論文
5-9-15

트리거 기반의 점진적 형성뷰 관리기법을 이용한 실시간 보고서 생성모델

Realtime Report Generation Model using Trigger Based Incremental Materialized View Maintenance Mechanism

이남일(Nam Il Lee)[1] 김진수(Jin Soo Kim)[2] 현득창(D. C. Hyun)[3]
류근호(Keun Ho Ryu)[4] 신예호,(Ye. Ho. Shin)[5]

요 약

정보화 및 온라인화의 발전에 따라 대규모 트랜잭션 환경에서 보고서의 의미는 특별하다. 이는 대규모 트랜잭션 연산을 수행하면서 성능의 제약 없이 제한된 시간 내에 보고서를 생성할 수 있어야 하기 때문이다. 이와 같이 대규모 트랜잭션 환경에서 시간제약을 만족하면서 보고서를 생성할 수 있도록 하기 위하여 이 논문에서는 점진적 연산 기법과 형성뷰 기법을 트리거와 저장 프로시져를 이용하여 결합시킨 모델을 제안한다. 아울러 제안 모델에 대한 구현 및 평가를 통해 제안 모델의 특성을 분석한다.

ABSTRACT

Reports have a significant meaning in large transaction environments, such as advanced the information technology and online environment . This is due to the necessity of generating reports within a given time limit 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 proposes 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 the identification of the characteristics of the proposed model.

논문접수 : 2004. 11. 15.
심사완료 : 2004. 12. 20.

1) 준회원 : 극동대학교 교육대학원
2) 준회원 : 육군교육사령부
3) 정회원 : 극동대학교 정보통신학부
4) 준회원 : 충북대학교 전기전자컴퓨터공학부
5) 종신회원 : 극동대학교 정보통신학부

❖ 이 연구는 산업자원부 지원 2004 지역혁신 특성화(RIS) 시범사업의 지원에 의해 수행되었음.

1. 서 론

통신 및 정보기술의 발달로 온라인 상에서 유통되는 데이터량의 증가가 폭발적으로 나타나고 있으며 이에 따라 과도한 데이터의 유통은 불필요한 데이터 필터링을 위해 필요한 연산 부하 발생의 부작용을 초래하고 있다. 온라인 은행업무, 온라인 증권 거래, 온라인 전자상거래 등 보편화된 온라인 환경은 이의 가장 대표적인 사례이다.

이와 같은 온라인 환경은 사용자의 요청에 대해 즉각적인 응답을 기대하는 경우가 대부분이며 이러한 요구사항, 즉 제한된 시간 내에서 사용자가 필요로 하는 요약 정보를 생성하기 위해서는 효율적인 데이터 요약 기법이 필요하다. 그러나 데이터베이스에서 보편적으로 이용되고 있는 기존의 데이터 요약 기술인 뷰(view or query modified view)나 질의(query)들은 대부분 요약 시점에 전체 데이터베이스를 대상으로 연산을 수행한다. 따라서 보고서 생성을 위한 데이터 요약 비용이 지나치게 높아 시간 제약을 갖는 온라인 환경에 적용하기에 다소 문제를 내포하고 있다.

이와 같은 문제를 해결하기 위해 이 논문에서는 점진적 갱신 기법을 기반으로 하는 형성뷰 모델을 도입한다. 형성 뷰는 데이터 요약 결과를 요청시간에 생성하는 것이 아니라 데이터베이스 갱신 시간에 구성한다. 따라서 데이터 요약을 요청하는 시점에는 이미 형성되어 있는 결과를 검색하는 수준에서 연산을 완료할 수 있게 되어 높은 성능 특성을 갖는다. 그러나 형성뷰 기법은 데이터 갱신 시점에 뷰 형성을 위한 추가적 연산 부하를 갖는 문제가 있다. 이 뷰 형성을 위한 연산 부하의 문제는 점진적 갱신기법을 통해 개선할 수 있다. 즉 뷰 형성 시점에 변경된 차분(difference)만을 대상으로 하는 갱신 연산을 수행함으로써 갱신 비용을 상당히 줄일 수 있으며 전체적으로 갱신 부하를 크게 줄일 수 있는 효과를 얻을 수 있다.

이 때 데이터 갱신 시점에 데이터 갱신에 대응해서 데이터 요약 연산을 기동시킬 수 있는 수단의 문제가 있다. 이 논문에서는 데이터 갱신 시점에 데이터 갱신에 대응해서 데이터 요약 연산을 개시하도록 하기 위한 수단으로서 데이터베이스 트리거(dataabse trigger)를 도입한다. 트리거는 데이터베이스 상태변경 연산에 대응해서 자동으로 조건부 조치를 수행할 수 있도록 하는 데이터베이스 프리미티브(database primitive)이다[1, 2].

이 논문에서는 앞에서 제시한 바와 같이 대규모 온라인 환경에서 효율적인 데이터 요약 기법으로서 트리거를 기반으로 하는 점진적 갱신 기법을 통한 형성뷰 연산을 트리거와 저장 프로시져를 통하여 구현하는 모델을 제안하고 이의 구현 및 실험을 통해 특성 분석을 수행한다. 이를 위한 이 논문의 구조는 다음과 같다. 제2장에서는 관련 연구로 점진적 기법과 형성 뷰 및 트리거와 관련된 기존 연구들을 분석한다. 제3장에서는 점진적 형성뷰 연산을 위한 연산 모델을 제시하고 제4장에서는 제안된 점진적 형성 뷰 연산 모델을 트리거와 결합하기 위한 모델을 제안하고 구현 예를 제시한다. 그리고 제5장에서 제안 모델의 실험을 위한 실험 환경을 제시하고 제6장에서 제시된 실험 환경에서 실험을 통

해 모델을 평가하며 제7장에서 결론을 내린다.

