관계 데이타베이스에서 시그니쳐를 이용한 뷰인덱스 기법

용 환 승†

요 약

뷰 인덱스 기법은 뷰에 대한 질의를 신속히 처리하기 위한 기법으로 제안되었다. 시그니쳐 참조 기법은 참조 관계를 가지는 데이타에서 피참조 데이타에 대한 시그니쳐를 포인터와 함께 저장한다. 피참조 데이타에 대한 조건을 가지는 질의가 주어지는 경우 피참조 데이타의 시그니쳐를 이용하여 조건 검사를 하므로써 디스크 접근 횟수를 감소시킨다.

본 논문은 시그니쳐 참조 기법을 뷰인덱스에 적용하여 투플식별자와 그 투플에 대한 시그니쳐를 함께 저장하는 시그니쳐 뷰인덱스(signature view index) 기법을 제안하였다. 이 기법을 사용하므로서 뷰에 대한 조건을 가지는 질의가 주어지면 뷰인덱스에 저장된 시그니쳐를 이용 조건을 검사하여 만족하는 투플만을 검색하도록 한다.

View Index Technique using Signatures in Relational Databases

Hwan-Seung Yong†

ABSTRACT

View index techniques are proposed to process queries on views. Signature referencing techniques keep pointers with signatures of the referred object when there is any reference relationship between objects. When queries having conditions on referred object are given, disk I/Os can be reduced by checking conditions using stored signatures.

Signature view index techniques proposed in this paper apply signature referencing techniques to view index by keeping not only tuple identifiers but also signatures for the tuples. When queries having conditions on views are given, we can retrieve only tuples from a view index which satisfy given conditions by checking those with stored signatures.

1. 서 론

관계 데이타베이스 모델에서는 뷰(view)의 정의를 통해 전체 사용자들이 바라보는 데이타베이스에 대한 내부 스키마와 각 사용자 또는 응용에 따른 외부 스키마를 분리 제공하므로서 데이타 독립성을 제공한다.

뷰에 대한 질의 처리는 기본 릴레이션(base relation)에 대한 질의로 변경되어 처리되어야 하며 이 과정에서 많은 입/출력이 요구되어 처리 속도의 저하를 가져오게 된다. 이 문제를 해결하기 위해 여러가지 방법이 제안되었는 데 대표적인 것으로 뷰구체화 (view materialization) [1, 2]와 뷰인덱스 (view index) [3, 4] 기법을 들 수 있다. 첫째 기법의 문제점은 뷰를 구체

* 이 논문은 1994년도 한국학술진흥재단의 신진연구인력 연구비에 의해 연구되었음
† 정 회 원:이화여자대학교 전자계산학과
논문접수:1996년 1월 4일, 심사완료:1996년 3월 26일

화하여 저장하므로써 저장공간을 많이 사용한다는 것이다. 그러나 뷰에 대한 조건이 주어지는 경우에는 베이스 릴레이션을 검색하지 않고 구체화된 내용을 이용하여 질의에 응답할 수 있으므로 신속한 응답시간(response time)을 제공하는 장점이 있다.

뷰인덱스 기법[4]은 적은 저장공간을 사용한다는 면에서 뷰구체화기법에 비해 장점이 있지만 베이스 릴레이션의 갱신시 뷰인덱스들을 갱신하여야 하며 또한 인덱스에는 뷰가 사용하는 베이스 릴레이션의 투플 주소만을 저장하기 때문에 뷰 조건이 주어지는 경우는 실제 릴레이션을 검색하여 비교하여야 하는 문제가 있다.

시그니쳐(signature)는 집합값에 대한 일종의 코드로 적은 저장비용을 사용하여 주어진 정보를 요약한 것으로 볼 수 있다. 시그니쳐를 검사하면 주어진 값이 해당 시그니쳐와 매치(match)가 되는가를 파악할 수 있는 데 이 특징을 이용하여 역 인덱스 (inverted index)와 별도의 접근 기법으로 사용된다. 그러나 시그니쳐와의 매치는 완전하지 않다는 단점이 있다. 즉 주어진 값이 있는 경우에는 매치가 반드시 일어나지만 없는 경우에도 매치가 일어나는 경우가 발생한다. 이와 같은 경우를 매치 오류 (False Drop)라고 한다. 매치 오류 확률 F_d는 다음과 같이 계산된다.

$$F_d = \frac{\text{시그니쳐 매치 오류에 의한 레코드 수}}{\text{실제 조건을 만족하지 않는 레코드 수}}$$

시그니쳐(signature)를 이용한 접근 기법에 대한 연구는 오래전부터 수행되어 왔으며 특히 최근 들어 다매체 (multimedia) 데이타와 새로운 응용의 필요성은 이 기법의 적용 가능성을 확대시키고 있다 [5, 6, 7, 8, 9, 10]. 그러나 시그니쳐를 이용한 연구는 주로 시그니쳐 레코드로 구성되는 화일을 생성하여 적은 저장비용으로 검색을 지원하는 기법으로 사용되어 왔다.

