객체지향 데이타베이스 시스템에서 객체 관계성 테이블을 이용한 색인기법

부 기 동† · 이 상 조††

요 약

본 연구에서는 객체 관계성 테이블을 이용하여 객체지향 데이타베이스의 집단화 및 상속 계층을 접근하는데 효율적인 색인 기법을 제안하였다. 이 색인 기법은 클래스 간의 집단화와 상속 관계성에 관한 정보와 인스턴스 간의 참조 정보를 메타 데이타로서 사용하였으며, 그 결과 테이블 간의 탐색항해를 통해 다양한 질의들을 효과적으로 평가할 수 있도록 하였다. 본 연구에서는 제안된 기법을 기존의 색인 기법들과 비교하고 그 효용성에 관해 논의하였으며, 또한 저장비용과 검색비용 측면에서 그 성능을 시뮬레이션한 결과를 제시하였다.

An Indexing Technique using Object Relationship Table in Object-Oriented Database Systems

Kidong Bu† · Sangjo Lee††

ABSTRACT

Using object relationship table, this study proposes an efficient indexing technique for accessing both aggregation and inheritance hierarchies in object-oriented databases. This indexing technique uses the information on the aggregation and inheritance relationship between classes and the information on referencing instances as meta data. As a result, the technique makes it possible to effectively evaluates various queries navigating between tables. The study compares the proposed technique with the existing indexing techniques and discusses its usefulness. It also provides the result of simulating its performance in terms of storage and retrieval cost.

1. 서 론

객체지향 데이타베이스는 실세계의 개념을 객체로서 표현하고 이들 간의 복잡한 의미적 관계성을 보다 원활하게 모델링 할 수 있는 차세대의 데이타베이스 시스템으로서, 최근 인간의 지식을 중심으로 하는 지식베이스 시스템이나 CAD/CAM, 멀티미디어 데이타베이스 시스템, 소프트웨어 공학 등의 분야에서 그 응용이 확산되고 있다[1].

객체지향 데이타베이스에서는 실세계의 모든 개념적 개체들이 한 개의 객체가 될 수 있으며 각 객체마다 객체 식별자(OID:Object Identifier), 속성과 메소드 등을 가지며, 속성과 메소드가 같은 객체들끼리 하나의 클래스를 구성한다. 이러한 방법으로 구성된 클래스들은 클래스/서브클래스 관계에 따른 상속 계층(inheritance hierarchy)과 복합객체(complex object)를 표현하기 위한 집단화 계층(aggregation hierarchy)

† 정 회 원:경북산업대학교 전자계산학과 부교수
†† 정 회 원:경북대학교 컴퓨터공학과 교수
논문접수:1995년 11월 17일, 심사완료:1996년 1월 24일

을 구성하게 된다[2].

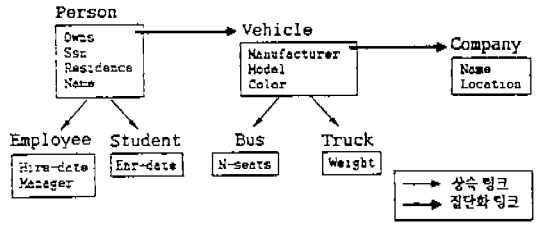
기존의 데이타베이스에서는 데이타에 대한 최적의 접근 경로를 찾아 효율적인 입출력을 수행하기 위한 색인 기법이 중요하게 취급되고 있는데, 객체지향 데이타베이스 시스템에서도 상속 계층이나 복합객체로 구성되는 집단화 계층을 접근하는 질의를 효율적으로 처리하기 위한 색인 기법에 관한 연구[3, 4, 5, 6, 7, 8, 9]가 활발하게 이루어지고 있다.

지금까지 개발된 객체지향 데이타베이스의 색인 기법들은 객체지향 데이타 모델의 집단화 계층이나 상속 계층 중 어느 한가지만을 접근하는 질의에 대해서 고려하는 경우가 대부분이었다. 최근의 연구에서는 이 두 계층을 동시에 접근하는 질의를 효율적으로 평가하기 위한 색인구조를 설계하는데 초점을 두고 있으며, 이 분야의 대표적 색인기법으로는 Bertino와 Foscoli[9]에 의해 제안된 중첩-상속 색인(nested-inherited index)이 있다. 이 연구에서는 B^+-트리를 응용한 기본 색인과 보조 색인을 사용하여 두 계층을 동시에 접근할 수 있도록 하였으나, 다음과 같은 몇가지 문제점을 갖고 있다. 첫째는 갱신연산을 위한 보조 색인의 사용으로 기억장소의 낭비가 심하고, 둘째는 보조 색인으로 인해 갱신 연산 시에 발생하는 색인의 유지보수 절차가 복잡하며, 셋째는 질의가 목표 클래스와 그 서브 클래스의 인스턴스까지를 요구할 경우 색인의 기본 레코드부의 클래스 디렉토리만으로는 해당 서브 클래스들을 식별해 낼 수 없어 스키마에 관한 추가의 의미 정보 없이는 "SELECT-ALL" [10]과 같은 검색 질의를 처리할 수가 없다. 그리고 이 방법은 단일키에 의한 접근 방식이기 때문에 다중키에 의한 제한 질의를 처리하고자 할 때 다중키의 갯수만큼 별도의 색인을 편성하는 수밖에 없어 기억 공간 및 입출력 성능상의 부담이 크다.

본 연구에서는 중첩-상속 색인의 이러한 취약점을 해결할 수 있도록 보다 단순하고 크기가 작은 객체관계성 테이블(Object Relationship Table)을 사용한 새로운 색인 기법을 제시하였다. 이 기법에서는 클래스 간의 집단화 및 상속 관계에 관한 정보와 인스턴스 간의 참조 정보를 테이블 내에 함축시킨 테이블 구조를 사용함으로써, 테이블 간의 탐색항해를 통해 다양한 질의들을 효과적으로 처리할 수 있을 뿐 아니라, 크기가 작아 주기억 장치에서 운영이 가능하다는 장점이 있다. 제안된 방법의 성능은 기존의 색인 기법 중 집단화 계층 및 상속 계층을 동시에 접근하는 질의에 적용이 가능한 다중-상속 색인(multi-inherited index)과 중첩-상속 색인을 비교 대상으로 하여 다양한 실험을 통해 평가하였다.

2. 질의 모델

객체지향 데이타베이스에서 사용되는 질의 형식은 상속 계층이나 집단화 계층, 혹은 이 두가지 계층을 모두 탐색하여 객체에 접근할 수 있다.



(그림 1) 객체지향 데이타베이스의 스키마 예
(Fig. 1) An example of OODB schema

(그림 1)은 본문에서 응용할 객체지향 데이타 모델[2]의 개념에 따르는 스키마 표현의 예이며, (그림 2)는 이 스키마에 대한 인스턴스들이 저장된 형태를 나타내는 예이다. 이 저장 모델[10]은 객체지향 데이타베이스로서 잘 알려진 ORION[11]에서 사용되는 방식이다. 그림에서 보는바와 같이 각 인스턴스는 그가 속하는 특정 클래스 내에 중복없이 한번만 나타나며 단일 인스턴스 검색 시 모든 속성 값이 해당 클래스 내에 존재하기 때문에 클래스 간의 추가적인 탐색이 필요 없다.

객체지향 데이타베이스에서 사용되는 단일키에 의한 질의 형식은 크게 세가지 형식의 범주로 구분할 수 있으며, (그림 1)에서 제시한 데이타베이스 스키마를 대상으로 세가지 형태의 검색 질의 예를 들면 다음과 같다.

(1) 상속 계층만을 접근하는 질의:

"Rome에 거주하는 클래스 Person과 그 서브 클래스의 모든 인스턴스를 검색하라."

(2) 집단화 계층만을 접근하는 질의:

PERSON

OID	Owns	Ssn	Residence	Name
person<o>	vehicle<i>	345-56-678	Rome	John
person<p>	vehicle<j>	389-66-698	Pisa	Mark
person<q>	vehicle<k>	389-66-677	Rome	Alex

Employee

OID	Owns	Ssn	Residence	Name	Hire-date	Manager
employee<i>	vehicle<j>	574-11-365	Florence	Charles	02-04-84	Brown
employee<j>	truck<j>	947-33-469	Pisa	Sarah	06-08-85	Rossi

Student

OID	Owns	Ssn	Residence	Name	Enr-date
student<i>	vehicle<i>	737-21-345	Florence	mana	01-02-94

Vehicle

OID	Manufacturer	Model	Color
vehicle<i>	company<j>	Tipo	White
vehicle<j>	company<j>	Panta	Red
vehicle<k>	company<i>	R5	Red

Bus

OID	Manufacturer	Model	Color	N-seat
bus<j>	company<j>	X80	Red	40

Truck

OID	Manufacturer	Model	Color	Weight
truck<j>	company<j>	Y12	Red	200

Company

OID	Name	Location
company<i>	Renault	Paris
company<j>	Fiat	Turin

(그림 2) 그림 1의 스키마에 대한 인스턴스들의 예
(Fig. 2) An example of instances with respect to schema (Fig. 1)

"Renault 회사에서 제작한 Vehicle의 모든 인스턴스를 검색하라."

(3)집단화 계층과 상속 계층을 모두 접근하는 질의:

"Paris에 위치한 회사에 의해 제작된 Vehicle을 소유한 클래스 Person과 그 서브 클래스의 모든 인스턴스를 검색하라."

