A Bit Vectors Algorithm for Accelerating Queries in Multilevel Secure Databases

Ramzi A. Haraty and Imad Rahhal
Lebanese American University
P.O. Box 13-5053 Chouran
Beirut, Lebanon 1102 2801
Email: rharaty@lau.edu.lb

Abstract

In this paper, we present an algorithm called the Bit Vectors Algorithm or BVA based on bit vectors to accelerate queries in multilevel secure database systems. This algorithm recovers query output from single-level relations in a faster and more space-efficient manner than most previous work performed on this subject. In addition, the BVA algorithm does not produce spurious or extra tuples.

1. Introduction

Multilevel secure databases are becoming more prevalent. Multilevel databases partition multilevel relations into single-level relations and store each one separately. This results in poor performance since multilevel queries would results in applying repeated joins to single-level relations, which is rather expensive. Moreover, previous work [3][4][7][8] has shown that is it prone to spurious tuples, which is rather a serious problem in the world of databases. This work proposes an algorithm – BVA – that is faster and more space efficient than previous methods.

This paper is organized as follows. Section 1.1 starts with a description of some access control mechanisms used to maintain data confidentiality in secure database systems. Then, the concept of polyinstantiation and its effects on multilevel recovery algorithms is described, followed by an explanation of multilevel relations. In section 2.0 an overview of the previous work on this subject is provided to give way to the BVA algorithm, which is the core of this paper. Section 3 presents the BVA algorithm and its data structures. Section 4 contains the conclusion comparing the BVA algorithm to related work, namely, the Sea View Model and the DVA algorithm.

1.1 Security Policies

Many types of security policies or access control mechanisms have been integrated into database systems to provide data security. Two well-known access control mechanisms are the MAC and DAC mechanisms [2]. The DAC mechanism, or Discretionary Access Control mechanism, is usually implemented in most commercial products. In this mechanism, the owner of an object (i.e., file, directory, etc.) decides who can access his/her object and in what manner (i.e., read-only, write-only, etc.). The Mandatory Access Control or MAC mechanism, developed by Bell and LaPadula [1], does not leave protection decisions of objects to the discretion of the owners. The system enforces the protection decisions. This mechanism defines a database by its subjects and objects. A subject is an active entity such as a process and an object is a passive entity such as a data item or table. Subjects have clearance levels and objects are assigned sensitivity levels (i.e., Top Secret, Secret, Confidential and Unclassified). In order for a certain subject to access an object, one of following two conditions must be satisfied (depending on the type of access):

1- *The Simple Security Policy*: A subject *X* can read access object *Y*, if *X*'s clearance level dominates (is greater than or equal to) *Y*'s sensitivity level.
2- *The *-Policy*: A subject *X* can write access object *Y*, if *X*'s clearance level is dominated by (is less than or equal to) *Y*'s sensitivity level.

In short, the MAC policy states that reads should propagate downwards and writes should propagate upwards. All security authentication functions are stored in a TCB (Trusted Computing Base) away from the DBMS. When a request for object *X* by subject *Y* is issued, it is first

authenticated in the TCB. If *Y* can access *X* then the request is forwarded to the DBMS; if not, then the request is rejected.

1.2 Polyinstantiation

Due to security and access control mechanisms, some tuples in multilevel relations may be polyinstantiated. A polyinstantiated tuple is a tuple that exists more than once in a relation with the same apparent key (refer to section 1.3) but with some other attribute values being changed. This is due to the fact that different subjects are authorized to update or view different data. For example, suppose that a subject *X* with clearance *C* (Classified) is attempting to write a new value to a data item *Y* with sensitivity *S* (Secret). The old value in item *Y* is not viewable by subject *X* but the new value written into *Y* by *X* is viewable (it has *X*'s clearance level which *C*). To preserve the old value of *Y*, a new tuple is inserted into the relation with same apparent key (and same attribute values except for *Y* in this case). Now *X* can view the new value inserted into *Y* and Secret and Top Secret users will view the two values (the old value with sensitivity *S* and the new value with sensitivity *C*).