2. 관련 연구

생성규칙(production rule) 기반의 인공지능 분야에서 규칙의 조건평가 방법으로 제시되었던 점진적 연산 기법(incremental recomputation method)[3] 기법은 결과를 산출하기 위해 전체를 재계산하지 않고 계산 결과를 누적하면서 다음시점의 결과는 현재 시점의 결과에 다음 시점에 변경된 차분만을 반영하는 기법을 의미한다[4]. 이와 같은 점진적 연산 기법은 컴퓨터 과학 분야에서 자원의 효율적 운영과 성능 향상이라는 목표를 성취하기 위해 오래동안 논의되어 왔던 주제로서 데이터베이스 분야에서는 능동ㄷ데이터베이스의 핵심인 능동규칙 조건절의 최적화를 위해 이용되었으며[3,5,6] 최근에는 마이닝 분야[4,6,7] 등에도 활용되고 있다. 특히 [4] 및 [5]에서는 점진적 연산이 능동규칙의 조건절을 구성하는 복잡한 술어를 계산하는데 있어 갖는 효율성을 대수적으로 증명함으로써 점진적 연산의 효율성을 증명하고 있다.

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[8]. 이와 같은 형성 뷰는 분산 데이터베이스에서의 데이터 통합[9,10] 데이터 웨어하우스(data warehouse : dw)[11,12] 등에서 주요하게 이용되고 있다. 이 형성 뷰는 뷰 형성 방법에 따라 뷰 형성 시점에 전체 데이터베이스에 대한 완전한 재계산을 필요로 하는 재계산 방법과 뷰 형성 시점에 베이스 릴레이션에 가해진 변경 사항인 차분만을 반영하는 점진적 뷰 형성 기법으로 구분할 수 있다[8].

한편 데이터베이스 상태변경에 대응해서 자동으로 조건부 조치(conditional action)를 취할 수 있도록 하는 능동 규칙(active rule) 또는 데이터베이스 트리거(database trigger)는 HiPAC[13] 프로젝트 이후 데이터베이스의 주요한 요소로 확립되었다[1, 14]. 또한 최근 차세대 데이터베이스 언어 표준으로 확립된 SQL3[2]에서도 트리거(trigger)가 기본 요소로 포함되었고 대부분의 상용시스템에서도 기본적인 요소(primitive component)로 정착할 만큼 중요한 요소로 자리잡고 있다[15].

3. 트리거 기반의 점진적 형성뷰 연산모델

3.1 트리거

트리거는 데이터베이스 상태변경에 대응해서 자동으로 조건부 조치를 취할 수 있는 데이터베이스 프리미티브로서 다음의 (그림 3.1)과 같은 구문 구조를 갖는다.

```
CREATE TRIGGER <Trigger-name>
  [ <trigger action time> ] <trigger events> ON
   <table name>
  [ REFERENCING <temporal references> ]
  [ FOR EACH [ ROW | STATEMENT] ]
  [WHEN <trigger condition predicate>]
  BEGIN
    {<SQL DML> | <stored procedure>}+
  END;
```

(그림 3.1) SQL3 기반의 트리거 언어 구문

여기서 <trigger event>는 트리거를 기

동시키는 원인이 되는 연산으로서 삽입, 삭제, 갱신 연산과 "시스템 시작" 등과 같이 시스템 차원에서 정의되어 있는 사건들 중의 하나이며 이 연산이 <table name>에 가해졌을 때 트리거는 자동으로 처리 절차를 개시한다. 한편 <trigger action time>은 <trigger event>에 의해 명세된 연산이 <table name>에 가해지기 이전 또는 이후를 결정하는 옵션으로서 BEFORE 또는 AFTER를 선택적으로 적용할 수 있다[1, 2, 8]. REFERENCING 절은 <trigger event> 연산에 의해 데이터베이스에 가해지는 값들을 트리거 조치절에 전달하기 위한 수단이다. 따라서 이 REFERENCING 절을 이용하면 쉽게 차분을 확보할 수 있다[1, 2, 15].

한편 트리거 조치절(trigger action)은 BEGIN 과 END; 사이에 기술되는 <SQL DML> 또는 <SQL DML>을 포함하는 <stored procedure> 코드로 기술되며, 궁극적인 트리거 수행의 목표가 되는 연산들의 집합으로 정의할 수 있다.

3.2 점진적 뷰 형성 연산 모델

뷰 명세에 대응하는 값들을 물리적으로 관리하는 형성 뷰 V는 기본 테이블 R에 대한 뷰 정의 e에 의해 형성된 유도 릴레이션으로서 $V = e(R)$로 표현한다. 이와 같은 형성 뷰는 기본 테이블 R이 갱신 연산 u에 의해 기본테이블 R^u로 변경될 때 이 변경된 상태를 전파 받아야 한다. 이 때 기본 테이블의 변경에 대한 전파 연산의 형태에 따라 재계산 형성 뷰와 점진적 형서뷰로 구분할 수 있다. 재계산 뷰는 기본테이블 전체에 대해 뷰 정의 e에 의해 완전한 재계산 연산을 수행하는 형태이고 점진적 뷰는 변경된 차분에 대해서만 연산을 수행한다.

즉 기본테이블 R의 변경 연산 u에 의해 변경된 테이블 R^u의 차이값인 $|R - R^u|$에 해당하는 값을 차분 Δ라 하고 이 차분에 대해서만 뷰 정의 연산 e를 적용하는 연산규칙을 따른 형성뷰를 의미한다. 따라서 이 차분 Δ를 이용한 점진적 형성뷰의 뷰 형성 연산에 의해 규정되는 형성뷰 V_{new}는 다음과 같은 연산 의미를 갖는다.

