의 연관된 변경을 하나의 트랜잭션으로 간주하

기 위한 것이다. 예를 들어, XML 문서가 삽입되었을 경우에, 이것은 일련의 엘리먼트들의 삽입으로 간주되고 모든 엘리먼트의 삽입이 완료되었을 때 변경로그 레코드의 작성이 완료된다.

```
<StartUpdateLog, DTDID, DID, ObjType, OpType>
[<BaseEID, Ename, Content>[,< BaseEID, Ename, Content>, …]]
<EndUpdateLog>
```

(그림 5) 변경로그 레코드 자료구조

DTDID와 DID는 변경이 발생한 XML 문서의 DTD와 문서 ID를 각각 나타낸다. ObjType은 변경의 단위가 엘리먼트('ELEMENT')인지 문서('DOCUMENT')인지를 나타낸다. OpType은 XML 문서에 대한 변경의 종류를 나타낸다. 즉, OpType이 'MODIFY'이면 엘리먼트 수정, 'INSERT'이면 새로운 문서의 삽입, 'DELETE'이면 기존 문서의 삭제를 의미한다. <BaseEID, Ename, Content>는 변경된 엘리먼트에 대한 정보로서 ObjType과 OpType에 따라 생략되거나 한번 또는 그 이상 기록될 수 있다. BaseEID는 변경되는 엘리먼트의 ID를 나타내고, Ename은 변경된 엘리먼트 명을 나타낸다. Content는 Optype = 'MODIFY' 또는 'INSERT'일 경우 수정된 또는 삽입된 엘리먼트의 값을 나타낸다. Optype = 'DELETE'인 경우 Content는 NULL로 설정된다.



(그림 6) ViewRefresh 테이블과 변경로그

3.3 형성뷰의 점진적 갱신

형성뷰의 점진적 갱신은 첫째, 변경로그의 검색, 둘째, 그를 통한 갱신정보의 생성, 그리고 셋째, 그것의 형성뷰로의 반영으로 구성된다.

3.3.1 변경로그의 검색

변경 로그의 검색은 이전의 뷰 갱신 이후에 기록된 변경로그 레코드들을 대상으로 수행된다. 새로이 검색해야 할 첫 번째 변경로그 레코드의 위치는 ViewRefresh 테이블에 기록되어 있다. (그림 6)은 XML 형성뷰의 점진적 갱신에 필요한 정보를 저장하고 있는 ViewRefresh 테이블의 구조 및 변경로그와의 관계를 나타낸 것이다. ViewRefresh 테이블에는 각 형성뷰에 대해서 한 개의 레코드가 저장되며, 각 레코드는 ViewID, 뷰의 하부 소스 XML 문서들이 참조하는 DTD의 ID(DTDID), 갱신정보 생성을 위해서 검색해야 하는 변경로그 레코드들의 시작 옵셋(offset)을 나타내는 LastULOffset, 그리고 뷰의 조건을 만족하는 하부 XML 문서들의 ID 리스트인 DIDList가 저장된다.

LastULOffset은 각 형성뷰가 갱신정보를 생성하기 위해서 가장 마지막으로 읽은 변경로그 레코드의 다음 옵셋을 나타낸다. 이를 통해 다음 갱신이 요구될 때, LastULOffset이 가리키는 곳에 기록된 변경로그 레코드부터 마지막 변경로그 레코드까지 검색함으로써 변경로그의 검색 양을 줄일 수 있다. 형성뷰가 생성될 때 LastULOffset은 그 시점에서 변경로그의 끝 옵셋으로 설정된다. 갱신정보를 생성하기 위해 변경로그 검색을 마친 후에도 LastULOffset은 변경로그의 끝 옵셋으로 설정된다. (그림 6)에서 V_1에 대한 갱신정보가 요청되면 XML 하부 문서 관리 모듈은 V_1의 LastULOffset '150'부터 '1800'까지의 변경로그 레코드를 읽고, LastULOffset을 변경로그의 끝 옵셋인 '1800'으로 변경한다.

변경로그 공간의 조각모음(garbage collection)은 LastULOffset을 이용하면 쉽게 수행할 수 있다. 삭제해도 되는 불필요한 변경로그 레코드들은 ViewRefresh 테이블의 LastULOffset 값들 중에서 가장 작은 값이 가리키는 변경로그 레코드 이전에 기록된 것들이다. 이들을 삭제한 후에는 ViewRefresh 테이블에 저장된 각 LastULOffset 값에서 가장 작은 LastULOffset의 값을 빼서 LastULOffset을 조정해야 한다. (그림 6)의 경우에는 V_1의 LastULOffset이 가장 작으므로 옵셋 '150' 이전에 기록된 변경로그 레코드들은 삭제해도 된다. 그 후에 V_1, V_2, V_3, V_4의 LastULOffset을 각각 '0', '50', '850', '1650'으로 조정한다.

3.3.2 형성뷰 갱신정보의 생성

변경로그의 검색을 통해 각 로그 레코드별로 해당 뷰에 영향을 미치는지의 여부를 판단하고 연관성이 있을 경우 해당 형성뷰에 반영해야 할 내용을 갱신정보 레코드로 구성하여 갱신정보에 기록한다. 각 갱신정보 레코드에는 RefType, DID, BaseEID, ViewEID, Content가 저장된다((그림 10) 참조). RefType은 갱신 유형(refresh type)을 나타내는 것으로 'MODIFY', 'INSERT', 그리고 'DELETE'의 세 가지 유형이 있다(아래와 3.3.3절을 참조). DID는 변경된 문서의 ID, BaseEID는 변경된 하부 문서 엘리먼트의 ID, ViewEID는 삽입될 뷰 엘리먼트의 ID, 그리고 Content는 형성뷰에 반영(수정 또는 삽입)될 엘리먼트의 값을 나타낸다. 이들 중 해당 사항이 없는 것은 NULL로 설정된다.

