데이타 웨어하우스에서 효과적인 점진적 뷰 관리

(An Efficient Incremental View Maintenance in Data Warehouses)

이 기 용 † 김 명 호 ††
(Ki Yong Lee) (Myoung Ho Kim)

요 약 데이타 웨어하우스는 외부 데이타를 통합 요약하여 저장하는 시스템으로, 의사 결정에 필요한 정보를 효과적으로 제공할 수 있다. 대부분의 데이타 웨어하우스에서는 데이타를 정리한 결과를 형성(materialized) 뷰의 형태로 저장한다. 이 때 뷰가 정의된 데이타 소스가 변화하면 뷰는 이를 반영하기 위해 갱신되어야 한다. 뷰에 대한 갱신 작업은 상당한 부하를 야기시킬 수 있으므로, 이러한 갱신 작업을 효율적으로 수행하는 것은 매우 중요한 문제가 된다. 이미 뷰의 효율적인 갱신 방법에 대해서는 많은 연구가 이루어져 왔다. 그러나 뷰가 여러 개의 데이타 소스에 의해 정의되고 이들 중 둘 이상의 데이타 소스가 변화된 경우, 이를 뷰에 반영하기 위해서 기존의 방법들은 데이타 소스에 대해 많은 수의 접근이 이루어져야 한다. 본 논문에서는 여러 개의 데이타 소스에 변화가 일어난 경우, 기존 연구에 비해 데이타 소스에 대한 접근을 줄일 수 있는 효율적인 뷰 갱신 방법을 제안한다. 그리고 TPC-D 데이타를 사용한 실험을 통하여 본 논문에서 제안한 방법이 기존의 방법들보다 성능이 우수하다는 것을 보인다.

Abstract A data warehouse is an integrated and summarized collection of data that can efficiently support decision making process. The summarized data at the data warehouse is often stored in materialized views. These materialized views need to be updated when source data change. Since the propagation of updates to the views may impose a significant overhead, it is very important to update the warehouse views efficiently. Though various strategies have been proposed to maintain views in the past, they typically require too much accesses to the data sources when the changes of multiple data sources have to be reflected in the view. In this paper we propose an efficient view update strategy that uses relatively small number of accesses to the data sources. We also show the performance advantage of our method over other existing methods through experiments using TPC-D data and queries.

1. 서 론

데이타 웨어하우스(data warehouse)는 의사 결정 시스템을 지원하기 위한 시스템으로, 통합 및 정제의 과정을 통해 외부의 데이타를 저장한다[1]. 대부분의 데이타 웨어하우스 시스템은 자주 사용되는 종류의 데이타에 대해 이를 정리하고 요약한 내용을 형성 뷰의 형태로 저장한다[2]. OLAP(On-Line Analytical Processing) 질의들은 이렇게 형성 뷰로 미리 계산되어 저장된 데이타를 사용함으로써 좀 더 빠르게 처리될 수 있다. 이렇게 데이타 소스들에 의해 정의된 형성 뷰들은 데이타 소스가 변화함에 따라 이를 반영하기 위해 갱신되어야 하는데 보통 이러한 갱신 작업을 뷰 관리라고 한다[3]. 일반적으로 데이타 웨어하우스에 저장된 뷰가 갱신되는 동안에는 데이타 웨어하우스에 대한 접근이 제한되거나 접근이 허용되더라도 OLAP 질의에 대한 처리들이 뷰의 갱신 작업에 의해 방해를 받게 된다. 따라서 뷰를 효

· 본 연구는 한국과학재단 특정기초연구(과제번호 1999-1-303-007-3) 지원으로 수행되었음.
† 비 회 원 : 한국과학기술원 전산학과
kylee@dbserver.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 1999년 8월 16일
심사완료 : 2000년 4월 20일

율적이고 빠르게 갱신하는 일은 매우 중요한 일이다. 어떤 뷰가 기본 릴레이션(base relation)들에 의해 정의되어 있을 때, 기본 릴레이션의 변화를 뷰에 반영하는 방법은 크게 다음의 두 가지로 나뉜다.

- 재계산(recomputation): 각각의 기본 릴레이션들을 먼저 갱신하고, 갱신된 기본 릴레이션들로부터 뷰를 새로 계산한다. 기본 릴레이션의 변화가 적을 경우에 이렇게 뷰 전체를 다시 계산하는 일은 매우 비효율적이다.
- 점진적 관리(incremental maintenance): 기본 릴레이션들과 이들 중 변경된 튜플들만으로 된 차이 릴레이션(delta relation)들로부터 변경될 뷰의 내용만을 계산한 뒤, 이 계산된 내용을 뷰에 반영하는 방법이다. 기본 릴레이션의 변화가 적은 경우, 이 방법은 재계산에 비해 매우 효율적이다.

점진적 관리 방법은 뷰의 변경될 부분만을 계산하는 방법이다. 이에 따라 점진적 관리 방법은 기본 릴레이션에 변화가 생길 때마다 뷰를 전부 새로 계산하지 않고도 뷰를 효율적으로 관리할 수 있게 해준다. 본 논문에서는 이러한 점진적 관리 방법에 대해 논의한다.

1.1 점진적 뷰 관리

데이타 웨어하우스에서의 뷰의 점진적 관리 방법에 대해서는 이미 많은 연구가 이루어져 왔다[4][5][6][7]. 이미 다양한 형태의 뷰들에 대해 점진적인 관리를 가능하게 해주는 표준식들이 제안되었으며, 이러한 표준식들은 뷰를 올바르게 관리함이 증명되었다[8][9][10][11]. 그러나 주어진 뷰에 대해 적용 가능한 표준식은 여러 가지가 있을 수 있다. 이러한 적용 가능한 표준식들 중에 어떠한 식을 사용하느냐에 따라 뷰에 대한 점진적 관리 방법이 결정된다. 다음의 예는 주어진 뷰에 대해 적용 가능한 점진적 관리 방법이 여러 가지가 있음을 보이고 있다.

예 1 데이타 웨어하우스에 릴레이션 Sale(item, clerk)과 Emp(clerk, age)가 저장되어 있으며, 이로부터 형성 뷰 Sold(item, clerk, age)가 다음과 같이 정의되어 있다고 하자.

$$Sold(item, clerk, age) = Sale(item, clerk) \bowtie Emp(clerk, age)$$

그리고 Sale과 Emp의 변경을 나타내는 차이 릴레이션을 ΔSale과 ΔEmp라 하자. 이 때 Sold를 관리하는 방법에는 몇 가지가 있다.

첫 번째 방법은 제안된 표준식을 사용하여 ΔSale과 ΔEmp에 따른 Sold의 차이 릴레이션 ΔSold를 한 번에 계산한 뒤, 이를 Sold에 반영하는 방법이다.

