대규모 트랜잭션 환경에서의 실시간 보고서 생성을 위한 점진적 형성뷰 관리모델

김 진 수[†] · 신 예 호[††] · 류 근 호[†††]

요 약

항공관제 시스템, 워게임 등과 같이 시간제약을 갖는 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 프로시저를 이용하여 결합시킨 모델을 제안한다. 아울러 제안 모델에 대한 구현 및 평가를 통해 제안 모델의 특성을 분석한다.

Incremental Materialized View Management Model for Realtime Report Generation on Large Transaction Processing Environment

Jin Soo Kim[†] · Ye Ho Shin[††] · Keun Ho Ryu[†††]

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 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.

키워드 : 대규모 트랜잭션 환경(Large Transaction Environments), 실시간 보고서 생성(Realtime Report Generation), 점진적 형성뷰 관리모델(Incremental Materialized View Maintenance Model)

1. 서 론

수많은 항공기들의 이/착륙 및 안전한 항로 유지를 책임지고 관리해야 하는 항공관제 시스템은 수 많은 항공기들로부터 실시간으로 획득되는 좌표 및 고도 정보를 토대로 항공기들을 안전하게 유도하고 조정하는 역할을 수행한다. 이와 같은 항공 관제 시스템이 안전하게 항공기들을 유도하기 위해서는 실시간으로 획득되는 대규모 데이터들로부터 적절한 시간 내에 항공기들의 안전한 운항에 장애가 될 수 있는 사항들을 추출해 낼 수 있어야 한다. 한편 가상의 환경에서 전쟁 수행 능력을 훈련하는 워게임(war game)의 경우 게임 참가자들에 의해 취해지는 모든 전투 행위가 게임 서버의 연산으로 사상되며 따라서 게임이 진행되는 동안은 게임 참가자들에 의해 설정되는 모든 상태들을 포함하는 대규모 트랜잭션이 상시적으로 나타나게 된다. 워게임 환경에서 게임 참가자들은 자신들의 전술 및 전략의 수행을 위해 수시로 자신들의 상태를 파악할 수 있어야 한다.

항공관제 시스템이나 워게임과 같이 대규모 트랜잭션 환경에서 제한된 시간 즉 시간 제약을 충족하면서 보고서를 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 대규모 트랜잭션 환경에 적용하기 어렵다.

이와 같은 문제를 해결하기 위해 이 논문에서는 점진적 갱신 기법을 기반으로 하는 형성뷰 기법을 도입한다. 형성

※ 이 연구는 건설교통부 국토연구원 NGIS과제지원과 과학기술부 RRC(정보통신 연구센터)의 연구비 지원으로 수행되었음.
† 정 회 원 : 육군 교육사령부 체제분석실
†† 정 회 원 : 국동대학교 정보통신학부
††† 종신회원 : 충북대학교 전기전자 및 컴퓨터공학부 교수
논문접수 : 2003년 9월 18일, 심사완료 : 2003년 10월 7일

뷰는 데이터 요약 결과를 요청시간에 구성하는 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.

점진적 갱신 연산을 기반으로 하는 형성 뷰 기법은 데이터베이스 갱신 시점에 점진적 뷰 형성 연산을 수행해야 한다. 이를 위해 이 논문에서는 뷰 형성 연산을 개시하는 메커니즘으로 트리거(trigger)를 이용한다. 트리거는 데이터베이스 상태변경 연산에 대응해서 자동으로 조건부 조치를 수행할 수 있도록 하는 데이터베이스 프리미티브(database primitive)이다[1, 2]. 점진적 뷰 형성 연산은 트리거의 조치절에 명세 함으로써 트리거의 자동화된 조건부 조치 기능을 통한 점진적 뷰 형성 연산의 호출이 가능해 진다. 트리거 조치절에 명세되는 점진적 뷰 형성 연산은 저장 프로시져를 통해 명세한다.

이 논문에서는 앞에서 제시한 바와 같이 대규모 트랜잭션 환경에서 보고서를 생성하기 위한 효율적 방안으로 점진적 갱신 기법을 바탕으로 한 형성뷰 연산을 트리거와 저장 프로시져를 통하여 구현하는 모델을 제안하고 이의 구현 및 실험을 통한 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 통해 모델을 평가하며 제7장에서 결론을 내린다.

2. 관련 연구

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[3]. 이와 같은 형성 뷰는 분산 데이터베이스에서의 데이터 통합[4, 5] 데이터 웨어하우스[6, 7] 등에서 주요하게 이용되고 있으며 특히 데이터웨어하우스(data warehouse : dw) 분야의 경우, 상용 시스템에서 제공하고 있을 정도로 보편적인 도구로 이용되고 있다[8]. 이와 같은 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[3].

한편 점진적 뷰 형성 연산의 기반이 되는 점진적 기법(incremental method)은 컴퓨터 과학 분야에서 자원의 효율적 운영과 성능 향상이라는 목표를 성취하기 위해 오래동안 논의되어 왔던 주제이다[9-13]. 이 점진적 연산 기법은 결과를 산출하기 위해 전체를 재계산하지 않고 계산 결과를 누적하면서 다음시점의 결과는 현재 시점의 결과에 다음 시점에 변경된 차분만을 반영하는 기법을 의미한다[13]. 이와 같은 점진적 기법은 주로 능동데이터베이스에서 조건절의 최적화를 위해 이용되었으나[11, 12, 14] 마이닝 분야[13-15] 등에도 활용되었다. 특히 [12] 및 [13]에서는 점진적 연산이 능동규칙의 조건절을 구성하는 복잡한 술어를 계산하는데 있어 갖는 효율성을 대수적으로 증명함으로써 점진적 연산의 효율성을 증명하고 있다.

