트리거 기반의 형성뷰 관리기법을 이용한 실시간 보고서 생성 모델

*최미란, **전근환, *현득창, *신예호

*극동대학교 정보통신학부, **군장대학 컴퓨터응용학부

A Realtime Report Generation Model using Mterialized View Management Technique Based on Database Trigger

*Mi Ran Choi, **Keun Hwan Jeon, *Hyun, Deuk-chang, *Ye Ho Shin

*School of Communication Infomation in Far East University,

**School of Computer Application in Kunjang College

Abstract

Reports have a significant meaning in time-constrained large transaction environments, such as airplane control systems or wargame simulations. This is due to the necessity of generating reports within a limited scope of time without restraining the operation performance of large transaction environments. In order to generate reports in large transaction environments while satisfying time-constrained requirements, this paper propose a model which combines the incremental operation mechanism and materialized view mechanism using triggers and stored procedures. Further, the implementation and evaluation of the proposed model provides analysis for model characteristics.

1. 서론

가상의 환경에서 전쟁 수행 능력을 훈련하는 워게임(WarGame) 등과 같이 대규모 트랜잭션이 상시적으로 나타나는 환경에서 제한된 시간 즉 시간 제약을 충족하면서 보고서를 생성하기 위해서는 효율적인 데이터 요약 기법이 필요하다. 그러나 데이터베이스에서 보편적으로 이용 되고 있는 기존의 데이터 요약 기술들은 대부분 요약 시점에 전체 데이터베이스를 대상으로 연산을 수행한다. 따라서 보고서 생성을 위한 데이터 요약 비용이 지나치게 높아 시간 제약을 갖는 대규모 트랜잭션 환경에 적용하기 어렵다.

이와 같은 문제를 해결하기 위해 이 논문에서는 트리거와 결합된 점진적 갱신연산에 의 관리되는 형성뷰 관리 모델을 도입한다. 점진적 갱신 연산을 기반으로 하는 형성 뷰 관리 모델은 트리거와 결합하여 데이터베이스 갱신 시점에 자동으로 점진적 뷰 형성 연산을 호출함으로써 갱신 연산을 완료할 수 있도록 한다. 아울러 이 논문에서

는 제안하는 모델에 대한 구현 및 실험을 통해 성능특성을 분석한다.

이를 위한 논문의 구조는 다음과 같다. 제 2장에서는 관련 연구로 점진적 기법 과 형성 뷰 및 트리거와 관련된 기존 연구들을 분석한다. 제 3장에서는 점진적 형성뷰 연산을 위한 연산 모델을 제시하고 4장 3장에서 제시한 점진적 형성 뷰 연산 모델을 트리거와 결합하기 위한 모델을 제안하고 구현 예를 제시한다. 그리고 5장에서 제안 모델의 실험을 위한 실험 환경 및 실험 결과를 제시하고 6장에서 결론을 내린다.

2. 관련연구

뷰 구조에 해당하는 물리적 값들을 갖고 있는 형성 뷰는 뷰 형성에 필요한 연산 부하로 인해 검색 연산이 갱신 연산에 비해 훨씬 높은 비율을 점유할 때 주로 활용된다[1]. 이와 같은 형성 뷰는 분산 데이터베이스에서의 데이터 통합[2] 데이터 웨어하우스[3] 등에서 주요하게 이용되고 있다.

한편 점진적 뷰 형성 연산의 기반이 되는 점진적 기법(incremental method)은 컴퓨터 과학 분야에서 자원의 효율적 운영과 성능 향상이라는 목표를 성취하기 위해 오랫동안 논의되어 왔던 주제이다[4]. 이와 같은 점진적 기법은 주로 능동데이터베이스에서 조건절의 최적화를 위해 이용되었다[5].

아울러 HiPAC[6] 이후 데이터베이스의 주요한 요소로 확립된 능동데이터베이스 개념은 최근 차세대 데이터베이스 언어 표준으로 확립된 SQL 3[7]에서 트리거(trigger)가 기본 요소로 포함되었을 뿐만 아니라 대부분의 상용시스템에서 기본적인 요소(primitive component)로 정착할 만큼 중요한 요소로 자리잡고 있다[7].