(1) 점진적 관리를 위한 표준식에 따라 다음과 같이 ΔSold를 계산한다.

$$\Delta Sold = \Delta Sale \bowtie Emp \cup Sale \bowtie \Delta Emp \cup \Delta Sale \bowtie \Delta Emp$$

(2) 기본 릴레이션 및 뷰에 각각의 변경을 반영한다.

$$Sale' = Sale \cup \Delta Sale$$
$$Emp' = Emp \cup \Delta Emp$$
$$Sold' = Sold \cup \Delta Sold$$

두 번째 방법은 기본 릴레이션들에 대한 변경을 한 번에 하나씩 고려하는 방법이다. 즉, ΔSale과 ΔEmp를 한 번에 하나씩 고려하여 각각에 대한 ΔSold를 부분적으로 계산하고 이들을 종합하여 Sold를 갱신한다.

(1.1) ΔSale만을 고려한 Sold의 변경 $\Delta Sold_{sale}$을 계산한다.

$$\Delta Sold_{sale} = \Delta Sale \bowtie Emp$$

(1.2) ΔSale로 Sale 테이블을 갱신한다.

$$Sale' = Sale \cup \Delta Sale$$

(2.1) ΔEmp만을 고려한 Sold의 변경 $\Delta Sold_{emp}$을 계산한다.

$$\Delta Sold_{emp} = Sale' \bowtie \Delta Emp$$

(2.2) ΔEmp로 Emp 테이블을 갱신한다.

$$Emp' = Emp \cup \Delta Emp$$

(3.1) $\Delta Sold_{sale}$와 $\Delta Sold_{emp}$로 Sold 테이블을 갱신한다.

$$Sold' = Sold \cup \Delta Sold_{sale} \cup \Delta Sold_{emp}$$

첫 번째 방법을 이단계(dual stage) 방식이라 하고 두 번째 방법을 1-way 방식이라 한다[12]. 이단계 방식에서는 모든 기본 릴레이션의 변경을 한꺼번에 고려하여 뷰의 변경을 계산하며, 1-way 방식에서는 기본 릴레이션의 변경을 한번에 하나씩만 고려하여 이를 순차적으로 뷰에 반영한다. □

1.2 관련 연구

모든 기본 릴레이션의 변경을 한꺼번에 반영하는 이단계 방식은 [8][13]등에서 사용된 방식이다. 한편, 기본 릴레이션의 변경을 단계적으로 반영하는 1-way 방식은 [9]에서 처음 제안된 방식이다. 일반적으로, n개의

기본 릴레이션 R_1, R_2, ..., R_n의 조인으로 정의된 뷰 $V \equiv R_1 \bowtie R_2 \bowtie ... \bowtie R_n$에 대해 이단계 방식과 1-way 방식은 뷰의 변경을 구하기 위해 각각 다음의 (식 1)과 (식 2)를 사용한다. 단, R′는 갱신된 R을 의미한다. (즉, $R' = R \cup \Delta R$)

$$\Delta V = \Delta R_1 \bowtie R_2 \bowtie ... \bowtie R_n \cup R_1 \bowtie \Delta R_2 \bowtie ... \bowtie R_n \cup \Delta R_1 \bowtie \Delta R_2 \bowtie ... \bowtie R_n \cup ... \cup \Delta R_1 \bowtie \Delta R_2 \bowtie ... \bowtie \Delta R_n \quad (1)$$

$$\Delta V = \Delta R_1 \bowtie R_2 \bowtie ... \bowtie R_n \cup R_1' \bowtie \Delta R_2 \bowtie ... \bowtie R_n \cup ... \cup R_1' \bowtie R_2' \bowtie ... \bowtie \Delta R_n \quad (2)$$

이단계 방식의 (식 1)은 모든 기본 릴레이션의 변경을 한꺼번에 반영하기 위한 식이며, 이를 위해 $(2^n - 1)$개의 항(term)으로 이루어져 있다. 한편, 1-way 방식의 (식 2)는 기본 릴레이션의 변경을 한번에 하나씩 반영하기 위한 식이며, 이를 위해 각 기본 릴레이션의 변경에 대해 하나의 항씩, 총 n개의 항으로 이루어져 있다. 그러나 뷰의 변경을 구하는 데에는 이 두 방식 외에도 여러 가지의 방법이 있을 수 있다. 예를 들어, 기본 릴레이션의 변경을 한 번에 두 개씩 반영함으로써 $\lceil n / 2 \rceil$ 단계에 걸쳐 뷰의 변경을 구하는 방법도 있을 수 있다. 또한, 먼저 $\lceil n / 2 \rceil$ 개의 기본 릴레이션의 변경을 반영하고, 그 다음 나머지 $\lfloor n / 2 \rfloor$ 개의 기본 릴레이션의 변경을 반영함으로써 2단계에 걸쳐 뷰의 변경을 구하는 방법도 있을 수 있다. 이러한 많은 방법들 가운데 가장 효율적으로 뷰를 관리할 수 있는 방법을 찾는 문제는 [12]에서 연구되었다.

그러나, 이단계 방식과 1-way 방식을 포함하여 지금까지 제안된 점진적 관리 방법들은 모두 뷰의 정의와 같은 형태를 가지는 항들만을 사용하고 있다. 즉, 지금까지 제안된 관리 방법들은 모두 뷰의 정의를 그대로 사용하는 형태의 항들만을 사용하고 있으며, 관리 식은 이러한 항들의 합집합의 형태로 이루어진다. 다만, 각 항은 뷰의 정의에서 기본 릴레이션이 나타나는 자리에 기본 릴레이션 또는 차이 릴레이션 또는 갱신된 기본 릴레이션이 대신 나타나는 형태를 가지게 된다. 이단계 방식을 나타내는 (식 1)은 모두 뷰의 정의와 같은 형태의 항들만을 사용하고 있으며, 1-way 방식을 나타내는 (식 2)도 (식 1)과 마찬가지로 모두 뷰의 정의와 같은 형태의 항들만을 사용하고 있다. 일반적인 뷰 관리 방식에서, 뷰의 정의와 같은 형태의 항만을 사용하는 경우, n개의 기본 릴레이션 중 m개의 기본 릴레이션의 변경을 반영하기 위해서는 $(2^m - 1)$개의 항이 필요하게 된다. 예를 들어, 기존의 방식에 따라 n개의 기본 릴레이션 중 R_1과 R_2의 변경을 반영하기 위해서는 다음과 같이 3개의 항으로 이루어진 식을 계산해야 한다.

$$\Delta V = \Delta R_1 \bowtie R_2 \bowtie ... \bowtie R_n \cup R_1 \bowtie \Delta R_2 \bowtie ... \bowtie R_n \cup \Delta R_1 \bowtie \Delta R_2 \bowtie ... \bowtie R_n$$

