論文94-31B-6-1

중첩 릴레이션 데이타베이스에서 질의 처리를 위한 시그니쳐 기반 저장 구조

(Storage Structure using Signatures for Query Processing in Nested Relational Databases)

龍 煥 昇*, 李 錫 浩**

(Hwan-Seung Yong and Sukho Lee)

要 約

이 논문은 중첩 릴레이션 데이타베이스에서 중첩 애트리뷰트에 대한 조건을 가지는 질의를 신속하게 처리하기 위한 기법으로 시그니쳐를 이용한 저장 구조를 제시하였다. 이 기법은 중첩 투플의 저장 구조에서 서브릴레이션에 대한 시그니쳐를 생성하여 서브릴레이션 포인터와 함께 저장한다. 중첩 조건을 가지는 질의문의 처리 과정에서 조건식을 먼저 서브릴레이션 시그니쳐와 검사하여 매치가 되는 경우에만 실제 서브릴레이션을 검색하므로써 디스크 I/O를 감소시킨다.

Abstract

A storage structure using signatures is proposed to evaluate efficiently queries including conditions of nested attributes in the nested relational databases This method stores a subrelation signature into the storage structure for a nested tuple with its subrelation pointer. During query processing steps, the subrelation signatures are matched first with the nested predicates in the query.

When the match operation completes with success then physical retrieval of the subrelation occurrs resulting in reduction of disk I/Os.

Ⅰ. 서 론

최근 들어 컴퓨터를 이용한 설계나 인공 지능 분야, 사무 자동화 시스템등의 새로운 응용을 지원하는 데이타베이스 시스템에 관한 연구가 활발히 진행되고 있다. 이러한 응용을 효과적으로 지원하기 위한 연구로 기존의 관계 데이타베이스 시스템을 확장한 중첩 관계 데이타베이스 (Nested Relational Database) 시스템이 있다. [1-4]

중첩 릴레이션은 애트리뷰트가 기존의 원자값 (atomic value)외에 릴레이션을 값으로 가질 수 있는 릴레이션 값 애트리뷰트(Relation Valued Attribute: RVA)가 허용된 릴레이션이다. 애트리뷰트의 타입으로 투플들의 집합인 서브 릴레이션외에 중첩 릴레이션 모델을 확장하여 투플들의 순서정보를 표현하는 리스트 (list) 데이타 타입을 제안하기도 하였다. [5]

* 正會員, 서울大學校 컴퓨터工學科
(RIACT., Seoul Nat'l Univ.)
** 終身會員, 서울大學校 컴퓨터工學科
(Dept. of Computer Eng., Seoul Nat'l Univ.)
接受日字 : 1993年 8月 17日

정수나 실수, 문자열등의 원자값을 갖는 애트리뷰트를 원자값 애트리뷰트(Atomic Valued Attribute: AVA)라고 한다. 릴레이션값 애트리뷰트의 값을 서브 릴레이션(sub-relation)이라 하고, 서브 릴레이션에 속한 투플을 서브 투플(sub-tuple)이라고 한다. 중첩 릴레이션에서 서브 투플이 아닌 투플은 중첩 투플(nested tuple)이라고 한다.

서브 릴레이션에 정의된 애트리뷰트들은 중첩 애트리뷰트(nested attribute)라고 한다. 질의시 제공되는 조건식(predicate)에는 이와 같은 중첩 애트리뷰트를 사용할 수가 있는 데 중첩 애트리뷰트를 사용한 조건식을 중첩 조건식(nested predicate)이라고 한다. [5]

dname	office	Professor			
		pname	age	Lectures	
				cname	credit
Computer	38-410	S.Park	42	Fortran	1
				Logic	2
				English	3
		K.Kim	36	Archi.	3
				O.S	2
Mechanics	31-220	Y.Lim	45	Math.	3
				Physics	3
				English	3
		R.Choi	36	Dynamics	3
				Fortran	2

그림 1. 중첩 릴레이션 Department
Fig. 1. Nested relation Department.

그림 1은 두개의 중첩 투플로 구성된 중첩 릴레이션 Department를 보여준다.

사용자에 의해 주어진 질의문은 중첩 조건식을 가지게 되는 데 그 예는 다음과 같다.

Professor.Lectures.cname = 'English'

이러한 질의를 처리하기 위해서는 중첩 애트리뷰트를 포함하는 서브릴레이션을 검색하여야 한다. 중첩 조건식을 가지는 질의를 신속히 처리하기 위해서 중첩 인덱스[6] 기법이 제시되었다. 그러나 중첩 인덱스를 정의하는 데 필요한 저장 공간과 갱신에 따른 인덱스 유지 비용은 모든 중첩 애트리뷰트에 대해 인덱스를 정의할 수 없고 중요한 애트리뷰트에 대해서만 정의가 가능하게 된다.

