삽입/삭제연산의 구분을 통한 실체뷰 갱신시간의 단축정책

김 근 형† · 이 동 철††

요 약

데이터웨어하우스내에서 실체뷰의 개수가 많을수록 사용자의 질의요구를 실체뷰내에서 처리할 수 있는 확률이 높아지므로 신속한 응답이 가능하다. 그러나, 실체뷰가 많아지면 전체적인 실체뷰 갱신시간이 길어진다. 실체뷰 갱신작업 동안에는 데이터웨어하우스로의 접근이 통제되기 때문에 사용자의 질의요구를 처리할 수 없다. 따라서, 실체뷰의 갱신시간을 단축시킬 필요가 있다. 본 논문에서는 삽입/삭제연산을 구분하여 관리함으로써 MIN/MAX 함수를 포함하는 실체뷰의 갱신시간을 단축시킬 수 있는 알고리즘을 제안한다. 실험결과, 삽입/삭제 구분을 위한 오버헤드를 감안하더라도 제안한 알고리즘의 성능이 기존 기법에 비하여 우수함을 알 수 있었다.

Policy for reducing update duration of materialized views by distinguishing Insertion/Deletion operations

Keun Hyunng Kim† · DongCheol Lee††

ABSTRACT

More views in data warehouse, can respond to the users more rapidly because the user's requests may be processed by utilizing only the materialized views with higher probabilities rather than accessing base relations. But, more views cause longer update duration of all the materialized views. During the update time, queries of users can not be processed because accesses to the datawarehouse are blocked. Thus, the update durations of the materialized views are necessary to be reduced. In this paper, we propose algorithm for reducing the update duration of the materialized views owing to managing by distinguishing insertion/deletion operations. Though there might be overheads for distinguishing insertion/deletion, we concluded that the proposed algorithm could be excellent than others.

키워드 : 데이터웨어하우스(Data warehouse), 실체뷰(Materialized view), 갱신시간(Update duration), 삽입(Insertion), 삭제(Deletion)

1. 서 론

데이터웨어하우징이란, 조직내의 각 데이터 저장소들로부터 추출한 데이터를 중앙의 단일 데이터베이스에 통합하여 저장한 '데이터웨어하우스(data warehouse)'를 관리하는 기법을 의미한다[12].

OLAP 시스템은 이러한 데이터웨어하우스에 저장된 방대한 양의 데이터에 대하여, 의사결정과정에 도움을 주는 빠르고 직접적인 자료분석 작업을 수행한다[12].

데이터웨어하우스에서 지원하고자 하는 의사결정지원 질의(decision support query)는 개별적인 레코드 정보의 검색이 아닌, 레코드 집합의 전체적인 정보와 경향분석등을 수행하므로 처리시 많은 시간이 소요된다. 그러나, OLAP 시스템은 의사결정 과정에 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 질의처리시에는 저장된 정보를 활용하여 질의 응답 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[8]. 사용자의 정보요구가 기본릴레이션 대신 실체뷰에 의하여 처리된다면, 요구사항을 새롭게 계산할 필요없이 미리 계산한 결과값을 보여주는 방식이 될 것이므로 보다 신속한 응답을 제공할 수 있다.

† 정 회 원 : 제주대학교 경영정보학과 교수
†† 종신회원 : 제주대학교 경영정보학과 교수
논문접수 : 2002년 9월 3일, 심사완료 : 2003년 5월 6일

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. 데이터웨어하우스내에 보다 많은 실체뷰들을 유지하기 위해서는 각 실체뷰들의 갱신시간을 단축시킬 방안이 필요하다.

점진적 뷰관리 정책은 실체뷰 갱신시간을 단축시킬 수 있는 방법중의 하나로, 기본릴레이션의 변화가 생길 때 뷰를 전부 새로 계산하지 않고 뷰의 변경될 부분만을 계산하는 방법이다[2, 3, 8]. 실체뷰중에서 집단함수를 포함하는 실체뷰는 갱신작업에 많은 시간이 소요된다. SQL에서는 표준 집단함수로서 SUM, AVG, COUNT, MAX, MIN의 5가지를 지정하고 있다. SUM, AVG, COUNT 함수의 실체뷰 갱신은 점진적 뷰관리 정책으로 처리할 수 있지만, MIN/MAX의 처리는 기본릴레이션에 접근해야 한다고 알려져 있다[1]. 따라서, MIN/MAX 집단함수를 포함하는 실체뷰의 경우, 기본릴레이션의 MIN/MAX 값의 변경 정도가 많아지면 실체뷰의 갱신시간은 길어진다. 현실적인 예로 날짜와 부서를 차원으로 하면서 영업사원의 최저 실적과 최고 실적을 보유하는 실체뷰의 경우, 기본릴레이션에서 영업사원 실적의 MIN/MAX 값은 매일 변경될 필요가 생기고 실체뷰의 갱신부담은 커져서 시스템 성능이 떨어질 것이다.

그러나, 기본릴레이션의 삽입연산으로 인한 실체뷰의 MIN/MAX 값의 갱신은 점진적 뷰관리 정책을 적용하여 처리할 수 있다. 또한, 데이터웨어하우스에서는 일반적으로 삽입연산이 주로 일어나므로 기본릴레이션으로의 삽입/삭제 연산을 구분하여 실체뷰 갱신정책에 반영하면 실체뷰의 MIN/MAX 값의 갱신시간을 단축시킬 수 있다. 기본릴레이션의 갱신연산도 삭제연산과 삽입연산의 연속적인 과정으로 간주할 수 있으므로 기본릴레이션에 대한 모든 연산은 삽입/삭제연산으로 간주할 수 있다.