뷰의 정의와 같은 형태의 항들만을 사용하게 되면, 식에 나타나는 모든 항에는 항상 모든 기본 릴레이션(또는 그의 차이 릴레이션)들이 나타나야 한다. 따라서, 항의 개수가 많아지면 많아질수록 각 기본 릴레이션들이 각 항에 반복적으로 나타나는 횟수가 증가하게 되며, 이에 따라 기본 릴레이션들이 각 항의 계산에 반복적으로 참여해야 하는 횟수가 증가하게 된다. 일반적으로, n개의 기본 릴레이션의 조인으로 정의된 뷰에 대해 이단계 방식을 사용하면, 각 기본 릴레이션에 대해 $(2^{n-1} - 1)$번의 접근이 반복적으로 이루어지게 된다. 한편, 1-way 방식을 사용하면, 각 기본 릴레이션에 대해 $(n - 1)$의 접근이 반복적으로 이루어지게 된다. 따라서, 뷰의 정의에 나타나는 기본 릴레이션의 수 n이 커지면 각 기본 릴레이션에 대한 접근 횟수도 증가하게 되므로, 뷰 관리 방식의 성능이 크게 저하될 수 있다. 따라서, 본 논문에서는 기본 릴레이션에 대한 이러한 반복적인 접근을 줄이면서 효과적으로 뷰를 점진적으로 관리하는 방법에 대해 논의하고자 한다. 본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 고려하는 뷰의 정의 및 뷰의 관리 작업 비용을 예측하기 위한 비용 모델을 설명한다. 3장에서는 본 논문에서 제안하는 뷰의 점진적 관리 기법에 대해 설명하며 4장에서는 제안한 기법에 대한 성능 평가 결과를 소개한다. 5장에서는 제안한 기법에 관한 기타 사항에 대해 언급하고, 마지막으로 6장에서는 결론을 맺는다.

2. 데이타 웨어하우스 모델

일반적으로 데이타 웨어하우스에는 외부의 운영계 데이타(operational data)나 또는 데이타 웨어하우스 내부에 저장된 데이타에 의해 정의된 형성 뷰들이 저장된다. 흔히 데이타 웨어하우스에는 외부의 데이타들이 가공 및 정제의 과정을 거쳐 "사실 테이블(fact table)"이나 "차원 테이블(dimension table)"로 저장된다. 이 때, 사실 테이블과 차원 테이블들로부터 "요약 테이블(summary table)"이 정의되어 저장되기도 하는데 이 경우 요약 테이블은 사실 테이블과 차원 테이블을 기본

릴레이션으로 하는 형성 뷰라고 볼 수 있다. 기본 릴레이션들의 변경은 하루나 일주일의 간격을 두고 주기적으로 데이타 웨어하우스에 넘겨져 뷰의 갱신이 일어나게 된다.

2.1 뷰의 정의 및 점진적 관리 방법

본 논문에서는 기본 릴레이션에 대해 SPJ-식(selection-projection-join expression)으로 정의된 뷰에 대해 논의하기로 한다. 즉, 본 논문에서 고려하는 뷰는 다음과 같이 정의된다.

정의 1 n개의 기본 릴레이션 R_1, R_2, ..., R_n에 대해서 뷰 V는 다음과 같이 정의된다.

$$V \equiv \Pi_A \sigma_P (R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n)$$

여기서 A는 애트리뷰트의 집합이며 P는 부울 식(Boolean expression)을 의미한다. □

본 논문에서는 문제의 단순화를 위해 SPJ-식으로 표현되는 뷰만을 고려하였다. 그러나 일반화된 프로젝션(generalized projection)[11]과 같은 연산자를 사용하면 집계 함수(aggregate function)나 Group-by와 같은 식이 포함되어 있는 뷰에 대해서도 쉽게 확장이 가능하다.

위와 같이 정의된 뷰에 대해서 기본 릴레이션들의 변경이 각각 ΔR_1, ΔR_2, ..., ΔR_n로 주어졌다고 하자. 이 때, 이단계 방식과 1-way 방식을 적용하면 뷰의 변화 ΔV는 각각 다음과 같은 식으로 구할 수 있다. (단, $R' = R \cup \Delta R$)

- 이단계 방식에 따른 ΔV의 계산

$$\begin{aligned}\Delta V = {} & \Pi_A \sigma_P (\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) \\ & \cup \Pi_A \sigma_P (R_1 \bowtie \Delta R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) \\ & \cup \Pi_A \sigma_P (\Delta R_1 \bowtie \Delta R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) \\ & \cup \Pi_A \sigma_P (R_1 \bowtie R_2 \bowtie \Delta R_3 \bowtie \ldots \bowtie R_n) \\ & \cup \ldots \\ & \cup \Pi_A \sigma_P (\Delta R_1 \bowtie \Delta R_2 \bowtie \Delta R_3 \bowtie \ldots \bowtie \Delta R_n)\end{aligned}$$

- 1-way 방식에 따른 ΔV의 계산

$$\begin{aligned}\Delta V = {} & \Pi_A \sigma_P (\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) \\ & \cup \Pi_A \sigma_P (R_1' \bowtie \Delta R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) \\ & \cup \Pi_A \sigma_P (R_1' \bowtie R_2' \bowtie \Delta R_3 \bowtie \ldots \bowtie R_n) \\ & \cup \ldots \\ & \cup \Pi_A \sigma_P (R_1' \bowtie R_2' \bowtie R_3' \bowtie \ldots \bowtie \Delta R_n)\end{aligned}$$

2.2 비용 모델

뷰를 갱신하는데 드는 비용을 예측하기 위해 본 논문에서는 선형 작업 계량법(linear work metric)[12]을 사용하기로 한다. 여기서는 주어진 식에 대해 각 항의 계산은 독립적으로 이루어진다고 가정한다. 이 모델에 의하면 뷰의 변화를 구하기 위한 식의 예상 비용(estimate cost)은 식을 이루는 각 항에 대한 예상 비용을 합한 값이 된다. 또한, 각 항에 대한 예상 비용은 항에 나타나는 모든 피연산자들의 예상 비용을 합한 값이며 피연산자가 릴레이션인 경우의 예상 비용은 릴레이션의 크기에 비례한다. 다음의 예에서 뷰의 변화를 구하기 위해 이단계 방식과 1-way 방식을 적용했을 때, 선형 작업 계량법을 사용하여 각각의 예상 비용을 구하는 방법을 설명한다.

예 2 $V \equiv R_1 \bowtie R_2$로 정의된 뷰 V에 대해 차이 릴레이션 ΔR_1과 ΔR_2가 주어졌을 때 이단계 방식과 1-way 방식을 사용하여 ΔV를 구하는 경우 각각의 예상 비용은 다음과 같다. Cost(E)는 식 E의 예상 비용을 의미한다.

