# 객체 지향 데이타베이스에서 시그니쳐를 이용한 전진 운행 질의 처리 기법

## (Applying Signatures for Forward Traversal Query Processing in Object-Oriented Databases)

용 환 승† 이 석 호†† 김 형 주††
(Hwan-Seung Yong) (Sukho Lee) (Hyoung-Joo Kim)

**요 약** 객체 지향 데이타베이스에서 중첩 술어를 가지는 질의를 처리하기 위해서는 전진 운행법이 사용된다. 이 논문에서는 이러한 전진 운행을 신속히 처리하기 위해서 참조된 객체의 객체 시그니쳐를 참조 객체에 저장하여 전진 운행 탐색시 먼저 시그니쳐를 검사한 후에 만족하는 경우에만 참조된 객체를 검색하는 기법을 제시하였다. 이 기법은 적은 저장 비용으로 모든 애트리뷰트에 대한 조건식을 신속히 처리할 수 있으며 중첩 인덱스와 혼합하여 질의 처리가 가능하다는 장점이 있다.

***ABSTRACT*** Forward traversal methods are used to process queries having nested predicates in object-oriented databases. To expedite the forward traversal, a signature replication technique is proposed. When an object refers to other objects through its attribute, the object signature of the referred object is stored into the referring object. Using object signatures, nested predicates can be checked without inspecting referred objects.

## 1. 서 론

객체 지향 데이타베이스에서는 한 객체의 애트리뷰트가 값으로 다른 객체의 객체지시자(object identifier : OID)를 사용하므로써 객체간의 참조(refer) 관계를 형성한다. 이와 같은 객체간의 참조 관계는 객체들에 대해 애트리뷰트를 통한 계층 구조를 구성하게 되는 데 이를 클래스 애트리뷰트 계층(class attribute hierarchy)[1]이라고 하며 그 예가 그림 1에 있다.

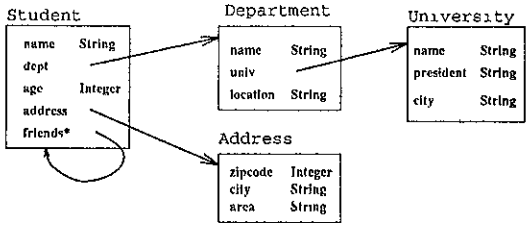



그림 1 클래스 애트리뷰트 계층

하나의 질의문에 의해 주어지는 술어(predicate)들은 클래스 애트리뷰트 계층에 속하는 여러 클래스의 애트리뷰트에 대해서도 주어질 수 있는 데 이와 같은 술어들을 중첩 술어(nested predicate)라고 한다[1]. Student 클래스에 대한 중첩 술어의 예는 Student.dept.name="Computer"와 같다. 이러한 중첩 술어들을 가지는 질의를 수행하기 위한 기법으로 전진 운행법(forward traversal), 후진 운행법(backward traversal), 그리고 혼합 운행법(mixed traversal)이 제시되었다[2].

그러나 이와 같은 운행법을 이용하여 질의를 처리하는 경우에도 많은 처리 시간을 필요로 하기 때문에 별도의 기법들이 제시되었다[1,3]

신속한 질의 처리를 위해 제시된 기본적인 방법으로 중첩 인덱스(nested index) 기법이 있으며[1], 전진 운행시 하위 객체들의 검색을 줄이기 위해 애트리뷰트 중복(field replication) 기법[3]이 있다.

애트리뷰트 중복 기법은 참조 관계를 통해 자주 검색되는 하위 객체의 특정 애트리뷰트 값을 상위 객체에다가 중복하여 저장시켜 놓음으로서 전위 탐색과정에서 하위 객체에 대한 검색을 하지 않아도 된다는 장점이 있다. 그러나 단점으로는 여러 개의 하위 객체 애트리뷰트에 대해 중복시키고자 하는 경우 각각에 대해 별도로 상위 객체에 저장하므로, 값 자체를 중복 저장하는 데 따르는 많은 저장 비용을 들 수 있다.