(그림 7)은 갱신정보 생성 알고리즘 Create_RefreshInfo를 기술한 것이다. Create_RefreshInfo에는 사용자에 의해 요청된 뷰의 ViewID와 해당 뷰의 생성조건을 정의한 뷰

정의 ViewDef가 파라미터로 전달된다. Create_RefreshInfo는 갱신정보 RefreshInfo를 초기화하고(init_refreshinfo()) 변경로그에 검색해야 할 레코드가 존재하는지 검사한다. 변경로그 레코드가 존재하지 않으면 뷰에 반영해야 할 변경이 없는 것이므로 갱신정보 생성이 종료되고 RefreshInfo를 반환한다. 그렇지 않을 경우, 변경로그를 개방하여 (open_updatelog()) 갱신정보가 요청된 형성뷰의 LastULOffset이 가리키는 변경로그 레코드부터 변경로그의 끝까지 차례로 검색하면서 해당 변경로그 레코드의 변경 내용과 뷰와의 연관성을 검사한다(relevance_check()). 해당 변경이 뷰에 영향을 준다면 연관성 유형(rel_type)에 따라 뷰의 갱신에 필요한 내용을 RefreshInfo

```
Create_RefreshInfo(ViewID, ViewDef)
{
    RefreshInfo = init_refreshinfo(); /* 갱신정보 초기화 */
    i=0;                              /* 갱신정보 index 초기화 */
    lastoffset = ViewRefresh[ViewID].LastULOffset;
              /* ViewRefresh 테이블에서 ViewID를 이용해 해당
                 뷰의 LastULOffset을 검색*/
    if (lastoffset != NULL && UpdateLog[lastoffset] != end_of_log)
                /* 변경 있음 */
    {
              DTDID = ViewRefresh[ViewID].DTDID;
              DIDList = ViewRefresh[ViewID].DIDList;
                        /* ViewRefresh 테이블에서 ViewID를
                           이용해 해당 뷰의 DIDList를 검색*/
              DidListUpdated = False;
              open_updatelog(lastoffset);    /* 변경로그 개방 */
              do {
                   lastoffset = scan_updatelog(lastoffset, &Ulog_
                                Rec);    /* 변경로그 검색 */
                   rel_type = relevance_check(ViewDef, DTDID,
                              DIDList, Ulog_Rec);
                             /* 현재 변경로그 레코드가 해당 뷰에
                                영향을 주는지 연관성 검사 */
                   if (rel_type != NULL) {/* 연관성이 있으면 */
                        i = append_refreshinfo(rel_type,
                          RefreshInfo, DIDList, Ulog_Rec , i);
                          /* 새로운 갱신정보 추가 */
                        if (rel_type != MODIFY) DidListUpdated
                        = TRUE;
                   }
              } while (!end_of_log);
              ViewRefresh[ViewID].LastULOffset = lastoffset;
              /* 해당 뷰의 LastULOffset 변경 */
              if (DidListUpdated) ViewRefresh[ViewID].DIDList =
              DIDList; /* 해당 뷰의 DIDList 변경 */
    }
    return (RefreshInfo);
}
```

(그림 7) 갱신정보 생성 알고리즘

에 추가한다(append_refreshinfo()). 변경로그의 끝에 도달하면 ViewRefresh 테이블의 해당 뷰 레코드의 내용을 적절히 변경하고 RefreshInfo를 반환한다.

변경로그 레코드의 변경 내용과 뷰와의 연관성 검사에는 해당 뷰의 정의(ViewDef), DTDID, 그리고 DIDList가 참조된다. 뷰의 정의로부터는 뷰의 조건 P를 규정하는 조건 엘리먼트(condition element)와 뷰의 내용을 구성하는 추출 엘리먼트(target element)의 집합을 도출한다. 전자를 CE, 후자를 TE라 하면, 예를 들어 (그림 4)의 뷰 V_1의 경우, 그 정의로부터 CE = '제목', TE = {'제목', '요약', '참고문헌'}이 된다. DTDID와 DIDList는 ViewRefresh 테이블로부터 검색된다.

연관성 검사에서는 우선 해당 변경로그 레코드의 DTDID가 뷰를 도출한 문서들의 DTDID와 동일한지를 검사한다. 변경로그 레코드와 뷰가 연관이 있으려면 기본적으로 이들 DTDID가 일치해야 한다. DTDID가 일치한다고 할 때, 해당 변경과 뷰가 연관성이 있기 위한 조건은 아래 다섯가지 연관성 유형(rel_type)으로 나눌 수 있다. 이들 유형에 해당되면 각 유형별로 형성뷰 갱신에 필요한 갱신정보 레코드(들)이 RefreshInfo에 추가되며, 또한 유형에 따라서 DIDList가 변경된다. 이들 다섯가지 유형의 조건과 추가되는 갱신정보 레코드의 내용 및 DIDList의 변경 여부는 다음과 같다.

유형 MODIFY-M :

해당 변경로그 레코드의 (OpType = 'MODIFY' AND Ename $\in$ TE AND DID $\in$ DIDList)인 경우, 뷰의 조건을 만족하는 문서의 추출 엘리먼트가 수정된 것이므로 이를 형성뷰에서도 수정하기 위해 (MODIFY, DID, BaseEID, NULL, Content)로 구성된 갱신정보 레코드를 RefreshInfo에 추가한다.

