데이터 웨어하우스에서 의사결정 트리를 이용한 실체화 뷰 선택 기법

[0]장윤경*, 유병섭*, 어상훈*, 김경배**, 배해영*
* 인하대학교 컴퓨터 정보 공학과
**서원대학교 컴퓨터 교육과

e-mail : {ykjang, subi, eosanghun}@dblab.inha.ac.kr,
gbkim@seowon.ac.kr, hybae@inha.ac.kr

Materialized View Selection using Decision Tree in Data Warehouse

[0]Youn-Kyung Jang*, Byeong-Seob You*, Sang-Hun Eo*,
Gyung-Bae Kim**, Hae-Young Bae*
* Dept. of Computer Science & Information Engineering, Inha University
** Dept. of Computer Education, Seowon University

요 약

실체화 뷰 선택은 질의 수행 시간과 제한된 저장 공간 등의 유지 비용을 고려하여 최적의 실체화 뷰 집합을 선택하고 유지하는 것이다. 본 논문에서는 의사결정 트리를 이용한 실체화 뷰 선택 기법을 제안한다. 제안기법은 의사결정 트리를 이용하여 실체화 뷰로 생성될 질의를 판단하고 실체화 뷰 교체가 필요한 경우 메타데이터 테이블을 이용하여 교체 대상을 결정한다. 의사결정 트리는 높은 우선순위를 가진 속성으로부터 차례대로 데이터를 분류하기 때문에 이용도가 높은 실체화 뷰를 선택하는 방법을 제공하고 메타데이터 테이블은 실체화 뷰 집합의 빠른 교체 수행과 효율적인 유지보수를 제공한다. 성능평가를 통해 제안된 기법은 실체화 뷰 비율에 따른 질의처리 시간이 기존기법보다 약 13%의 성능 향상을 보였다.

1. 서론

데이터 웨어하우스는 데이터를 주제 중심적이고 통합적이며 시간성을 가지는 비휘발성 자료로 저장하여 효율적인 의사결정을 지원하는 시스템이다. 이러한 데이터 웨어하우스의 OLAP 연산은 많은 양의 데이터를 검색하고 복잡한 집계 연산이 필요하므로 연산 시간이 일반 데이터베이스의 OLTP 연산보다 훨씬 증가한다.

OLAP 연산의 실행 시간을 단축하기 위해 제안된 기법 중 대표적인 것으로 자주 사용되는 질의의 결과를 미리 계산하여 저장해 놓는 실체화 뷰 기법이 있다[1]. 실체화 뷰 기법은 오랜 시간을 필요로 하는 복잡한 연산의 결과를 빠른 시간 내에 얻을 수 있다는 장점을 갖는다. 하지만 질의 결과를 실체화 뷰로 만드는 방법과 제한된 공간에서 최적의 실체화 뷰 집합을 유지하는 방법을 결정하는 것은 결정하기 어려운 문제이다.

*본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성 지원사업의 연구결과로 수행되었음.

최적의 실체화 뷰 구축을 위한 기존 연구로는 큐브 연산자의 격자 노드를 실체화 뷰로 구성 하는 방법[4], 인덱스까지 실체화 뷰 대상으로 하는 방법[5], 그리디 알고리즘을 이용한 격자 노드의 실체화 뷰 구축 방법[6] 등이 있었다. 위 방법들은 큐브 연산자와 같은 집합 연산으로 모델을 구성하므로 질의모델이 단순하고 수행 알고리즘이 복잡한 문제점이 있었다.

본 논문에서는 의사결정 트리를 이용한 실체화 뷰 선택 기법을 제시한다. 제안 기법은 질의 수행 시 의사결정 트리를 이용하여 질의 결과를 실체화 뷰로 저장할 것인지 판단하고 메타데이타 테이블을 이용하여 최적의 실체화 뷰 집합을 유지하는 기법이다. 이러한 의사결정 트리는 높은 우선순위에 따라 테스트 노드가 구성되기 때문에 합리적인 모델을 제시한다. 또한 이용도가 높은 실체화 뷰를 캐쉬에 상주시키고 효율적인 뷰 교체 수행으로 성능을 향상시킨다.