3. 색인기법의 관련 연구

지금까지 연구된 집단화 계층에 대한 색인 기법으로서는 Maier와 Stein[3]에 의하여 제안된 다중 색인(multiindex)과 Valduriez[4]에 의해 제안된 죠인 색인(join index), Bertino와 Kim[5]에 의해 제안된 경로 색인(path index), 그리고 Kemper와 Moerkotte[6]에 의해 제안된 접근-릴레이션(access-relation) 등이 있다. 또한 상속 계층에 적용되는 대표적 색인 기법으로는 Kim과 Dale[7]에 의해 제안된 상속 색인(inherited index)과Low, Beng, Lu[8]에 의해 제안된 H-트리 등이 연구된 바 있다. 그러나 이러한 색인 기법은 집단화 계층 혹은 상속 계층 중 한가지 계층만을 접근하는 질의를 위해 사용되는 색인이며, 이 두 계층을 모두 접근하는 색인 기법으로는 Bertino와 Foscoli[9]에 의해 연구된 중첩-상속 색인과 상속 계층마다 상속 색인을 다중으로 유지하는 다중-상속 색인이 있다.

이 장에서는 상속 색인과 다중-상속 색인, 그리고 중첩-상속 색인에 대해서 좀더 자세히 논의하고자 한다. 다중-상속 색인과 중첩-상속 색인은 본 연구와 성능을 비교하기 위해서이고 상속 색인은 이 두 기법의 이론적 기본이 되는 중요한 색인 기법이기 때문이다.

관련 연구의 서술에 앞서, 색인 기법과 관련된 몇 가지 용어를 정의하면 다음과 같다. 클래스 C_i이 주

어졌을 때 $C_{i,*}$는 C_i을 루트로 하는 상속 계층의 모든 클래스를 나타낸다. 집단화 계층에서 경로(path) P는 $P=C_1.A_1.A_2.\ldots.A_n$으로 표기하며, 이 때 A_1은 C_1의 속성들 중 하나이고, $A_i(i\geq 2)$는 클래스 C_{i-1}의 속성인 A_{i-1}의 도메인이 되는 클래스 C_i의 속성이다. 즉, 경로는 집단화 계층의 한 가지(branch)를 표현한 것이다. 경로의 길이를 표현하는 len(p)는 경로 상에 나타나는 속성의 수를 나타내며, 경로의 작용영역(scope)은 경로 상에 속하는 모든 C_i들을 루트로 하는 상속 계층의 모든 $C_{i,*}$를 나타낸다. 또한 경로 상에서 색인의 키 값을 갖는 속성을 색인 속성, 색인 속성이 포함된 클래스를 색인 클래스라 정의한다.

3.1 상속 색인

객체지향 데이타베이스에서 클래스 간의 상속 계층만을 고려할 때 상속 계층에서 루트가 되는 클래스의 특정 속성 값을 키로 하여 상속 계층간에 분산되어 나타나는 관련 인스턴스들을 효율적으로 검색하기 위해서는 상속 계층 전체의 인스턴스 분포를 파악할 수 있는 색인 기법이 필요하다. 상속 색인은 이러한 경우에 적용되는 색인 기법으로 B^+-트리 상에서 단말 노드의 구조를 적절한 형태로 설계하여 구현할 수 있다.

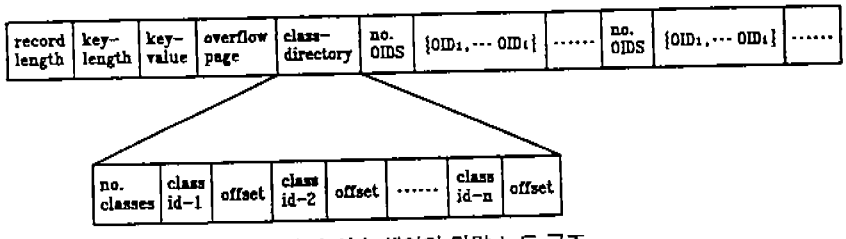
상속 색인의 단말 노드는 (그림 3)에서 보는 것처럼 레코드 길이, 키 길이, 키 값, 오버플로우 페이지 포인터, 클래스 목록, 그리고 상속 계층내의 각 클래스에 대하여 색인된 속성의 키값을 보유하고 있는 객체들의 OID 리스트와 OID 리스트의 갯수 필드로 구성된다. 이 중 클래스 목록은 색인된 키 값과 연관된 인스턴스들이 속하는 클래스들의 식별자, 단말 노드 내에서 해당 객체에 대한 OID 리스트의 위치 정보를 나타내는 변위 및, 해당 상속 계층에 포함되는 클래스 식별자들의 갯수 필드로 구성된다.

이러한 상속 색인을 운영하는 질의 예로서 "Rome에 거주하는 클래스 Person과 Student의 모든 인스턴스를 검색하라."라는 질의가 주어졌을 때, 속성값 "Rome"를 키로 하여 B^+-트리를 순회하여 단말 노드를 채취한 후, 단말 노드의 클래스 목록 부에 나타나는 클래스 식별자 중에서 클래스 Person과 Student를 식별하여 해당 변위에 의해 관련된 OID들을 추출하면 된다.

3.2 다중-상속 색인

다중-상속 색인은 경로에 포함되는 속성에 대해 상속 색인을 다중으로 유지하여 구현할 수 있다. 즉, 길이가 n인 경로 $P=C_1.A_1.\ldots.A_n$이 주어지면 경로 상의 루트 클래스인 C_i들의 상속계층에 대하여 A_i속성 값을 키로 하는 상속 색인을 각각 별도로 편성한다. 이 방법을 집단화 및 상속 계층을 동시에 접근하는 질의에 적용할 경우에는, 색인 속성인 A_n에 대해 설정된 상속 색인을 탐색하여 얻어진 OID를 가지고 이와 관련된 OID를 A_{n-1}의 속성에 설정된 상속 색인에서 찾고, A_{n-1}의 상속 색인에서 얻어진 OID로 다시 A_{n-2}의 상속 색인을 탐색하고, 이러한 탐색을 목표 클래스가 포함된 상속에서 원하는 OID 리스트를 추출할 수 있을 때까지 진행해나가면 된다.

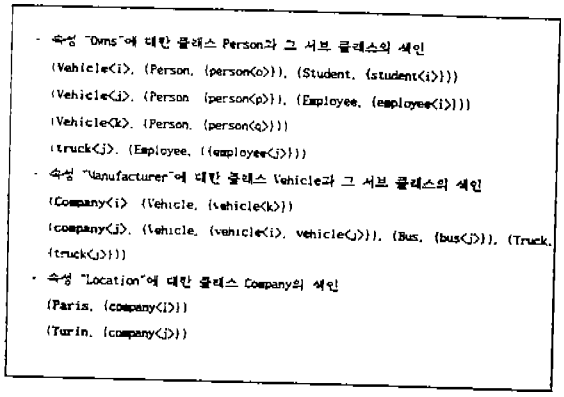
다음의 (그림 4)는 본문의 객체 인스턴스 예에 대하여 세개의 상속 색인으로 다중-상속 색인을 구성했을 때 각각의 색인에 포함되는 내용을 논리적으로 나타낸 것이다. 그림의 예에서 "Turin에 위치한 공장에서 제작된 Vehicle을 소유한 클래스 Person과 그 서브 클래스의 모든 인스턴스를 검색하라"라는 질의가 발생했을 경우에는 먼저 클래스 Company에 대한 색인을 탐색하여 키 값 Turin에 의해 company⟨j⟩를 추출한



(그림 3) 상속 색인의 단말 노드 구조

(Fig. 3) A leaf node of a inherited index

다. 그 다음은 company⟨j⟩를 키 값으로 하여 클래스 Vehicle과 그 서브 클래스의 색인을 탐색하여 키 값과 연관된 OID들, 즉 vehicle⟨i⟩, vehicle⟨j⟩, bus⟨j⟩, truck ⟨j⟩를 추출한다. 그리고 이 OID들을 키 값으로 하여 클래스 Person과 그 서브 클래스의 색인을 탐색하여 최종적인 OID 리스트로서 person⟨o⟩, student⟨i⟩, person⟨p⟩, employee⟨i⟩, employee⟨j⟩를 추출하게 된다. 다중-상속 색인은 상속 색인의 기본 구조를 응용하여 상속 및 집단화 계층에 적용할 수 있도록 하였으나 예에서 보여주는 것처럼 각각의 색인에 대해 검색해야 할 키가 여러개가 발생하게 됨으로 검색 효율이 떨어진다는 단점이 있다.



(그림 4) 다중-상속 색인의 논리적 구조
(Fig. 4) Logical Structure of multi-inherited index

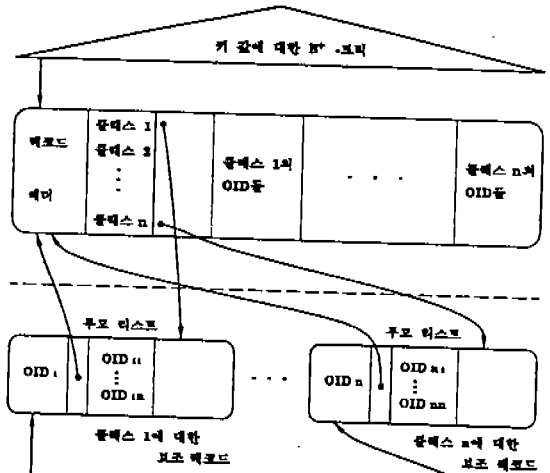
3.3 중첩-상속 색인

중첩-상속 색인은 집단화 계층과 상속 계층을 모두 접근할 수 있으며 (그림 5)와 같이 두개의 B^+-트리 구조를 기반으로 한다. 먼저 색인 속성에 대한 키 값으로 구성된 기본 색인에 대한 B^+-트리를 기본 색인이라 부르고, 키 값과 연관된 인스턴트들의 부모 리스트를 보관하는 또 다른 B^+-트리를 보조 색인이라 부른다. 이 때 기본색인의 단말노드는 기본 레코드, 보조 색인의 단말 노드는 보조 레코드라 부르며, 이들은 포인터로 서로 연결되어있다.

예를 들어 "Turin에 위치한 공장에서 제작된 Vehicle을 소유한 Person을 탐색하라"라는 질의가 발생했을 경우 색인 키로서 "Turin"을 갖는 기본 레코드를 접근하여 클래스 목록부에서 클래스 Person을 식별한 후 변위에 의해 해당 OID 리스트를 탐색함으로써 원하는 결과를 빠른 속도로 검색할 수 있다.

