객체지향 데이타베이스 시스템에서 식별자를 이용한 로킹 프로토콜

배 석 찬† 황 부 현††

요 약

본 논문에서는 객체지향 데이타베이스 시스템에서 동시성을 향상 시킬 수 있는 로킹 프로토콜을 제안하였다. 제안한 방법은 복합 객체의 요소 객체에 그 객체의 루트를 확인하도록 부여된 루트 객체 식별자를 이용한다. 복합 객체에 로크를 요구하는 하나의 트랜잭션은 접근하고자 하는 요소 객체의 루트 객체 식별자가 있는지, 그 루트 객체에 로크되어 있는 지를 검사한다. 그러나 하나의 트랜잭션에 의해 어느 한 복합 객체의 루트 객체가 로크되어 있더라도 또 다른 트랜잭션들은 복합 객체의 요소 객체가 속한 클래스내의 다른 루트 객체 식별자를 갖는 인스턴스를 접근해서 읽거나 갱신하게 함으로써 동시성을 증가시킬 수 있다.

The Locking Protocol using Identifier in an Object-Oriented Database Systems

Seok Chan Bae† and Bu Hyun Hwang††

ABSTRACT

In this paper, a new locking protocol which can increase the concurrency in an Object Oriented Database Systems is proposed. The proposed locking protocol uses the Root object identifier to identify the root of a component object in a composite object. The transaction that requests a lock on the component object checks whether the Rid of the component object has been locked by another transaction. Even though the class of the component object is locked by one transaction, other transactions can concurrently access other instances in the class, if each Rid of them is not locked by any other transactions. This can increase the degree of concurrency.

1. 서 론

컴퓨터 기술이 급속도로 발전함에 따라 사무 자동화, 컴퓨터 이용 설계 및 제작, 멀티미디어 데이타 처리등 여러 분야로 컴퓨터를 이용한 응용 환경이 확산되어 가고 있다.

기존의 데이타베이스 시스템에서 트랜잭션 처리 기법은 2단계 로킹(two-phase locking) 방법, 시간 표지 순서화(timestamp ordering)방법, 낙관적인(optimistic)방법등 여러가지가 있으나 상업용 시스템에서는 2단계 로킹 방법을 사용하고 있다. 그러나 새로이 부각되고 있는 객체지향 데이타베이스 환경에서는 복합 객체와 계층형 데이타 등을 표현할 수 있어야 하고, 사용자가 새로이 정의한 데이타형을 추가할 수 있어야 한다. 또한 클래스의 속성과 행위를 계승받아 하위 클래스를 생성할 수 있어야 하고, 클래스의 변경을 위해서는 하위 클래스까지 로크해야 하므로, 기존 데이타베이스 시스템의 트랜잭션 기법으로는 처리하기가 어렵다[10, 11, 12].

트랜잭션 처리의 목적은 데이타베이스내의 데이타를 동시에 여러 사용자들이 서로간의 간섭없이 접근 및 처리를 하여, 데이타의 일관성이 유

† 정 회 원 : 서남대학교 전산통계학과 전임강사
†† 정 회 원 : 전남대학교 전산학과 교수
논문접수 : 1994년 2월 18일, 심사완료 : 1994년 7월 20일

지되도록 하는 데 있다. 트랜잭션은 데이타베이스의 데이타에 대한 읽기와 쓰기의 연속이다. 데이타베이스의 일관성을 보장하기 위하여 트랜잭션의 원자성(Atomicity)과 직렬성(Serializability)은 보장되어야 한다[3, 11].

원자성은 트랜잭션의 전체가 수행되거나 아니면 전혀 수행되지 않거나 하는 원자적 행위를 의미한다. 이는 트랜잭션이 완료(commit)되지 않았을 때, 시스템은 데이타베이스에 기록된 모든 쓰기를 철회시키나 트랜잭션이 완료되었을 때, 데이타베이스에 모든 쓰기를 보장한다.

직렬성이란 하나 이상 트랜잭션들을 동시에 실행한 효과가 한번에 한 트랜잭션씩 순서적으로, 즉 직렬로 실행한 결과와 동일함을 의미한다. 다중 프로그래밍(multiprogramming)환경하에서 여러개의 트랜잭션이 동시에 실행될 수 있다. 이와 같이 실행되는 트랜잭션들이 데이타베이스의 일관성(consistency)을 유지하도록 데이타베이스 시스템은 트랜잭션간의 상호작용(interaction)을 제어할 필요가 있다[3, 11].

직렬성을 보장하는 하나의 방법은 데이타의 접근을 상호 배타적으로 하는 것이다. 즉, 한 트랜잭션이 데이타를 접근하는 동안 다른 트랜잭션이 그 데이타를 변경할 수 없다. 그래서 데이타에 대한 로크를 소유한 트랜잭션만 그 데이타에 접근할 수 있도록 하는 것이다[3].