본 논문의 구성은 다음과 같다. 2 장에서는 기존 기법과 의사결정 트리에 대해 알아보고, 3 장에서는 본 논문에서 제안하는 의사결정 트리를 이용한 실체화 뷰 선택 기법에

대해서 서술한다. 4 장에서는 제안 기법의 성능평가를 한 뒤, 마지막 5 장에서 결론 및 향후 연구를 논한다.

2. 관련 연구

이 장에서는 기존의 실체화 뷰 선택 기법에 대해 비교를 하고 의사결정 트리에 대해서 알아본다.

2.1 기존 실체화 뷰 선택 기법

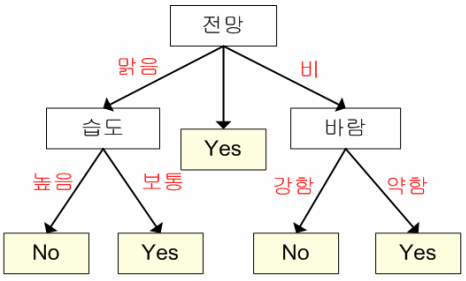
저장 공간을 줄이기 위해서 실체화 뷰의 수를 줄이거나 일부분만을 저장하는 기법[1]은 실체화 뷰 저장 공간은 줄일 수 있었으나 질의 수행 횟수에 관한 고려가 없었고, 질의 수행 횟수로 실체뷰 구축 대상을 선택하는 기법[8]은 효율적인 질의 수행을 지원하나 구축될 실체화 뷰 크기를 고려하지 않았다.

또 다른 방법은 큐브 연산자를 사용했을 때 생기게 되는 격자의 각 노드를 실체뷰 구축의 대상으로 고려하는 기법[4]이다. 이 기법은 격자내의 실체뷰들 중 적은 공간을 차지하면서 최적과 가까운 질의 처리를 하기위해 효율적인 실체뷰의 집합을 찾아내는 알고리즘[5,6]을 제안하였다. 데이터 큐브를 이용하여 실체뷰 구축을 하는 방법[5]에서는 사용할 수 있는 디스크의 크기에 제한이 있을 때 격자 내의 노드들 중에서 실체뷰로 구축할 것을 그리디 알고리즘으로 찾아내는 것을 제안하였고, 실체뷰 뿐만 아니라 인덱스까지 실체뷰 구축 대상으로 확장하는 방법이 있었다[6]. 하지만 위 방법들은 공통적으로 큐브 연산자와 같은 GROUP BY 집합 연산으로 질의모델을 구성하므로 질의모델이 단순하고 수행 알고리즘이 복잡한 문제가 있었다.

2.2 의사결정 트리

의사결정 트리는 데이터 마이닝의 분류와 예측 작업에 주로 사용되는 기법으로, 과거에 수집된 데이터의 레코드들을 분석하여 이들 사이에 존재하는 패턴, 즉 분류모형을 트리의 형태로 만드는 것이다. 트리의 노드는 속성 값에 대한 테스트, 각 트리의 가지는 테스트의 결과, 트리의 잎은 클래스 분포를 나타낸다. 각 속성은 가장 높은 정보 이득율을 가진 순서대로 루트에서부터 잎 노드까지 각 레벨로 배정된다. 따라서 잎 노드에 해당하는 각각의 클래스는 루트로부터 잎 노드까지 포함된 속성들에 따라 데이터의 분류별 특성을 잘 반영하는 집합이 된다.

