SECURE TRANSACTION MANAGEMENT AND QUERY PROCESSING IN MULTILEVEL SECURE DATABASE SYSTEMS

Brajendra Panda
The College of West Virginia

William Perrizo and Ramzi Haraty
North Dakota State University

Key words
Multilevel Secure Database Systems, Data and User Classifications, Concurrency Control, Querry Processing.

Abstract

In a multilevel secure database system, every data item is assigned a classification level and each user that accesses the data has a clearance level. Users can read data items that exist at a lower level and write at their own level. In such systems multilevel databases may be partitioned and stored as single-level databases. To construct a multilevel relation, repeated joins of different single level base relations are taken, thus resulting in delayed query response time. However, trying to accelerate transactions might establish covert channels which can send high level sensitive information to low level users. This paper describes secure algorithms for both concurrency control and query processing in such systems and shows how these two techniques could be integrated together to give best performance. The data structures, needed in these algorithms, are based on bit vector techniques developed in [13], and [14]. Our method accelerates both read-only (queries) and read-write transactions in a secure and correct manner.

Introduction

Many military and industrial applications require the use of multilevel secure database systems (MLS/DBSs) where every user has a clearance level and every data item has a classification level. In the kernelized architecture, as described in [1], data are separated and stored in different containers according to classification level. Thus, when a user at a higher clearance level needs to read low level data, he/she must access different containers. This results in a delayed query response time. So query acceleration for high level users, in order to make a fast decision, is quite essential. But without a proper concurrency control technique this method would establish covert channels that could transmit sensitive information downward violating the system's security policy [9]. Thus, along with the query acceleration technique, a supportive concurrency control mechanism is absolutely necessary in such secure systems. Again, integrating both methods needs to be done in a secure fashion to avoid any possibility of illegal information flow. Since this area is fairly new, no research has been done in developing an integrated concurrency control and query acceleration mechanism. A number of published papers about concurrency control algorithms are: ([6], [7], [8], [12], and [17]) and about query acceleration algorithms are: ([16], and [15]). In this paper, we have developed a concurrency control technique based on the ROLL object [13] and then integrated this method with the query acceleration technique based on the secure DVA method [15].

In the multilevel secure database environment, a relation, R, is represented by the schema $R(A_1,C_1,\ldots,A_n,C_n,TC)$, where C_i is the classification of the attribute A_i. The domain of C_i is the range of classifications for data that can be associated with attribute A_i and the domain range$(A_i) = [L_i,H_i]$ which is the sub lattice of the lattice of access classes. The TC attribute in the schema represents the Tuple Class of the record, which is the least upper bound of the attribute classifications in the tuple. Figure 1 illustrates a multilevel relation MISSILE with three attributes: name, range, and speed of different missiles.

Name	Range	Speed	TC
MT1 U	350 U	800 C	C
NT5 U	450 U	800 U	U
NT5 U	480 C	750 C	C
DNT U	400 U	800 U	U
DNT U	450 C	800 U	C
KR1 U	500 U	700 U	U
FD7 C	450 C	850 C	C
KR1 C	400 C	null C	C

Figure 1: A multilevel relation MISSILE.

Our database model follows the SeaView model [11] that was developed in a joint effort by SRI International and Gemini Computers. This model is a research prototype based on the Kernelized approach, and uses element level classification of data. In this model, the multilevel relations are partitioned both vertically and horizontally into single-level base relations and then are stored separately. For the decomposition and recovery algorithms of multilevel relations in the SeaView, readers are referred to ([5], [10]). Figure 2 below shows the base relations obtained after decomposing the MISSILE relation given in Figure 1.