객체지향 데이타베이스 시스템에서도 기존의 데이타베이스 시스템에서와 같이 트랜잭션들이 동시 수행한 결과는 트랜잭션들이 직렬로 실행한 결과와 동일해야 한다. 이처럼 객체지향 데이타베이스 시스템에서도 원자성과 직렬성을 만족하여야 한다[11].

Kim[5]는 로킹 방법을 확장하여 객체지향 시스템에서 다른 트랜잭션들에 속하는 요소 객체들의 모임인 복합 객체에 적용하기 위한 복합 객체 로크 모드를 제안하였고, 복합 객체 계층에 속하는 객체 내부의 한 인스턴스를 접근하기 위하여 그 복합 객체가 로크된다. 이 복합 객체를 로크하기 위하여 그의 요소 객체가 속한 클래스까지도 로크되므로, 다른 트랜잭션들이 이 클래스에 속한 다른 객체들의 접근을 불가능하게 한다.

본 논문에서는 기존 객체지향 시스템에서 사용한 동시성 제어 기법[5, 7, 8]의 동시성이 낮아지는 문제점을 개선하기 위해서 여러 트랜잭션들이 동시에 복합 객체의 요소 객체가 속한 클래스에 접근하여 다른 객체를 읽거나 갱신할 수 있는 로킹 프로토콜을 제안한다.

제안한 방법은 복합 객체의 요소 객체에 그 객체의 근원, 즉 루트(Root)를 확인하도록 부여된 루트 객체 식별자(Root object identifier)를 이용한다. 하나의 트랜잭션이 복합 객체의 요소 객체가 가지고 있는 루트 객체 식별자를 이용하여, 읽거나 갱신하고자 하는 객체에 대해 로크를 요구한다. 로크를 요구한 트랜잭션은 자신이 접근하고자 하는 복합 객체의 요소 객체가 가지고 있는 루트 객체 식별자를 이용하여, 루트 객체가 로크되어 있는지를 검사할 수 있다. 어떤 복합 객체의 루트 객체가 로크되어 있더라도, 다른 트랜잭션들은 이 복합 객체의 요소 객체가 속한 클래스 내의 로크되지 않은 다른 루트 객체 식별자를 갖는 인스턴스들을 읽거나 갱신할 수 있게 된다.

제안한 로킹 프로토콜은 여러 트랜잭션들이 동시에 복합 객체의 요소 객체가 속한 클래스에 접근하여 읽거나 갱신을 하도록 허용하기 때문에 동시성이 향상된다.

본 논문의 구성은 다음과 같다. 2장에서는 객체지향 개념 및 기존의 객체지향 데이타베이스 시스템에서 동시성 제어 기법에 관하여 소개한다. 3장에서는 동시성을 향상시킬 수 있는 새로운 로킹 프로토콜을 제시하고, 4장에서는 본 연구를 통하여 얻어진 결론과 앞으로의 연구방향을 서술한다.

2. 관련연구

2.1 객체지향 개념

객체지향 개념은 복잡한 소프트웨어 시스템 설

계등을 관리하기 위하여 설계되었다. 이 개념은 캡슐화(encapsulation) 및 계승을 통해 속성과 행위를 재사용하여 보다 더 복잡한 데이타베이스와 프로그램을 구축하는 기초가 되었다.

객체지향 시스템에서 모든 객체들은 시스템내에서 유일한 객체 식별자(object identifier)를 통해 참조된다. 또한 객체들은 상태(state)와 행위(behavior)로 서술된다. 객체의 상태는 속성으로 표현되며, 행위는 메소드(method)라고도 하며 속성을 조정하는 함수를 정의하는 프로그램이다. 각 객체는 그 객체의 속성과 행위를 정의한 클래스의 인스턴스라 하고 모두가 같은 메시지(message)에 응답한다. 어떤 객체에게 연산을 수행시키기 위해서는 그 객체에게 연산과 그에 관련된 정보를 표시한 메시지를 보내야 한다. 메시지 전송은 객체사이의 유일한 통신수단이며 다른 언어에서 처럼 프로시듀어 호출이나 변수로 직접적인 접근은 하지 않는다[5, 7, 12].

클래스는 성질이 유사한 객체들의 모임이고, 클래스 계층에서 하위 클래스는 상위 클래스의 속성과 행위를 계승받으며, 자신의 고유한 속성과 행위를 확장해서 가질 수 있다[12].