1. 이단계 방식의 예상 비용

$$\begin{aligned}Cost(\Delta V) &= Cost(\Delta R_1 \bowtie R_2 \cup R_1 \bowtie \Delta R_2 \cup \Delta R_1 \bowtie \Delta R_2) \\ &= Cost(\Delta R_1 \bowtie R_2) + Cost(R_1 \bowtie \Delta R_2) + Cost(\Delta R_1 \bowtie \Delta R_2) \\ &= c \cdot (|\Delta R_1| + |R_2|) + c \cdot (|R_1| + |\Delta R_2|) + c \cdot (|\Delta R_1| + |\Delta R_2|) \\ &= c \cdot (|R_1| + |R_2| + 2 \cdot |\Delta R_1| + 2 \cdot |\Delta R_2|)\end{aligned}$$

2. 1-way 방식의 예상 비용

$$\begin{aligned}Cost(\Delta V) &= Cost(\Delta R_1 \bowtie R_2 \cup R_1' \bowtie \Delta R_2) \\ &= Cost(\Delta R_1 \bowtie R_2) + Cost(R_1' \bowtie \Delta R_2) \\ &= c \cdot (|\Delta R_1| + |R_2|) + c \cdot (|R_1'| + |\Delta R_2|) \\ &= c \cdot (|R_1'| + |R_2| + |\Delta R_1| + |\Delta R_2|)\end{aligned}$$

여기서 c는 비례상수이다. □

선형 작업 계량법은 비교적 단순하면서도 뷰의 갱신을 위한 복잡한 식에 대한 비용을 효과적으로 예측할 수 있게 해주는 모델이다. 이미 이와 유사한 모델들이 데이타웨어하우스를 위한 알고리즘에서 사용되고 있다[12][14]. 릴레이션에 대한 변경을 나타내는 차이 릴레이션은 보통 본 릴레이션에 비해 크기가 매우 작다. 따라서 차이 릴레이션이 포함되어 있는 식을 계산하는 경우, 계산 중에 발생하는 중간 결과 및 최종 결과는 기본 릴레이션들의 연산 결과에 비해 크기가 상대적으로 아주 작게 된다. 이러한 중간 결과 및 최종 결과가 모두 메모리에 저장되어 조인, 셀렉션, 프로젝션 및 집계 함수 연산 등이 메모리 내에서 처리될 수 있으면, 각 항의 계산에서 대부분을 차지하는 시간은 릴레이션들을 메모리로 읽어들이는 시간이라 할 수 있다. 따라서 뷰의 변화를

구하는 식에서 각 항에 차이 릴레이션이 적어도 하나 이상 포함되어 있는 경우, 그 식에 대한 예상 비용은 선형 작업 계량법에 의해 적절하게 모델링된다고 할 수 있다.

3. Delta 방식

이단계 방식과 1-way 방식을 포함한 기존의 점진적 뷰 관리 방법들 중, 1-way 방식의 예상 비용이 가장 적다는 사실이 선형 작업 계량법을 사용하여 증명되었으며, 이에 따라 1-way 방식이 기존의 방법들 중 가장 좋은 성능을 가지고 있음이 보여졌다[12]. 그러나 서론에서 논의한 바와 같이 1-way 방식에서도 각 기본 릴레이션들이 반복적으로 조인에 참여해야 한다는 문제가 있었다. 이를 방지하기 위해 조인의 분배 법칙을 사용하면, 각각의 기본 릴레이션들이 조인에 참여하는 횟수를 줄여 예상 비용을 크게 감소시킬 수 있다.

예 3 뷰 V가 $V \equiv R_1 \bowtie R_2 \bowtie R_3$와 같이 정의되고 차이 릴레이션들이 ΔR_1, ΔR_2, ΔR_3로 주어진 경우 1-way 방식에 관한 식은 다음과 같이 수정될 수 있다.

$$\begin{aligned}\Delta V &= \Delta R_1 \bowtie R_2 \bowtie R_3 \cup R_1' \bowtie \Delta R_2 \bowtie R_3 \cup R_1' \bowtie R_2' \bowtie \Delta R_3 \\ &= (\Delta R_1 \bowtie R_2 \cup R_1' \bowtie \Delta R_2) \bowtie R_3 \cup R_1' \bowtie R_2' \bowtie \Delta R_3\end{aligned}$$

수정된 식에서 R_3는 두 번의 조인 즉, "$\Delta R_1 \bowtie R_2 \bowtie R_3$"와 "$R_1' \bowtie \Delta R_2 \bowtie R_3$"에 참여하는 대신, 한 번의 조인 즉, "$(\Delta R_1 \bowtie R_2 \cup R_1' \bowtie \Delta R_2) \bowtie R_3$"에만 참여하면 된다. 이 경우, R_3에 대한 접근 횟수는 두 번에서 한 번으로 줄어들며 예상 비용에서 $c \cdot |R_3|$ 만큼의 이득을 보게 된다. □

3.1 Delta 함수의 정의

앞서 설명한 특성을 고려하여 예 3에서의 수정된 식을 일반화함으로써, 뷰의 변경을 구하기 위한 Delta 함수를 다음과 같이 순환적으로(recursively) 정의한다.

정의 2 정의 1에 의해 정의된 뷰 V에 대해 Delta 함수는 다음과 같이 순환적으로 정의된다.

$$\text{Delta}(V) = \begin{cases} \Pi_{A\sigma_P}(\Delta R_1) & \text{if } n = 1 \\ \Pi_{A\sigma_P}(\text{Delta}(\Pi_{A'\sigma_{P'}}(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_{n-1})) \bowtie R_n) \cup \Pi_{A\sigma_P}(R_1' \bowtie R_2' \bowtie \ldots \bowtie R_{n-1}' \bowtie \Delta R_n) & \text{if } n > 1 \end{cases}$$

여기서 P'는 논리곱 정규형(CNF)로 표현된 $P = P_1 \wedge P_2 \wedge \ldots \wedge P_m$ 에 대해 R_n의 애트리뷰트가 포함된 P_i $(1 \leq i \leq m)$들이 제거된 P를 의미한다. 이 때, $\{R_1, R_2, \ldots, R_{n-1}\}$의 애트리뷰트 중 제거된 P_i에 포함된 애트리뷰트의 집합을 PA라 하자. 그리고 릴레이션 R_n에 대해 $A(R_n)$을 R_n의 애트리뷰트의 집합이라 하면 $A' = A - A(R_n) \cup PA$이다. □

정의 2에 따라 $V \equiv R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$으로 정의된 뷰 V에 대한 Delta 함수는 다음과 같이 간단하게 표현된다.

$$\text{Delta}(V) = \begin{cases} \Delta R_1 & \text{if } n = 1 \\ \text{Delta}(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_{n-1}) \bowtie R_n \cup R_1' \bowtie R_2' \bowtie \ldots \bowtie R_{n-1}' \bowtie \Delta R_n & \text{if } n > 1 \end{cases}$$

예 4 $V \equiv R_1 \bowtie R_2 \bowtie R_3$로 정의된 뷰 V에 대해 차이 릴레이션 ΔR_1, ΔR_2, ΔR_3가 주어졌다고 하자. 이 때, Delta 함수를 사용하여 ΔV를 구하는 방법은 다음과 같다.

(1) Delta(V)은 다음과 같이 전개된다.

$$\text{Delta}(R_1 \bowtie R_2 \bowtie R_3) = \text{Delta}(R_1 \bowtie R_2) \bowtie R_3 \cup R_1' \bowtie R_2' \bowtie \Delta R_3$$
$$\text{Delta}(R_1 \bowtie R_2) = \text{Delta}(R_1) \bowtie R_2 \cup R_1' \bowtie \Delta R_2$$
$$\text{Delta}(R_1) = \Delta R_1$$

(2) $\text{Delta}(R_1)$, $\text{Delta}(R_1 \bowtie R_2)$, $\text{Delta}(R_1 \bowtie R_2 \bowtie R_3)$를 차례로 계산한다.

