CORBA 기반에서 효율적인 질의 처리를 위한 실체뷰와 시그니쳐 뷰인덱스의 혼용

이 승 용[†] · 김 명 희[††] · 주 수 종[†††]

요 약

현재 멀티 데이터베이스 시스템에서 질의 처리의 향상을 위해 응용되고 있는 뷰 관리에 대한 대표적인 연구 경향은 실체 뷰 기법과 시그니쳐 뷰인덱스 기법에 관심이 집중되고 있다. 그러나 이들 두 기법의 비교에서 보면, 실체 뷰 기법은 빠른 액세스 시간을 가지지만 많은 저장 공간이 필요하고, 뷰 인덱스 기법은 적은 저장공간을 사용하지만 액세스 시간이 느리다는 상반된 특성을 가지고 있다. 따라서, 본 논문에서는 실체 뷰와 시그니쳐 뷰인덱스 혼용 기법을 제안하여 질의처리시에 뷰 관리를 사용함으로써 시스템 성능을 향상시키고, 디스크 입/출력에 대한 액세스 비용을 줄이고자 한다. 본 논문에서는 이 기법에 대한 메타데이타 구조와 알고리즘들에 대해 제안하고 있다.

Mixed Uses of Materialized View and Signature View-Index Mechanism for Efficient Query Processing on CORBA

Seung Yong Lee[†] · Myung Hee Kim[††] · Su Chong Joo[†††]

ABSTRACT

Now, the representative researching trends of view managements for improving the query processing in multi-database system are focused on the materialized view mechanism and the signature view index mechanism. But when we compare with both mechanisms, the former mechanism's access time is faster than one of the latter's, and needs large space. The latter mechanism needs small space and the access time is slower than one of the former. These mechanisms are trade-off each other. Therefore, in case of query process using the view management, we are to improve the system performance and to reduce the access cost of disk input and output by suggesting a new mechanism mixing both the materialized view mechanism and the signature view index mechanism. We suggested that the structure of metadata and the algorithm about the new mixed mechanism.

키워드 : 멀티데이타베이스(Multi-Database), 뷰(View), 실체뷰(Materialized View), 시그니쳐 뷰인덱스(Signiture View Index), 혼용 뷰 (Mixed View)

1. 서 론

기관이나 기업들이 데이터를 효율적으로 처리하기 위해 고유의 업무특성에 맞도록 독자적인 데이터베이스를 구축하여 사용하고 있다. 이들 데이터베이스는 구축된 시기가 다르고 서로 다른 DBMS(Database Management System)을 사용하고 있다. 또한 각 DBMS에서는 데이터 모델, SQL 그리고 트랜잭션 관리 기법을 서로 다르게 사용할 뿐만 아니라, 데이터의 형태, 단위 그리고 저장형식도 서로 다를 수 있다. 이러한 환경에서 응용업무가 복잡해지고 다양해지면서 기존에 구축되어 있는 여러 데이터베이스 시스템에 저장되어 있는 데이터를 공유해야만 하는 전역 사용자를 위한 데이터 공유 시스템의 필요성이 대두되었다. 응용업무와 관련 있는 데이터베이스들을 상호 참조하기 위해 이를 하나의 통합된 시스템으로 구성하여 상호 연동으로 사용자 입장에서는 데이터베이스 모음을 마치 하나의 데이터베이스처럼 투명하게 사용할 수 있도록 지원해야 한다. 이러한 시스템을 멀티 데이터베이스 시스템(Multi-Database System : MDBS)[1-5]이라고 한다.

본 연구에서는 분산환경에서 이질적인 멀티 데이터베이스 운영시 시스템의 성능을 향상시키고, 질의 처리시에 디스크 입/출력에 대한 액세스 비용을 줄일 수 있도록 데이터베이스에서 뷰에 대한 질의 처리 방안으로 사용하는 시그니쳐 뷰인덱스 기법과 멀티 데이터베이스 관리 시스템(Multi-Database Management System : MDBMS)에서 사용하는 실체 뷰 기법을 혼용한 혼용 뷰 관리기법을 정의하고자 한다.

이를 위해, 본 논문의 2장에서 MDBMS 구조를 제안하고 기존의 실체 뷰 기법과 시그니쳐 뷰인덱스 기법에 대한 구조와 알고리즘을 설명한다. 3장에서 본 연구에서 제안한 혼용 뷰 관리기법에 대한 구조와 처리과정을 기술한다. 4장에서는 생성뷰를 실체화하거나 시그니쳐 뷰인덱스화하는 과

※ 본 논문은 2003학년도 원광보건대학 교내연구비지원에 의하여 이루어짐.
† 종신회원 : 원광보건대학 컴퓨터응용개발과 교수
†† 정 회 원 : 원광디지털대학교 게임소프트웨어학과 교수
††† 정 회 원 : 원광대학교 컴퓨터공학과 교수
논문접수 : 2003년 3월 5일, 심사완료 : 2003년 10월 2일

정 그리고 시그니쳐 뷰인덱스를 실체뷰화하는 과정간의 절차를 설명하고 5장에서 결론을 내리도록 한다.

2. 관련 연구

이 장에서는 MDBS에서 정보 검색을 위해 부여된 전역 질의 처리를 신속하게 하기 위하여 사용되던 뷰의 정의를 이용한 실체 뷰와 시그니쳐 뷰인덱스 기법에 대해서 알아본다.

기존의 방법들은 뷰를 통해 MDB를 사용하는 전체 사용자들이 바라보는 데이터베이스의 내부 스키마와 각 사용자 또는 응용에 따른 외부 스키마를 분리 제공함으로서 데이터 독립성을 제공한다. 뷰에 대한 질의 처리는 기본 릴레이션에 대한 질의로 변경시켜야 하며, 이 과정에서 많은 입/출력의 요구로 처리 속도가 저하된다. 이 문제를 해결하기 위해 여러 가지 방법이 제안되었는데 대표적인 것으로는 실체 뷰(materialized view)[6-9]와 시그니쳐 뷰인덱스(signatured view Index)기법[10-14]이 있다.