인덱스를 사용하는 기법은 후진 운행 탐색을 신속히 제공할 수 있는 데 애트리뷰트 중복 기법과 마찬가지로 저장 공간과 인덱스 관리 비용때문에 클래스 애트리뷰트 계층에 속한 모든 애트리뷰트에 대해 인덱스를 만들어 줄 수는 없으므로 반드시 필요한 애트리뷰트에 대해

†종신회원 서울대학교 컴퓨터공학과
††종신회원 서울대학교 컴퓨터공학과 교수
논문접수 1993년 5월 26일
심사완료 1993년 11월 8일

서만 선택적으로 정의하여 사용하게 된다

이 논문에서는 중첩 조건을 가진 질의를 신속히 처리하기 위해 새로운 방법으로 전진 운행 탐색의 비용을 줄이기 위해 피참조 객체의 애트리뷰트 값들로 구성되는 시그니쳐(signature)[4]를 참조 객체에 저장하는 방법을 제시한다. 이 방법을 사용하면 적은 저장 공간으로 참조 관계를 통해 피참조 객체를 검색할 때 이미 저장된 피참조 객체의 시그니쳐를 이용하여 해당 클래스에 대한 질의 조건을 검사할 수 있다 그리고 만족하는 값이 있다고 판단되는 경우에만 전진 운행을 계속 수행하게 되므로서 불필요한 객체 검색을 생략할 수 있다. 또한 시그니쳐가 가지는 적은 저장 비용 특성[5]은 참조 관계에 따르는 피참조 객체의 모든 애트리뷰트의 값들에 대해 시그니쳐로 만드는 것이 가능하기 때문에 특정한 애트리뷰트에 대한 조건만이 아닌 임의의 애트리뷰트에 대한 조건을 가지는 중첩 질의어에 대해서도 신속한 전진 운행 탐색을 가능하게 한다. 그리고 중첩 인덱스가 함께 제공되는 경우에는 인덱스와 시그니쳐를 모두 사용하는 것이 가능하다는 특징이 있다.

이 논문의 구성은 2장에서 문제 정의와 해결을 위해 객체 지향 데이타베이스에서 질의 처리를 위한 전진 운행법과 시그니쳐에 대해 기술하였으며 3장에서 시그니쳐를 이용한 전진 운행법을 제시한다. 그리고 4장에서 성능을 평가하기 위한 모델과 평가 결과를 제시하였으며 5장에서 앞으로의 연구방향을 제시한다.

## 2. 전진 운행법과 시그니쳐

### 2.1 전진 운행법

객체 지향 데이타베이스에 대한 질의어를 질의 그래프(query graph)로 표현하여 처리하는 방법으로는 다음과 같은 3가지가 제시되었다[2]. 그리고 질의 처리 방법을 선택하는 데 있어서 중첩 인덱스가 지원되지 않는 경우에 있어서는 전진 운행법을 선택하는 것이 좋다고 제시했다.

- 전진 운행법 : 질의 그래프에서 루트 클래스를 먼저 탐색하고 리프 클래스들을 나중에 탐색한다.
- 후진 운행법 . 질의 그래프에서 리프 클래스를 먼저 탐색하고 루트 클래스들을 나중에 탐색한다.
- 혼합 운행법 전진과 후진 운행법을 혼합하여 탐색한다.

전진 운행 탐색과정을 설명하기 위해서 그림 1에 있는 예제 클래스 애트리뷰트 계층을 사용한 질의문의 예는 Q1과 같고 그에 따른 질의 그래프는 그림 2에 있다

Q1 . "Seoul"에 위치한 대학의 "Computer" 학과에 속한 학생들을 검색하라.

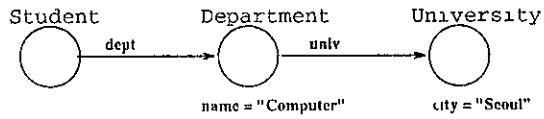



그림 2 Q1에 대한 질의 그래프

만일 이 질의문 Q1의 조건에 주어진 두 애트리뷰트에 대해 중첩 인덱스가 정의되어 있지 않은 경우에는 이 질의어를 처리하기 위해서는 전진 운행법을 선택하게 된다. 이 방법에서는 먼저 모든 Student 클래스의 객체들을 검색해야 하고 이들 각 객체에 대해 애트리뷰트 dept의 값을 통해 다시 Department 클래스의 객체를 검색하여 그 객체의 name 애트리뷰트 값이 "Computer"인지 검사해야 한다. 이 때 이 조건을 만족하지 못하면 다른 Student의 객체에 대해 똑같은 동일한 검색 과정을 반복한다. 그러나 만일 조건을 만족하는 경우에는 Department 객체의 univ 애트리뷰트 값을 통해 University 클래스의 객체를 검색한다. 그리고 검색된 University의 객체의 애트리뷰트인 city의 값이 "Seoul"인지 검사한다.

### 2.2 시그니쳐

다중키를 지원하기 위해 제시되었던 시그니쳐(signature) 기법은 최근 들어 문서 검색에 많이 적용되며[4] 또한 다매체(Multi-media) 데이타베이스등에도 사용되고 있다[6,7].