유형 MODIFY-I :

(OpType = 'MODIFY' AND Ename = CE AND DID $\notin$ DIDList AND P(Content))인 경우(여기서 P(Content)는 수정된 엘리먼트의 값 Content가 뷰의 조건 P를 만족함을 의미), 뷰의 조건을 만족하지 않던 문서의 조건 엘리먼트 값이 수정되어 뷰의 조건을 만족하게 되었으므로 형성뷰에 해당 문서의 추출 엘리먼트(들)을 삽입해야 한다. 이를 위해, 먼저 DID에 대한 인덱스를 통해 XML 테이블에 접근하여 해당 문서의 추출 엘리먼트(들)을 검색한 후, 형성뷰에 삽입될 내용으로 (INSERT, DID, BaseEID, ViewEID, Content) 레코드(들)을 RefreshInfo에 추가한다. 그리고 DIDList에 해당 DID 값을 삽입한다.

유형 MODIFY-D :

(OpType = 'MODIFY' AND Ename = CE AND DID $\in$ DIDList AND (NOT P(Content)))인 경우, 뷰의 조건을 만족하던 문서의 조건 엘리먼트 값이 수정되어 뷰의 조건을 만

족하지 않게 되었으므로 형성뷰에서 해당 문서의 엘리먼트들을 모두 삭제해야 한다. 이를 위해, (DELETE, DID, NULL, NULL, NULL) 레코드를 RefreshInfo에 추가한다. 그리고 DIDList에서 해당 DID값을 삭제한다.

유형 INSERT :

(OpType = 'INSERT' AND P(Content where Ename = CE))인 경우(여기서 'Content where Ename = CE'는, 삽입된 문서의 엘리먼트 정보를 기록한 로그 레코드의 <Base EID, Ename, Content>(들) 중 Ename = CE인 것의 Content를 의미), 뷰의 조건을 만족하는 문서가 삽입되었으므로 형성뷰에 해당 문서의 추출 엘리먼트(들)을 삽입해야 한다. 이를 위해, 먼저 로그 레코드에서 뷰의 추출 엘리먼트(들)에 대한 기록 <BaseEID, Ename, Content>을 참조하여, 형성뷰에 삽입될 내용으로 (INSERT, DID, BaseEID, ViewEID, Content) 레코드(들)을 RefreshInfo에 추가한다. 그리고 DID List에 해당 DID 값을 삽입한다.

유형 DELETE :

(OpType = 'DELETE' AND DID ∈ DIDList)인 경우, 뷰의 조건을 만족하던 문서가 삭제되었으므로 형성뷰에서 해당 문서의 엘리먼트들을 모두 삭제해야 한다. 이를 위해, (DELETE, DID, NULL, NULL, NULL) 레코드를 RefreshInfo에 추가한다. 그리고 DIDList에서 해당 DID값을 삭제한다.

3.3.3 형성뷰의 갱신

(그림 8)은 알고리즘 Create_RefreshInfo에 의해 생성된 갱신정보를 해당 뷰에 반영하는 알고리즘 Refresh_MV를 기술한 것이다. Refresh_MV는 먼저 파라미터로 넘어온 갱신정보 RefreshInfo를 검사한다. RefreshInfo가 비어 있으면, 기존

```
Refresh_MV (ViewID, RefreshInfo)
{
  i = 0 ;  /* 갱신정보 index 초기화 */
  if (check_empty(RefreshInfo) != EMPTY) /* 갱신정보가 존재하면 */
       do {
            RInfo = fetch(RefreshInfo[i++]) ;
            switch (RInfo.RefType) {
              case MODIFY :  /* 엘리먼트의 수정 */
                modify_content(ViewID, RInfo) ;
                /* 해당 엘리먼트의 content 수정 */
                break ;
              case INSERT : /* 문서의 삽입 */
                i = insert_document(ViewID, RInfo, RefreshInfo, i) ;
                break ;
              case DELETE : /* 문서의 삭제 */
                delete_document(ViewID, RInfo.DID) ;
                break ;
            } /* end of switch */
       } while (!end_of RefreshInfo)
}
```

(그림 8) XML 형성뷰의 점진적 갱신 알고리즘

의 형성뷰가 유효하다는 뜻이므로 뷰 갱신을 종료한다. 그렇지 않으면 RefreshInfo의 각 레코드를 (그림 10)의 갱신정보 레코드와 같은 구조를 갖는 RInfo 구조체에 차례로 읽어들여 다음을 수행한다.

먼저 RInfo.RefType을 검사한다. RInfo.RefType이 'MODIFY'이면 ViewElement 테이블에서 ViewID, RInfo.DID, 그리고 RInfo.BaseEID 값으로 해당 엘리먼트를 찾아 Content를 RInfo.Content 값으로 대체한다(modify_content()). 이때, ViewElement 테이블로부터 해당 뷰의 레코드(들)을 검색하는 과정에는 ViewID에 대한 인덱스를 이용한다.