여기서 R_3는 $\text{Delta}(R_1)$와 $\text{Delta}(R_1 \bowtie R_2)$의 계산에는 참여하지 않고 $\text{Delta}(R_1 \bowtie R_2 \bowtie R_3)$의 계산에서 한 번의 조인에만 참여하게 됨을 알 수 있다. 따라서 R_3가 매우 큰 릴레이션인 경우, R_3에 대해서는 한 번의 접근만 이루어지면 되므로 이 경우 Delta 함수를 사용하는 것이 1-way 방식에 비해 유리하다. □

정의 2에서 순환적으로 정의된 Delta 함수는 다음과 같이 반복(iterative) 구조를 사용해서 정의될 수 있다.

정의 3 정의 2에서 순환적으로 정의된 Delta 함수는 반복 구조를 사용하여 다음과 같이 정의된다.

```
function Delta(V)
    if n = 1 then return Π_{Aσ_P}(ΔR_1);
    ΔTemp = ΔR_1;
    for i = 1 to n − 1 do
```

$\Delta Temp = \Delta Temp \bowtie R_{i+1} \cup R_1' \bowtie R_2' \bowtie \ldots \bowtie R_i' \bowtie \Delta R_{i+1}$;

end for

return $\Pi_A \sigma_P (\Delta Temp)$;

end function

□

Delta 함수를 사용하여 뷰의 변경을 구하는 방법을 Delta 방식이라고 한다. Delta 방식을 사용할 경우, 뷰의 각 기본 릴레이션에 대한 접근 횟수는 다음과 같다.

소정리 1 (Delta 방식에서 기본 릴레이션들에 대한 접근 횟수)

정의 1에 의해 정의된 뷰에 대해 Delta 방식을 사용하는 경우, 각 기본 릴레이션 R_i ($1 \leq i \leq n$)에 대한 접근 횟수 A_i는 다음과 같다.

$$A_i = \begin{cases} n - 1 & \text{if } i = 1 \\ n - i + 1 & \text{if } 1 < i \leq n \end{cases}$$

<증명>

뷰의 정의에 포함된 프로젝션 및 셀렉션 연산자는 기본 릴레이션에 대한 접근 횟수에 영향을 미치지 않으므로 계산의 단순화를 위해 뷰는 $V \equiv R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$로 정의되었다고 가정하자. Delta(V)를 구하기 위해서는 Delta(R_1), Delta($R_1 \bowtie R_2$), Delta($R_1 \bowtie R_2 \bowtie R_3$), ..., Delta($R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$)을 차례대로 계산하기만 하면 된다. 이 때, Delta(V)의 정의에 의해 R_1은 Delta($R_1 \bowtie R_2$), Delta($R_1 \bowtie R_2 \bowtie R_3$), ..., Delta($R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$)의 계산에 각각 한 번씩 참여하면 되므로 모두 (n - 1)번의 접근이 이루어진다. 마찬가지로 R_i ($1 < i \leq n$)는 Delta($R_1 \bowtie R_2 \bowtie \ldots \bowtie R_i$), Delta($R_1 \bowtie R_2 \bowtie \ldots \bowtie R_i \bowtie R_{i+1}$), ..., Delta($R_1 \bowtie R_2 \bowtie \ldots \bowtie R_i \bowtie R_{i+1} \bowtie \ldots \bowtie R_n$)의 계산에 각각 한 번씩 참여하면 되므로 모두 (n - i + 1)번의 접근이 이루어진다. □

1-way 방식에서는 각 기본 릴레이션들에 대한 접근이 모두 (n - 1)번씩 일어난다. 따라서 소정리 1에 의해, Delta 방식은 1-way 방식에 비해 모든 기본 릴레이션에 대해 같거나 더 적은 수의 접근을 요구함을 알 수 있다. 이로 인해 Delta 방식은 예상 비용에 있어서도 1-way 방식에 비해 이득을 얻게 된다.

3.2 Delta 방식의 예상 비용

1-way 방식이 기존 방식 중에서 가장 적은 비용을 가진다는 사실은 이미 증명되었으므로[12] 이 절에서는 1-way 방식의 비용과 Delta 방식의 비용만을 비교해 보도록 한다. 단, 프로젝션 및 셀렉션 연산자는 비용 계산에 영향을 미치지 않으므로 계산의 단순화를 위해서 뷰는 $V \equiv R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$로 정의되었다고 가정한다. 차이 릴레이션 ΔR이 릴레이션 R에 비해 크기가 매우 작다고 가정하면 $|R| \approx |R'|$라고 할 수 있으므로 각 방식의 예상 비용은 다음과 같다.

관찰 1 (1-way 방식의 예상 비용)

뷰 V에 대해 1-way 방식을 적용했을 때 ΔV를 구하기 위한 예상 비용은 다음과 같다.

$$Cost(\Delta V) \approx c \cdot \sum_{i=1}^{n}((n-1) \cdot |R_i| + |\Delta R_i|)$$

<증명>

$$\begin{aligned} Cost(\Delta V) &= Cost(\Delta R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) \\ &\quad + Cost(R_1' \bowtie \Delta R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n) \\ &\quad + Cost(R_1' \bowtie R_2' \bowtie \Delta R_3 \bowtie \ldots \bowtie R_n) \\ &\quad + .. \\ &\quad + Cost(R_1' \bowtie R_2' \bowtie R_3' \bowtie \ldots \bowtie \Delta R_n) \\ &= c \cdot ((n-1) \cdot |R_1'| + (n-2) \cdot |R_2'| + \ldots \\ &\quad + (n-n) \cdot |R_n'|) \\ &\quad + c \cdot (0 \cdot |R_1| + 1 \cdot |R_2| + \ldots + (n-1) \cdot |R_n|) \\ &\quad + c \cdot (|\Delta R_1| + |\Delta R_2| + \ldots + |\Delta R_n|) \\ &= c \cdot \sum_{i=1}^{n} ((n-i) \cdot |R_i'| + (i-1) \cdot |R_i| + |\Delta R_i|) \\ &\approx c \cdot \sum_{i=1}^{n} ((n-1) \cdot |R_i| + |\Delta R_i|) \end{aligned}$$

□

관찰 2 (Delta 함수의 예상 비용)

뷰 V에 대해 Delta 함수를 사용했을 때 ΔV를 구하기 위한 예상 비용은 다음과 같다.

$$Cost(\Delta V) \approx c \cdot \sum_{i=1}^{n}((n-i+1) \cdot |R_i| + |\Delta R_i|) - c \cdot |R_1|$$

