ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

객체지향 데이터베이스의 다중계승을 위한 동시성 제어 기법 개발☆

Development of a Concurrency Control Technique for Multiple Inheritance in Object-Oriented Databases

전 우 천[1] 홍 석 기[2*]
Woochun Jun Suk-Ki Hong

요 약

인공지능과 웹 데이터베이스와 같은 분야에서는 기존의 관계 데이터 모형보다 더 고급 모델링 기능을 필요로 한다. 이러한 분야에서 객체지향데이터베이스는 객체를 모아 클래스를 제공하고 또한 클래스 사이에서 상위클래스는 하위 클래스에게 물려주는 계층구조를 제공하기 때문에 더 좋은 데이터 모형이 될 수 있다.

본 논문의 목적은 객체지향데이터베이스에서 다중 계승을 위한 동시성 제어 기법을 개발하는 것이다. 본 논문에서 제안하는 MIIL(Multiple Inheritance Implicit Locking) 기법은 기존의 Implicit 로킹(Locking) 기법에 기반을 두었다. MIIL 기법은 기존의Implicit 로킹에서 불필요한 로킹을 제거하였다. 또한 본 논문에서 제안하는 MIIL 기법에서의 Intention 로킹은 기존의 Implicit 로킹기법과 동일하게 작동한다. 본 논문에서 제안한 MIIL 기법은 기존의 Implicit 로킹 기법보다 로킹 오버헤드가 적음을 증명하였다. 또한, 본 논문에서는 단일 계승과 다중 계승 등 계승구조만을 이용함으로써 로킹 오버헤드를 줄이기 위한 추가적인 비용을 필요로 하지 않는다.

☞ 주제어: 객체지향 데이터베이스, 동시성 제어, 로킹 모형, 클래스 계층

ABSTRACT

Currently many non-traditional application areas such as artificial intelligence and web databases require advanced modeling power than the existing relational data model. In those application areas, object-oriented database (OODB) is better data model since an OODB can providemodeling power as grouping similar objects into class, and organizing all classes into a hierarchy where a subclass inherits all definitions from its superclasses.

The purpose of this paper is to develop an OODB concurrency control scheme dealing with multiple inheritance. The proposed scheme, called Multiple Inheritance Implicit Locking (MIIL), is based on so-called implicit locking. In the proposed scheme, we eliminate redundant locks that are necessary in the existing implicit locking scheme. Intention locks are required as the existing implicit locking scheme. In this paper, it is shown that MIIL has less locking overhead than implicit locking does. We use only OODB inheritance hierarchies, single inheritance and multiple inheritance so that no additional overhead is necessary for reducing locking overhead.

☞ Keywords: Object-oriented Database, Concurrency Control, Locking Model, Class Hierarchy

1. Introduction

Many new database applications such as computer-aided design (CAD), computer-aided software engineering (CASE), and office information systems have emerged. These new areas require advanced modeling capabilities to handle complex data and complex relationships among data. In those areas, complex modeling is impossible or very difficult, if the existing relational data model is adopted. An OODB is suitable for such applications, since it provides modeling power as grouping similar objects into class, and organizing all classes into a hierarchy where a subclass inherits all definitions from its superclasses.

In [11], an OODB is defined as "a collection of objects whose behavior and state, and the relationships are defined in accordance with an object-oriented data model". Also, an object-oriented database system (OODBS) is defined as "a

1 Dept. of Computer Education, Seoul National University of Education, Seoul,137-742, Korea.
2 Dept. of Business Administration, Dankook University, Jukjeon, Gyeonggido, 448-701, Korea
* Corresponding author (skhong017@dankook.ac.kr)
[Received 22 October 2013, Reviewed 24 October 2013, Accepted 2 December 2013]
☆ A preliminary version of this paper appeared in APIC-IST 2013, Aug 12-14, Jeju Island, Korea. This version is improved considerably from the previous version by including new results and features.

database system which allows the definition and manipulation of an OODB". The followings are basic concepts in OODBs [11].

- Object: any real world entity can be an object. Also, each object is associated with a unique identifier.
- Attribute: an object has one or more *attributes*whose values are also objects. The values of an attribute represent the state of an object.
- Method: an object has one or more *methods* which operates on the state of the object.
- Class: all objects sharing the same set of attributes and methods can be grouped into a class.