객체가 지닌 속성들이 원자적(atomic)인 것이 아니고 객체 자체를 의미하며 튜플(tuple), 집합(set), 리스트(list), 배열(array)로 구성되는 것을 복합 객체라 하고, 하나의 복합 객체를 구성하기 위한 각각의 객체를 요소 객체(component object)라 한다. 이처럼 요소 객체의 존재는 그 객체의 상위 객체의 존재에 의존되므로 종속 객체(dependent object)라 한다. 또한 복합 객체는 이질적인 객체들의 집합으로 하나의 객체가 다른 객체의 요소가 되는 것으로서 is part of" 관계를 표현한다[2, 12].

2.2 동시성 제어 기법

객체지향 데이타베이스 시스템에서 동시성 제어 기법에 대하여 살펴보면 Servio Logic에서 개발한 GemStone[10]에서는 낙관적인 방법과 쉐도잉(shadowing)방법을 이용함으로써 하나의 트랜잭션을 처리하는 동안 다른 트랜잭션의 접근을 제한함으로써 데이타베이스가 일관적인 상태를 유지하게 한다[6, 10]. MCC의 ORION[5]에서는 복합 객체를 로킹 단위로 하였으며, 단위 로킹, 클래스 격자, 복합 객체 로킹 방법을 이용하였다[5, 7]. Hewlett Packard에서 개발한 IRIS[4]는 동시성 제어 기법이 아직 완벽히 구현되지 않았으나 기본적으로 2단계 로킹 방법을 이용하였다[4, 11]. Altair의 O2[9]에서는 WISS(SIsconsin Storage System)는 화일이나 페이지를 로킹 단위로 하는 2단계 로킹 방법을 선택하였다[9, 13].

단위 로킹(granularity locking)프로토콜은 데이타베이스를 참조하는 데 얻어야 할 로크의 수를 최소화한다. 이 로킹 방법에서는 5가지의 로크 모드(IS, IX, S, X, SIX)가 있다. 그래서 클래스 인스턴스들은 5가지 모드중 어떤 로크 모드라도 가질 수 있다. 클래스에 대해 IS(Intention Shared) 로크 모드는 암시적 공유 접근을 허용하는 것으로 그 클래스의 인스턴스가 필요에 의해 S 모드로 로크로 되어 있다는 의미이다. 클래스에 대한 IX(Intention eXclusive) 로크 모드는 그 클래스내의 어떤 인스턴스가 X 모드로 로크되어 있다는 의미이다. 클래스의 S(Shared) 로크 모드는 그 클래스내의 어떤 인스턴스들이 암시적으로 S 모드로 로크되어 그 하위 클래스에 공유 접근을 허용한다. 클래스에 대한 SIX(Shared Intention eXclusive) 로크 모드는 클래스의 정의에는 S 모드로 로크를 걸고, 그 클래스의 모든 인스턴스에 암시적으로 S 모드로 로크를 걸고, 갱신될 인스턴스는 X 모드로 로크된다. 클래스의 X(eXclusive) 로크 모드는 클래스 정의와 클래스의 모든 인스턴스들을 검색 및 갱신할 수 있는 로크 모드이다[1].

하나의 클래스내 대부분의 인스턴스를 접근하고자 할 때 각각의 인스턴스에 로크하기보다는 그 클래스에 대한 하나의 로크가 훨씬 효율적이나 클래스내의 몇몇의 인스턴스에만 접근하고자 할 때에는 다른 트랜잭션들이 접근할 수 있도록 인스턴스별로의 로크가 바람직하다[3, 12].

단위 로킹을 객체지향 시스템에 그대로 적용하

면 전체의 복합 객체를 로크하기 위해서는 복합 객체의 루트 클래스가 5가지 모드중 하나로 로크된다. 만약 복합 객체의 루트 클래스가 IS, S, IX, SIX로 로크되면 복합 객체의 루트 객체는 각각 S, S, X, X 모드로 로크되며 요소 객체도 5가지 모드중 하나로 로크된다. 그러나 복합 객체의 요소 객체들은 암시적으로만 로크되어, 즉 로크되지 않은 요소 객체들을 다른 트랜잭션들이 갱신할 때 문제가 발생하게 된다[5, 12].

Garza[7]에서는 복합 객체를 하나의 클래스나 클래스의 인스턴스처럼 로크 가능한 단위로 취급하였다. 만약 단위 로킹을 복합 객체에 그대로 적용한다면, 복합 객체 계층의 모든 요소 클래스를 로크하거나, 하나의 복합 객체내에 있는 모든 요소 객체를 로크해야 한다. 결국 복합 객체 계층에 속하는 모든 객체를 로크 하여 로크의 수가 증가하게 되므로, 복합 객체 로킹 프로토콜에서는 복합 객체를 하나의 로크 단위로 해야 한다.