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[16] 프로젝트 이후 데이터베이스의 주요한 요소로 확립되었다[1, 17]. 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[8]. 이 논문은 형성 뷰의 뷰 형성을 위해 트리거 기능과 점진적 갱신 기법을 결합하여 구현하는 모델을 적용한다. 이는 효율성이 입증된 점진적 연산 기법을 트리거를 통해 자동 수행하도록 함으로써 사용자의 개입 없이 자동으로 형성뷰의 뷰 형성을 효과적으로 수행할 수 있도록 하기 위한 방안으로서 적합하기 때문이다. 다음의 3장에서 트리거와 점진적 갱신 기법을 결합한 점진적 뷰 형성 연산 모델을 기술한다.

3. 뷰 형성을 위한 점진적 연산 모델

이 장에서는 뷰 형성 연산의 효율성을 증대하기 위한 점진적 뷰 형성 연산 모델을 정립한다. 이를 위해 먼저 뷰에 대한 대수적 정의를 수행하고 이 정의된 뷰 의미를 토대로 한 점진적 연산 모델을 정립한다. 아울러 정립된 점진적 뷰 형성 연산 모델을 트리거와 결합하기 위한 방안도 함께 제시한다.

3.1 트리거와 결합한 점진적 뷰 형성 연산 모델

일반적으로 데이터베이스에서 접근 빈도가 높은 자료를 형성 뷰 기법으로 유지 관리할 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(materialized view)

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

$$V=e(R)$$

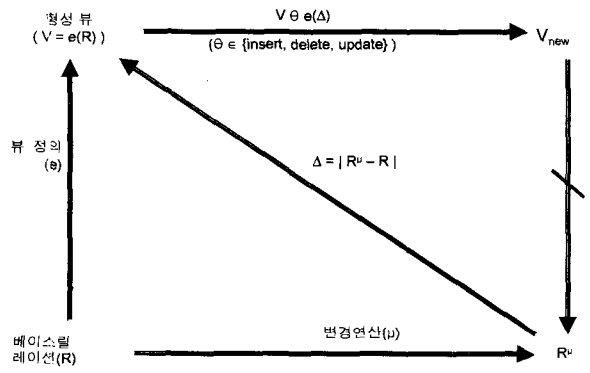
- 기본 테이블 R의 변경 연산 μ에 의해 기본 테이블은 R^{μ}로 변경되며 이는 뷰의 갱신을 유도한다(p->는 전파 연산을 나타낸다).

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

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

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

(그림 3.1)은 [정의 3.1]의 형성 뷰를 도식화한 것이다. (그림 3.1)에서 기본 테이블의 변경 정보는 뷰에 전파되지만 뷰 자체적인 변경은 허락하지 않는 것으로 표현되고 있는데 이는 형성뷰가 [정의 3.1]에서 정의하고 있는 바와 같이 기본 테이블에 대한 뷰 정의에 의해서 유도된 테이블이기 때문에 뷰를 유도하는 기본 테이블 및 뷰 정의에 대한 데이터 일관성(data consistency) 유지를 위해서 반드시 기본 테이블의 변경에 대한 전파는 허용하지만 반대로 뷰에 대한 자체적 변경을 허락하여서는 안되기 때문이다.



(그림 3.1) 형성 뷰 관리 메커니즘

(그림 3.1)의 형성 뷰 관리 메커니즘은 [정의 3.1]의 뷰 정의를 토대로 점진적 연산에 기반한 뷰 관리 메커니즘을 기술하고 있다. 이 점진적 연산에 기반한 뷰 형성 연산 모델은 다음의 3.2절에서 정립한다.

3.2 형성뷰의 점진적 관리를 위한 연산 모델

[정의 3.1]에서 형성 뷰에 대한 점진적 연산을 다음과 같이 정의하였다.

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

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 삽입 차분, 삭제차분 그리고 갱신 차분으로 구분할 수 있으며 이들은 각각 다음과 같은 의미를 갖는다.

삽입차분 $\Delta^{+}_{R_i} = \{{}_k r \mid \exists k\ {}_k r \in R_i \wedge {}_k r \notin R_{i-1}\}$

삭제차분 $\Delta^{-}_{R_i} = \{{}_k r \mid \exists k\ {}_k r \notin R_i \wedge {}_k r \in R_{i-1}\}$

갱신차분 $\Delta^{\mu}_{R_i} = \{{}_k r \mid \forall k\ {}_k r \in R_i \wedge {}_k r \in R_{i-1} \wedge \exists j\ {}_k r_i(a_j) \neq {}_k r_{i-1}(a_j)\}$

여기서 각각의 차분을 정의하는 기호 속에 내포되어 있는 아래첨자 R_i는 시점 i에서의 테이블에 대한 차분임을 나타내기 위한 기호이고 윗 첨자 $+$, $-$, u는 각각 삽입(insert), 삭제(delete), 갱신(update)연산에 대응해서 발생한 차분임을 나타내기 위한 기호이다. 한편 테이블 R에 첨부된 아래첨자 i는 시점을 나타내는 시점 인덱스로서 $i-1$시점에 데이터베이스 상태를 변경시키는 연산이 발생해서 이 연산의 수행 결과로서 i시점의 데이터베이스 상태에 도달하게 된다. 즉 시점 인덱스 i가 의미하는 바는 안정적 상태에서 다른 안정적 상태에 이르는 두 시점을 의미하는 것이다. 그리고 테이블 R의 원소인 튜플 r의 앞에 붙은 아래첨자는 테이블 내 특정 튜플을 식별하기 위한 색인을 의미한다.