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

여기서 핵심적 역할을 담당하는 요소는 점진적 연산을 가능하게 하는 수단인 차분(difference)이다. 뷰를 형성하기 위한 기본 테이블로부터 발생할 수 있는 차분으로는 데이터베이스 수정 연산인 삽입, 삭제, 및 갱신에 대응하는 삽입차분, 삭제차분 그리고 갱신 차분으로 구분할 수 있으며 이들은 각각 삽입차분 Δ^+_R, 삭제차분 Δ^-_R 그리고 갱신차분 Δ^u_R로 구성된다.

이와 같은 차분들 중 갱신 차분을 생성시키는 연산인 갱신 연산이 삭제 후 삽입으로 분해될 수 있으며 따라서 갱신 차분 역시 삭제차분과 삽입 차분을 이용하여 표현 할 수 있다. 즉 갱신 연산을 $insert(delete(R))$로 재정의 할 경우 갱신 연산에 의해 발생하는 차분은 삭제차분과 삽입 차분의 합집합으로 규정할 수 있다. 반면 순수한 삽입차분과 삭제 차분은 동일한 단위 연산 안에서 동시에 발생할 수 없다. 따라서 이와 같은 의미를 모두 반영한 차분 집합은 다음과 같은 정형의미를 갖는다.

$$\text{차분 } \Delta_R = \Delta_R^+ \cup \Delta_R^-$$

이와 같은 차분 의미를 이용하여 점진적 연산을 다시 정의하면 다음과 같다.

$$V_i = V_{i-1} \theta e(\Delta_R), \text{ 단 } \theta \in \{insert, delete, insert(delete)\}$$

이는 곧 i 시점의 차분에 대한 형성뷰 정의 연산을 수행한 결과를 이전 시점의 형성뷰에 합산하는 의미를 갖는 것으로서 형성뷰 정의 e를 이용하여 전개하면 다음과 같다.

$$V_i = (V_{i-1} - e(\Delta_R^-)) \cup e(\Delta_R^+)$$

즉 $i-1$ 시점에서의 형성뷰 상태에서 삭제 차분에 해당하는 데이터의 제거 또는 삽입차분에 해당하는 데이터의 추가 연산만을 수행하는 것을 의미한다.

3.3 트리거와 점진적 뷰 형성 연산의 결합

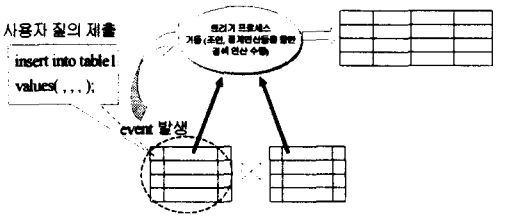
2장에서 트리거는 데이터베이스 상태변경 연산에 대응해서 자동으로 조건부 조치를 수행할 수 있도록 하는 데이터베이스 프리미티브임을 확인하였다. 이 트리거와 점진적 뷰 형성 연산 모델의 결합은 점진적 뷰 형성 연산을 트리거 조치절에 명세함으로써 가능해 진다.

일반적으로 트리거 조치절은 순수한 질의문, 또는 저장 프로시져를 통해 명세할 수 있다. 이 논문에서는 점진적 뷰 형성 연산을 저장 프로시져를 이용하여 트리거 조치절에 명세하는 방법을 이용하여 트리거와 점진적 뷰 형성 연산 모델을 구현한다. 아울러 점진적 뷰 형성 연산의 핵심 요소라 할 수 있는 차분의 획득은 트리거의 REFERENCING 절을 이용할 경우 간단하게 해결 할 수 있다. 이는 REFERENCING 절이 <trigger event> 연산에 의해 데이터베이스에 가해지는 값들을 포착해서 이를 트리거 조치절에 전달하는 역할을 수행하기 때문에 자연스럽게 차분을 확보할 수 있도록 해 준다. 이와 같은 트리거 특성을 이용하여 점진적 뷰 형성 연산과 결합된 트리거 프레임을 정의하면 다음의 (그림 3.2)와 같다.

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

(그림 3.2) 점진적 뷰 형성 모델을 위한 트리거 프레임

(그림 3.2)에서 삽입연산에 대응하는 점진적 뷰 형성 연산 모델을 위한 트리거 프레임을 정의하였다. 삭제 연산에 대응하는 트리거 프레임의 경우 REFERENCING 절의 참조자를 OLD AS <old alias>로 하면 되고 갱신 연산의 경우 NEW와 OLD를 모두 명세하면 된다. 이와 같은 구조에서 트리거 기반의 점진적 뷰 형성 연산 모델의 동작은 다음의 (그림 3.3)과 같다.



(그림 3.3) 트리거 기반의 점진적 뷰 형성 연산 동작 시나리오

(그림 3.3)에서 데이터베이스 상태 변경을 유발하는 사용자 질의가 발생할 경우 이 연산에 의해 트리거 사건이 발생한다. 사건이 발생하면 트리거 수행이 개시되면서 사건을 발생시킨 사용자질의로부터 사건발생 데이터를 획득하는 REFERENCING 절이 작동해서 차분을 획득하고 획득된 차분을 트리거 조치절에 전달한다. 트리거 조치절은 트리거 조치절 내에 명세된 점진적 뷰 형성 연산의 수행을 통해 형성 뷰의 상태를 관리함으로써 기본 테이블에 가해진 변경을 형성 뷰에 전파하는 동작을 완료한다.