3. 뷰 형성을 위한 점진적 연산 모델

일반적으로 데이터베이스에서 접근 빈도가 높은 자료를 형성 뷰 기법으로 유지 관리 할 경우 데이터의 처리 속도를 높일 수 있는 장점이 있다. 그러나 형성 뷰 기법으로 뷰를 유지 관리하기 위해서는 기본 테이블의 변경 정보를 뷰에 전파하는 전략과 뷰 자체적인 갱신 여부에 관한 관리 기법이 요구된다. 형성 뷰에 대한 정의는 정의 3.1과 같이 나타낼 수 있다.

[정의 3.1] 형성 뷰

• 형성 뷰 V는 기본 테이블 R에 대한 뷰정의 e에 의해 형성된 유도 릴레이션이다.

V=e(R)

• 기본 테이블 R의 변경 연산 μ에 의해 기본 테이블은 R^{μ}로 변경되며 이는 뷰의 갱신을 유도한다(p->는 전파 연산을 나타낸다).

$$|R - R^{\mu}|\ p\ ->\ V$$

• $|R - R^{\mu}|$의 정보를 Δ라 표현하며 뷰에 전파 p->는 다음과같이 새로운 버전의 뷰 V_{new}를 유도한다.

$$V_{new} = V\ \theta\ e(\Delta)(\theta \in \{insert, delete, update\})$$

□

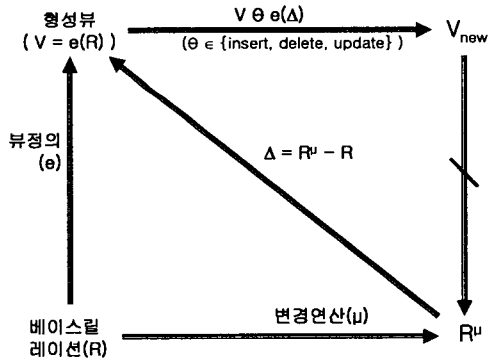

그림 3.1 형성 뷰 관리 메커니즘

그림 3.1의 형성 뷰 관리 메커니즘은 정의 3.1의 뷰 정의를 토대로 점진적 연산에 기반한 뷰 관리 메커니즘을 기술하고 있다. 정의 3.1 형성 뷰에 대한 점진적 연산을 다음과 같이 정의하였다.

$$V_{new} = V\ \theta\ e(\Delta)\ (\theta \in \{insert, delete, upda$$

여기서 핵심적 역할을 담당하는 요소는 점진적 연산을 가능하게 하는 수단인 차분(differenc e) 이다. 뷰를 형성하기 위한 기본 테이블로부터

발생할 수 있는 차분으로는 데이터베이스 수정 연산인 삽입, 삭제, 및 갱신에 대응하는 삽입차분, 삭제차분 그리고 갱신 차분이로 구분할 수 있으며 이들은 다음과 같은 의미를 갖는다.

삽입차분

$$R(\Delta_i^+)=\{r_i \mid r_i \in R_i \wedge r_i \notin R_{i-1}\}$$

삭제차분

$$R(\Delta_i^-)=\{r_i \mid r_i \notin R_i \wedge r_i \in R_{i-1}\}$$

갱신차분

$$R(\Delta_i^u)=\{r_i \mid r_i \in R_i \wedge r_i \in R_{i-1} \wedge \exists r(a_j)\ for\ all\ j\ r_i(a_j) \neq r_{i-1}(a_j)\}$$

이와 같은 의미를 모두 반영한 차분 집합은 다음과 같은 정형의미를 갖는다.