3. 식별자 로킹 프로토콜

3.1 복합 객체 로킹 방법

복합 객체 전체를 로크하기 위해 루트 클래스는 IS, IX, S, SIX, X 모드로 로크되는데, 만약 복합 객체의 루트 클래스가 IS, S, IX, SIX 모드로 로크되면 그 복합 객체의 루트 객체는 각각 S, S, X, X 모드로 로크되며, 요소 클래스(component class)는 IS, IX, S, SIX, X 모드로 로크된다. 예를 들어 트랜잭션 T1이 복합 객체 계층에 있는 하나의 복합 객체를 접근하여 갱신하고자 할 때 루트 클래스에 IX 모드로 로크하고, 갱신하고자 하는 객체에 X 모드로 로크, 요소 클래스들을 각각 IX 모드로 로크한다. 그리고 트랜잭션 T2는 T1이 지금 접근하고 있는 복합 객체 계층의 요소 클래스내의 한 인스턴스와 관련된 요소 클래스내의 다른 인스턴스를 갱신 하고자 할 때, T2도 그 루트 클래스와 해당 인스턴스가 속한 클래스에 IX 모드로 로크를 하고, 해당 인스턴스에 X 모드로 로크를 한 후 갱신하므로써, 동일한 클래스내의 관련된 인스턴스를 트랜잭션 T1과 T2가 동시에 갱신하여 데이타베이스의 일관성을 파괴하는 문제가 발생될 수 있다[7, 12].

이러한 문제점을 해결하기 위해 Kim[5]에서는 IS, IX, SIX 모드와 유사하게 ISO, IXO, SIXO 모드를 도입 하는데 (그림 1)에서 ISO 모드는 IX 모드와 충돌하고, IXO와 SIXO 모드도 IS, IX 모드와 충돌한다. 여기서 충돌이란 하나의 트랜잭션은 다른 트랜잭션과의 현재 모드와 요구 모드가 서로 호환될 수 없는 상태를 말한다. 새로운 로크 모드를 도입한 프로토콜에서는 복합 객체를 로크하기 위해서 루트 클래스가 IS, IX, S, SIX 또는 X 모드로 로크 되어야 하지만, 복합 객체 계층에서 각각의 요소 클래스들은 ISO, IXO, S, SIXO 또는 X 모드로 로크 된다[5, 7, 8].

이와같이 세개의 새로운 로크 모드를 도입함으로써 단위 로킹을 복합 객체에 적용했을때의 문제점은 해결했으나 복합 객체 계층을 통해 단지 하나의 판독자만 있을지라도 클래스내의 다른 인스턴스들에 대해 동시 수행이 불가능하게 된다.

복합 객체 계층의 인스턴스 예를 나타내는 (그림 6)에서 Kim[5]의 로킹 방법을 이용하여 하나의 트랜잭션 T1이 클래스 학회 의 인스턴스 1401을 접근하여 갱신하고자 한다면, 루트 클래스 문서를 IX 모드로 로크, 루트 객체 1101은 X 모드로 로크되고, 요소 클래스 '문서-저자', '프로젝트', '학회' 그리고 '도시'는 IXO 모드로 로크되고, 인스턴스 1401에 X 모드로 로크를 한다. 이때 또하나의 트랜잭션 T2가 T1이 지금 접근하고 있는 복합 객체와는 무관하게 요소 클래스 '학회'의 한 인스턴스 1402를 접근하여 읽고자 한다면, 읽고자하는 인스턴스가 포함된 요소 클래스 '학회'에 IS 모드로 로크를 요구하려고 하지만, 그 IS 로크 모드가 (그림 1)에서 IXO 모드와 충돌이 발생하므로 접근이 불가능하게 된다. (그림 1)은 단위 로킹과 복합 객체 로킹에 대한 양립성 행렬을 나타낸다[5, 7, 8, 12].

복합 객체에 대한 로킹 방법[12]은 한 트랜잭션이 복합 객체를 접근하여 읽거나 갱신하고 있

을 때, 다른 트랜잭션들은 복합 객체 계층의 요소 클래스에 접근이 불가능하게 됨에 따라 동시성이 낮아지는 문제가 발생한다. 즉, 복합 객체를 접근하기 위해 요소 클래스들 모두에 로크됨으로써 그 클래스내의 다른 인스턴스까지도 함께 로크되어 다른 트랜잭션들의 접근이 제한된다.