An object belongs to only one class as an instance of the class.

- Encapsulation: it is the process of packaging the data elements and functionality together. That is, the state of an object can be manipulated and read only by invoking the object's methods.
- Class hierarchy: the classes form a hierarchy which is directed-acyclic graph) called a *class hierarchy*. It is based on generalization and specialization concepts, which will be discussed later.

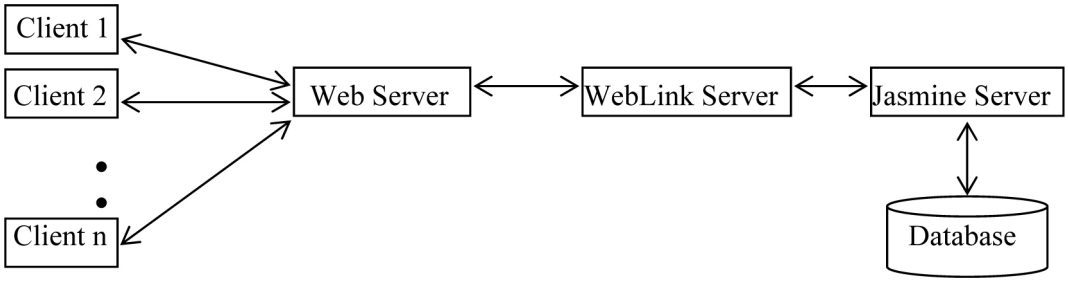
One of the major properties of OODBs is inheritance. That is, a subclass inherits the definitions defined on its superclasses. Also, there is an is-a relationship between a subclass and its superclasses. Thus, an instance of a subclass is a specialization of its superclasses (and conversely, an instance of a superclass is a generalization of its subclasses) [4,5,11]. There are two types of inheritance: single inheritance and multiple inheritance. In single inheritance, a class can inherit the class definition from one superclass. On the other hand, a class inherits the class definition from more than one class in multiple inheritance.

In general, there are two types of access to an object: instance access and class definition access [1]. Especially, there are two types of access on a class hierarchy: class definition write and IACH (Instance Access to Class Hierarchy) [3,11]. IACH represents an instance access to all or some instances of a given class and itssubclasses. A query is an example of IACH where a query is defined as instance reads to a given class and its subclasses [3]. Due to the inheritance rules, while a class and its instances are being accessed, the definitions of the class' superclasses should not be changed. Also, due to the is-a relationship between classes, the search space for a query against a class, says C, may include the instances of all classes on the class hierarchy rooted at C as well as all instances of C. For convenience, we call MCA (Multiple Class Access) for class definition write and IACHs, and SCA (Single Class Access) for other accesses such as class definition read and instance access to a single class.

For class hierarchy, there are two locking-based approaches: explicit locking [1,15] and implicit locking [3,11,13,14]. These two approaches have different philosophies dealing with a class hierarchy and will be discussed in Section 2. In this paper, we present a locking-based concurrency control scheme for OODBs that is based on implicit locking but incurs less overhead.

Recently Web database systems have become popular for many applications because of many advantages of the Web technology. For example, the multiple-platform issue becomes undisputable since web browsers are available on almost all platforms [10]. Also, Web's hypermedia-based model becomes familiar to most users. Given the advantages of the Web technology, the big concern is about the connectivity between a Web server and a database server. That is, how cana Web browser be used to access information stored and managed by commercial database systems? The common solution is to use Common Gateway Interface (CGI) programs. These programs become middle layer applications between the Web server and the database server. Also, the CGI becomes a standard for interfacing external applications with Web servers [10].

For example, Figure 1 shows the connectivity between the Web server and the Jasmine object-oriented database server. The primary job of WebLink is that it serves as an intermediate server between the Web server and the Jasmine database server. A client's requests for a new database connection are sent to the WebLink server via CGI programs. Then, the WebLink server opens a database session for that client. After the request for login or data access is processed, the client can access data from Jasmine by using subsequent HTML pages. When the client closes



(Figure 1) Architecture of the Jasmine Object-Oriented Database [10]

the connection, the database session opened for that client is closed.