그리고 중첩-상속 색인에서는 객체에 대한 갱신 연산 시에 객체 탐색없이 변화된 내용을 색인의 기본 레코드부에 반영할 수 있도록 다수의 보조 레코드들과 보조 레코드에 대한 비단말 노드들로 구성된 보조 색인을 사용하는데[9], 이러한 보조 색인만으로도 키 값과 연관되어 함께 갱신해야 할 OID 리스트들을 식별해 낼 수 있다. 중첩-상속 색인은 전술한 바와 같이 집단화 및 상속계층을 동시에 접근하는 질의를 효율적으로 평가하고 보조색인을 사용하여 추가의 객체 탐색 없이 갱신 결과를 색인에 반영할 수 있으나, 다음과 같은 몇가지 관점에서 그 기능이 취약하다.

첫째, 중첩-상속 색인은 기본 레코드의 구조 중 클래스 디렉토리부의 구조로서 상속 색인이 가지고 있는 디렉토리 구조를 동일하게 적용하였기 때문에 상속 계층에 대한 스키마 의미정보를 시스템 카탈로그 등에서 추가로 채취하여야만 한다. 예를 들어 "Turin에 위치한 회사에 의해 제작되는 Vehicle을 소유한 Person 클래스와 그 서브 클래스의 모든 인스턴스를 검색하라"라는 질의에서 기본 레코드의 디렉토리부에서는 집단화 계층의 경로에 포함되는 여러 상속 계



(그림 5) 중첩-상속 색인의 구조
(Fig. 5) Structure of a nested-inherited index

층의 클래스들이 나타나기 때문에 디렉토리만으로는 특정 상속 계층인 클래스 Person의 서브 클래스들, 즉 Student 클래스와 Employee 클래스를 식별할 수 있는 방법이 없다. 따라서 SQL 형태로 "SELECT-ALL"과 같은 목표 클래스와 그 모든 서브 클래스들에 대한 검색 질의는 스키마에 관한 의미 정보를 추가로 채취하지 않고서는 처리가 불가능하다.

둘째, 객체들의 삽입, 삭제 연산 시에 인스턴스의 직접적인 탐색을 회피하기 위해 고안된 보조 색인부가 객체들의 물리적 저장장소를 탐색하지 않고서도 색인부 만으로 키 값과 연관된 각 OID들의 부모 리스트를 추적할 수 있기 때문에 효율적이나, 기본 레코드부에 한번 나타났던 OID들이 보조 색인부에 또 한번 중복 저장되어야 하기 때문에 기억 장소를 많이 소모하게된다.

셋째, 갱신연산이 발생할 때마다 변화된 내용을 기본 색인뿐만 아니라 보조 색인에 반영하기 위해서 [9]에서 제시한 색인 갱신 절차를 밟아야 하는데, 이러한 절차가 지나치게 복잡하여 색인의 유지보수 측면에서는 비효율적이다.

넷째, 중첩-상속 색인은 다중키에 의한 접근이 취약하다. 즉 다중키에 대한 제한 질의를 처리하고자 할 때 다중키의 갯수만큼 별도의 색인을 편성하는 수밖에 없어 기억 공간 및 입출력 성능상의 부담이 크다. 위와 같은 취약점들은 본 연구에서 제안하는 객체 관계성 테이블을 이용한 색인기법으로서 해결할 수 있다.

4. 객체 관계성 테이블을 이용한 색인기법

4.1 객체 관계성 테이블의 구조

본 논문에서 제안한 객체 관계성 테이블은 키 값과 연관된 OID를 효율적으로 식별해낼 수 있도록 클래스 간의 상속관계 및 집단화 관계뿐만 아니라 객체 간의 참조관계도 함축시킨 일종의 메타 데이타로서의 성격을 가진다. 객체 관계성 테이블은 기존의 색인구조 보다 소량의 정보만을 포함하기 때문에 그 크기가 아주 작아 주기억 장치 및 가상 기억장치에서 효율적인 운영이 가능하다. 객체 관계성 테이블은 데이타베이스 스키마에 대해 클래스마다 한개씩 설정되며 유일한 번호, 즉 테이블 식별자(TID : Table Identifier)를 부여받게 되는데, 이 TID는 클래스마다 깊이 우선 탐색 순서로 부여된 유일한 일련번호를 객체 관계성 테이블에 대해서도 동일하게 부여한 것이다. 이 TID는 나중에 테이블을 찾기 위한 색인으로 사용된다.

객체 관계성 테이블의 구성은 (그림 6)에서 보는 것처럼 머리부와 몸체부로 구분할 수 있고, 머리부에는 클래스간의 관계성을 나타내는 정보를 포함하며 다음과 같은 필드로 구성된다.

- 테이블 식별자 필드
- 클래스 식별자 필드
- 집단화-부모 클래스 필드
- 상속 자식 클래스 필드

먼저 TID 필드에는 전술한 테이블 번호를 기록하고 클래스 식별자 필드에는 해당 테이블에 대응되는 클래스의 명을 기록한다. 집단화-부모 클래스 필드에는 집단화 계층에서 자신의 부모 클래스에 대응되는 TID를 기록한다. 만약 집단화 계층에서 부모 클래스가 존재하지 않을 경우에는 0을 기록한다.

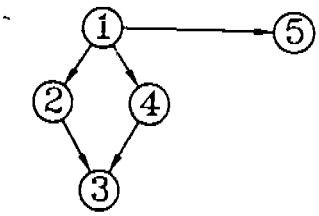
그리고 상속 자식 클래스 필드에는 상속 계층에서 자신의 서브 클래스에 대응되는 TID 들의 집합(혹은 리스트)을 기록하게 된다. 이때도 만약 자신의 서브 클래스가 존재하지 않으면 0을 기록한다. 예를들어 (그림 7)과 같은 스키마 그래프의 예에서 1번 클래스에 대응되는 1번 객체 관계성 테이블의 집단화-부모 클래스 필드에는 0, 상속 자식 클래스 필드에는 {2, 4}가 기록되고, 2번과 4번 테이블의 경우는 집단화-부모 필드에는 0, 상속 자식 필드에는 둘다 3을 기록하게 된다. 5번 테이블의 경우에는 집단화 계층에서 부모 클래스가 존재함으로 집단화 상속 필드에는 부모 클래스에 대응되는 TID인 1을 기록하고 상속 자식 필드는 존재하지 않음으로 0을 기록한다.

머리부	테이블 식별자	클래스 식별자	집단화-부모 클래스	상속-자식 클래스
	TID$_i$	C_i	TID$_j$	{TID$_1$,…,TID$_n$}
몸체부	레코드 식별자	객체 식별자	참조	
	RID$_1$	OID$_1$	{(TID$_i$,RID$_1$),…,(TID$_j$,RID$_1$)}	
	RID$_2$	OID$_2$	{(TID$_i$,RID$_1$),…,(TID$_j$,RID$_2$)}	
	RID$_3$	OID$_3$	{(TID$_i$,RID$_1$),…,(TID$_j$,RID$_3$)}	
	⋮	⋮	⋮	
	RID$_n$	OID$_n$	{(TID$_i$,RID$_1$),…,(TID$_j$,RID$_n$)}	

(그림 6) 객체 관계성 테이블의 구조
(Fig. 6) Structure of object relationship table

다음으로 객체 관계성 테이블의 몸체부는 인스턴스간의 관계성을 나타내는 정보를 포함하며 각 엔트리에 대해 다음과 같은 필드로 구성된다.

- 레코드 식별자(RID : Record Identifier) 필드
- 객체 식별자(OID) 필드
- 참조 필드(referencing field)



(그림 7) 스키마 그래프의 예
(Fig. 7) An example of schema graph

먼저, RID 필드는 해당 테이블에서 필요한 엔트리를 식별하기 위한 주소로 사용되어지는데, RID의 부여 순서는 테이블 내에서 OID의 위치를 식별하기 위한 순서일 뿐이며 인스턴스의 물리적 순서와는 아무런 상관이 없다.

그리고 OID 필드에는 해당 클래스에 속하는 인스턴스들의 OID가 기록되며, 참조 필드에는 각 인스턴스가 집단화 관계에서 참조하는 다른 클래스와 인스턴스에 관한 정보가 (TID, RID) 쌍으로서 기록된다. 여기에서 TID는 이 인스턴스가 집단화 관계에서 참조하는 클래스에 대응되는 테이블 번호를 나타내고 RID는 참조하는 테이블 내에서의 인스턴스의 위치를 나타내는 주소를 의미한다. 만약 한 인스턴스가 속성 값으로 집단화 계층에서 다른 인스턴스에 대한 여러개의 참조를 가질 때는 매 참조에 대해 (TID, RID) 쌍으로 표현되기 때문에 참조 필드는 (TID, RID) 쌍을 요소로 하는 집합이 된다. 예를 들어 테이블에서 임의의 엔트리의 참조 필드 값이 {(4, 2), (5, 1)}이라면 그 엔트리에 대응되는 인스턴스가 TID 4번에 대응되는 클래스의 2번 엔트리에 해당하는 인스턴스와 5번에 대응되는 클래스의 1번 엔트리에 해당하는 인스턴스에 대하여 집단화 계층 상에서 두개의 참조를 갖고있다는 것을 의미한다. 집단화 계층에서 다른 인스턴스에 대한 참조가 없을 때에는 (TID, RID)의 값은 역시 0이 된다.

이러한 객체 관계성 테이블 간의 탐색항해는 테이블 내에 기록된 TID와 RID를 주소로 하여 테이블에 대한 임의접근 형태로 이루어지며, 키 값에 의해 최초의 테이블에 대한 신속한 진입을 위하여 B^+-트리를 경유한다. 이때 B^+-트리의 단말노드에는 객체 관계성 테이블로의 최초 진입을 위한 테이블 번호와 엔트리 번호의 쌍인 (TID, RID)를 관리한다. 이 TID와 RID는 색인 클래스와 색인 클래스에서 해당 키 값을 갖는 인스턴스와 대응되는 객체 관계성 테이블과 엔트리에 대한 주소를 나타낸다. 또한 해당 키 값을 갖는 인스턴스들이 여러개 존재할 수도 있기 때문에 B^+-트리의 단말노드도 (TID, RID) 쌍들을 요소로 하는 집합을 수용한다.