2.1 실체 뷰 기법

실체 뷰는 뷰에 의존하는 응용을 빠르게 하기 위해서 뷰처럼, 복잡한 질의 결과를 실체화한 뷰[6, 9]이다. 이러한 실체 뷰를 이용함으로써 사용자에게 멀티 데이터베이스를 검색하는데 소요되는 오버헤드를 줄일 수 있다.

(그림 1)은 실체 뷰의 메타데이타가 메타데이터 저장소에 저장되는 구조를 표현한 것이다.

```
interface emp
{ int        id ;
  int        ReplaceDataSource ;
  String     QueryOp ;
  int        DirtyBit ;
  float      NoAccess ;

  질의처리 method()
  질의처리 method() … }
```

(그림 1) 실체 뷰의 메타데이터 구조

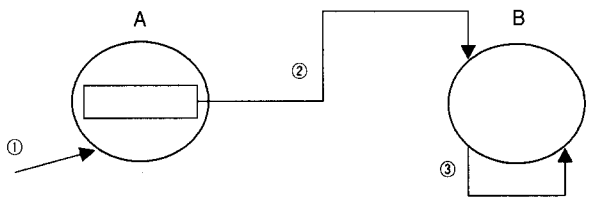
*id*는 실체 뷰의 ID를 가리킨다. *ReplaceDataSource*는 데이터베이스나 파일 시스템등의 실체 뷰의 원래 위치를 가리키는데, 이것으로 실체 뷰가 참조한 테이블을 알 수 있고, 참조된 테이블이 변화할 경우 이를 보고하여, 실체 뷰의 유지보수를 돕는다. 데이터베이스에 각 질의가 처리될 때마다 실체 뷰가 생성되며, 실체 뷰의 메타데이터도 메타데이터 저장소에 저장된다. 실체 뷰의 메타데이터 구조는 위의 내용을 참고로 매 질의가 처리될 때마다, 기존의 실체 뷰에 같은 내용이 있는지 검색하여 보고, 있을 경우 그 실체 뷰를 사용한다. *QueryOp*는 이미 수행된 질의 내용을 참조한다. *DirtyBit*는 실체 뷰의 원위치에 있는 데이터가 변경(변형)되었는지를 나타낸다. 원위치에 있는 데이터가 변경되었을 경우에는 "1"로 표시하고, 변경되지 않았을 경우에는 "0"으로 표시한다. *NoAccess*는 실체 뷰의 접근 지수로서, 이것을 이용하여 실체 뷰의 저장공간에서 자주 사용되지 않는 뷰를 삭제한다. 처음 실체 뷰가 생성될 때 메타데이터의 *NoAccess*는 0으로 설정된다. 그 이후에 사용될 때마다 1씩 증가하게 된다.

2.2 시그니쳐 뷰인덱스 기법

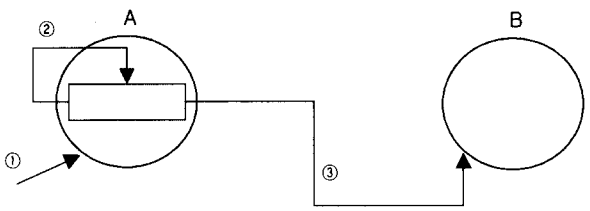
뷰 인덱스 기법은 적은 저장 공간을 사용한다는 면에서 실체 뷰 기법에 비해 장점이 있지만, 기본 릴레이션의 갱신 시 뷰인덱스들을 갱신하여야 하며, 또한 인덱스에는 뷰가 사용하는 기본 릴레이션의 투플 주소만을 저장하기 때문에 뷰 조건이 주어지는 경우는 실제 릴레이션을 검색하여 비교하여야 하는 문제가 있다. 이러한 단점을 보완한 연구가 시그니쳐 뷰인덱스 기법[13, 14]이다.

시그니쳐는 집합 값에 대한 일종의 코드로 적은 저장비용을 사용하여 주어진 정보를 요약한 것이다. 시그니쳐를 검사하면 주어진 값에 해당 시그니쳐와 매치가 되는가를 파악할 수 있다. 최근에 제시된 시그니쳐 참조기법은 기존의 시그니쳐 파일 기법에서와 같이 시그니쳐 파일을 별도로 구성하여 질의 처리시 전체 시그니쳐 파일을 검색하는 방식이 아니라, 객체지향 데이터베이스의 예와 같이 객체간에 어트리뷰트를 통한 참조 관계가 있거나 또는 중첩 릴레이션 데이터베이스에서 서브 릴레이션을 포인터를 통해 접근하여야 하는 포인터 관계에 적용된다.

객체 A가 다른 객체 B를 어트리뷰트의 값을 통해 참조한다는 것은 객체 A에 대한 연산과정에서 이 참조관계를 통해 객체 B의 정보를 검색한다는 것이다. 객체지향 데이터베이스에서 참조관계를 통한 객체들의 검색 과정은 피참조 객체를 검색하기 위해 별도의 입/출력이 필요하다. 이러한 검색 과정은 일반적으로 참조 객체(A)가 피참조 객체(B)에 대한 조건이 필요한 경우 이 처리하기 위해 수행된다. 만일 참



(a) 기존의 검색과정



(b) 시그니쳐를 사용한 검색과정

(그림 2) 참조 관계를 통한 객체 검색 과정

조객체에 피참조 객체에 대한 정보가 저장되어 있다면, 미리 조건을 검사할 수 있으므로 불필요한 검색을 하지 않아도 되어 신속한 연산 처리가 가능하게 된다. (그림 2)는 참조 관계를 통한 객체 검색과정을 비교하여 보인다.

(그림 3)은 시그니쳐 뷰인덱스 기법을 설명하기 위한 실제 예를 보여준다.

TID	SNO	SNAME	DEPT	AGE	Year
$STID_1$	S100	Kim	Computer	20	1
$STID_2$	S200	Lee	Mechanics	21	2
$STID_3$	S300	Park	English	22	3
$STID_4$	S400	Choi	Computer	23	4

(그림 3) Student 릴레이션

이 릴레이션에 대해 다음과 같은 뷰 CompStud가 정의되었다고 하자.

```
Create View CompStud
  As Select SNAME, AGE, YEAR
    From Student
      Where DEPT = 'Computer'
```

이 뷰에 대해 뷰인덱스에는 {$STID_1$, $STID_4$}와 같이 student 릴레이션의 두 TID가 저장될 것이다. 이 때 다음과 같은 질의가 뷰 CompStud에 주어진다고 하자.