시그니쳐 화일을 기반으로한 인덱스 응용 분야 [5]에서는 텍스트 데이타베이스에서 인덱스로 사용하는 것을 제안하였는 데, 적은 인덱스 저장공간을 필요로 하는 점이 특히 우수하다고 제시하고 있다. 시그니쳐의 크기 b는 객체가 속한 클래스마다 다를 수 있다. 일반적으로 b는 한 시그니쳐에 해싱되는 애트리뷰트 값들의 갯수와 시그니쳐의 매치 오류 확률에 의해 정해지며 계산식은 다음과 같다 [9].

애트리뷰트의 수를 nattr라고 하고 매치 오류의 확률을 F_d라고 하면

$$b = (1/\log 2)^2 \cdot nattr(1/F_d)$$

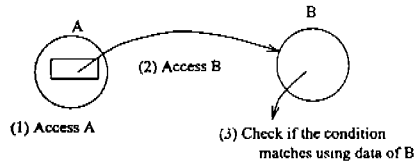
최근에는 제시된 시그니쳐 참조 (reference) 기법 [11]은 기존의 시그니쳐 화일 기법에서와 같이 시그니쳐 화일을 별도로 구성하여 질의 처리시 전체 시그니쳐 화일을 검색하는 방식이 아니다. 이 기법은 객체지향 데이타베이스의 예와 같이 객체 간에 애트리뷰트를 통한 참조 관계 (reference relationship)가 있거나 또는 중첩 릴레이션 데이타베이스에서 서브릴레이션을 포인터를 통해 접근 (access)하여야 하는 포인터 관계에 적용한다.

이 기법은 시그니쳐 객체 또는 임의의 집합값을 가지는 데이타에 대해 포인터나 기타 방법으로 참조 (reference) 관계가 형성되어 있는 경우 피참조 객체의 시그니쳐를 생성하여 참조 객체에 미리 저장하여 놓는다. 그러므로 참조 관계를 통해 피참조 객체를 검색할 때에 피참조 객체가 특정 조건을 만족하는 지 여부에 따라 실제 데이타 검색을 수행하게 된다.

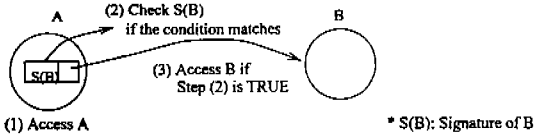
객체 A가 다른 객체 B를 애트리뷰트의 값을 통해 참조한다 (refer)는 것은 객체 A에 대한 연산과정에서 이 참조관계를 통해 객체 B의 정보를 검색한다는 것을 뜻한다. 객체 지향 데이타베이스에서 참조 관계를 통한 객체들의 검색 과정은 피참조 객체를 검색하기 위해 별도의 입/출력을 필요로 한다. 이러한 검색 과정은 일반적으로 참조 객체(A)에 피참조 객체(B)에 대한 조건 (중첩 조건:nested predicate)이 주어지는 경우 이 조건을 처리하기 위해 수행된다. 만일 참조 객체에 피참조 객체에 대한 정보가 저장되어 있다면 미리 조건을 검사할 수 있으므로 불필요한 검색을 하지 않아도 되어 신속한 연산 처리가 가능하게 된다. (그림 1)에는 참조 관계를 통한 객체 검색과정을 비교하여 보여주고 있다.

이 때 기존의 참조 관계를 통한 검색 과정은 다음과 같다.

1. 먼저 참조하는 객체 A를 검색한다.
2. 객체 A에 있는 피참조 객체 포인터를 통해 피참조 객체 B를 검색한다.



(a) 기존의 참조 객체 검색 과정



(b) 시그니쳐를 사용한 참조 객체 검색 과정

(그림 1) 참조 관계를 통한 객체 검색 과정 비교
(Fig. 1) Comparison of an object access procedure using a reference relationship.

3. 객체 B의 데이타를 가지고 조건식을 검사한다.

한 객체에 속한 원자값 애트리뷰트로 정의된 모든 애트리뷰트의 값들을 가지고 생성한 시그니쳐를 객체 시그니쳐라고 한다. 피참조 객체의 객체 시그니쳐를 참조 객체에 저장하는 기법을 사용하는 경우 피참조 객체에 대한 조건들이 주어졌을 때 검색 과정은 다음과 같다.

1. 먼저 참조 객체 A를 검색한다.
2. 객체 A에 저장된 피참조 객체의 시그니쳐를 가지고 조건식을 검사한다.
3. 조건이 만족되는 경우에는 피참조 객체 B를 검색한다.