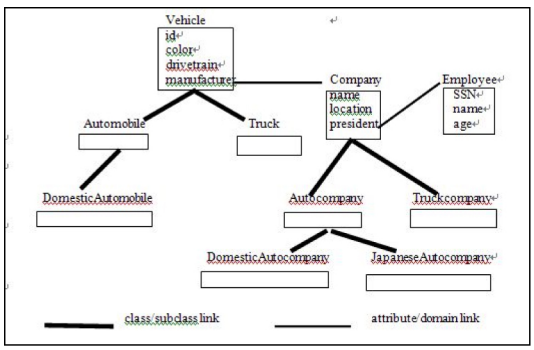
Many Web database systems are based on OODBs [4,18,19,20] since object models are suitable for representing complex multimedia data types in Web databases. Also, multiple inheritance is a fundamental concern in OODBs since new objects may be derived from existing objects in modular design. In Web database environments, transactions are naturally long, navigating objects from many classes, and must processed quickly online. Therefore, it is very important to have an efficient concurrency control scheme that allows many transactions, which access multiple classes with multiple inheritance to be processed at the same time. The concurrency control scheme should also meet the response time requirements for a large number of users connected through the Internet. Our aim is to develop a concurrency control scheme that can be used for Web applications to meet such requirements.

This paper is organized as follows. In Section 2, two existing schemes, implicit locking and explicit locking, are discussed. In Section 3, a new scheme, called scheme, is proposed. In Section 4, the correctness of the MIIL scheme is proved. Finally, conclusions and future work are given in Section 5.

2. Related Works

These existing works deal with three features of access: *conflicts among methods*, *class hierarchy locking*, and *nested method invocations*.

In order to illustrate each type of access, consider the following Figure 2. Assume that class *vehicle* has four attributes *id*, *color*, *drivetrain* and *manufacturer* and class *company* has three attributes *name*, *location*, and *president*. Class *employee* has three attributes *ssn*, *name*, *age*.



(Figure 2) An OODB Schema

2.1. Conflicts Among Methods

In general, there are two types of access to an object : *instance access* and *class definition access* [1]. An instance access consists of consultations and modifications of attribute values in an instance or a set of instances. A class definition access includes consulting class definition, adding/deleting an attribute or a method, changing the implementation code of a method or changing the inheritance relationship between classes, etc. In Figure 2, for class *vehicle*, a possible instance access is a modification of the attribute *color* of an instance, and a possible class definition access is changing domain of the attribute *id* from integer to character.

In OODBs, oneof the main concerns is to increase concurrency among methods so that more transactions can

run in parallel. Otherwise, aborting or blocking a transaction to meet database consistency may waste system resources or delay other transactions. Commutativity is a widely used criterion to determine whether a method can run concurrently with those in progress on the same object [14]. Two methods commute if their execution orders do not affect the results of the methods. Two methods conflict with each other if they do not commute.

Two types of access to an object induce three different types of conflicts among accesses to a class: *conflicts between instance accesses*, *conflicts between class definition accesses*, and *conflict between instance access and class definition access*. For example, a conflict between instance accesses occurs if two instance methods are trying to modify an attribute value of the same instance at the same time. Also, updating the same class definition such as modifying the implementation code of the same instance method at the same time induces conflict between class definition accesses.

2.2. Nested Method Invocation

In OODBs, objects can have nested structures. That is, an object can be composed of complex objects or atomic objects. For example, in Figure 2, an object *vehicle* can consist of three atomic objects (i.e., *id*, *color*, and *drivetrain*) and a complex object *manufacturer*. It is natural that, in OODBs, each class can define its own method and a method on a class can invoke another method on its subobject (also called *nested method invocation*) [16].