다중키를 지원하기 위해 제시되었던 시그니쳐(signature) 기법은 최근 들어 문서 검색에 많이 적용되며[7] 또한 다매체(Multi-media) 데이타베이스 등에도 사용되고 있다.[8-9] 데이타로부터 시그니쳐를 만드는 방법은 여러가지가 있지만[10] 이 논문에서는 superimposed code기법을 사용하는 것으로 한정한다. 일반적으로 시그니쳐는 여러 개의 값들을 각각 해싱하여 시그니쳐의 크기 b비트위에 k개의 비트를 1로 세트하여 얻어지는 결과를 다시 비트 단위로 OR 연산을 통해 생성한다. 예를 들어 데이타 D={G.D. Hong, 25, Seoul}의 세 값에 대한 시그니쳐 S_D는 표 1과 같다.

표 1. 시그니쳐 생성 과정 예 (b=16, k=4인 경우)
Table 1. Signature generation example. (in case, b=16, k=4)

값	해싱결과
G.D.Hong	1001 0000 1000 0001
25	1000 1100 0000 0100
Seoul	1000 0000 1100 1000
시그니쳐 SD	1001 1100 1100 1101

이 SD에 'G.D.Yong'이란 값이 포함되어 있는지 여부를 확인하는 방법은 질의에 주어진 키를 똑같은 방법으로 해싱하여 질의 시그니쳐 SQ를 만든 후 이 SQ에 1로 세트된 비트들이 SD에도 1로 세트되어 있는지만 확인하면 되는 데 이 과정은 '∩'을 비트 단위 AND 연산자라고 하는 경우 다음과 같이 표현된다.

$$(S_D \cap S_Q) = S_Q$$

또한 이 SD에 'Seoul'과 25의 두 값이 모두 포함되어 있는지 검사(AND에 의한 다중 조건)하는 경우에도 단일 값이 주어진 경우와 똑같은 과정으로 수행된다. 그러나 만일 값 26의 해싱 결과가 '1000 0100 1000 1000'라고 하는 경우 이 값을 S_D와 비교하면 성공하게 되는 데 이와 같은 경우를 매치 오류(False Drop)라고 한다.

매치 오류 확률 Fd는 다음과 같이 계산된다.

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

시그니쳐가 가지는 장점은 적은 저장 공간을 필요로 하는 점과 다중 애트리뷰트에 대해 인덱스를 구성할 수 있다는 점이다. 그러나 시그니쳐를 인덱스로 사용하는 경우 데이타의 갯수가 많아지는 경우 검사해야할 시그니쳐의 갯수도 증가하게 되어 검색 성능이 저하되는 단점이 있다. 최근 들어[11]에서는 시그니쳐를 이용하여 객체 지향 데이타베이스에 대한 인덱스 기법을 제시하기도 했다.

이 논문에서는 중첩 릴레이션 데이타베이스에서 중첩 조건식을 가지는 질의 처리를 효율적으로 처리하기 위해 시그니쳐를 이용한 중첩 투플의 저장 구조를 제안하였다. 이 기법에서는 각 서브릴레이션에 대해

하나의 시그니쳐를 생성하여 서브릴레이션을 포함하는 투플에 저장한다. 서브릴레이션에 대한 조건을 가지는 질의가 주어지면 질의 처리하는 과정에서 먼저 미리 저장된 서브릴레이션에 대한 시그니쳐를 검사하여 만족하는 경우에만 서브릴레이션을 검색한다. 투플의 리스트로 구성된 서브릴레이션에 대해서도 시그니쳐 생성이 가능하며 영역(range) 질의문을 지원할 수 있도록 서브릴레이션 시그니쳐 생성 방법도 기술하였다.

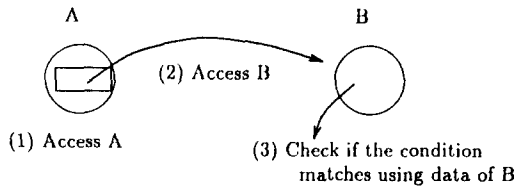
이 논문의 구성은 2장에서 시그니쳐 참조 기법과 시그니쳐를 이용한 중첩 투플의 저장 구조에 대해 기술하였으며 3장에서 서브릴레이션과 영역 질의문 처리를 위한 시그니쳐 생성 기법을 제시한다. 그리고 4장에서 질의 처리 과정과 성능을 평가하기 위한 모델, 평가식을 제시하였으며, 5장에서 결론을 맺었다.

Ⅱ. 시그니쳐 참조 기법과 중첩 릴레이션 저장 구조

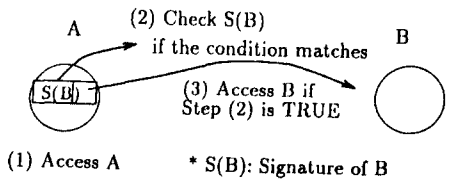
1. 시그니쳐 참조 기법

시그니쳐 참조 기법[12]은 객체 또는 임의의 집합값을 가지는 데이타에 대해 포인터나 기타 방법으로 참조(reference) 관계가 형성되어 있는 경우 피참조 객체의 시그니쳐를 생성하여 참조 객체에 미리 저장하여 놓는 방법이다.