뷰인덱스를 살펴보면 뷰인덱스에는 투플에 대한 지시자 (tuple identifier, TID)를 통해 특정 투플을 참조하고 있다. 그러므로 본 논문에서는 시그니쳐 참조 기법을 뷰인덱스에 적용하여 뷰인덱스에 투플에 대한 TID뿐만아니라 해당 투플에 대한 시그니쳐를 생성하여 저장할 수 있다. 그러므로 저장된 시그니쳐를 이용하므로써 조건이 주어진 질의를 뷰구체화기법에서와 같이 기본 릴레이션을 검색하지 않고 처리할 수 있는 기법을 제안하였다.

본 논문의 구성은 다음과 같다. 2장에서는 뷰인덱스 기법에 대해 구체적으로 설명하고 문제점을 제시한다. 그리고 3장에서는 시그니쳐 참조 기법을 이용한 시그니쳐 뷰인덱스 기법을 제안한다. 4장에서는 제안된 기법에 대한 비교 평가를 위해 비용식을 산출한 후 비교 결과를 보여주며, 5장에서 결론 및 앞으로의 연구 방향에 대해 기술했다.

2. 뷰인덱스 기법과 문제점

뷰를 신속히 처리하기 위한 기법으로 뷰 구체화(view materialization) [1, 2]와 뷰인덱스 기법[4]이 있다. 뷰 구체화 기법[1, 2]에서는 베이스 릴레이션이 갱신되면 구체화된 뷰도 부수적으로 갱신되어야 한다. 베이스 릴레이션의 갱신에 따라 갱신되어야 할 뷰의 선택과 구체화된 뷰 전체를 갱신하지 않고 이전 내용에서 변화된 부분만을 최소한으로 갱신하는 차등 갱신(differential update) 알고리즘에 대해서는 이미 제안되어 있다 [2].

뷰 구체화 기법의 문제점은 뷰를 구체화하여 저장하므로써 저장공간을 많이 사용한다는 것이다. 그러나 뷰에 대한 조건이 주어지는 경우에는 베이스 릴레이션을 검색하지 않고 구체화된 내용을 이용하여 질의에 응답할 수 있으므로 신속한 응답시간(response time)을 제공하는 장점이 있다.

TID	SNO	SNAME	DEPT	AGE	PHONE
$STID_1$	S100	Kim	Computer	20	880-7299
$STID_2$	S200	Lee	Mechanics	21	529-1087
$STID_3$	S300	Park	English	22	222-3423
$STID_4$	S400	Choi	Computer	23	333-4444

(a) Student 릴레이션

TID	SNO	CNAME	GRADE
$CTID_1$	S100	Compiler	A-
$CTID_2$	S200	Fluids	B+
$CTID_3$	S300	Language	C0
$CTID_4$	S400	Poem	B-

(b) Course 릴레이션

(그림 2) 예제 데이터베이스
(Fig. 2) Example database.

뷰인덱스 기법[4]은 적은 저장공간을 사용한다는 면에서 뷰구체화기법에 비해 장점이 있지만 베이스 릴레이션의 갱신시 뷰인덱스들을 갱신하여야 하며 또한 인덱스에는 뷰가 사용하는 베이스 릴레이션의 투플 주소만을 저장하기 때문에 뷰 조건이 주어지는 경우는 실제 릴레이션을 검색하여 비교하여야 하는 문제가 있다.

(그림 2)에는 **Student**와 **Course** 릴레이션의 예가 나타나 있다. 이 릴레이션에 대해 다음과 같은 뷰 **CompStud**가 정의되었다고 하자.

```
Create View CompStud
        As Select SNAME, AGE, PHONE
           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 **CompStud**에 주어진다고 하자.