Select × From CompStud Where AGE = 20

뷰인덱스를 사용하지 않는 경우에는 Student 릴레이션의 전체 튜플에 대해 검색하여야 하지만 뷰인덱스가 사용되는 경우에는 뷰인덱스에 저장된 두 개의 TID에 해당하는 튜플만을 검색하여 AGE 어트리뷰트의 값이 20인지 여부를 검사하게 된다. 그러나 시그니쳐 뷰인덱스 기법은 시그니쳐 참조기법을 사용하여 뷰인덱스에 튜플의 TID만이 아니고 시그니쳐를 함께 저장하는 경우에는 뷰인덱스에 저장된 튜플의 시그니쳐만을 이용하여 'AGE = 20'이라는 조건을 만족하는지 검사할 수 있으며 만족하는 경우에만 실제 튜플을 검색함으로써 입/출력을 감소할 수 있게 된다. 그러므로 뷰인덱스에서 시그니쳐를 비교하여 조건을 만족하는 $STID_1$ 튜플만을 검색하게 된다. (그림 4)는 위의 뷰 CompStud에 대한 시그니쳐 뷰인덱스이다.

TID	시그니쳐
$STID_1$	S(Kim, 20, 1)
$STID_4$	S(Choi, 23, 4)

(그림 4) 뷰 CompStud의 시그니쳐 뷰인덱스

시그니쳐 뷰인덱스를 사용하기 위해서는 다음과 같은 정의가 필요하다.

- Attr(V) : 뷰 V의 정의에서 사용된 목표 어트리뷰트 리스트
- Cond(V) : 뷰 V를 정의하는 데 사용된 조건식들

위의 기호에 따른 뷰 CompStud에 대한 예는 아래와 같다.

- Attr(CompStud) : {SNAME, AGE, YEAR}
- Cond(CompStud) : DEPT = 'Computer'

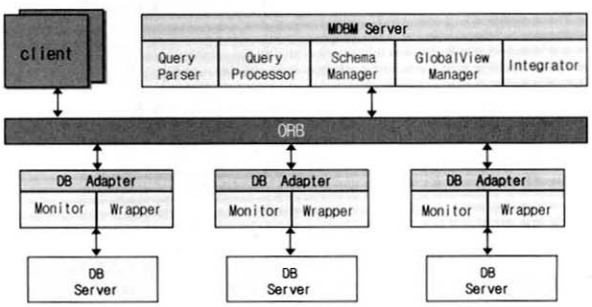
- 정의 1 : 릴레이션 R에 정의된 뷰 V에 대한 뷰인덱스는 릴레이션 R에서 Cond(V)를 만족하는 튜플들의 TID의 집합 {TID_i}로 정의된다.
- 정의 2 : 릴레이션 R에 정의된 뷰 V에 대한 시그네쳐 뷰인덱스는 집합 {(TID_i, $S(TID_i)$)}이다. 이때 TID_i는 릴레이션 R에서 Cond(V)를 만족하는 튜플들의 TID이고 $S(TID_i)$는 해당 튜플의 Attr(V)에 속하는 어트리뷰트의 값들로 만든 시그니쳐이다.

3. 제안된 혼용 뷰 관리 기법

본 장에서는 기존의 뷰관리 기법 및 혼용 뷰 관리 기법을 비교 분석하고 적용시키기 위한 기본적인 환경에 해당될 MDBMS의 구조를 설명하고, 기본구조에 각각의 뷰 관리기법들을 적용하여 기법들간의 특성을 파악하며, 실체 뷰와 시그니쳐 뷰 인덱스 혼용 기법을 설명한다. 혼용 기법에 대한 기본적인 원리 및 메타데이타 구조와 알고리즘등을 설명한다.

3.1 MDBMS의 구조

본 연구에서 제안한 MDBMS는 클라이언트, MDBM 서버(MDBM Server), LDB(Local DB) 어댑터(Adapter)와 LDB로 구성되며, 이들간의 통신은 ORB(Object Request Broker)를 이용한다. 제안된 MDBMS의 구조는 (그림 5)와 같다.



(그림 5) 제안된 MDBMS 구조

MDBM 서버는 질의 파서기(Query Parser), 질의 처리기(Query Processor), 스키마 관리자(Schema Manager), 전역 뷰 관리자(Global View Manager), 통합기(Integrator)로 구성된다.

질의 파서기는 클라이언트의 전역 질의를 분석하여 LDB에 적합한 지역 질의로 파싱한다. 질의 처리기는 지역 질의로 분할된 전역 질의를 처리하기 위하여, 적절한 LDB에 해당되는 지역 질의를 전달하고 LDB로부터 처리되어 오는 결과를 처리한다. 스키마 관리자는 전역 스키마 관리를 담

당한다. 통합기는 전역 질의에 대한 최종 결과를 클라이언트에게 반환하기 위하여, 질의 파서기에 의해 분할된 지역 질의들의 결과를 통합 및 관리한다. 전역 뷰 관리자는 실체 뷰와 시그니쳐 뷰인덱스 혼용기법을 이용하여 전역 뷰에 대한 요청을 처리하고 뷰를 관리한다.

DB 어댑터는 랩퍼(Wrapper)와 모니터(Monitor)로 구성된다. 모니터는 LDB의 데이터가 변화되었을 때 일관성 유지를 위하여 사용되며, 랩퍼는 LDB와의 인터페이스 역할을 담당하며, LDB에 대한 지역 질의에 대한 처리를 담당한다.