RInfo.RefType이 'INSERT'이면, ViewID와 RInfo.DID 값을 이용하여 ViewElement 테이블에 먼저 (ViewID, RInfo.DID, 01, 01, /root) 레코드를 삽입한 후, RInfo의 값으로 첫 번째 추출 엘리먼트에 대한 레코드를 삽입한다. 추출 엘리먼트가 하나 이상일 수 있으므로 RefreshInfo로부터 다음 레코드를 읽어서 RefType = 'INSERT'이고 DID 값에 변화가 없는 동안 계속 RInfo의 값으로 해당 추출 엘리먼트에 대한 레코드를 삽입한다(insert_document()). RInfo.RefType이 'DELETE'이면 ViewElement 테이블로부터 ViewID 값으로 해당 뷰의 레코드를 찾아 이들 중 DID = RInfo.DID 인 것을 모두 삭제한다(delete_document()).

4. 예

본 절에서는 사용자가 뷰를 요청하였을 경우 해당 형성뷰의 점진적 갱신 과정의 예를 기술한다. (그림 9)는 View Refresh 테이블에서 (그림 4)의 뷰 V_1에 해당되는 레코드와, 사용자가 V_1을 다시 요청하였을 때 그동안 변경로그에 기록된 하부 영역에서의 변경을 나타낸 아홉개의 변경로그 레코드를 나타낸 것이다.

알고리즘 Create_RefreshInfo는 ViewRefresh 테이블에서 V_1의 레코드를 검색하여 lastoffset = 150, DTDID = 1, DIDList = {2}로 초기화한 후, 변경로그를 개방하고 옵셋 150에 위치한 레코드부터 하나씩 차례로 읽어서 해당 변경과 V_1과의 연관성을 검사하고 그 연관성 유형에 따라 갱신정보 레코드(들)를 생성하여 RefreshInfo에 추가한다.

(그림 9)의 아홉 개 로그 레코드는 모두 DTDID = 1인 문서에 대한 변경을 기록한 것이므로 DTDID = 1인 문서들로부터 도출한 V_1과 연관성이 있을 수 있다. 첫 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 1, ELEMENT, MODIFY>
<0101, 제목, 'XML 문서의 효율적 검색 및 변경을 위한 저장 관리기의 평가'>
<EndUpdateLog>
```

즉, DID = 1인 문서의 '제목' 엘리먼트 값의 수정을 기록한 것이다. '제목' = CE이므로 수정된 값('XML 문서의 효율적 검색 및 변경을 위한 저장 관리기의 평가')에 대해 V_1의 조건을 검사한 결과 만족하지 않는다. 또한 1 ∉ DIDList이므로 이 변경은 V_1과 연관성이 없다. 두 번째 변경로그 레코드의 내용은 다음과 같다.

ViewRefresh 테이블

ViewID	DTDID	LastULOffset	DIDList
V_1	1	150	2

변경로그

```
<StartUpdateLog, 1, 1, ELEMENT, MODIFY> ← 옵셋 150
<0101, 제목, 'XML 문서의 효율적 검색 및 변경을 위한 저장
 관리기의 평가'>
<EndUpdateLog>
<StartUpdateLog, 1, 4, DOCUMENT, INSERT>
<01, 논문, '/root'>
<0101, 제목, 'XML View and Its Refresh'>
<0102, 요약, 'XML 문서 저장소로부터 XML 뷰를 …'>
<0103, 절, 1. '서론'>
<010301, 구, 'XML 문서 검색의 효율성을 높이기 위해 …'>
<010302, 구, '뷰와 객체는 …'>
<0104, 결론, 'W3C에서 XML을 웹 문서 표준으로 …'>
<0105, 참고문헌, 'S. Abiteboul et al., On Views and XML, …'>
<EndUpdateLog>
<StartUpdateLog, 1, 1, ELEMENT, MODIFY>
<0105, 결론, '본 논문에서는 XML 저장관리기의 구조를 제시하고
 …'>
<EndUpdateLog>
<StartUpdateLog, 1, 2, ELEMENT, MODIFY>
<0106, 결론, 'A database snapshot is a read-only table whose
 contents …'>
<EndUpdateLog>
<StartUpdateLog, 1, 2, ELEMENT, MODIFY>
<0107, 참고문헌, '[HAAS 82] L. Haas, P. Selinger, E. Bertino, D.
 Daniels, B. Lindsay, …'>
<EndUpdateLog>
<StartUpdateLog, 1, 3, ELEMENT, MODIFY>
<0106, 참고문헌, '[1] S. Abiteboul, Querying Semistructured
 Data …'>
<EndUpdateLog>
<StartUpdateLog, 1, 2, ELEMENT, MODIFY>
<0101, 제목, '스냅샷의 점진적 갱신 알고리즘'>
<EndUpdateLog>
<StartUpdateLog, 1, 3, ELEMENT, MODIFY>
<0101, 제목, 'Document Links and View Refresh in XML
 Repository'>
<EndUpdateLog>
<StartUpdateLog, 1, 4, DOCUMENT, DELETE>
<EndUpdateLog>
(End of Update Log) ← 옵셋 1800
```

(그림 9) ViewRefresh 테이블 및 변경로그의 예