```
Select *
From CompStud
Where AGE = 20
```

뷰인덱스를 사용하지 않는 경우에는 **Student** 릴레이션의 전체 투플에 대해 검색하여야 하지만 뷰인덱스가 사용되는 경우에는 뷰인덱스에 저장된 두개의 TID에 해당하는 투플만을 검색하여 AGE 애트리뷰트의 값이 20인지 여부를 검사하게 된다. 그러나 다음절에서 설명하는 시그니쳐 뷰인덱스 기법은 시그니쳐 참조기법을 사용하여 뷰인덱스에 투플의 TID만이 아니고 시그니쳐를 함께 저장하는 경우에는 뷰인덱스에 저장된 투플의 시그니쳐만을 이용하여 'AGE = 20'이라는 조건을 만족하는 지 검사할 수 있으며 만족하는 경우에만 실제 투플을 검색하므로써 입/출력을 감소할 수 있게 된다. 그러므로 뷰인덱스에서 시그니쳐를 비교하여 조건을 만족하는 $STID_1$ 투플만을 검색하게 된다.

3. 시그니쳐 뷰인덱스(SVI) 기법

이 장에서는 시그니쳐 뷰인덱스를 단일 릴레이션과 조인에 대한 뷰에 대해 각각 정의한 후 이 인덱스를 사용한 검색과 갱신 연산 알고리즘에 대해 기술한다. 그리고 실제 활용예를 들어 설명한다.

3.1 시그니쳐 뷰인덱스의 정의

3.1.1 단일 릴레이션에서의 SVI의 정의

뷰 SV를 단일 릴레이션에서 셀렉션과 프로젝션 연산에 의해 정의된 뷰라고 하자. 사용할 기호는 다음과 같다.

- Attr(SV): 뷰 SV의 정의에서 사용된 목표 애트리뷰트 리스트
- Cond(SV): 뷰 SV를 정의하는 데 사용된 조건식들

예를 들어 뷰 **CompStud**에 대해 위의 기호에 따른 예는 아래와 같다.

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

정의 1: 릴레이션 R에 정의된 뷰 SV에 대한 뷰인덱스는 릴레이션 R에서 Cond(SV)를 만족하는 투플들의 TID의 집합 $\{TID_i\}$로 정의된다.

정의 2: 릴레이션 R에 정의된 뷰 SV에 대한 시그니쳐 뷰인덱스는 집합 $\{(TID_i, S(TID_i))\}$이다. 이 때 TID_i는 릴레이션 R에서 Cond(SV)를 만족하는 투플들의 TID이고 $S(TID_i)$는 해당 투플의 Attr(SV)에 속하는 애트리뷰트의 값들로 만든 시그니쳐이다.

3.1.2 조인뷰에서의 SVI의 정의

뷰는 다음과 같이 두개 이상의 릴레이션을 조인하여 정의될 수 있다. 다음은 **Student**와 **Course** 릴레이션에 대한 조인을 이용하여 뷰 **JoinView**의 정의 예를 보여주고 있다.