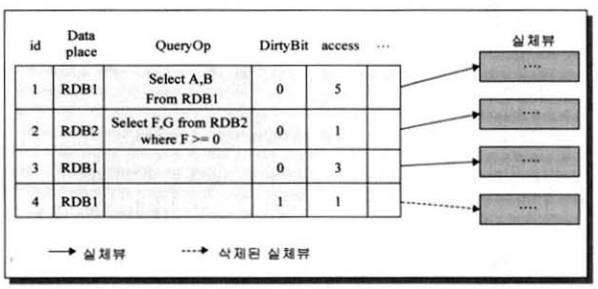
3.2 실체 뷰의 적용

본 논문에서는 (그림 1)에서 설명한 실체 뷰의 메타데이터 구조를 기반으로 실체 뷰를 관리한다. 실체 뷰의 저장소는 전역 뷰 저장소를 사용한다. 질의 처리가 일어날 때마다, 질의 처리 결과로 나타나는 실체 뷰를 전역 뷰 저장소에 저장한다. 같은 질의가 일어나는 경우 메타데이터의 *NoAccess*값을 사용하여 이미 저장된 실체 뷰를 재사용하여, 질의 처리 시간을 줄일 수 있다.

각 사용자는 질의를 수행할 때마다 메타데이터의 *Query Op*를 참조하여, 자신의 질의 수행내용이 실체 뷰에 있는지 확인한다. 이때, 수행되어야 할 질의와 같은 의미, 혹은 질의의 서브질의의 실체 뷰가 이미 존재한다면, 그 실체 뷰를 사용하여 질의 수행절차를 줄인다.

실체 뷰의 변경 관리는 기존 데이터와 실체 뷰의 유지보수를 위해 필요하며, 필드 단위의 변경 보고를 지원한다. 기존 데이터, 실체 뷰, 메타데이터 저장소의 정보는 일관되게 유지되어야 한다. 메타데이터가 가리키는 실체 뷰의 원위치에 있는 데이터가 변경되었을 때에는 실체 뷰를 삭제한다. 이때, 페이지 교체 기법은 LRU(Least Recently Used) 기법을 적용한다. 데이터가 변경된 실체 뷰를 삭제한 후, 그에 대한 질의가 다시 수행되면, 실체 뷰와 실체 뷰에 대한 메타데이터가 새로 생성된다.

이러한 기법은 자주 변화되는 데이터를 참조하는 경우보다는 데이터 변경이 비교적 적은 환경에 적합하다. (그림 6)은 실체 뷰에 대한 메타데이터의 예이다.



(그림 6) 실체 뷰의 메타데이터의 예

(그림 6)에서 보면, ID 1은 'select A, B from RDB1'의 질의에 의해 생성된 실체 뷰를 가리킨다. 이 실체 뷰 데이터의 원위치는 RDB1에 해당된다. *Dirtybit*가 0으로 설정된 것은 이 실체 뷰가 생성된 이후에, 원위치의 데이터가 변경되지 않았음을 나타내며, *NoAccess*가 5인 것은 이 실체 뷰가 5번의 질의 요청을 수행했다는 것을 의미한다. 반면에, ID 4는 *Dirtybit*가 1로 설정되어 있음을 알 수 있다. 이것은 ID 4가 가리키는 질의에 대한 실체 뷰가 형성된 이후에 원위치에 있는 데이터가 변경되었음을 나타낸다. 다시 말해서, 원위치의 데이터가 변경되었을 때, 실체 뷰의 메타데이터의 *Dirtybit*값은 1로 바뀌고, 실체 뷰는 전역데이터 저장소에서 삭제된다. 이때, ID 4에 대한 질의가 다시 발생하면, 새로운 실체 뷰가 생성되고 그에 대한 메타데이터가 갱신된다.

(그림 7)은 실체 뷰 관리에 대한 수행 알고리즘을 나타낸다.

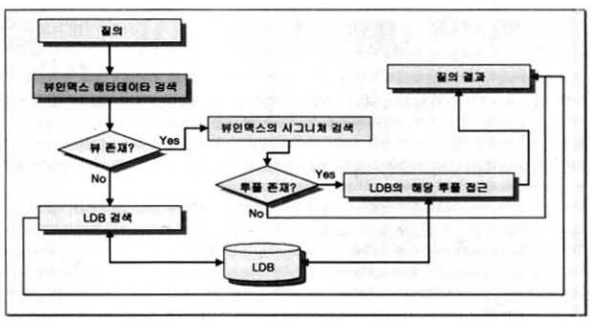
```
Materialized_View_Management Algorithm

/* input query */
q ← query
/* check the meta-data of Materialized View */
/* view data exist in meta-data */
if QueryOp_i = q in meta-data for id index i = 1, ···, n then
   access Global View Repository indicating id_i
   get the view data
   return the result about query q
/* view data not exist in meta-data */
else
   access the local DBs
   check the entire data for finding the result data of q
   return the result about query q
```

(그림 7) 실체 뷰 관리 알고리즘

3.3 시그니쳐 뷰인덱스의 적용

(그림 8)은 본 연구 모델에서 전역 뷰 관리자가 시그니쳐 뷰인덱스 관리기법을 이용하여 전역 질의에 대한 처리를 수행할 경우의 과정을 나타낸다.



(그림 8) 시그니쳐 뷰인덱스 관리 기법

(그림 8)을 살펴보면, 시그니쳐 뷰인덱스 관리 기법을 이용하는 전역 뷰 관리자는 질의 처리 요청을 받게 된다. 이때 전역 뷰 관리자는 시그니쳐 뷰인덱스에 대한 메타데이터를 검색하여 요청된 질의에 대해 뷰인덱스가 존재하는지를 검사한다. 요청된 질의에 대해 뷰인덱스가 존재한다면, 전역 뷰 관리자는 뷰인덱스에 저장된 튜플의 시그니쳐만을

이용하여 질의 조건을 만족하는지 검사한다. 그렇지 않다면 시그니쳐를 이용하여 질의 조건을 만족하는 튜플이 있는 경우에만 LDB에 있는 실제 튜플을 검색함으로써 LDB의 입/출력을 감소할 수 있다. 또한 시그니쳐를 이용하여 질의 조건을 만족하는 튜플이 없다면, 그것은 LDB에 만족되는 실제 데이터가 없음을 의미하므로 불필요한 LDB의 입/출력을 없앨 수 있다. 그렇지 않고 요청된 질의에 대해 뷰인덱스가 존재하지 않는다면, LDB에서 해당 릴레이션과 관련된 전체 튜플을 검색한 후 질의에 대한 결과를 반환한다.