그리고 색인으로서 객체 관계성 테이블을 운영하기 위해서는 질의에 내포된 목표 클래스의 TID를 사전에 파악할 수 있어야 한다. 왜냐하면 객체 관계성 테이블을 색인으로 하여 필요한 최종 OID들을 추출할 때까지의 과정은 B^+-트리에 의한 최초 테이블의 임의접근으로부터 목표 TID에 해당하는 테이블까지 도달하기 위한 탐색항해 과정으로 요약할 수 있기 때문이다. 또한 객체의 삽입, 삭제 연산 시에 변화된 내용을 색인에 반영하기 위해서는 삽입, 삭제가 일어난 목표 테이블뿐만 아니라 집단화 관계로서 연관된 다른 테이블의 OID들에게도 파급효과가 미치게 되는데, 이러한 갱신 작업의 질의에 포함된 목표 클래스의 TID도 사전에 파악할 수 있어야 한다. 따라서 클래스 명으로부터 TID를 추출할 수 있도록 (그림 8)과 같은 클래스-TID 사상 테이블이 필요하다. 이 그림은 본문의 스키마 예에 대하여 구성한 클래스-TID 사상 테이블의 구조이며, 이에 대한 자세한 운영의 예는 다음절에서 설명하기로 한다.

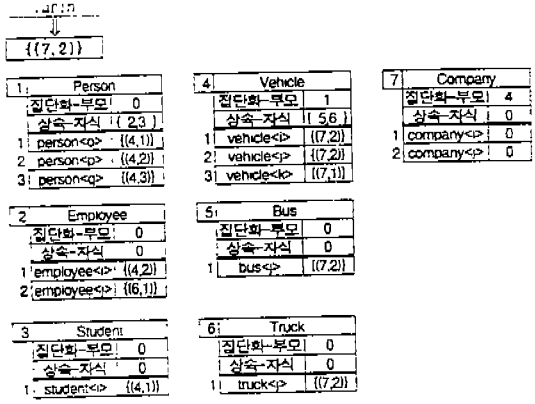
클래스 식별자	Bus	Company	Employee	Person	Student	Truck	Vehicle
TID	5	7	2	1	3	6	4

(그림 8) 클래스-TID 사상 테이블의 예
(Fig. 8) An example of class-TID mapping table

4.2 색인의 검색 운영

검색 질의에 대한 색인의 운영 예를 들기 위하여 먼저 (그림 2)의 인스턴스 예에 대한 색인을 본 논문

에서 제안한 객체 관계성 테이블로 구성해 보면 (그림 9)와 같다.



(그림 9) 객체 관계성 테이블의 예
(Fig. 9) An example of object relationship table

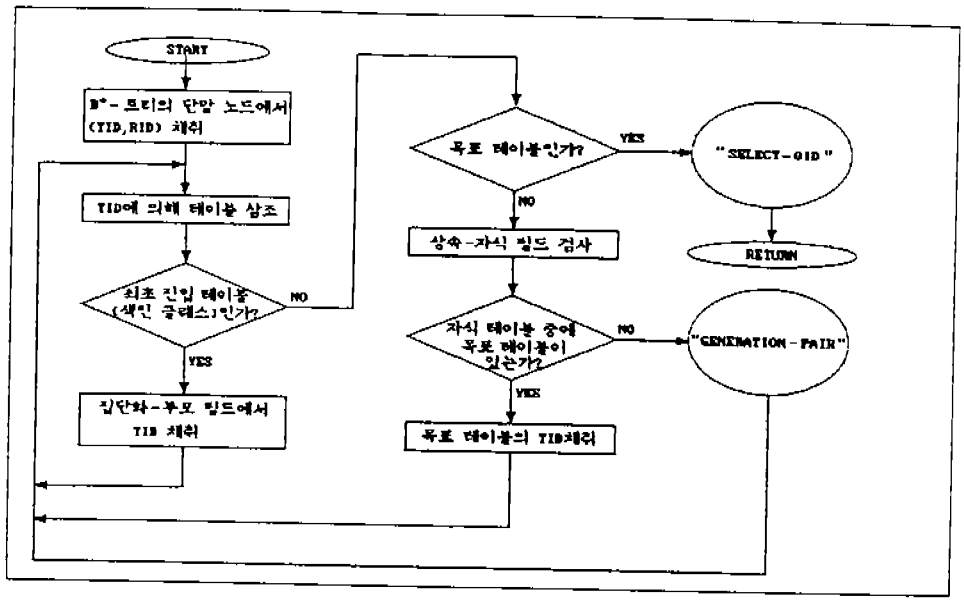
그리고 이러한 객체 관계성 테이블을 운영하기 위한 검색 절차를 흐름도로 나타내면 (그림 10)과 같다. 검색 절차는 "MAIN", "SELECT-OID", 그리고 "GENERATION-PAIR"로 구성되며, "MAIN"에서 필요할 경우에 "SELECT-OID"와 "GENERATION-PAIR"를 호출하는 형식으로 되어있다. 이때 "MAIN"에서는 테이블 간의 탐색항해를 통하여 목표 테이블을 찾기까지의 과정을 처리하고, "SELECT-OID"는 질의의 유형이 "SELECT-ALL"인지 "SELECT-ONLY"인지를 구분하여 결과 OID 리스트를 출력할 수 있도록 한다. 그리고 "GENERATION-PAIR"에서는 테이블의 탐색 과정에서 키 값과 연관된 새로운 (TID, RID)쌍을 생성해내는 과정을 나타낸다.

단일 키에 의한 색인의 검색 예로서 "Turin에 위치한 회사에서 제작한 Vehicle을 소유한 Person과 그 서브 클래스의 모든 인스턴스를 검색하라"라는 질의가 발생했을 때, 위 흐름도에 의하면 다음과 같은 절차로 처리되게 된다.

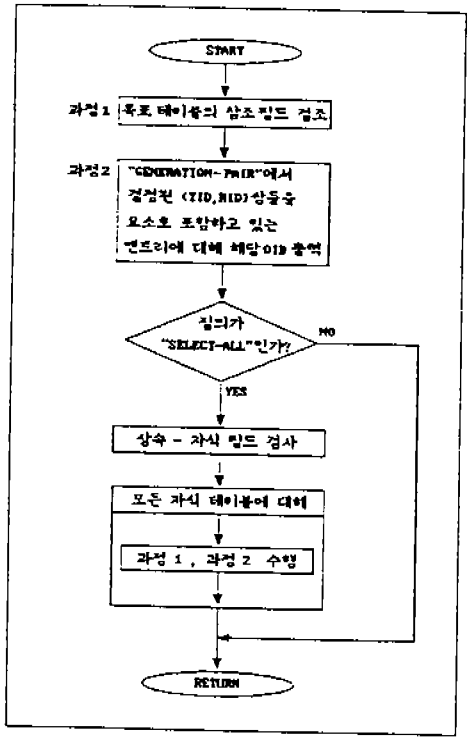
먼저 클래스-TID 사상 테이블을 참조하여 질의의 목표 클래스인 Person의 TID 1을 채취한 후 Company.Location 속성에 대한 B^+-트리의 단말노드에서 (7, 2)를 채취하여 TID가 7인 테이블을 참조한다. 해당 테이블이 최초로 진입한 테이블이기 때문에 집단화-부모 필드에서 TID 값인 4를 채취하여 TID가 4인 테이블을 참조한다. 이때 4번 테이블이 아직 목표 테이블이 아니기 때문에 "GENERATION-PAIR"를 호출하여 4번 테이블과 4번 테이블의 자식 테이블을 대상으로 하여 참조 필드에서 (7, 2)를 요소로 포함하고 있는 엔트리를 찾아 해당 RID와 테이블의 TID를 결합하여 새로운 (TID, RID)쌍인 (4, 1), (4, 2), (5, 1), (6, 1)을 생성한다. 다시 4번 테이블의 집단화-부모 필드에서 TID 값인 1을 채취하여 TID가 1인 테이블을 탐색한다. 해당 테이블의 TID가 질의에 포함된 목표 클래스의 TID와 일치하기 때문에 "SELECT-OID"를 호출한다. 이 질의가 "SELECT-ALL" 형태의 질의이기 때문에 목표 테이블과 목표 테이블의 모든 자식 테이블을 대상으로 하여 참조 필드에서 (TID, RID)쌍인 (4, 1), (4, 2), (5, 1), (6, 1)과 부합되는 엔트리들을 찾아 해당 OID 리스트인 person⟨o⟩, person⟨p⟩, employee⟨i⟩, employee⟨j⟩, student⟨i⟩를 추출한다.

위 절차에서 보는 것처럼 객체 관계성 테이블의 탐색항해와 테이블 내에서의 검조 기능을 통하여 집단화 및 상속 계층을 동시에 접근하는 모든 종류의 질의를 신속히 처리할 수 있다. 중첩-상속 색인의 경우에는 클래스 디렉토리에 포함되는 정보만으로는 Person 클래스의 서브 클래스인 Employee 클래스와 Student 클래스를 식별할 수 없어 스키마에 관련된 추가의 의미 정보 없이는 위 질의, 즉 SQL 형태로 "SELECT-ALL"에 해당하는 질의를 처리할 수 없게 된다.