차분 $R(\Delta_i) = R(\Delta_i^+) \cup R(\Delta_i^-)$

이 차분 의미를 이용하여 점진적 연산을 다시 정의하면 다음과 같다.

$$V_i = V_{i-1}\ \theta\ R(\Delta_i),$$
$$단\ \theta \in \{insert, delete, insert(delete)\}$$

여기서 θ 연산자의 단위 연산자들에 대한 연산 대상은 차분을 정의하는 각각의 차분들 즉,

$R(\Delta_i^+)$, $R(\Delta_i^-)$,

$R(\Delta_i^+) \cup R(\Delta_i^-)$ 을 대상으로 하며 특히 마지막 연산의 명세 $insert(delete(R))$ 는 갱신 연산을 규정하는 의미로서 일반적으로 갱신 연산은 삭제후 재 삽입으로 규정하는 연산 의미를 반영한 것이다.

4. 트리거와 결합한 점진적 뷰 형성 모델

이 장에서는 트리거의 자동화된 조건부 조치 기능과 점진적 연산 모델의 결합을 통한 점진적 뷰 형성 연산 모델을 제시한다.

4.1 트리거와 점진적 뷰형성 연산모델의 결합

트리거는 데이터베이스 상태변경 연산에 대응해서 자동으로 조건부 조치를 수행할 수 있도록 하는 데이터베이스 프리미티브이다. 이 트리거와 점진적 뷰 형성 연산 모델의 결합은 점진적 뷰 형성 연산을 트리거 조치절에 명세함으로써 가능해 진다. 트리거와 결합된 점진적 뷰 형성 연산과 결합된 트리거 프레임은 다음의 그림 4.1과 같다.

```
CREATE TRIGGER incrementalInsertTrigger
AFTER INSERT ON <target table name>
REFERENCING NEW AS <new alias>
FOR EACH ROW
BEGIN
        trigger body
END;
```

그림 4.1 점진적 뷰 형성 모델을 위한 트리거 프레임

4.2 점진적 뷰 형성 연산의 구현

이 절에서는 4.1절에서 구성한 점진적 뷰 형성 모델을 위한 트리거의 바디 부분을 구성할 점진적 연산의 구현 예를 제시한다. 그림 4.1의 트리거 프레임에서 트리거 바디 부분에 들어갈 점진적 뷰 형성 연산은 형성 뷰를 요구하는 요구사항에 따라 다양하게 구현된다. 따라서 보편적인 모델을 갖지 않으며 상황에 따라 적절하게 대응하는 뷰 형성 연산을 구현해야 한다. 다음의 그림 4.2는 뷰 형성 연산을 설명하기 위한 예제 질의이다.

```
select sum(aa.a1), sum(aa.a2),
from (select a.k1, a.a1, from tab1 a,
          tab2 b  where a.k2 = b.k2 ) aa
where aa.k1 = 사용자지정코드
group by aa.k1
         union all
select sum(aa.a1), sum(aa.a2),
from (select a.k1, a,k3 a.a1, a.a2, from tab1 a,
          tab2 b, (select k3, k4 from tab3
                where k0 = '사용자지정상수') c
       where a.k2=b.k2 and b.k4=c.k4 ) aa
where aa.key1 = 사용자지정코드
group by aa.k1, aa.k3
```

그림 4.2 갱신 뷰 형성 질의 명세

그림 4,2에서 명세하는 갱신 뷰 형성질의(1)

```
TRIGGER InsertTab1Trigger
AFTER INSERT ON TAB1
REFERENCING NEW AS newTab1
FOR EACH ROW
DECLARE
   key1_ok              BOOLEAN;
   material_cnt         NUMBER;
   key3_value           tab3.k3%TYPE := NULL;
   key4_value           tab2.k4%TYPE := NULL;
BEGIN
   key1_ok := key1_assign(tab1_list, :newTab1.key1, :newTab1.key2);
   key4_value := compare_key2(tab1_list, tab2_list, :newTab1.key1);
   key3_value := select_key3(tab3_list, key4_value);
   IF key1_ok = TRUE THEN
       IF key4_value != NULL THEN
           FOR material_cnt IN material_1.FIRST..material_1.LAST LOOP
               IF material_1(material_cnt).key1 = :newTab1.key1 THEN
                   key1, key3 속성을 제외한 material_1(material_cnt)의 모든 속성값 1 증가
               END IF;
           END LOOP;
           FOR material_cnt IN material_2.FIRST..material_2.LAST LOOP
               IF (material_2(material_cnt).key1 = :newTab1.key1)
                       AND (material_2(material_cnt).key3 = key3_value) THEN
                   key1, key3 속성을 제외한 material_2(material_cnt)의 모든 속성값 1 증가
               END IF;
           END LOOP;
       END IF;
   ELSIF key1_ok = FASLE THEN
       material_1 리스트의 material_1.LAST+1 위치에 새로운 레코드 값 설정
           // key1 := :newTab1.key1; key3 := NULL, 나머지 속성은 모두 1;
       IF key4_value != NULL TEHN
           material_2 리스트의 material_2.LAST+1 위치에 새로운 레코드 값 설정
           // key1 := :newTab1.key1; key3 := key3_value; 나머지 속성은 모두 1;
       END IF;
   END IF;
   UPDATE 연산 수행
       // 위 알고리즘에서 변경된 레코드들을 일괄적으로 데이터베이스에 반영
END TRIGGER;
```