질의 처리 과정에서는 일반적으로 참조 관계를 통해 피참조 객체를 검색하게 된다.



(a)기존의 참조 개체 검색 과정



(b)시그니쳐를 사용한 참조 객체 검색 과정

그림 2. 참조 관계를 통한 객체 검색 과정 비교
Fig. 2. Comparison of access procedure using reference relationship.

먼저 기존의 참조 관계를 통한 검색 과정은 다음과 같이 수행된다.

1) 먼저 참조하는 객체 A를 검색한다.
2) 객체 A에 있는 피참조 객체 포인터를 통해 피참조 객체 B를 검색한다.
3) 객체 B의 데이타를 가지고 조건식을 검사한다.

반면에 피참조 객체의 객체 시그니쳐를 참조 객체에 저장하는 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 경우에만 피참조 객체 B를 검색한다.

그러므로 피참조 객체가 조건을 만족하는 경우에만 실제 피참조 객체의 검색이 수행되기 때문에 입/출력을 감소시키는 것이다. 시그니쳐 참조 기법과 객체 시그니쳐를 이용하여 객체 지향 데이타베이스에서 전진 운행(forward traversal) 방법으로 질의 처리를 신속히 할 수 있는 기법이 제시되기도 하였다.[12]

2. 중첩 릴레이션의 저장 구조

중첩 릴레이션에 대한 저장 구조는 서브 릴레이션을 저장하는 기법에 따라 크게 직접 저장(Direct Storage Model: DSM) 또는 클러스터(cluster) 기법과 정규화(normalize) 하여 여러 개의 테이블에 분산 저장하는 정규화 저장(Normalized Storage Model: NSM) 기법으로 나눌 수 있다.[13]

직접 저장 기법은 계층적 구조를 가지는 각 중첩 투플을 가능한 한 적은 수의 페이지에 클러스터 형태로 저장하는 방법이다. 이 방법은 중첩 투플 전체를 검색하는 경우에 효과적이지만 일부의 애트리뷰트나 서브 릴레이션에 대한 검색 연산시 전체 중첩 투플을 검색하는 데 따르는 많은 입/출력으로 인해 성능 저하를 가져 온다. 그러므로 DASDBS[14]에서는 이를 보완하여 중첩 투플이 저장되는 첫번째 페이지(헤더 페이지라고 함)내에 각 서브릴레이션에 대한 포인터를 사용하므로써 중첩 투플의 일부분만을 검색하는 경우에도 2번(헤더 페이지와 서브릴레이션의 데이타 페이지)의 페이지 I/O로 검색이 가능하도록 하고 있다.

정규화 저장 구조(NSM: Normalized Storage Model)는 하나의 중첩 릴레이션을 각 서브릴레이션에 따라 서브릴레이션이 속한 상위 투플의 키 정보와 함께 별도의 테이블로 저장 한다. Teeuw등[13]은 정규화 구조와 클러스터 구조를 혼합하여 한 단계의 중

첩 구조를 허용하는 DASDBS-NSM 구조를 제시하였으며 성능 평가를 통해 종합적인 측면에서 이 구조가 가장 우수한 성능을 보여준다고 하였다.

위에서 기술한 두가지 저장 구조에서 공통적인 점은 하나의 중첩 투플에서 특정 서브릴레이션을검색하고자 하는 경우 서브릴레이션의 포인터나 키를 사용한 추가적인 디스크 I/O를 필요로 한다는 점이다.

이와 같은 중첩 조건식을 가지는 질의문을 처리하기 위해 위에서 설명한 시그니쳐 참조 기법을 사용하여 서브릴레이션을 검색하고자 할 때 추가적인 디스크 I/O를 감소시킬 수 있는 기법을 제시하였다.

중첩 투플은 원자값 애트리뷰트에 따른 원자값들과 릴레이션값 애트리뷰트에 따른 릴레이션 값들로 구성된다. 원자값들은 원자값 레코드에 저장되고 릴레이션 값들은 다시 중첩 투플의 집합으로 구분되어 동일한 방법이 재귀적으로 (recursively) 적용되어 저장된다.

각 중첩 투플은 헤더와 데이타 페이지로 구분되는 페이지의 집합을 이용하여 저장된다. 헤더 페이지에는 원자값 레코드 링크 (Atomic Record Link: ARL)와 서브릴레이션 링크 (Subrelation Link: SL)들로 구성되는 링크 레코드 (Link Record: LR)들이 저장된다. 데이타 페이지는 헤더 페이지의 ARL들에 의해 링크된 원자값 레코드들이 저장된다. SL은 서브릴레이션을 구성하는 각 투플들을 연결하는 LR들의 집합을 연결할 뿐만아니라 서브릴레이션 시그니쳐를 가지고 있다. 그러므로 하나의 LR은 하나의 중첩 투플을 연결하거나 또는 하나의 원자값 레코드를 연결할 수 있게 된다. 그림 3은 Department 중첩 릴레이션의 첫번째 중첩 투플을 저장 구조로 표현한 예를 보여주고 있다.