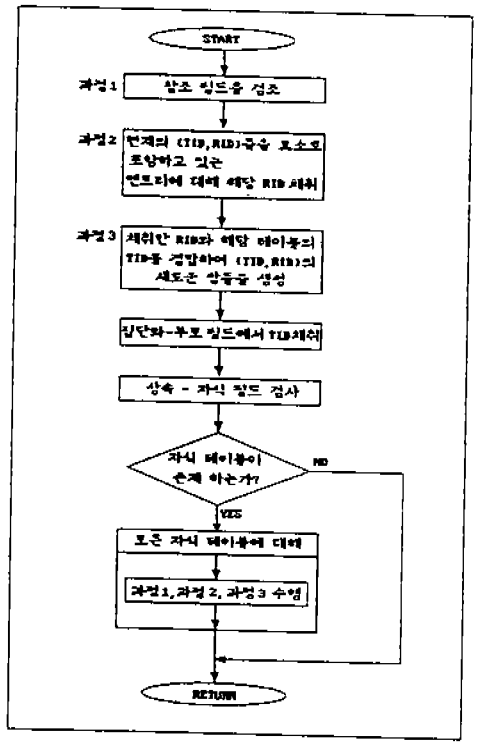
또 다른 질의 예로서 "Turin에 위치한 회사에서 제작된 Bus를 검색하라"라는 질의를 고려해보자. 이 질의는 목표 클래스가 특정 클래스 한개로 한정되어있는 "SELECT-ONLY"형태의 질의이며 목표 클래스가 집단화 계층의 경로 상에서 상속 계층의 루트가 아니라는 점에서 앞서의 예와는 다소 차이가 있다. 이 경우 테이블의 집단화-부모 필드에 나타나는 값으로 집단화 계층의 경로를 탐색항해하는 과정에서 목표 테이블이 소속된 상속 계층까지만 진행해야 하므로 집단화 계층의 경로에 속하는 테이블을 방문할 때마다 상속-자식 필드의 값에 목표 테이블의 TID가 포함되어 있는지를 검사해야한다. 만약 목표 TID가 포함되어 있으면 더이상 집단화 계층의 부모에 대응되는 테이블을 방문할 필요가 없이 목표 테이블을 접근하여 키 값과 연관된 해당 OID를 추출한다. 세부적인



(a) the procedure "MAIN"



(b) the procedure "SELECT-OID"



(c) the procedure "GENERATION-PAIR"

(그림 10) 객체 관계성 테이블에서 검색 절차의 흐름도

(Fig. 10) A flowchart of retrieval procedure in object relationship table

절차는 다음과 같은 과정으로 처리된다. 먼저, 클래스-TID 사상 테이블을 참조하여 질의의 목표 클래스인 BUS의 TID 5를 채취한 후, 키 값 "Turin"에 의해 B^+-트리의 단말노드에서 (TID, RID) 쌍인 (7, 2)를 채취한다. (7, 2) 쌍으로부터 TID가 7인 테이블을 참조한다. 이때 최초로 진입한 테이블임으로 집단화-부모 필드를 통해 4번 테이블을 참조한다. 4번 테이블이 목표 테이블이 아님으로 상속-자식 필드에 목표 테이블의 TID가 포함되는지를 검사한다. 목표 테이블인 5번 테이블이 존재함으로 5번 테이블을 참조한 후, 각 엔트리의 참조 필드를 검조하여 (7, 2)에 부합되는 엔트리에서 해당 OID인 bus⟨j⟩를 채취하면 된다.

여기에서 만약 경로의 끝에 위치한 색인 클래스가 여러개라면 B^+-트리의 단말노드에 TID 값과 RID 값이 서로다른 여러개의 (TID, RID) 쌍이 존재하게 되지만, 이러한 경우라도 처리 절차에는 큰 변화가 없다. 단지 테이블 참조 시에 최초 진입인 경우(B^+-트리의 단말노드에 참조하고자하는 TID가 포함된 경우)를 구분하여 최초 진입일 경우에는 여러개의 TID 중 TID 값이 가장 작은 테이블(해당 상속 계층의 루트 테이블)만 참조하도록 하면 된다. 왜냐하면 경로상에서 상속 계층의 루트가 되는 테이블의 머리부에 있는 집단화-부모 필드를 통해 다른 테이블에 대한 참조를 시행하며, 각 엔트리의 참조 필드를 검조함으로써 현재의 (TID, RID)들로부터 목표 테이블에 도달할 때까지 새로운 (TID, RID)들을 생성해나가기 때문이다.

이번에는 다중키 접근의 예로서 "Turin에 위치한 회사에서 제작되었고 색깔이 red인 Vehicle을 소유한 클래스 Person과 그 서브 클래스의 모든 인스턴스를 검색하라."를 고려해 보기로 한다. 이 경우에 Person.Owns.Manufaturer.Location과 Person.Owns.Color라는 다중 경로 상에서 Company.Location과 Vehicle.Color를 색인 속성으로 하는 다중키 색인을 구현할 수 있다면, 단일 키 색인과 객체탐색만으로 질의를 처리하는 것 보다 성능상의 이득을 얻을 수도 있을 것이다. 본 논문에서 제안한 객체 관계성 테이블 구조에서는 두개의 색인속성에 대하여 각각 한개씩 B^+-트리를 별도로 편성하고 테이블 구조는 공유함으로써 이러한 다중키 색인을 손쉽게 구현할 수 있다. 다시말하면, 다중 경로를 색인하기 위해 필요한 객체들 간의 관련성이 이미 테이블 구조에 내포되어 있다는 것을 의미한다.

위 다중키 질의를 처리하는 과정은 Company.Location 속성에 대한 B^+-트리의 단말노드에서 (7, 2)를 채취하여 목표 테이블까지 테이블의 탐색항해를 통해 추출된 OID(person⟨o⟩, person⟨p⟩, employee⟨i⟩, student⟨i⟩)와 Vehicle.Color 속성에 대한 B^+-트리의 단말노드에서 (4, 2), (4, 3), (5, 1), (6, 1)을 채취하여 목표 테이블까지 탐색항해한 결과인 OID(person⟨p⟩, person⟨q⟩, employee⟨i⟩, emoloyee⟨j⟩)들 간의 공통집합(person⟨p⟩, employee⟨i⟩)을 구하는 과정으로 요약할 수 있다. 다중키에 의한 OR 제한질의도 이와 마찬가지로 각 키에 의해 이루어진 테이블의 탐색항해 결과로 추출된 OID 리스트들에 대한 합집합을 구하는 과정으로 요약할 수 있다.

객체 관계성 테이블은 다중키 접근 시에 테이블 구조에 대한 공유가 가능함으로 다중키 접근을 위한 부가의 자료구조가 필요 없지만, 중첩-상속 색인은 다중키에 의한 제한 질의를 처리하고자 할 때 다중키의 갯수만큼 별도의 색인을 편성하는 수밖에 없어 기억공간 및 입출력 성능상의 부담이 크다.

4.3 색인의 갱신 운영

객체지향 데이타베이스에서 한 객체에 대한 삽입, 삭제와 같은 갱신 연산은 집단화 계층과 상속 계층에서 연관된 다른 객체들에게도 파급효과를 미친다. 따라서 이러한 변화를 객체에 대한 직접 탐색 없이 색인에 효과적으로 반영하기 위한 방안이 필요하다.

Bertino와 Foscoli는 집단화 계층의 경로 상에서 루트 클래스의 상속 계층을 제외한 모든 클래스((그림 1)의 예에서 Vehicle, Bus, Truck, Company)의 키 값과 연관된 인스턴스들에 대하여 부모 리스트를 수록한 보조 색인을 사용하였다. 그러나, 전술한 바와 같이 보조 색인의 사용은 기본 레코드부에 한번 나타났던 OID들이 보조 색인부에 또 한번 중복 저장되어 기억 장소를 많이 소모하게 될 뿐만 아니라 색인의 유지보수 절차가 지나치게 복잡하다는 단점이 있다.

본 논문에서 제안한 객체 관계성 테이블로 구성된 색인은 이러한 보조의 자료구조가 전혀 필요치 않으며 테이블에 대한 탐색 항해만으로 객체의 갱신된 내

용을 색인에 효과적으로 반영할 수 있다. 본문의 예에 대하여 구성된 (그림 9)의 객체 관계성 테이블의 구성 하에서 Manufacturer 속성의 값으로 company ⟨j⟩를 가지는 vehicle⟨p⟩를 추가할 경우, 이 질의에 대한 삽입 연산은 다음과 같은 절차로 처리되게 된다.

먼저, 클래스-TID 사상 테이블을 참조하여 Vehicle 클래스의 TID 4와 Company 클래스의 TID 7을 채취한다. 7번 테이블을 접근하여 OID 필드의 값이 company⟨j⟩인 엔트리의 RID를 채취한 다음 (TID, RID) 쌍인 (7, 2)를 생성한다. 이후 4번 테이블을 접근하여 vehicle⟨p⟩를 위한 새로운 엔트리(RID는 4)를 한개 할당하고 OID 필드에 vehicle⟨i⟩를 기록한 후 참조 필드에는 (7, 2)를 기록한다.

또한, 삭제 연산의 예로서 vehicle⟨i⟩를 삭제하고자 하였을 경우에는 다음과 같은 절차로 처리된다. 먼저 클래스-TID 사상 테이블을 참조하여 Vehicle 클래스의 TID 4를 채취한 후 4번 테이블을 참조한다. 4번 테이블의 OID 필드를 검조하여 vehicle⟨i⟩가 위치한 엔트리의 RID를 채취한 후 (TID, RID) 쌍인 (4, 2)를 생성한다. 그리고 4번 테이블의 머리부에서 집단화-부모 필드 값인 1을 채취한 후 1번 테이블의 각 엔트리의 참조 필드를 검조하여 (4, 2)와 부합되는 엔트리(예에서 person⟨p⟩)를 삭제한다. 1번 테이블의 상속-자식 필드에서 TID 2와 3을 채취한 다음, 2번과 3번 테이블에 대해 각 엔트리의 참조 필드를 검조하여 (4, 2)와 부합되는 엔트리(예에서 employee⟨i⟩)를 삭제한다.