요구 모드

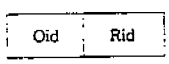
현재모드	IS	IX	S	SIX	X	ISO	IXO	SIXO
IS	Y	Y	Y	Y	N	Y	N	N
IX	Y	Y	N	N	N	N	N	N
S	Y	N	Y	N	N	Y	N	N
SIX	Y	N	N	N	N	N	N	N
X	N	N	N	N	N	N	N	N
ISO	Y	N	Y	N	N	Y	Y	Y
IXO	N	N	N	N	N	Y	Y	N
SIXO	N	N	N	N	N	Y	N	N

(Y : compatible N : incompatible)

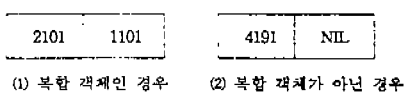
(그림 1) 단위 로킹과 복합 객체 로킹를 위한 양립성행렬
(Fig. 1) Compatibility Matrix for Granularity Locking and Composite Object Locking

3.2 식별자를 이용한 로킹 프로토콜

복합 객체의 요소 객체에 그 객체의 루트를 확인할 수 있도록 루트 객체 식별자를 부여한다. 그러나 복합 객체 계층에서 복합 객체의 요소 객체가 속한 클래스에 있는 객체중 루트 객체의 식별자가 없는 것도 있다. 즉, 별도의 인스턴스로 존재할 수 있다.



(그림 2) 자료 구조
(Fig. 2) Data Structure



(그림 3) 복합 객체의 자료 구조 예
(Fig. 3) Example Data Structure of Composite Object

(그림 2)는 복합 객체 계층에서 이용되는 자료 구조를 나타낸다. 자료 구조내의 Oid는 시스템내에서 유일한 객체 식별자(Object identifier), Rid는 루트 객체를 확인할 수 있도록 부여된 루트 객체 식별자이다.

(그림 3)은 자료 구조에서 복합 객체인 경우와 복합 객체가 아닌 경우를 예를들어 보였다. (그림 3)의 (1)에서 첫번째 항목은 시스템내에 유일한 각 객체의 식별자를 나타냈고, 그리고 두번째 항목은 객체의 루트를 확인할 수 있도록 부여된 루트 객체 식별자이다. (그림 3)의 (2)에서 첫번째 항목은 (1)에서와 동일하고, 두번째 항목은 루트 객체 식별자가 없는 인스턴스로 단일 인스턴스로 존재할 수 있다는 의미이다.

제안한 식별자를 이용한 동시성 제어 알고리즘에 필요한 변수들의 의미는 다음과 같다.

1) Instance : 임의의 한 instance.
2) Root_object : 임의의 한 instance의 루트 객체.
3) Root_class : Root_object가 속한 루트 클래스.
4) Component_class : 복합 객체 계층상에 나타나는 Root-object의 요소 클래스.
5) Class-contain_instance : 임의의 한 instance를 포함하고 있는 클래스.
6) Waiting_Transaction : 기다리고 있는 하나의 트랜잭션.
7) Req_Lock(Name_var, Lock_mode) : Name_var는 임의의 한 instance의 Root_class, Root_object, Component_class, Class_contain_instance, 검색 및 갱신하고자 하는 instance를 나타내고, Lock_Mode는 5가지 로크 모드(IS,IX,S,X, SIX)중 하나로 로크될 수 있는 로크 모드를 요구.
8) Rel_Lock(Name_var, Lock_mode) : Name_var의 lock을 해제.
9) Lock_Status(Name_var) : Name_var의 lock 상태를 표현.
10) Rid(Instance) : 임의의 한 instance의 root object identifier.
11) Update(Instance) : 임의의 한 instance를 갱신.
12) Retrieve(Instance) : 임의의 한 instance를 검색.
13) Update_Instance_Rid : 갱신하고자 하는 instance의 Rid.
14) Retrieve_Instance_Rid : 검색하고자 하는 instance의 Rid.
15) Pre_Instance_Rid : 이미 로크된 instance의 Rid.
16) Compare(Update_Instance_Rid, Pre_Instance_Rid) : 갱신하고자 하는 instance의 Rid와

이미 로크된 instance의 Rid의 충돌 관계를 비교.

17) Compare(Retrieve_Instance_Rid, Pre_Instance_Rid) : 검색하고자 하는 instance의 Rid와 이미 로크된 instance의 Rid의 충돌 관계를 비교.

18) Find(Name_var) : 임의의 한 instance가 속한 Name_var를 찾음.

19) Get(Waiting_Transaction) : 기다리고 있는 트랜잭션을 가져옴.

다음은 제안한 알고리즘이다.