데이타로부터 시그니쳐를 만드는 방법은 여러가지[8]가 있지만 본 논문에서는 Superimposed Code를 사용하는 것으로 한정한다. 일반적으로 시그니쳐는 여러개의 값들을 각각 해싱하여 시그니쳐의 크기 b 비트위에 k 개의 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 $S_D$는 표 1과 같다.

표 1 시그니쳐 생성 과정 예(b=16, k=4인 경우)

| 값 | 해 | 싱 | 결 | 과 |
|---|---|---|---|---|
| G.D.Hong | 1001 | 0000 | 1000 | 0001 |
| 25 | 1000 | 1100 | 0000 | 0100 |
| Seoul | 1000 | 0000 | 1100 | 1000 |
| 시그니쳐 $S_D$ | 1001 | 1100 | 1100 | 1101 |

이 $S_D$에 'G.D.Hong'이란 값이 포함되어 있는지 여부

를 확인하는 방법은 질의에 주어진 키를 똑같은 방법으로 해싱하여 질의 시그니쳐 $S_Q$를 만든 후 이 $S_Q$에 1로 세트된 비트들이 $S_D$에도 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 $S_D$에 "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 $S_D$와 비교하면 성공하게 되는 데 이와 같은 경우를 매치 오류(False Drop)라고 한다

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. 최근에는 시그니쳐 화일을 이용하여 객체지향 데이타베이스에 대한 인덱스 역할을 하도록 하는 기법이 제시되기도 했다[9]. 그러나 이 기법은 시그니쳐들을 별도의 화일로 구성하여 질의 처리시 전체 시그니쳐 화일을 검색하는 방식으로 이 논문에서 제시하는 바와 같이 시그니쳐를 객체내에 저장하는 기법과는 매우 다르게 된다.

## 3. 시그니쳐를 이용한 전위 탐색 기법

### 3.1 OID 테이블과 객체 시그니쳐

객체에 정의된 단일 값을 가지는 모든 애트리뷰트의 값들을 가지고 생성한 시그니쳐를 객체 시그니쳐라고 한다. 객체 지향 데이타베이스 시스템에서 OID를 구현하는 방법으로 크게 두가지를 들 수 있다. 첫번째는 실제 객체가 저장된 주소를 사용하는 경우와 두번째는 OID를 논리적인 값(surrogate)으로 표현하고 각 OID와 실제 객체의 물리적 주소를 연관하는 테이블(이하 OID 테이블)을 사용하는 경우로 나누어진다. 첫번째 방법은 OID를 가지고 검색하는 경우 검색 성능은 좋지만 저장된 객체의 저장 위치 변경이 빈번해짐에 따라 검색 성능이 저하되게 되며 두번째 방법은 객체의 저장 구조에 대한 독립성을 제공한다는 장점은 있으나 객체를 검색할 때마다 OID 테이블을 검색해야하는 부담이 따르게 된다[10]. OID를 이용하여 객체를 검색하는 과정은 객체에 대해 조건식이 주어지는 경우 그림 3의 (a)에 나타나 있드시 다음과 같은 3단계로 이루어진다.

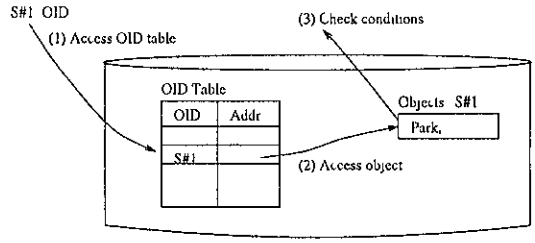
1. OID 테이블을 탐색하여 객체의 실제 주소를 검색한다.
2. 객체의 실제 주소에서 객체의 정보를 검색한다.
3 검색된 객체가 주어진 조건을 만족하는 지 검사한다.

그러므로 최소한 2번의 I/O가 수행되어야 한다.

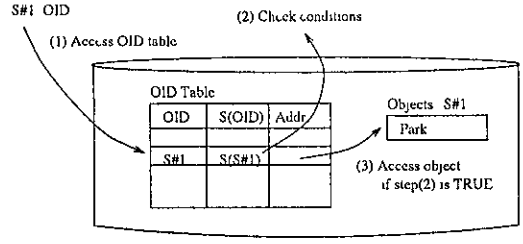
그러나 객체 시그니쳐를 OID 테이블에 OID와 함께 저장하여 놓는 경우에는 다음과 같은 검색 과정으로 이루어지는 데 그림 3의 (b)에 나타나 있다. 이 논문에서는 OID에 대한 객체 시그니쳐를 S(OID)로 표현하여 사용한다.

1. OID 테이블을 탐색하여 객체의 실제 주소와 객체 시그니쳐를 검색한다
2. 검색된 객체가 주어진 조건을 만족하는 지 객체 시그니쳐를 이용해 검사한다.
3. 만일 조건이 만족하는 경우에는 객체의 실제 주소에서 객체의 정보를 검색한다.