```
Create View JoinView
        As Select S.SNAME, S.DEPT, C.CNAME,
        C.GRADE
           From Student S, Course C
```

Where S. SNO = C.SNO
And S.DEPT = 'Computer'
And C.Grade = 'A-'

뷰 JV가 릴레이션 R1과 R2를 조인하여 정의되었다고 하자. 정의에서 사용되는 기호는 다음과 같다.

- $Attr_1$(JV): 뷰 JV의 정의에서 사용된 릴레이션 R1의 목표 애트리뷰트들
- $Attr_2$(JV): 뷰 JV의 정의에서 사용된 릴레이션 R2의 목표 애트리뷰트들
- $Cond_1$(JV): 뷰 JV의 정의에서 사용된 릴레이션 R1에 대한 조건식들
- $Cond_2$(JV): 뷰 JV의 정의에서 사용된 릴레이션 R2에 대한 조건식들

위에서 정의된 뷰 **JoinView**에 대해 기호를 사용한 예는 아래와 같다.

- $Attr_1$(**JoinView**): {SNAME, DEPT}
- $Attr_2$(**JoinView**): {CNAME, CRADE}
- $Cond_1$(**JoinView**): S.DEPT = 'Computer'
- $Cond_2$(**JoinView**): C.Grade = 'A-'

정의 3: 두 릴레이션의 조인뷰 JV에 대한 뷰인덱스는 $\{(TID_i, TID_j)\}$이다. 이때

- TID_i는 $Cond_1$(JV)을 만족하는 릴레이션 R1의 TID이고,
- TID_j는 TID_i와 조인 조건, $Cond_2$(JV)을 모두 만족하는 릴레이션 R2의 TID이다.

뷰 JV에 시그니쳐 뷰인덱스는 $\{(TID_i, S(TID_i), TID_j, S(TID_j))\}$이다.
이때

- TID_i는 $Cond_1$(JV)을 만족하는 릴레이션 R1의 TID이고,
- $S(TID_i)$는 TID_i에서 $Attr_1$(JV)에 속하는 애트리뷰트들에 대한 시그니쳐이고,
- TID_j는 TID_i와 조인 조건, $Cond_2$(JV)을 만족하는 릴레이션 R2의 TID이고,
- $S(TID_j)$는 TID_j에서 $Attr_2$(JV) 애트리뷰트들에 대한 시그니쳐이다.

뷰 **CompStud**에 대한 시그니쳐 뷰인덱스의 생성 예는(그림 3)의 (a)에 나타나 있다.

(그림 2)의 릴레이션에 대해서 조인뷰 **JoinView**에 대한 시그니쳐 뷰인덱스를 생성한 예는 (그림 3)의 (b)와 같다. 이 예를 보면 **Student** 릴레이션의 시그니쳐를 생성할 때 뷰의 정의에서 사용된 애트리뷰트들인 SNAME, DEPT의 값만을 사용하였다는 것을 알 수 있다.

TID	시그니쳐
$STID_1$	S(Kim, 20, 880-7299)
$STID_4$	S(Choi, 23, 333-4444)

(a) 뷰 CompStud의 시그니쳐 뷰인덱스

Student TID	Student 시그니쳐	Course TID	Course 시그니쳐
$STID_1$	S(Kim, Computer)	$CTID_1$	S(Compiler, A-)
$STID_2$	S(Lee, Mechanics)	$CTID_2$	S(Fluids, B+)
$STID_3$	S(Park, English)	$CTID_3$	S(Language, C0)
$STID_4$	S(Choi, Computer)	$CTID_4$	S(Poem, B-)

(b) 조인뷰 JoinView의 시그니쳐 뷰인덱스

(그림 3) 시그니쳐 뷰인덱스 생성 예
(Fig. 3) Signature view index examples.

조인뷰에 대해 시그니쳐를 생성할 때는 조인에 참여한 릴레이션별로 별도의 시그니쳐를 만들어 뷰인덱스에 저장하는 기법을 사용하였는 데 시그니쳐를 하나로 묶어서 생성하는 것도 고려할 수 있다. 그러나 이러한 방식의 문제는 갱신에 따른 연산 부담이 크다는 것이다. 즉 기본 릴레이션이 갱신되는 경우 조인 뷰인덱스의 시그니쳐를 갱신하여야 하는 데 두 릴레이션의 애트리뷰트들을 하나의 시그니쳐로 만들었을 경우에는 조인에 참여한 다른 릴레이션의 투플도 검색하여야만 검색이 가능하게 되므로 입/출력 부담이 발생하게 된다. 그러므로 조인에 참여한 릴레이션에 대해 별도로 시그니쳐를 만드는 것이 바람직하다.