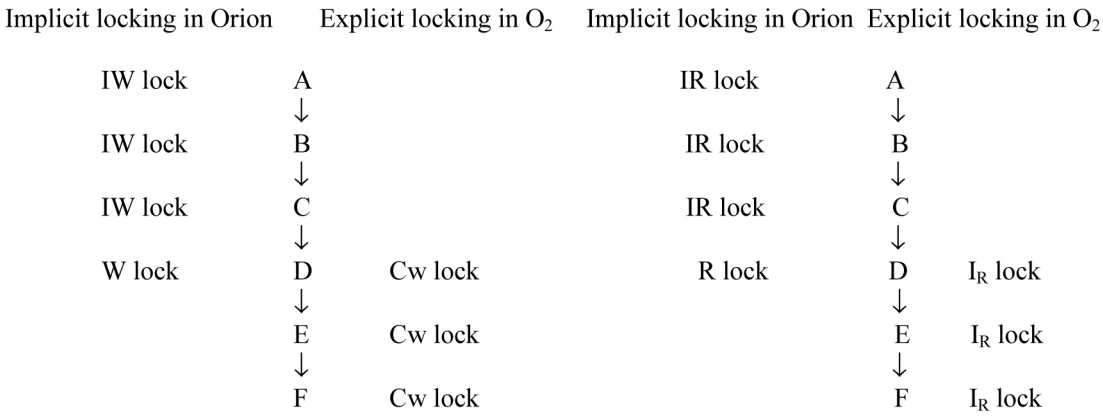
In OODBs, two different objects can share a common object in anunderlying hierarchy [6]. We call the common object a *referentially shared object* (RSO). Once again, in Figure 1, two different instance objects *vehicles* may share the same instance object *company* in an underlying nested object hierarchy. Thus, methods on different objects may not commute [16]. The RSO (also called non-disjoint complex object) is a fundamental concern of OODB since new objects may be composed of existing objects in modular design as indicated in [17]. Thus, a nested object hierarchy may result in referential sharing.

Existing works have many disadvantages as follows. For*conflicts among methods*, application programmers have a burden to provide commutativity relationships for instance access. That is, in order to provide better concurrency among methods, application programmers should know possible states of objects and results of each method. Also, for class definition access, existing works either provide less concurrency due to big locking granularity or incur too much run-time overhead for higher concurrency. For *class hierarchy locking*, existing studies, which can be classified into two types (i.e., *explicit locking* and *implicit locking*), incur too much locking overhead and aim at a special type of access to class hierarchy (i.e., *explicit locking* aims at access to a higher-level class of the hierarchy while *implicit locking* aims at access to a class near the leaf-level). For *nested method invocations*, either concurrency is still limited since semantic information is not utilized or too much run-time overhead is incurred since locks are required for each atomic operation. Also, most existing studies do not consider referentially shared objects (non-disjoint complex objects) which is a necessary condition for modular design in an OODB [17].

2.3. Class Hierarchy Locking

In explicit locking, for an MCA access such as class definition write, a lock is set not only the class, say C, but also on each subclass of C on the class hierarchy. For an SCA access such as class definition read, a lock is set for only the class to be accessed (also called target class). Thus, for an MCA access, transaction accessing a class near the leaf level of a class hierarchy will require fewer locks than transactions accessing a class near the root of a class hierarchy. But, it increases the number of locks required by transactions accessing a class at a higher level in the class hierarchy.

In implicit locking, setting a lock on a class C requires extra locking on a path from C to its root as well as on C. Intention locks [12] are set on all ancestors of a class before the target class is locked. An intention lock on a class indicatesthat some lock is held on a subclass of the class. That is, the purpose of intention locks is to detect the possible conflicts earlier. For an MCA access on a target



(a) Locking for class definition write in Orion and O_2 (b) Locking for query in Orion and O_2

(Figure 3)

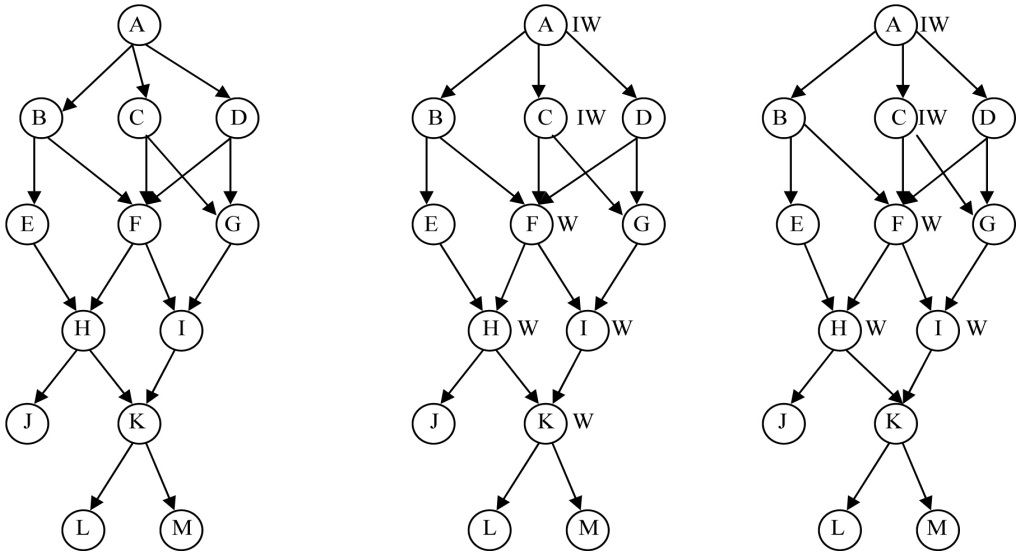
class, locks are not required for every subclass of the target class. It is sufficient toput a lock on only the target class (in single inheritance) or locks on the target class and subclasses of the target class, which have more than one superclass (in multiple inheritance). Thus, it can reduce lock overhead over explicit locking. But, implicit locking requires a higher cost when a target class is near the leaf level in the class hierarchy due to intention lock overhead.

