경영정보학연구
제13권 제4호
2003년 12월

데이터웨어하우징에서 자립유지를 지향하는 실체뷰 관리 정책

김 근 형*

Policy of Managing Materialized Views by Orienting toward Self-Maintenance in Data Warehousing

Keun-Hyung Kim

More views in data warehouse, can respond to the users more rapidly because the user's requests might be processed by accessing only the materialized views with higher probabilities rather than accessing base relations. But, the update duration for maintaining materialized views limits the number of materialized views in data warehouse.

In this paper, we propose the algorithm for reducing update duration of materialized views, of which aggregation functions are maintained by self-maintenance. We also implement the proposed algorithm and evaluate the performance of the algorithm.

* 제주대학교 경영정보학과 교수

Ⅰ. 서 론

의사결정지원시스템(Decision Support System)은 기업 경영활동에서 비즈니스 구성원들의 합리적인 의사결정을 지원하기 위한 정보시스템으로, 사용자의 요구에 신속하게 대응하여 필요한 데이터를 추출하고 분석하는 기능을 제공한다. 90년대 중반 이후 주요 정보기술중의 하나로 떠오른 데이터웨어하우징(DW, data warehousing)과 OLAP(On Line Analytical Processing)은 의사결정지원시스템의 중요한 요소가 되었고 현재 기업경영혁신 및 정보 전략화 프로젝트의 핵심이 되고 있다. 데이터웨어하우징이란, 조직내의 각 데이터 저장소들로부터 추출한 데이터를 중앙의 단일 데이터베이스에 통합하여 저장한 '데이터웨어하우스(DW, data warehouse)'를 관리하는 기법을 의미한다[정희정, 2000]. OLAP 시스템은 이러한 데이터웨어하우스에 저장된 방대한 양의 데이터에 대하여, 의사결정과정에 도움을 주는 빠르고 직접적인 자료분석 작업을 수행한다[정희정, 2000].

데이터웨어하우스에는 일반적으로 수 백 기가바이트(GB)에서 테라 바이트(TB)에 이르는 매우 방대한 양의 데이터가 저장된다. 또한, 대부분의 의사결정지원 질의(decision support query)는 개별적인 레코드 정보의 검색이 아닌, 레코드 집합의 전체적인 정보와 경향분석 등을 수행하므로 많은 수의 집단연산(aggregation)을 포함한다. 이와 같이 데이터의 양이 방대하고 수행되는 질의가 복잡하기 때문에, 대부분의 의사결정지원 질의는 처리 시 많은 시간이 소요된다. 그러나, OLAP 시스템은 의사결정과정에 즉시 반영될 수 있는 직접적인 데이터 분석을 수행해야 하므로, 빠른 질의응답 속도의 보장이 필수적이다. 이를 위하여 데이터웨어하우징 분야에서는 자주 수행되는 질의의 처리에 필요한 정보를 미리 계산하여 저장하고, 실제 질의처리 시에는 저장된 정보를 활용하여 질의응답 속도를 줄이고자 하는 선계산(pre-computation)기법이 활발히 연구되고 있다.

선계산 기법은 기본릴레이션에서 자주 사용되는 종류의 데이터에 대해 이를 정리하고 요약한 내용을 실체뷰(materialized view)의 형태로 저장한다[이기용, 2000]. 사용자의 정보요구가 기본릴레이션 대신 실체뷰에 의하여 처리된다면, 요구사항을 새롭게 계산할 필요없이 미리 계산한 결과값을 보여주는 방식이 될 것이므로 보다 신속한 응답을 제공할 수 있다. 실체뷰가 많아질수록 사용자 질의를 기본릴레이션 대신 실체뷰에서 처리할 확률이 높아지게 되고 이로 인하여 신속한 처리가 가능하게 되므로 사용자의 정보요구에 대한 만족도는 높아지게 된다. 실체뷰에 포함되는 요약 정보는 가능한 한 최신의 기본릴레이션의 데이터를 기반으로 구성되어야 정확한 정보로서의 역할을 할 수 있다. 기업활동으로 발생한 기본릴레이션의 변경은 모아졌다가 기업활동이 중단되는 밤시간을 이용하여 실체뷰에 반영 및 갱신된다. 이러한 갱신작업은 기업활동이 재개되는 다음날 아침까지는 완료되어야 하는 시간제약이 있다. 실체뷰의 갱신작업 동안에는 데이터웨어하우스로의 접근을 통한 정보분석이 중단될 수 있기 때문이다. 따라서, 실체뷰가 많아질 수록 실체뷰 갱신시간 또한 길어지므로 유지할 수 있는 실체뷰의 개수에는 한계가 있다. 데이터웨어하우스내에서 실체뷰 갱신작업을 자동으로 처리해주는 모듈(즉, 알고리즘)에 의하여 개별적인 실체뷰들의 갱신이 짧은 시간에 처리될 수 있다면 보다 많은 실체뷰들이 데이터웨어하우스내에 유지될 수 있고 사용자의 정보요구에 대한 보다 신속한 응답이 가능하게 되어 데이터웨어하우스의 성능은 향상될 수 있다.

본 논문에서는 실체뷰 정의내에 포함된 보다 많은 집단함수들을 자립유지(self-maintenance)에 의하여 처리함으로서 실체뷰의 갱신시간을 단축시킬 수 있는 알고리즘을 제안한다.

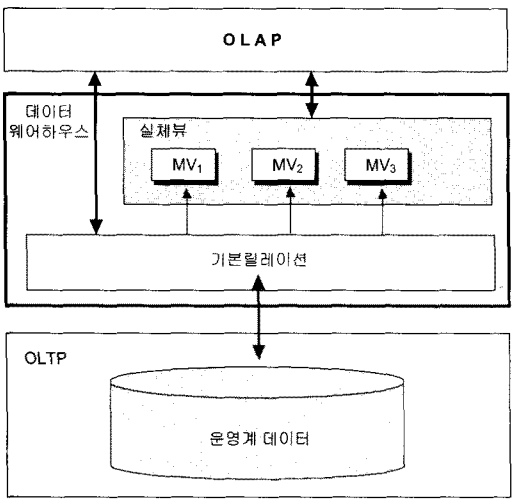
본 논문의 구성은 다음과 같다. Ⅱ장에서는

데이터웨어하우징에서의 뷰관리 정책과 관련된 기존 연구들을 고찰한다. Ⅲ장에서는 보다 많은 집단함수들을 자립유지에 의하여 처리함으로써 실체뷰 갱신시간을 단축시킬 수 있는 자립유지지향(self-maintenance-oriented) 알고리즘을 제안한다. Ⅳ장에서는 자립유지지향 알고리즘을 실제 예제를 통하여 고찰해 본다. Ⅴ장에서 제안한 알고리즘의 성능을 평가 분석하고 Ⅵ장에서 결론을 맺는다.