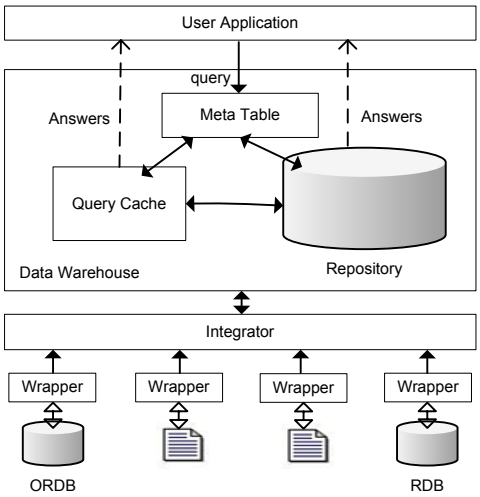
다음은 전망, 습도, 바람의 3 가지 속성에 따라 테니스를 치는지 여부(YES, NO)에 대한 의사결정 트리 모델이다. 3 가지 속성은 엔트로피와 정보 이득율 값에 따라 우선순위가 결정되게 된다. 그리고 속성들은 우선순위에 따라 루트로부터 하향으로 배정된다. 예시 모델에서는 전망, 습도/바람 순으로 우선순위가 부여되었음을 알 수 있고 각 속성은 조건(맑음, 비), (높음, 보통), (강함, 약함)에 따라 다른 하향 노드로 구분되어 최종 클래스(YES, NO)를 분류됨을 볼 수 있다.



(그림 1) 의사결정 트리 모델의 예

3. 의사결정 트리를 이용한 실체화 뷰 선택 기법

3.1 제안된 기법을 위한 데이터 웨어하우스의 구조



(그림 2) 제안된 기법을 위한 데이터 웨어하우스의 구조

그림 2 는 본 논문에서 제안한 데이터 웨어하우스의 구조를 나타낸다. 각 이질 데이터 소스로부터 들어온 데이터는 각각의 wrapper 를 거쳐서 integrator 를 통해 통합되고 데이터 웨어하우스 repository 에 적재된다. 사용자 질의가 들어오면 데이터 웨어하우스의 메타데이터 테이블을 이용하여 필요한 데이터가 query cache 에 존재하는지 검사한다. 만약 존재할 경우에는 바로 cache 에서 연산을 수행하고 그렇지 않을 경우 repository 에서 필요한 데이터를 가져온 후 연산을 수행한다. Query Cache 는 선택된 질의들의 결과가 실체화 뷰로 구축되어 저장되는 장소이고 메타데이터 테이블은 실체화 뷰의 유지보수를 위한 정보를 관리한다.

3.2 메타데이터 테이블

id	Query	View flag	Dirty bit	Position	
1	Select Ename from Source1	1	0		Materialized View
2	Select Ename from Source1 where Ename='kay'	0	0		Virtual View
3	Select class, date from Source2	1	1		Materialized View
4	Select class from Source2 where date='Mon'	-1	1		Candidate View
5	Select student_id from Source3 where class='Music'	0	0		Virtual View

Materialized View Candidate View Virtual View

(그림 3) 메타데이터 테이블

그림 3 은 메타데이터 테이블을 나타내는 그림이다. 메타데이터 테이블은 질의 요청이 들어왔을 경우 질의문을 저장하는 Query 필드와 뷰의 종류를 나타내는 View flag, 수정 여부를 나타내는 Dirty bit, 뷰의 위치를 나타내는 Position 필드로 구성되어 있다. View flag 는 다음 섹션에서 설명하게 될 의사결정 트리를 이용해서 Materialized View flag, Candidate View flag, Virtual View flag 중 하나로 그 값이 결정된다. Materialized View flag 는 Query 필드의 질의문의 결과가 실체화 뷰로 생성되었음을 나타내고 해당 질의들은 실체화된 내용을 이용하여 빠르게 수행될 수 있다. Candidate View flag 는 현재 Materialized View 로 저장되어 있지만 의사결정

