시그니쳐를 이용한 내포 애트리뷰트 인덱스 기법에 관한 연구

○
용 환승, 김 형주, 이 석호
서울대학교 컴퓨터공학과

A Study on Indexing Techniques For Nested Attributes using Signatures

Hwan-Seung Yong, Hyoung-Joo Kim, Sukho Lee
Dept. of Computer Engineering, Seoul Nat'l University

요 약

응용이 복잡하여짐에 따라 기존의 릴레이션 데이타베이스는 내포된 구조를 허용하는 내포 릴레이션 데이타베이스에 대하여 연구가 이루어지고 있으며 이러한 내포 릴레이션이 가지는 내포 애트리뷰트는 새로운 인덱스 기법을 필요로 한다 본 고에서는 내포 릴레이션 데이타베이스 시스템에서 내포 애트리뷰트에 대한 인덱스로 시그니쳐 화일을 사용한 기법을 제시하였다
이 방식은 기존의 역인덱스 기법과 비교하여 볼 때 낮은 저장 공간을 필요로 하며 또한 인덱스 정보 관리가 간단하다는 장점을 가진다 본 기법은 내포 단계를 유지하는 방법과 무시하는 방법으로 나누어 각각에 대한 장단점을 비교하여 기술하였다

1 서론

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 내포 관계 데이타베이스(Nested Relational Database)시스템이 있다[1][2][3][4]

내포 릴레이션(nested relation)은 확장된 애트리뷰트(extended attribute)를 가지는 릴레이션이고, 확장된 애트리뷰트는 원자값 애트리뷰트(atomic attribute)이거나 릴레이션값 애트리뷰트(relation-valued attribute)이고 이 때 원자값 애트리뷰트는 정수나 실수, 문자열(string)의 데이타 타입을 갖는 애트리뷰트이고 릴레이션값 애트리뷰트는 릴레이션의 인스턴스(instance)를 애트리뷰트의 값으로 가질 수 있는 애트리뷰트를 말한다. 릴레이션값 애트리뷰트의 임의의 값은 서브 릴레이션(sub-relation)이라 하고, 서브 릴레이션의 임의의 투플은 서브 투플(sub-tuple)이라고 한다. 내포 릴레이션에서 서브 투플이 아닌 투플은 최상위 투플(top-level tuple)이라고 한다

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

<그림 1.1> 은 내포 릴레이션 University를 보여 주고

Dept-Name	Course-Info		
	Semester	Courses	
		Cname	Credit
Computer	1	Fortran	1
		Logic	2
		English	3
	2	Archi.	3
		O S	2
Mechanics	1	Math	3
		Physics	3
		English	3
	2	Dynamics	3
		Fortran	3

<그림 1.1> 예제 내포 릴레이션 'University'

위의 예에서는 두 애트리뷰트 Dept-Name 원자값 애트리뷰트, Course-Info 릴레이션값 애트리뷰트가 정의되어 있다. 내포 애트리뷰트로는 University Course-Info Semester가 있으며 내포 조건식의 예로는 University Course-Info Courses Cname = "English"를 들 수 있다

내포 애트리뷰트에 대한 기존의 연구 결과로는 Bertino[5]와 Deshpande[6]을 들 수 있는 데 두 가지 모두 고전적으로 많이 사용되는 역인덱스(inverted index)를 사용한다는 점에 있어서 공통점이 있다 Bertino[5]는 내포 릴레이션 문제가 객체 지향 데이타베이스 에서 객체간의 내포 관계에서도 동일한 문제로 파악하고 두 경우 모두에게 적용가능한 내포 인덱스 기법을 제시하였다 이 방법은 보조키(secondary key)에 대한 가장 고전적

인 방법인 역 인덱스(inverted index)[7]를 사용한다 즉 B-트리 인덱스 구조를 조금 변형하여 트리의 중간 노드 구조는 동일하지만 마지막 리프 노드에는 하나의 키값에 대해 여러 개의 투플 주소를 보관하도록 변형하여 사용하고 있다