4. 점진적 뷰 형성 연산의 구현

이 장에서는 3.3절에서 구성한 점진적 뷰 형성 모델을 위한 <trigger body> 부분을 구성할 점진적 연산의 구현 예를 제시한다. (그림 3.2)의 트리거 프레임에서 <trigger body> 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. 이 절에서는 다음의 6장 실험에서 적용할 형성 뷰를 위한 뷰 형성 연산을 예시한다. 이를 위해 다음의 그림 4.1에 뷰 명세를 위한 검색 질의를 예시한다.

```
select sum(aa.attrv1), sum(aa.attrv2), sum(aa.attrv3),
       sum(aa.attrv4), sum(aa.atrv5)
from ( select a.key1, a.attrv1, a.attrv2, a.attrv3,
              a.attrv4, a.attrv5,
       from tab1 a, tab2 b
       where a.key2 = b.key2 ) aa
where aa.key1 = 사용자지정코드
group by aa.key1
```

(그림 4.1) 뷰 명세를 위한 검색 질의

(그림 4.1)에서 명세하는 질의는 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. 이와 같은 질의를 형성 뷰를 이용하여 구현하기 위해서는 먼저 질의의 최종 결과인 다섯 개의 속성에 대한 sum 연산 결과를 유지하기 위한 속성과 group by 연산에 대응하는 key1 속성을 포함해야 한다. 반면 질의 내부에서 조인 조건으로 참여하는 key2는 질의 내부에서 중간 결과들을 생성하는데만 영향을 미칠 뿐 최종 결과를 구성하는 데는 관여하지 않으므로 형성 뷰 테이블에는 포함할 필요가 없다. 따라서 (그림 4.1)의 질의에 대응하는 형성 뷰 테이블의 구조는 다음의 (그림 4.2)와 같은 구문에 의해 생성된다.

```
CREATE TBALE VIEWTAB(
key1    tab1.key1%TYPE          NOT NULL,
attrv1  NUMBER   NOT NULL,
attrv2  NUMBER   NOT NULL,
attrv3  NUMBER   NOT NULL,
attrv4  NUMBER   NOT NULL,
attrv5  NUMBER   NOT NULL
);
```

(그림 4.2) (그림 4.1) 질의를 위한 뷰 형성 테이블 생성

(그림 4.1)의 질의에 의해 생성된 테이블에 대하여 질의 의미를 반영한 뷰 형성 연산은 저장 프로시져(stored procedure)를 이용하여 구현한다. 이는 저장 프로시져가 질의와 강 결합된 상태에서 절차적 프로그램이 가능할 뿐만 아니라 <trigger body>를 명세할 수 있는 가장 적합한 수단이기 때문이다. (그림 4.1)에 대응한 뷰 형성 연산을 구성하기 위해 먼저 전역변수 선언이 필요하다. 여기서 선언하는 전역 변수는 점진적 연산 과정에서 뷰 테이블의 튜플에 대한 갱신 또는 새로운 튜플

의 삽입 여부를 판단하는데 이용되는 전역 변수들로 모두 리스트 구조를 갖는다.

```
tab1_list  tab1_list_type ; // key1, key2
를 갖는 레코드의 리스트
```

여기서 tab1_list의 경우 key1과 key2의 값을 유지하기 위한 레코드 구조를 기반 구조(element structure)로 갖고 있으며 이 레코드 구조를 다시 리스트 구조로 하는 2차 구조를 취한다. 이와 같이 하는 이유는 이들 레코드의 리스트가 해당 속성 값을 메인 메모리에 유지함으로써 데이터베이스에 대한 접근 없이 (그림 4.1)의 질의에서 명세하는 조인 연산을 트리거 바디 부분에서 구현할 수 있도록 하기 때문이다. 한편 조인 결과 역시 메모리 상에서 유지될 수 있어야 한다. 왜냐하면 점진적 연산의 핵심이 바로 이전 시점의 조인 결과 유지와 이를 토대로 한 차분 연산에 있기 때문이다. 이를 위해 (그림 4.2)에서 명세한 VIEWTAB 테이블의 구조와 일치하는 구조를 갖는 전역변수가 추가로 필요하다. 이는 행 타입(row type)을 이용하여 생성하며 리스트 구조를 갖도록 구성한다.

```
TYPE  material_type  IS  TABLE  OF
VIEWTAB%ROWTYPE
INDEX BY BINARY_INTEGER;

material_1  material_type ; //(그림 4.1)의 질의를 위한
                            뷰 형성 결과 유지 전역변수
```

이 material_1 레코드는 데이터베이스 시작 초기에 VIEWTAB에 있는 레코드들을 모두 가질 수 있도록 초기화 되며 다음의 (그림 4.3)에서 명세하는 트리거에 의해 변화되는 상태를 추적 유지할 수 있도록 한다. 이와 같은 조건 하에서 점진적 연산을 위한 완전한 트리거 선언부는 다음의 (그림 4.3-a)와 같다.

```
TRIGGER InsertTab1Trigger
AFTER INSERT ON TAB1
REFERENCING NEW AS newTab1
FOR EACH ROW
DECLARE
  key1_ok          BOOLEAN ;
  material_cnt     NUMBER ;
```

(그림 4.3-a) 갱신 뷰 형성 질의를 위한 점진적 트리거 선언부

트리거 선언부에서 선언하는 변수들 중 key1_ok는 새로 데이터베이스에 반영되는 레코드가 (그림 4.2)에서 제시하는 VIEWTAB 내에 존재하는 레코드인지 아니면 존재하지 않는 전혀 새로운 레코드인지를 판단하기 위한 변수이다. 그리고 _cnt로 종료하는 변수는 카운터 변수이다. 이렇게 선언된 변수들에 대한 초기값 설정은 다음의 (그림 435-b)에서 명세하는 코드에 의해 수행된다.

```
BEGIN
  key1_ok := key1_assign(tab1_list, :newTab1.key1,
:newTab1.key2) ;
```