그러므로 OID 테이블에 저장된 객체 시그니쳐를 사용하므로써 주어진 OID를 가지는 객체가 조건을 만족하는 지 검사하는 것을 실제 객체를 검색하지 않아도 가능하게 된다. 즉 먼저 OID 테이블에 저장된 객체 시그니쳐를 이용하여 조건이 만족되는 경우에만 실제 데이타를 검색하도록 하므로써 조건이 만족되지 않는 객체의 경우 나머지 단계의 검색과정을 생략시킨다.




(a) 기존의 객체 검색 과정




(b) 시그니쳐를 사용한 객체 검색 과정

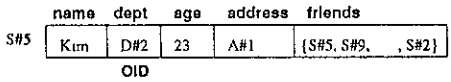
그림 3 OID를 통한 객체 검색 과정 비교

### 3.2 참조 관계와 객체 시그니처

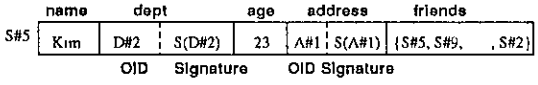
객체 A가 다른 객체 B를 애트리뷰트의 값을 통해 참조한다(refer)는 것은 객체 A에 대한 연산과정에서 이 참조관계를 통해 객체 B의 정보를 검색한다는 것을 뜻한다. 객체 지향 데이타베이스에서 참조 관계를 통한 객체들의 검색 과정은 피참조 객체를 검색하기 위해 별도의 입/출력을 필요로 한다. 이러한 검색 과정은 일반적으로 참조 객체(A)에 피참조 객체(B)에 대한 술어들(중첩 술어 . nested predicate)이 주어지는 경우 이 술어들을 처리하기 위해 수행되는 데 만일 참조 객체에 피참조 객체에 대한 정보가 저장되어 있다면 별도의 검색과정이 필요없이 미리 조건을 검사할 수 있으므로 만족하지 않는 경우에는 검색하지 않아도 되어 신속한 연산 처리가 가능하게 된다. 애트리뷰트 중복기법[3]은 빈번하게 참조되는 피참조 객체의 특정 애트리뷰트 값을 참조 객체에 중복하여 저장하는 기법을 제시하였다

이 논문에서 사용하는 방법은 애트리뷰트 중복 기법과 달리 피참조 객체의 애트리뷰트 값을 저장하는 것이 아니라 객체 시그니처를 저장하는 방법을 사용한다. 즉 객체와 객체 사이에 애트리뷰트를 통한 참조 관계가 만들어 지는 경우 이 논문에서는 OID 뿐만아니라 피참조 객체의 객체 시그니쳐를 함께 가지고 있도록 한다.

그림 4에는 참조 관계에 따른 객체의 저장 구조를 비교하여 Student 클래스에 속한 객체 S#5를 예로 들어 보여 준다.




(a) 기존의 객체 참조 구조




(b) 객체 시그니쳐와 OID를 가진 구조

그림 4 OID를 통한 참조 관계 구조

### 3.3 전진 운행 탐색 알고리즘

전위 탐색을 통한 질의 처리 과정을 설명하기 위해 객체 A가 애트리뷰트를 통해 객체 B를 참조하고, 객체 B에 대해 조건이 주어져 있다고 하자. 이 때 기존의 전위 탐색 과정은 다음과 같다.

1 먼저 참조하는 객체 A를 검색한다.
2. 객체 A의 애트리뷰트를 통해 피참조 객체 B를 검색한다.
3. 객체 B의 데이타를 가지고 조건식을 검사한다.

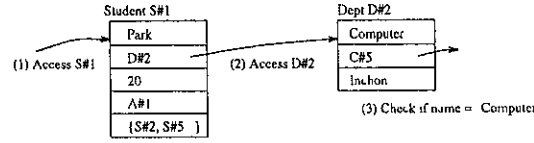
반면에 피참조 객체의 객체 시그니처를 참조 객체에 저장하고 피참조 객체에 대한 조건들이 주어졌을때 질의 처리를 위한 전진 운행 과정은 다음과 같다.

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

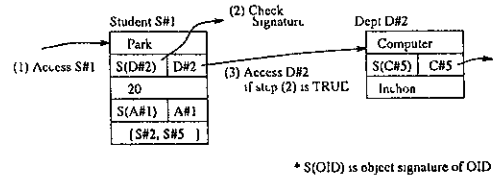
이 과정을 실제 예를 들어보자. 그림 1의 클래스 계층에서 다음과 같은 질의문 Q2가 있다고 하자.