Deshpande[6]는 내포 릴레이션의 경우에서 발생하는 모든 인덱스 요구를 통합하여 제공할 수 있는 도메인 접근 인덱스 기법을 제시하였다. 여기서는 VALTREE라는 하나의 내포 릴레이션을 이용하여 도메인 접근 방식으로 인덱스를 제공한다 여기서는 하나의 키값과 연관된 모든 투플의 주소에 대해서 최상위 투플뿐아니라 모든 서브 투플들까지 저장한다 Korth[8]는 이 방식을 Exhaustive 방식이라고 부르는 데 그 이유는 키로 정의되는 모든 애트리뷰트에 대해서 이에 속하는 모든 키값과 해당 키값을 포함하는 모든 투플 주소를 연관하여 저장한다는 의미에서 부르고 있다

기존의 내포 애트리뷰트 인덱스 기법으로 제시된 내포 인덱스나 Exhaustive 방식의 공통점은 모두 기본 사용 기법이 역인덱스를 이용하고 있다는 것이다 역인덱스 방식은 검색 속도가 우수하기 때문에 가장 많이 사용되어 왔다 그러나 문제점으로 들 수 있는 것은 인덱스를 위한 부가 저장 공간이 많이 필요하다는 점과 인덱스 정보의 관리가 복잡하다는 것이다 일반적으로 역인덱스의 경우 50%-300%의 부가 공간이 소요된다고 알려져 있다 그리고 각 키에 연관된 투플 주소의 리스트 관리와 레코드의 삽입, 삭제에 따른 인덱스 구조의 변경은 복잡한 알고리즘을 필요로 한다[9][10]

본 논문에서는 이와 같은 문제점을 해결할 수 있는 인덱스 방식으로 기존의 다중키를 지원하는 인덱스 기법인 시그니처 방식을 내포 애트리뷰트에 적용할 수 있도록 애트리뷰트 시그니처와 릴레이션 시그니처의 개념을 정의하여 사용하였으며 그에 따른 인덱스 화일의 구성 방법을 제시하였다

2 다중 키 인덱스를 위한 시그니처 화일

2 1 시그니처 화일 인덱스

일반적으로 시그니처는 여러 개의 값들을 각각 해싱하여 얻어지는 결과를 비트 단위로 OR (Superimposed Coding)연산을 통해 생성한다 H S Yong, S.K Park, G.D Hong과 같이 세 값이 주어지고 각각의 해싱 결과가 아래와 같을 때 만들어 지는 SCW의 예는 다음과 같다.

값	해싱 결과
H S.Yong	1001 0000 1000 0001
S K.Park	1000 1100 0000 0100
G.D.Hong	1000 0000 1100 1000

SCW1 = 1001 1100 1100 1101 = 시그니처1

이렇게 만들어진 SCW1이 있으면 우리는 하나의 S K Park이란 값이 이 SCW1에 있는 지 여부를 확인할 수 있다 확인하는 방법은 동일한 방법으로 해싱하여 SCW인 SCW0가 만들어졌다고 하면 이 SCW0에 1로 세트된 비트를 SCW1과 비트단위로 비교하여 모두 일치하면 확인이 되는 것이다. 이러한 확인 기능은 하나의 값이 주어진 경우뿐만아니라 2개 이상의 값이 있는 지의 여부도 주어진 값들에 대해서 SCW를 만들어 비교하기 때문에 다중 키를 이용한 검색이 가능하여진다 다시말하면 하나의 SCW는 그 SCW를 만드는 데 참여한 모든 값들에 대한 정보를 가지고 있다고 볼 수 있다