그림 4.3 갱신 뷰 형성 질의를 위한 점진적 트리거

은 형성 뷰를 이용하지 않고 순수한 질의만으로 보고서 생성을 위한 검색을 수행하는 질의이다. 이와 같은 질의 명세에서 최종적으로 획득하고자 하는 결과는 결국 sum 연산의 결과값들을 획득하고자 하는 것이다. 여기서 주어지는 조건은 tab1에 대한 삽입 연산이 발생한다는 조건이 붙는다. 즉 tab1에 대한 삽입에 따라 위 그림 4.2의 질의는 수행 결과가 달라지게 된다. 다음의 그림 4.3은 그림 4.2에 대응해서 구현한 트리거 기반의 전짐적 뷰 형성 연산 예이다.

그림 4.3의 트리거 명세에서 key1_assign, compare_key2 및 select_key3는 각각 저장 프로시져로 구현한 함수들로서 그림 4.2의 질의 명세에서 조인 연산을 구현한다. key1_assign은 새로 삽입된 튜플 값 중 key1과 key2 값이 기존의 tab1_list에 존재하는지 여부를 검사하여 존재하면 TRUE를 존재하지 않으면 key1 및 key값을 갖는 새로운 레코드를 추가하고 FALSE 값을 반환한

다. compare_key2 및 select_key3 역시 비슷한 기능을 갖지만 반환 값이 BOOLEAN 값이 아니라 key4와 key3 형의 데이터를 반환한다는 차이를 갖는다.

5. 실험환경

대규모 트랜잭션 환경을 실험하기 위해 이 논문에서는 두 대의 서버를 이용한 모의 환경을 구성하여 실험

하였으며 실험을 위한 동작 시나리오는 다음의 그림 5.1과 같다.

단계 1 : 클라이언트는 릴레이 서버에게 데이터베이스 수정을 요청한다.

단계 2 : 릴레이 서버는 DB 서버에 데이터베이스 수정 연산을 제출한다.

단계 3 : DB 서버는 릴레이 서버의 수정 연산 요청의 처리 결과를 릴레이서버에 반환한다.

단계 4 : DB 서버의 질의 처리 결과를 반환받은 릴레이 서버는 클라이언트에 수정 완료신호를 반환한다.

단계 5 : 수정 완료 신호를 수신한 클라이언트는 MiddleTier DB 서버에게 검색 질의를 요청한다.

단계 6 : MiddleTier DB 서버는 분산 트랜잭션을 통해 검색된 질의 결과를 클라이언트에 반환한다

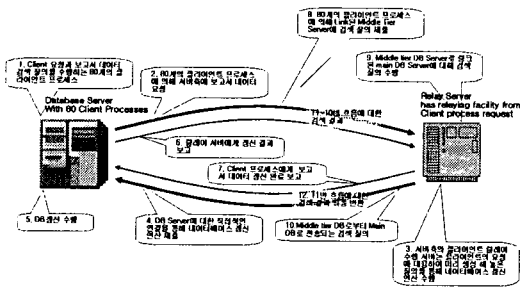

그림 5.1 질의 수행 시나리오

아울러 실험에 사용된 시스템 환경은 다음의 표 5.1과 같다.