$\text{MISSILE}_{name,u}$
MT1
NT5
DNT
KR1

$\text{MISSILE}_{range,u,u}$	
MT1	350
KR1	500
NT5	450
DNT	400

$\text{MISSILE}_{speed,u,u}$	
NT5	800
DNT	800
KR1	700

$\text{MISSILE}_{range,u,c}$	
DNT	450
NT5	480

$\text{MISSILE}_{speed,u,c}$	
NT5	750
MT1	800

$\text{MISSILE}_{name,c}$
FD7
KR1

$\text{MISSILE}_{range,c,c}$	
FD7	450
KR1	400

$\text{MISSILE}_{speed,c,c}$	
FD7	850

Figure 2: The base relations for the MISSILE relation.

The query acceleration mechanism used in this paper is based on the DVA technique that was initially introduced in [14] and modified in [15] to fit in the multilevel secure database system environment. For the rest of this paper, we denote their technique as MLS-DVA. As described in [15], the acceleration of queries in the MLS-DVA results from the speed of operation on bit vectors and the restrictions of I/O to only those pages that contain the tuples that participate in the final result. In their work, the authors have shown that the MLS-DVA approach achieves a significant performance improvement when the multilevel query involves selection on one or more attributes of the multilevel relation. Unlike the SeaView recovery mechanism, this approach generates no spurious tuples. The concurrency control algorithm, based on the ROLL model [13], is designed in such a manner that it uses minimal extra structure needed beyond what is described in [15]. This method is correct, secure, free from livelocks and also free from deadlocks. As for the security model, we assume the well-known Bell-LaPadula model [2].

The MLS-DVA Method

The MLS-DVA algorithm requires that for each base relation a bit vector, called a Domain Vector (or DV) must be maintained. A DV helps in determining the presence or absence of a value in a relation's joining attribute (i.e., the primary key attribute, in this situation, for each base relation) by the presence or absence of a 1-bit in the corresponding position in the vector. The correspondence between a value and its position in the DV (denoted by value identifier or vid), at each level, is provided by the primary key relation and the relative record numbers of its tuples. Thus, the number of bits in the DVs, at a given level, is the same as the number of records in the primary key relation that exists at that level. The reader should note that the DVs for the primary key relations will consist of all ones and hence need not be maintained.

A second structure Domain Value Index (DVI) might be necessary for each base relation, if the base relations are not indexed on the primary key attribute. A DVI provides the mapping between a vid and the address of the tuple containing the corresponding domain value. Figure 3.1 and 3.2 show the data structures for the base relations given in Figure 2.

$\text{DV.MISSILE}_{range,u,u} = 1111$ $\text{DV.MISSILE}_{speed,u,u} = 0111$
$\text{DV.MISSILE}_{range,u,c} = 0110$ $\text{DV.MISSILE}_{speed,u,c} = 1100$
$\text{DV.MISSILE}_{range,c,c} = 11$ $\text{DV.MISSILE}_{speed,c,c} = 10$

Figure 3.1: The DVs for the base relations

$\text{MISSILE}_{name,u}$	
vid	rec#
1	1
2	2
3	3
4	4

$\text{MISSILE}_{range,u,u}$	
vid	rec#
1	1
2	3
3	4
4	2

$\text{MISSILE}_{speed,u,u}$	
vid	rec#
2	1
3	2
4	3

$\text{MISSILE}_{range,u,c}$	
vid	rec#
2	2
3	1

$\text{MISSILE}_{range,u,c}$	
vid	rec#
1	2
2	1

$\text{MISSILE}_{name,c}$	
vid	rec#
1	1
2	2

MISSILErange,c,c	
vid	rec#
1	1
2	2

$\text{MISSILE}_{speed,c,c}$	
vid	rec#
1	1

Figure 3.2: The DVIs for the base relations.

The MLS-DVA algorithm is as follows:
First, a Query Vector at each level, x, (denoted by QV_X) is constructed in which the number of bits is the same as the number of entries in the primary key relation at level x. A bit is set to 1 in this vector at position i, if the corresponding vid at relative record number i in the key relation participates in the query. However, if the query does not involve any selection at level x, the QV_X would be entirely 1's. Otherwise, the base relations having the participating attributes are read at each level, x, and the selected vid positions in QV_X are set to ones. If there is more than one attribute involved in the selection criteria, then the smallest participating base relation is read. The selected vids are dropped in the index of the next smallest participating base relation to avoid reading non-participating pages, and then the selection criteria are applied to further reduce the number of vids. By continuing this process for all participating base relations, a fully reduced list is obtained and then the query vector is built.