Ⅱ. 선행 연구

2.1 데이터웨어하우스

<그림 1>에서 볼 수 있는 것처럼, 일반적으로 데이터웨어하우스에는 외부의 운영계 데이터(operational data)들에 대해 통합 및 정제의 과정을 거친 기본릴레이션들이 생성 및 저장된다.



<그림 1> 데이터웨어하우스 모델

기본릴레이션들은 다차원 데이터분석을 위하여 사실테이블(fact table)이나 차원테이블(dimension table)의 형태로 존재한다. 대부분의 데이터웨어하우스 시스템은 이러한 기본릴레이션들에서 자주 사용되는 종류의 데이터에 대해 이를 정리하고 요약한 내용을 실체뷰들(MV_1, MV_2, MV_3, …)의 형태로 저장한다.

데이터웨어하우징은 데이터웨어하우스내의 기본릴레이션이나 실체뷰 등을 생성하고 갱신하는 등 데이터웨어하우스를 관리하는 작업이라고 할 수 있다.

운영계 데이터의 변화는 데이터웨어하우스내의 기본릴레이션들의 변경을 유도하고 또한 실체뷰의 변경을 유발한다. 이러한 실체뷰 및 기본릴레이션의 갱신작업을 데이터웨어하우징의 뷰관리 기능이라고 한다[이기용, 2000]. 기본릴레이션의 변경으로 인한 실체뷰 갱신방식은 3가지로 고려될 수 있다 [Colby, 1997].

- 즉각갱신(immediate update):
  기본릴레이션의 변경은 즉각적으로 실체뷰에 반영된다.
- 지연갱신(deferred update):
  사용자의 질의요구가 발생될 때까지 실체뷰의 갱신을 지연한다.
- 스냅샷갱신(snapshot update):
  일정기간이 경과한 후 한꺼번에 실체뷰를 갱신한다.

기본릴레이션의 변경을 즉각적으로 실체뷰에 반영하면 시스템의 부담으로 사용자 요구에 대한 응답시간이 느려질 수 있다. 질의요구를 처리할 때 마다 실체뷰를 변경하는 것도 신속한 질의응답을 저해한다. 따라서, 데이터웨어하우징에서는 하루나 일주일의 간격을 두어 주기적으로 실체뷰의 갱신작업을 수행하는 스냅샷 갱신기법이 일반적이다[Colby, 1997].

2.2 실체뷰 관리

어떤 실체뷰가 기본릴레이션들에 의해 정의되어 있을 때 기본릴레이션의 변화를 뷰에 반영

하는 방법은 크게 다음의 두 가지로 나뉜다.

- 재계산(recomputation): 각각의 기본릴레이션들을 먼저 갱신하고 갱신된 기본릴레이션들로부터 실체뷰들을 새로 계산한다. 기본릴레이션의 변화가 적을 경우에 이렇게 실체뷰 전체를 다시 계산하는 일은 매우 비효율적이다.
- 점진적 관리(incremental maintenance): 변경전 기본릴레이션들과 변경된 튜플들(삽입, 삭제, 갱신의 결과로 생김)만으로 된 델타릴레이션(delta relation)들(즉, 기존 릴레이션과 차이나는 부분)로부터 변경된 뷰의 내용만을 계산한 뒤, 이 계산된 내용을 뷰에 반영하는 방법이다. 기본릴레이션의 변화가 적은 경우, 이 방법은 재계산에 비해 효율적이다.

<표 1> 재계산과 점진적 관리방법

	실체뷰 갱신과정
재계산	[step1]: $R_i' = R_i + \Delta R_i$ [step2]: $MV_i' = R_i' \bowtie R_j$
점진적 관리	[step1]: $\Delta MV_i = \Delta R_i \bowtie R_j$ [step2]: $R_i' = \Delta R_i \cup R_i$ [step3]: $MV_i' = \Delta MV_i \cup MV_i$

<표 1>은 재계산 방법과 점진적 관리에 의한 실체뷰의 갱신작업과정을 나타내고 있다. <표 1>은 기본릴레이션 R_i의 변화가 생겼을 때 실체뷰 MV_i를 갱신하는 과정이다. <표 1>에서 ΔMV_i는 MV_i의 델타릴레이션이고 ΔR_i는 R_i의 델타릴레이션을 의미한다. 점진적 관리방식의 기존 연구들은 두개 이상의 기본릴레이션이 변경될 때, 실체뷰를 갱신하기 위한 효율적인 표준식들을 제안하고 있으나 본 논문에서는 하나의 기본릴레이션의 변경만을 가정하겠다. 또한, 점진적 관리 관련 대부분의 기존 연구들은 SPJ(Selection-Projection-Join) 식으로 표현되는 뷰만을 주로 고려하였지만 본 논문에서는 집단함수나 Group By와 같은 식이 포함되어 있는 뷰로 확장하여 논의한다.

2.3 자립유지 집단함수

집단함수(Aggregation Function)는 세 부류 즉, 분배집단함수(distributive),대수집단함수(algebraic), 홀리스틱 집단함수(holistic)로 구분된다[Mumick, 1997]. 분배집단함수는 처리할 영역의 데이터들을 분할하여 각각을 계산한 다음, 각각의 결과를 통합하여 계산하면 전체 영역에 대한 타당한 결과를 얻을 수 있는 함수이다. SQL 표준 집단함수 중 COUNT, SUM, MIN, MAX는 분배집단함수이다. 예를 들면, COUNT는 분할된 영역의 COUNT 결과값들을 다시 합하여 전체 영역의 최종적인 결과를 유도할 수 있다. 대수집단함수는 분배 집단함수의 수식 계산으로 표현될 수 있는 함수이다. 예를 들면, AVG는 SUM/COUNT로 표현될 수 있다. 홀리스틱 함수는 계산할 영역을 분할하여 처리할 수 없는 함수이다. MEDIAN이 그 예이다.

[정의 1] 자립유지(self-maintainable) 집단함수
실체뷰에 포함된 집단함수의 새로운 결과값(new value)을 구하는 경우, 기본릴레이션의 변경부분과 집단함수의 이전 결과값(old value)에 의해서 타당하게 계산될 수 있다면, 그 집단함수는 자립유지가 가능하다. □

분배 집단함수는 삽입 및 삭제갱신에 대해서 일반적으로 자립유지가 가능하지만 MIN/MAX의 경우는 삽입갱신일 때만 자립유지가 가능하고 삭제갱신일 경우는 자립유지가 불가능하다[Mumick, 1997].