3.2 SVI의 연산 알고리즘

3.2.1 검색 연산

뷰에 대해 조건을 가지는 질의문이 주어지는 경우 질의처리 과정을 살펴보면 다음과 같다.

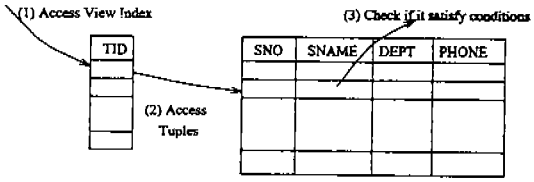
먼저 뷰인덱스 기법에서는

1. 뷰인덱스를 검색하여 판독할 TID의 집합을 구한다.
2. TID 집합에 속한 투플들을 디스크로부터 검색한다.
3. 검색된 투플이 조건을 만족하는 지 검사한다.

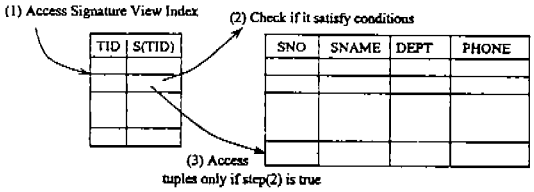
그러나 시그니쳐 뷰인덱스 기법에서는

1. 시그니쳐 뷰인덱스를 검색하여 TID와 해당 시그니쳐인 S(TID)의 집합을 구한다.
2. 시그니쳐 S(TID)를 이용하여 투플이 조건을 만족하는 지 검사한다.
3. 위의 조건이 만족하는 경우에만 해당 투플들을 디스크로부터 검색한다.

이 과정이 (그림 4)에 나타나 있다.



(a) 뷰인덱스의 경우



(b) 시그니쳐 뷰 인덱스의 경우

(그림 4) 뷰에 대해 조건을 가지는 질의 처리 과정 비교
(Fig. 4) Comparison of query processing procedure when conditions on view are given.

3.2.2 갱신 연산

기본 릴레이션의 값이 변경될 때 관련된 뷰인덱스를 유지하기 위해 최소한의 뷰인덱스를 갱신하는 알고리즘에 대해서는 많은 연구가 이루어졌다 [2, 4].

시그니쳐 뷰인덱스 기법에서의 갱신 연산은 뷰인덱스에서의 갱신과 동일한 알고리즘을 사용하면 된다. 단지 투플지시자를 뷰인덱스에 추가하거나 삭제하는 경우에 해당 투플에서 뷰인덱스에 정의된 애트리뷰트들로 구성된 시그니쳐를 별도로 저장하는 과정이 필요하다. 그러므로 시그니쳐를 갱신을 통하여 유지하는 데 따르는 별도의 부담은 크지 않다.

시그니쳐 뷰인덱스가 갱신되는 경우와 방법은 다음과 같다. 예로 **Student** 릴레이션과 뷰 **CompStud**가 시그니쳐 뷰인덱스에 의해 구성되어 있다고 하자.

첫째는 기본 릴레이션에 새로운 투플의 삽입으로 인하여 뷰에 포함되어야 하는 경우이다. 예를 들어 삽입되는 **Student** 릴레이션에서 새로운 투플의 DEPT 애트리뷰트의 값이 'Computer'로서 새로운 학생의 정보가 입력되는 경우로 이 때는 시그니쳐 뷰인덱스에 새로운 TID와 뷰정의에 사용된 애트리뷰트로 만든 시그니쳐인 S(TID)를 추가한다.

둘째는 기본 릴레이션에서 투플이 삭제되어 뷰에서 제거해야 하는 경우로 **Student** 릴레이션에서 삭제되는 투플의 DEPT 애트리뷰트의 값이 'Computer'로서 그 학과에 속한 학생의 정보가 삭제되는 경우에 해당한다. 이때는 시그니쳐 뷰인덱스에서 그 투플의 TID와 S(TID)를 함께 삭제한다.

세째는 기존의 투플값이 변경되어 뷰의 정의에 포함되지 않던 투플이 새로이 뷰정의에 포함되는 경우이다. 즉 **Student** 릴레이션에서 한 학생의 DEPT 애트리뷰트 값이 'Computer'로 변경되면 이 투플의 TID와 S(TID)를 시그니쳐 뷰인덱스에 추가한다.

네째는 기존의 투플값이 변경되어 뷰의 정의에 포함되던 투플이 새로이 뷰정의에 포함되지 않는 경우이다. 즉 **Student** 릴레이션에서 한 학생의 DEPT 애트리뷰트 값이 'Computer'에서 'Mechanics'로 변경되므로써 뷰정의에서 제거되어야 한다. 이때는 시그니쳐 뷰인덱스에서 이 투플의 TID와 S(TID)를 삭제한다.

마지막으로 기존의 투플값이 변경되는 데 뷰의 정의에는 계속 포함되고 변경된 애트리뷰트가 뷰의 정의에 사용되는 경우이다. 이때는 시그니쳐 뷰인덱스

에서 TID는 변동이 없고 단지 S(TID)만 새로운 값을 이용하여 갱신하면 된다. 이 경우에만 뷰인덱스가 시그니쳐를 유지할 때 추가되는 부담이다.

3.3 SVI의 활용예

위에서 정의된 조인뷰 **JoinView**에 대해 다음과 같이 조건을 가지는 질의문이 입력된다고 가정해 보자.