이러한 시그니처를 레코드 단위로 만들어 저장한 시그니처 화일은 사용자의 질의가 주어지면 질의문속에 있는 애트리뷰트의 값들을 다시 같은 방법으로 해싱한 후 Superimposed Coding하여 레코드 데이타 화일과 비교하는 것이 아니라 시그니처 화일과 비교하게 된다 이 비교 연산을 통해 매치(Match)가 이루어진 시그니처에 해당하는 레코드들은 전부는 아니지만 질의어에 주어진 값들을 가지고 있는 레코드들인 것이다 그러므로 이 레코드들의 주소 집합에 대해 실제로 검사과정을 통해 레코드를 검색하여 비교 확인함으로써 최종 검색 레코드로 결정되는 것이다 검색 성능에 주요 영향을 주는 것은 시그니처 화일의 크기이다 이 인덱스 방식에서는 전체 시그니처 화일을 모두 읽어서 비교 검사를 해야 하는 데 시그니처 화일이 원래의 데이타 화일의 10%에 불과하지만 데이타 화일 자체가 매우 큰 경우는 커다란 입/출력을 필요로 한다 이 문제를 해결하기 위해 두가지 방법이 제시되었는 데 먼저 Pfaltz[11]에서는 시그니처 화일을 비트 스트링이 아니고 비트 슬라이스(Bit slice)로 만들어 저장하는 방식을 제안했다

2 2 역 인덱스와의 비교

시그니처 인덱스와 기존의 전통적인 역인덱스 기법을 비교하는 데 있어서 기본 접근 원칙에 따라 다음과 같이 도메인 지향 인덱스와 릴레이션 구조 지향 인덱스로 나누어 설명할 수 있다

2 2 1 도메인 지향 인덱스(Domain oriented index)

역인덱스는 키값을 기준으로 그 키값을 가지는 레코드의 주소를 리스트로 기록함으로써 구성한다 그러므로 역인덱스의 엔트리 수는 키값이 속한 도메인의 크기에 영향을 받게 된다 릴레이션의 레코드 구조가 내포되는 것과는 상관없이 구성된다 또한 인덱스를 만들고자 하는 레코드가 가지는 모든 키값들에 대해서 키값과 그 키값을 가지는 레코드들의 주소를 리스트로 유지하므로 하나의 레코드 주소는 여러개의 키값들에 분산되어 저장된다 그러므로 레코드단위의 연산 즉 임의의 레코드가 삽입되거나 삭제되는 경우 그 레코드에 속한 모든 키값들에 대해 각각 별도 인덱스 정보를 찾아서 레코드 주소를 삽입하거나 삭제하여야 한다

2 2 2 릴레이션 구조 지향 인덱스

역인덱스와 달리 시그니처 방식에서는 시그니처 레코드로 구성되는 데 이 레코드는 데이타 레코드와 데이타 레코드와 1:1의 관계로 구성된다 즉 레코드의 키값이 속한 도메인과 상관없이 데이타 레코드의 수와 관련이 있으며 내포 릴레이션과 같이 릴레이션의 구조가 복잡해짐에 따라 시그니처 레코드의 구성도 릴레이션의 구조에 연관된다 데이타 레코드 단위의 연산을 수행하는 경우 해당 레코드에 속하는 각 키값들에 따라 별도의 인덱스를 검색하는 과정이 필요없고 해당 인덱스 레코드를 삽입하거나 삭제하면 된다

역 인덱스 방식에 비해 시그니처 화일 방법이 갖는 또 다른 장점은 다음과 같다 첫째로 구현하기가 간단하다는 점이다 둘째로는 여러 개의 키값이 주어지는 경우 모든 키값을 가지는 교집합(intersection)을 처리해야 하는 데 역인덱스와 달리 각 시그니처에 이미 교집합 정보가 있으므로 별도의 교집합 처리가 불필요하다는 점이다 세째로는 검색 연산에 필요한 비트 단위의 연산을 고속화하기위한 하드웨어 구성이 용이하다는 점이다 [Faloutsos]