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.

차분 $\Delta_{R_i} = \Delta^{+}_{R_i} \cup \Delta^{-}_{R_i}$

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

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

여기서 θ 연산자의 단위 연산자들에 대한 연산 대상은 차분을 정의하는 각각의 차분들 즉, $\Delta^+_{R_i}$, $\Delta^-_{R_i}$, $\Delta^+_{R_i} \cup \Delta^-_{R_i}$을 대상으로 하며 특히 마지막 연산의 명세 *insert*(*delete* (*R*)) 는 갱신 연산을 규정하는 의미로서 일반적으로 갱신 연산은 삭제 후 재 삽입으로 규정하는 연산 의미를 반영한 것이다.

한편 차분 의미에서 삽입 차분과 삭제 차분이 삽입 및 삭제 연산을 명세하는 단위 연산에서는 절대로 동시에 발생할 수 없음을 고찰하였으며 아울러 갱신 차분은 삭제 후 삽입 연산으로 재정의된 연산 의미를 반영하여 연산 발생 시점의 삽입 차분과 삭제 차분의 합집합으로 규정하였다. 따라서 이들 차분의미를 점진적 연산에 반영할 경우 다음과 같은 정형 의미를 갖는다.

$$V_i = (V_{i-1} - \Delta^-_{R_i}) \cup \Delta^+_{R_i}$$

만일 차분을 발생시키는 연산이 삽입 연산이라면 삭제차분 $\Delta^-_{R_i}$는 공집합이므로 이 식은 성립한다. 그리고 삭제 연산 역시 삽입 차분 $\Delta^+_{R_i}$가 공집합이므로 이 식은 성립한다. 마지막으로 갱신 연산 역시 삭제 후 삽입 연산의 특성에 따라 이 식이 성립한다. 이렇게 차분을 이용하는 점진적 연산의 효율성은 차분을 고려하지 않으면서 전체 데이터베이스를 대상으로 하는 재계산 연산 보다 성능이 우수하다. 이것은 조인 연산에 대한 연산 비용의 비교를 통해 간단히 증명할 수 있다. 조인 연산은 최소 두 개 이상의 테이블 사이의 연관성을 토대로 수행되는 연산이며 조인 속성의 대응 관계를 조사하기 위해 두 테이블의 카디널리티의 곱에 대응하는 연산 비용을 갖는다. 이를 대수식으로 표현하면 다음과 같다.

$$Cardinality(R \triangleright\triangleleft_{r\theta x} X) \le Cardinality(R) \times Cardinality(X)$$

여기서 θ의 의미는 조인을 규정하는 두 속성 사이의 대응관계를 표현하는 연산자를 대표하는 기호이다. 이를 차분을 이용하여 표현하면 다음과 같다.

$$R_i \triangleright\triangleleft_{r\theta x} X_i =$$
$$\{(R_{i-1} - \Delta^-_{R_i})) \cup \Delta^+_{R_i}\} \triangleright\triangleleft_{r\theta x} \{(X_{i-1} - \Delta^-_{X_i}) \cup \Delta^+_{X_i}\}$$

여기서 수정 연산은 삭제 후 삽입이 발생하는 것이므로 논의 전개의 편의를 위하여 갱신 연산이 아닌 삽입 또는 삭제만 발생한다고 가정하고 전개하기로 하겠다. 수정 연산의 경우 이 두 연산을 합산하면 된다. 우선 삽입 연산에 대해서 먼저 전개하기로 하겠다. 삽입 연산에 대해 위 식을 전개하면 다음과 같다.

$$\{R_{i-1} \cup \Delta^+_{R_i}\} \triangleright\triangleleft_{r\theta x} \{X_{i-1} \cup \Delta^+_{X_i}\} =$$
$$\{R_{i-1} \triangleright\triangleleft_{r\theta x} X_{i-1}\} \cup \{R_{i-1} \triangleright\triangleleft_{r\theta x} \Delta^+_{X_i}\}$$
$$\cup \{X_{i-1} \triangleright\triangleleft_{r\theta x} \Delta^+_{R_i}\} \cup \{\Delta^+_{R_i} \triangleright\triangleleft_{r\theta x} \Delta^+_{X_i}\}$$

조인 연산자는 분배법칙을 허용하므로 위의 결과는 타당하다. 여기서 릴레이션 R 또는 X 둘 중에 하나의 릴레이션에 대해서만 삽입 연산이 발생할 경우와 둘 모두에 대해서 삽입 연산이 발생할 경우로 나누어 해석할 필요가 있다. 이를 위해 R 릴레이션에 대한 카디널리티를 n이라 하고 X 릴레이션에 대한 카디널리티를 m이라 하고 $\Delta^+_{R_i}$에 대한 카디널리티를 $n\Delta$라 하고 $\Delta^+_{X_i}$에 대한 카디널리티를 $m\Delta$라 하면 이 연산에 의해 발생하는 조인 비용은 다음과 같이 표현할 수 있다.

$$(n_{i-1} \times m_{i-1}) + (n_{i-1} \times m\Delta) + (m_{i-1} \times n\Delta) + (n\Delta \times m\Delta)$$

[13]에서 증명하였듯이 $i-1$ 시점의 계산결과는 이미 완료된 상태로 누적되어 있으므로 계산 비용은 0이다. 아울러 나머지 부분에 대한 계산 비용 역시 차분의 규모가 베이스릴레이션이 갖는 카디널리티에 비해 매우 작으므로 연산 비용 역시 매우 작다. 따라서 점진적 연산에 의한 연산 비용이 재계산 연산에 비해 작게 드는 것은 자명하다.