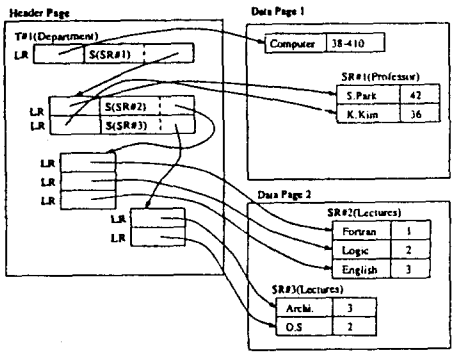


그림 3. 시그니쳐를 이용한 저장 구조
Fig. 3. Storage structure using signatures.

Ⅲ. 시그니쳐 생성 및 관리

1. 서브릴레이션에 대한 시그니쳐 생성

서브릴레이션에 대해 시그니쳐를 만드는 방법은 다음과 같이 세가지가 가능하다 첫번째 방법은 투플의 경우와 같이 투플 단위로 투플이 가지는 모든 원자값들을 가지고 시그니쳐를 만드는 것이다. 그러나 투플 단위로 만드는 경우 서브릴레이션이 가지는 투플의 갯수가 가변적이기 때문에 시그니쳐의 갯수도 가변적이 된다. 이는 서브릴레이션 시그니쳐를 저장하기 위해가변 저장 공간 관리 기법이 필요하게 되어 처리 알고리즘이 복잡하게 된다.

두번째 방법은 서브릴레이션에 속하는 모든 값들을 투플과 애트리뷰트를 구별하지 않고 단일 집합으로 간주하여 하나의 시그니쳐로 만드는 것이다. 그러나 이 방법의 문제는 하나의 서브릴레이션에 속하는 두개의 애트리뷰트가 동일한 도메인을 가지는 경우 다중 조건식을 가지는 질의를 처리하는 과정에서 시그니쳐 매치 오류에 따른 검색 성능 저하를 가져 오게 된다.

세번째 방법은 서브릴레이션을 애트리뷰트 단위로 나누어 각 투플들에서 특정한 애트리뷰트의 값들로 별도의 시그니쳐를 만들어 접합(concatenate)하는 것이다. 이 방법은 인덱스된 시그니쳐 화일 기법[15] 에서와 같이 정형화된 레코드에 대해 시그니쳐를 만들 때 superimposed code 기법을 사용하지 않고 각 애트리뷰트 값들을 별도로 disjoint code 기법을 사용하여 레코드 시그니쳐를 만든 후 동일한 블럭에 속하는 레코드 시그니쳐들을 OR 연산을 통해 블럭 시그니쳐를 만드는 기법과 유사하다. 이 논문에서는 세번째 방법을 사용하였다.

그림 3에 있는 Lectures 서브릴레이션 SR#1과 서브릴레이션 시그니쳐 생성 예가 그림 4에 있다.

cname	credit
Fortran	1
Logic	2
English	3

(a)서브릴레이션 예

S({Fortran, Logic, English})	S({1,2,3})

(b) 서브릴레이션 시그니쳐 예

그림 4. 서브릴레이션과 시그니쳐 생성 예
Fig. 4. Subrelation and its signature generation example.

그림에서 'S(값의 집합)'은 값의 집합을 이용하여 생성한 시그니쳐를 나타낸다. 이 방법은 또한 투플들의 리스트로 정의된 서브릴레이션에 대해 시그니쳐를 생성하는 것을 가능하게 한다. 투플 리스트에 대한 시그니쳐는 원자값의 리스트에 대해 시그니쳐를 생성하는 기법[16]을 이용하여 각 애트리뷰트 별로 값들의 리스트에 대해 시그니쳐를 만들어 접합하면 된다.

2. 영역 질의를 위한 시그니쳐 생성

시그니쳐는 대소관계를 비교하는 영역 조건(range condition)을 가지는 질의를 지원하지 못한다. 이를 해결하는 방안으로 시그니쳐와 별도로 태그 필드(tag field)[7]이 제시되었다. 태그 필드 기법에서는 영역 조건의 비교에서 사용되는 값들을 영역 구분자(range delimiter: RD)라고 부르는 데 그 갯수를 $|RD|$로 나타내도록 하겠다. 태그 필드는 각 영역 구분자에 대해 '≤'와 '>'의 두 조건에 따라 한 비트를 할당하여 표현하여 $2 \cdot |RD|$ 비트가 필요하게 된다. 그러나 태그 필드 방법은 다음과 같은 두가지 문제점이 있다. 첫째로 '≤'와 '>'의 두 조건 형태만 처리 가능하고 '≥'와 '>'의 조건은 처리하지 못한다는 점이다.