For example, consider the class hierarchy shown in Figure 3. Note that two OODBs, Orion [3] and O2 [1], are chosen for the illustration of two existing works. In order to update the class definition in class, say D, each scheme works as in Figure 3.a. For implicit locking, intention locks IWs corresponding to W (Write) locks are required for all superclasses on the path from D to the root A. Thus, if another transaction, say T1, needs to update the class definition in class A, it does not have to search through each class in the class hierarchy because of the help of the intention lock IW on class A. That is, since IW and W conflict with each other, T1's lock request is blocked on class A. On the other hand, an explicit locking does not require any intention locks. But, it does require a Cw (Class Write) lock on each subclass of the target class through the class hierarchy since any modification of the class definitions in D may affect the definitions of its subclasses. Also, locking for a query on D (assuming that the query needs access to all instances of D, E and F) can be done as in Figure 3 .b.

3. The Multiple Inheritance Implicit Locking (MIIL) Scheme

Our proposed scheme called MIIL is based on implicit locking. The reason wechoose to base the proposed technique on implicit locking, but not on explicit locking, is as follows. In this work, our concern is to reduce locking overhead for an MCA access on a class, say C, and all of its subclasses. In explicit locking, as discussed in Section 2, a lock is set not only on the class C, but also on each subclass of C on the class hierarchy. On the other hand, in implicit locking, locks are required on the class C and subclasses of the class C that have more than one superclass. Since the implicit locking incurs fewer locks than explicit locking, our concern is to reduce the locking overhead in implicit locking.

For SCA access, the MIIL scheme works the same way as the implicit locking scheme does. The difference is to deal with MCAaccess in multiple inheritance. Assume that a target class C needs an MCA lock. In the MIIL scheme, as in the implicit locking, each superclass along any superclass chain of C needs an intention lock. The difference is as follows. In implicit locking, foran MCA access, a lock is required for all subclasses of the target class C, which have more than one superclass. But, in the MIIL scheme, lock is



a. A Class hierarchy

b. Locks required by implicit locking

c. Locks required by the MIIL scheme

(Figure 4)

required for all subclasses of the target class C, which have more than one superclass and are directly reachable from classes other than the target class C and subclasses of C. That is, the MIIL scheme requires fewer locks than the implicit locking scheme.

Consider the class hierarchy shown in Figure 4.a. Assume that a class definition needs to be changed in class F. Also, assume that one of the superclass chains from F is arbitrarily chosen so that classes A and C need intention locks. The implicit locking scheme adopted in Orion [3,8] needs to get locks as in Figure 4.b. On the other hand, locks are required as in Figure 4.c if the MIIL scheme is applied. Note that both schemes necessitate IW mode locks on one of the superclass chains from F. In the MIIL scheme, only classes H and I need to be locked since they can be reached directly from classes E and G, respectively. Note that two classes E and G do not belong to the class hierarchy rooted at F.

4. Correctness of the MIIL Scheme

In this section, we show that the MIIL scheme performs better than the implicit locking scheme.

Based on the discussion in Section 3, since the MIIL scheme incurs fewer or equal number of locks compared with implicit locking for any kinds of accesses, it is sufficient to show that the MIIL scheme is correct, that is, it satisfies serializability [2]. More specifically, we prove that, for any requester, any conflict with a lock holder is always detected. With this proof, since the MIIL scheme is based on two-phase locking, it is guaranteed that the MIIL scheme satisfies serializability [2].