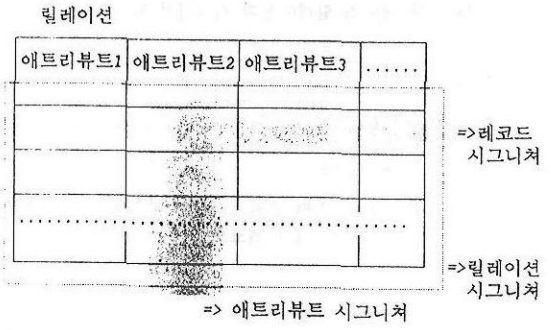
3 내포 애트리뷰트에 대한 시그니처 인덱스

3 1 애트리뷰트 시그니처와 릴레이션 시그니처

단일 내포 애트리뷰트에 대한 인덱스를 지원하기 위해 애트리뷰트 시그니처를 정의해야 한다 먼저 임의의 릴레이션에 대한 애트리뷰트 시그니처(*Attribute Signature*)는 그 애트리뷰트가 내포된 애트리뷰트가 아니고 원자값 애트리뷰트인 경우에만 정의되며 그 애트리뷰트에 속한 모든 값들에 대한 *Superimposed Code*화한 값으로 정의된다 레코드 시그니처는 <그림 3.1>과 같이 하나의 레코드에 속하는 여러 애트리뷰트의 키값들을 *Superimposed Code*화하여 구하는 데 이와는 달리 애트리뷰트 시그니처는 수직으로 동일한 애트리뷰트에 속하는 값들을 취한다

레코드, 애트리뷰트 시그니처와 함께 또한 우리는 릴레이션 시

그니쳐(*relation signature*)를 정의할 수 있는 데 이는 한 릴레이션에 속하는 모든 원자값 타입의 애트리뷰트가 가지는 값들을 *Superimposed Code*한 것이다.



<그림 3.1> 레코드, 애트리뷰트, 릴레이션 시그니쳐

3.2 용어 정의

3.2.1 경로와 경로 표현

내포 릴레이션에 속하는 임의의 서브 투플이나 서브 릴레이션을 설명을 위하여 지칭하거나 주소를 나타내기 위하여 경로(path)를 정의할 수 있다. 경로 정보를 표현하기 위해서는 다음과 같은 방식을 사용한다. 만일 t가 릴레이션 R의 임의의 투플을 나타낸다고 하면 t 다음의 아래 소문자(subscript)는 투플 순서에 따른 번호를 나타낸다. 즉 t_1은 R의 첫번째 투플을 나타낸다. 또한 t 다음의 위 소문자(superscript)는 그 투플내에서 애트리뷰트의 순서에 따른 위치를 나타낸다. 그러므로 t^2 는 두번째 애트리뷰트의 값을 나타낸다. 이와 같은 아래/위 소문자는 투플 지시자에 연이어 나타내어 지는 데 이러한 경우 순서대로 해석하면 된다. $t_1{}^2$는 릴레이션 R의 첫째 투플에서 두번째 애트리뷰트를 가리킨다.

3.2.2 수퍼 릴레이션과 베이스 릴레이션

임의의 서브 투플 t에 대한 수퍼 릴레이션(*super-relation*)은 그 투플을 서브 투플로 포함하는 모든 상위 릴레이션을 나타낸다. 마찬가지로 서브 릴레이션 r에 대한 수퍼 릴레이션은 그 릴레이션을 서브 릴레이션으로 포함하는 릴레이션을 말한다. 내포된 단계가 여러 단계로 이어질 때에는 임의의 투플이나 릴레이션에는 여러 개의 슈퍼 릴레이션이 있을 수 있다. 이 때 바로 한 단계 위의 수퍼 릴레이션을 특히 첫째 수퍼 릴레이션(*first super-relation*) 이라고 한다.

반면에 임의의 서브 투플 t가 최상위 투플이 되는 릴레이션을 베이스 릴레이션(base-relation)이라고 한다.

예로 내포 애트리뷰트 NA1이 다음과 같다고 하자.