<증명>

$$\begin{aligned} Cost(\Delta V) &= Cost(Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_{n-1})) \\ &\quad + Cost(|R_n|) + Cost(R_1' \bowtie R_2' \bowtie \ldots \bowtie R_{n-1}' \\ &\quad \bowtie \Delta R_n) \\ Cost(Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_{n-1})) &= Cost(Delta(R_1 \\ &\quad \bowtie R_2 \bowtie \ldots \bowtie R_{n-2})) + Cost(|R_{n-1}|) \\ &\quad + Cost(R_1' \bowtie R_2' \bowtie \ldots \bowtie R_{n-2}' \bowtie \Delta R_{n-1}) \\ Cost(Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_{n-2})) &= \\ &\quad Cost(Delta(R_1 \bowtie R_2 \bowtie \ldots \bowtie R_{n-3})) \\ &\quad + Cost(|R_{n-2}|) + Cost(R_1' \bowtie R_2' \bowtie \ldots \\ &\quad \bowtie R_{n-3}' \bowtie \Delta R_{n-2}) \end{aligned}$$

$$Cost(Delta(R_1)) = c \cdot |\Delta R_1|$$

위의 식들을 정리하면 다음과 같이 된다.

$$Cost(\Delta V) = c \cdot \sum_{i=1}^{n} ((n - i) \cdot |R_i'| + |R_i| + |\Delta R_i|) - c \cdot |R_1|$$

$$\approx c \cdot \sum_{i=1}^{n} ((n - i + 1) \cdot |R_i| + |\Delta R_i|) - c \cdot |R_1|$$

□

1-way 방식을 사용하는 경우의 예상 비용과 Delta 방식을 사용하는 경우의 예상 비용을 각각 $Cost_{1\text{-}way}(\Delta V)$, $Cost_{Delta}(\Delta V)$라 하면, 관찰 1과 관찰 2에 의해 두 예상 비용의 차는 다음과 같다.

$$Cost_{1-way}(\Delta V) - Cost_{Delta}(\Delta V) = c \cdot \sum_{i=3}^{n}(i-2) \cdot |R_i| \geq 0$$

따라서 Delta 방식을 사용하는 것은, 1-way 방식을 사용하는 것에 비해 예상 비용에서 항상 위의 차에 해당하는 이득을 얻는다는 것을 알 수 있다.

3.3 Delta 방식의 최적화

차이 릴레이션의 크기가 작은 경우, 1-way 방식의 예상 비용은 기본 릴레이션 R_1, R_2, ..., R_n의 순서에 따라 크게 영향을 받지 않는다. 그러나 관찰 2를 통해 Delta 방식의 예상 비용은 기본 릴레이션의 순서에 따라 크게 영향을 받음을 알 수 있다. 이는 순서에 따라 각각의 기본 릴레이션에 대한 접근 횟수가 달라지기 때문이다. 예를 들어 가장 앞에 나타나는 릴레이션 R_1은 (n - 1)번의 접근이 이루어지는 반면, 가장 뒤에 나타나는 릴레이션 R_n은 1번의 접근만이 이루어지게 된다. 이 절에서는 Delta 방식을 최적화하기 위한 기본 릴레이션의 순서에 대해 논의한다.

관찰 2에 따라 Delta 방식의 예상 비용은 다음과 같이 쓸 수 있다.

$$Cost(\Delta V) \approx c \cdot \sum_{i=1}^{n}((n-i+1) \cdot |R_i| + |\Delta R_i|) - c \cdot |R_1|$$

$$= c \cdot ((n-1) \cdot |R_1| + (n-1) \cdot |R_2| + (n-2) \cdot |R_3| + \ldots + 1 \cdot |R_n|) + c \cdot \sum_{i=1}^{n}|\Delta R_i|$$

여기서 두 번째 항은 기본 릴레이션의 순서에 관계없이 항상 일정한 값을 가지게 되므로 예상 비용을 최소화하기 위한 릴레이션의 순서는 첫 번째 항만을 고려하면 된다. n은 릴레이션의 개수를 나타내는 상수이므로 (n - 1), (n - 1), (n - 2), ..., 1은 모두 상수가 된다. 여기서 (n - 1) = (n - 1) > (n - 2) > ... > 1이므로 첫 번째 항을 최소화하는 릴레이션의 순서는 가장 작은 릴레이션부터 가장 큰 릴레이션까지를 크기에 따라 배열한 순서가 된다. 따라서 Delta 함수의 예상 비용을 최소화하기 위한 기본 릴레이션의 순서는 기본 릴레이션을 그의 크기에 따라 정렬한 순서임을 알 수 있다. 이 때, 가장 작은 릴레이션부터 가장 큰 릴레이션까지는 Delta 방식에 의해 각각 (n - 1), (n - 1), (n - 2), ..., 1번씩의 접근이 이루어게 된다. 이것은 매우 큰 릴레이션이 포함된 경우, 릴레이션의 순서에서 뒤에 나타날수록 이에 대한 접근 횟수가 상당히 줄어들 수 있다는 의미를 가진다.

그러나 단순히 선형 작업 계량법에 의한 예상 비용으로 결정된 릴레이션의 순서는 Delta 함수의 계산에서 중간 결과가 매우 커질 수 있다는 문제점이 있다. 예를 들어 $|R_1| < |R_2| < \ldots < |R_n|$의 관계가 있을 때, 예상 비용만으로 결정된 릴레이션의 순서에 따르면 $Delta(R_1)$, $Delta(R_1 \bowtie R_2)$, $Delta(R_1 \bowtie R_2 \bowtie R_3)$, ..., $Delta(R_1 \bowtie R_2 \bowtie R_3 \bowtie \ldots \bowtie R_n)$을 차례대로 계산해야 한다. 이 경우 R_1과 R_2 사이에 아무런 조인 관계가 없다면 $Delta(R_1 \bowtie R_2)$에서 카티션 프로덕트(cartesian product)가 발생하게 되어 중간 결과가 매우 커질 수 있다. 이와 마찬가지로 R_1과 R_2 사이의 조인 선택도(join selectivity)가 매우 높은 경우에도 중간 결과가 상당히 커질 수 있다. 이렇게 중간 결과가 커지게 되면 선형 작업 계량법의 가정에 어긋나는 한편, 계속되는 계산에 의해 성능이 크게 저하될 수 있으므로 중간 결과의 크기도 고려해야 한다. 다음은 지금까지 언급한 사항들을 고려하여 릴레이션의 순서를 결정하는 방법이다.

알고리즘 1 (Delta 방식을 위해 최적화된 릴레이션의 순서를 구하는 방법)

입력 : n개의 기본 릴레이션 R_1, R_2, ..., R_n 및 차이 릴레이션 ΔR_1, ΔR_2, ..., ΔR_n

출력 : 최적화된 기본 릴레이션들의 순서를 나타내는 리스트

(1) ResultList ← R_1, R_2, ..., R_n 중 크기가 가장 작은 릴레이션

(2) Remaining ← {R_1, R_2, ..., R_n} - {ResultList에 포함된 릴레이션}

(3) For i = 1 to n - 1 do

(4) Next ← Remaining의 릴레이션 중 $Delta(R_1 \bowtie \ldots \bowtie R_i \bowtie R_j)$의 크기를 가장 작게 한다고 예상되는 릴레이션 R_j

(5) ResultList의 맨 뒤에 Next 삽입
(6) Remaining ← Remaining - {Next}
(7) End for
(8) Return ResultList
□

알고리즘 1의 (4)에서 조인을 통한 중간 결과의 크기를 예상하기 위해 조인 선택도와 같은 정보를 사용할 수 있다. 여기서 차이 릴레이션과 같이 작은 크기의 릴레이션이 참여할 때, 릴레이션간에 조인 관계가 있기만 하면 이 조인의 결과는 일반적으로 작은 크기를 가진다고 가정할 경우 (4)는 다음과 같이 수정될 수 있다.