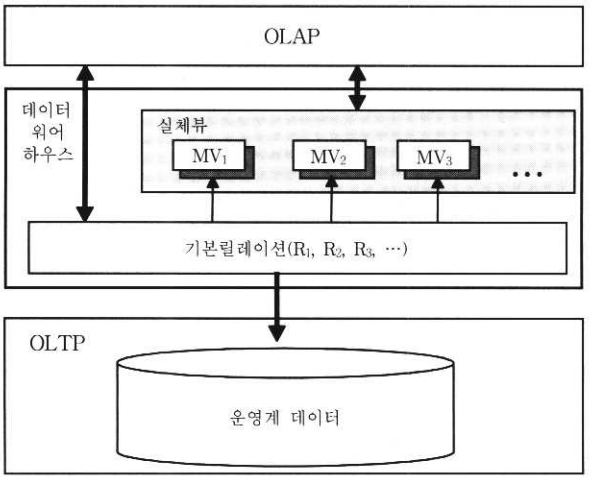
본 논문에서는 기본릴레이션에 대한 연산을 삽입/삭제 연산으로 구분하여 관리함으로써, 실체뷰의 MIN/MAX값 갱신시 기본릴레이션으로의 접근횟수를 최소화시킬 수 있고 결과적으로 실체뷰 갱신시간을 단축시킬 수 있는 알고리즘을 제안한다. 실험결과, 삽입/삭제 구분을 위한 오버헤드를 감안하더라도 제안한 알고리즘의 성능이 기존 기법에 비하여 우수함을 알 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 데이터웨어하우징에서의 뷰관리 정책과 관련된 기존 연구들을 고찰한다. 3장에서 실체뷰의 갱신작업 시간을 단축시킬 수 있는 알고리즘들을 제안한다. 4장에서 제안한 알고리즘의 예제를 제시하고 5장에서 제안한 알고리즘의 성능을 평가 분석하며 6장에서 결론을 맺는다.

2. 선행 연구

2.1 데이터웨어하우스 모델

(그림 1)에서 볼 수 있는 것 처럼, 일반적으로 데이터웨어하우스에는 외부의 운영계 데이터(operational data) 또는 데이터웨어하우스 내에 저장된 데이터(기본릴레이션들, R_1, R_2, R_3, …)에 의해 정의된 실체뷰들(MV_1, MV_2, MV_3, …)이 저장된다. 흔히 데이터웨어하우스에는 외부의 데이터들이 가공 및 정제의 과정을 거쳐 "사실테이블(fact table)"이나 "차원테이블(dimension table)"로 저장된다. 이 때, 사실테이블과 차원테이블 들로부터 "요약테이블(summary table)"이 정의되어 저장되기도 하는데 이 경우 요약테이블은 사실테이블과 차원테이블을 기본릴레이션(Base Relation)으로 하는 실체뷰라고 볼 수 있다. 데이터웨어하우징은 데이터웨어하우스내의 기본릴레이션이나 실체뷰등을 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[8]. 기본릴레이션의 변경을 즉각적으로 실체뷰에 반영하면 시스템의 부담으로 사용



(그림 1) 데이터웨어하우스 모델

자 요구에 대한 응답시간이 느려지므로 하루나 일주일의 간격을 두어 주기적으로 실체뷰의 갱신작업을 수행하는 스냅샷 갱신기법이 일반적이다[6].

2.2 점진적 뷰관리 기법

어떤 실체뷰가 기본릴레이션들에 의해 정의되어 있을 때 기본릴레이션의 변화를 뷰에 반영하는 방법은 크게 두 가지로 나뉜다. 첫째는 재계산(recomputation) 방법이다. 이 방법은 각각의 기본릴레이션들을 먼저 갱신하고 갱신된 기본릴레이션들로부터 실체뷰들을 새로 계산한다. 기본릴레이션의 변화가 적을 경우에 이렇게 실체뷰 전체를 다시 계산하는 일은 매우 비효율적이다. 둘째는 점진적 관리(incremental maintenance)이다. 이 방법은 기본릴레이션들과 이들 중 변경된 튜플들만으로 된 차이릴레이션(delta relation)들로부터 변경된 뷰의 내용만을 계산한 뒤, 이 계산된 내용을 뷰에 반영하는 방법이다. 기본릴레이션의 변화가 적은 경우, 이 방법은 재계산에 비해 효율적이다. 실체뷰는 일반적으로 [정의 1]과 같이 정의되어진다[8].

[정의 1] 실체뷰 모델

n개의 기본릴레이션 $R_1, R_2, \cdots, R_n$에 대해서 실체뷰 MV는 다음과 같이 정의된다.

$$MV = \pi_A \sigma_P (R_1 \bowtie R_2 \bowtie R_3 \bowtie, \cdots, \bowtie R_n)$$

여기서 A는 속성들의 집합이며 P는 부울식(boolean expression)을 의미한다. □

점진적 관리 관련된 대부분의 기존 연구들은 주로 기본릴레이션의 효율적인 죠인단계를 연구대상으로 하였으므로 SPJ(Selection-Projection-Join) 식으로 표현되는 뷰만을 주로 고려하였지만 본 논문에서는 집단함수나 Group By와 같은 식이 포함되어 있는 (그림 2)과 같은 실체뷰로 확장하여 논의한다. 또한, 논의의 범위를 좁히기 위하여 다음과 같이 두 기본릴레이션의 죠인으로 이루어진 실체뷰를 다룬다.

$$MV_i = \pi_A \sigma_P (R_i \bowtie R_j)$$

(그림 2)는 기본릴레이션 R_i와 R_j에 대하여 Group By 및 집단함수가 포함된 실체뷰 MV_i를 나타내고 있다. <표 1>은 재계산 방법과 점진적 관리에 의한 실체뷰의 갱신작업 과정을 나타내고 있다. 점진적 관리방식의 기존 연구들은 두개 이상의 기본릴레이션이 변경될 때, 실체뷰를 갱신하기 위한 효율적인 표준식들을 제안하고 있으나 본 논문에서는 하나의 기본릴레이션의 변경만을 가정하겠다.