1) 임의의 인스턴스 a를 접근하여 갱신하고자 할 때

```
Begin
  If Rid(a) Exists
  then Find(Root_object of a);
    if Lock_Status(Root_object of a) = '0' /*
    '0' means no locked */
    then while Find(Root_class of Root_object
    for a) do /* find Root_class of Root_
    object for a */
        begin Invokes Algorithm for Update_
        Scheduling Ts.
        end /* end of while */
    else /* Lock_Status(Root_object of a) =
    '1' : '1' means locked */
      Case Compare(Update_Instance_Rid, Pre
      _Instance_Rid) of
        1 : wait; /* 1 means occuring
        conflict */
        0 : while Find(Root_class of Root_
        object for a) do /* find Root_class of
        Root_object for a */
          begin
            Req_Lock(Root_class of a, IX);
            Req_Lock(Root_object of a, X);
            Req_Lock(Component_classes of
            Root_object for a, IX);
            Req_Lock(a, X);
            Update(a);
            Rel_Lock(Name_var, Lock_
            mode);
            Get(Wating_Transaction);
            Invokes Algorithm for Update_
            Scheduling Ts.
          end /* end of while */
    endif /* Lock_Status(Root_object of a) */
  else /* Rid(a) not Exists */
    Req_Lock(Component_classes of Root_
    object for a, IX);
    Req_Lock(a, X);
    Update(a);
    Rel_Lock(Name_var, Lock_mode) and
    Exit;
  endIf /* Rid(a) Exists */
End
```

```
Algorithm for Update_Scheduling Transactions
(AUST for short):
Begin
    Req_Lock(Root_class of a, IX);
    Req_Lock(Root_object of a, X);
    Req_Lock(Component_classes of Root_
    object for a, IX);
    Req_Lock(a, X);
    Update(a);
    Rel_Lock(Name_var, Lock_mode);
End
```

2) 임의의 인스턴스 a를 접근하여 검색하고자 할 때

```
Begin
  If Rid(a) Exists
  then Find(Root_object of a);
    if Lock_Status(Root_object of a) = '0' /*
    '0' is no locked */
    then while Find(Root_class of Root_object
    for a) do /* find Root_class of Root_
    object for a */
        begin
          Invokes Algorithm for Retrieval_
          Scheduling Ts.
        end /* end of while */
    else /* Lock_Status(Root_object of a) =
    '1' : '1' is locked */
      Case Compare(Retrieve_Instance_Rid, Pre_
      Instance_Rid) of
        1:wait;/* 1 means occuring conflict */
        0 : while Find(Root_class of Root_
        object for a) do /* find Root_class of
        Root_object for a */
          begin
            Req_Lock(Root_class of a, IS);
            Req_Lock(Root_object of a, S);
            Req_Lock(Component_classes of
            Root_object for a, IS);
            Req_Lock(a, S);
```

```
            Retrieve(a);
            Rel_Lock(Name_var,  Lock_
            mode);
            Get(Wating_Transaction);
            Invokes Algorithm for Retrieval_
            Scheduling Ts;
          end /* end of while */
    endif /* Lock_Status(Root_object of a) */
  else /* Rid(a) not Exists */
    Req_Lock(Component_classes  of  Root_
    object for a, IS);
    Req_Lock(a, S);
    Retrieve(a);
    Rel_Lock(Name_var, Lock_mode);
  endIf /* Rid(a) Exists */
End
```

Algorithm for Retrieval_Scheduling Transactions (ARST for short):

```
Begin
  Req_Lock(Root_class of a, IS);
  Req_Lock(Root_object of a, S);
  Req_Lock(Component_classes of Root_object
  for a, IS);
  Req_Lock(a, S);
  Retrieve(a);
  Rel_Lock(Name_var, Lock_mode);
End
```

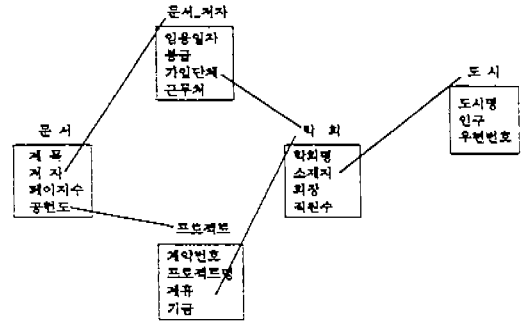
요구 모드

현재 모드	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	N
IX	Y	Y	N	N	N
S	Y	N	Y	N	N
SIX	Y	N	N	N	N
X	N	N	N	N	N

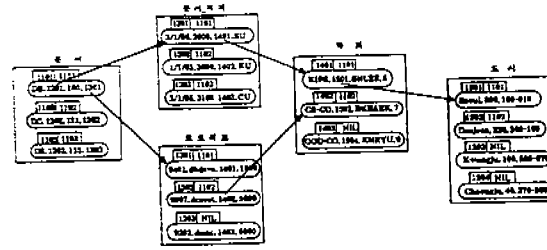
(Y : compatible N : incompatible)

(그림 4) 복합 객체 로킹의 양립성 행렬
(Fig. 4) Compatibility Matrix for Composite Object Locking