(그림 9)는 본 연구 모델에서 전역 뷰 관리자가 시그니쳐 뷰인덱스 관리기법을 이용하여 전역 질의에 대한 처리를 수행하는 알고리즘을 나타낸다.

```
ViewIndex_Management Algorithm

/* input query */
q ← query
/* check the meta-data of Signature View Index */
/* view index information exist in meta-data */
if QueryOp_i = q in meta-data for id index i = 1,···, n then
   get the set of TID in the view index of id_i
/* check the signature information about query */
   check the signature in S(TID)_j for j = 1,···, n
/* the tuple information relating query exist */
   if S(TID)_j = the attributes of q then
      access the tuple indicating TID_j in the local DB
      return the result about q
/* the tuple information relating query not exist */
   else
      return the result 'no data'
/* view index information not exist in meta-data */
else
   access the local DBs
   check the entire local DBs for finding the result data of q
   return the result about q
```

(그림 9) 시그니쳐 뷰인덱스 알고리즘

3.4 실체 뷰와 시그니쳐 뷰인덱스 혼용기법

앞에서 살펴본 바와 같이 실체 뷰 기법은 뷰를 실체화하여 저장함으로써 저장 공간을 많이 사용한다는 단점을 가지고 있다. 그러나 뷰에 대한 조건이 주어지는 경우에는 기본 릴레이션을 검색하지 않고 실체화된 내용을 이용하여 질의에 응답할 수 있으므로 신속한 응답시간을 제공하는 장점이 있다. 또한 시그니쳐 뷰인덱스 기법은 적은 저장 공간을 사용한다는 면에서 실체 뷰 기법에 비해 장점이 있지만, 기본 릴레이션의 갱신시 뷰인덱스들을 갱신하여야 하며, 또한 인덱스에는 뷰가 사용하는 기본 릴레이션의 투플 주소만을 저장하기 때문에 뷰 조건이 주어지는 경우는 실제 릴레이션을 검색하여 비교하여야 하는 문제가 있다. 이러한 기존의 뷰 관리기법들의 장단점을 분석하여 본 연구에서는 혼용 기법을 제안한다.

3.4.1 혼용 기법의 구조

본 연구에서 실체 뷰와 시그니쳐 뷰인덱스를 혼용하기 위해서는, 앞에서 정의한 실체 뷰의 메타데이터 구조[15-17]를 변형해야한다. 변형된 뷰에 대한 메타데이터 구조는 (그림 10)과 같다.

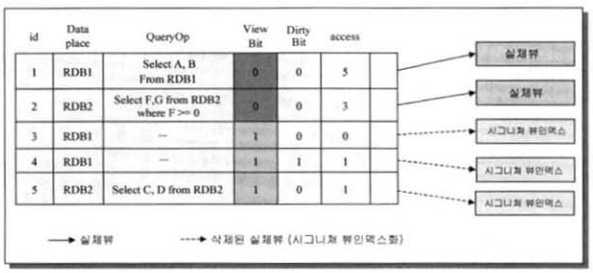
```
interface emp
{  int      id ;
   int      ReplaceDataSource ;
   String   QueryOp ;
   int      ViewBit ;
   int      DirtyBit ;
   float    NoAccess ;

   질의처리 method( )
   질의처리 method( ) ··· }
```

(그림 10) 본 연구의 혼용기법에 사용되는 메타데이터 구조

실체 뷰와 시그니쳐 뷰인덱스 기법을 혼용하여 사용하기 위해서는 실체 뷰만을 사용할 때 정의했던 메타데이터 구조를 일부 확장해야 한다. 메타데이터에서 나타내는 정보가 실체 뷰에 대한 참조인지 시그니쳐 뷰인덱스에 대한 참조인지를 구별할 수 있는 *ViewBit*를 추가한다. *ViewBit*가 0일 때는 실체 뷰에 대한 정보이며, *ViewBit*가 1일때는 시그니쳐 뷰인덱스에 대한 정보이다. 여기에서 유의할 점은 *NoAccess* 값이다. 실체 뷰의 메타데이터에서 정의하였던 *NoAccess*와 같은 개념으로 사용하나, 혼용기법의 메타데이터에서 사용하는 *NoAccess*는 시그니쳐 뷰인덱스를 위해 일부 기능이 추가된다. 먼저, 실체 뷰에 대한 접근횟수를 나타내며 더불어 시그니쳐 뷰인덱스에 대한 접근횟수도 나타낸다. 그러나, 시그니쳐 뷰인덱스는 삭제된 실체 뷰에 대한 인덱스를 나타내므로, 실체 뷰가 삭제되면서 *NoAccess* 값은 0으로 설정된다. 즉 실체 뷰에서 시그니쳐 뷰인덱스로 변환되면서 접근 횟수는 0으로 재설정 된다. 이후에 시그니쳐 뷰인덱스에 대한 접근이 발생하면 *NoAccess* 값은 증가한다. 이때 검색된 뷰의 실체화는 전역 뷰 저장소와 *NoAccess*값에 의해 결정된다. 만일 현재 전역 뷰 저장소의 공간이 비어있다면, 바로 뷰를 실체화시킨다. 그러나 전역 뷰 저장소에 빈 공간이 없다면 실체 뷰 각각의 *NoAccess* 값 중에 시그니쳐 뷰인덱스의 *NoAccess* 값보다 작은 것이 있으면, 실체 뷰를 삭제하고 지금 검색된 뷰를 실체화시킨다. 시그니쳐 뷰인덱스의 *NoAccess*값보다 작은 실체 뷰가 없을 때에는 검색된 뷰는 시그니쳐 뷰인덱스로 유지된다. 이러한 메타데이터의 필드들에 대한 기능에 대해 (그림 11)의 예로 설명한다.