4. 트리거와 결합한 점진적 뷰 형성 모델

트리거는 2장에서 언급한 바와 같이 데이터베이스 상태 변경 연산에 대응해서 자동으로 조건부 조치를 수행할 수 있는 데이터베이스 프리미티브이다. 이와 같은 트리거를 점진적 뷰 형성 연산과 결합할 경우 데이터베이스 변경 연산이 발생할 때 마다 트리거에 의해 자동으로 점진적 뷰 형성 연산을 호출하도록 함으로써 사용자의 개입 없이 형성 뷰를 관리할 수 있다.

4.1 트리거

이 절에서는 점진적 형성뷰 연산을 데이터베이스 수정 연산에 대응해서 자동으로 기동(invoke) 시키는 역할을 담당하는 트리거에 대해 간단히 고찰한다. 왜냐하면 트리거의 특성을 명확히 해야 나중에 트리거와 결합한 점진적 형성 뷰 연산을 명세하는데 보다 정밀한 설명이 가능하기 때문이다. 이를 위해 먼저 SQL3[2]에서 규정하고 있는 트리거

구문을 다음의 (그림 4.1)을 통해 제시한다.

```
CREATE TRIGGER <Trigger-name>
   [ <trigger action time> ] <trigger events> ON
    <table name>
   [ REFERENCING <temporal references> ]
   [ FOR EACH [ ROW | STATEMENT] ]
   [WHEN <trigger condition predicate>]<trigger action>
```

(그림 4.1) SQL3 기반의 트리거 언어 구문

여기서 <trigger event>는 트리거를 기동시키는 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 BEFORE로 명세되면 <trigger event> 연산이 수행되기 이전에 먼저 트리거 조치의 처리를 수행하라는 의미이고 AFTER는 그 반대의 의미이다[1, 2, 8].

전이값(transition value)으로 정의되는 REFERENCING 절은 <trigger event>연산에 의해 데이터베이스에 반영되어야 할 값에 대한 참조를 명세하는 것으로서 삭제된 값은 OLD 키워드와 함께 명세되고 삽입되는 값은 NEW 키워드와 함께 명세한다[2, 8]. 이 REFERENCING 절을 이용할 경우 점진적 연산의 차분을 데이터베이스에 대한 검색 없이 획득할 수 있다. 그러나 REFERENCING 절은 다음 줄에 명세된 트리거 수행 단위 중 ROW로 명세된 트리거만이 REFERENCING 절을 명세할 수 있으며 STATEMENT로 명세되는 트리거에서는 REFERENCING 절을 명세할 수 없다[2, 8].

한편 트리거 조치절(trigger action)인 <trigger action>은 궁극적인 트리거 수행의 목표가 되는 연산들의 집합으로 정의할수 있으며 일반적으로 시스템에서 허용하는 저장 프로시져(stored procedure)로 확장된다. 다음은 <trigger action>을 확장한 구문이다.

```
<trigger action> : = BEGIN <trigger body> END;
<trigger body> : = {<SQL DML> | <stored procedure
                    statement>}+
```

여기서 트리거 조치절은 BEGIN과 END에 의해 묶여지는 블록 내에 명세되는 SQL 문이거나 또는 저장 프로시져 문이 최소한 한번 이상 나타나는 형태로 구현되어야 하며, 복잡한 연산의 경우 저장 프로시져와 SQL이 결합되어 구현됨으로써 복잡한 연산을 명세하는 것이 가능해 진다.

4.2 트리거와 점진적 뷰 형성 연산 모델의 결합

4.1절에서 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. 먼저 점진적 뷰 형성 연산 모델과 트리거를 결합하는데 있어 4.1절에서 고찰한 트리거 옵션들을 정리하면 다음의 <표 4.1>과 같다.

<표 4.1> 점진적 뷰 형성 연산을 위한 트리거 옵션

트리거 옵션	트리거 옵션 값		점진적 뷰 형성 연산을 위한 선택
<trigger action time>	BEFORE | AFTER		AFTER
REFERENCIN 절	insert 대응		NEW AS <new alias>
	delete 대응		OLD AS <old alias>
	update	delete	OLD AD <old alias>
		insert	NEW AS <new alias>
트리거 수행단위	ROW | STATEMENT		FOR EACH ROW

<표 4.1>에서 규정한 옵션 값들을 이용하여 점진적 뷰 형성 연산과 결합된 트리거 프레임을 정의하면 다음의 (그림 4.2)와 같다.

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

(그림 4.2) 점진적 뷰 형성 모델을 위한 트리거 프레임

(그림 4.2)에서 점진적 뷰 형성 연산 모델을 위한 트리거 프레임을 정의하였다. (그림 4.2)는 삽입 연산에 대응한 트리거 프레임을 정의 한 것으로서 삭제 연산 및 갱신 연산 모두에 대해 (그림 4.2)와 유사한 프레임을 정의 할 수 있다. 점진적 뷰 형성 연산 모델을 트리거와 결합하기 위해서는 트리거를 통해 구현하고자 하는 뷰의 규모, 뷰 형성 연산의 최적화를 위한 질의문 분석 등을 선행적으로 수행해야 한다. 그런다음 결정된 뷰의 구조에 따라 뷰 테이블을 생성하고 이 뷰 테이블을 관리하기 위한 점진적 뷰 형성 연산 트리거를 구현해야 한다. 트리거와 결합된 점진적 뷰 형성 연산은 뷰 명세에 따라 다양한 형태를 가질 수 있으

므로 구체적 구현예는 다음의 4.3절에서 예시한다.