(그림 4.3-b) 변수 설정 코드

(그림 4.3-b)에서 BEGIN은 선언부에 이어 바로 나타나는 구문으로서 트리거 조치절의 코드 시작을 나타낸다. 아울러 key1_assign은 (그림 4.3-a)의 REFERENCING 절에서 선언한 전이값 레코드 newTab1의 값들을 이용하여 새로 삽입되는 레코드가 (그림 4.2)의 VIEWTAB에 존재하는 레코드 인지 아니면 존재하지 않는 레코드 인

지를 판단하는 함수이다. 이미 존재하는 레코드라면 key1_ok는 TRUE값을 가지며 그렇지 않다면 FALSE값을 갖는다. 이렇게 설정된 key1_ok값을 이용한 점진적 뷰형성 연산 코드는 다음의 (그림 4.3-c)와 같다.

```
/* VIEWTAB에 이미 존재하는 레코드에 대한 처리 부분*/
IF key1_ok = TRUE THEN
    FOR material_cnt IN material_1.FIRST..material_1.LAST
      LOOP
       IF material_1(material_cnt).key1 = :newTab1.key1
THEN
            key1 속성을 제외한 material_1(material_cnt)의
            모든 속성값 1 증가
        END IF ;
      END LOOP ;

/* VIEWTAB에 존재하지 않는 레코드에 대한 처리 부분*/
ELSIF key1_ok = FASLE THEN
  material_1 리스트의 material_1.LAST+1 위치에 새로운 레코드
  값 설정
     //key1 : = :newTab1.key1 : 나머지 속성은 모두 1 :
END IF :

/* 위 연산에서 갱신된 상태를 일괄적으로 데이터베이스에 반영
*/
UPDATE 연산 수행
    // 위 알고리즘에서 변경된 레코드들을 일괄적으로 데이터베
      이스에 반영
END TRIGGER :
```

(그림 4.3-c) 갱신 뷰 형성 질의를 위한 점진적 트리거 바디

이와 같은 구조를 통해 데이터베이스 삽입이 발생할 때 마다 삽입된 레코드의 상태를 판단하여 자동으로 형성뷰의 상태를 점진적으로 갱신할 뿐만 아니라 모든 연산들을 메모리에서 수행하도록 하므로써 연산 성능의 향상은 물론 갱신시 발생하는 연산 부하를 상당히 줄일 수 있는 장점이 있다.

5. 실험 환경

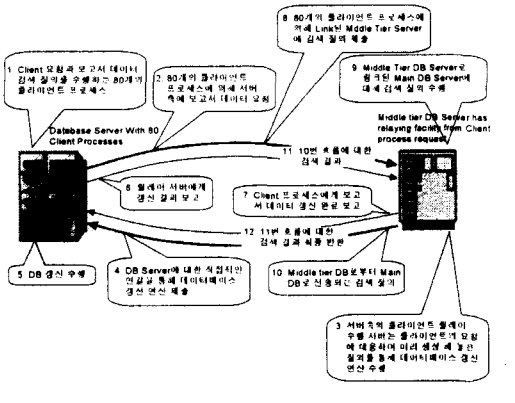
이 논문에서는 트리거를 기반으로 하는 점진적 뷰 형성 모델의 성능 특성을 파악하기 위해 다음의 <표 5.1>과 같은 유형의 질의문 집합을 설정하였다.

〈표 5.1〉 실험에 사용된 질의 유형

유 형	질의 문수	비 고
비 갱신 형성뷰 이용 질의	2	갱신이 발생하지 않는 형성 뷰 이용 질의
갱신 형성뷰 이용 질의	2	트리거 기반의 점진적 뷰 형성 기법을 이용하여 관리되는 형성 뷰 이용 질의
형성뷰를 이용하지 않는 질의	4	위 두 가지 유형의 질의에 대해 동일한 의미를 갖으면서 형성뷰를 이용하지 않는 질의

특히 갱신이 발생하는 형성 뷰의 경우 조인 및 집계 질의를 포함하는 복잡한 술어를 포함하도록 구성하여 성능 평가의 특성을 명확히 할 수 있도록 하였다.

한편 실험에 참여하는 테이블의 수는 모두 7개이며 이 중 조인에 참여하는 테이블은 5개이다. 그리고 각각의 테이블에는 1500건 이상의 튜플들을 갖고 있으며 각각의 질의 유형마다 20개의 갱신 연산과 검색연산을 쌍으로 구성하였다. 다음의 (그림 5.1)은 이와 같은 환경을 반영한 모의 실험환경이다.



(그림 5.1) 모의 환경의 구성 및 모의 환경에서의 질의 수행 시나리오

(그림 5.1)에서 도시하고 있는 질의 수행 시나리오는 분산 환경을 고려한 다중 시스템 내에서의 질의 수행을 설명한다. 먼저 질의 참여 요소들을 고찰하면 첫째 클라이언트는 질의 수행을 요청하는 프로세스로 정의할 수 있으며 최초 4개의 프로세스 생성 및 실행이 발생하고 그 결과를 측정한다. 그런 다음 다시 8개의 프로세스 생성 및 수행과 그 결과 측정이 이루어진다. 이와 같은 동작은 80개의 프로세스 생성 및 수행 단계까지 매 단계마다 수행되어야 할 프로세스의 수를 4개씩 증가시키면서 진행된다.