둘째로는 집합으로 주어지는 값들에 대해 하나의 태그 필드를 구성하지 못한다는 것이다. 이 점은 서브릴레이션에서 애트리뷰트 시그니쳐를 구성하는 데 적용하지 못하게 된다. 이 문제들을 해결하는 방안으로 제시하는 확장된 태그 필드 (extended tag field) 기법을 Department 중첩 릴레이션에서 Professor.age 중첩 애트리뷰트를 예로 들어 설명하면 다음과 같다.

- TF_D(i) : 주어진 집합 데이타 D에 대한 i번째의 태그 필드 비트
- TF_Q(i) : 주어진 영역 조건 Q에 대한 i번째 태그 필드 비트

이 방식에서 태그 필드의 각 비트 구성은 영역 구분자를 통해 만들어지는 각 영역 구간과 구분자 값 자체를 나타내는 데 사용 한다. 그러므로 영역 구간을 나타내기 위해서는 $|RD|+1$개의 비트가 필요하므로 태그 필드의 크기는 $2 \cdot |RD|+1$이 된다.

영역 구분자의 집합이 {30, 40, 50, 60}인 경우 데이타를 위한 태그필드인 TF_D의 구성과 할당 예는 다음과 같다. 이 테이블에서 데이타 조건에 (x, y)인 경우는 데이타가 x와 y의 사이에 있는 경우에 해당 비트가 '1'로 세트됨을 뜻한다.

비트 번호	1	2	3	4	5	6	7	8	9
데이타 조건	(-∞, 30)	30	(30, 40)	40	(40, 50)	50	(50, 60)	60	(60, +∞)

예를 들어 D={42, 58}의 두 값이 주어진 경우 태그 필드를 구성하면 각각의 태그 필드인 '000010000'과 '000001000'에 대해 OR 연산을 수행한 값 '000011000'이 된다. 질의어에 영역 조건이 주어지는 경우 조건식 대한 태그 필드 TF_Q는 주어진 영역 조건을 만족하는 모든 비트를 1로 세트함으로써 구성되는데 그 예는 다음과 같다.

경우	영역 조건	태그 필드
1	30 <age ≤ 50	001111000
2	age ≤ 50	111111110
3	60 <age ≤ 50	000000011

데이타 D의 태그 필드 TF_D와 조건 태그 필드 TF_Q에 대해 매치는 비트 단위 'AND' 연산인 '∩'에 대해 다음 조건을 만족하면 된다.

- 'EXIST' 조건의 경우 $TF_D \cap TF_Q \neq NULL$
- 'UNIVERSAL' 조건의 경우 $TF_D \cap TF_Q = TF_D$

3. 시그니쳐의 관리

서브릴레이션 시그니쳐는 중첩 투플이 생성되어 저장될 때 만들어져 저장된다.

일반적으로 시그니쳐의 크기 b(단위: 비트)는 한 시그니쳐에 해싱되는 값들의 갯수(s)와 시그니쳐의 매치 오류 확률 (F_d)에 의해 정해지며 계산식은 다음과 같다.[17]

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

그러므로 서브릴레이션 시그니쳐의 크기 Sigsize는 서브릴레이션의 애트리뷰트 수와 평균 투플수(값들의 갯수)에 따라 다르게 된다. Sigsize는 애트리뷰트의 수를 nattr이라고 하고 한 애트리뷰트 시그니쳐의 크기를 b라고 하는 경우 $nattr \cdot b$로 계산된다.

한 서브릴레이션 시그니쳐는 그 서브릴레이션의 투플들이 가지는 원자값이 갱신될 때 마다 변경되어야 한다. 그러나 서브릴레이션들은 객체 지향 데이타베이스의 객체들과 달리 다른 중첩 투플들에 의해 공유되지 않으므로 해더 페이지에 서브릴레이션 포인터와 함께 저장된 하나의 서브릴레이션 시그니쳐만 변경하면 된다.

Ⅳ. 질의 처리 및 성능 평가

1. 질의 처리

중첩 조건식을 가지는 질의문이 주어졌을 때 각 중첩 투플에 대한 기존의 질의 처리 과정은 다음과 같다.

1) 먼저 헤더 페이지로부터 중첩 조건식의 중첩 애트리뷰트에 해당하는 서브릴레이션의 포인터를 검색한다.
2) 서브 릴레이션을 읽어 온다.
3) 검색된 서브릴레이션에 대해 중첩 조건식을 검사한다.

반면에 서브릴레이션에 대한 시그니쳐를 포인터와 함께 저장한 방식을 사용하는 경우 질의 처리 과정은 다음과 같다.