이상 살펴본 바와 같이 객체 관계성 테이블은 클래스 간의 집단화 및 상속 관계에 관한 정보와 인스턴스 간의 참조 정보를 함축시킨 메타 데이타로서 검색 및 삽입, 삭제를 위한 질의들을 효과적으로 처리할 수 있을 뿐만 아니라, 다중 경로 상에서 여러개의 색인 속성에 대해 발생하는 다중키 제한 질의도 테이블 구조를 공유함으로써 효율적으로 처리할 수 있다.

5. 성능 평가

5.1 시뮬레이션 환경

본 연구에서 제안한 객체 관계성 테이블의 성능은 집단화 및 상속 계층을 동시에 접근하는 질의에 적용이 가능한 다중-상속 색인과 중첩-상속 색인을 비교 대상으로 하여 저장비용과 검색비용 측면에서 평가하였으며, 시뮬레이션은 SONY-NEWS 워크스테이션(UNIX 운영체제)에서 "C"언어를 사용하여 수행하였다. 시뮬레이션에 적용된 비용식과 인수는 [5, 7, 9, 13]에서 제시된 내용을 참고하였으며, 다음과 같은 가정을 기반으로 하여 수행하였다.

1) 성능 평가는 질의 모델 중 집단화 계층과 상속 계층을 동시에 접근하는 질의만을 대상으로 한다.
2) 객체 관계성 테이블의 머리부의 크기는 몸체부의 한개 엔트리 크기와 동일하다.
3) 임의의 상속 계층에서 각 클래스의 인스턴스 수와 한 속성에 대해 서로 다른 키 값 의 수는 같다. (즉, $N_{i,1}=N_{i,2}$, ..., $N_{i,n}$ 그리고 $D_{i,1}=D_{i,2}$, ...,= $D_{i,n}$)

이때, 사용된 인수는 다음의 표 1과 같다.

⟨표 1⟩ 성능 평가에 사용된 인수
⟨Table 1⟩ Parameters used in performance evaluation

D_i	클래스 C_i의 한 속성에 대해 서로 다른 키 값의 수
N_i	클래스 C_i의 인스턴스 수
K_i	클래스 C_i의 한 속성의 키당 평균 OID의 수($K_i=\lceil N_i/D_i \rceil$)
$C_{i,j}$	클래스 C_i를 루트로 하는 상속 계층에서 j번째 클래스
$D_{i,j}$	클래스 $C_{i,j}$의 한 속성에 대해 서로 다른 키 값의 수
$N_{i,j}$	클래스 $C_{i,j}$의 인스턴스의 수
$K_{i,j}$	참조 공유 차수($K_{i,j}=\lceil N_{i,j}/D_{i,j} \rceil$)
σ	상관인자
OIDL	객체 식별자의 길이
f	비단말 노드의 평균 분기 수
P	페이지의 크기
pf	포인터 필드의 길이
kl	색인된 속성의 키 값의 평균 길이
kll	키 길이 필드의 크기
rl	레코드 길이 필드의 크기
no	OID 리스트의 개수 필드의 길이
of	중첩-상속 색인에서 범위 필드의 길이
L1	단말노드의 크기(페이지 수)
NL	비단말 노드의 크기(페이지 수)
nch(i)	클래스 Ci를 루트로 하는 상속 계층의 모든 클래스 수
len(p)	경로의 길이
XM(i)	경로 상에서 i번째 다중-상속 색인의 단말노드의 평균길이
XNI	중첩-상속 색인의 기본 레코드의 평균길이
np	레코드가 페이지 크기 보다 클 때 그 레코드가 점유하는 페이지 수
rf	객체 관계성 테이블에서 참조 필드의 평균길이
ridf	객체 관계성 테이블에서 RID 필드의 평균길이

그리고 상속 계층 전체를 지칭하는 첨자로서 "*" 표기를 사용하기로 한다. 즉, $C_{i,*}$는 클래스 C_i를 루트

로 하는 상속 계층의 모든 클래스를 지칭하고, 클래스 C_i를 루트로 하는 상속 계층의 모든 인스턴스의 수를 $N_{i,*}$, 클래스 C_i를 루트로 하는 상속 계층의 모든 서로 다른 키 값의 수를 $D_{i,*}$ 등과 같이 표기하기로 한다. 〈표 1〉의 인수 중에서 참조 공유 차수(degree of reference sharing)를 나타내는 $K_{i,j}$는 클래스 $C_{i,j}$의 한 속성에 대해 같은 값을 갖는 평균 인스턴스의 수를 나타내며, 이는 다른 클래스의 동일한 객체에 대해 $K_{i,j}$개 만큼 공유 참조가 있음을 의미한다. 본문의 (그림 2)의 예에서 vehicle(i)와 vehicle(j)가 동일한 객체인 company(j)에 대해 참조를 공유하고 있음을 알 수 있다.

또한 상관인자(correlation factor) σ는 한 인스턴스가 속성 값으로 집단화 계층에서 다른 인스턴스에 대한 여러개의 참조를 가질 때 이 참조 회수를 나타내는 것으로 본문의 (그림 2)의 예에서는 모든 인스턴스들이 $\sigma=1$인 경우를 나타내고 있다.

5.2 저장공간 평가를 위한 비용모델

5.2.1 다중-상속색인

다중-상속 색인은 경로의 길이만큼 색인 속성에 대해 상속 색인을 다중으로 유지하는 방법인데, 다중-상속 색인의 크기를 계산하기 위한 비용식을 기술하면 다음과 같다.

F1: 경로 상에서 i번째 색인의 크기:

$$XM(i) = K_{i,*} \times OIDL + kl + kll + rl + (OIDL + of) \times nch(i)$$

F2: 단말 노드의 페이지 수:

$$\begin{cases} L1_i = D_{i,*} / \lfloor P/XM(i) \rfloor & \text{if } XM(i) \le P \\ L1_i = D_{i,*} \times \lceil XM(i)/P \rceil & \text{if } XM(i) > P \end{cases}$$

F3: 비단말 노드의 페이지 수: $NL_i = \lceil L1_i/f \rceil + \lceil \lceil L1_i/f \rceil /f \rceil + \ldots + X$
(단, f로 나누어지는 각항의 결과가 f보다 작아지는 마지막 항을 X로 두었을 때, X가 1이 아니면 루트 노드를 위해 1을 더해줌)

F4: 경로 상에서 i번째 색인의 크기: $CS_i = L1_i + NL_i$

F5: 다중-상속 색인의 크기: $CS_* = \sum_{i=1}^{n} CS_i$

5.2.2 중첩-상속 색인

중첩-상속 색인의 크기는 기본 색인의 크기와 보조 색인의 크기를 합한 것이며, 기본 레코드의 크기는 경로의 마지막에 나타나는 색인 클래스의 속성에 대한 도메인의 크기와 참조 공유 차수의 크기에 의존한다. 또한 보조 색인의 크기는 집단화 계층의 루트 클래스와 그 서브 클래스를 제외한 나머지 클래스들의 참조 공유 차수의 크기에 비례한다. 중첩-상속 색인의 크기를 계산하기 위한 비용식은 다음과 같다.

F6: 기본 레코드의 평균 길이

$$XNI = OIDL \times \left[\sum_{i=1}^{n} \sum_{j=1}^{nch(i)} K_{i,j} \right] + kl + kll + rl + (OIDL + of + Pf) \times \left(\sum_{i=1}^{n} nch(i) \right)$$

F7: 기본 색인에서 기본 레코드(단말 노드)의 페이지 수:

$$\begin{cases} L1 = \lceil D_{n,*} / \lfloor P/XNI \rfloor \rceil & \text{if } XNI \le P \\ L1 = D_{n,*} \times \lceil XNI/P \rceil & \text{if } XNI > P \end{cases}$$

F8: 기본 색인에서 비단말 노드의 페이지 수:
$NL = \lceil L1/f \rceil + \lceil \lceil L1/f \rceil /f \rceil + \ldots + X$
(단, X는 F3와 동일함)

F9: 기본 색인의 크기: $IS(pri) = L1 + NL$

F10: $C_{i,*}$를 제외한 키 값과 연관된 OID의 수를 계산하는 비용식:

$$n(K) = \sum_{j=2}^{n-1} \left(\prod_{i=j}^{n} K_{i,*} \right)$$

F11: 보조 색인의 단말 노드의 페이지 수:

$$L1_{aux} = \lceil (n(K) \times OIDL + (OIDL + pf + no) \times \left[\sum_{i=2}^{n} \sum_{j=1}^{nch(i)} K_{i,j} \right]) / P \rceil$$

F12: 보조 색인의 비단말 노드의 페이지 수:
$NL_{aux} = \lceil L1_{aux}/f \rceil + \lceil \lceil L1_{aux}/f \rceil /f \rceil + \ldots + X$
(단, X는 F3와 동일함)

F13: 보조 색인의 크기: $IS(aux) = L1_{aux} + NL_{aux}$

F14: 다중-상속 색인의 크기: $IS = IS(pri) + IS(aux)$

5.2.3 객체 관계성 테이블

객체 관계성 테이블의 크기는 데이타베이스 스키마에 있는 클래스의 수와 클래스마다 할당된 인스턴스의 수에 비례한다.

F15: 참조 필드의 크기 : $rf = SIZE(TID,RID) \times \sigma$ (단, SIZE(TID,RID)는 참조 필드 의 요소인 (TID, RID) 쌍의 크기를 나타냄)

F16: 객체 관계성 테이블 한개의 크기:

$$TS1_{i,j} = \lceil (N_{i,j} \times (ridf + OIDL + rf))/P \rceil$$

F17: 객체 관계성 테이블 구조 전체의 크기:

$$TS = \sum_{i=1}^{n} \sum_{j=1}^{nch(i)} TS1_{i,j}$$

5.3 검색을 위한 비용 모델

5.3.1 다중-상속 색인

다중-상속 색인의 검색비용을 I/O 수로 계산하기 위한 비용식을 표현하면 다음과 같다.