$R_i (a_{i1}, a_{i2}, a_{i3}, \cdots, a_{ip})$
$R_j (b_{j1}, b_{j2}, b_{j3}, \cdots, b_{jq})$

```
CREATE VIEW MVi (m1, m2, m3, m4, m5, ···, mr)
AS   SELECT ai1, bj2,
            COUNT(*) AS m3,
            MIN(ai3) AS m4,
            SUM(ai4) AS m5,
            ...
     FROM Ri, Rj
     WHERE Ri.ai1 = Rj.bj1
     GROUP BY ai1, bi3
```

(그림 2) Group By 및 집단함수가 포함된 실체뷰

<표 1>은 기본릴레이션 R_i의 변화가 생겼을 때 실체뷰 MV_i를 갱신하는 과정이다. <표 1>에서 ΔMV_i는 MV_i의 차이릴레이션이고 ΔR_i는 R_i의 차이릴레이션을 의미한다.

〈표 1〉 재계산과 점진적 관리방법

	실체뷰 갱신과정
재계산	[step1] : $R_i' = R_i \cup \Delta R_i$ [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] : $MV_i' = \Delta MV_i \cup MV_i$

2.3 자립유지(self-maintainable) 집단함수

집단함수(Aggregation Function)는 세 부류 즉, 분배 집단함수(distributive), 대수 집단함수(algebraic), 홀리스틱 집단함수(holistic)로 구분된다[1]. 분배 집단함수는 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 분배 집단함수이다. 예를 들면, COUNT는 분할된 영역의 COUNT 결과값들을 다시 합하여 전체 영역의 최종적인 결과를 유도할 수 있다. 대수 집단함수는 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 2] 자립유지 집단함수

실체뷰에 포함된 집단함수의 새로운 결과값을 구하는 경우, 기본릴레이션의 변경부분과 집단함수의 이전 결과값에 의해서 타당하게 계산될 수 있다면, 그 집단함수는 자립유지가 가능하다[1]. □

분배 집단함수는 삽입 및 삭제갱신에 대해서 일반적으로 자립유지가 가능하지만 MIN/MAX의 경우는 삽입갱신일 때만 자립유지가 가능하고 삭제갱신일 경우는 자립유지가 불가능하다. 왜냐하면, 실체뷰의 MIN/MAX 값이 기본릴레이션에서 삭제된다면 다시 기본 릴레이션을 접근하여 새로운

MIN/MAX 값을 계산해야 되기 때문이다.

2.4 기존 관련연구들의 고찰 및 분석

실체뷰와 관련된 연구는 실체뷰 선택과 실체뷰 관리의 분야로 나눌 수 있다. 실체뷰 선택은 사용자의 신속한 응답 요구를 만족시키기 위하여 어떤 뷰들을 실체뷰로 선택해야 할 것인가의 이슈를 다룬다[9]. 실체뷰 관리는 기본릴레이션의 변경에 따라 이를 보다 신속하고 효율적으로 실체뷰에 반영하기 위한 이슈를 다룬다. [9]는 조인 비용을 기반으로 한 실체뷰 선택 알고리즘을 제안하고 있고 [7]은 다양한 목적하의 뷰선택 문제를 일반화시킬 수 있는 프레임웍을 제안하였다. [10]은 데이터큐브의 일부를 나타내는 세분화된 뷰의 표현 및 선택 문제를 다루고 있다. 데이터웨어하우징에서 실체뷰의 갱신작업을 효율적으로 수행하여 사용자의 정보요구에 대한 신속한 응답을 제공하려는 연구들이 있었는데 [2, 3, 8]은 운영계 데이터 또는 기본렐레이션의 변화가 생길 때 뷰를 전부 새로 계산하지 않고 뷰의 변경될 부분만을 계산하여 신속한 뷰갱신작업을 처리하기 위한 점진적 뷰관리 기법들을 제안하고 있다. [6]은 OLTP 환경에서의 뷰관리 정책을 다루고 있는데, 지연갱신, 즉각갱신, 스냅샷 갱신의 장단점을 고려하여 효율적으로 혼합 적용할 수 있는 방안을 제안하고 있다. [4, 5]는 데이터웨어하우징 환경에서 임의의 사용자 질의를 특정 척도를 기준으로 평가하여 자동적으로 실체뷰화시키는 정책을 제안하고 있다. [1]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 알고리즘을 제안하고 있다. 그러나, 실체뷰의 MIN/MAX값 갱신은 점진적 방식이 아닌 재계산 방법에 의하여 처리하고 있다. [11, 12]는 실체뷰가 아닌 기본릴레이션에서 MIN/MAX 집단함수를 신속하게 처리하기 위하여 효율적인 인덱싱등의 기법을 사용하는 방안을 제안하고 있다.

본 논문에서는 [1]의 알고리즘을 개선하여 실체뷰의 MIN/MAX값 갱신시 일부는 점진적 방식에 의한 처리를 가능하게 한다. 즉, 기본릴레이션의 삽입연산으로 인한 실체뷰의 MIN/MAX 값의 갱신은 점진적 뷰관리 정책을 적용하여 처리할 수 있다. 반면, 기본릴레이션의 삭제연산으로 인한 실체뷰의 MIN/MAX 값의 갱신은 재계산 방법을 이용하여 처리한다. 이를 위하여 기본릴레이션에 대한 연산을 삽입/삭제 연산으로 구분하여 관리해야 한다. 삽입/삭제 연산으로 구분하기 위한 오버헤드는 실체뷰 갱신시간의 단축이라는 결과로 상쇄될 수 있다.