1) 먼저 헤더 페이지로부터 서브릴레이션의 포인터와 시그니쳐를 검색한다.
2) 검색된 서브릴레이션 시그니쳐에 이용하여 중첩 조건식을 검사한다.
3) 단계 2에서 조건이 만족되는 경우에만 서브릴레이션을 검색한다.

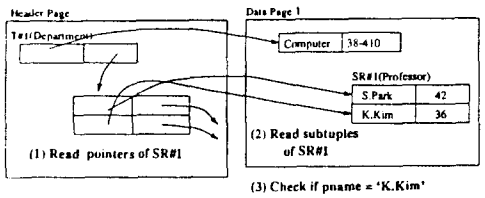
이 과정을 실제 예를 들어 설명하기 위해 다음과 같은 질의문 Q1이 주어졌다고 하자. 이 논문에서는 중첩 릴레이션에 대한 질의어 표현으로 SQL/NF [18]를 사용하였다.

Q1: 이름이 'K.Kim'인 교수가 소속한 학과명을 검색하라.

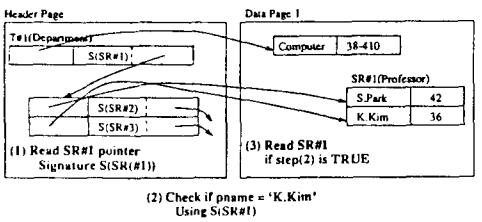
이와 같은 질의문은 다음과 같은 질의어로 표현된다:

SELECT dname
FROM Department D
WHERE D.Professor.pname = 'K.Kim'

그림 5는 Q1에 대한 질의 처리 과정을 비교하여 보여주고 있다.



(a) 기존의 질의 처리 과정



(b) 시그니쳐를 사용한 질의 처리 과정

그림 5. 질의 처리 과정 비교
Fig. 5. Comparison of query processing procedure.

2. 성능 평가 모델

중첩 애트리뷰트 R. SR_1. SR_2. …. SR_n.A는 중첩 릴레이션 R에서 SR_1이 R의 RVA이고, SR_i가 $SR_{i-1}(2 \le i \le n)$의 RVA이며 A가 SR_n의 AVA를 나타낸다고 하자.

이 논문에서는 성능 평가의 기준으로 하나의 중첩 투플내에서 질의 처리를 위해 검색해야 하는 서브릴레이션의 갯수를 가지고 비교하였다.

성능 평가를 위해 사용된 기호는 다음과 같다.

· $NST(SR_i)$: SR_i 서브릴레이션들이 가지는 평균 투플 수

· $NST(SR_i)$: SR_i 의 서브릴레이션 갯수

· $P_i(1 \le i \le n)$: SR_i의 서브릴레이션에서 주어진 조건을 만족할 확률

· V : 시그니쳐를 사용하지 않는 경우 검색해야 할 서브릴레이션의 수

· V_{sig} : 시그니쳐를 사용하는 경우 검색해야 할 서브릴레이션의 수

예를 들어 한 학과의 평균 교수는 4 명이고 한 교수당 평균 3 과목의 강의를 하는 경우 NST(Professors) = 4, NST(Lectures) = 3 과 같이 나타낸다. 그러므로 $NST(SR_i)(1 \le i \le n)$가 주어지는 경우 $NSR(SR_i)(1 \le i \le n)$은 다음과 같이 계산된다.

$$NSR(SR_i) = \prod_{j=1}^{i-1} NST(SR_j)$$

예를 들어 Department.Professor.Lectures.cname = 'Fortran'과 같은 조건식의 처리를 위해 하나의 중첩 투플내에서 검색해야 하는 Lectures 서브릴레이션의 갯수 NSR(Lectures)는 NST(Professor)로 4개가 된다.

3. 성능 평가

1) 단일 술어를 가진 질의어

중첩 애트리뷰트 R. SR_1. SR_2…. SR_n.A를 사용하는 중첩 조건식 하나를 가진 질의어가 주어진 경우 $V = NSR\ (SR_n)$이 된다. 그러나 시그니쳐를 사용하는 경우

V_{sig} = SR_n · 조건을 만족할 확률 + 시그니쳐 매치 오류에 의한 서브릴레이션 수의 갯수
= $NSR\ (SR_n)$ · $(P_i + (1-P_i)F_d)$이 된다.

2) 다중 조건식을 가진 질의어

여러 개의 중첩 조건식이 주어진 질의어는 조건식에 주어진 중첩 애트리뷰트들이 동일한 서브릴레이션에서 정의된 경우와 그렇치 않은 경우의 두가지 형태로 나누어진다.

질의어 Q2는 조건식에 중첩 애트리뷰트들이 동일한 서브릴레이션에 해당하지 않는 경우의 예를 보여준다.

Q2: 이름이 'DB'인 연구실을 가지고 있고 'Fortran' 과목을 강의하는 교수를 포함하는 학과의 이름을 검색하라.