```
Select *
From JoinView S
Where S.DEPT = 'Computer'
  And S.GRADE = 'B0'
```

뷰인덱스를 사용하는 경우는 먼저 뷰인덱스로부터 조인에 참여한 릴레이션들의 TID 집합을 검색한 후 이 TID들을 사용하여 다시 각 베이스 릴레이션의 투플을 검색하여 질의문에 주어진 조건을 만족하는 지 검사하여야 한다. 그러나 시그니쳐 뷰인덱스로 조인뷰 **JoinView**가 생성되어 있는 경우는 먼저 시그니쳐 뷰인덱스로부터 조인에 참여한 릴레이션들의 TID 집합뿐만아니라 인덱스에 함께 저장되어 있는 각 베이스 릴레이션의 투플에 대한 시그니쳐를 검색하게 된다. 그리고 검색된 베이스 릴레이션의 시그니쳐를 이용하여 조인뷰에 대한 질의문에 주어진 다른 조건들을 만족하는 지 검사하여 조건을 만족하지 않는 투플은 실제로 검색하지 않도록 하고, 시그니쳐에 의해 만족하는 투플들만을 검색하므로서 디스크 입출력을 수행하는 검색연산에 대해 시그니쳐를 이용하여 여과(filter) 하는 효과를 얻을 수 있다.

즉 이 질의문에서 S.DEPT = 'Computer'에 대한 조건은 시그니쳐 뷰인덱스에서 첫째 시그니쳐인 **Student** 시그니쳐를 사용하여 만족여부를 검사하고, S.GRADE = 'B0'의 조건은 둘째 시그니쳐인 Course 시그니쳐를 이용하여 만족여부를 검사한 후 만족하는 경우에만 해당 투플을 실제로 검색하게 된다.

4. 성능 평가

이절에서는 지금까지 제안된 시그니쳐 뷰인덱스의 검색 성능 평가를 위해 비용식을 산출하고 평가 비교한 내용을 기술하였다. 평가 방법으로는 기존의 뷰인덱스 기법과의 비교를 통해 상대적으로 시그니쳐 뷰인덱스 기법이 가지는 성능의 우수성을 보인다.

단일 릴레이션에서 정의된 뷰보다 일반화된 두 개 이상의 릴레이션에서 조인을 통하여 정의된 조인뷰의 경우에 대해 평가 모델을 정의하였다. 성능 평가에서 문제를 간단히 하기 위해 다음과 같은 가정을 하였다.

1. 조인뷰의 정의는 n개의 릴레이션에 대해 n-1의 조인 조건이 주어진다. 즉 n개의 릴레이션은 선형으로 조인되고, 동일한 릴레이션들이 다른 애트리뷰트들에 의해 두번이상 조인되지 않는다.
2. 주어진 질의에서 조인에 사용된 각 릴레이션에 해당하는 조건을 만족할 확률은 모두 동일하다.

비교 평가 기준으로 기본 릴레이션에서 검색하는 투플수를 사용하였다.

성능 평가를 위해 사용된 변수들은 다음과 같다.

- n: 조인뷰에 사용된 릴레이션의 수
- R_i: 조인 정의에 사용된 i번째 릴레이션, $(1 \le i \le n)$
- N_i: i번째 릴레이션의 실제 투플 수
- NV: 조인결과로 생성되는 결과 뷰의 총 투플수, 이 값은 뷰가 구체화되었을 때 만들어지는 투플수로 뷰인덱스의 레코드수와 같다.
- C_i: 질의에 사용된 조건에서 i번째 릴레이션에 대한 조건
- P_i: 뷰의 총 투플중에서 릴레이션 R_i가 C_i를 만족할 확률
- M_i: NV개 중에서 릴레이션 R_i가 C_i를 만족하는 투플 수,

  $M_i = P_i \cdot MV$
- F_d: 시그니쳐에 대한 매치 오류 확률

비용식은 질의문에 주어진 조건들이 n개의 릴레이션중 i개에 대한 조건을 가지는 경우에 대해 산출하였다.

먼저 뷰인덱스 기법을 사용하는 경우 비용식은 다음과 같다. 질의 처리 과정은 별도의 인덱스가 없다고 할 경우 조건을 가지는 첫번째 릴레이션의 TID들을 뷰인덱스에서 모두 검색한다. 검색된 TID들을 사

용하여 실제 투플을 검색한 후 조건을 검사하여 만족하는 TID들을 구한다. 다음에는 구해진 TID들과 뷰 인덱스의 동일한 레코드에 있는 질의에 조건이 주어진 다른 릴레이션의 TID들을 뷰인덱스에서 찾아 실제 투플들을 검색한다. 이와 같이 릴레이션 단위로 뷰인덱스에서 TID를 찾아 실제 투플 검색과 조건 검사과정을 반복하므로써 질의 처리를 완료한다. 그러므로 질의에 주어진 조건들에서 i번째로 선택된 릴레이션으로 부터 검색해야하는 투플수는 이전 i-1개의 릴레이션에 대한 투플중 i-1개의 조건을 모두 만족할 확률을 곱한 것이 되고 i번째 이후의 릴레이션(n-i개)에 대해서는 검색하여야 할 투플수가 i개의 조건을 모두 만족하는 경우와 같다. 그러므로 다음과 같은 식이 유도된다.

총 투플수 = 첫번째 릴레이션에서 검색하는 투플수
+ 두번째 릴레이션에서 검색하는 투플수
...
+ i번째 릴레이션에서 검색하는 투플수
+ i+1번째 이후 릴레이션에서 검색하는 투플수

$$= NV + NV \cdot P_1 + NV \cdot P_1 \cdot P_2 + \cdots + NV \cdot P_1 \cdot P_2 \cdots P_{i-1} + NV \cdot P_1 \cdot P_2 \cdots P_i \cdot (n-i)$$

$$= NV \cdot (1 + P_1 + P_1 \cdot P_2 + \cdots + P_1 \cdot P_2 \cdots P_{i-1} + P_1 \cdot P_2 \cdots P_i \cdot (n-i))$$

시그니쳐 뷰인덱스 기법의 질의처리 과정은 릴레이션 단위로 실제 투플을 검색하지 않고 시그니쳐 뷰인덱스에 저장된 시그니쳐와 조건을 모두 비교하여 만족하는 투플들만을 나중에 모두 검색하는 과정을 따른다. NVS를 시그니쳐 뷰인덱스에서 질의문에 주어진 조건과 시그니쳐를 비교할 때 모두 만족하는 투플수라고 하는 경우 n개의 릴레이션에 대한 TID가 있으므로 총 투플수에 대한 식은 다음과 같다.

총 투플수 $= NVS \cdot n$

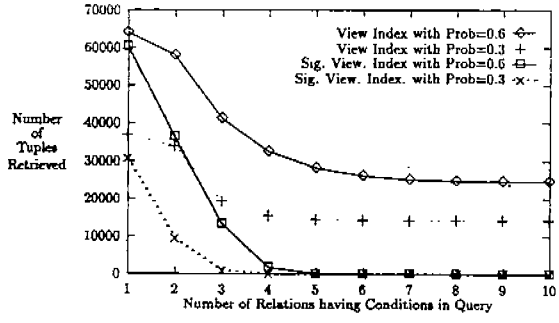
이 때 질의문에 i개의 릴레이션에 대한 조건이 주어지는 경우 NVS는 다음과 같이 계산된다.