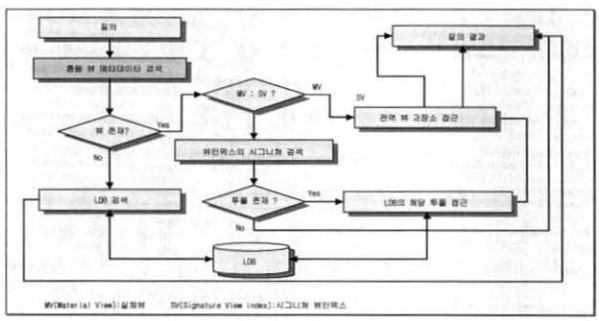
(그림 11)에서 보면, 'ID 3'의 경우는 실체화되었던 뷰가 삭제되면서 시그니쳐 뷰인덱스로 정보를 보관한다. *ViewBit*는 1로 설정되어 시그니쳐 뷰인덱스임을 나타내고, *DirtyBit*는 0으로 현재 뷰에 대한 원위치의 데이터가 변경되지 않았고, *NoAccess*는 0으로 실체 뷰에서 시그니쳐 뷰인덱스화 후 한번도 질의에 대한 접근이 없었음을 나타낸다. 'ID 4'의 경우는 시그니쳐 뷰인덱스에 대한 접근이 1번 있었음을 *NoAccess*로 알 수 있으며, 원위치의 데이터가 변경되었음을



(그림 11) 혼용 뷰 관리를 위한 메타데이터 구조의 예

*DirtyBit*값으로 나타내고 있다. 이 뷰에 대한 질의가 발생하면, 뷰는 재검색 한다. 또한 이 뷰에 대한 접근이 발생하였을 때, 검색된 뷰의 실체화 여부는 *NoAccess*값으로 판별한다. 전역 뷰 저장소에 빈 공간이 존재한다면, 뷰를 실체화시키면 된다. 그러나 전역 뷰 저장소의 빈 공간이 없다면, 현재 *NoAccess*값과 실체 뷰의 *NoAccess*값을 비교한다. 실체 뷰의 *NoAccess*값이 각각 5와 3으로 시그니쳐 뷰인덱스의 *NoAccess*값 1보다 크므로 검색된 뷰를 실체화하지 않는다. 'ID 5'의 경우는 시그니쳐 뷰인덱스에 대한 접근이 1번 있었음을 나타내며, 검색된 뷰에 대한 실체화가 이루어지지 않았음을 나타낸다.

(그림 12)는 본 연구 모델에서 전역 뷰 관리자가 혼용 뷰 관리기법을 이용한 전역 질의에 대한 처리 수행 과정을 나타낸다.



(그림 12) 뷰 관리 혼용 기법

(그림 12)의 혼용 뷰 관리 알고리즘은 (그림 13)과 같이 나타낼 수 있다.

```
Mixed_View_Management Algorithm

/* input query */
q ← query
/* check the meta-data of Mixed View */
/* view information exist in meta-data of Mixed-View */
if QueryOp_i = q in meta-data for id index i = 1,…, n then
/* test the type of view : materialized view or signature view
   index */
   if ViewBit_i = 0 then        /* materialized view */
      access Global View Repository indicating id_i
      get the materialized view data
      return the result about query q
   else                         /* signature view index */
      get the set of TID in the view index of id_i
/* check the signature information about query */
   check the signature in S(TID)_j for j = 1,…, n
/* the tuple information relating query exist */
   if S(TID)_j = the attributes of q then
      access the tuple indicating TID_j in the local DB
      return the result about query

/* the tuple information relating query not exist */
   else
      return the result 'no data'
/* view information not exist in the meta-data of Mixed-View */
   else
      access the local DBs
      check the entire local DBs for finding the result
      data of q
      return the result about q
```

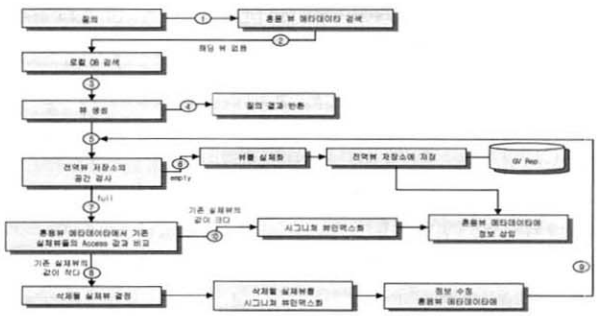
(그림 13) 혼용 뷰 관리 알고리즘

4. 뷰 관리기법간의 전환 절차

본 연구에서 사용하는 뷰 관리 기법의 실체 뷰와 시그니쳐 뷰인덱스간의 관계를 정의할 필요가 있다.

4.1 생성뷰의 실체화/시그니쳐 뷰인덱스화 과정

처리해야 할 질의에 대한 데이터가 존재하지 않을 때, LDB에서 질의 조건에 맞는 해당 데이터를 검색하여 뷰를 생성한다. 이때 생성된 뷰를 관리하는 기법은 앞에서 설명한 실체 뷰/시그니쳐 뷰인덱스 혼용기법을 기반으로 다음 (그림 14)와 같은 관리상 절차를 정의한다.



(그림 14) 생성뷰 관리 절차

(그림 14)의 절차를 살펴보면 다음과 같다.

① 질의가 들어오면 혼용 뷰 메타데이터를 검색하여 질의에 대한 뷰가 존재하는지의 여부를 판별한다.

② 질의에 맞는 뷰가 존재하지 않으면 LDB 전체를 검색한다.

③ LDB에서 질의 조건에 맞는 뷰를 생성한 후,

④ 질의 결과를 반환하고,

⑤ 전역 뷰 관리자는 생성된 뷰의 관리기법을 결정하게 된다. 먼저 전역 뷰 저장소의 저장 공간을 확인하여,

⑥ 빈 공간이 존재하면 생성된 뷰를 실체화시켜 실체 뷰

로 만들고 전역 뷰 저장소에 저장한 후, 혼용 뷰 메타데이터에 새로운 실체 뷰의 정보를 삽입시킨다.

⑦ 반면에 전역 뷰 저장소에 빈 공간이 없을 때에는 두 가지 관리기법이 존재하게 된다.