3. 새로운 알고리즘

3.1 개 요

본 논문에서는 실체뷰 갱신시간을 줄이기 위하여 실체뷰 갱신작업을 2 단계로 나누어서 처리한다. 첫 번째 단계인 선계산(precomputation) 단계에서는 신속한 갱신작업을 하기 위하여 미리 계산을 하는 과정인데 실체뷰를 잠금상태로 만들지 않으면서 처리 가능하다. 선계산 단계에서는 두 가지 작업이 이루어진다. 하나는, 기본릴레이션(R_i)의 변화부분인 차이릴레이션(ΔR_i)에 대하여 실체뷰(MV_i)의 정의를 적용하여 실체뷰의 차이릴레이션(ΔMV_i)을 만드는 과정이다. 다른 하나는, 실체뷰의 차이릴레이션의 MIN/MAX 값이 삽입된 값인지 삭제된 값인지의 여부를 판별하는 과정이다. 삽입된 값이면 자립유지가 가능하여 기본릴레이션에 접근할 필요가 없고 삭제된 값이면 기본릴레이션에 접근하여 새로운 MIN/MAX 값을 계산해야 하기 때문이다. [1]에서는 이러한 부분을 고려하지 않기 때문에 기본릴레이션의 MIN/MAX 값의 변경이 빈번할 경우 실체뷰의 MIN/MAX 값을 구하기 위하여 기본릴레이션에 접근하는 횟수가 많아지게 되어 갱신시간이 느려지게 된다. 두 번째 단계인 갱신단계(update phase)에서는 선계산 단계동안에 생성된 실체뷰의 차이릴레이션을 기반으로 실체뷰를 갱신하는 과정이다.

3.2 선계산 단계의 알고리즘

선계산 단계는 실체뷰를 잠금상태로 만들지 않고 처리되므로 선계산 단계동안 사용자들로부터 실체뷰로의 접근들은 지속될 수 있다. 따라서, 이 단계에서는 시간이 걸리더라도 실질적인 갱신단계의 작업시간을 단축킬 수 있는 보다 많은 계산을 해 두는 것이 바람직하다.

실체뷰의 차이릴레이션을 만드는 과정은 차이릴레이션 생성을 위한 준비단계와 차이릴레이션의 생성단계로 나누어 볼 수 있다. 기본릴레이션의 차이릴레이션은 삽입릴레이션과 삭제릴레이션으로 나누어지는데, 삽입릴레이션은 기본릴레이션에 삽입된 튜플들로 이루어지고 삭제릴레이션은 기본릴레이션에서 삭제된 튜플들로 이루어진다. 준비단계에서는 삽입릴레이션과 삭제릴레이션 각각에 대하여 실체뷰 정의의 셀렉션 조건(selections conditions)과 프로젝션 조건(projection condition)만을 적용한 후 각각 삽입뷰릴레이션과 삭제뷰릴레이션을 생성한다. 그런 다음, 삽입뷰릴레이션과 삭제뷰릴레이션에 유니온(union)연산을 적용하여 혼합뷰릴레이션을 생성한다. 프로젝션 조건에서는 실체뷰 정의에 나타나는 Group By 속성들과 집단함수 속성들을 모두 포함한다.

준비단계에서 일시적으로 생성되는 삽입뷰릴레이션 또는 삭제뷰릴레이션은 Group By 연산이나 집단함수 연산이 적

용되지 않는다. 집단함수 속성의 값들은 <표 2>에서 나타내는것 처럼 유도된다. 예를 들면, sum(A * B)와 같은 집단함수 속성의 경우, 삽입뷰릴레이션에서는 (A * B)가, 삭제뷰릴레이션에서는 -(A * B)가 그 값으로 대체된다.

〈표 2〉 집단함수 속성들의 대체값

	삽입릴레이션	삭제릴레이션
COUNT(*)	1	-1
COUNT(expr)	if expr = null then 0 else 1	if expr = null then 0 else -1
SUM(expr)	expr	-expr
MIN(expr)	expr	expr
MAX(expr)	expr	expr