F18: 경로 상에서 n번째 색인의 검색 비용(페이지 수):

$$\begin{cases} N(I/O)_n = \log_f NL_{n,*} + 1 & \text{if } XM(n) \le P \\ N(I/O)_n = \log_f NL_{n,*} + \lceil XM(n)/P \rceil & \text{if } XM(n) > P \end{cases}$$

이때, i번째 색인에 대한 검색비용을 계산하기 위해서는 i+1번째 색인을 검조했을 때 키 값과 연관되어 추출 될 OID수를 알아야 한다. 이 OID수를 NOID(i)로 표기 했을 때 NOID(i)를 구하는 식은 다음과 같다.

F19: $NOID(n-1) = K_{n,*}$

$NOID(i) = K_{i+1,*} \times K_{i+2,*} \times \ldots \times K_{n-1,*} \times K_{n,*}$

(단, $1 \le i < n-1$)

NOID(i)개의 OID들을 접근하는 비용은 12)에서 개발된 Yao의 공식을 사용하기로 한다. 이 공식은 m개의 페이지들로 구성되는 n개의 레코드 집합에 대해 k개의 레코드들을 임의 접근 했을 때의 검색 비용(페이지 수)을 나타낸다.

F20: Yao의 공식:

$$H(k, m, n) = m \times [1 - \sum_{i=1}^{k} \frac{(n - (\frac{n}{m} - i + 1)}{(n + m + 1)}]$$

F21: 다중-상속 색인에서 i번째 색인의 검색비용:

$$\begin{cases} N(I/O) = H(NOID(i), LI_{i,*}, D_{i,*}) + \log_f NL_i & \text{if } XM(i) \le P \\ N(I/O)_n = NOID(i) \times \lceil XM(i)/p \rceil + \log_f NL_i & \text{if } XM(i) > P \end{cases}$$

다중-상속 색인의 갱신 연산은 갱신될 레코드를 포함하는 단말 노드의 페이지를 읽는 비용과 이 페이지를 다시 쓰는 비용으로 대별할 수 있다.

F22: 다중-상속 색인의 갱신비용:

$$\begin{cases} N(I/O)_{update} = \log_f NL_{i,*} + 2 & \text{if } XM(i) \le P \\ N(I/O)_{update} = \log_f NL_{i,*} + 2 + (np-1)/np & \text{if } XM(i) > P \end{cases}$$

(단, $np = \lceil XM(i)/P \rceil$)

5.3.2 중첩-상속 색인

중첩-상속 색인에서는 집단화 및 상속 계층을 동시에 접근하는 질의를 처리하기 위해서 한번의 색인 탐색이 필요하며, 이러한 검색비용을 I/O수로 계산하기 위한 비용식은 다음과 같다.

F23: 중첩-상속 색인의 검색비용:

$$\begin{cases} N(I/O) = \log_f NL + 1 & \text{if } XNI \le P \\ N(I/O) = \log_f NL + \lceil XNI/P \rceil & \text{if } XNI > P \end{cases}$$

중첩-상속 색인의 갱신 비용은 보조 색인의 단말 노드에 접근해서 읽는 비용과 수정 후 다시 쓰는 비용, 그리고 보조 색인으로부터 포인터에 의해 접근하여 기본 색인를 읽고 쓰는 비용 다시 기본 색인으로부터 포인터에 의해 보조 색인을 접근하여 읽고 쓰는 비용으로 대별할 수 있다.

F24: 중첩-상속 색인의 갱신비용:

$$N_{update}(I/O) = \log_f NL_{aux} + 2b + 4$$

(단, $\begin{cases} b = 1 & \text{if } XNI \le P \\ b = \lceil XNI/P \rceil & \text{if } XNI > P \end{cases}$)

5.3.3 객체 관계성 테이블

객체 관계성 테이블은 클래스의 수만큼만 설정되고 테이블 한개당 수용하는 정보의 양이 적기 때문에 테이블 구조 전체를 주기억 장치에 상주시켜 처리하게 된다. 따라서 주기억 장치에 상주된 테이블에 대한 탐색항해는 별도의 I/O 비용을 필요로 하지 않는다.

5.4 시뮬레이션 결과 및 평가

본 연구에서는 시뮬레이션에 사용된 인수들 가운데 고정 값을 갖는 인수를 〈표 2〉와 같이 가정하였으며, K_i와 nch(i)의 크기의 변화에 따른 다양한 실험을 통하여 성능을 평가하였다. 먼저 한 클래스의 인스턴스 수가 N_i=10,000이고 클래스들 간에는 서로 같은 크기의 N_i와 D_i를 가진다고 가정했을 때, nch(i)=5일 경우에 K_i의 크기에 따르는 저장비용의 실험 결과는 〈표 3〉과 같다.

〈표 2〉 시뮬레이션을 위한 인수의 가정
〈Table 2〉 Assumptions of parameters for simulation

P=4096 바이트	OIDL	=8 바이트
kl=8 바이트	kll	=2 바이트
f=218	rl	=2 바이트
len(p)=5	of	=2 바이트
pf=2 바이트	cl	=2 바이트
no=2 바이트	σ	=1
ridf=2 바이트		

〈표 3〉의 실험 결과에서 보는 것처럼 본 논문에서 제안한 객체 관계성 테이블이 다른 기법에 비해 저장비용 측면에서 성능이 가장 우수함을 알 수 있다. 중첩-상속 색인의 경우에는 K_i 값이 증가할 수록 키 값과 연관된 부모 리스트의 크기가 커지기 때문에 보조 색인의 크기가 증가하여 전체 색인의 크기가 커지게 되는 반면, 다중-상속 색인은 K_i 값이 증가할 수록 D_i 값이 작아지기 때문에 색인의 크기가 작아지게 된다. 그러나 객체 관계성 테이블은 클래스 내에 분포된 서로 다른 키 값의 수인 D_i나 참조 공유 차수인 K_i의 크기와는 독립적이며, 단지 객체의 수인 N_i의 크기에 비례하기 때문에 K_i 값의 증가 여부와는 상관없이 항상 일정한 값을 유지하게 된다. 위 실험 결과는 한 클래스의 객체 수가 10,000개, 전체 객체 수가 250,000개인 대용량 데이타베이스의 경우를 대상으로 하였으나, 본 논문에서 제안한 객체 관계성 테이블 기법은 저장비용으로서 단지 875 페이지만을 점유하기 때문에 실제 적용에 있어서 충분히 주기억장치에 적재하여 운영이 가능함을 알 수 있다.

〈표 4〉는 N_i=10,000으로 설정했을 때, 한 상속 계층의 클래스의 수를 나타내는 nch(i)의 변화에 따른 저장비용을 K_i=5, K_i=10, K_i=20에 대해 각각 실험한 결과이다. 이때, nch(i) 값을 증가시키면 한 상속 계층의 클래스 수가 증가하기 때문에 전체 객체의 수가 증가하게 된다. 이 실험에 있어서도 본 논문에서 제안한 객체 관계성 테이블이 저장비용 측면에서 압도적으로 유리함을 알 수 있다. nch(i)가 1일 경우에는 전체 객체의 수가 50,000에 불과하여 중첩-상속 색인의 경우에도 보조 색인이 차지하는 크기가 상대적으로 작아서 그다지 큰 격차가 나지 않으나, nch(i)의 값이 증가하여 객체의 수가 늘어날 수록, 또 K_i 값이 증가할 수록 중첩-상속 색인의 전체 색인의 크기가 급격하게 증대됨을 알 수 있다. 또 저장비용 측면에서 중첩-상속 색인보다 유리한 다중-상속 색인과 비교했을 때도 본 논문에서 제안한 객체 관계성 테이블의 성능이 우수하다는 것을 보여준다.

다음으로 검색비용에 관해 고찰하기로 한다. 다중-상속 색인은 색인 속성의 키 값과 연관된 OID 리스트를 추출하기 위해서 K_i에 의존하여 여러번의 색인 탐색을 시행해야 하기 때문에 비용 부담이 클 수밖에

〈표 3〉 N_i = 10,000일때 K_i의 변화에 따른 저장비용의 실험 결과
〈Table 3〉 The experimental results of storage costs in accordance with the change of K_i when N_i = 10,000

저장 비용 (페이지수) N_i=10,000, nch(i)=5								
K_i / 색인종류	K_i=2	K_i=4	K_i=6	K_i=8	K_i=10	K_i=12	K_i=14	K_i=16
다중-상속 색인	4,490	3,495	3,225	3,140	3,140	2,995	2,995	2,995
중첩-상속 색인	4,210	4,519	5,832	8,293	17,538	30,055	51,373	84,526
객체관계성테이블	875	875	875	875	875	875	875	875

〈표 4〉 $N_i = 10,000$일 때 nch(i)의 변화에 따른 저장비용의 실험결과

〈Table 4〉 The experimental results of storage costs in accordance with the change of nch(i) when $N_i = 10,000$

저장비용 (페이지수) $N_i = 10,000$							
K_i	색인종류	nch(i) = 1	nch(i) = 2	nch(i) = 3	nch(i) = 4	nch(i) = 5	nch(i) = 6
$K_i = 5$	다중-상속 색인	160	565	1,200	2,120	3,355	4,645
	중첩-상속 색인	137	598	1,314	3,112	4,150	7,673
	관계성 테이블	175	350	525	700	875	1,050
$K_i = 10$	다중-상속 색인	130	485	1,080	1,830	3,140	4,310
	중첩-상속 색인	149	837	3,152	7,163	17,538	31,890
	관계성 테이블	175	350	525	700	875	1,050
$K_i = 20$	다중-상속 색인	120	460	1,080	2,945	3,140	5,025
	중첩-상속 색인	458	5,656	27,369	83,396	203,220	416,316
	관계성 테이블	175	350	525	700	875	1,050

없다. 중첩-상속 색인의 검색 비용은 기본 레코드를 채취하기까지의 B^+-트리의 탐색비용으로 볼 수 있고, 기본 색인의 단말노드의 크기가 페이지 크기 보다 클 때는 추가의 I/O 비용을 부담해야 한다. 객체 관계성 테이블은 테이블 구조 전체를 주기억 장치에 상주시켜 처리하게 됨으로 별도의 입출력을 필요로 하지 않는다.