SELECT dname
FROM Department D
WHERE D.Labs.labname = 'DB'
AND D.Professor.Lectures.cname = 'Fortran'

이와 같은 경우는 검색하여야 하는 서브릴레이션의 갯수는 각각의 단일 조건식에 대해서 검색 해야할 서브릴레이션의 수를 합한 것으로 계산된다. 그러나 질의어 Q3는 질의문에서 다중 조건의 중첩 애트리뷰트가 이 동일한 서브릴레이션을 사용하는 경우를 보여준다.

Q3: 교수의 이름이 'S.Park'이며 'Fortran' 과목을 강의하는 학과의 이름을 검색하라.

SELECT dname
FROM Department D
WHERE D.Professor.pname = 'S.Park'
AND D.Professor.Lectures.cname = 'Fortran'

이와 같은 질의문의 처리과정은 먼저 첫째 조건을 적용하여 Professor 서브릴레이션을 선택한 후 선택된 Professor 서브릴레이션에 중에서 Lectures 서브릴레이션을 검색하여 다음 조건을 검사하는 단계별 검사를 수행하게 된다.

다음과 같은 두 중첩 애트리뷰트 N_1과 N_2를 가지는 질의문이 주어졌다고 하자.

N_1=$R.\ SR_1.\ SR.\ \cdots.\ SR_i.\ A_1$

N_1=$R.\ SR_1.\ SR.\ \cdots.\ SR_{i0}\cdots.\ A_2$

두 중첩 애트리뷰트는 $\underline{SR_j}(1\leq j\leq i)$ 서브릴레이션을 공유하고 있다. 이 때 시그니쳐를 사용하지 않는 경우 두 조건을 사용하는 질의문을 위해 검색하여야 할 서브릴레이션의 수 $V(N_1\cap N_2)$는 다음과 같이 계산된다.

$V(N_1\cap N_2)$ = $V(N_1)+V(N_2)$

$V(N_1)$ = $NSR(SR_i)$

$V(N_1)$ = N_1을 처리하기 위한 SR_i 서브릴레이션 갯수 · 매치 확률 · 매치된 SR_i 서브릴레이션 내의 SR_n 서브릴레이션의 갯수

= $V(N_1)\cdot P_i\cdot NST(SR_i)\cdots NST(SR_{n-1})$

반면에 시그니쳐를 사용하는 경우는 Professor 서브릴레이션을 선택하는 과정에서 시그니쳐를 사용하고, 다음 Lectures 서브렐레이션을 선택하는 과정에서 또한 해당 시그니쳐를 사용하게 된다. 검색해야 할 서브릴레이션의 갯수는 다음과 같이 계산된다.

$V_{sig}(N_1\cap N_2)$ = $V_{sig}(N_1)+V_{sig}(N_2)$

$V_{sig}(N_1)$ = SR_i 서브릴레이션중 조건을 만족하는 갯수 + 시그니쳐 매치 오류에 의한 갯수 $NSR(SR_i)\cdot(P_i+(1-P_i)F_d)$

$V_{sig}(N_2)$ = (N_1을 만족하는 서브릴레이션 갯수 · 매치된 SR_i 서브릴레이션 내의 SR_n 서브릴레이션에서 조건을 만족하는 갯수 + 시그니쳐 매치 오류에 의한 서브릴레이션 갯수)

= $NSR(SR_i)\cdot P_i$ $\cdot NST(SR_i)\cdots NST(SR_{n-1})\cdot(P_n+(1-P_n)F_d)$

인덱스를 사용하는 방법은 조건식이 많아질수록 각 조건식에 대한 인덱스들을 별도로 검색해야 하며 검색 결과에 대해 교집합 연산을 수행하는 과정이 필요하게 된다. 그러나 시그니쳐를 이용한 이 기법에서는 조건식의 수가 많아질수록 시그니쳐에 의한 여과(filter) 효과가 증대되기 때문에 검색 성능은 상대적으로 더욱 좋아지며 또한 질의 처리 과정이 조건식의 수와는 무관하게 되는 장점이 있다.

Ⅴ. 맺음말

중첩 릴레이션 데이타베이스에서 중첩 조건식을 가진 질의문을 처리하기 위해서는 해당 서브릴레이션을 검색하여야 한다. 그러나 기존의 중첩 투플에 대한 저장구조는 각 서브릴레이션을 검색하기 위해서는 추가적인 디스크 I/O를 필요로하여 검색시 많은 시간이 소요된다. 이 논문에서는 이와 같은 중첩 조건식을 가지는 질의를 신속히 처리하기 위해 시그니쳐를 사용한 중첩 투플의 저장구조를 제안하였다. 제안된 구조에서는 중첩 투플이 저장되는 헤더 페이지에 각 서브릴레이션에 대한 시그니쳐를 생성하여 해당 서브릴레이션에 대한 포인터와 함께 저장한다. 그러므로 질의 처리시 먼저 서브릴레이션의 시그니쳐와 조건식을 비교하여 만족하는 경우에만 서브릴레이션을 검색하므로써 디스크 I/O 비용을 감소시킨다.