다음으로 데이터베이스 서버는 검색 대상 데이터를 관리하는 역할을 담당한다. 그리고 릴레이 서버(Relay Server)는 클라이언트의 요청에 대응한 데이터베이스 갱신 연산을 실행하는 프로세스이며 MiddleTire DB 서버는 분산 데이터베이스 기법을 통해 원격지 데이터베이스에 대한 접근을 허용하는 역할을 담당한다. 이와 같은 시나리오를 수행하기 위한 수행 단계를 요약하면 다음과 같다.

단계 1 : 클라이언트는 릴레이 서버에게 데이터베이스 수정을 요청한다.

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

단계 3 : 데이터베이스 서버는 릴레이 서버의 수정 연산 요청의 처리 결과를 릴레이 서버에 반환한다.

단계 4 : 데이터베이스 서버의 질의 처리 결과를 반환 받은 릴레이 서버는 클라이언트에 수정 완료 신호를 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.

여기서 주목해야할 단계는 단계 2이다. 이 단계 2에서 릴레이 서버가 데이터베이스 서버에게 수정 연산을 제출하게 되고 제출된 수정 연산이 수행되는 동안 데이터베이스 서버에서는 트리거와 결합된 점진적 뷰 형성 연산에 의해 뷰 갱신이 자동으로 수행되기 때문이다. 이와 같은 실험 환경에서 성능 평가는 먼저 점진적 뷰 형성 연산을 자동으로 수행하는 트리거를 활성화시킨 상태에서 성능평가를 수행하고 다시 데이터베이스를 초기 상태로 환원한 뒤 뷰 형성 연산을 수행하는 트리거를 비활성화시킨 상태에서 형성뷰를 이용하지 않는 질의를 이용하여 성능 평가를 수행하는 과정을 통해 성능 평가를 수행한다.

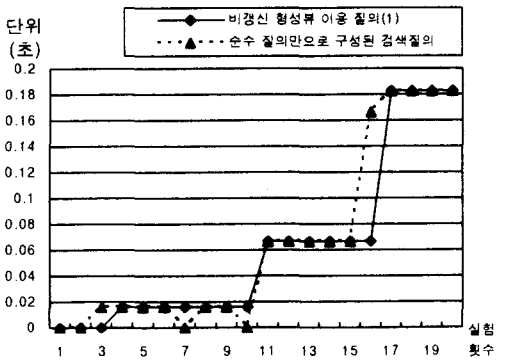
6. 성능 평가

이 장에서는 앞에서 구성한 모델 및 실험환경을 토대로 성능 평가를 수행한다. 성능 평가 수행결과 획득한 결과를 분석한다. 먼저 실험에 사용된 시스템 환경은 다음의 <표 6.1>과 같다.

〈표 6.1〉 실험에 이용된 시스템 환경

구 분	사 양	용 도
SUN Fire v880 Midrange Server	CPU : 900Mhz Ultra Sparc CPU×2 MM : 4GB HDD : 72GB 10k rpm Fibre Cahnnel HDD×5 DBMS : Oracle 9i Enterprise Edition OS : Sun Solaris 8 Operating environment	DB 서버 & 80개의 클라이언트 프로세스 수행
SUN Blade 2000 Workstation	CPU : 900Mhz Ultra Sparc CPU×1 MM : 2GB HDD : 72GB 10k rpm Fibre Cahnnel HDD×2 DBMS : Oracle 9i Enterprise Edition OS : Sun Solaris 8 Operating environment	Relay Server & Middle Tier DB Server 역할을 수행

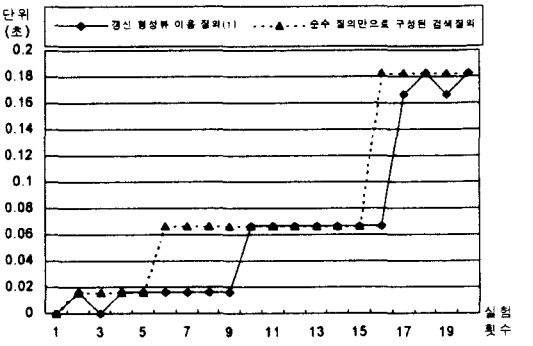
이와 같은 구성에서 첫 번째 수행한 실험은 비 갱신 뷰를 이용하는 질의와 동일한 의미의 뷰를 이용하지 않고 순수한 질의만으로 구성된 질의 수행 결과를 비교하는 것이다. 다음의 (그림 6.1) 실험의 평가 결과를 나타낸다.



(그림 6.1) 비 갱신 형성뷰 이용 질의에 대한 실험 결과

(그림 6.1)의 실험에 참여한 형성 뷰는 갱신이 발생하지 않는 형성 뷰를 이용한다. 이 비 갱신 형성 뷰는 두 개의 테이블로부터 조인 연산을 통해 결과를 검색하는 연산을 내포한다. 그리고 이 형성 뷰의 값들과 본 질의의 조인 참여 테이블이 다시 조인되는 구조를 갖는다. (그림 6.1)의 비 갱신 형성 뷰 이용 질의는 형성 뷰를 이용한 질의 수행을 명세한 질의이며 순수한 질의만으로 구성된 질의는 형성 뷰를 이용하지 않고 형성뷰에 형성 해야할 결과를 질의 수행 시점에 직접 조인 연산으로 명세한 질의를 의미한다. 두 가지 질의 수행 결과 실험 횟수가 적어서 동시 트랜잭션 수가 적을 때는 단순 질의가 약간 좋은 성능을 나타내나 실험 횟수의 증가에 따른 동시 트랜잭션 수가 증가하면서 형성뷰를 이용하는 질의가 약간 좋은 성능을 나타내고 있음을 확인할 수 있다.