NA1 : *'University.Course-Info.Courses.Cname'*

내포 애트리뷰트의 베이스 투플(base tuple)은 최종 애트리뷰트의 값을 원자값으로 가지는 서브 투플을 나타낸다. 형제 베이스 릴레이션은 동일한 내포 애트리뷰트의 베이스 릴레이션중에서 같은 최상위 투플에 속하는 베이스 릴레이션들을 말한다.

Fortran	3
Logic	2
English	3

(a) 베이스 릴레이션1

Archi.	3
O.S	2

(b) 베이스 릴레이션2

<그림 3.2> <그림 1.1> 릴레이션의 투플 t_1에서 내포 애트리뷰트 NA1 에 대한 형제 베이스 릴레이션

내포 단계가 L인 내포 애트리뷰트가 갖는 베이스 릴레이션의 갯수가 중요한 데 이 것을 계산하기 위하여 평균적인 서브 투플의 갯수가 필요하게 된다. 평균 서브 투플의 갯수는 애트리뷰트의 도메인마다 서로 다른 데 본 고에서는 모든 내포 단계에서의 서브 투플수는 서로 같다고 가정하고 이를 N_{st}로 나타낸다.

그러면 임의의 내포 애트리뷰트가 내포 단계가 L인 경우 이 내포 애트리뷰트에 대한 베이스 릴레이션의 갯수는 $N_{st}{}^{L-1}$ 과 같으며 베이스 투플의 갯수는 $N_{st}{}^{L}$이 된다.

3.3 애트리뷰트 인덱스의 특징

이와 같이 구성한 시그니쳐 인덱스는 사용자가 내포 애트리뷰트에 대해서 키값을 가지고 검색할 수 있으며 특히 역인덱스보다 우수한 점은 동일 내포 애트리뷰트에 대해서 한 개 이상의 키값을 제공되는 AND 조건에 대해서도 역인덱스 방식에서와 같이 별도의 교집합(intersection) 연산없이 단일 키값의 경우와 동일하게 간단하게 처리할 수 있다는 점이다.

1NF 릴레이션에서 다중 키 인덱스 방법(이하 다중 키 방법)으로 시그니쳐 화일을 사용한 경우와 지금 위에서 기술한 대로 내포 애트리뷰트(이하 내포 애트리뷰트 방법)에 적용되는 방법 사이에는 세가지 다른 점이 있다.

첫째는 다중키 방법에서는 레코드 시그니쳐를 만드는 데 참여하는 키값들의 수가 일정한 데 반하여 내포 애트리뷰트에서는 하나의 애트리뷰트 시그니쳐를 구성하는 데 사용한 키 값들의 수가 가변적이라는 점이다.

두번째는 한 시그니쳐에 참여하는 키값들의 도메인에 있어서 구별된다. 즉 다중키 방식에서는 각 키값들이 일반적으로 거의 상이한 도메인을 갖는 데 반하여 내포 애트리뷰트 방식에서는 모든 키값들이 동일한 도메인을 갖는다.

마지막으로 N개의 레코드가 있을 때 다중키 방식에서는 시그니쳐 레코드의 수가 반드시 N개 인 데 반하여 내포 애트리뷰트 방식에서는 이보다 더 클 수 있다는 점이다.

3.4 내포 애트리뷰트 인덱스 구성 방법

내포 단계 1인 경우에는 베이스 릴레이션과 최상위 투플과 1:1의 관계가 있으므로 인덱스 시그니쳐를 구성하는 문제가 비교적 간단하지만 내포 단계가 1이상인 경우에는 하나의 최상위 투플에 속하는 베이스 릴레이션의 갯수가 N개가 되므로 다음과 같은 추가 고려 사항이 있게 된다. 즉 이들 형제 베이스 릴레이션을 각 베이스 릴레이션까지의 경로와 함께 별도로 취급하여 인덱스를 구성할 수도 있고 다른 방법으로는 형제 베이스 릴레이션을 하나의 릴레이션으로 만들어 하나의 최상위 투플 주소와 함께 인덱스를 구성하는 두가지 방법을 생각할 수 있다.