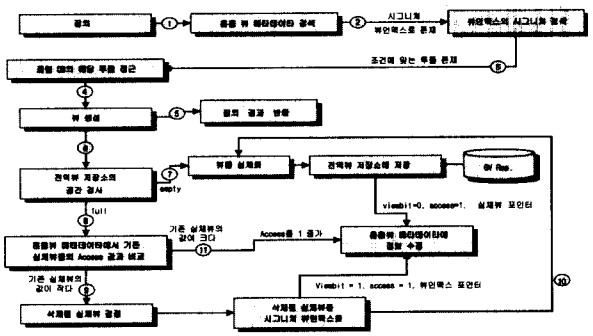
⑧ 먼저 혼용 뷰 메타데이터에 있는 기존의 실체 뷰들의 정보를 이용하여 *NoAccess*값을 검색하여 1이하의 값을 가지는 실체 뷰가 존재한다면, 이것은 전역 뷰 저장소에 있는 실체 뷰가 생성이후에 재 질의되지 않은 뷰라는 것을 의미하므로 전역 뷰 저장소에서 삭제될 실체 뷰로 판별된다. 삭제될 실체 뷰가 결정되면, 전역 뷰 저장소에서 삭제함과 동시에 이 실체 뷰를 시그니쳐 뷰인덱스화 시키고 혼용 뷰 메타데이터에 뷰 종류와 포인터등의 정보를 수정시킨다.

⑨ 그리고 생성된 뷰를 전역 뷰 저장소에 저장한다.

⑩ 두 번째로 실체 뷰의 *NoAccess*값들이 모두 1보다 크다면, 이는 기존의 실체 뷰들이 1번이상 재질의 되었음을 의미하므로, 생성된 뷰는 시그니쳐 뷰인덱스화만 시켜놓는다. 시그니쳐 뷰인덱스화된 뷰에 대한 정보는 혼용뷰 메타데이터에 삽입한다.

4.2 시그니쳐 뷰인덱스의 실체 뷰화 과정

앞에서 LDB에 대한 질의에 의해 생성된 뷰가 실체 뷰나 시그니쳐 뷰인덱스화되는 과정과 실체 뷰가 전역 뷰 저장소에서 삭제되어 시그니쳐 뷰인덱스화되는 과정을 설명하였다. 또 다른 고려사항으로 시그니쳐 뷰인덱스화 되어있던 뷰를 실체화하는 과정은 (그림 15)로 나타낸다.



(그림 15) 뷰인덱스의 실체 뷰화 과정

(그림 15)의 절차를 살펴보면 다음과 같다.

① 질의가 들어오면 혼용 뷰 메타데이터를 검색하여 질의에 대한 뷰가 존재하는지의 여부를 판별한다.

② 질의에 맞는 뷰가 시그니쳐 뷰인덱스 형태로 존재하면, 질의 조건에 맞는 뷰인덱스의 시그니쳐를 검색한다.

③ 뷰인덱스의 시그니쳐 검색으로 조건에 맞는 튜플이 존재하면, LDB의 해당 튜플에 접근한다.

④ 접근된 해당 튜플들로 뷰를 생성한다.

⑤ 생성된 뷰에 의하여 질의 결과를 반환하고,

⑥ 전역 뷰 관리자는 뷰인덱스화 되어있는 뷰의 실체화 여부를 결정한다. 이를 위해, 먼저 전역 뷰 저장소의 저장 공간의 여유를 판별하고 전역 뷰 저장소에 있는 기존의 실체 뷰들과의 접근 횟수의 비교가 필요하다.

⑦ 전역 뷰 저장소에 여유 공간이 존재하면, 뷰인덱스에 의해 접근되어 생성된 뷰를 실체화시켜 실체 뷰를 만들고 전역 뷰 저장소에 저장한 후 혼용 뷰 메타데이터에 이 뷰에 대한 정보를 수정한다. 뷰에 대한 종류는 시그니쳐 뷰인덱스에서 실체 뷰로 변경되며(*ViewBit*가 1에서 0으로), *NoAccess*도 1로 설정되고, 뷰인덱스를 가리키던 포인터도 전역 뷰 저장소의 실체 뷰 포인터로 변경된다.

⑧ 만일 전역 뷰 저장소에 여유 공간이 없으면, 생성뷰의 실체화/뷰인덱스화 과정과 유사한 두 가지 관리기법이 정의된다.

⑨ 혼용 뷰 메타데이터에 있는 기존의 실체 뷰들의 정보를 이용하여 *NoAccess* 값을 현재 질의된 뷰의 *NoAccess*값과 비교한다. 만일 기존 실체 뷰의 NoAccess값이 현재 뷰의 *NoAccess*값보다 작다면, 이것은 전역 뷰 저장소에 있는 실체 뷰보다 시그니쳐 뷰인덱스화된 뷰가 재질의되는 경우가 더 많다는 것을 의미한다. 따라서, 전역 뷰 저장소에 있는 실체 뷰를 시그니쳐 뷰인덱스화시키고, 혼용 뷰 메타데이터에 있는 삭제된 실체 뷰의 정보를 수정한다. 뷰에 대한 종류는 실체 뷰에서 시그니쳐 뷰인덱스로 변경되며(*ViewBit*가 0에서 1로), *NoAccess*도 1로 설정되고, 실체 뷰를 가리키던 포인터는 뷰인덱스를 가리키는 포인터로 변경된다.

⑩ 현재 질의된 뷰는 실체화시킨다. 물론 이 과정에서 뷰에 대한 혼용 뷰 메타데이터 정보는 수정한다(⑦과정을 수행).

⑪ ⑧번의 비교에서 만일 현재 질의되어 생성된 뷰의 *NoAccess*값이 전역 뷰 저장소에 있는 기존의 실체 뷰들의 *NoAccess*값보다 작다면, 이는 실체 뷰들의 재질의 빈도수가 더 높음을 의미하므로, 현재 생성된 뷰는 실체화시키지 않는다. 그러나, 혼용 뷰 메타데이터에 있는 *NoAccess* 정보는 1증가된 값으로 수정한다.

5. 결 론

본 연구는 다양한 LDBS에 대한 통합 질의환경(MDBS)에서 전역 사용자의 질의 성능 향상을 위한 방법으로 혼용 뷰 관리기법을 제안하였다. 그동안 뷰와 관련된 연구로서 실체 뷰 기법과 시그니쳐 뷰인덱스 기법이 제시되었다. 실체 뷰 기법은 빠른 액세스 시간을 얻을 수 있다는 장점과 뷰를 유지하는데 많은 저장 비용이 필요하다는 단점을 가지고 있다. 시그니쳐 뷰인덱스 기법은 적은 저장비용이 소요되지만 많은 액세스 시간이 소요되는 단점이 있다.