Claim: The MIIL scheme detects any conflicts between a lock requester and a lock holder.

Proof:

Assume that a class hierarchy has multiple inheritance. If not, the MIIL scheme works the same way as the implicit locking scheme does.

Assume that a class C has subclasses with more than one superclass and is locked in MCA mode by a lock holder.

Assume that a lock requester needs to access class K where C=K. Note that, if C=K, the conflict will always be detected on C. Without loss of generality, there are two cases as follows.

Case a) A lock requester needs an SCA access

If K is a superclass of C, there is no conflict. Also, if there is neither superclass nor subclass relationship between C and K, there is no conflict. Assume that K is subclass of C. In this case, if C is on the same path on which K sets intention locks, conflict is detected on C. If not, K must get through one of subclasses of C, say, Ci, which has more than one superclass. Otherwise, K would not be a subclass of C. Thus, conflict will be detected on Ci.

Case b) A lock requester needs an MCA access

If K is neither a superclass nor a subclass of C, there are two cases as follows. If there is no common subclass between C and K through the subclass chains of C and K,there is no conflict. Otherwise, the conflict is detected on the first common subclass through the subclass chain of both C and K based on the MIIL scheme.

Assume that K is a superclass of C. Then, if K is on the same path with C, conflict is detected on K. If not, C must get through one of subclasses of K, say Ki, which has more than one superclass. Otherwise, K would not be a superclass of C. Thus, conflict is detected on Ki. On the other hand, assume that K is a subclass of C.

If C is on the same path on which K sets intention locks, conflict will be detected on C. If not, K must get through one of subclasses of C, say Cj, which has more than one superclass. Otherwise, K would not be a subclass of C. That is, conflict is detected on Cj.

From case a) and b), we can conclude that, for any lock requester, it is guaranteed that its conflicts with a lock holder are always detected. Since the MIIL scheme is based on two-phase locking, serializability is guaranteed. In turn, this means that the MIIL scheme performs better than the implicit locking scheme.

5. Conclusions and Further Works

Many new database applications such as CAD, CASE, office automation systems, and artificial intelligence have emerged. These new areas require advanced modeling capabilities to handle complex data and complex relationships among data. In those areas, complex modeling is impossible or very difficult, if the existing relational data model is adopted. An object-oriented database is suitable for such applications, since it provides modeling power as grouping similar objects into class, and organizing all classes into a hierarchy where a subclass inherits all definitions from its superclasses.

In this paper, we presented a locking-based concurrency control scheme for OODBs called MIIL. The MIIL scheme is based on the implicit locking scheme but incurs less locking overhead for the case of multiple class access with multiple inheritances. We proved theoretically that the MIIL scheme is correct and has less locking overhead than implicit locking does. While some concurrency control techniques require some additional overhead such as access frequency information on classes and instances [7, 8, 9], the proposed techniques does not need any kinds of overhead to reduce locking overhead. Our techniques require only class hierarchy structure, single inheritance hierarchy and multiple inheritance hierarchy.

Recently database systems have been used to manage data for many Web applications. Many of these Web database systems are based on OODBs since OODBs provide advanced modeling power for representing complex multimedia data types in Web databases. Multiple inheritances are a natural property in OODBs since new objects may be derived from existing objects in modular design. In Web databaseenvironments, transactions are usually navigating objects from many classes that may relate to each other through inheritance. To guarantee database correctness while processing transactions concurrently through the Internet, an efficient concurrency control scheme which reduces locking overhead for MCA access type with

multiple inheritances is needed. The proposed scheme, MIIL, is developed to fulfill this need.

Currently we are planning to do a simulation work in order to compare the proposed work with the implicit locking scheme. Although the proposed incurs less locking overhead over the existing the implicit locking theoretically, we are interested in how the proposed scheme is really working in the real OODB transaction processing environments.

The possibledrawback of the MIIL scheme is that it requires a higher locking overhead when a target class is near the leaf level in the class hierarchy due to the intention lock overhead. Thus, we are also developing a new scheme, which is based on both implicit locking and explicit locking, in order to reduce the locking overhead for all kinds of accesses. In this case, we may need additional information or overhead in order to achieve less locking overhead.