트리를 이용하여 중요도가 떨어졌다고 판단되었을 경우 뷰 교체 정책 시 희생자 후보 표시로 부여하는 flag 이다. 따라서 Candidate View flag 를 가진 뷰는 다음 교체가 발생할 때 우선적으로 채택된다. Virtual View flag 가 나타내는 가상뷰는 실체화 뷰보다 적은 저장 공간을 가지지만 기본 릴레이션 변경 시 뷰도 함께 갱신해야 하는 특성을 가지고 있다. Query Cache 내의 실체화 뷰 교체가 필요한 경우 Candidiate View flag 를 가진 뷰는 우선적으로 선택되어 Virtual View 로 전환되고 새롭게 생성될 실체화 뷰를 위한 저장 공간을 양보하게 된다. 마지막으로 Position 필드는 각 뷰가 위치하는 곳의 포인터를 가지고 있다. Candidate View flag 를 이용한 뷰 교체 방식은 교체대상 선정에 대한 추가적인 연산 없이 환경에 따라 교체 대상이 동적으로 채택되기 때문에 빠른 속도로 뷰 교체를 수행하고 실행 비용을 감소시킨다. 이 때 채택된 뷰의 Dirty bit 가 참이라면 수정사항을 모두 반영한 후 교체를 수행한다.

3.3 의사결정 트리를 이용한 뷰 선택

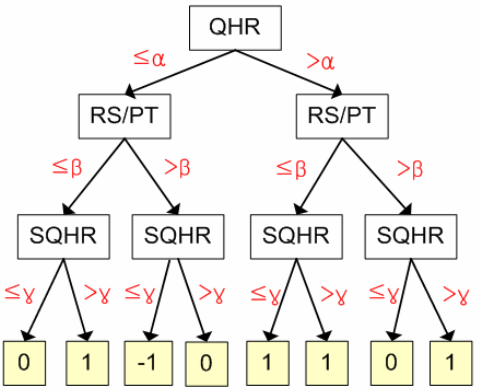
이번 장에서는 의사결정 트리를 이용한 뷰 선택 기법을 설명한다. 알고리즘 1 은 질의 요청이 들어왔을 때 의사결정 트리를 이용하여 생성될 뷰 종류를 선택하고 메타데이터 테이블의 View flag 를 셋팅하는 과정을 보여준다.

[알고리즘 1] Assign View Flag 알고리즘

```
Input  : Queries
Output : View Flag
Variables
QHR : query hit rate 쿼리 수행 횟수
RS : query result size 쿼리 결과 크기
PT : processing time 쿼리 수행 시간
SQHR : sub query hit rate 서브 쿼리의 수행 횟수
Threshold Parameters : α, β, γ
Materialized View Class Flag: 1 (질의 결과가 실체화 뷰로 생성될 질의)
Virtual View Class Flag: 0 (질의 결과가 가상 뷰로 생성될 질의)
Candidate View Class Flag: -1 (다음 교체시 가상 뷰로 전환 될 실체화뷰)

Algorithm AssignViewFlag (Queries)
Begin Algorithm
01 : if (QHR is equal to α or less than α)
02 :    if (RS/PT is equal to β or less than β)
03 :       if (SQHR is equal to γ or less than γ)
04 :          Query View Flag is 0;
05 :       else
06 :          Query View Flag is 1;
07 :       end if
08 :    else
09 :       if (SQHR is equal to γ or less than γ)
10 :          Query View Flag is -1;
11 :       else
12 :          Query View Flag is 0;
13 :       end if
14 :    end if
15 : else
16 :    if (RS/PT is equal to β or less than β)
17 :       if (SQHR is equal to γ or less than γ)
18 :          Query View Flag is 1;
19 :       else
20 :          Query View Flag is 1;
21 :       end if
22 :    else
23 :       if (SQHR is equal to γ or less than γ)
24 :          Query View Flag is 0;
25 :       else
26 :          Query View Flag is 1;
27 :       end if
28 :    end if
29 : end if
30 : return Query View Flag;
End Algorithm
```