본 고에서는 이 두가지 방법에 대해서 각각 내포 단계 무시 방법과 고려 방법으로 나누어 설명한다.

3.4.1 내포 단계 무시 인덱스 방법(Nesting Level Ignoring)

이 방법은 내포 애트리뷰트의 내포 단계를 고려하지 않고 인덱스를 구성하는 방법으로써 결과로 만들어지는 시그니쳐는 한 최상위 투플에 대해 하나씩 생성되게 되며 오직 최상위 투플 주소만이 저장된다. 시그니쳐는 내포 애트리뷰트의 내포 단계에 따라 하나의 최상위 투플에 포함된 여러 개의 베이스 릴레이션에 대해서 릴레이션에 대한 UNION 연산을 수행하여 하나의 릴레이션으로 만든 후 이 릴레이션(이하 유니온 베이스 릴레이션이라 부른다)에 대해 애트리뷰트 시그니쳐를 구성하는 것이다.

예로서 <그림 1.1>의 릴레이션에 대해서 내포 애트리뷰트 NA1의 인덱스를 내포 단계를 무시하고 만드는 경우를 생각해 보자.

첫째 최상위 투플인 t_1의 시그니쳐 레코드를 만들기 위해 먼저 베이스 릴레이션에 대한 유니온 베이스 릴레이션은 <그림 3 3>의 (a)와 같으며 이에 따른 시그니쳐 레코드는 <그림 3.3>의 (c)와 같다

Fortran	3
Logic	2
English	3
Archi	3
O.S	2

(a) <그림 1 1>의 첫째 투플 t_1의 유니온 베이스 릴레이션

```
Fortran : 1000 0001 0000 0001
Logic   : 0000 0001 0001 1000
English : 0100 1000 0000 0100
Atchi.  : 0010 1000 0000 0100
O.S       0000 0000 0100 1100
-------------------------------
$1      : 1110 1001 0101 1101
```

(b) (a)의 애트리뷰트 시그니쳐 $1

1110 1001 0101 1101	t_1

(c) (a)의 시그니쳐 레코드

<그림 3 3> 유니온 베이스 릴레이션과 시그니쳐 레코드 예

이 때 내포 단계를 고려하지 않는 경우에 있어서 하나의 시그니쳐 레코드가 가지는 키값들의 갯수 s는 하나의 최상위 투플이 가지는 인덱스 서브 투플의 수와 같으므로 N_{st}이 된다.

3 4 2 내포 단계 고려 인덱스 기법(Nesting Level Preserving)

이와 같은 예에서 다음과 같은 두 개의 질의를 생각해보자.

Q3 Fortran'과목을 개설하고 있는 Department의 이름을 검색하라

Q4 'Physics' 과목을 개설하고 있는 Department의 이름과 Semester의 정보를 검색하라

Q1과 Q2는 NA1에 대한 동일한 내포 조건식을 갖는 면에서는 유사하다 이와 같은 질의를 효과적으로 지원하기 위해서 만일 이 내포 애트리뷰트에 대해서 인덱스가 만들어져 있다면 좋을 것이다 그러나 인덱스에 저장되는 정보의 형태에 따라 실제 수행과정에 차이가 있을 수 있다. Q1에서는 검색되는 정보가 최상위 애트리뷰트인 Dept-name만을 필요로하기 때문에 인덱스에는 최상위 투플 주소만 가지고 있는 것으로 충분하다고 볼 수 있다 그러나 Q2의 경우에는 Course-Info 릴레이션값 애트리뷰트의 소속 애트리뷰트인 Semester 정보도 요구하게 되므로 만일 최상위 투플 주소만 가지고 있게 되는 경우 그 주소를 가지고 다시 하위 투플들을 찾아야만 한다 이와 같은 전위 순회(forward traversal)는 특히 서브 투플들의 저장 구조가 최상위 투플과 별도 저장되어 있는 경우에는 매우 많은 소요 시간을 필요로 하므로 인덱스의 효과가 크게 감소된다