As mentioned in [15], the polyinstantiated elements create spurious tuples during the outer join process. Polyinstantiation is necessary in order to avoid covert channels and element polyinstantiation means more records than one exist with the same primary key value but have different non-key values at different levels. In order to avoid the generation of spurious tuples, polyinstantiated elements need to be processed in a different manner. The attribute relations that are investigated for this purpose are the ones having the attributes required for the output of the query result only. At each level x, a Polyinstantiated Domain Vector, $PDV_{x,y}$ is created in the following way for each level y such that $x \leq y$.

Let $PDV'_{Ai,x,y}$ be the vector obtained by logically ORing the domain vectors: $DV.R_{Ai,x,z}$ where $x \leq z < y$ and A_i is the attribute required in the output. Next, $PDV_{Ai,x,y}$ is constructed by logically ANDing each $PDV'_{Ai,x,y}$ with the corresponding $DV.R_{Ai,x,y}$. The positions of 1 bits in this vector denotes the positions of those vids at level x, that have at least one polyinstantiated element for the particular attribute A_i up to level y. Then, PDVx,y is constructed by ORing all $PDV_{Ai,x,y}$s, which represents the vids having polyinstantiated elements in their records that are visible to users up to level y.

Next, for each level x, the vids that do not participate in the query and/or do not have any polyinstantiated elements visible up to level y are filtered out. The vector that represents such information is obtained by logically ANDing QV_x with $PDV_{x,y}$, and is denoted by $PQV_{x,y}$. The reader should note that QV_x represents the key values at level x that participate in the query irrespective of the fact whether or not their corresponding records have any polyinstantiated elements.

Before carrying on any build or probe phase of joins, a table, called Select Omit Table (denoted by SOT_x at each primary key level x) is built. The number of columns in each SOT is the same as the number of attributes needed in the output, and the number of rows is the exact number of records that would appear in the output. Each element in such a table at level x consists of two components: the first gives the address of a tuple and the second represents the attribute classification y of the base relation, $R_{Ai,x,y}$, where the tuple appears.

To construct SOT_x, QV_x is taken first and scanned to find the position of one-bits in it. The position of such a bit indicates that the corresponding key position would appear in the result. To find out the record number of such a key value in attribute relation A_i that is required in the output, the corresponding position in the domain vector, $DV.R_{Ai,x,x}$, is searched. If the bit is one, then the vid is dropped in the DVI of $R_{Ai,x,x}$ to get the record position and the (record address, x) pair is entered in SOT_x under the column A_i. Otherwise, $DV.R_{Ai,x,z}$, where z is the next higher level of x in the security lattice, is checked. The search continues up to level

b. If a one bit is detected, the corresponding index is checked, and the (record address, z) pair is entered in SOT_x under A_i column. Next $PQV_{x,y}$ is scanned for the presence of 1 bits. But this time the search starts from the DV of base relation $R_{Ai,x,y}$ and, if not found, continues downward in the security lattice until the DV of $R_{Ai,x,x}$ is searched.

After constructing SOT_x for a given x, the element values in each column are sorted in ascending order. This helps minimize the number of page reads from the disk [14]. Then the records are retrieved from the base relations in the following way. If the element (n,z) appears under column A_i, the record with address n is retrieved from the relation $R_{Ai,x,z}$. Then the build and probe phases of join are performed. For a better clarification of the algorithm, its security, and performance analysis, the reader is referred to [15].