제안 기법은 실체화 뷰를 구축할 때와 유지를 위한 교체를 수행할 때 QHR, RS, PT, SQHR 과 같은 속성들을 고려한다. QHR(query hit rate)은 데이터 웨어하우스의 초기 적재, 또는 갱신 후 질의 수행 횟수를 나타낸다. 자주 수행되는 질의일수록 실체화 뷰로 구축되었을 때 유용하고, 구축 후에 해당 실체화 뷰가 드물게 이용된다면 교체 정책 수행 시 희생자로 선택될 가능성이 높아진다. RS(result size)와 PT(processing time)는 각각 쿼리 결과 크기와 쿼리수행 시간을 나타내고 RS/PT 은 저장공간과 접근 비용을 함께 고려한 것으로 사용자의 연산 지연시간에 영향을 주게 된다. 즉, 의사결정 트리의 검사 조건이 RS/PT ≤ β 라고 했을 때, 사용자 질의에 대한 지연 시간이 긴 환경에서는 β 값을 크게 한다. β 파라미터 값이 크다는 것은 연산 처리 시간(PT)에 비해 결과 크기(RS)가 큰 질의가 실체화 뷰로 구축될 가능성이 높아지는 것이고 이는 해당 질의가 디스크 접근 없이 캐쉬 내에서 수행될 확률이 높아진다는 것을 의미한다. 결과적으로 β 파라미터 값의 증가는 사용자의 연산 지연시간을 줄이고 빠른 질의 처리를 제공한다. 반대로 사용자의 연산 지연시간이 짧은 상황이라면 낮은 β 값을 선택하여 연산처리 시간(PT)이 길고 결과 크기(RS)가 작은 질의 결과가 실체화 뷰로 구축되어 캐쉬 내 저장공간을 절약한다. SQHR 은 sub query hit rate 로 어떤 질의 Qs 의 모든 결과가 질의 Qa 의 결과에 포함된다면 Qs 를 Qa 의 sub query 라고 하고 Qs ⊆ Qa 로 나타낼 수 있다. 질의문의 수행 빈도가 낮다고 하더라도 자주 수행되는 sub query 들이 많이 존재한다면 실체화 뷰로 구축하는 것이 더욱 효율적이다. 질의 결과 크기가 매우 크고 sub query 수행 빈번도를 나타내는 SQHR 의 값이 작다면 저장공간과 이용도를 고려할 때 실체화 뷰로 구축하지 않는 것이 바람직하다.



(그림 4) 의사결정 트리를 이용한 뷰 선택

의사결정 트리에서 고려되는 QHR, RS/PT, SQHR 등의 속성은 엔트로피와 정보 이득값을 계산하여 정보 이득율이 높은 순서대로 우선 순위가 매겨지게 된다. 각 노드의 속성값에 대한 테스트가 되는 임계값 파라미터 α, β, γ 는 비용과 저장측면의 tradeoff 측면을 고려하여 최적기 또는 시뮬레이션 모델에 의해 동적으로 할당되게 된다. 그림 4 는 속성 순위가 QHR > RS/PT > SQHR 이라고 할 때 생성된 의사결정 트리 모델의 예이다. 의사결정 트리에 질의문이 들어오면 우선순위로 매겨진 속성으로 검사되고 각각 α, β, γ 파라미터 값에 따라 하위 노드로 구분되어진다. 최종적으로 leaf 노드에 도달하면 각각 -1(Candidate View flag), 0(Materialized View flag), 1(Virtual View flag) 클래스로 분류되어 메타데이터 테이블의 View flag 필드에 할당된다. 의사결정 트리를 이용한 뷰 선택기법은 중요도에 따라 정해진 속성을 가지고 뷰 클래스를 구분하기 때문에 뷰 선택기준이 국소적이지 않고 신뢰성

이 높다. 또한 후보뷰에 대한 flag 를 가짐으로써 캐쉬 내의 저장공간이 부족할 경우 추가적인 연산 없이 신속한 교체를 수행할 수 있다.