4.3 점진적 뷰 형성 연산의 구현

이 절에서는 4.2절에서 구성한 점진적 뷰 형성 모델을 위한 <trigger body> 부분을 구성할 점진적 연산의 구현 예를 제시한다. (그림 4.2)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.

이 절에서 제시하는 뷰 형성 연산은 5장 실험 환경에서 제시하는 갱신 뷰 형성 질의(1)에 해당하는 뷰 형성 연산 질의를 구현한다. 갱신 뷰 형성 질의(1)의 명세는 다음의 (그림 4.3)과 같다.

```
1   select sum(aa.attrv1), sum(aa.attrv2), sum(aa.attrv3),
        sum(aa.attrv4), sum(aa.attrv5)
2   from (select a.key1, a.attrv1, a.attrv2, a.attrv3, a.attrv4,
        a.attrv5,
3    from tab1 a, tab2 b
4    where a.key2 = b.key2 ) aa
5   where aa.key1 = 사용자지정코드
6   group by aa.key1
7   union all
8   select sum(aa.attrv1), sum(aa.attrv2), sum(aa.attrv3),
    sum(aa.attrv4), sum(aa.atrv5)
9   from ( select a.key1, c.key3, a.attrv1, a.attrv2, a.attrv3,
    a.attrv4, a.attrv5,
10   from tab1 a, tab2 b, (select key3, key4 from tab3 where
    key0 = '사용자지정상수') c
11   where a.key2 = b.key2 and b.key4 = c.key4 ) aa
12  where aa.key1 = 사용자지정코드
13  group by aa.key1, aa.key3
```

(그림 4.3) 갱신 뷰 형성 질의(1)의 명세

(그림 4.3)에서 명세하는 갱신 뷰 형성질의(1)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 4.3)의 질의는 수행 결과가 달라지게 된다. 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 및 key3 속성을 포함해야 한다. 반면 질의 내부에서 조인 조건으로 참여하는 key2 및 key4는 질의 내부에서 중간 결과들을 생성하는데만 영향을 미칠 뿐 최종 결과를 구성하는데는 관여하지 않으므로 뷰 테이블에는 포함할 필요가 없다. 따라서 (그림 4.3)의 질의에 대응하는 형성 뷰 테이블의 구조는 다음의 (그림 4.4)와 같은 구문에 의해 생성된다.

```
CREATE TBALE VIEW1(
        key1        tab1.key1%TYPE          NOT NULL,
        key3        tab3.key3%TYPE          NOT NULL,
        attrv1      NUMBER      NOT NULL,
        attrv2      NUMBER      NOT NULL,
        attrv3      NUMBER      NOT NULL,
        attrv4      NUMBER      NOT NULL,
        attrv5      NUMBER      NOT NULL
);
```

(그림 4.4) 갱신 뷰 형성질의(1)을 위한 뷰 형성 테이블 생성

(그림 4.4)의 질의에 의해 생성된 테이블에 대하여 (그림 4.3)의 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 4.3)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 튜플의 삽입 여부를 판단하는데 이용되는 전역 변수들로 (그림 4.3)의 줄 번호 4, 6, 11, 13번째 줄에 명세된 연산 의미를 구현하기 위해 필요하며 모두 리스트 구조를 갖는다.

```
tab1_list    tab1_list_type ; // key1, key2를 갖는 레코드의 리스트
tab2_list    tab2_list_type ; // key2, key4를 갖는 레코드의 리스트
tab3_list    tab3_list_type ; // key0, key3, key4를 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. tab2 리스트 및 tab3 리스트 역시 마찬가지이다. 이와 같이 하는 이유는 이들 레코드의 리스트가 해당 속성 값을 메인 메모리에 유지함으로써 데이터베이스에 대한 접근 없이 (그림 4.3)의 질의에서 명세하는 조인 연산을 트리거 바디 부분에서 구현할 수 있도록 하기 때문이다. 아울러 이들 전역 변수들은 시스템의 시작과 동시에 데이터베이스에 대한 검색 연산을 통해 생성 되도록 "시스템 시작" 사건에 의해 기동되는 트리거를 이용하여 초기값을 설정하며 해당 테이블에 대한 삽입이 발생할 때마다 이들 전역변수를 관리하는 트리거를 통해 각 리스트의 값을 관리한다. 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 4.4)에서 명세한 VIEW1 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 VIEW1%ROWTYPE
INDEX BY BINARY_INTEGER;

material_1 material_type ;   // (그림 4.3)의 첫 번째 질의를 위
                                한 뷰 형성 결과 유지 전역변수
material_2 material_type ;   // (그림 4.3)의 두 번째 질의를 위
                                한 뷰 형성 결과 유지 전역변수
```