```
<StartUpdateLog, 1, 4, DOCUMENT, INSERT>
<01, 논문, '/root'>
<0101, 제목, 'XML View and Its Refresh'>
<0102, 요약, 'XML 문서 저장소로부터 XML 뷰를 …'>
<0103, 절, 1. '서론'>
<010301, 구, 'XML 문서 검색의 효율성을 높이기 위해 …'>
<010302, 구, '뷰와 객체는 …'>
<0104, 결론, 'W3C에서 XML을 웹 문서 표준으로 …'>
<0105, 참고문헌, 'S. Abiteboul et al., "On Views and XML," …'>
<EndUpdateLog>
```

즉, DID = 4인 문서의 삽입을 기록한 것이다. V_1의 CE는 '제목'이므로 해당 엘리먼트의 값('XML View and Its Refresh')을 검사한 결과 삽입된 문서는 V_1의 조건을 만족함을 알 수 있다. 따라서 형성뷰에도 이 문서의 추출 엘리먼트들이 삽입되어야 한다. 즉, 이 변경은 3.3.2절에 기술된 연관성 유형 INSERT에 해당된다. 따라서 V_1의 TE에 속하는 '제목', '요약', 그리고 '참고문헌' 엘리먼트에 대한 내용으로부터 세 개의 갱신정보 레코드를 생성하여 RefreshInfo에 추가한다((그림 10)의 첫 번째~세번째 레코드). 그리고 DIDList에 4를 추가한다(DIDList = {2,4}). 세 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 1, ELEMENT, MODIFY>
<0105, 결론, '본 논문에서는 XML 저장관리기의 구조를 제시하고
 …'>
<EndUpdateLog>
```

즉, DID = 1인 문서의 '결론' 엘리먼트 값의 수정을 기록한 것이다. 1 ∉ DIDList이고 '결론' != CE이므로, 이 변경은 V_1과 연관성이 없다. 네 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 2, ELEMENT, MODIFY>
<0106, 결론, 'A database snapshot is a read-only table whose
 contents …'>
<EndUpdateLog>
```

즉, DID = 2인 문서의 '결론' 엘리먼트 값의 수정을 기록한 것이다. 2 ∈ DIDList이지만 '결론' ∉ TE 이므로, 이 변경은 V_1과 연관성이 없다. 다섯 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 2, ELEMENT, MODIFY>
<0107, 참고문헌, '[HAAS 82] L. Haas, P. Selinger, E. Bertino, D.
 Daniels, B. Lindsay, …'>
<EndUpdateLog>
```

즉, DID = 2인 문서의 '참고문헌' 엘리먼트 값의 수정을 기록한 것이다. 2 ∈ DIDList이고 '참고문헌' ∈ TE 이므로, 형성뷰의 해당 '참고문헌' 엘리먼트의 값도 수정해야 한다. 즉, 이 변경은 연관성 유형 MODIFY-M에 해당된다. 따라서 RefreshInfo에 갱신정보 레코드를 추가한다((그림 10)의 네 번째 레코드). 여섯 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 3, ELEMENT, MODIFY>
<0106, 참고문헌, '[1] S. Abiteboul, Querying Semistructured Data ...'>
<EndUpdateLog>
```

즉, DID = 3인 문서의 '참고문헌' 엘리먼트 값의 수정을 기록한 것이다. '참고문헌' ∈ TE 이지만 3 ∉ DIDList 이므로, 이 변경은 V_1과 연관성이 없다. 일곱 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 2, ELEMENT, MODIFY>
<0101, 제목, '스냅샷의 점진적 갱신 알고리즘'>
<EndUpdateLog>
```

즉, DID = 2인 문서의 '제목' 엘리먼트 값의 수정을 기록한 것이다. 2 ∈ DIDList이고 '제목' = CE 이므로 수정된 값('스냅샷의 점진적 갱신 알고리즘')에 대해 V_1의 조건을 검사한 결과 만족하지 않는다. 따라서 형성뷰로부터 이 문서의 추출 엘리먼트들을 삭제해야 한다. 즉, 이 변경은 연관성 유형 MODIFY-D에 해당된다. 따라서 RefreshInfo에 갱신정보 레코드를 추가한다((그림 10)의 다섯 번째 레코드). 그리고 DIDList로부터 2를 삭제한다(DIDList = {4}). 여덟 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 3, ELEMENT, MODIFY>
<0101, 제목, 'Document Links and View Refresh in XML Repository'>
<EndUpdateLog>
```

즉, DID = 3인 문서의 '제목' 엘리먼트 값의 수정을 기록한 것이다. '제목' = CE이므로 수정된 값('Document Links and View Refresh in XML Repository')에 대해 V_1의 조건을 검사한 결과 만족한다. 3 ∉ DIDList이므로 형성뷰에도 이 문서의 추출 엘리먼트를 삽입해야 한다. 즉, 이 변경은 연관성 유형 MODIFY-I에 해당된다. 따라서 RefreshInfo에 갱신정보 레코드를 추가한다((그림 10)의 여섯 번째~여덟 번째 레코드). 그리고 DIDList에 3을 추가한다(DIDList = {3,4}). 마지막 아홉 번째 변경로그 레코드의 내용은 다음과 같다.