전체적으로 살펴보면, 해당 질의에 대해서 메타데이터 테이블의 Query 필드를 보고 Query Cache 내에서 직접 수행 가능한지 판단한 후 의사결정 트리를 이용하여 결과로 생성된 View flag 로 실체화 뷰 구축 여부를 결정한다. Query Cache 내의 저장공간이 부족하여 실체화 뷰 교체가 필요한 경우 메타데이타 테이블의 View flag 가 -1(Candidate View flag)인 것을 교체 대상으로 선택한다.

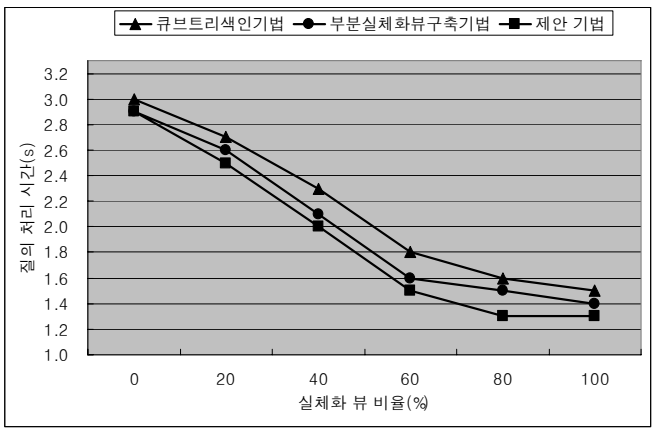
4. 성능평가

본 장에서는 제안된 기법의 성능 평가를 한다. 실험환경은 표 1 과 같으며, Query Cache 내의 실체화 뷰 비율이 늘어남에 따른 질의처리시간 변화를 측정하였다. 성능평가를 위하여 10 개의 임의 테이블을 만들었으며, 각 테이블은 10,000 개의 튜플을 갖도록 하였다. 또한 질의는 각 테이블의 데이터를 10~50%까지 10% 단위로 고르게 분포되도록 하였다.

<표 1> 실험환경

기종	IBM PC 호환
중앙처리장치	Pentium4 2.6 GHz
주 기억장치	2GB
보조 기억장치	120GB, 7200RPM, ATA 방식
운영체제	MS Window 2000
개발 언어	C/C++

그림 5 는 실체화 뷰의 비율에 따라 기존의 큐브 트리 색인기법, 부분 실체화 뷰 구축 기법과 제안기법의 질의처리 시간을 평가한 것이다. 그림에서 보면 제안기법이 기존 기법보다 질의 처리 시간이 감소한 것을 볼 수 있다. 이는 의사결정 트리가 실체화 뷰 구축 대상 선정에 있어서 저장공간이나 질의 수행횟수 같은 단일 속성이 아닌 여러 속성을 이용하여 신뢰성이 높은 기준을 가지기 때문이다. 또한 실체화 뷰 비율이 커져서 교체를 수행할 때 이용도가 낮아져서 후보뷰 비트가 할당된 뷰를 교체 대상으로 선택하기 때문에 추가적인 교체 횟수를 줄이고 이용도가 높은 뷰를 캐쉬에 상주시킴으로써 질의처리시간을 감소시킬 수 있다. 본 논문에서 제안된 기법은 큐브 트리 색인기법과 부분 실체화 뷰 구축 기법보다 최대 13%의 성능향상을 보였다.