Q2 : "Computer" 학과에 소속된 학생들의 정보를 검색하라

이 질의문을 처리하기 위해 검색한 Student 클래스의 객체로 OID가 S#1인 객체가 dept 애트리뷰트로 Dept 클래스의 객체 D#2를 참조하고 있다고 하자




(a) 기존의 전진 운행 탐색 과정




(b) 시그니처를 이용한 탐색 과정

그림 5 전진 운행 탐색 과정 비교

그러면 기존의 전진 운행 방식에서는 그림 5의 (a)와 같이 먼저 S#1 객체를 검색하고, 다시 D#2 객체를 검색한 후에 이름이 "Computer" 인지 검사해야 한다. 그러나 객체 시그니처를 이용한 탐색 과정에서는 그림 5의 (b)와 같이 S#1 객체를 검색한 후에 D#2의 OID와 함께 저장된 D#2의 객체 시그니처인 S(D#2)를 가지고 이름이 "Computer"인지 검사하여 조건을 만족하는 경우에만 D#2를 검색하고 만족하지 않는 경우에는 검색을 생략한다

인덱스를 사용하는 방법은 조건식이 많아질수록 각 조건식에 대한 인덱스들을 별도로 탐색해야하지만 시그니처를 이용한 전진 운행법은 조건식의 수가 많아질수록 검색 성능은 상대적으로 더욱 좋아지며 검사해야 하

는 시그니쳐는 조건식의 수와는 무관하게 되는 장점이 있다.

### 3.4 객체 시그니쳐의 관리

#### 3.4.1 객체 시그니쳐의 생성 및 저장

객체 시그니쳐는 객체가 생성될 때 만들어져 OID 테이블에 저장된다. 저장된 객체 시그니쳐는 그 객체가 다른 객체에 의해 참조될 때마다 검색되어 참조 객체에 *OID*와 함께 저장된다. 그러므로 하나의 객체가 여러 개의 애트리뷰트를 통하여 각각 다른 객체를 참조하는 경우에는 참조 애트리뷰트 수 만큼 피참조 객체의 시그니쳐가 저장된다.

#### 3.4.2 객체 시그니쳐의 크기

시그니쳐의 크기 b는 객체가 속한 클래스마다 다르게 된다. 일반적으로 b는 한 시그니쳐에 해싱되는 애트리뷰트 값들의 갯수와 시그니쳐의 매치 오류 확률에 의해 정해지며 계산식은 다음과 같다[11].

애트리뷰트의 수를 $nattr$라고 하고 매치 오류의 확률을 $F_d$라고 하면

$$b=(1/log2)^2 \cdot nattr \cdot log(1/F_d)$$

#### 3.4.3 시그니쳐 정보의 갱신

한 객체의 시그니쳐는 그 객체의 값들이 갱신될 때마다 변경된다. 그러므로 그 객체를 참조하는 모든 객체에 저장된 시그니쳐도 변경되어야 한다. 이와 같이 객체가 갱신됨에 따라 그 객체를 참조하는 객체에 저장된 객체 시그니쳐의 부수적인 갱신을 위해서는 임의의 객체가 주어지는 경우주어진 객체를 참조하고 있는 모든 객체들을 쉽게 검색할 수 있어야만 한다. 만일 시스템 자체에서 참조가 일어날 때 참조 객체에 대한 역방향(reverse) 포인터를 제공하는 경우에는 문제가 없지만 그렇지 않는 경우를 처리하기 위해서는 *link object* [3]를 사용하면 된다.

그러나 객체들에 대해 주로 검색 연산이 수행되고 갱신 연산으로는 삽입이나 삭제 연산만이 허용되는 데이타베이스 환경에서는 역방향 포인터를 유지할 필요가 없다.

### 3.5 중첩 인덱스를 이용한 혼합 탐색

이 논문에서 제시하는 기법은 중첩 인덱스(nested index)[1]가 지원되는 경우 오히려 더 효율적으로 질의 처리를 수행할 수 있다. 그 이유는 인덱스는 후진 운행법을 사용하는 반면에 시그니쳐 기법은 전진 운행법을 사용하기 때문에 탐색에 있어서 서로 상충되지 않고 혼합 운행 탐색을 가능하게 한다.

질의문 Q1은 두 개의 중첩 애트리뷰트 Student.dept.name, Student.dept.univ.city 대한 조건을 가지고 있다. 만일 이 두 개의 애트리뷰트들중 하나에 대해 이미 인덱스가 구성되어 있는 경우 시그니쳐 전진 운행법과 혼합하여 질의를 처리하면 더욱 효과적이 된다. 인덱스는 후진 운행 탐색을 지원하므로 인덱스가 지원되는 경우 먼저 그 인덱스를 사용한 후 그 결과를 가지고 시그니쳐를 사용하여 질의 처리를 수행한다.