```
<StartUpdateLog, 1, 4, DOCUMENT, DELETE>
<EndUpdateLog>
```

즉, DID = 4인 문서의 삭제를 기록한 것이다. 4 ∈ DIDList 이므로 형성뷰에서 이 문서의 추출 엘리먼트를 삭제해야 한다. 즉, 이 변경은 연관성 유형 DELETE에 해당된다. 따라서 RefreshInfo에 갱신정보 레코드를 추가한다((그림 10)의 아홉 번째 레코드). 그리고 DIDList로부터 4를 삭제한다(DIDList = {3}).

RefType	DID	BaseEID	ViewEID	Content
INSERT	4	0101	0101	XML View and Its Refresh
INSERT	4	0102	0102	XML 문서 저장소로부터 XML 뷰를 …
INSERT	4	0105	0103	S. Abiteboul et al., "On Views and XML," …
MODIFY	2	0107	NULL	[HAAS 82] L. Haas, P. Selinger, E. Bertino, D. Daniels, B. Lindsay, …
DELETE	2	NULL	NULL	NULL
INSERT	3	0101	0101	Document Links and View Refresh in XML Repository
INSERT	3	0102	0102	XML 문서의 폭발적인 확산에 따라 Web 상의 각 사이트는 …
INSERT	3	0106	0103	[1] T. Bray et al., "Extensible Markup Language (XML) …
DELETE	4	NULL	NULL	NULL

(그림 10) XML 형성뷰 갱신정보의 구조 및 예

(그림 10)은 (그림 9)의 변경로그를 검색해서 생성된 뷰 V1의 갱신정보를 나타낸 것이다. 알고리즘 Refresh_MV는 이들 갱신정보 레코드들을 차례로 읽어 형성뷰에 반영하게 된다. 한편, (그림 11)은 (그림 3)의 XML 테이블이 (그림 9)의 변경로그에 기록된 변경들에 의해 변경 결과를 나타낸 것이다. 그리고 (그림 12)는 V1에 대한 (그림 4)의 형성뷰가 (그림 10)의 갱신정보에 의해 점진적으로 갱신된 후의 View Refresh 테이블과 ViewElement 테이블의 모습을 나타낸 것이다. (그림 11)과 (그림 12)의 ViewElement 테이블을 대조해보면 (그림 9)의 변경로그 검색을 통한 V1에 대한 형성뷰의 점진적 갱신이 정확히 수행되었음을 알 수 있다.

ViewRefresh 테이블

ViewID	DTDID	LastULOffset	DIDList
V_1	1	1800	3

ViewElement 테이블

ViewID	DID	BaseEID	ViewEID	Content
V_1	3	01	01	/root
V_1	3	0101	0101	Document Links and View Refresh in XML Repository
V_1	3	0102	0102	XML 문서의 폭발적인 확산에 따라 Web 상의 각 사이트는 …
V_1	3	0106	0103	[1] T. Bray et al., "Extensible Markup Language (XML) …

(그림 12) 뷰 V_1의 점진적 갱신 후 ViewRefresh 테이블과 ViewElement 테이블의 모습

DID	DTDID	EID	Ename	Content
1	1	01	논문	/root
1	1	0101	제목	XML 문서의 효율적 검색 및 변경을 위한 저장 관리기의 평가
1	1	0102	요약	XML는 차세대 인터넷 어플리케이션을 위한 문서 표준으로 …
1	1	0103	절	1. 서론
1	1	010301	구	1990년대 인터넷의 발전에 가장 큰 영향을 주었던 웹 …
1	1	010302	구	웹이 지금까지 발전하는 데에는 누구나 만들고 쉽게 …
1	1	010303	구	급속히 발전하고 있는 웹상에서 또는 컴퓨터상의 각각이 …
1	1	0104	절	2. XML 문서의 유형
1	1	010401	구	XML 문서는 DTD가 있는가 없는가에 따라 문서를 나누어 …
1	1	010402	구	데이터 중심 XML 문서는 매우 정형적인 구조를 가지며 …
1	1	0105	결론	본 논문은 XML 저장관리기의 구조를 제시하고 …
1	1	0106	참고 문헌	[1] Document Object model(DOM) Level 1 Specification …
2	1	01	논문	/root
2	1	0101	제목	스냅샷의 점진적 갱신 알고리즘
2	1	0102	요약	This article presents an algorithm to refresh the contents of database …
2	1	0103	절	1. Introduction
2	1	010301	구	A DBMS provides a mechanism for maintaining, access, and updating …
2	1	010302	구	The notion of a database snapshot was introduced in [ADIBA80] …
2	1	0104	절	Snapshot Refresh Objectives
2	1	010401	구	Snapshot refresh should make the snapshot reflect the current, …
2	1	0105	절	2. Alternative Refresh Methods
2	1	010501	구	Several alternatives are available for implementing snapshot refresh …
2	1	010502	구	Another alternative is to buffer the changes to the base table and …
2	1	0106	결론	A database snapshot is a read-only table whose contents …
2	1	0107	참고 문헌	[HAAS 82] L. Haas, P. Selinger, E. Bertino, D. Daniels, B. Lindsay, …
3	1	01	논문	/root
3	1	0101	제목	Document Links and View Refresh in XML Repository
3	1	0102	요약	XML 문서의 폭발적인 확산에 따라 Web 상의 각 사이트는 …
3	1	0103	절	1. 서론
3	1	010301	구	현재의 컴퓨팅 환경에서는 Web의 발전과 더불어 전자 …