(4) Next ← Remaining의 릴레이션 중 ResultList에 포함된 릴레이션과 조인 관계가 있는 릴레이션들 중에서 크기가 가장 작은 릴레이션

결론적으로 위의 방법에 따라 기본 릴레이션들에 대한 순서를 결정한 뒤, 그의 순서에 따라 Delta 함수를 계산하는 것이 중간 결과가 커지는 일을 피하면서 Delta 방식을 최적화하는 방법이 된다.

4. 성능 평가

본 논문에서 제안하는 Delta 방식의 성능을 측정하고자 몇 가지 종류의 실험이 수행되었다. 실험을 위해 SUN UltraSPARC에 설치된 Oracle7이 사용되었으며 점진적 뷰 관리 방식의 성능은 뷰의 갱신에 사용되는 시간을 측정하여 평가되었다.

전체 실험을 통해, 기본 릴레이션으로 TPC-D[15]에서 사용되는 *CUSTOMER*(편의를 위해 C라고 표기), *ORDERS*(O), *LINEITEM*(L), *SUPPLIER*(S), *NATION*(N), *REGION*(R) 릴레이션을 사용하였으며, TPC-D의 "Local Supplier Volume Query" 질의를 사용하여 이들 기본 릴레이션들로 정의된 뷰를 생성하였다. 또한 각 기본 릴레이션에는 TPC-D에서 제공되는 데이타가 삽입되었다.

실험 1에서는 기본 릴레이션의 순서에 따른 Delta 방식의 성능과 기존의 방식에서 가장 좋은 성능을 가지고 있다고 평가된 1-way 방식[12]의 성능을 비교 평가하였다. [그림 1]은 기본 릴레이션들이 각각 5%씩 증가되었을 때 1-way 방식과 Delta 방식의 성능을 나타낸 것이다. [그림 1]에서 Delta(RNSCOL)은 릴레이션의 순서를 R-N-S-C-O-L로 했을 때의 Delta 방식을 의미한다. 기본 릴레이션의 크기간에는 |R| < |N| < |S| < |C| < |O| < |L|의 관계가 있으므로 Delta(RNSCOL)은 예상 비용이 최소가 되는 Delta 방식을 나타내며, 반대로 Delta(LOCSNR)은 예상 비용이 최대가 되는 Delta 방식을 나타낸다. [그림 1]에서 볼 수 있듯이, Delta (RNSCOL)과 Delta(RNSLOC)는 1-way 방식보다 좋은 성능을 보이고 있으며 최악의 경우를 나타내는 Delta(LOCSNR)은 1-way 방식과 비슷한 성능을 보이고 있음을 알 수 있다.

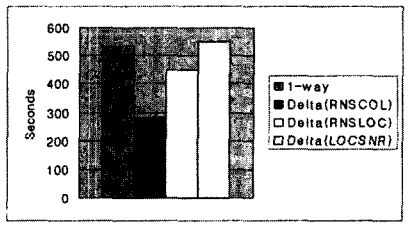

그림 1 1-way 방식과 Delta 방식의 성능 비교

실험 2에서는 기본 릴레이션의 변화율에 따른 세 가지 뷰 관리 방식(재계산 방식, 1-way 방식, Delta 방식)의 성능을 비교 평가하였다. [그림 2]는 기본 릴레이션들이 2%에서 10%까지 증가되었을 때의 각 방식의 성능의 변화를 나타낸 것이다. [그림 2]에서 볼 수 있듯이 점진적 관리 방식은 10%의 변화율 내에서 재계산 방식보다 좋은 성능을 보이고 있다. 또한 점진적 관리 방식 중에서도 Delta 방식은 1-way 방식보다 좋은 성능을 보이고 있음을 알 수 있다.

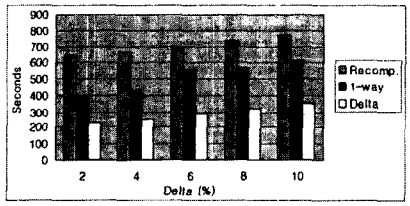

그림 2 기본 릴레이션의 변화율에 따른 뷰 관리 방식의 성능

실험 3에서는 데이타 크기의 변화에 따른 각 뷰 관리 방식의 성능 변화를 측정하였다. [그림 3]은 실험 1과 실험 2에 사용된 데이타의 크기를 1로 했을 때, 크기가 0.5, 1, 1.5, 2인 데이타에 대한 각 방식의 성능을 나타낸다. 실험 1과 마찬가지로 기본 릴레이션들은 5%씩 증가되었다. [그림 3]에서 볼 수 있듯이 데이타 크기의 증가에 따라서도 Delta 방식이 가장 좋은 성능을 보이고

있다.

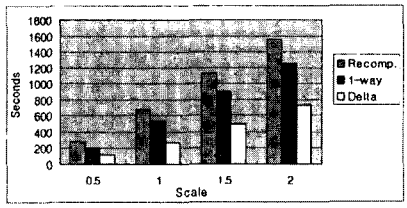

그림 3 데이타 크기의 변화에 따른 뷰 관리 방식의 성능

5. 기타 논의 사항

n개의 기본 릴레이션 R_1, R_2, ..., R_i, R_{i+1}, ..., R_n의 조인으로 정의된 뷰 V에 대해, 정의 2에서 정의된 Delta 함수는 다음과 같이 일반화된 형태로 정의될 수 있다.

$$\begin{aligned}\Delta(V) = {} & \Delta(R_1 \bowtie \ldots \bowtie R_i) \bowtie R_{i+1} \bowtie \ldots \bowtie R_n \\ & \cup R_1' \bowtie \ldots \bowtie R_i' \bowtie \Delta(R_{i+1} \bowtie \ldots \bowtie R_n)\end{aligned}$$

둘 이상의 뷰가 공통적인 기본 릴레이션들을 사용하는 경우를 생각해 보자. 이 경우에 일반화된 Delta 함수를 사용하면, Delta 함수의 계산 중에 발생하는 중간 결과들을 각각의 뷰에 대한 Delta 함수의 계산에서 재사용할 수 있게 된다. 예를 들어, 릴레이션 $\{R_1, R_2, \ldots, R_i\} \cup \{R_{i+1}, R_{i+2}, \ldots R_j\}$의 조인으로 정의된 뷰 V_1과 릴레이션 $\{R_{i+1}, R_{i+2}, \ldots R_j\} \cup \{R_{j+1}, R_{j+2}, \ldots R_n\}$의 조인으로 정의된 뷰 V_2가 있다고 가정하자. 이 때, 일반화된 Delta 함수를 사용하면 $\Delta(V_1)$와 $\Delta(V_2)$는 각각 다음과 같이 표현될 수 있다.