2.4 실체뷰 관련 연구들의 고찰

실체뷰와 관련된 연구는 실체뷰 선택과 실체뷰 관리의 분야로 나눌 수 있다. 실체뷰선택은

사용자의 신속한 응답요구를 만족시키기 위하여 어떤 뷰들을 실체뷰로 선택해야 할 것인가의 이슈를 다룬다. 윤원식[2001]은 조인 비용을 기반으로 한 실체뷰 선택 알고리즘을 제안하고 있고 Theodoratos[2000]은 다양한 목적하의 뷰선택 문제를 일반화시킬 수 있는 프레임웍을 제안하였다. 김민정[2001]은 데이터큐브의 일부를 나타내는 세분화된 뷰의 표현 및 선택 문제를 다루고 있다. 실체뷰관리는 기본릴레이션의 변경에 따라 이를 보다 신속하고 효율적으로 실체뷰에 반영하기 위한 이슈를 다룬다. Zhuge[1995], Agrawal[1997], 이기용[2000] 등은 운영계 데이터 또는 기본릴레이션의 변화가 생길 때 뷰를 전부 새로 계산하지 않고 뷰의 변경될 부분만을 계산하여 신속한 뷰갱신 작업을 처리하기 위한 점진적 뷰관리 기법들을 제안하고 있다. Colby[1997]는 OLTP(On Line Transaction Processing) 환경에서의 뷰관리 정책을 다루고 있는데, 지연갱신, 즉각갱신, 스냅샷 갱신의 장단점을 고려하여 효율적으로 혼합 적용할 수 있는 방안을 제안하고 있다. Kotidis[1999, 2000]은 데이터웨어하우징 환경에서 임의의 사용자 질의를 특정 척도를 기준으로 평가하여 자동적으로 실체뷰화 시키고 관리하는 정책을 제안하고 있다.

Mumick[1997]은 데이터웨어하우징 환경에서 점진적 뷰관리 정책과 스냅샷 갱신방식이 적용될 때, 실체뷰 갱신을 위한 선계산 작업을 하는 과정인 전파(propagation)단계와 실질적인 갱신작업을 하는 과정인 회복(refresh) 단계로 나누어 갱신시간을 단축시킬 수 있는 요약델타 알고리즘(summary delta algorithm)을 제안하고 있다. 양우석[2000], 정희정[2000]은 효율적인 인덱싱 등의 기법을 사용하여 기본릴레이션에서 MIN/MAX 집단함수를 신속하게 처리할 수 있는 방안을 제안하고 있다. 실체뷰와 관련된 기존의 연구들을 정리하면 <표 2>와 같다.

DW 환경에서의 실체뷰관리는 현실적으로 스냅샷갱신이 일반적이므로 본 논문에서는 실체뷰 관리 기법중 Mumick이 제안한 요약델타 알고리즘의 문제점을 분석하고 이를 개선하여 실체뷰의 갱신시간을 단축시킬 수 있는 알고리즘을 제안할 것이다. Kotidis[1999, 2001]이나 이기용[2000]은 본 논문의 관점보다 거시적인 차원에서 실체뷰관리의 문제를 다루고 있으므로 본 논문에서 제안하는 알고리즘은 Kotidis[1999, 2001]이나 이기용[2000]에도 적용될 수 있을 것으로 기대한다.

<표 2> 실체뷰 관련 연구들의 정리

구 분	연 구 논 문	비 고
실체뷰 선택	윤원식[2001] 김민정[2001] Theodoratos [2000]	• 바람직한 실체뷰의 선택을 통한 DW의 성능 향상
실체뷰 관리	Zhuge[1995]	• DW 환경에서 즉각갱신시의 이상현상(anomaly) 처리
	Agrawal[1997]	
	Colby[1997]	• OLTP 환경에서의 뷰관리 • 즉각갱신, 지연갱신, 스냅샷갱신의 혼합처리 방안 제시
	Mumick[1997]	• DW 환경에서 스냅샷 갱신 가정 • 선계산 작업을 통한 실체뷰 갱신시간 단축
	이기용[2000]	• 기본릴레이션들의 효율적인 조인을 통한 실체뷰 갱신방안 제시 • 실체뷰 정의내에 포함된 집단함수는 고려하지 않음
통합	Kotidis[1999] Kotidis[2001]	• 실체뷰선택 및 관리의 통합모델 제시 • MIN/MAX 집단함수는 고려하지 않음
기타	양우석[2000], 정희정[2000]	• 기본릴레이션에서의 MIN/MAX 집단함수의 처리

2.5 요약델타 알고리즘의 분석

데이터웨어하우스에서 유지하는 실체뷰들의

갯수가 많을 수록 사용자의 정보요구에 대한 만족도는 높아진다. 왜냐하면, 사용자들의 다양한 요구질의들에 대하여 기본릴레이션에 의존하기 보다는 실체뷰들을 이용하여 보다 신속하게 응답할 수 있는 확률이 높아지기 때문이다. 많은 실체뷰들을 유지하기 위한 제약사항으로 첫 번째, 저장공간의 한계를 고려해야 하지만 기억장치 가격의 점차적 하락으로 저장용량의 제약은 비교적 민감하지 않다[Kotidis, 1999]. 두 번째 제약사항으로 기본릴레이션의 변화에 따른 실체뷰의 갱신시간이다. 실체뷰의 갱신작업이 이루어지는 동안에는 데이터베이스 일관성(consistency) 유지 측면에서 실체뷰들을 잠금상태(lock)로 만들어야 하고 데이터웨어하우스로의 정보조회 요구는 중단된다. 따라서, 실체뷰의 갱신작업은 기업 경영활동이 중단되는 밤시간(night duration)을 이용하여 이루어져야 하는데, 각 실체뷰의 갱신시간이 짧을 수록 보다 많은 수의 실체뷰들을 제한된 시간(밤시간)내에 갱신할 수 있다. 이런 관점에서 볼 때, 데이터웨어하우징에서의 뷰관리의 중요한 목적중의 하나는 실체뷰들의 갱신시간을 단축시키는 것이다[Mumick, 1997]. 요약델타 알고리즘은 실체뷰 갱신시간을 줄이기 위하여 실체뷰 갱신작업을 2단계로 나누어서 처리할 수 있다(여기서 요약은 실체뷰와 동일한 의미임). 첫번째 단계인 전파단계(propagation phase)에서는 기본릴레이션의 변화부분인 델타릴레이션(delta relation)에 대하여 실체뷰의 정의를 적용하여 실체뷰의 델타릴레이션을 만드는 과정이다. 즉, 신속한 갱신작업을 위한 선계산(pre-computation)을 하는 과정인데, 실체뷰를 잠금상태로 만들지 않고 처리될 수 있다. 두번째 단계인 회복단계(refresh phase)에서는 전파단계 동안에 생성된 실체뷰의 델타릴레이션을 기반으로 실체뷰를 갱신하는 과정이다.

요약델타 알고리즘은 MIN/MAX 집단함수를 포함하는 실체뷰의 경우, 기본릴레이션의 MIN/MAX 값의 변경 정도가 많아지면 실체뷰의 갱신시간은 길어지는 단점이 있다. 현실적인 예로 부서별 영업사원의 최저 실적과 최고 실적을 보유하는 실체뷰의 경우 실체뷰의 MIN/MAX 속성인 최저/최고 실적 값은 매일 변경될 것이다. 이러한 예를 만들려면 실체뷰를 여러개 정의하여야 하고 복잡한 상황이 될 수 있으므로 본 논문에서는 다소 비현실적이라 할 지라도 다음과 같은 단순한 예를 도입한다. 즉, 어떤 매장에서 하루중 특정 부류의 제품이 판매된 최저시점 및 판매갯수 등을 보유하는 실체뷰를 다룬다.