3	1	010302	구	다양한 정보 형태를 가진 전자 문서의 효과적인 관리를 …
3	1	010303	구	실제 Web 상에서는 HTML 문서와 같이 XML 문서들도 …
3	1	0104	절	2. 관련 연구
3	1	010401	구	XML 문서는 기존의 정보와는 달리 논리적 구조를 갖는다. …
3	1	010402	구	XML에서는 링크는 XLink[6]와 Xpointer[7]mf 사용하여 …
3	1	0105	결론	본 논문에서는, 실제 Web 상에 분산되어 저장된 전자 문서 …
3	1	0106	참고 문헌	[1] S. Abiteboul, "Querying Semistructured Data …

(그림 11) 변경된 XML 테이블

5. 결론 및 향후연구

본 논문에서는 XML 저장소의 문서들로부터 생성한 XML 뷰를 형성뷰로 유지하고 이를 하부 XML 문서의 변경에 대해 점진적으로 갱신하는 XML 형성뷰 관리 프레임워크를 제안하였다. 제안된 프레임워크는 다음 요소들로 구성된다 : (1) 형성뷰를 지원하는 XML 저장소의 구조, (2) XML 문서의 변경 관리, (3) XML 문서의 변경 내용과 형성뷰와의 연관성 검사, (4) 형성뷰의 점진적 갱신을 위한 갱신정보 생성 알고리즘, 그리고 (5) 갱신 정보를 형성뷰에 반영하는 알고리즘. 이들에 대한 연구 결과를 요약하면 다음과 같다.

XML 저장소는 XML 형성뷰의 소스인 XML 문서를 저장하는 하부 문서 영역과 형성뷰를 저장하는 뷰 영역으로 나누어 구성된다. 하부 영역에는 XML 문서를 저장하는 XML 테이블, XML 문서 검색을 위한 인덱스, XML 문서의 변경을 기록하는 변경로그, 그리고 XML 형성뷰를 점진적으로 갱신하기 위해 필요한 정보를 기록하고 있는 ViewRefresh 테이블이 저장된다. 뷰 영역에는 각 뷰에 대한 정보를 저장하고 있는 ViewInfo 테이블, XML 형성뷰를 저장하기 위한 View Element 테이블, 그리고 형성뷰 검색을 위한 인덱스가 저장된다. 하부 영역에서 XML 문서의 변경은 관련 형성뷰의 점진적 갱신을 지연 수행(deferred refresh)하기 위해 변경 로그에 기록된다. 각 변경로그 레코드는 문서 또는 엘리먼트 단위의 변경을 기록한다. 해당 변경과 어떤 뷰 V와의 연관성 여부는 ViewInfo 테이블로부터 검색된 V의 정의, ViewRefresh 테이블 내의 V에 대한 레코드, 그리고 해당 변경로그 레코드 자체만 참조하여 판단할 수 있다. 뷰와의 연관성 유형(relevance type)은 MODIFY-M, MODIFY-I, MODIFY-D,

INSERT, 그리고 DELETE의 다섯가지이다. 이들 각 유형별로 XML 형성뷰 갱신정보를 생성하는 알고리즘 Create_RefreshInfo()를 기술하였다. 갱신정보의 레코드 구조는 View Element 테이블의 구조에 {INSERT, DELETE, MODIFY} 중의 한가지 값을 갖는 갱신 유형(refresh type)을 덧붙인 것이다. 갱신정보를 갱신 유형별로 ViewElement 테이블에 반영하는 알고리즘 Refresh_MV()를 기술하였다. 마지막으로 예제를 통해 상기 저장 구조와 알고리즘에 따라 XML 형성뷰의 점진적 갱신이 정확히 수행됨을 보였다.

제안된 프레임워크의 장단점을 기술하면 다음과 같다. 장점은 첫째, 자주 제기되는 XML 질의 결과를 형성뷰로 유지함으로 인한 질의 처리의 성능 향상이다. 이는 해당 질의가 다시 제기되었을 때 저장소 내 방대한 양의 XML 문서 검색을 피하고 이전 질의 처리 이후의 변경 내용만을 검토하면 되기 때문이다. (단, 변경의 양이 많지 않을 경우에 한해서이다.) 둘째, XML 저장소의 구조를 하부 문서 영역과 뷰 영역으로 분리함으로써 중앙집중 환경과 분산 환경에 모두 적용 가능하다. 중앙집중 환경의 경우 하부 영역과 뷰 영역은 같은 사이트에 존재하고, 분산 환경의 경우 서로 다른 사이트에 존재한다. 셋째, 실용성이다. 이는 XML 문서 및 형성뷰의 저장을 위해 현재 가장 널리 사용되고 있는 관계 DBMS를 채택하였기 때문이다.

한편 단점은 저장 오버헤드이다. 형성뷰란 뷰의 내용을 직접 저장하고 있는 것이므로 저장 공간을 요한다. 많은 수의 그것도 특히 용량이 큰 뷰들을 형성뷰로 유지하고자 할 경우 저장 공간의 제약에 부딪힐 수 있다. 따라서 접근 빈도가 높은 뷰들만 형성뷰로 유지하기 위한 적절한 뷰 선택(view selection) 체계를 필요로 한다. 저장 오버헤드는 변경로그의 유지에서도 발생한다. 단, 본 논문의 변경로그 관리에서는 더 이상 보존이 필요없는 변경로그 레코드들에 대한 간단한 조각모음(garbage collection)이 가능하다(3.3.1절 참조).

향후 연구 과제로는 다음을 들 수 있다. 첫째, 본 논문에서는 하부 XML 문서의 변경으로 엘리먼트 단위의 변경과 문서 단위의 변경을 함께 고려하였는데, 문서, 엘리먼트, 애트리뷰트의 변경을 모두 지원하기 위한 기법의 연구가 필요하다. 둘째, 본 논문에서는 형성뷰의 갱신을 지연 수행하는 기법을 제안하였는데, 그로 인해 즉각 갱신 기법에 비해 동일한 데이타 객체의 중복 갱신(예를 들어, 새로운 엘리먼트의 삽입 이후 그 엘리먼트 값의 수정)을 형성뷰에 효율적으로 반영할 수 있는 장점이 있다. 이에 대한 연구가 필요하다. 셋째, 형성뷰의 점진적 갱신의 성능을 뷰 재생성에 비해 평가하는 것이 필요하다. 즉, 성능 파라미터를 설정하고 두 기법의 비용을 공식화하여 정량적인 비교를 하는 것이 필요하다. 특히, 본 논문에서 제안한 지연 갱신의 경우, 변경 로그의 양에 따라 점진적 갱신이 뷰 재생성에 비해 성능이 우수할 조건을 도출하는 연구가 필요하다.

참 고 문 헌

[1] S. Abiteboul et al., "Active Views for Electronic Commerce," Proc. Int'l Conf. on VLDB, 1999, pp.138-149.

[2] J. Shanmugasundaram et al., "Relational Databases for Querying XML Documents : Limitations and Opportunities," Proc. Int'l Conf. on VLDB, 1999, pp.302-314.