서브릴레이션의 시그니쳐 생성 기법으로 애트리뷰트 단위로 만드는 방법을 제시하였는데 이 방법은 투플 리스트 타입의 서브릴레이션에 대해서도 적용이 가능한 특징이 있다. 그리고 영역 질의를 위한 시그니쳐 생성 기법으로 확장된 태그 필드 기법도 제시하였다.

그리고 성능 평가를 위해서 질의 모델과 비용식을 제시했다. 향후 연구 방향으로는 실제 구현을 통한 실험적인 평가가 연구되어야 할 것이다.

參 考 文 獻

[1] S. Abiteboul et al., "Towards DBMSs For Supporting New Applications," in Proc. on Very Large Data Bases, pp. 423-435, August 1986.

[2] P. Dadam, "A DBMS Prototype to Support NF2 Relations: An Integrated View on Flat Tables and Hierarchies," in Proc. ACM SIGMOD Conf., pp.356-367, 1986.

[3] P. Pistor and P. Dadam, "The Advanced Information Management Prototype," in Nested Relations and Complex Objects in Databases, LNCS 361, pp. 3-26, Springer-Verlag, 1989.

[4] M. Scholl et al., "VERSO: A Database Machine Based on Nested Relations," in Nested Relations and Complex Objects in Databases, LNCS 361, pp. 100-200, Springer-Verlag, 1989.

[5] P. Dadam and V. Linnermann, "Advanced Information Management (AIM): Advanced database technology for integrated applications," *IBM Systems Journal*, vol.28, no.4, pp. 661-681, 1989.

[6] E. Bertino and W. Kim, "Indexing Technique for Queries on Nested Objects," IEEE Trans. on Knowledge and Data Eng., vol. 1, pp. 196-214, March 1989.

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

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

[9] F. Rabitti and P. Zezula, "A Dynamic Signature Technique for Multimedia Databases," in Proc. of ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 193-210, Sept. 1990.

[10] C. Faloutsos, "Signature Files: Design and Performance comparison of some signature extraction methods," in ACM SIGMOD Conference, pp.63-82, 1985.

[11] W. Lee and D. Lee, "Signature File Methods for Indexing Object-Oriented Database Systems," in Proc. of Int'l Computer Science Conference(ICSC), pp. 616-622, 1992.

[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, Houston, Feb., 1994, pp. 518-525.

[13] W. Teeuw, C. Rich, M. Scholl, and H. Blanken, "An Evaluation of Physical Disk I/Os for Complex Object Processing," in Proc. of Int'l Conf. on Data Engineering, pp. 363-371, 1993.

[14] U. Deppisch, H. Paul, and H. Schek, "A Storage System for Complex Objects," in 1st Int'l Workshop on OODB, pp.183-195, 1986.

[15] L. Pfaltz et al., "Partial Match Retrieval using indexed descriptor files," Communications of ACM, vol. 23, no. 9, pp.522-528, 1980.

[16] H.-S. Yong and S. Lee, "Signature File Generation Techniques for Query Processing in Object-Oriented Databases," in ACM Computer Science Conference, Phoenix, March 1994, pp. 364-371.

[17] 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.

[18] M. Roth, H. Korth, and D. Batory, "SQL/NF: A Query Language for 1NF relational Databases," Information Systems, vol. 12, no. 1, pp. 99-114, 1987.

著 者 紹 介

龍 煥 昇(正會員)

1983年 2月 서울대학교 컴퓨터공학과 졸업. 1985年 2月 서울대학교 대학원 컴퓨터공학과 공학석사 1985年 1月 ~ 1989年 2月 한국전자통신연구소 연구원. 1989년 3月 ~ 1994年 2月 서울대학교 대학원 컴퓨터공학과 공학박사. 1991年 4月 ~ 1993年 6月 서울대학교 중앙교육연구전산원 연구조교. 1993年 3月 ~ 현재 서울대학교 컴퓨터신기술공동연구소 특별연구원. 주관심 분야는 객체 지향/중첩 릴레이션 데이타베이스, 4세대 언어, 전문가 시스템 등임.

李 錫 浩(終身會員)

1964年 연세대학교 정치외교학과 졸업. 1975年과 1979年에 미국 텍사스대학교 전산학 석사와 박사학위 취득. 1979年 ~ 1982年 한국과학원 전산학과 조교수. 1982 年~ 1986年 한국정보과학회 논문편집위원장. 1986年 ~ 1988年 한국정보과학회 부회장. 1988年 ~ 1989年 IBM Watson 연구소 객원교수. 1989年 ~ 1991年 서울대학교 중앙교육연구전산원 원장. 1994年 현재 한국정보과학회 회장. 1982年 부터 현재까지 서울대학교 컴퓨터공학과 교수로 재직중이며 데이타베이스, 화일처리, 자료구조 등을 강의