[예제]: 데이터웨어하우스에 기본릴레이션 pos, items가 저장되어 있고 이들로부터 실체뷰 sales가 정의되어 있다고 하자.

```
pos (storeID, itemID, time, qty, price)
items (itemID, name, category, cost)

CREATE VIEW sales (storeID, category,
      t_count, e_time, t_qty) As
SELECT storeID, category,
          COUNT ( * ) AS t_count,
          MIN (time) AS e_time,
          SUM (qty) AS t_qty
FROM pos, items
WHERE pos.itemID = items.itemID
GROUP BY storeID, category
```

기본릴레이션의 델타릴레이션은 삽입릴레이션과 삭제릴레이션으로 나누어지는데, 삽입릴레이션은 기본릴레이션에 삽입된 튜플들로 이루어지고 삭제릴레이션은 기본릴레이션에서 삭제된 튜플들로 이루어진다. 전파단계에서는 삽입릴레이션과 삭제릴레이션 각각에 대하여 실체뷰 정의의 셀렉션 조건(selection conditions)과 프로젝션 조건(projection condition)만을 적용한 후 유니온(union)연산을 적용하여 실체뷰의 델타릴레이션을 생성한다. 프로젝션 조건에서는 실체뷰 정의에 나타나는 Group By 속성들과 집단함수 속성들을 모두 포함한다. 이때, 집단함수 COUNT 속성값으로는 삽입릴레이션의 경우는

1값을 갖고 삭제일레이션의 경우는 -1을 그 값으로 갖는다. 집단함수 SUM(attr) 속성값으로는 삽입릴레이션의 경우는 attr 값을 갖고 삭제릴레이션의 경우는 -attr을 그 값으로 갖는다.

```
CREATE VIEW D_sales (
SELECT storeID, category,
        SUM(count) AS D_count,
        MIN(time) AS D_E_Time,
        SUM(quantity) AS D_qty
FROM ((SELECT storeID, category, time
                1 AS count, qty AS quantity
        FROM pos_ins, items
        WHERE pos_ins.itemID = items.itemID)
        UNION ALL
        (SELECT storeID, category, time
                -1 AS count, -qty AS quantity
        FROM pos_del, items
        WHERE pos_del.itemID = items.itemID))
GROUP BY storeID, category
```

<그림 2> 전파단계의 예

<그림 2>는 실체뷰 sales의 델타릴레이션 D_sales를 계산하는 과정을 나타내고 있다. <그림 2>에서 pos_ins는 pos의 삽입릴레이션이고 pos_del은 pos의 삭제릴레이션이다. 주목할 사항으로서, 실체뷰 sales의 델타릴레이션 D_sales의 MIN/MAX 집단함수 속성 D_E_Time의 값은 삽입릴레이션으로부터 유도된 것인지 삭제릴레이션으로부터 유도된 것인지 구분되어 있지 않으므로, 삽입릴레이션으로부터 유도되었다 할 지라도 자립유지가 가능하지 않다. 이 부분은 요약델타 알고리즘의 문제점이라 할 수 있다.

<그림 3>은 요약델타 알고리즘에서 회복단계의 처리과정을 나타내고 있다. 실체뷰 sales와 실체뷰의 델타릴레이션 D_sales의 각각의 튜플들에 대하여 Group By 속성값을 기준으로 부합하는 튜플을 찾아 실체뷰 sales의 집단함수 속성들 t_count, t_qty를 갱신한다((3)의 처리절차). 집단함수 SUM과 COUNT 속성은 기본릴레이션에 접근하지 않고 자립유지에 의하여 처리 가능함을 알 수 있다. sales와 D_sales에 있는 각 튜플들은 Group By 연산이 적용된 결과이기 때문에 D_sales에 있는 임의의 튜플 δt와 부합하는 sales 내의 튜플은 유일하거나 존재하지 않는다. 부합하는 튜플이 없으면 D_sales의 해당 튜플은 sales에 삽입된다((1)의 처리절차). 부합하는 튜플들에 대하여 COUNT 속성값의 합이 0이면 그 튜플은 sales에서 삭제된다((2)의 처리절차). 처리절차 (4)에서는 D_sales 에 있는 임의의 튜플 δt와 부합하는 sales내의 튜플 t의 MIN 속성값을 비교하여 δt의 MIN 속성값이 작거나 같으면 기본릴레이션에 접근하여 재계산하는 과정을 나타내고 있다. 그러나, δt의 MIN 속성값이 삽입릴레이션으로부터 유도된 것이라면 자립유지가 가능하기 때문에 기본릴레이션에 접근하지 않고도 타당한 값으로 갱신할 수 있지만 요약델타 알고리즘은 기본릴레이션에 접근하여 재계산을 해야 한다. 따라서, δt의 MIN 속성값이 삽입릴레이션으로부터 유도된 것인지, 삭제릴레이션으로부터 유도된 것인지를 판별하여 재계산 횟수를 줄일 필요가 있다.

```
For each tuple δt in D_sales
  Let tuple t =
    (SELECT *
     FROM sales
     WHERE sales.storeID = δt.storeID AND
           sales.category = δt.category)
 If t is not found
(1)   Insert tuple δt into sales ;
 Else
   If δt.D_count + t.t_count = 0
(2)   Delete tuple t from sales ;
 Else
(3)   Update tuple t.t_count += δt.D_count,
          t.t_qty += δt.D_qty ;
      If δt.D_E_Time ≤ t.e_time
(4)   recompute;
```

<그림 3> 회복단계의 예

Ⅲ. 자립유지지향 알고리즘

요약델타 알고리즘은 실체뷰의 MIN/MAX 속성값들을 갱신할 때, 삽입연산으로 인한 갱신임에도 불구하고 자립유지가 보장되지 않는다. 본 논문에서 제안하는 자립유지지향(self-maintenance-oriented) 알고리즘은 실체뷰의 MIN/MAX 속성값들을 갱신할 때, 삽입연산으로 인한 갱신은 자립유지를 보장한다. 자립유지지향 알고리즘은 실체뷰 갱신작업을 위하여 3단계의 처리과정을 수행한다. 단계 1과 단계 2는 선계산 과정이고 단계 3은 단계 1, 2의 결과를 이용하여 실체뷰를 갱신하는 과정이다.