이제 갱신 형성 뷰에 대한 실험 결과를 제시한다. 갱신 형성 뷰는 질의 수행 과정에서 데이터베이스에 가해지는 갱신 연산이 검색 질의의 조인 항에 영향을 미치는 상태를 가정하는 질의이다. 이 논문에서는 이와 같은 상황을 개선하기 위해 갱신 질의가 검색 질의에 영향을 미치는 경우 이를 트리거와 결합된 점진적 뷰 형성 연산을 통해 해결하였다. 갱신 형성 뷰 이용 질의의 두 가지 유형에 대한 실험 결과를 다음의 (그림 6.2)와 (그림 6.3)에 도시하였다.

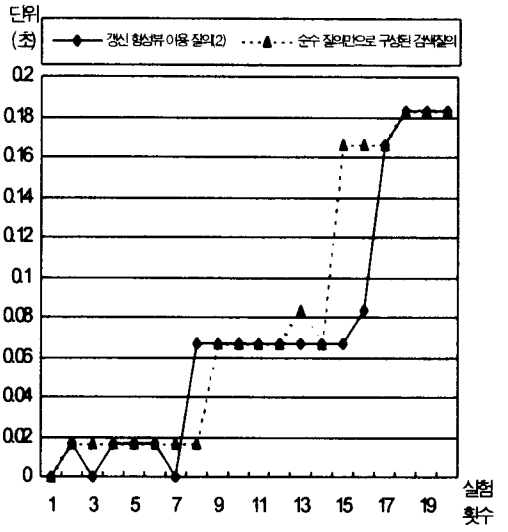


(그림 6.2) 갱신 형성 뷰 이용 질의에 대한 실험 결과(유형1)

(그림 6.2)에서 갱신 형성 뷰 이용 질의와 순수한 질의만으로 구성된 질의 사이의 성능차이가 크게 나타나는 이유는 (그림 6.2)를 명세하는 형성 뷰 이용 질의가 여러개의 집계함수와 조인복잡도를 갖는 질의 유형으로 구성되었기 때문인 것으로 판단할 수 있다.

(그림 6.3)에서 갱신 형성 뷰 이용질의(유형2)의 수행 결과가 (그림 6.2)의 갱신 형성 뷰 이용질의의 수행 결과와 비교하여 갱신 형성 뷰 이용 질의와 순수한 질의 사이의 성능 차이가 다소 감소하는 추세를 보이는데 이는 (그림 6.3)을 명세하는 질의가 (그림 6.2)를 명세하는 질의와 유사한 특성을 갖지만 (그림 6.2)를 명세

하는 질의에는 포함되어 있지 않은 외부 조인을 포함하고 있을 뿐만 아니라 인라인 뷰에서 상수값을 이용한 조인술어 제한이 포함되기 때문에 (그림 6.2)의 명세가 연산 부하를 덜 받기 때문인 것으로 판단할 수 있다.



(그림 6.3) 갱신 형성 뷰 이용 질의에 대한 실험 결과(유형2)

(그림 6.2) 및 (그림 6.3)은 각각 트리거와 결합된 점진적 뷰 형성 연산을 기반으로 하는 갱신 형성 뷰를 이용한다. 그리고 질의가 단순히 조인만을 명세하는 것이 아니라 조인 및 집계 함수를 포함한 복잡한 질의 상황을 내포한다. (그림 6.3) 및 (그림 6.4)가 형성 뷰를 이용하지 않는 질의와 확연한 성능 차이를 보이는 이유는 트리거와 결합된 점진적 뷰 형성 연산의 효율성에 의한 연산 이익이 크게 발생하기 때문에 즉, 뷰 형성에 소요되는 비용이 상대적으로 적게 나타나기 때문에 발생하는 현상으로 파악할 수 있다.

한편 (그림 6.1)에서 (그림 6.3)에 이르는 실험은 3가지 질의 유형에 대하여 각각 20개의 질의로 구성된 60개의 프로세스를 동시에 생성한 후 (그림 5.1)의 시나리오에 따라 수행되도록 구성하였다. 이와 같은 구성은 대규모 온라인 환경에서 대규모의 다중 클라이언트가 동시에 접속하는 상황을 모의하기 위하여 구성 한 것이다. 따라서 60개의 개별 프로세스는 각각 독립적인 클라이언트로 간주 할 수 있으나 동일한 시스템 내에서 수행되기 때문에 프로세스들 사이의 CPU 스케줄링 문제가 발생한다. 따라서 각각의 질의 유형에 대한 실험횟수의 증가는 결국 프로세스들 사이의 경합에 의한 지연이 발생하게 되는 문제를 피할 수 없다. 그러므로 (그림 6.1)에서 (그림 6.3)에 이르는 실험 결과들이 후반부로 갈수록 소요시간의 증가가 나타나는 이유는 각각의 프로세스에 필요한 단위 시간에 더하여 프로세스들 사이의 경합에 의해 발생하는 지연이 포함되어서 나타나는 것으로 평가 할 수 있다.

7. 결 론

온라인 뱅킹 등과 같은 대규모 온라인 트랜잭션 환경에서 실시간 보고서 생성이 가능하도록 하기 위해서는 기존의 질의만으로는 해결할 수 없다. 이와 같은 문제를 해결하기 위해 이 논문에서는 점진적 갱신 기법을 기반으로 하는 형성 뷰 기법을 도입하였다. 형성 뷰는 데이터베이스 갱신 시점에 데이터베이스 상태변경에 대응해서 요약데이터를 물리적으로 형성하도록 하는 것으로서 요약데이터를 생성하는데

있어 전체 데이터베이스를 대상으로 하지 않고 이미 형성되어 있는 형성뷰에서 간단한 검색 연산만으로 데이터요약이 가능한 효율적 도구이다. 그러나 이 형성뷰는 데이터베이스 갱신 시점에 요약데이터를 생성해야 하므로 데이터베이스 갱신 부하가 게 걸리는 문제가 있기 때문에 긴 갱신주기와 짧은 검색주기를 갖는 응용에 효과적으로 적용할 수 있는 방법이다.