그림 6에는 중첩 인덱스가 정의되어 있는 유형과 각 유형에 따른 질의 처리과정을 보여준다. 그림에서 A,B, C,D는 클래스를 나타내고, 각 클래스마다 조건이 주어지며 중첩 인덱스가 있는 경우는 주어진 조건의 애트리뷰트에 대해 정의되어 있음을 뜻한다. Type Ⅳ의 경우는 클래스 B에 다중 조건이 주어지고, 그 중 하나의 애트리뷰트에 대해 인덱스가 정의되어 있는 경우를 뜻한다.

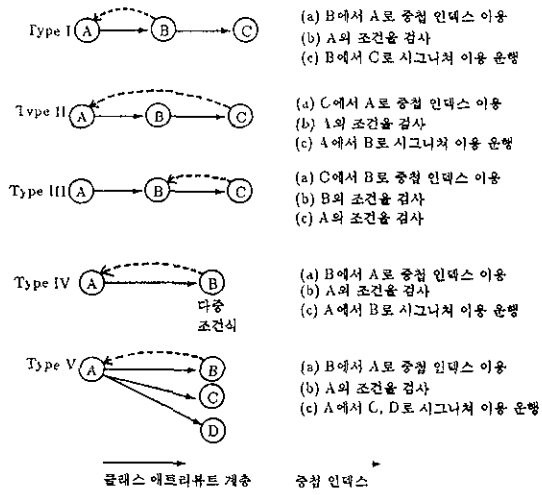



그림 6 중첩 인덱스와 시그니쳐를 이용한 질의 처리

이 그림에서 알 수 있드시 혼합 질의 처리의 기본 과정은 인덱스가 정의되어 있는 경우 먼저 인덱스를 사용하여 객체를 선택한 후 선택된 객체로 부터 객체 시그니쳐를 이용하여 전진 운행 탐색을 하면 된다.

예를 들어 Student.dept.univ.city 대해서만 인덱스가 구성되어 있다고 하면 이 인덱스를 사용하여 Student 클래스의 객체들을 선택할 수 있으며 이 선택된 객체들에 대해서 시그니쳐 전진 운행 탐색을 이용하여 Department의 객체들을 탐색하면 된다. 반대로 Student.dept.name에 대해 인덱스가 구성되어 있다면 이 인덱스를 이용하여 Student의 객체를 검색한 후 이 객체들에 대해 전진 운행 탐색을 하면 된다.

또 다른 예로, 같은 클래스의 객체에 대해 다중 조건이 주어질 수 있다. 질의문 Q3는 Address 클래스에

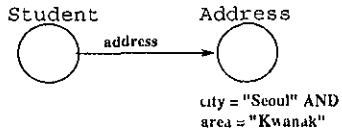



그림 7 Q3에 대한 질의 그래프

두 개의 조건이 주어진 경우를 보여 준다.

Q3 서울시 관악구에 사는 학생의 정보를 검색하라.

질의문 Q3는 중첩애트리뷰트 Student.address.city와 Student.address.area에 대한 조건으로 구성된다. 이때 두 애트리뷰트 중 하나인 Student.address.city에 대해서만 인덱스가 구성되어 있는 경우에도 먼저 이 인덱스를 이용하여 Student 객체들을 검색한 다음 검색 객체에 대해서 Address 클래스의 시그니처를 이용하여 다른 애트리뷰트에 대해 전진 운행 탐색을 사용할 수 있다.

### 3.6 집합값을 갖는 애트리뷰트

위에서 사용한 예제 클래스들중에서 Student 클래스의 friends 애트리뷰트와 같이 객체들의 집합을 애트리뷰트의 값으로 가지는 경우가 있다. 이러한 집합값을 가지는 애트리뷰트에 대해서는 집합에 속하는 각 OID들마다 별도의 시그니처들을 저장하는 방법을 사용한다.

friends={S#1, S#2, S#5, S#10}와 같이 OID로 구성된 집합값 애트리뷰트의 값은 friends={(S#1, S(S#1)), (S#2, S(S#2)), (S#5, S(S#5)), (S#10, S(S#10))}와 같이 객체 시그니처와 함께 저장된다.

그러므로 "친구의 이름이 "G D.Hong" 인 학생을 검색하라"와 같은 질의문을 처리하기 위해서는 각각의 Student 객체에 대해 friends 애트리뷰트에 저장된 객체들의 시그니처를 검색하여 검사하는 전진 운행 알고리즘과 같은 과정으로 처리하면 된다.