(그림 3)의 PreComputation() 알고리즘은 선계산 단계를 처리하는 알고리즘인데, ΔI_R은 기본릴레이션 R의 삽입릴레이션, ΔD_R은 기본릴레이션 R의 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, PreComputation() 알고리즘에서 ①, ②, ③ 과정을 통하여, 삽입릴레이션 ΔI_R_i와 삭제릴레이션 ΔD_R_i로부터 실체뷰 MV_i의 차이릴레이션 ΔMV_i를 구할 수 있다.

```
Algorithm 1 : PreComputation ( )
input : ΔI_Ri, ΔD_Ri, Rj, SP_Func, Group_Func
output : ΔMVi
variable : I_R_view, D_R_view
Begin
① I_R_view ← SP_Func (ΔI_Ri ⋈ Rj)
② D_R_view ← SP_Func (ΔD_Ri ⋈ Rj)
③ ΔMVi ← Group_Func (I_R_view ∪ D_R_view)
④ add ΔMVi to attribute DEL of which value is set to "no"
⑤ For each tuple δt in ΔMVi
    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 3) 선계산 단계의 알고리즘

선계산 단계의 두 번째 작업은 실체뷰의 차이릴레이션의 MIN/MAX 값이 삽입된 값인지 삭제된 값인지의 여부를 판별하는 과정이다. 이 과정은 PreComputation() 알고리즘의 ⑤에서 처리된다. ΔMV_i와 D_R_view를 Group By 속성들을 기준으로 조인하여, ΔMV_i의 MIN/MAX 속성값과 D_R_view의 MIN/MAX 속성값이 같으면 ΔMV_i의 DEL 속성값을 "yes"로 변경한다. 이것은 ΔMV_i의 MIN/MAX 값이 삽입릴레이션에서 생성된 것일 경우 자립유지에 의하여 실체뷰의 MIN/MAX 값을 갱신하기 위한 선계산 작업이다. 본 알고리즘에 대한 예제가 4장에서 제시된다.

3.3 갱신 단계

이 단계에서는 선계산 단계동안에 생성된 실체뷰의 차이릴레이션 ΔMV_i를 기반으로 실체뷰 MV_i를 갱신하는 과정이다. (그림 4)는 갱신단계를 위한 알고리즘 **update_MV$_i$**()을 나타내고 있다.

ΔMV_i에 있는 튜플 δt와 MV_i의 튜플 t는 Group By 연산이 적용된 튜플이다. 갱신단계 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. R 단계 알고리즘의 ①의 단계를 통하여 ΔMV_i에는 존재하고 MV_i에는 존재하지 않는 튜플은 삽입된다. 즉, ΔMV_i에 있는 임의의 튜플 δt에 대하여 실체뷰의 Group By 속성값을 기준으로 부합하는 튜플이 MV_i에 존재하지 않는 경우 그 튜플 δt는 MV_i에 삽입된다. δt.COUNT(*)는 Gropu 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 의미이다. δt와 t가 group by 속성값을 기준으로 부합하고 「δt.COUNT(*) + t.COUNT(*) = 0」이면 ②의 단계를 통하여 MV_i의 튜플 t는 삭제되어야 한다. δt와 t가 Group By 속성값을 기준으로 부합하고 「δt.COUNT(*) + t.COUNT(*) > 0」이면 튜플 t의 속성값들을 δt의 속성값들을 기반으로 갱신해야 한다. 튜플 t의 속성값이 MIN/MAX 함수에 의하여 유도되었을 경우 δt의 MIN/MAX 값의 크기와 비교하고 삭제 태그를 체크하여 기본릴레이션으로부터의 재계산(recomputation) 여부를 결정한다. 재계산이 필요없을 경우 δt의 속성값들과 기존의 t의 속성값을 이용하여 새로운 t의 속성값들을 계산한다.

```
Algorithm 2 : update_MVi
Input : ΔMVi, MVi
Output : updated MV
Variable :
Begin
For each tuple δt in ΔMVi
   Let tuple t = tuple in MVi having the same values
         for its group-by attributes as δt
   If t is not found then
```

```
①   Insert tuple δt into MVi
  Else
     If δt.COUNT(*) + t.COUNT(*) = 0 then
②       Delete tuple t from MVi
     Else
       recompute = false
       for each aggregation function a(e) in the MVi
          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"))
                  recompute = true
       end if
          if (recompute) then
               Update tuple t by recomputing its aggregate
               functions from the base data from t's group.
          else
               if a is COUNT or SUM
                    t.a = t.a + δt.a
               else If a is MIN
                    t.a = MIN(t.a, δt.a)
               else If a is MAX
                    t.a = MAN(t.a, δt.a) }
            end if
       end for
     end if
  end if
end for
End
```

(그림 4) 갱신단계 알고리즘

4. 예 제

여기에서는 제안했던 두 개의 알고리즘들, PreComputation()과 update_MVi()을 예제를 통하여 고찰한다. 데이터웨어하우스에 기본릴레이션 pos, items가 저장되어 있고 이들로부터 실체뷰 sales가 정의되어 있다고 하자.

```
pos(storeID, itemID, date, qty, price)
items(itemID, name, category, cost)
sales(storeID, category, t_count, e_date, t_qty) =
          pos(storeID, itemID, date, qty, price)
          ⋈ items(itemID, name, category, cost)
```

아래의 질의는 실체뷰 sales의 구체적인 정의를 나타내고 있다.

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

pos 릴레이션에는 다음과 같은 데이터가 저장되어 있다.

pos

storeID	itemID	date	qty	price
1	A	8월 1일	20	2000원
1	B	6월 10일	10	2000원
1	C	4월 5일	5	2500원
2	A	5월 5일	5	500원
3	E	3월 10일	1000	5000원

item 릴레이션에는 다음과 같은 데이터가 저장되어 있다.

item

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 sales에는 다음과 같은 데이터가 저장되어 있다.

sales

storeID	category	t_count	e_date	t_qty
1	공구	3	4월 5일	35
2	공구	1	5월 5일	5
3	재료	1	3월 10일	1000

pos에 대한 삽입릴레이션 ⊿I_pos에는 다음과 같은 데이터가 저장되어 있다.

⊿I_pos

storeID	itemID	date	qty	price
1	C	8월 25일	10	5000원
1	A	8월 28일	20	2000원

pos에 대한 삭제릴레이션 ⊿D_pos에는 다음과 같은 데이터가 저장되어 있다.

⊿D_pos

storeID	itemID	date	qty	price
1	B	6월 10일	10	2000원
3	E	3월 10일	1000	5000원

삽입뷰릴레이션 I_pos_view에는 다음과 같은 질의가 적용된다.

```
SELECT storeID, category, 1 AS count,
       date AS e_date, qty AS t_qty
       FROM pos_ins, items
       WHERE pos_ins.itemID = items.itemID
```

따라서, I_pos_view에는 다음과 같은 데이터가 저장된다.

I_pos_view

storeID	category	count	e_date	t_qty
1	공구	1	8월 25일	10
1	공구	1	8월 28일	20

삭제뷰릴레이션 D_pos_view에는 다음과 같은 질의가 적용된다.