Algorithm1: Phase-1( )
input: ΔI_R_i, ΔD_R_i, R_j, SP_Func, Group_Func
output: ΔMV_i
variable: I_R_view, D_R_view
Begin
(1) I_R_view ← SP_Func ($\Delta I\_R_i \bowtie R_j$)
(2) D_R_view ← SP_Func ($\Delta D\_R_i \bowtie R_j$)
(3) ΔMV_i ← Group_Func (I_R_view ∪ D_R_view)
(4) add attribute DEL (of which value is set to "no") to ΔMV_i
End

<그림 4> 단계 1의 알고리즘

단계 1의 Phase-1() 알고리즘은 요약델타 알고리즘의 전파단계와 유사하다. 단, 실체뷰의 델타릴레이션의 MIN/MAX 속성값이 삽입릴레이션으로부터 유도된 것인지, 삭제 릴레이션으로부터 유도된 것인지를 기록하기 위한 속성을 추가한다. <그림 4>은 Phase-1() 알고리즘을 나타내고 있다. <그림 4>에서 ΔI_R_i는 기본릴레이션 R_i의 삽입릴레이션, ΔD_R_i는 기본릴레이션 R_i의 삭제릴레이션을 의미한다. SP_Func(X)는 X릴레이션에 실체뷰 정의의 셀렉션 조건과 프로젝션 조건을 적용한다는 의미이다. Group_Func (X)는 X릴레이션에 Group By 연산 및 집단함수 연산을 적용한다는 의미이다. 따라서, Phase-1() 알고리즘에서 (1), (2), (3) 과정을 통하여, 삽입릴레이션 ΔI_R_i 와 삭제릴레이션 ΔD_R_i로부터 실체뷰 MV_i의 델타릴레이션 ΔMV_i를 구할 수 있다. (4)의 과정은 실체뷰의 델타릴레이션 ΔMV_i의 MIN/MAX속성값이 삽입릴레이션으로부터 유도된 것인지, 삭제 릴레이션으로부터 유도된 것인지를 기록하기 위한 속성 DEL을 추가하는 과정이다.

단계 2에서는 실체뷰의 델타릴레이션의 MIN/MAX 값이 삽입된 값인지 삭제된 값인지를 판별하는 과정이다. <그림 5>는 단계 2의 처리과정을 나타내는 Phase-2() 알고리즘이다. ΔMV_i와 D_R_view를 Group By 속성들을 기준으로 조인하여, ΔMV_i의 MIN/MAX 속성값(δt.m)과 D_R_view의 MIN/MAX 속성값(t.m)이 같으면 ΔMV_i 의 DEL 속성값을 "yes"로 변경한다. 이것은 ΔMV_i 의 MIN/MAX 값이 삽입릴레이션에서 생성된 것일 경우 자립유지에 의하여 실체뷰의 MIN/MAX 값을 갱신하기 위한 선계산 작업이다.

Algorithm2: Phase-2( )
input: ΔMV_i, D_R_view
variable: t, δt
Begin
  For each tuple δt in ΔMV_i
    Let tuple t = tuple in D_R_view having the same values for its group-by attributes as δt
    If δt.m = t.m then
      δt .DEL = "yes"
    end if
  end for
End

<그림 5> 단계 2의 알고리즘

<그림 5>의 알고리즘의 처리시간을 단축시키려면 ΔMV_i, D_R_view 릴레이션에 인덱스를 설

정하여 <그림 6>과 같은 조인 연산에 의한 갱신을 적용할 수 있다. 그러나, 이러한 방법은 인덱스를 설정하는 시간뿐만 아니라 인덱스를 유지하는 시간에 대한 오버헤드가 존재할 수 있다.

```
UPDATE ⊿MVi
 SET DEL = "yes"
 WHERE EXISTS (
        select *
        from D_R_view
        where ⊿MVi.group-by attribute
                = D_R_view.group-by attribute);
```

<그림 6> 단계 2를 위한 SQL 질의

단계 3 에서는 단계 1, 단계 2 동안에 생성된 실체뷰의 델타릴레이션 ΔMV_i를 기반으로 실체뷰 MV_i를 갱신하는 과정이다. <그림 7>는 단계 3을 위한 알고리즘을 나타내고 있다. ΔMV_i에 있는 튜플 δt와 MV_i의 튜플 t는 Group By 연산이 적용된 튜플이다. Phase-3( ) 알고리즘은 ΔMV_i에 있는 각 튜플 δt에 대하여 실체뷰의 Group By 속성값을 기준으로 MV_i에서 부합되는 튜플 t를 찾아 갱신작업을 수행한다. MV_i에 있는 각 튜플들은 Group By 연산이 적용된 결과이기 때문에 ΔMV_i에 있는 튜플 δt와 부합하는 튜플은 유일하거나 존재하지 않는다. Phase-3( ) 알고리즘의 (1)의 과정을 통하여 ΔMV_i에는 존재하고 MV_i에는 존재하지 않는 튜플은 삽입된다. 즉, ΔMV_i에 있는 임의의 튜플 δt 에 대하여 실체뷰의 Group By 속성값을 기준으로 부합하는 튜플이 MV_i에 존재하지 않는 경우 그 튜플 δt는 MV_i에 삽입된다. δt.COUNT(*)는 Group By 연산에 의하여 δt를 유도한 튜플들의 갯수이다.

δt를 유도한 튜플들은 삽입 튜플들과 삭제튜플들이 혼합되어 있는 경우이므로 δt. COUNT(*) 값은 정수값이다. δt.COUNT(*)값이 음수인 경우 δt를 group by 연산에 의하여 유도한 튜플들은 삽입튜플들보다 삭제튜플들이 더 많다는 의미

```
Algorithm3: Phase-3()
input: ⊿MVi , MVi
variable: t, δt
begin
  /* M_Vi 는 실체뷰 중의 하나 */
  /*⊿MVi 는 실체뷰 MVi 의 델타릴레이션*/
For each tuple δt in ⊿MVi {
  Let tuple t = tuple in MVi having the same
                values
          for its group-by attributes as δt
  If t is not found
(1)   { Insert tuple δt into MVi } /* 튜플 삽입*/
  Else   /* COUNT함수를 이용하여 튜플 t가
            삭제될 필요가 있는지 체크 */
       If δt .COUNT(*) + t.COUNT(*) = 0
(2)   { Delete tuple t from MVi }
    /* MIN/MAX 값을 계산하기 위하여 기본
       릴레이션으로의 접근여부 체크 */
    Else {
(3)    recompute = false
       for each aggregation function a(e) in the
       MVi
(4)       { if ((a is MIN function) AND
                (δt.MIN(e) ≤ t.MIN(e)) AND
                (δt.DEL = "YES")) OR
                ((a is MAX function) AND
                (δt.MAX(e) ≥ t.MAX(e)) AND
                (δt.DEL = "YES"))
(5)                { recompute = true }
          if (recompute)
(6)                { Update tuple t by
                     recomputing its aggregate
                     functions from the base
                     data from t's group.}
          else {
               if a is COUNT or SUM
(7)                t.a = t.a + δt.a
               else If a is MIN
(8)                t.a = MIN(t.a, δt.a)
               else If a is MAX
(9)                t.a = MAX(t.a, δt.a) }
          }        }
end
```

<그림 7> 단계 3 알고리즘

이다. δt와 t가 group by 속성값을 기준으로 부합하고 「δt .COUNT(*) + t.COUNT(*) = 0」이면 (2)의 단계를 통하여 MV_i 의 튜플 t 는 삭제되어야 한다.