Concurrency Control Issues

Our concurrency control model is based on the Request Order Linked List (ROLL) object [13]. This non-blocking protocol eliminates deadlocks, livelocks, and also the need for a critical section "scheduler" module in the system. A ROLL object is a linked list of bit vectors that uses three simple operations, POST, CHECK, and RELEASE, which can be performed on that list by individual transaction managers.

The basic ROLL method uses a bit vector, known as Request Vector (RV), for each active transaction to indicate requests for access to data items by that transaction. For each data item, there are two bits, one read bit and one write bit, representing the read and write request by the transaction to that item. A bit-value 1 means the transaction requests access to the item corresponding to that bit position, while bit-value 0 means that it does not.

After creating a RV as described above, a transaction POSTs its RV into the tail of the ROLL. This POST operation establishes the serialization partial order of transactions and is the only operation that must be atomic. The CHECK operation allows a transaction to determine the availability of data items. A transaction CHECKs by performing the logical OR of all RVs ahead of its own RV in the list. In the resulting vector, a 1 means the item corresponding to that position is unavailable and 0 means it is available. CHECK can be repeated at any time to determine which needed items have become available since the last CHECK operation. To RELEASE a data item after processing it, the transaction simply flips the corresponding bit from a 1 to a 0. The next transaction POSTed for that item will then find the item available upon performing the CHECK operation.

Next we describe a modified ROLL model to adapt to the multilevel secure database systems environment.

The CC Algorithm

In this model, the ROLL object consists of two layers. The first is the Inter-container ROLL (IROLL) layer, and the second is the container ROLL (denoted by ROLL for the rest of the paper) layer, one at each level. The IROLL is responsible for maintaining consistency of transaction POSTings across containers, and a ROLL at each container helps in achieving serializability among all transactions accessing that container.

When a high transaction requests a read access to a low data item, it must not set any lock on it; otherwise, covert channels could be established. On the other hand, having such a transaction wait creates unavailability problems. To eliminate these two problems, multiversioning of data is utilized. In our model, each write on a data item x produces a new version which follows the POST order of the transaction that wrote x. In this paper, the term "version" refers to a value written by a committed transaction.

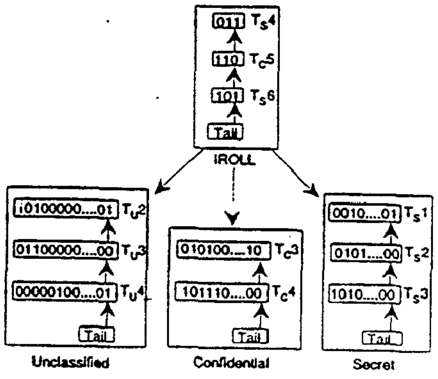

Figure 4: The ROLLs at different containers and the IROLL

Figure 4 shows an instance of ROLL structures at three different levels and the IROLL structure. In this figure $T_x n$ denotes a transaction with security level x and post order n.
In the kernelized architecture, all write operations are "local" and some read operations are "global" (across security levels); hence we call a transaction, local or global depending on the types of operations the transaction requests. The concurrency control algorithm is presented below. The proof of the algorithm is omitted due to space constraints. Interested readers can refer to [18] for the proof.