〈표 5〉는 $N_i = 10,000$, nch(i) = 2, len(p) = 2, $K_2 = 1$로 가정했을 때 K_i의 변화에 따른 검색비용을 실험한 결과이다. 표에서 보는 것처럼 다중-상속 색인의 검색 시에는 다중으로 유지되는 상속 색인마다 키 값과 연관되어 검색해야 할 OID 수가 많이 발생하기 때문에 비용 부담이 가장 높고, 이러한 비용 부담은 K_i값이 증가할 수록 계속 증대됨을 알 수 있다. 중첩-상속 색인의 경우에는 기본 레코드를 채취하기 위해 B^+-트리를 탐색하는데 드는 입출력 비용만을 부담하게 된다. 실험 결과에서는 평균 3회의 입출력 비용이 필요한 것으로 나타나지만, 만약 K_i 값이 증가하여 기본 색인의 단말노드의 크기가 페이지 크기 보다 커지게 되면, 이에 따르는 추가의 입출력 비용만을 부담하면 된다. 또한 객체 관계성 테이블의 경우에는 〈표 3〉의 실험 결과에서 나타낸 바와 같이 객체 수가 250,000개일 때 저장비용으로 단지 875 페이지만을 점유하기 때문에 충분히 주기억장치에 적재하여 운영이 가능

〈표 5〉 $N_i = 10,000$, nch(i) = 2, len(p) = 2, $K_2 = 1$일 때 K_1의 변화에 따른 검색비용

〈Table 5〉 The experimental results of retrieval costs in accordance with the change of K_i when $N_i = 10,000$, nch(i) = 2, len(p) = 2, $K_2 = 1$

검색 비용 (I/O수) $N_i = 10,000$, nch(i) = 2, len(p) = 2, $K_2 = 1$								
K_1 / 색인종류	$K_1 = 2$	$K_1 = 4$	$K_1 = 6$	$K_1 = 8$	$K_1 = 10$	$K_1 = 12$	$K_1 = 14$	$K_1 = 16$
다중-상속 색인	9	16	25	33	39	41	43	45
중첩-상속 색인	3	3	3	3	3	3	3	3
객체관계성테이블	주기억장치 상주							

〈표 6〉 N_i = 10,000일 때 K_i의 변화에 따른 갱신비용의 실험 결과

〈Table 6〉 The experimental results of retrieval costs in accordance with the change of K_i when N_i = 10,000

갱신비용 (I/O수) N_i = 10,000, nch(i) = 5, len(p) = 5								
K_i / 색인종류	K_i = 2	K_i = 4	K_i = 6	K_i = 8	K_i = 10	K_i = 12	K_i = 14	K_i = 16
다중-상속 색인	3	3	3	3	3	3	3	3
중첩-상속 색인	6	6	6	6	6	6	7	7
객체관계성테이블	주기억장치 상주							

하며, 따라서 별도의 입출력 비용을 들일 필요가 없게 된다.

다음의 〈표 6〉은 한 클래스의 인스턴스 수가 N_i = 10,000이고 클래스들 간에는 서로 같은 크기의 N_i와 D_i를 가진다고 가정했을 때, nch(i) = 5일 경우에 K_i의 크기에 따르는 갱신비용의 실험 결과이다. 다중-상속 색인은 보조 색인을 사용하는 중첩-상속 색인 보다 성능이 우수한 것으로 나타났다. 즉, 다중-상속 색인의 갱신 연산은 갱신될 레코드를 포함하는 단말 노드의 페이지를 읽는 비용과 이 페이지를 다시 쓰는 비용으로 대별할 수 있으나, 중첩-상속 색인의 경우에는 보조 색인의 단말 노드에 접근해서 읽는 비용과 수정 후 다시 쓰는 비용, 그리고 보조 색인으로부터 포인터에 의해 접근하여 기본 색인을 읽고 쓰는 비용, 다시 기본 색인으로부터 포인터에 의해 보조 색인을 접근하여 읽고 쓰는 비용으로 대별할 수 있기 때문에 평균 3회 정도의 I/O 비용을 추가로 지불해야 한다. 물론 기본색인의 단말 노드의 크기가 페이지 크기를 초과하면 그만큼의 I/O 비용을 더 부담해야 하는데, 〈표 6〉에서 K_i = 14 이상일 때가 중첩-상속 색인의 단말 노드의 크기가 페이지 크기보다 커지는 경우이다. 그리고 객체 관계성 테이블의 갱신비용은 앞서의 검색의 경우와 동일하게 주기억장치에 상주하기 때문에 별도의 입출력 비용을 필요로 하지 않는다.

6. 결 론

객체지향 데이타베이스 시스템에서는 집단화 계층과 상속 계층에서 최적의 접근 경로를 찾아 효율적인 입출력을 수행할 수 있도록 하는 색인기법이 중요하게 취급되고 있다. 본 연구에서는 집단화 계층과 상속 계층을 동시에 접근하는 질의를 효율적으로 처리할 수 있는 객체 관계성 테이블을 사용한 색인 기법을 제안하였다. 제안된 기법은 클래스 간의 집단화 및 상속 관계에 관한 정보와 인스턴스 간의 참조 정보를 메타 데이타 형태로 테이블 내에 함축시킨 테이블 구조를 사용함으로써, 다양한 질의를 효과적으로 처리할 수 있을 뿐 아니라, 크기가 작아 주기억 장치에서 운영이 가능하다.

이 분야의 대표적 기법인 중첩-상속 색인에 비해 제안된 방법의 장점은 테이블간의 탐색항해 만으로도 추가의 스키마 의미 정보의 도움없이 "SELECT-ALL" 형태의 질의를 처리할 수 있으며, 보조 색인부에서 발생하는 데이타의 중복 저장을 회피하여 기억장소의 낭비를 줄임과 동시에, 갱신 연산 시에 발생하는 색인의 유지 보수 절차를 대폭 간략화 했다는 것이다. 또한, 이 객체 관계성 테이블 구조를 서로 다른 다중키 접근으로부터 공유가 가능하도록 함으로써 별도의 색인 편성 없이도 다중 경로에 대한 다중키 접근을 가능하게 하였다.

제안된 방법은 기존의 색인 기법들과 다양하게 비교하고 효용성에 관해 논의하였으며, 저장비용과 검색비용 측면에서 그 성능을 시뮬레이션 한 결과를 제시하였다. 본 연구는 객체지향 데이타베이스인 ORION에서 사용되는 저장 모델을 중심으로 연구되었으며, 후속 연구로는 보다 다양한 저장모델을 대상으로 중첩 및 상속 계층을 동시에 접근하는 질의에 대해 적용이 가능하도록 확장하는 것이다.

참 고 문 헌

[1] Hurson A.R. and Simin h. Parkzad, "Object-Ori-

ented Database Management Systems: Evolution and Performance Issues," IEEE COMPUTER, vol. 26, NO. 2, pp. 48-60, 1993.

[2] Kim W., Introduction to Object-Oriented Databases, The MIT Press, pp. 107-126, 1990.

[3] Maier D. and Stein J., "Indexing in an object-oriented DBMS," Proc. IEEE Workshop on Object-Oriented DBMS, pp. 171-182, 1986.

[4] Valduriez, P., "Join indices," ACM Trans. on Database Systems, Vol. 12, NO. 2, pp. 218-246, 1987.

[5] Bertino E. and Kim W., "Indexing Techniques for Queries on nested objects," IEEE Trans. on Knowledge and Data Engineering, vol. 1, NO. 2, pp. 196-214, 1989.

[6] Kemper A., Moerkotte G., "Access support in object bases," Proc. of the ACM-SIGMOD, pp. 364-374, 1990.

[7] Kim W., Kim K. C., Dale A., "Indexing Techniques Object-Oriented Databases in Object-Oriented Concepts, Databases, and applications," pp. 371-394, Edited by Addison-Wesley, 1989.

[8] Low C. C., Ooi B. C., Lu H. J., "H-trees: A Dynamic Associative Search Index for OODB," Proc. of the ACM SIGMOD, pp. 134-143, 1992.

[9] Bertino E. and Foscoli P., "Index Organizations for Object-Oriented Database Systems," IEEE Trans. on Knowledge and Data Engineering, vol. 7, NO. 2, pp. 193-209, 1995.

[10] Willshire M.J. and Kim H.J., "Properties of Physical Storage Models for Object-Oriented Databases," Proc. of Int'l. Conf. PARABASE-90, pp. 94-99, 1990.

[11] Kim W., et al., "Features of the ORION Object-Oriented Database System, in Object-Oriented Concepts, Databases, and Applications," pp. 251-282, Edited by Addison-Wesley, 1989.

[12] Bertino E., "An Indexing Technique for Object-Oriented Databases," Proc. IEEE Int'l. Conf. on Data Engineering, pp. 160-170, 1991.

[13] Yao S. B., "Approximating Block Accesses in Database Organizations," in Comm. of ACM, Vol. 20, No. 4, pp. 260-261, 1977.

부 기 동

1983년 경북대학교 전자공학과 전자계산기전공 졸업(학사)

1988년 경북대학교 대학원 전자공학과 전산공학전공 졸업(공학석사)

1988년~현재 경북대학교 대학원 전자공학과 전산공학전공 박사과정 수료

1983년~1985년 포항종합제철 시스템개발실

1988년~현재 경북산업대학교 전자계산학과 부교수

관심분야: 데이타베이스(특히, 객체지향 및 공간 데이타베이스)

이 상 조

1974년 경북대학교 수학교육과 졸업(학사)

1976년 한국과학원 전산학과 졸업(이학석사)

1994년 서울대학교 컴퓨터공학과 졸업(공학박사)

1976년~현재 경북대학교 컴퓨터공학과 교수

관심분야: 운영체제, 자연언어처리, 기계번역