이와 같은 형성뷰의 특성 중에서 데이터베이스 갱신 시점에 발생하는 연산 부하를 최소화 함으로써 대규모 온라인 환경에서 효과적인 데이터 요약기법으로 활용하기 위해 이 논문에서는 비 갱신 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 상대적으로 우수한 성능 특성이 나타남을 확인할 수 있었다. 이와 같은 특성은 제안 모델에서 채택하고 있는 형성 뷰를 이용한 연산이 형성 뷰를 이용하지 않는 연산에 비하여 질의 수행을 위한 연산 부하가 작게 걸리기 때문에 발생하는 현상이다. 따라서 제안 모델을 대규모 트랜잭션 환경에서 실시간에 보고서를 생성하기 위한 보고서 생성 모델로 채택하는 것이 타당하다.

한편 이 연구에서는 대규모 트랜잭션 환경을 모의하기 위해 동일 시스템 내에서 다중의 프로세스를 생성하는 방법을 이용하였으나 이는 프로세스들 사이의 간섭 및 경쟁에 의한 부하 요소가 포함됨으로써 제안 모델의 성능 평가를 정확하게 할 수 없는 원인이 되었다. 또한 실험에 참여한 테이블들에 포함된 튜플 수가 충분히 크지 않은 문제점도 내포하고 있다. 따라서 제안 모델의 특성을 정확하게 파악하기 위해서는 데이터량 및 실험 환경이 좀 더 현실에 가까운 환경에서 평가해 볼 필요가 있을 것으로 판단된다.

참 고 문 헌

[1] J. widom, S. Ceri, "chapter 1, Introduction to Active Database systems(Triggers and Rules for Advanced Database Processing)," Morgan kaufman Publishers, pp. 1-42, 1996.

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

[3] 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, pp.455-467, 1993.

[4] 신예호, 류근호, "실시간 연관규칙 탐사를 위한 능동적 후보항목 관리모델", 정보처리학회논문지, 제9권 제2호, 2002.

[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] E. Hanson, Sreenath Bodagala, Mohammed Hasan, Goutam Kulkarni, and Jayashree Rangarajan, "Optimized rule condition testing in ariel using gator networks," Technical Report TR-95-027, University of florida CIS Dept., Oct., 1995.

[7] David Wai-Lok Cheung, Jiawei Han, Vincent Ng, C. Y. Wong : Maintenance of Discovered Association Rules in Large Databases : An Incremental Updating Technique. ICDE : pp.106-114, 1996.

[8] 류근호, "시간지원 데이터베이스에서 뷰 형성 유지를 위한 실행트리", 한국정보과학회 논문지, 제20권 제8호, 1993.

[9] Mukesh K. Mohania, Guozhu Dong, "Materialized View Adaption in Distributed Databases," Asian Computing Science Conference, (ASIAN)'96, pp.353-354, 1996.

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

[11] Chuan Zhang, Jian Yang, "Materialized View Evolution Support in Data Warehouse Environment," (DASFAA)'99, pp.247-254, 1999.

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

[13] 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.

[14] Norman. W. Patton, andrew. Dinn, M. Howard Williams, "Active Rules in Database systems," Springer, 1998.

[15] "Oracle 9i Administration Guide," Oracle Press, 2003.

이남일

1987년청주사대수학교육과 졸업
2003년 극동대학교 교육대학원 컴퓨터교육 전공 입학
2002년 ~ 현재 여주여중 재직
관심분야 : 멀티미디어 데이터베이스, 교육정보화시스템

김 진 수

1981년 계명대학교 수학과(학사)
1987년 국방대학교 전산학과 (이학석사)
1998년 충북대학교 전산학과 박사과정이수
2002년 ~ 현재 육군교육사 체계분석실 근무
관심분야 : 실시간 객체, 분산컴퓨팅, 시공간 데이터베이스

현득창

email : hyundc@kdu.ac.kr

류 근 호

1976년숭실대학교 전산학과(이학사)
1980년연세대학교 공학대학원 전산전공 (공학석사)
1988년연세대학교 대학원 전산전공 (공학박사)
1976년 ~ 1986년 육군군수 지원사 전산실(ROTC 장교), 한국전자통신연구원(연구원), 한국방송통신대 전산학과 (조교수) 근무
1989년 ~ 1991년 Univ. of Arizona Research Staff(TempIS 연구원, Temporal DB)
1986년 ~ 현재 충북대학교 전기전자 및 컴퓨터공학부 교수
관심분야 : 시간 데이터베이스, 시공간 데이터베이스, Temporal GIS, 객체 및 지식베이스 시스템, 에이젼트기반 정보검색 시스템, 데이터마이닝, 데이터베이스 보안 및 Bioinformatics 등

신 예 호

1996년군산대학교 컴퓨터과학과 (학사)
1998년충북대학교 대학원 전자계산학과 졸업 (석사)
1998년충북대학교 대학원 전자계산학과 입학(박사과정)
2002년 충북대학교 대학원 전자계산학과 졸업 (이학박사)
2002년 ~ 현재 극동대학교 정보통신학부
관심분야 : 능동 데이터베이스, 시간 데이터베이스, 공간 데이터베이스, 데이터 마이닝