References

[1] M. Cart and J. Ferrie, Integrating Concurrency Control into an Object-Oriented Database System, *2nd Int. Conf. on Extending Data Base Technology*, Venice, Italy, pp. 363-377, 1990.

[2] K. Eswaran, J. Gray, R. Lorie, and I. Traiger, The notion of consistency and predicate locks in a database system, *Communication of ACM*, Vol. 19, No. 11, pp. 624-633, 1976.

[3] J. Garza and W. Kim, Transaction Management in an Object-Oriented Database Systems, *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 37-45, 1988.

[4] V. Geetha and N. Sreenath, Semantic Concurrency Control on Continuously Evolving OODBMS Using Access Control Lists, *9th International Conference on Distributed Computing and Internet Technology*, Bhubaneswar, India, pp. 523-534, 2013.

[5] V. Geetha, Semantic Based Concurrency Control in OODBMS, *2011 International Conference on Recent Trends in Information Technology*, Chennai, India, pp. 1313-1318, 2011.

[6] U. Herrmann, P. Dadam, K. Kuspert, E. Roman, and G. Schlageter, A Lock Technique for Disjoint and Non-disjoint Complex Objects, *Proceedings of 2nd International Conference on Extending Data Base Technology*, Venice, Italy, pp. 219-237, 1990.

[7] W. Jun, A Multi-granularity Locking-based Concurrency Control in Object-oriented Database Systems, *Journal of Systems and Software*, Vol. 54, No. 3, pp. 201-217, 2000.

[8] W. Jun and L. Gruenwald, An Optimal Locking Scheme in Object-oriented Database Systems, *In Proceeding of Web-Age Information Management*, pp. 95-105, 2000.

[9] W. Jun, Controlling Concurrent Accesses in Multimedia Database Systems, *MDIC 2001*, pp. 67-76, 2001.

[10] S. Khoshafian, S. Dasananda and S. Minassian, The Jasmine Object Database: Multimedia Applications on the Web, *Morgan Kaufmann Publishers*, San Francisco, California, USA, 1999.

[11] W. Kim, Introduction to Object-Oriented Databases, *The MIT Press*, Cambridge, MA, USA,1990.

[12] H. Korth and A. Silberschartz, Database System Concepts, 2nd Edition, *McGraw Hill*, New York, NY, USA,1991.

[13] S. Lee and R. Liou, A Multi-Granularity Locking Model for Concurrency Control in Object-Oriented Database Systems, *IEEE Trans. on Knowledge and Data Engineering*, Vol. 8, No. 1, pp. 144-156, 1996.

[14] C. Malta and J. Martinez, Controlling Concurrent Accesses in an Object-Oriented Environment, 2nd Int. Symposium on Database Systems for Advanced Applications, Tokyo, Japan, pp. 192-200, 1992

[15] C. Malta and J. Martinez,Automating Fine Concurrency Control in Object-Oriented Databases, *9th IEEE Conf. on Data Engineering*, Vienna, Austria, pp. 253-260, 1993.

[16] P. Muth, T. Rakow, G. Weikum, P. Brossler, and C. Hasse, Semantic Concurrency Control in Object-Oriented Database Systems, Proceedings of the 9th IEEE International Conference on Data Engineering, pp. 233-242, 1993.

[17] R. Resende, D. Agrawal, and A. Abbadi, Semantic Locking in Object-Oriented Database Systems, *Proceedings of OOPSLA 94*, Portland, Oregon, USA, pp. 388-402, 1994.
[18] Objectware database (http://www. objectwareinc.com) (2013).
[19] Ontos database (http://www.ontos.com) (2013). Objectivity database (http://www. objectivity. com) (2013).

◐ 저 자 소 개 ◑

전 우 천
1985년 서강대학교 졸업
1987년 서강대학교 대학원 졸업 (석사)
1997년 Univ. of Oklahoma졸업 (박사)
1998년~현재 서울교육대학교 컴퓨터교육과 교수
관심분야 : 장애인 정보화 교육, 정보 통신 윤리, 정보영재
E-mail: wocjun@snue.ac.kr

홍 석 기
1997년 University of Nebraska-Lincoln (경영학박사)
1997년~2003년 건국대학교 경영학과 교수
2003년~현재 단국대학교(죽전) 경영학과 교수
관심분야 : 생산시스템, e-Business, SCM
E-mail : skhong017@dankook.ac.kr