δt와 t가 Group By 속성값을 기준으로 부합하고 「δt .COUNT(*) + t.COUNT(*) > 0」이면 튜플 t의 속성값들을 δt 의 속성값들을 기반으로 갱신해야 한다. 튜플 t 의 속성값이 MIN/MAX 함수에 의하여 유도되었을 경우 δt 의 MIN/MAX값의 크기와 비교하고 삭제 태그를 체크하여 기본릴레이션으로부터의 재계산(recomputation) 여부를 결정한다. 재계산이 필요 없을 경우 δt의 속성값들과 기존의 t의 속성값을 이용하여 새로운 t의 속성값들을 자립유지에 의하여 계산한다.

Ⅳ. 자립유지지향 알고리즘의 예

여기에서는 자립유지지향 알고리즘인 Phase-1( ), Phase-2( ), Phase-3( )를 예제를 통하여 고찰한다. 2.5 절의 예제에서 나타내는 바와 같이, 데이터웨어하우스에는 기본릴레이션 pos, items가 저장되어 있고 이들로부터 실체뷰 sales가 정의되어 있다고 하자.

pos(알고리즘에서는 일반화의 의미로 R_i로 표현) 릴레이션에는 다음과 같은 데이터가 저장되어 있다.

● **pos**

storeID	itemID	time	qty	price
1	A	16시 00분	20	2000원
1	B	14시 00분	10	2000원
1	C	09시 30분	5	2500원
2	A	11시 00분	5	500원
3	E	9시 00분	1000	5000원

items 릴레이션(알고리즘에서는 일반화의 의미로 R_i로 표현)에는 다음과 같은 데이터가 저장되어 있다.

● **items**

itemID	name	category	cost
A	드라이버	공구	100원
B	톱	공구	200원
C	망치	공구	500원
D	나사	재료	10원
E	못	재료	5원

실체뷰 sales(알고리즘에서는 일반화의 의미로 MV_i로 표현)에는 다음과 같은 데이터가 저장되어 있다.

● **sales**

storeID	category	t_count	e_time	t_qty
1	공구	3	09시 30분	35
2	공구	1	11시 00분	5
3	재료	1	09시 00분	1000

pos에 대한 삽입릴레이션 pos_ins(알고리즘에서는 일반화의 의미로 $\triangle I_R_i$로 표현)에는 다음과 같은 데이터가 저장되어 있다.

● **pos_ins**

storeID	itemID	time	qty	price
1	C	16시 30분	10	5000원
1	A	17시 00분	20	2000원

pos에 대한 삭제릴레이션 pos_del(알고리즘에서는 일반화의 의미로 $\triangle D_R_i$로 표현)에는 다음과 같은 데이터가 저장되어 있다. 이 릴레이션의 데이터는 반품 또는 교환 등의 의미로 해석할 수 있다.

● **pos_del**

storeID	itemID	time	qty	price
1	B	14시 00분	10	2000원
3	E	09시 00분	1000	5000원

Phase-1( ) 알고리즘의 처리과정 (1)에 의하여 I_pos_view(알고리즘에서는 일반화의 의미로 I_R_view로 표현)에는 다음과 같은 데이터가 저장

된다.

● I_pos_view

storeID	category	count	time	qty
1	공구	1	16시 30분	10
1	공구	1	17시 00분	20

Phase-1( ) 알고리즘의 처리과정 (2)에 의하여 D_pos_view(알고리즘에서는 일반화의 의미로 ⊿D_R_view로 표현)에는 다음과 같은 데이터가 저장된다.

● D_pos_view

storeID	category	count	time	qty
1	공구	-1	14시 00분	-10
3	재료	-1	09시 00분	-1000

Phase-1( ) 알고리즘의 (3)의 처리과정 중 「I_pos_view ∪ D_pos_view」에는 다음과 같은 데이터가 저장되어 있다.

storeID	category	count	time	qty
1	공구	1	16시 30분	10
1	공구	1	17시 00분	20
1	공구	-1	14시 00분	-10
3	재료	-1	09시 00분	-1000

Phase-1( ) 알고리즘의 (3)과 (4)의 처리결과로, 실체뷰의 델타릴레이션 ⊿sales(알고리즘에서는 일반화의 의미로 ΔMV_i로 표현)에는 다음과 같은 데이터가 저장된다.

● ⊿sales

store ID	category	D_count	D_E_time	D_qty	DEL
1	공구	1	14시 00분	20	"no"
3	재료	-1	09시 00분	-1000	"no"

Phase-2( ) 알고리즘에 의해서 ⊿sales의 데이터는 다음과 같이 변화된다.

● ⊿sales

store ID	category	D_ count	D_E_ time	D_qty	DEL
1	공구	1	14시 00분	20	"yes"
3	재료	-1	09시 00분	-1000	"yes"

Phase-3( ) 알고리즘에 의해서 ⊿sales를 실체뷰 sales에 적용하면 다음과 같이 갱신된다.

● sales

storeID	category	t_count	e_time	t_qty
1	공구	4	09시 30분	55
2	공구	1	11시 00분	5

Ⅴ. 성능평가 및 분석

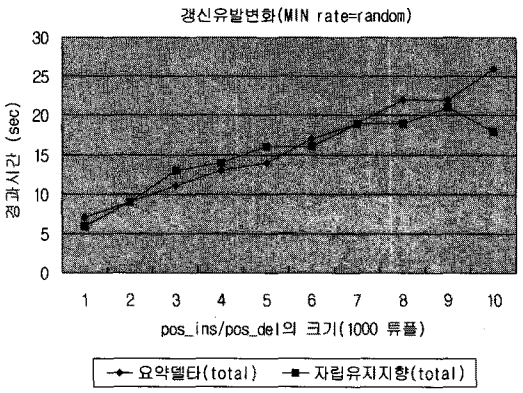
본 논문에서는 요약델타 알고리즘을 개선하여 자립유지지향 알고리즘을 제안하였다. 자립유지지향 알고리즘은 MIN/MAX 함수가 실체뷰 정의에 포함되고 기본릴레이션의 MIN/MAX 값의 변화가 빈번한 상황에 효율적으로 적용되어 실체뷰의 갱신시간을 단축시킬 수 있다.

자립유지지향 알고리즘의 성능을 평가하기 위하여 요약델타 알고리즘과 비교하였다. 성능평가를 위하여 비교대상의 각 알고리즘들을 실제로 구현하였다. 구현은 펜티엄Ⅲ dual CPU를 장착한 PC 서버상에서 윈도2000 기반의 오라클 9i버전과 응용프로그램 개발툴인 Pro-C/C++을 이용하여 이루어졌다. 테스트 데이터베이스 스키마는 2.5 절의 예제에서 고려했던 기본릴레이션 pos, items과 실체뷰 sales와 동일하다. pos 릴레이션은 1,000,000 튜플들을 포함하고 item 릴레이션은 1,000 튜플들을 포함한다. pos 테이블은 (storeID, itemID, time)을 기준으로 인덱싱되어 있고 실체뷰 sales도 group by 속성들을 기준으로 인덱싱되어 있다. 기본릴레이션 pos의 델타릴레이션들 pos_ins와 pos_del의 크기는 1,000 튜플에서 10,000 튜플까지 변한다. 기본릴레이션

pos의 델타릴레이션들의 유형은 다음과 같이 2가지로 분류된다.