(그림 4)는 본 논문에서 이용할 양립성 행렬을 나타낸다[1]. (그림 4)의 로크 모드로 복합 객체들을 검색 및 갱신하기 위해서는 S나 X 모드로 로크되어지나 클래스는 5가지 모드중 하나의 로크 모드를 가질 수 있다. 클래스에 대한 IS 로크 모드는 그 클래스내의 어떤 인스턴스가 명백히 S 모드로 로크되어 있다는 의미이다. 클래스의 IX 로크 모드는 그 클래스내의 인스턴스들이 S나 X 모드로 로크되어 있다는 의미이다. 클래스의 S 로크 모드는 그 클래스내의 어떤 인스턴스들이 암시적으로 S 모드로 로크되어 있다는 의미이다. 클래스의 X 모드 로크는 그 클래스의 정의와 모든 인스턴스들이 검색 또는 갱신될 수 있다는 의미이다.



(그림 5) 복합 객체 계층
(Fig. 5) Composite Object Hierarchy



(그림 6) 복합 객체 계층의 인스턴스 예
(Fig. 6) Example instance of Composite Object Hierarchy

(그림 5)는 도서 관리 모델을 이용하여 객체지향 데이타베이스에서 복합 객체와 요소 클래스사이의 관계를 그래프로 표현한 것이다. 또한 이 그림은 객체지향 데이타베이스 시스템에 복합 객체를 이용하여 객체간의 종속 관계를 나타내는 복합 객체 계층이다.

(그림 6)은 (그림 5)의 복합 객체 계층의 인스턴스 예를 표현한 것이다. (그림 6)에서 하나의 트랜잭션 T1이 클래스 '학회'의 한 인스턴스 1401을 갱신한다면, 식별자를 이용한 로킹 프로토콜에 따라 클래스 '학회'내 인스턴스 1401의

Rid를 확인한다. Rid가 존재하면 상위 클래스 '문서'의 루트 객체인 1101로 가서 Rid를 확인한 후, 루트 클래스 '문서'가 로크되어 있지 않으므로 클래스를 갱신가능한 IX 모드로 로크한다. 그리고 루트 클래스 '문서'내의 루트 객체 1101에 X 모드로 로크, 그 요소 클래스 '문서_저자', '프로젝트' '학회' 그리고 '도시'에 IX 모드로 로크, 갱신할려고 하는 인스턴스 1401을 X 모드로 로크한 후, 인스턴스 1401을 갱신한다. 또하나의 트랜잭션 T2가 클래스 '학회'의 인스턴스 1402를 갱신한다고 하자. 접근하여 갱신하려고 하는 인스턴스 1402의 Rid는 T1이 접근하고 있는 인스턴스 1401의 Rid와는 다르므로, 루트 클래스 문서 에 IX 모드로 로크, 루트 객체 1102에 X 모드로 로크, 요소 클래스 '문서_저자', '프로젝트', '학회' 그리고 '도시'에 IX 모드로 로크, 갱신할려고 하는 인스턴스 1402에 X 모드로 로크하여 인스턴스 1402를 갱신할 수 있다. 그리고 또 하나의 트랜잭션 T3가 클래스 '학회'의 한 인스턴스 1403을 읽고자 할 때, 인스턴스 1403의 Rid를 확인해보니 Rid가 없으므로 트랜잭션 T1, T2가 갱신하고 있는 객체와는 무관하게 요소 클래스 '학회'에 IS 모드로 로크하고, 읽고자 하는 인스턴스 1403는 S 모드로 로크하여 관독할 수 있다.

이처럼 제안한 로킹 프로토콜은 기존의 복합 객체 로킹 방법에 비해 한 트랜잭션에의해 복합 객체의 요소 객체가 속한 클래스내에 있는 한 객체의 루트 객체가 로크되어 있더라도 다른 트랜잭션들은 로크되어 있지 않은 다른 루트 객체 식별자를 갖는 인스턴스를 접근하여 읽거나 갱신할 수 있으므로 동시성을 향상시킬 수 있다.

4. 결 론

데이타베이스 시스템은 데이타베이스내의 데이타를 동시에 여러 사용자들이 이용할 수 있도록 허용해야 한다. 따라서 여러 트랜잭션들이 동시에 실행될 때 서로간의 간섭을 받지 않고 트랜잭션들을 실행하여 데이타베이스의 일관성을 유지하도록 제어를 해주어야 한다.

기존 객체지향 시스템의 동시성 제어 기법을 복합 객체에 적용하면 복합 객체의 요소 객체가 속한 클래스까지 로크되어 여러 트랜잭션들이 동시에 그 클래스의 인스턴스에 접근이 제한되어 동시성이 낮아진다.