1. When a local transaction enters the system for execution, its request vector (RV) will be POSTed in the ROLL object that exists at the same level of the transaction.
2. When a global transaction enters the system for execution, a Global Request Vector (GRV) is created in which 1 bits are set for each local container where access is needed. Then the GRV is POSTed in the IROLL.
3. The IROLL is CHECKed periodically to see if any of the containers requested by any of the transactions are available.
   3.1. If a container requested by a transaction, T_i, is available at a lower level, then the Transaction Manager running on behalf of T_i, TM_i is asked to get the entry point to the ROLL.
   3.2. If a container requested by T_i is available at the same level of T_i, then TM_i is asked to POST its RV in the ROLL.
4. When a transaction, T_i, is POSTed locally in a ROLL or gets an entry point to the ROLL, its corresponding bit is RELEASEd in the IROLL. The process continues until all the bits in the GRV of T_i are RELEASEd.
5. When a transaction T_i, wants to read a data item x such that $SL(x) < SL(T_i)$, then TM_i CHECKs the ROLL at SL(x) from its entry point onwards for any conflicts. SL(a) denotes the security level of item a.
   5.1. If a conflict exists, then TM_i waits to reCHECK later.
   5.2. If there is no conflict, then T_i reads the highest version of x that is less than or equal to its entry point.
6. When T_i wants to read or write x such that $SL(x) = SL(T_i)$, then TM_i CHECKs for any conflicting operations on the same data item, x, that are ahead of it in the ROLL.
   6.1. If a conflict exists, then TM_i waits to reCHECK later.
   6.2. If there is no conflict and T_i has a read request, then it reads the highest version of x that is less than or equal to its POST order.
   6.3. If there is no conflict and T_i has a write request, then it produces a new version (same as its POST order) of x.
7. Transactions RELEASE their bits in the RV during their commit time only.

Subsetting ROLL

To obtain maximum concurrency possible in each ROLL at the container level we need to have read and write bits for each attribute of each tuple of every base relation. But this unnecessarily increases the length or the RVs in the ROLL, although most of the transactions would need very few read/write bits as 1's and the rest of the bits as zeros. To eliminate this problem, we suggest subdividing the ROLL at each container into several sub-ROLLs.

In order to maintain execution consistency among transactions, one container ROLL is required, denoted by CROLL, in which two bits (one read bit and one write bit) are set for each base relation. Thus, the number of bits in a CROLL will be twice the number of base relations at that level. Then there needs to be a relation ROLL, denoted by RROLL, for each base relation, which has twice as many bits as the number of records in the corresponding primary key relation. Every global transaction (after obtaining a POST signal from IROLL) and every local transaction must POST its RV that represents the base relations needs to be accessed in CROLL. Performing a CHECK in CROLL, it determines which of the base relations are available for POSTing. If all the tuples in a base relation have to be read (or modified), the reading (or modification) could be done without RELEASing the read (or write) bit from the CROLL and there is no need for POSTing a RV in the corresponding RROLL. Otherwise, a RV representing the records to be accessed, must be POSTed in the RROLL for that relation. After a successful POSTing, the corresponding bit in the CROLL must be RELEASEd.

When a transaction at a higher level wants to get an entry point to ROLL at lower level, as described in the previous section, it directly does so at the required RROLLs bypassing the CROLL. In RROLL, transactions perform CHECK and RELEASE operations as mentioned in the previous algorithm. The bits in these ROLLs are RELEASEd during the commit time only. A new version of a data item bears the post order of the writing transaction at the RROLL.

Different RROLLs and the CROLL needed in confidential level for the relations given in figure 2 are shown in figure 5.