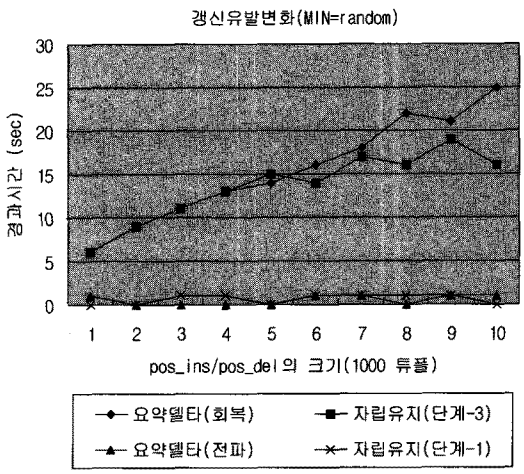
- 갱신유발변화(update-generating changes) 기본릴레이션의 임의의 튜플들의 속성값들을 변경한다. 특정 튜플의 속성값을 변경하는 과정은 그 튜플을 삭제한 후 변경된 값을 포함하는 새로운 튜플을 삽입하는 것과 같다. 따라서, 동일한 수의 삽입튜플과 삭제튜플을 포함하는 변경이다.
- 삽입유발변화(insertion-generating changes) 기본 릴레이션에 임의의 튜플들을 삽입하는 변경이다. 데이터웨어하우스의 기본릴레이션의 변화는 삽입유발변화가 일반적이다[Mumick,1997].

<그림 8>은 기본릴레이션 pos에 대한 갱신유발변화가 발생한 경우 실체뷰 sales의 갱신작업을 위하여 경과한 전체적인 시간(요약델타의 경우 전파단계 + 회복단계, 자립유지지향의 경우, 단계 1 + 단계 2 + 단계 3)을 비교하고 있다. pos_ins 및 pos_del 안의 MIN 값은 랜덤함수에 의해서 결정된다. 실험결과, 실체뷰 sales의 MIN 값보다 작을 확률이 1% 미만이었다. 자립유지지향 알고리즘은 요약델타 알고리즘에 비하여 단계 2를 더 포함하고 있지만 단계 3의 경과시간이 단축되므로 전체적인 경과시간은 줄어든다.



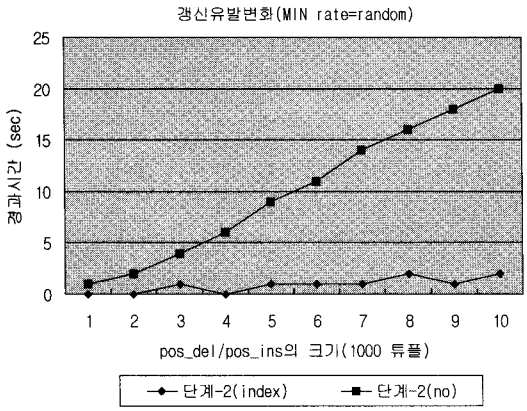
<그림 8> 전체적인 갱신작업의 경과시간 비교

<그림 9>는 요약델타 알고리즘과 자립유지지향 알고리즘에서 전파단계와 단계 1, 회복단계와 단계 3은 비슷한 처리과정을 수행한다. 요약델타 알고리즘과 자립유지지향 알고리즘에 대하여 갱신유발변화가 발생한 경우 각각의 전파단계와 단계 1, 회복단계와 단계 3을 완성하기 위하여 경과한 시간을 비교하고 있다. 전파단계와 단계 1은 비슷하고 회복단계와 단계 3의 비교에서는 자립유지지향 알고리즘이 우수함을 알 수 있다. 실제로 회복단계 및 단계 3에서의 경과시간이 중요한데, 단계 - 1이나 단계 2에서는 실체뷰를 잠금상태로 만들지 않으므로 사용자의 접근을 허락하지만 단계 3에서는 실체뷰를 잠금상태로 설정하므로 단계 3을 처리하기 위한 시간은 가능한한 짧아야 한다.



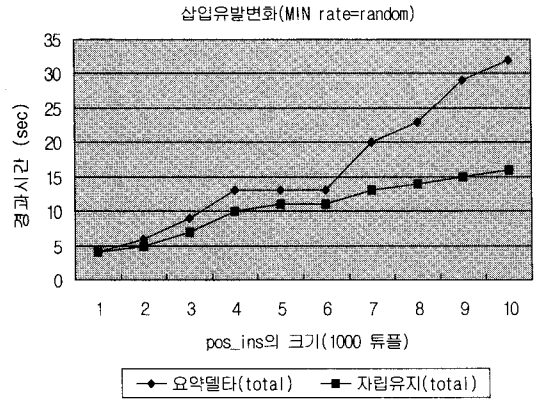
<그림 9> 단계 1, 단계 3의 경과시간 비교

<그림 10>은 자립유지지향 알고리즘에서 단계 2를 처리하기 위하여 D_sales와 pos_del, pos_ins에 인덱스를 설정한 경우와 그렇지 않은 경우의 경과시간을 보여주고 있다. 「단계 2(index)」는 인덱스를 생성하는 시간과 삭제여부를 판별하는 시간을 포함한다. <그림 8>에서의 단계 - 2의 처리시간은 「단계 2(index)」를 가정하고 있다.



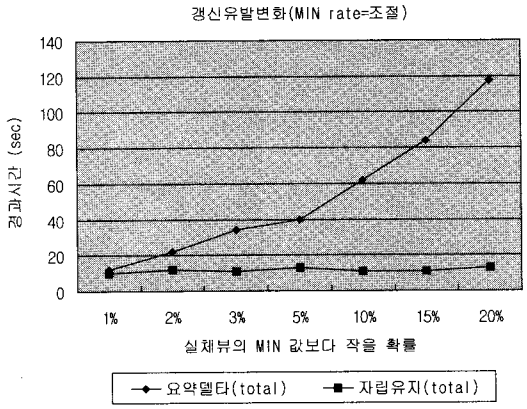
<그림 10> 단계 2의 경과시간 비교

<그림 11>는 기본릴레이션 pos에 대한 삽입 유발변화가 발생한 경우 실체뷰 sales의 갱신작업을 위하여 경과한 전체적인 시간(요약델타의 경우 전파단계 + 회복단계, 자립유지지향의 경우, 단계 1 + 단계 2 + 단계 3)을 비교하고 있다. 자립유지지향 알고리즘이 타 알고리즘에 비하여 우수함을 알 수 있다.