그러나 본 연구에서는 실체 뷰와 시그니쳐 뷰인덱스 혼용 기법을 사용함으로써 각 기법의 장점을 채택하여 뷰에

대한 질의 처리시 디스크 입/출력을 감소시킨다. LDB에 접근하는 횟수를 줄이고, 분할된 부분 질의들의 연산 결과를 실체 뷰와 시그니쳐 뷰인덱스 혼용 기법으로 관리함으로써 질의 수행 횟수를 감소시킨다. 또한 메타데이터를 이용하여 자주 참조되는 실체 뷰의 유지를 증가시켜서 다중 질의 처리를 더욱 빠르게 향상시킨다.

앞으로의 연구방향은 혼용 뷰 관리기법을 적용한 MDBMS (Multi-Database Management System)를 설계 및 구현하고 실체 뷰 기법과 시그니쳐 뷰인덱스 기법에 비해 어느 정도 우수한 성능을 나타내는가를 측정할 수 있는 수행검증에 관한 연구를 계속하고자 한다.

참 고 문 헌

[1] A. Dogac, C. Dengi, E. Kilic, G. Ozhan, F. Ozcan, S. Nural, C. Evrendilek, U. Halici, B. Arpinar, P. Koksal, N. Kesim, S. Mancuhan, "METU Interoperable Database System," ACM SIGMOD Record, Vol.24, No.3, September, 1995.

[2] E. Kilic, G. Ozhan, C. Dengi, N. Kesim, P. Koksal, A. Dogac, "Experience in Using CORBA for a Multidatabase Implementation," Proc. Of 6th International Workshop on Database and Expert System Applications, London, Sept., 1995.

[3] W. Litwin and A. Abdellatif, "Multidatabase Interoperability," 1986.

[4] 김지운, 김보경, 이근철, 문강식, 이전영, "CORBA 기반 멀티 데이터베이스 관리 시스템의 설계", Available from http://iislab.postech.ac.kr/mrg/publish/xwidd/kiss.doc.

[5] 변광준, "CORBA 기반의 이질적인 데이터베이스 통합", 한국정보과학회지, 제17권 제7호, 1999.

[6] A. Segev and J. Park, "Updating Distributed Materialized Views," IEEE Trans. on Knowledge and Data Engineering, Vol.1, No.2, 1989.

[7] J. Blakeley et al., "Efficiently Updating Meterialized Views," in ACM SIGMOD Conference, pp.61-71, 1986.

[8] N. Goyal et al., "Preliminary Report on View Materialization in GUI Programming," Proceeding of the Workshop on Materialized Views : Techniques and Applications, pp.56-64, June, 1996.

[9] S. Chaudhuri, R. Krishnamurthy, S. Potaminanos and K. Shim, "Materialized Views Optimizing Queries with Materialized Views," IEEE, 1995.

[10] D. Lee and C. Leng, "Partitioned Signature File : Design issues and performance evaluation," ACM Trans. on Office Information Systems, Vol.7, pp.212-223, April, 1989.

[11] F. Rabitti and P. Savino, "Image Query Processing Based on Multi-level Signatures," in Proc. of ACM SIGIR Conf. on Research and Development in Information Retrieval, pp.305-314, October, 1991.

[12] H. S. Yong, S. Lee, and H. J. Kim, "Applying Signatures for Forward Traversal Query Processing in Object-Oriented Databases," in International Conference on Data Engineering, pp.518-525, Feb., 1994.

[13] P. Zezula et al., "Dynamic Partitioning of Signature Files," ACM Trans. on Information Systems, Vol.9, No.4, pp.336-369, 1991.

[14] 용환승, "관계 데이터베이스에서 시그니쳐를 이용한 뷰인덱스 기법", 정보처리논문지, 제3권 제4호, 1996.

[15] N. Roussopoulos, "An Incremental Access Method for View Cache : Concept, Algorithms and Cost Analysis," ACM Trans. on Database Systems, Vol.16, No.3, pp.535-563, 1991.

[16] 문창주, 이선정, 박성공, 백두권, "CORBA 환경에서 메타데이터를 이용한 관계형 데이터베이스와 파일 시스템간의 상호운용성 해결 방안", 한국정보과학회 '98 봄 학술발표논문집(B), 제25권 제1호, 1998.

[17] 이선정, 문창주, 박성공, 백두권, "효과적 질의 처리를 위한 메타데이터 기반 실체 뷰 관리 기법", 한국정보과학회 가을 학술발표논문집, Vol.25, No.2, 1998.

이 승 용

e-mail : dragon@wkhc.ac.kr

1986년 원광대학교(공학사)

1990년 조선대학교 대학원 전산기공학과 (공학석사)

2002년 원광대학교 대학원 컴퓨터공학과 (공학박사)

1991년~현재 원광보건대학 컴퓨터응용개발과 교수

관심분야 : 멀티 데이터베이스, 분산시스템, 데이터통신, 인터넷 통신

김 명 희

e-mail : hee@wdu.ac.kr

1993년 원광대학교 컴퓨터공학과(공학사)

1996년 원광대학교 대학원 컴퓨터공학과 (공학 석사)

2001년 원광대학교 대학원 컴퓨터공학과 (공학 박사)

2001년~2002년 원광대학교 전기전자·정보공학부 BK21 계약교수

2002년~현재 원광디지털대학교 게임소프트웨어학과 전임강사

관심분야 : 분산운영체제, 분산 객체모델, 멀티데이터베이스

주 수 종

e-mail : scjoo@wonkwang.ac.kr

1986년 원광대학교 전자계산공학과(학사)

1988년 중앙대학교 대학원 컴퓨터공학과 (석사)

1992년 중앙대학교 대학원 컴퓨터공학과 (박사)

1993년~1994년 미국 Univ. of Massachusetts at Amherst, 전기 및 컴퓨터공학과, Post-Doc.

1990년~현재 원광대학교 컴퓨터공학과 교수

2002년~현재 미국 Univ. of California at Irvine 전기공학 및 컴퓨터과학과 연구교수

관심분야 : 분산실시간 컴퓨팅, 분산 객체 모델, 시스템 최적화, 멀티미디어 데이터베이스