이와 같은 조건 하에서 점진적 연산을 위한 완전한 트리거 정의는 다음의 (그림 4.5)와 같다.

```
TRIGGER InsertTab1Trigger
AFTER INSERT ON TAB1
REFERENCING NEW AS newTab1
FOR EACH ROW
DECLARE
  key1_ok              BOOLEAN ;
  material_cnt         NUMBER ;
  key3_value           tab3.key3%TYPE := NULL ;
  key4_value           tab2.key4%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 4.5) 갱신 뷰 형성 질의(2)을 위한 점진적 트리거

(그림 4.5)의 트리거 명세에서 호출함수 key1_assign, compare_key2 및 select_key3는 각각 저장 프로시져로 구현한 함수들로서 (그림 4.3)의 질의 명세에서 줄 5와 12의 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 반환한다. compare_key2 및 select_key3 역시 비슷한 기능을 갖지만 반환 값이 BOOLEAN 값이 아니라 key4와 key3 형의 데이터를 반환한다는 차이를 갖는다.

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. 아울러 각 유형의 질의들이 갖는 조인 술어의 복잡성을 평가하기 위한 요소들로는 다음의 <표 5.2>와 같다.

〈표 5.2〉 조인 평가 요소 식별 표

평가 요소	본질의 명세	인라인 뷰
조인참여 테이블 수	①	①
조인항 수	②	②
외부조인 포함	③	③
조인참여 상수	④	④
집계함수 포함	⑤	⑤

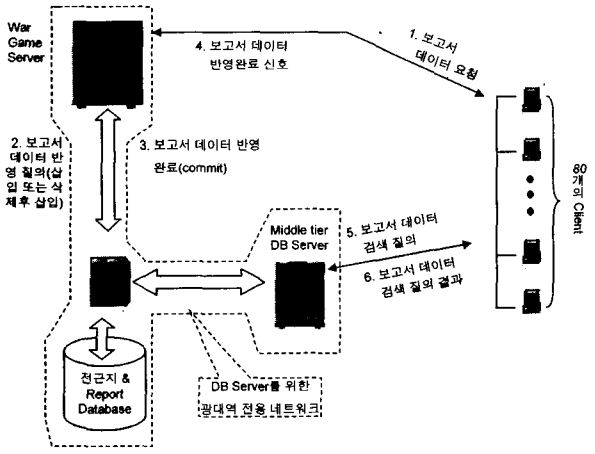
이 <표 5.2>에 기술된 평가 요소들을 이용한 질의 실험용 질의들의 조인 복잡도 표를 다음의 <표 5.3>에 기술하였다. 조인 참여 테이블 수는 조인 술어에서 조인항을 구성하는 각 술어에서 참조하는 테이블들의 총 수를 의미하며 조인항 수는 AND, OR 등에 의해 연결되는 조인항의 수를 의미한다. 조인 복잡도는 외부 조인(external join) 포함 여

부를 평가하며 아울러 조인 술어 내부에 상수를 포함하는지 여부도 평가한다. 이들 각각은 본질의와 본질의 내에 명세된 인라인 뷰에 대해 평가하도록 구성하였다. 인라인 뷰는 형성 뷰 연산의 대상이 되며 순수 질의에서는 인라인 뷰로 명세된다.

〈표 5.3〉 실험에 참여한 질의들의 조인 복잡도 평가

질의문 유형	본 질 의					인라인 뷰				
	①	②	③	④	⑤	①	②	③	④	⑤
비 갱신 형성 뷰 이용질의(1)	2	2	○	○	×	2	2	×	×	×
비 갱신 형성 뷰 이용질의(2)	3	3	○	○	×	×	×	×	×	×
갱신 형성 뷰 이용 질의(1)	1	1	×	○	5	3	3	×	○	×
갱신 형성 뷰 이용 질의(2)	1	1	×	○	5	3	3	○	×	×

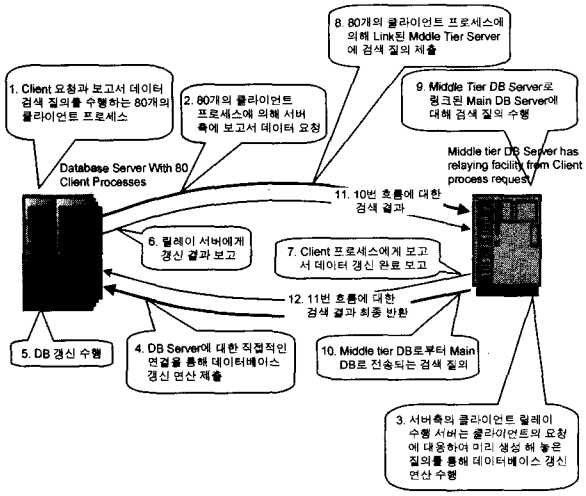
한편 실험에 참여하는 테이블의 수는 모두 7개이며 이 중 조인에 참여하는 테이블은 5개이다. 그리고 각각의 테이블에는 1500건 이상의 튜플들을 갖고 있으며 각각의 질의 유형마다 20개의 갱신 연산과 검색연산을 쌍으로 구성하였다. 다음의 (그림 5.1)은 이와 같은 실험환경을 반영한 시스템 구성도이다.



(그림 5.1) 이상적 실험을 위한 시스템 구성도

(그림 5.1)은 최적의 실험환경 구성을 나타낸 그림이다. (그림 5.1)에서 볼 수 있는 바와 같이 클라이언트가 80대 연결되어 있는데 이는 위 네 가지 질의 유형 각각에 대하여 20대식 할당하기 때문에 필요한 수치이다. 실험의 진행은 80대의 클라이언트 중 4대씩을 하나의 그룹으로 묶고 각각의 그룹은 다시 4개의 질의 유형에 해당하는 질의를 각각의 클라이언트에 배정하는 형태로 구성한다. 이렇게 준비된 상태에서 최초 1개의 그룹을 동시에 기동시킨 후 결과를 측정하고 다시 2개의 그룹을 동시에 기동시켜 결과를 측정하는 순으로 진행되며 최종적으로는 20개의 그룹을 동시에 기동시킨 후 그 결과를 측정하는 방식으로 진행한다.

그러나 현실적으로 80대의 클라이언트를 동시에 기동시키면서 집중되는 트랜잭션을 생성하는 것이 매우 힘들 뿐만 아니라 이와 같은 실험 환경을 구축할 수 있는 환경을 구성하기가 매우 어려운 상황이어서 위 (그림 5.1)의 실험환경을 모의(simulation)할 수 있는 환경을 구성하여 실험을 수행하였다. 다음의 (그림 5.2)는 (그림 5.1)을 두 대의 서버급 컴퓨터를 이용하여 모의한 후 모의 실험 환경에서 실험 수행에 따른 시나리오를 나타낸 그림이다.