```
SELECT storeID, category, -1 AS count,
       date AS e_date, -qty AS t_qty
       FROM pos_ins, items
       WHERE pos_ins.itemID = items.itemID
```

따라서, 삭제뷰릴레이션 D_pos_view에는 다음과 같은 데이터가 저장된다.

D_pos_view

storeID	category	count	e_date	t_qty
1	공구	-1	6월 10일	-10
3	재료	-1	3월 10일	-1000

혼합뷰릴레이션 「I_pos_view ∪ D_pos_view」에는 다음과 같은 데이터가 저장되어 있다.

storeID	category	count	e_date	t_qty
1	공구	1	8월 25일	10
1	공구	1	8월 28일	20
1	공구	-1	6월 10일	-10
3	재료	-1	3월 10일	-1000

실체뷰의 차이릴레이션 ⊿sales가 생성되기 위한 질의가 다음과 같이 적용되고 DEL 속성값을 결정하기 위하여 알고리즘 PreComputation()의 ⑤단계가 수행된다.

```
CREATE VIEW ⊿sales (
SELECT storeID, category,
       SUM(count) AS t_count,
       MIN(date) AS e_date,
       SUM(quantity) AS t_qty
       "No" AS DEL
FROM( I_pos_view ∪ D_pos_view)
GROUP BY storeID, category
```

따라서, 실체뷰의 차이릴레이션 ⊿sales에는 다음과 같은 데이터가 저장된다.

⊿sales

storeID	category	t_count	e_date	t_qty	DEL
1	공구	1	6월 10일	210	"yes"
3	재료	-1	3월 10일	1000	"yes"

갱신단계 알고리즘에 의해서 ⊿sales를 실체뷰 sales에 적용하면 다음과 같이 갱신된다.

storeID	category	t_count	e_date	t_qty
1	공구	4	4월 5일	45
2	공구	1	5월 5일	5

5. 성능평가 및 분석

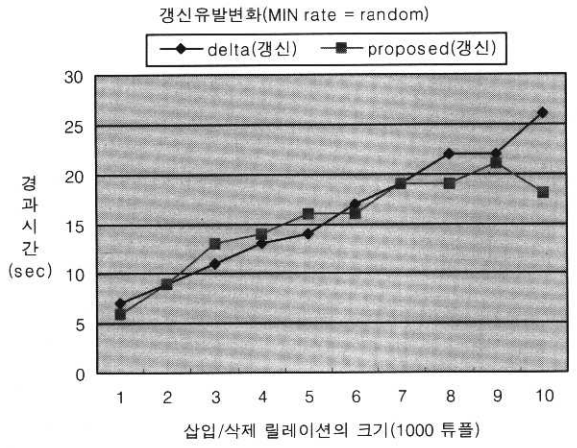
본 논문에서 제안한 알고리즘(즉, Pre-Computation() + update_MV$_i$(), 이후로 proposed 알고리즘으로 명명함)의 성능을 평가하기 위하여 [1]에서의 알고리즘(이후로 delta 알고리즘으로 명명함)과 비교하였다. 성능평가를 위하여 비교대상의 각 알고리즘들을 실제로 구현하여 시뮬레이션 실험을 하였다. 시뮬레이션을 위한 알고리즘 구현은 펜티엄Ⅲ dual CPU를 장착한 PC 서버상에서 윈도2000 기반의 오라클 9i DBMS와 응용 프로그램 개발툴인 Pro-C/C++을 이용하여 이루어졌다. 테스트 데이터베이스 스카마는 4장의 예제에서 고려했던 기본릴레이션 pos, items와 실체뷰 sales와 동일하고 테스트 데이터는 랜덤함수를 이용하여 생성하였다. pos 릴레이션은 1,000,000 튜플들을 포함하고 item 릴레이션은 1,000 튜플들을 포함한다. pos 테이블은 (storeID, itemID, date)를 기준으로 인덱싱되어 있고 실체뷰 sales도 group by 속성들을 기준으로 인덱싱되어 있다. 기본릴레이션 pos의 차이릴레이션들 ⊿I_pos와 ⊿D_pos의 크기는 1,000 튜플에서 10,000 튜플까지 변한다. 기본릴레이션 pos의 차이릴레이션들의 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.

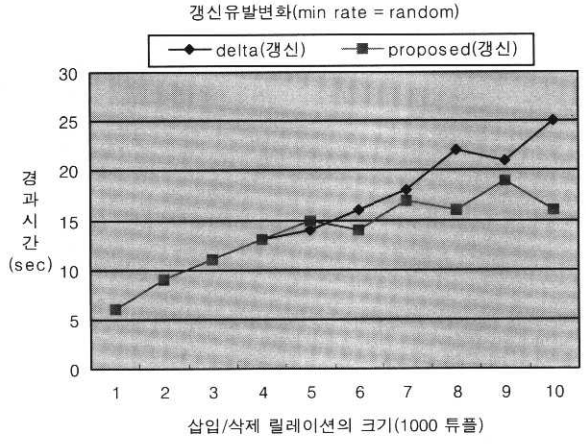
- 삽입유발변화(insertion-generating changes)

기본 릴레이션에 임의의 튜플들을 삽입하는 변경이다. 데이터웨어하우스의 기본릴레이션의 변화는 삽입유발변화가 일반적이다[1].