## 4. 성능 평가

성능 평가를 위해 다음과 같은 n개의 클래스에 대한 질의문 Q4를 가정한다. Q4는 클래스 $C_i$에 정의된 애트리뷰트 $A_i(1\leq i\leq n-1)$가 클래스 $C_{i+1}$의 객체를 참조하도록 정의되고, n개의 조건식 $Pred_i$는 클래스 $C_i$의 애트리뷰트 $A'_i(A'_i\neq A_i)$에 대한 조건으로 구성되는 데 이 질의에 대한 질의그래프는 그림 8에 나타나 있다.

성능 평가를 위해 사용되는 매개변수들은 다음과 같다.

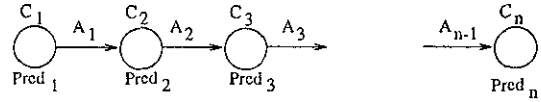



그림 8 Q4의 질의 그래프

- $N_i$ : 클래스 $C_i$의 객체 수
- $M_i$ : 클래스 $C_i$에서 $Pred_i$를 만족하는 객체 수
- $P_i$ : 클래스 $C_i$에서 $Pred_i$를 만족할 확률, $=M_i/N_i$
- $V_i$ : 질의 처리를 위해 검색해야 하는 클래스 $C_i$의 객체 수
- $F_d$ : 시그니처에 대한 매치시 오류가 발생할 확률
  $=\frac{\text{시그니처 매치 오류에 의한 객체 수}}{\text{실제 조건을 만족하지 않는 객체 수}}$

이 논문에서는 질의를 처리하기 위해 검색해야 하는 객체의 수를 기준으로 평가하였으며 전진 운행 기법을 이용한 중첩 루프 방식과 시그니처 중복 방식을 비교하였다.

### 4.1 중첩 루프(nested loop) 방식

클래스 $C_i$에 속한 검색해야 하는 객체의 수, $V_i(2\leq i\leq n)$는 다음식으로 계산된다.

$V_i=$(i−1 단계에서 검색해야할 객체수) . (i−1 단계의 조건을 만족하는 확률)
$=V_{i-1}\cdot P_{i-1}$

이때 $V_1$은 모든 객체를 검색해야 하므로 $N_1$이 된다 그러므로 검색해야 할 전체 객체수 V는 다음과 같다.

$$\begin{aligned}V&=V_1+V_2+V_3+\quad+V_n\\&=V_1+V_1\cdot P_1+V_1\cdot P_1\cdot P_2+\quad+V_1\cdot P_1\cdot P_2\cdots P_{n-1}\\&=V_1(1+P_1+P_1\cdot P_2+P_1\cdot P_2\cdot P_3+\quad+P_1\cdot P_2\cdot P_{n-1})\\&=V_1(1+\sum_{i=1}^{n-1}\prod_{j=1}^{i}P_j\end{aligned}$$

### 4.2 시그니처 중복 방식

클래스 $C_i$에 속한 검색해야 하는 객체의 수, $V_i(3\leq i\leq n)$는 아래와 같은 식으로 계산된다.

$V_i=$(클래스 $C_{i-1}$의 검색해야 할 객체 수
- 클래스 $C_{i-1}$의 객체가 $Pred_{i-1}$ 조건을 만족할 확률−클래스 $C_{i-1}$ 객체에서 매치 오류로 검색하는 객체수)
- (클래스 $C_i$의 객체가 시그니처를 통해 $Pred_i$ 조건을 만족하는 객체수+매치 오류로 인한 객체수)

$=V_{i-1}\cdot(P_{i-1}-(1-P_{i-1})F_d)\cdot(P_i+(1-P_i)F_d)$

이때 $V_1$은 $N_1$이고 $V_2$는 $V_1$의 객체에서 매치 오류

에 의한 것이 없으므로 $V_1P_1(P_2+(1-P_2)F_d)$이다.

그림 9에는 두 가지 방식에 대해 예제 데이타를 주고 계산하여 비교한 결과를 보여준다. 예제 클래스가 $C_i(1 \le i \le 3)$로 질의 그래프 Q4의 형태에 따라 주어지고 각 클래스에 대해 조건이 주어지는 데 이를 만족하는 확률이 모두 10%라고 하자. 이 때 각 클래스의 객체수와 시그니쳐 매치 오류 확률이 달라짐에 따라 클래스 $C_i(2 \le i \le 3)$에서 검색해야 하는 객체수를 비교하였다. 첫번째 클래스($C_1$)의 객체수(V1)를 제외한 이유는 클래스 C1의 객체는 두 방식 모두 전체 객체(N1)를 검색하기 때문이다.