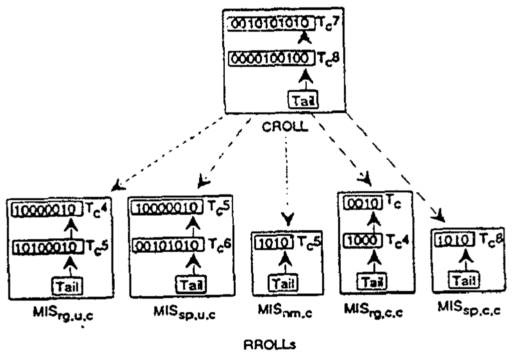

Figure 5: The RROLLs and the CROLL at confidential level

Query Processing

This section describes the process to be followed in order to complete a query. When a subject submits a query that would access different security levels, a GRV needs to be constructed and POSTed in the IROLL and upon availability of low level containers, an entry point to each required RROLL is made. Every local transaction and every global transaction upon indication from the IROLL must POST in the CROLL at its own level in the following way. First, a RV is constructed by determining which attributes of the multilevel relation need to be accessed and then the RV is POSTed in the CROLL. Next, if the query requires a selection, then a CHECK on the CROLL determines whether the base relation(s) to be read for the selection criteria is(are) available or not. If not, the CHECK is done periodically until so determined. Then the entire base relation is read, and a QV is constructed as described in section 2.2. At any lower level the QV is constructed by reading the correct version of the base relation (see section 3.1). Upon completion of the read, the corresponding bit is RELEASEd in the RV of the CROLL. Next, the QV so obtained is modified as follows. If a write access is required on any of the base relations, after every bit in the QV, a zero bit, as the read bit, is added. In case of a read access, a zero bit is added, as a write bit, after every bit present in the QV. This modified QV is then POSTed in the required RROLLs whenever a CHECK operation indicates the availability of the RROLLs. Then, for every such POSTing the bit in the CROLL is RELEASEd.

Using the DVA algorithm, the required pages are brought into main memory and the read/write operations are performed. The bits in the RROLLs are RELEASEd during the commit time.

An Example

Let us consider a query in the MISSILE relation given in Figure 1, by a user having clearance up to confidential level:

SELECT * FROM MISSILE WHERE range $\geq$ 400

Since this is a global query, a GRV = 1100 (assuming there are four classification levels) is constructed with the 1-bits for accessing unclassified and confidential containers. Then, it is POSTed in the IROLL. After performing a CHECK on the IROLL, if it is determined that the unclassified container is available for POSTing, the Transaction Manager is informed. Then the transaction manager obtains an entry point to the RROLLs of the relations $MISSILE_{range,u,u}$ and $MISSILE_{speed,u,u}$. If the local container is available, the RV = 0010101010 is created. This is because there are five base relations at the confidential level and all of them except for the $MISSILE_{name,c}$ must be read. We assume that in every pair of bits the first bit represents a read request and the second represents a write. This RV is indicated in figure 5 as T_c7 in CROLL.

If the correct version is available, the $MISSILE_{range,u,u}$ is read at the unclassified level and by applying the selection criteria, the QV_u' is built as 0111. By performing a CHECK in the CROLL at confidential level if it is found that $MISSILE_{range,u,c}$ is available for reading, the entire relation is read without having to POST an RV in the RROLL of $MISSILE_{range,u,c}$. The bit in CROLL is RELEASEd. Then applying the selection criteria QV_u'' = 0011 is constructed. By logically ORing QV_u' and QV_u'', QV_u = 0111 is obtained. Since a read access is required on the base relations, an extra zero bit for every bit is added as write bit to the QV_u to give the modified QV_u = 00101010. Similarly, upon availability, by reading the relation $MISSILE_{range,c,c}$ QV_c = 11 is built. Adding extra write bits the modified QV_c = 1010 is obtained. After reading $MISSILE_{range,u,c}$ and $MISSILE_{range,c,c}$, the bits in the RV on CROLL are RELEASEd. Then after performing CHECKs on the CROLL if determined that the base relation $MISSILE_{speed,u,c}$ is available, the QV_u is POSTed in the corresponding RROLL and the bit in CROLL is RELEASEd. Similarly, QV_c is POSTed in the RROLL of $MISSILE_{speed,c,c}$ and the corresponding bit in the CROLL is RELEASEd. These QVs are denoted in figure 5 by T_c6 and T_c8 in the RROLLs $MISSILE_{speed,u,c}$ and $MISSILE_{speed,c,c}$ respectively. Next, CHECK on the

RROLLs are performed and then by applying the DVA algorithm given in section 2.2 the SOT_c and a part of SOT_u are built. To complete SOT_u, correct versions (as determined by the concurrency control algorithm described in section 3.1) of the base relations at the unclassified level are read in a similar way. A detailed walk through of the DVA algorithm on the same example and the result of the query are given in [15] and will not be discussed here due to space constraints.