(그림 5) 전체적인 갱신작업의 경과시간 비교

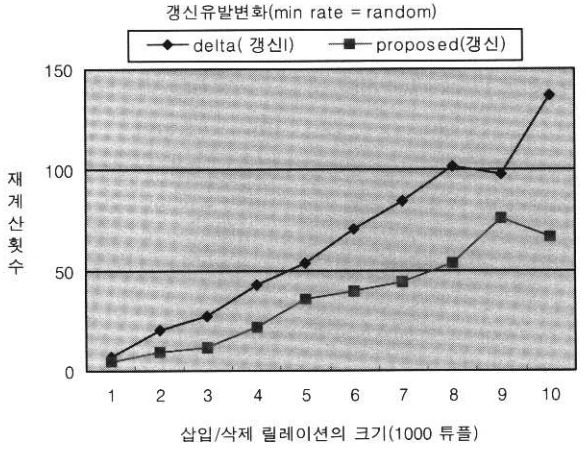
(그림 5)는 기본릴레이션 pos에 대한 갱신유발변화가 발생한 경우 실체뷰 sales의 갱신작업을 위하여 경과한 전체적인 시간(선계산 단계 + 갱신단계)을 비교하고 있다. 삽입/삭제릴레이션안의 MIN 값은 랜덤함수에 의해서 결정된다. 실험결과, 실체뷰의 MIN 값보다 작을 확률이 1% 미만이었다. proposed 알고리즘의 그래프가 들쑥날쑥한 이유는 랜덤함수로 생성된 삽입/삭제릴레이션의 MIN 값의 영향때문인 것으로 생각된다. 즉, 실체뷰의 MIN 값보다 작은 범위에서 삽입/삭제릴레이션안의 MIN값 변화가 자주 일어나면 갱신시간이 길어질 것이다. proposed 알고리즘은 delta 알고리즘에 비하여 삽입/삭제 연산을 구분하기 위한 과정을 더 포함하고 있지만 갱신단계의 경과시간이 단축되므로 전체적인 경과시간은 줄어든다.



(그림 6) 갱신단계의 경과시간 비교

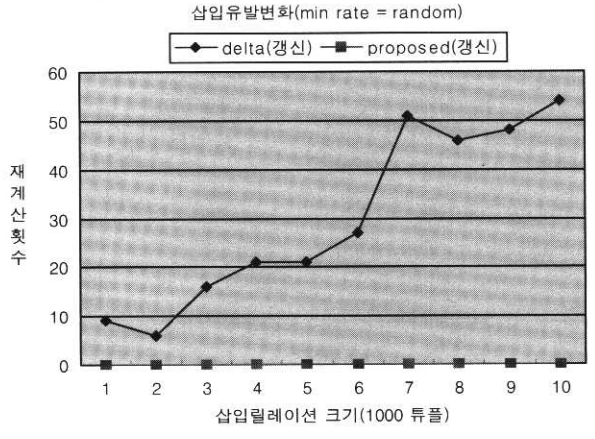
(그림 6)은 기본릴레이션 pos에 대한 갱신유발변화가 발생한 경우 실체뷰 sales의 갱신작업을 위하여 갱신단계의 경과시간을 비교하고 있다. ⊿I_pos 및 ⊿D_pos 안의 MIN값은 랜덤함수에 의해서 결정된다. proposed 알고리즘은 delta 알고리즘에 비하여 갱신단계의 경과시간이 짧다. 왜냐 하면 delta 알고리즘은 실체뷰의 차이릴레이션의 MIN/MAX 값이 삽입튜플인지 삭제튜플인지를 고려하지 않고 기본릴레이션으로의 접근을 고려하지만, proposed 알고리즘은 실체뷰의 차이릴레이션의 MIN/MAX 값이 삭제튜플인 경우에만 기본릴레이션으로의 접근을 고려하기 때문에 기본릴레이션을 이용한 재계산 횟수가 적어지기 때문이다.

(그림 7)은 갱신단계를 처리할 때 기본릴레이션으로 접근하여 재계산하는 횟수를 비교하고 있다. 실험결과, 단위항목에 대한 재계산 시간은 타 작업의 디스크접근 요구량 등 시스템 상황에 따라 달라질 수 있으므로, 재계산 횟수를 통한 비교가 보다 정확하다.

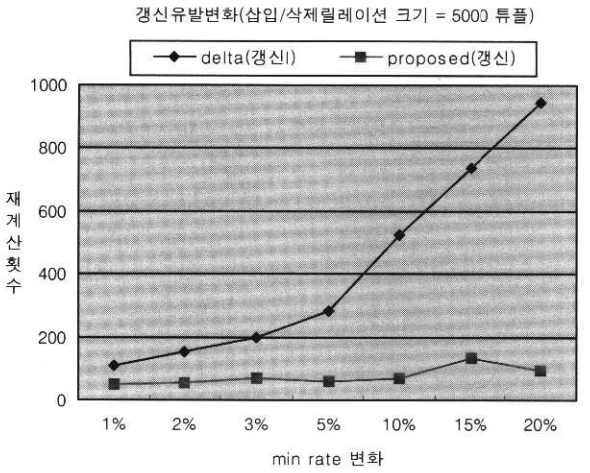


(그림 7) 갱신단계의 재계산 횟수(갱신유발변화의 경우)

(그림 8)은 기본릴레이션 pos에 대한 삽입유발변화가 발생한 경우 실체뷰 sales의 갱신작업을 위하여 갱신단계의 재계산 횟수를 비교하고 있다. proposed 알고리즘의 정책은 삽입유발변화의 경우 MIN/MAX 값을 자립유지할 수 있기 때문에 재계산을 할 필요가 없다. 데이터웨어하우스의 기본릴레이션의 변화는 삽입유발변화가 일반적이므로 삽입유발변화에서의 뛰어난 성능은 의미가 크다 할 수 있다.

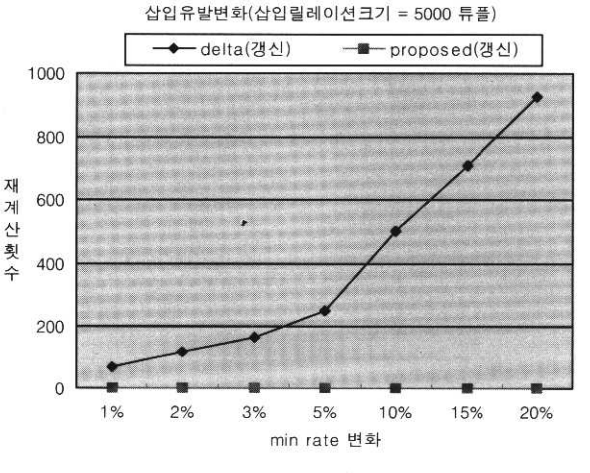


(그림 8) 갱신단계의 재계산 횟수(삽입유발변화의 경우)



(그림 9) Min Rate 변화에 따른 재계산 횟수 (갱신유발변화의 경우)

(그림 8)과 (그림 9)은 기본릴레이션에 대한 차이릴레이션안의 MIN 값이 실체뷰의 MIN 값보다 작을 확률을 변화시켰을 때 각 경우의 재계산 횟수를 비교하고 있다. 기본릴레이션의 차이릴레이션안의 MIN 값이 실체뷰의 MIN 값보다 작을 확률이 높다는 의미는 기본릴레이션에서의 MIN 값의 변화가 잦음을 의미한다. 차이릴레이션안의 MIN 값이 실체뷰의 MIN 값보다 작을 확률이 높을수록 proposed 알고리즘은 좋은 성능을 나타낸다.



(그림 10) Min Rate 변화에 따른 재계산 횟수 (삽입유발변화의 경우)

6. 결 론

데이터웨어하우스내의 기본릴레이션에서 자주 사용되는 종류의 데이터에 대해 이를 정리하고 요약한 실체뷰는 실제 질의처리시에 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 짧을 수록 보다 많은 수의 실체뷰들을 데이터웨어하우스 내에 포함할 수 있다.