$NVS = NV \cdot$ (P_1을 만족하는 갯수 + 매치 오류에 의해 검색되는 갯수)
$\cdot$ (P_1을 만족하는 갯수 + 매치 오류에 의해 검색되는 갯수)
...
$\cdot$ (P_i을 만족하는 갯수 + 매치 오류에 의해 검색되는 갯수)

$$= NV \cdot (P_1 + (1-P_1)F_d) \cdot (P_2 + (1-P_2)F_d) \cdots (P_i + (1-P_i)F_d) = NV \cdot \prod_{k=1}^{i} (P_k + (1-P_k)F_d$$

이 식에서 보면 질의에 조건이 더 많이 주어질 수록 시그니쳐 뷰인덱스 방식은 검색되는 투플수를 감소시킨다는 것을 알 수 있다.

(그림 5)는 이와 같은 비용식을 사용하여 평가 비교한 것이다. 여기서 사용한 데이타로 NV가 10,000, F_d는 0.01 그리고 릴레이션은 10개(n=10)로 가정하였다. 이때 뷰인덱스에 저장되는 총 TID의 수는 $NV \cdot n$ =100,000이 된다. P_i는 0.3과 0.6에 대해서 각각 그레프로 나타내었다. 그리고 질의에 주어진 조건의 수에 따른 검색 성능을 알기 위해서 조건의 개수를 변화하였다.



(그림 5)시그니쳐 뷰인덱스의 성능 비교 (n = 10, Prob = P_i)
(Fig. 5) Comparison of cost evaluation for the signature view index.(n = 10, Prob = P_i)

그림에서 보면 조건의 수가 4개이상 주어지면 시그니쳐 뷰인덱스의 비교만으로도 만족하는 투플이 없으므로 실제 검색을 거의 필요로 하지 않는다는 것을 알 수 있다.

5. 결 론

관계 데이타베이스 모델에서 뷰는 외부 스키마를

의미하며 중요한 역할을 하는 데 실제로 뷰를 사용하는 경우에는 질의 처리에 따르는 성능 저하가 문제로 인식되어 왔다. 그러나 뷰인덱스 기법을 통해 적은 저장 공간을 사용하며 상대적으로 신속한 뷰처리를 제안되었다. 그러나 뷰인덱스 기법은 질의문에 조건식이 주어지게 되면 뷰인덱스에 저장된 기본 릴레이션의 투플들을 검색하여야만 조건을 만족하는 지 판단할 수 있게 된다.

이 논문에서는 뷰인덱스 기법에 시그니쳐 참조 기법을 적용하여 뷰에 대한 질의 처리시 디스크 입/출력을 감소시킬 수 있는 시그니쳐 뷰인덱스 기법을 제안하였다. 제안된 기법에서는 뷰인덱스에 각 릴레이션에 대한 뷰에서 정의된 애트리뷰트들로 구성된 시그니쳐를 만들어 저장한다. 질의문이 주어지면 뷰인덱스에 저장된 투플지시자들을 모두 검색할 필요없이 조건식을 저장된 시그니쳐를 이용하여 검사하여 만족하는 투플들만을 검색하도록 하므로써 디스크 입/출력을 감소시킬 수 있다. 또한 성능 평가를 위해서 질의 모델과 비용식을 제시했다. 비용식에 의한 성능 평가 결과 10개의 릴레이션이 조인되고, 질의에 대한 조건을 만족할 확률이 0.6이며 질의에 조건이 2개 주어진 경우 37%, 조건이 3개 주어진 경우 67%의 검색 투플수의 감소를 보여주었다.

향후 연구 방향으로는 실제 구현을 통해 투플수가 아니고 디스크 블럭의 입/출력 횟수로 분석하는 연구가 필요하다. 그리고 이 기법을 확장하여 객체지향 데이타베이스나 중첩 관계 데이타베이스에서의 뷰처리를 위해서도 적용할 수 있을 것으로 전망된다.

참 고 문 헌

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

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

[3] N. Roussopoulos, "View Indexing in Relational Databases," ACM Trans. on Database Systems, vol. 7, no. 2, pp. 258-290, 1982.

[4] N. Roussopoulos, "An Incremental Access Method for ViewCache: Concept, Algorithms and Cost Analysis," ACM Trans. on Database Systems, vol. 16, no. 3, pp. 535-563, 1991.

[5] C. Faloutsos, "Access Methods for Text," ACM Computing Surveys, vol. 17, pp. 49-74, March 1985.

[6] 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.

[7] 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.

[8] C. Roberts, "Partial Match Retrieval via the Method of Superimposed Codes," Proc. of IEEE, vol. 67, pp. 1624-1642, Dec. 1979.

[9] R. Sacks-Davis and K. Ramamohanarao, "A Two level superimposed coding scheme for partial-match retrieval," Information Systems, vol. 8, no. 4, pp. 273-280, 1983.

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

[11] 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.

용 환 승

1983년 서울대학교 컴퓨터공학과 졸업(학사)

1985년 서울대학교 대학원 컴퓨터공학과 졸업(공학석사)

1994년 서울대학교 대학원 컴퓨터공학과 졸업(공학박사)

1985년~1989년 한국전자통신연구소 연구원

1994년~1995년 서울대 컴퓨터신기술공동연구소 특별연구원

1995년~현재 이화여자대학교 전자계산학과 조교수