이 방법은 위 절에서 제시한 방법과 다른 점은 애트리뷰트 시그니쳐가 각 서브 릴레이션마다 별도로 구성되며 또한 해당 서브 릴레이션까지의 경로정보도 함께 인덱스에 저장되는 점이다. 이와같이 함으로써 투플의 실제 검사 단계와 Q2 형태의 질의를 전위 순회가 필요 없이 빠르게 제공할 수 있다는 장점이 있다

투플 t_1에 대해서 두 개의 베이스 릴레이션이 있으므로 이 들에 대해 각각 시그니쳐를 만드는 과정과 알고리즘은 다음과 같다

Fortran	3
Logic	2
English	3

```
Fortran : 1000 0001 0000 0001
Logic   : 0000 0001 0001 1000
English : 0100 1000 0000 0100
-------------------------------
$1      : 1110 1001 0101 1101
```

Archi	3
O.S	2

```
Atchi.  : 0010 1000 0000 0100
O.S     : 0000 0000 0100 1100
-------------------------------
$2      : 0010 1000 0100 1100
```

(a) 두 베이스 릴레이션과 시그니쳐 $1, $2

1110 1001 0101 1101	$t_1{}^2{}_1$
0010 1000 0100 1100	$t_1{}^2{}_2$

(b) 시그니쳐 레코드

<그림 3.4> <그림 1.1>의 첫째 투플 t_1의 두 베이스 릴레이션과 시그니쳐 레코드

4 결 론

본 논문에서는 기존의 1NF 릴레이션에서 다중 키 인덱스기법으로 제시되어 사용되는 시그니쳐 기법을 이용하여 내포 릴레이션에서 내포 애트리뷰트에 대한 인덱스를 구성하는 방법을 제시하였다 본 기법은 적은 저장공간을 필요로하며 다중키에 대해 효과적으로 지원할 뿐만아니라 구성과 관리가 비교적 간단하다는 장점을 가진다 또한 구체적으로 내포 단계를 고려하는 경우와 고려하지 않는 경우를 구별하여 각각에 대해 설명하였다.

내포 릴레이션의 내포 애트리뷰트는 일반적으로 높은 선택력을 이 중요한 설계 기준이 된다고 할 수 있으므로 본 방법이 매우 적합하다고 생각한다. 추후 연구로는 다중 내포 애트리뷰트와 집합값을 갖는 애트리뷰트를 지원하는 방법과 함께 성능 평가를 수행하는 것이 필요하다

참고문헌

[1] Abiteboul,S et al, "Towards DBMSs For Supporting New Applications," Proc of VLDB, Kyoto, August 1986, pp 423-435

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

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

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

[5] Bertino, E , Kim, W , "Indexing Technique for Queries *on Nested Objects," IEEE Trans on Knowledge and* Data Eng , Vol.1, No.2, 1989, pp 196-214

[6] Deshpande, A , Gucht, D., "An Implementation for Nested Relational Databases," Proc of VLDB, Los Angeles, USA, 1988, pp 76-87

[7] Knuth, D , "The Art of Computer Programming, Vol 3, Searching and Sorting, Addison-Wesley, Reading, Mass , 1973

[8] Korth, H , "Optimization of Object-Retrieval Queries," Advances in Object-Oriented Database Systems, LNCS 334, *Springer-Verlag, 1988, pp.352-357*

[9] Christodoulakis, S , Faloutsos, C., Design Considerations for a Message File Server," IEEE Software Engineering, Vol 10, No 2, March 1984, pp.201-210

[10] Faloutsos, C , "Access Methods for Text," ACM Computing Surveys, Vol 17, No.1, March 1985, pp 49-74

[11] Ptaltz, L , et al , "Partial Match Retrieval using indexed descriptor files," Comm. of ACM, Vol 23, No 9, 1980, pp 522-528