[3] 연제원 외, "XML 문서 구조검색을 위한 저장 시스템 설계", 한국정보과학회, '99년 봄 학술발표논문집, 제26권 제1호, 1999, pp. 3-5.

[4] 이용석, 손기락, "XML 문서 저장 시스템 설계 및 구현", 한국정보과학회, '98년 가을 학술발표논문집, 제25권 제2호, 1998, pp.347-349.

[5] 한상웅, 홍의경, "ORDBMS를 이용한 XML 저장 시스템 설계", 한국정보과학회, 2000년 가을 학술발표논문집, 제27권 제2호, 2000, pp.3-5.

[6] J. Robie et al., "XML Query Language (XQL)," http ://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[7] A. Deutsch et al., "XML-QL : A Query Language for XML," http ://www.w3.org/TR/1998/NOTE-xml-ql-19980819/, 1998.

[8] S. Abiteboul, "On Views and XML," Proc. ACM Symp. on Principles of Database System, 1999, pp.1-9.

[9] A. Gupta and I. Mumick, "Maintenance of Materialized Views : Problems, Techniques, and Applications," Bulletin of TCDE, Vol.18, No.2, Jun. 1995, pp.3-18.

[10] B. Lindsay et al., "A Snapshot Differential Refresh Algorithm," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1986, pp.53-60.

[11] J. Blakeley et al., "Efficiently Updating Materialized Views," Proc. ACM SIGMOD Int'l Conf. on Management of Data, 1986, pp.61-71.

[12] N. Roussopoulos, "An Incremental Access Method for View Cache : Concept, Algorithms, and Cost Analysis," ACM Trans. on Database Systems, Vol.16, No.3, Sep. 1991, pp.535-563.

[13] D. Suciu, "Query Decomposition and View Maintenance for Query Languages for Unstructured Data," Proc. Int'l Conf. on VLDB, 1996, pp.227-238.

[14] Y. Zhuge and H. Garcia-Molina, "Graph Structured Views and Their Incremental Maintenance," Proc. Int'l Conf. on Data Engineering, 1998, pp.116-125.

[15] S. Abiteboul et al., "Incremental Maintenance for Materialized Views over Semistructured Data," Proc. Int'l Conf. on VLDB, 1998, pp.38-49.

[16] Y. Papakonstantinou and V. Vianu, "DTD Inference for Views of XML Data," Proc. of 19th ACM SIGACT-SIGMOD-SIGART Symp. on PODS, 2000.

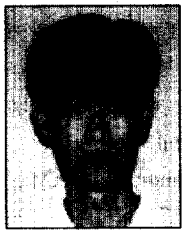
[17] P. Buneman et al, "Programming Constructs for Unstructured Data," Proc. DBPL, 1995.

[18] R. Cattell et al., "The Object Database Standard : ODMG-93," Morgan Kaufmann, 1994.

[19] Y. Papakonstantinou et al., "Object Exchange across Heterogeneous Information Sources," Proc. Int'l Conf. on Data Engineering, 1995, pp.251-260.

[20] J. McHugh et al., "Lore : A database Management System for Semistructured Data," SIGMOD Record, Vol.26, No.3, Sep. 1997, pp.54-66.

[21] S. Abiteboul et al, "The Lorel Query Language for Semistructured Data," J. of Digital Libraries, Vol.1, No.1, Nov. 1996.



임 재 국

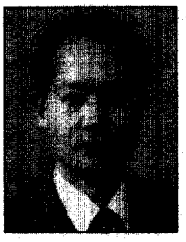
e-mail : jglim@tysystems.com

1999년 중앙대학교 컴퓨터공학과 졸업 (공학사)

2001년 중앙대학교 컴퓨터공학과 대학원 졸업(공학석사)

현재 동양시스템즈(주) 재직 중

관심분야 : 대용량 데이타베이스, Java & XML



강 현 철

e-mail : hckang@cau.ac.kr

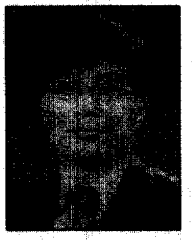
1983년 서울대학교 컴퓨터공학과 졸업 (공학사)

1985년 U. of Maryland at College Park, Computer Science(M.S.)

1987년 U. of Maryland at College Park, Computer Science(Ph.D.)

1988년~현재 중앙대학교 컴퓨터공학과 교수

관심분야 : 이동 데이타베이스, 웹 데이타베이스, DBMS 저장 시스템 등



서 상 구

email : skseo@daisy.gwu.ac.kr

1984년 서울대학교 컴퓨터공학과 졸업 (공학사)

1986년 한국과학기술원 전산학과(M.S.)

1995년 한국과학기술원 전산학과(Ph.D.)

1999년~현재 광운대학교 경영정보학과 조교수

관심분야 : 데이타웨어하우징, 웹 데이타베이스, 데이타베이스 최적화 등