본 논문에서는 MIN/MAX 집단함수가 포함된 실체뷰의 갱신시간을 단축시키기 위한 알고리즘을 제안하였다. 기존의 delta 알고리즘은 선계산 단계를 처리하기 위한 일반화된 알고리즘을 제시하지 않았지만 본 논문에서는 선계산 단계의 처리과정을 일반화시켰다. 또한, delta 알고리즘은 실체뷰의 차이릴레이션을 계산할 때 MIN/MAX 값의 삽입, 삭제 여부를 구별하지 않음으로써 MIN/MAX 값의 갱신을 위한 재계산 확률이 높았었다. 그러나, 제안한 알고리즘은 실체뷰의 차이릴레이션을 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 구별함으로써 실체뷰의 갱신시간이 단축됨을 알 수 있었다. 특히, 기본릴레이션의 차이릴레이션 안에 있는 MIN 값이 실체뷰내의 MIN 값보다 작을 확률이 높아질수록 제안한 알고리즘은 뛰어난 성능을 보여줌을 알 수 있었다.

참 고 문 헌

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

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

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

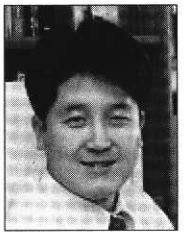
[4] Y. Kotidis, N. Roussopoulos, "DynaMat : A Dynamic View Management System for Data Warehouses," In proceedings of ACM SIGMOD Conference, pp.371-382, 1999.

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

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

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

ference, pp.1-8, 2000.
[8] 이기용, 김명호, "데이타웨어하우스에서 효과적인 점진적 뷰 리", 정보과학회논문지, Vol.27, No.2, pp.175-184, 2000.
[9] 윤원식, 신동천, "데이타웨어하우스 환경에서 조인비용을 기반으로 한 실체뷰 선택알고리즘", Vol.28, No.1, No.3, pp. 31-41, 2001.
[10] 김민정, 정연돈, 박웅제, 김명호, "데이터 큐브에서 세분화된 뷰실체화 기법", 정보과학회논문지, Vol.28, No.4, pp.587-595, 2001.
[11] 양우석, 김명호, "OLAP 환경에서 다중점 MAX/MIN 질의의 효율적인 처리기법", 정보과학회논문지, Vol.27, No.1, pp.13-21, 2000.
[12] 정희정, 김동욱, 김종수, 이윤준, 김명호, "OLAP에서 MAX-of-SUM 질의의 효율적인 처리기법", 정보과학회논문지, Vol. 27, No.2, pp.165-174, 2000.
[13] 조재희, "데이터웨어하우징 기술을 이용한 DB 마케팅 전략에 관한 연구", 정보기술과 데이터베이스 저널, 제6권 제1호, pp.104-113, 1998.



김 근 형

e-mail : khkim@cheju.ac.kr
1990년 서강대학교 컴퓨터학과(공학사)
1992년 서강대학교 컴퓨터학과(공학석사)
2001년 서강대학교 컴퓨터학과(공학박사)
1992년~1994년 현대전자 소프트웨어연구소
2001년~현재 제주대학교 경영정보학과 조교수
관심분야 : 데이터베이스, 데이터마이닝, 데이터웨어하우스, MIS

이 동 철

e-mail : dchlee@cheju.ac.kr
1981년 충남대학교 전기과(공학사)
1992년 국민대학교 MIS과(경영학석사)
한국과학기술원 서울분원 MIS과
2002년 성균관대학교 산업공학과(공학박사)
2003년~현재 제주대 경영정보학과 조교수
관심분야 : 전자상거래, MIS, 데이터베이스