(그림 5.2) 모의 환경의 구성 및 모의 환경에서의 질의 수행 시나리오

(그림 5.2)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. 이는 (그림 5.1)의 실험 환경에서 그룹화된 4개의 클라이언트들을 모델링 하기 위하여 도입한 방법이다.

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 릴레이서버에 반환한다.

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

단계 5 : 수정 완료 신호를 수신한 클라이언트는 Middle-Tier 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 비활성화 시킨 상태에서 형성뷰를 이용하지 않는 질의를 이용하여 성능 평가를 수행하는 과정을 통해 성능 평가를 수행한다.

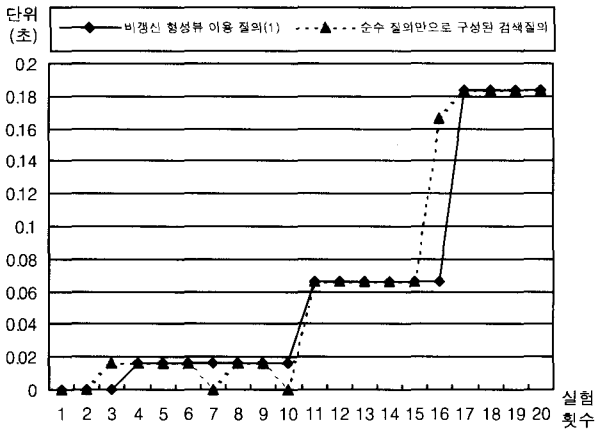
6. 성능 평가

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.

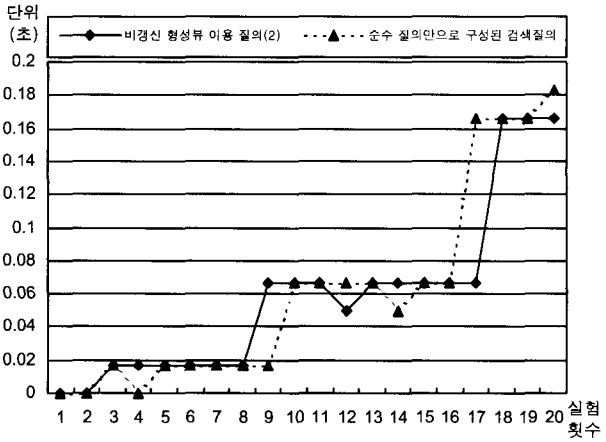
〈표 6.1〉 실험에 이용된 시스템 환경

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)과 (그림 6.2)는 이 두 실험의 평가 결과를 나타낸다.



(그림 6.1) 비 갱신 형성뷰 이용 질의 (1)번에 대한 실험 결과



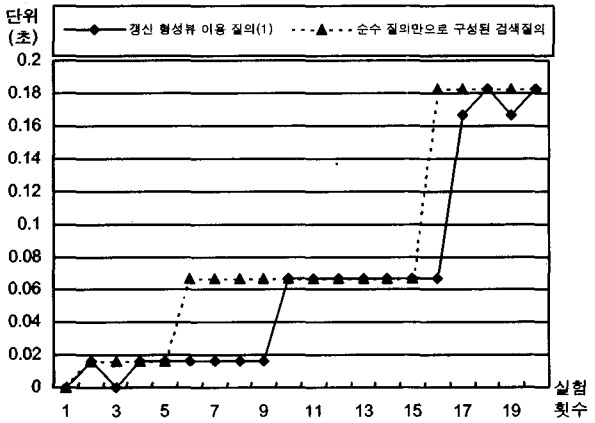
(그림 6-2) 비 갱신 형성뷰 이용 질의 (2)번에 대한 실험 결과

(그림 6.1)의 실험에 참여한 형성 뷰는 갱신이 발생하지 않는 형성 뷰를 이용한다. 이 비 갱신 형성 뷰는 <표 5.3>에서 명시한 바와 같이 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 질의(1)은 형성 뷰를 이용한 질의 수행을 명세한 질의이며 순수한 질의만으로 구성된 질의는 형성 뷰를 이용하지 않고 형성 뷰에 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 따

른 동시 트랜잭션 수가 증가하면서 형성뷰를 이용하는 질의가 약간 좋은 성능을 나타내고 있음을 확인할 수 있다.

한편 (그림 6.2)는 비 갱신 형성 뷰를 이용하는 두 번째 질의의 실험 결과를 나타낸다. 비 갱신 형성 뷰를 이용질의(2)의 성능을 나타내는 (그림 6.2)는 비 갱신 형성 뷰를 나타내는 (그림 6.1)과 유사한 성능 특성을 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 조인항에 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. 아울러 이와 같은 상황은 <표 5.3>의 세 번째와 네 번째 줄에 조인 복잡도를 제시하였다. 먼저 갱신 형성 뷰 이용 질의 (1)번 및 (2)번에 대한 실험 결과를 다음의 (그림 6.3)과 (그림 6.4)에 도시하였다.

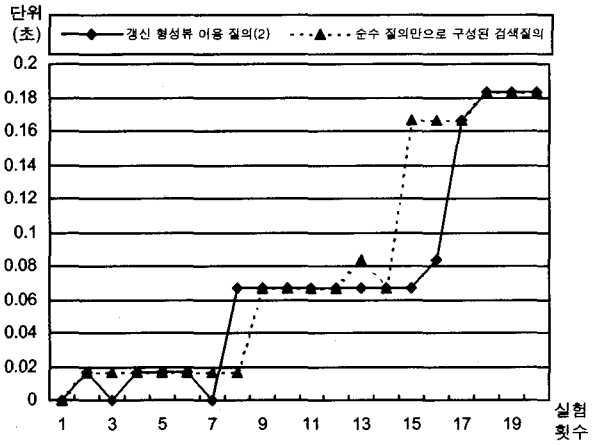


(그림 6.3) 갱신 형성 뷰 이용 질의 (1)번에 대한 실험 결과

(그림 6.3)에서 갱신 형성 뷰 이용 질의와 순수한 질의만으로 구성된 질의 사이의 성능차이가 크게 나타나는 이유는 (그림 6.3)을 명세하는 형성 뷰 이용 질의가 <표 5.3>에서 제시한 바와 같이 다섯 개의 집계 함수를 포함하고 있기 때문인 것으로 판단할 수 있다.

(그림 6.4)에서 갱신 형성 뷰 이용질의(2)의 수행 결과가 (그림 6.3)의 갱신 형성 뷰 이용질의의 수행 결과와 비교하여 갱신 형성 뷰 이용 질의와 순수한 질의 사이의 성능 차이가 다소 감소하는 추세를 보이는데 이는 (그림 6.4)를 명세하는 질의가 (그림 6.3)을 명세하는 질의와 유사한 특성을 갖지만 (그림 6.3)을 명세하는 질의에는 포함되어 있지 않은 외부 조인을 포함하고 있을 뿐만 아니라 인라인 뷰에서 상수값을 이용한 조인술어 제한이 포함되기 때문에 (그림 6.3)의 명세가 연산 부하를 덜 받기 때문인 것으로 판단할 수 있다.