<그림 11> 삽입유발변화의 전체 경과시간 비교

<그림 12>는 기본릴레이션 pos에 대한 갱신 유발변화가 발생한 경우 실체뷰 sales의 갱신작업을 위하여 경과한 전체적인 시간(요약델타의 경우 전파단계 + 회복단계, 자립유지지향의 경우, 단계 1 + 단계 2 + 단계 3)을 비교하고 있다. 그러나, pos_ins 및 pos_del 안의 MIN값을 인위적으로 조절하여 갱신작업의 처리시간을 비교하고 있다. pos_ins 및 pos_del 안의 MIN값이 실체뷰 sales의 MIN 값보다 작을 확률이 높아질수록 자립유지지향 알고리즘은 뛰어난 성능을 보여주고 있다.



<그림 12> MIN 값의 인위적 조절에 따른 경과시간 비교

Ⅵ. 결 론

데이터웨어하우스내의 기본릴레이션에서 자주 사용되는 종류의 데이터에 대해 이를 정리하고 요약한 실체뷰는 실제 질의처리 시에 질의응답 속도를 줄일 수 있어서 사용자의 정보요구 만족도를 높이는 주요한 수단이다. 데이터웨어하우스에서 유지하는 실체뷰들의 갯수가 많을수록 사용자의 정보요구에 대한 만족도는 높아지지만 전체적인 실체뷰 갱신시간의 제약으로 인하여 유지할 수 있는 실체뷰의 개수에는 한계가 있다. 전체적인 실체뷰 갱신작업시간의 제약하에서, 각 실체뷰의 갱신시간이 짧을수록 보다 많은 수의 실체뷰들을 데이터웨어하우스 내에 포함할 수 있다.

본 논문에서는 MIN/MAX 집단함수가 포함된 실체뷰의 갱신시간을 단축시키기 위하여 보

다 많은 집단함수들을 자립유지에 의하여 처리할 수 있는 실체뷰 관리 알고리즘을 제안하였다. 기존의 요약델타 알고리즘은 실체뷰의 델타릴레이션을 계산할 때 MIN/MAX 값의 삽입, 삭제 여부를 구별하지 않음으로써 MIN/MAX 값의 갱신을 위한 재계산 확률이 높았었다. 그러나, 자립유지지향 알고리즘은 실체뷰의 델타릴레이션을 계산할 때 MIN/MAX 값의 삽입, 삭제 여부를 구별함으로써, 실질적인 갱신작업시 삭제된 MIN/MAX 값에 대해서만 재계산을 수행하므로 갱신작업시간이 단축된다. 실험 결과, MIN / MAX 값의 삽입, 삭제 여부를 구별하는 과정도 인덱스를 설정하면 많은 시간이 소요되지 않음을 알 수 있었고, 실체뷰의 전체적인 갱신시간이 단축되었음을 알 수 있었다. MIN/MAX 값의 삽입, 삭제 여부를 구별하는 시간이 길어진다 할 지라도 실체뷰를 잠금상태로 만들지 않으므로 데이터웨어하우스의 성능에는 큰 영향을 끼치지 않는다. 특히, 기본릴레이션의 델타릴레이션 안에 있는 MIN 값이 실체뷰내의 MIN 값보다 작을 확률이 높아질수록 자립유지지향 알고리즘은 뛰어난 성능을 보여줌을 알 수 있었다.

〈참 고 문 헌〉

[1] 김민정, 정연돈, 박웅제, 김명호, "데이터 큐브에서 세분화된 뷰실체화 기법," 정보과학회논문지, Vol. 28, No. 4, 2001, pp. 587-595.

[2] 양우석, 김명호, "OLAP 환경에서 다중점 MAX/MIN 질의의 효율적인 처리기법," 정보과학회논문지, Vol. 27, No. 1, 2000, pp. 13-21.

[3] 윤원식, 신동천, "데이타웨어하우스 환경에서 조인비용을 기반으로 한 실체뷰 선택알고리즘," Vol. 28, No. 1, 03, pp. 31-41, 2001.

[4] 이기용, 김명호, "데이타웨어하우스에서 효과적인 점진적 뷰 관리," 정보과학회논문지, Vol. 27, No. 02, 2000, pp. 175-184.

[5] 정희정, 김동욱, 김종수, 이윤준, 김명호, "OLAP에서 MAX-of-SUM 질의의 효율적인 처리기법," 정보과학회논문지, Vol. 27, No. 2, 2000, pp. 165-174.

[6] 조재희, "데이터웨어하우징 기술을 이용한 DB 마케팅 전략에 관한 연구," *정보기술과 데이터베이스 저널*, 제6권, 제1호, 1998, pp. 104-113.

[7] D. Agrawal, A. El Abbadi, A. Singh, T. Yurek, "Efficient View Maintenence at Data Warehouse," *In proceedings of ACM SIGMOD Conference*, 1997, pp. 417-427.

[8] L.S.Colby, A.Kawaguchi, D.F.Lieuwen, I.S. Mumick, "Supporting Multiple View Maintenance Policies," *In proceedings of ACM SIGMOD Conference*, 1997, pp. 405-416.

[9] Y. Kotidis, N. Roussopoulos, "DynaMat: A Dynamic View Management System for Data Warehouses," *In proceedings of ACM SIGMOD Conference*, 1999, pp. 371-382.

[10] Y. Kotidis, N. Roussopoulos,, "A Case for Dynamic View Management," *ACM Transactions on Database Systems*, Vol. 26, No. 4, December 2001, pp. 388-423.

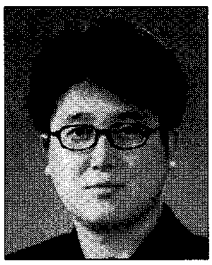
[11] I.S.Mumick, D.Quass, B.S.Mumick, "Maintenance of Data Cubes and Summary Tables in a Warehouse," *In proceedings of ACM SIGMOD Conference*, 1997, pp. 100-111.

[12] D. Theodoratos, M. Mouzeghoub, "A General Framework for the View Selection Problem for Data Warehouse Design and Evolution," *In proceedings of ACM SIGMOD*

Conference, 2000, pp. 1-8.

[13] Y. Zhuge, H. Garcia-Molina, J. Hammer, J. Wodim, "View Maintenance in a Warehousing Environment," *In proceedings of ACM SIGMOD Conference*, 1995, pp. 316-327.

◆ 저자소개 ◆



김근형 (Kim, Keun-Hyung)
서강대학교 컴퓨터학과를 졸업하였고 동대학원 컴퓨터학과에서 데이터베이스 전공으로 박사학위를 취득하였다. 현대전자 소프트웨어연구소에서 연구원을 지냈으며 현재 제주대학교 경영정보학과 조교수로 재직하고 있다. 주요 관심분야는 데이터베이스, 데이터웨어하우징, 데이터마이닝, e-비즈니스 분야이다.

◈ 이 논문은 2002년 9월 15일 접수하여 1차 수정을 거쳐 2003년 7월 21일 게재확정되었습니다.