본 논문에서는 객체지향 데이타베이스 시스템에서 여러 트랜잭션들이 동시에 복합 객체를 접근하여 처리가 가능한 로킹 프로토콜을 제안하였다. 제안한 방법은 복합 객체의 요소 객체에 그 객체의 루트를 확인하도록 부여된 루트 객체 식별자를 이용한다. 하나의 트랜잭션이 복합 객체의 요소 객체가 가지고 있는 루트 객체 식별자를 이용하여 읽거나 갱신하고자 하는 객체에 대해 로크를 요구한다. 로크를 요구한 트랜잭션은 자신이 접근하고자 하는 복합 객체의 요소 객체가 루트 객체 식별자를 가지고 있고, 루트 객체가 로크되어 있는 지를 검사하여 하나의 트랜잭션에 의해 복합 객체의 루트 객체가 로크되어 있더라도 다른 트랜잭션들은 복합 객체의 요소 객체가 속한 클래스내의 다른 루트 객체 식별자를 갖는 인스턴스들을 접근하여 읽거나 갱신할 수 있게 된다. 이처럼 제안한 로킹 프로토콜은 여러 트랜잭션들이 동시에 복합 객체의 요소 객체가 속한 클래스에 접근하여 읽거나 갱신을 하도록 허용하기 때문에 동시성이 증가됨을 알 수 있다.

제안한 로킹 프로토콜의 효율성과 타당성을 보이기 위한 성능분석에 관한 연구를 진행하고 있으며, 각 객체마다 루트 객체 식별자를 부여함으로써 발생할 수 있는 과부하에 대한 연구가 필요하다.

참 고 문 헌

[1] J. Gray, Notes on Database Operating systems, IBM Research Lab., California, pp. 38-83, 1978.

[2] F. Bancilhon and S. Khoshafian,"A Calculus for Complex Objects," ACM SIGACT - SIGMOD Principles of

Database Systems, pp. 53-59, March, 1986.

[3] P. A. Bernstein, V. Hadzilacos and N. Goodman, Concurrency Control and Recovery in Database systems, Addison-wesley, MA, 1987.

[4] D. Fishman, et al., "IRIS: An object-oriented database system," ACM Transaction Office Information Systems, Vol. 5, No. 1, pp. 46-69, 1987.

[5] W. Kim, J. Banerjee, H. T. Chou, J. F. Garza and D. Woelk, "Composite Object Support in an Object-Oriented Database System," Proc. Object - Oriented Programming Systems, Languages and Applications, Vol. 22, No. 12, pp. 118-125, 1987.

[6] D. J. Penney and J. Stein, "Class Modification in the Gemstone Object-Oriented DBMS," Proc. Object-Oriented Programming Systems, Languages and Applications, Vol. 22, No. 12, pp. 121-127, 1987.

[7] J. F. Garza and W. Kim,"Transaction Management in an Object-Oriented Database Systems," Proc. ACM SIGMOD, pp. 37-45, 1988.

[8] W. Kim, E. Bertino, and J. F. Garza, "Composite Objects Revisted," Proc. ACM SIGMOD, pp. 337-347, 1989.

[9] O. Deux, et al., "The Story of O2," IEEE Transaction Knowlege Data Engineering, Vol. 2, No. 1, pp. 91-108, 1990.

[10] P. Butterworth, A. Otis and J. Stein, "The GEMSTONE Object Database Management System" Comm. ACM, Vol. 34, No. 10, pp. 64-77, 1991.

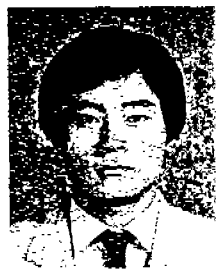
[11] J. G. Hughes, Object-Oriented DataBases, Prentice-Hall, NJ, pp. 179-196, 1991.

[12] W. Kim, Introduction to Object-Oriented Databases, Massachusetts Institute of Technology Press, Massachusetts, 1991.

[13] F. Velez, G. Bernard and V. Darnis, "The O2 Object Manager:An Overview," Building an Object-Oriented Database System, Morgan Kaufmann Pub., pp. 343-368, 1992.

황 부 현

1978년 숭실대학교 전산학과 졸업(공학사)
1980년 한국과학기술원 전산학과 졸업(공학석사)
1994년 한국과학기술원 전산학과 졸업(공학박사)
1980년~현재 전남대학교 전산학과 교수
관심분야 : 분산시스템, 분산 데이타베이스 보안, 객체지향 시스템



배 석 찬

1983년 전남대학교 전산통계학과 졸업(이학사)
1988년 전남대학교 대학원 전산통계학과 졸업(이학석사)
1992년 전남대학교 대학원 전산통계학과 박사과정 수료
1983년~1985년 R.O.T.C
1993년~현재 서남대학교 전산통계학과 전임강사
관심분야 : 데이타베이스, 객체지향 시스템, 전문가 시스템