(그림 5) 질의처리 시간 비교

5. 결론 및 향후 연구

본 논문에서는 의사결정 트리를 이용한 실체화 뷰 선택 기법을 제안하였다. 실체화 뷰를 구축함에 있어서 쿼리 수행 횟수, 쿼리 결과 크기, 쿼리 수행 시간, 서브 트리 수행 횟수 등 여러 속성을 고려한 의사결정 트리 모델을 이용하여 저장공간과 비용의 측면을 최적화 시킨다. 또한 Query Cache 에 실체화 뷰를 위한 저장공간이 부족한 경우 의사결정 트리 기법의 결과로 할당된 메타 데이터 테이블의 view flag 를 이용하여 후보뷰 클래스에 속하는 뷰를 희생자로 선정하여 빠른 뷰 교체를 수행한다. 이러한 후보뷰 이용은 이용도가 낮아진 뷰를 교체 대상으로 선택하고 이용도가 높은 뷰를 캐쉬에 상주시킴으로써 빠르고 효율적인 질의처리를 제공한다.

본 논문에서 이용된 의사 결정 트리는 데이터가 소규모일 때 정확한 분류를 제공하지 않지만 데이터 웨어하우스와 같은 대용량 데이터에서는 신뢰성 있는 분류를 수행한다. 결과적으로, 제안된 기법은 실체화 뷰 비율에 따른 질의 처리시간에 대하여 기존 기법보다 13 %의 성능 향상을 보였다.

향후 연구로는 의사결정 트리의 확장성을 이용하여 공간 데이터 웨어하우스에서의 효율적인 실체화 뷰 선택 기법에 관한 연구가 필요하다.

참고문헌

[1] N. Roussopoulos, “Materialized Views and Data Warehouse”, SIGMOD Record, 27(1), pp. 21-26, March 1998.

[2] A. Y. Levy, A. Rajaraman, A. O. Mendelzon, Y. Sagiv, D. Srivastava, “Answering Queries using Views”, In Proceedings of ACM PODS, 1995.

[3] H. Gupta, “Selection of Views to Materialized in a Data Warehouse”, In Proceedings of ICDT, pp. 98-112, 1997.

[4] J. Gray, A.Bosworth, A.Layman, H.Piramish, “Data Cube : A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals”, In Proc. of ICDE, pp. 152-159, 1996

[5] V. Harinarayan, A. Rajaraman, J. Ullman, “Implementing Data Cubes Efficiently”, In Proc. of ACM SIGMOD, pp. 205-216, 1006

[6] H.Gupta, V.Harinarayan, A.Rajaraman, J. Ullman, “Index Selection for OLAP”, In Proc. of ICDE, pp.208-219, 1997

[7] D. Agrawal, A. El Abbadi, A. Singh, T. Yurek, “Efficient View Maintenance at Data Warehouse”, In Proc. of ACM SIGMOD, 1997

[8] 이승용, 박재복, 김명희, 주수종, “분산환경에서 혼용 뷰 관리기법을 채택한 이질적인 멀티데이타베이스 상호운용 모델 설계”, 한국정보처리학회 논문지 D, VOL. 12-D NO. 04 pp. 0531 ~ 0542 2005 . 08

[9] Zohn Bellahsene, Philippe Marot, “Materializing a Set of Views: Dynamic Strategies and Performance Evaluation”, IEEE, 2000

[10] Aditya N. Saharia, Yair M. Babad, “Enhancing Data Warehouse Performance through Query Caching”, The DATABASE for Advances in Information Systems, Vol.31 NO.3, Summer 2000

[11] 이태희, 장재영, 이상구, “데이터 웨어하우스 환경에서 최적 실체뷰 구성을 위한 효율적인 탐색공간 생성 기법”, 정보과학회논문지 데이터베이스 제 31 권 제 6 호, 2004.12.

[12] 최준호, 유병섭, 박순영, 배해영, “공간 데이터 웨어하우스에서 분포 지역 질의 처리를 위한 확장된 큐브 트리 기법”, 한국정보과학회 춘계 학술발표 논문지 Vol.31, No.2, 2004