표 5.1 실험에 이용된 시스템 환경

구분	사양	용도
SUN Fire v880 Server	CPU :900Mhz UltraSparc CPU × 2 MM : 4 GB HDD : 72GB 10k rpm FCAL DBMS : Oracle 9i R2 OS : Sun Solaris 8	DB 서버 & 80개의 클라이언트 프로세스 수행
SUN Blade 2000 W/S	CPU : 900Mhz UltraSparc CPU × 1 MM : 2 GB HDD : 72GB 10k rpm FCAL DBMS : Oracle 9i R2 OS : Sun Solaris 8	Relay Server, Middle Tier DB Server 역할을 수행

이와 같은 구성에서 실험은 트리거를 이용한 점진적 뷰 형성 연산 모델과 트리거를 이용하지 않는 순수 질의만을 이용한 결과를 비교하는 것으로 하였다. 다음의 그림 5.2는 실험 결과를 나타낸다.

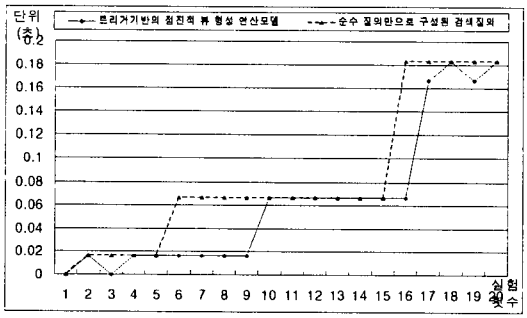

그림 5.2 실험 결과

그림 5.2의 결과에서 확인할 수 있는 바와 같이 트리거 기반의 점진적 뷰 갱신 형성 연산이 순수한 질의만을 이용한 연산 보다 높은 성능 특성을 나타냄을 확인 할 수 있다.

6. 결론

워게임 환경과 같이 대규모 트랜잭션 환경에서 실시간 보고서 생성이 가능하도록 하기 위해서는 기존의 질의만으로는 해결할 수 없다. 이와 같은 문제를 해결하기 위해 이 논문에서는 비 갱

신 형성뷰 및 갱신 형성 뷰를 이용하여 검색의 성능을 향상시킬 수 있는 모델을 제시하고 이의 구현 및 평가를 통해 제안 모델의 특성을 분석하였다. 실험 결과 형성 뷰를 이용한 질의가 형성 뷰를 이용하지 않는 질의에 비하여 상대적 으로 우수한 성능 특성이 나타남을 확인할 수 있었다. 따라서 제안 모델을 대규모 트랜 잭션 환경에서 실시간 보고서 생성을 위한 보고서 생성 모델로 채택하는 것이 타당하다.

참고문헌

[1]류근호, "시간지원 데이터베이스에서 뷰 형성 유지를 위한 실행트리," 한국정보과학회 논문지, 20권 8호, 1993

[2]Zohra Bellahsene, "Adapting Materialized Views after Redefinition in Distributed Environments," ER 2000, International Conference on Conceptual Modeling (ER) 2000, pp. 239-252, 2000

[3]Dimitri Theodoratos, "Detecting redundant materialized views in data warehouse evolution," Information Systems, Vol. 26, No. 5, pp. 363-381, 2001

[4]F. Fabret, M. Reginer, and E. simon, "An adaptive algorithm for incremental evaluation of production rules in databases", In Proceedings of the Nineth International Conference on VLDB, P 455-467, 1993

[5]K. D. Moon, Park Jeong Seok, Y. H. Shin, and K. H. Ryu, "Incremnetal Condition Evaluation for Active Temporal Rules," 4th International Conference on Intelligent Data Engineering and Automated, Springer, 2003

[6]S. Chakravarthy et al "HiPAC : Research project in active, time-constrained database management", Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, Massachusetts, 1989

[7]ANSI X3H2-99-079/WG3:YGJ-011(ANSI/ISO Working Draft) Foundation(SQL/Foundation), ANSI, 1999