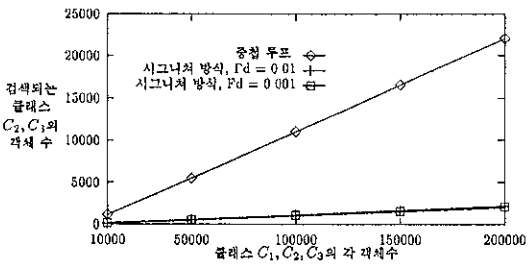



그림 9 전진 운행 탐색의 성능 비교, $P_i=0.1(1 \le i \le 3)$인 경우

시험 결과 두 번째 클래스의 객체에서부터 시그니쳐를 사용하느냐에 따라 검색 객체수가 많은 차이를 보여준다. 그리고 $F_d$의 차이에 따른 객체수는 근소하게 나타나고 있다.

그러나 만약 첫번째 클래스에 대해 인덱스가 정의되어 있어서 전체 $N_1$를 검색할 필요가 없는 경우 시그니쳐 기법에서도 인덱스를 통해 선택된 객체에 대해서만 전진 운행 탐색을 수행하면 된다.

또한 클래스 $C_i$에 속한 $V_i$개의 객체를 검색하는 과정에서도 시그니쳐 방식에서는 OID 테이블의 객체 시그니쳐를 사용하여 조건 만족여부를 검사할 수 있으므로 조건을 만족하지 않는 객체에 대해서는 OID 테이블 검색 과정으로 종료되므로 실제 입/출력은 감소하게 된다.

## 5. 맺음말

이 논문에서는 객체 지향 데이타베이스에서 질의 처리를 위한 전진 운행 탐색과정을 신속히 수행하기 위한 방법으로 객체 시그니쳐를 만들어 참조 객체에 저장하므로써 질의 처리시 먼저 시그니쳐를 검사하여 주어진 조건을 만족하는 경우에만 해당 객체를 검색하도록 하는 시그니쳐 중복 기법을 제시하였으며, 그에 따른 성능 평가 모델과 평가 결과를 제시했다.

이 논문에서 제시한 기법은 기본적으로 시스템이 역포인터를 제공하는 경우 간단하게 구현이 가능하며 제공되지 않는 경우에는 역포인터 정보를 별도로 관리해야만 한다. 그러나 참조 객체에 피참조 객체 시그니쳐를 저장하지 않고 단지 객체 시그니쳐를 OID 테이블에 추가하여 객체를 검색하기 전에 질의에 주어진 중첩 조건들을 객체 시그니쳐와 비교하는 방법만 사용해도 조건을 만족하지 않는 모든 객체에 대해서는 실제 데이타 검색 과정을 생략시킴으로써 검색 비용이 감소된다.

향후 연구 방향으로 1단계 참조 관계에 대한 객체 시그니쳐 중복 기법만을 제시하였으나, 2단계 이상의 참조 관계에 대해서도 적용하는 연구가 필요하다. 그리고 현재는 단일값을 가지는 애트리뷰트의 값들에 대해서만 시그니쳐를 생성하도록 하였으나 집합값과 다매체(multi-media) 데이타를 가지는 객체에 대한 시그니쳐 생성 방안에 대해서도 연구되어야 할 것이다.

## 참 고 문 헌


[1] 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.

[2] K-C Kim, W Kim, D. Woelk, and A. Dale, "Acyclic Query Processing in Object-Oriented Databases," in *Proc. Entity-Relationship Conference*, Nov. 1988.

[3] E Shekita and M. Carey, "Performance Enhancement Through Replication in an Object-Oriented DBMS," in *Proc ACM SIGMOD*, June 1989.

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

[5] S Christodoulakis and C Faloutsos, "Design Considerations for a Message File Server," *IEEE Software Engineering*, vol 10, pp. 201-210, March 1984.

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

[7] 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.

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

[9] 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.

[10] R. Cattel, *Object Data Management : Object-oriented and Relational Database Systems*, pp. 151－152, Addison-Wesley Publishing Company, 1991.

[11] 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.

용 환 승

1983년 2월 서울대학교 컴퓨터공학과 졸업. 1985년 2월 서울대학교 대학원 컴퓨터공학과 공학석사. 1985년 1월～1989년 2월 한국전자통신연구소 연구원. 1989년 3월～1994년 2월 서울대학교 대학원 컴퓨터공학과 공학박사. 1991년 4월～1993년 6월 서울대학교 중앙교육연구전산원 연구조교. 관심 분야는 객체 지향/중첩 릴레이션 데이타베이스, 4세대 언어, 전문가 시스템임.

김 형 주

제 21권 2호 참조

이 석 호

제 21권 1호 참조