Conclusion

Database researchers have constantly expressed concern about the performance of multilevel database management systems based on the kernelized architecture. One of the several reasons for this is that, satisfying multilevel queries involves taking repeated joins of base relations, and joins are always expensive operations. The performance degrades when a high level transaction waits to obtain a read access of a low data because of the concurrency control protocol that is designed to reduce the risk of covert channels. In this paper, we have introduced a correct and secure concurrency control technique that accelerates joins in kernelized multilevel secure database systems by making the use of the DVA technique given in [15]. Our method is correct, deadlock free, livelock free, and secure. It establishes no downward information flow by any direct or indirect means that violate the system's security policy. As a future research objective, we would like to investigate the area of multilevel secure object oriented databases.

References

[1] "Multilevel Data Management Security", Committee on Multilevel Data Management Security, Air Force Studies Board, National Research Council, Washington D.C., 1983.

[2] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation", The Mitre Corporation, March 1976.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency Control and Recovery in Database Systems", Addison-Wesley, Massachusetts, 1987.

[4] Dorothy E. Denning, "Cryptography and Data Security", Addison-Wesley, Reading, Massachusetts, 1982.

[5] Dorothy E. Denning and Teresa F. Lunt, "A Multilevel Relational Data Model", Proc. of the IEEE Symposium on Security and Privacy, p 220- 234, Oakland, CA, April 1987.

[6] S. Jajodia and V. Atluri, "Alternative Correctness Criteria for Concurrent Execution of Transactions in Multilevel Secure Databases", Proc. of the IEEE Symposium on Security and Privacy, p. 216-224, Oakland, CA, May 1992.

[7] T. F. Keefe and W. T. Tsai, "Multiversion Concurrency Control for Multilevel Secure Database Systems", Proc. of the IEEE Symposium on Security and Privacy, p. 360-368, Oakland, CA, May 1990.

[8] B. Kogan and S. Jajodia, "Secure Concurrency Control", Proc. of the 3rd RADC Workshop in Multilevel Database Security, Castille, NY, June 1990

[9] B. W. Lampson, "A Note on the Confinement Problem", CACM, (16)10 p 613-615, October 1973.

[10] T. F. Lunt, R. R. Schell, W. R. Shockley, and D. Warren,"Toward a Multilevel Relational Data Language", Proc. of the IEEE Symposium on Research in Security and Privacy, p. 72-79. 1988.

[11] Teresa F. Lunt et al., "The SeaView Security Model", IEEE Transactions on Software. Engineering, Vol. 16, No. 6, June 1990.

[12] W. T. Maimone and I. B. Greenberg, "Single-level Multiversion Schedulers for Multilevel Secure Database Systems", Proc. of the 6th Annual Computer Security Applications Conf., p. 137-147, Tucson, AZ, Dec. 1990.

[13] W. Perrizo, "Request Order Linked List (ROLL): A Concurrency Control Object for Centralized and Distributed Database Systems", Proc. of the 7th International Conference on Data Engineering, p. 278-285, Kobe, Japan, April 1991.

[14] W. Perrizo, J. Gustafson, D. Thureen, D. Wenberg, and W. Davidson, "Domain Vector Accelerator (DVA): A Query Accelerator for Relational Operations", Proc. of the 7th International conference on Data Engineering, Kobe, Japan, 1991.

[15] W. Perrizo and B. Panda, "Query Acceleration in Multilevel Secure Database Systems", Proc. of the 16th National Computer Security Conference, Baltimore, MD, September 1993.

[16] B. Thuraisingham, W. T. Tsai, and T. F. Keefe, "Secure Query Processing Using AI Techniques", Proc. of the 21st Hawai International Conference on System Sciences, IEEE Computer Society Press, 1988.

[17] R. Haraty, "Transaction Management in Multilevel Secure Database Systems", Ph.D. Dissertation, North Dakota State University, Fargo, ND, December 1992.

[18] B. Panda, "Query Processing in Multilevel Secure Database Systems", Ph.D. Dissertation, North Dakota State University, Fargo, ND, December 1993.