$$\begin{aligned}\Delta(V_1) = {} & \Delta(R_1 \bowtie \ldots \bowtie R_i) \bowtie R_{i+1} \bowtie \ldots \bowtie R_j \\ & \cup R_1' \bowtie \ldots \bowtie R_i' \bowtie \Delta(R_{i+1} \bowtie \ldots \bowtie R_j) \\ \Delta(V_2) = {} & \Delta(R_{i+1} \bowtie \ldots \bowtie R_j) \bowtie R_{j+1} \bowtie \ldots \bowtie R_n \\ & \cup R_{i+1}' \bowtie \ldots \bowtie R_j' \bowtie \Delta(R_{j+1} \bowtie \ldots \bowtie R_n)\end{aligned}$$

이 때, $\Delta(V_1)$와 $\Delta(V_2)$의 식은 모두 $\Delta(R_{i+1} \bowtie \ldots \bowtie R_j)$을 포함하고 있으므로 $\Delta(R_{i+1} \bowtie \ldots \bowtie R_j)$를 계산하면 이 결과는 $\Delta(V_1)$와 $\Delta(V_2)$의 계산에서 모두 사용될 수 있게 된다.

이렇게 둘 이상의 뷰가 공통적인 기본 릴레이션들을 사용하는 경우, 이를 고려하는 점진적 뷰 관리 방식에 대해서는 아직 기존 연구에서 논의된 바가 없다. 이단계 방식이나 1-way 방식에서는 앞서 예로든 뷰 V_1과 V_2의 갱신 작업이 완전히 독립적으로 일어나며 V_1과 V_2의 변경을 계산하는 중에 발생하는 중간 결과가 서로 공유되거나 재사용 되지 않는다. 이미 설명한 바와 같이, 일반화된 Delta 함수를 사용하면 V_1과 V_2의 변경을 계산하는 중에 발생하는 중간 결과를 서로 재사용할 수 있으므로 전체적인 성능의 향상을 기대할 수 있다. 따라서 현재 둘 이상의 뷰가 공통적인 기본 릴레이션들을 사용하는 경우, 일반화된 Delta 함수를 사용하여 뷰의 갱신 중 발생하는 중간 결과를 재사용할 수 있도록 하는 방법을 고려 중이다.

6. 결 론

본 논문에서는 여러 개의 기본 릴레이션으로 정의된 뷰에 대해, 각 기본 릴레이션에 대한 접근 횟수를 줄이면서 효과적으로 뷰의 관리 작업을 할 수 있는 기법에 대해 논의하였다. 일반적으로 n개의 릴레이션의 조인으로 이루어진 뷰에 대해, 모든 릴레이션의 변화를 한꺼번에 고려하여 뷰의 변경을 계산하는 이단계 방식은 $(2^n - 1)$과 같은 많은 수의 항을 계산해야 한다. 따라서 이 단계 방식은 n이 커짐에 따라 뷰의 변경을 계산하는데 적합하지 않게 된다. 한편, 릴레이션의 변화를 한번에 하나씩만 고려하여 순차적으로 뷰에 반영하는 1-way 방식은 이단계 방식에 비해 효율적이나 각 기본 릴레이션에 대한 접근이 그의 크기에 관계없이 모두 (n - 1)번씩 이루어져야 한다는 단점이 있다. 그러나 본 논문에서 제안하는 Delta 방식은 각 릴레이션에 대해 각각 1, 2, ..., (n - 1)번씩의 접근만 이루어지면 된다. 따라서 각 릴레이션에 대한 차이 릴레이션의 크기가 작은 경우, 뷰를 갱신하기 위한 작업의 대부분의 시간은 릴레이션들을 메모리로 읽어들이는 시간에 사용되므로 이러한 경우에 Delta 방식은 매우 효과적으로 사용될 수 있다. 또한 릴레이션의 크기가 커질수록 그에 대한 접근 횟수가 줄어들게 되므로, 커다란 릴레이션이 포함된 경우 이에 대한 접근 횟수를 크게 줄임으로써 전체적인 성능의 향상을 가져올 수 있다. TPC-D 데이타를 사용한 실험은 Delta 방식이 이단계 방식이나 1-way 방식에 비해 좋은 성능을 가지고 있음을 보이고 있다.

참 고 문 헌

[1] W. H. Inmon, "Building the Data Warehouse," WILEY COMPUTER PUBLISHING, 1996.

[2] J. Widom. "Research Problems in Data Warehousing," In Proceedings of 4th International Conference on Information and Knowledge

Management, November 1995.
[3] A. Gupta, I. S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications," IEEE Data Eng. Bulletin, Special Issue on Materialized Views and Data Warehousing, Vol 18, No. 2, 1995.
[4] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, "View Maintenance in a Warehousing Environment," In Proceedings of ACM SIGMOD Confernece, 1995.
[5] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener, "The Strobe Algorithms for Multi-Source Warehouse Consistency," In Proceedings of the International Conference on Parallel and Distributed Information Systems, December 1996.
[6] D. Agrawal, A. E. Abbadi, A. Singh, and T. Yurek, "Efficient View Maintenance at Data Warehouses," In Proceedings of ACM SIGMOD Conference, 1997.
[7] I. S. Mumick, D. Quass, and B. S. Mumick, "Maintenance of Data Cubes and Summary Tables in a Warehouse," In Proceedings of ACM SIGMOD Conference, 1999.
[8] J. A. Blakeley, P. Larson, F. W. Tompa, "Efficiently Updating Materialized Views," In Proceedings of ACM SIGMOD Conference, p61-71, 1986.
[9] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining views incrementally," In Proceedings of ACM SIGMOD Conference, p157-166, 1993.
[10] T. Griffin and L. Libkin, "Incremental maintenance of views with duplicates," In Proceedings of ACM SIGMOD Conference, p328-339, 1995.
[11] D. Quass, "Maintenance expressions for views with aggregation," In Workshop on Materialized Views: Techniques and Applications, June 1996.
[12] W. J. Labio, R. Yerneni, and H. Garcia-Molina, "Shrinking the Warehouse Update Window," In Proceedings of ACM SIGMOD Conference, 1999.
[13] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey, "Algorithms for deferred view maintenance", In Proceedings of ACM SIGMOD Conference, p469-492, 1996.
[14] V. Harinarayan, A. Rajaraman, and J. Ullman, "Implementing data cubes efficiently," In Proceedings of ACM SIGMOD Conference, p205-216, 1996.
[15] TPC Committee, Transaction Processing Council, http://www.tpc.org/.

이 기 용
1998년 한국과학기술원 전산학과 학사. 2000년 한국과학기술원 전산학과 석사. 2000년 ~ 현재 한국과학기술원 전산학과 박사과정 재학중. 관심 분야는 데이타웨어하우스, 무선 통신.

김 명 호
정보과학논문지 : 데이타베이스
제 27 권 제 1 호 참조