(그림 6.3) 및 (그림 6.4)는 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.



(그림 6.4) 갱신 형성 뷰 이용 질의 (2)번에 대한 실험 결과

한편 (그림 6.1)에서 (그림 6.4)에 이르는 실험은 4가지 질의 유형에 대하여 각각 20개의 질의로 구성된 80개의 프로세스를 동시에 생성한 후 (그림 5.1)의 시나리오에 따라 수행되도록 구성하였다. 이와 같은 구성은 워게임 환경 과 같이 대규모의 다중 클라이언트가 동시에 접속하는 상황을 모의하기 위하여 구성 한 것이다. 따라서 80개의 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 6.4)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

우게임 환경과 같이 대규모 트랜잭션 환경에서 실시간 보고서 생성이 가능하도록 하기 위해서는 기존의 질의만으로는 해결할 수 없다. 이와 같은 문제를 해결하기 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 좀더 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] 류근호, "시간지원 데이터베이스에서 뷰 형성 유지를 위한 실행트리", 한국정보과학회 논문지, 제20권 제8호, 1993.

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

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

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

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

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

[9] C. Forgy, "Rete, a fast algorithm for the main memory patterns many objects pattern mathching problem," J. Artificial Intelligence, p.19, 1982.

[10] D. P. Miranker, "TREAT : A better match algorithm for AI production systems," In Proceedins of AAAI 87 Conference on Artificial Inteligence, 1987.

[11] 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.

[12] 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.

[13] 신예호, 류근호, "실시간 연관규칙 탐사를 위한 능동적 후보 항목 관리모델", 정보처리학회논문지D, 제9-D권 제2호, 2002.

[14] 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.

[15] 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.

[16] 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.

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

김 진 수

e-mail : kjs9990@yahoo.co.kr

1981년 계명대학교 수학과(학사)

1987년 국방대학교 전산학과(이학석사)

1998년 충북대학교 전산학과 박사과정 이수

2002년~현재 육군교육사 체계분석실 근무

관심분야 : 실시간 객체, 분산컴퓨팅, 시공간 데이터베이스

신 예 호

e-mail : snowman@kdu.ac.kr

1996년 군산대학교 컴퓨터과학과(학사)

1998년 충북대학교 대학원 전자계산학과 (석사)

1998년. 충북대학교 대학원 전자계산학과 입학(박사과정)

2002년 충북대학교 대학원 전자계산학과(이학박사)

2002년~현재 극동대학교 정보통신학부

관심분야 : 능동 데이터베이스, 시간 데이터베이스, 공간 데이터베이스, 데이터 마이닝

류 근 호

e-mail : khryu@dblab.chungbuk.ac.kr

1976년 숭실대학교 전산학과(이학사)

1980년 연세대학교 공학대학원 전산전공 (공학석사)

1988년 연세대학교 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