1.3 Multilevel Relations

A multilevel relation is a relation of the form R (*A1*, *C1*, ..., *An*, *Cn*, *TC*) where *Ai* is an attribute and *Ci* is its classification (or sensitivity level). *TC* is the classification of the tuple. *Ci* belongs to the domain of classifications for data items. We denote *A1* to be the apparent key of R.

The concept of a key is a little bit different in multilevel relations because keys can be duplicated; this is why we refer to them as *apparent keys* instead of just *keys*. The reason behind this duplication of keys is polyinstantiation.

2. Related Work

The following are two approaches for maintaining and recovering multilevel relations from single-level relations.

2.1 The Sea View Model

The Sea View model [5] is considered as one of the most important moves towards multilevel secure relations. It consists of two algorithms: a decomposition algorithm and a recovery algorithm. The decomposition algorithm divides a multilevel relation R into a set of single level relations. The multilevel relation exists only at logical level; single level relations are stored physically. For every query, an output multilevel relation is reconstructed from the single-level relations using the recovery algorithm. Unfortunately, the recovery algorithm of the Sea View model suffers from the following:

1- Creation of spurious tuples in the output (due to polyinstantiation),
2- Space inefficiency due to temporary tables, and
3- Time inefficiency due to unions and joins, which are two of the most expensive database operations.

2.2 The DVA Algorithm

The DVA algorithm [6], in short, is an algorithm motivated by the recovery algorithm of the Sea View Model based on domain vector accelerators, DVAs, to accelerate the recovery of multilevel relations from single-level relations. DVAs accelerate joins between relations and thus lead to reducing the response time of queries requiring many joins. The DVA algorithm solves the problems of the Sea View Model recovery algorithm; it doesn't create spurious tuples in the output table and is space and time efficient. It shows significant improvement especially in environments where queries involve selections on some (one or more) attributes of the multilevel relations. In spite of these facts, this algorithm uses a lot of temporary data structures, some of which can be omitted to improve the algorithm's space efficiency without negatively affecting its overall performance or its functionality.

3. The BVA Algorithm

Perhaps the best way to describe how the algorithm works is by going through an example. In this algorithm, we will assume that a multilevel relation is decomposed into single level relations using the decomposition algorithm of the Sea View model. Now, the BVA algorithm will be applied to recover the multilevel relation from those single level relations. Suppose, we have the following multilevel relation representing all missiles deployed in Iraq:

R = Iraqi Missiles								
***Name**		**Developed by (devby)**		**Length (M)**		**Range (KM)**		**TC**
AS30L	*U*	France	*U*	3.65	*U*	10	*U*	***U***
AS-9 Kyle	*U*	Russia	*U*	6	*U*	90	*C*	***C***
Al Hussein	*U*	Iraq	*U*	12.2	*C*	650	*C*	***C***
Aspide	*U*	Italy	*C*	3.7	*C*	35	*C*	***C***
Roland1/2/3	*C*	France	*C*	2.4	*C*	6.3	*C*	***C***
Roland1/2/3	*C*	Germany	*S*	NULL	*S*	8	*S*	***S***

Figure 1: The multilevel relation

This multilevel relation is based on single-level relations as shown in figure 2.

Rname,*u*
AS30L AS-9 Kyle Al Hussein Aspide

Rname,*c*
Roland1/2/3

Rdevby,*u*,*u*	
AS30L AS-9 Kyle Al Hussein	France Russia Iraq

Rdevby,*u*,*c*	
Aspide	Italy

Rdevby,*c*,*c*	
Roland 1/2/3	France

Rdevby,*c*,*s*	
Roland 1/2/3	Germany

Rlength,*u*,*u*	
AS30L AS-9 Kyle	3.65 6

Rlength,*u*,*c*	
Al Hussein Aspide	12.2 3.7

Rlength,*c*,*c*	
Roland 1/2/3	2.4

Rrange,*u*,*u*	
AS30L	10

Range,*u*,*c*	
AS-9 Kyle Al Hussein Aspide	90 650 35

Rrange,*c*,*c*	
Roland 1/2/3	6.3

Range,*c*,*s*	
Roland 1/2/3	8

Figure 2: The decomposed single-level relations

All relations containing only the key - i.e., Rname,*u* (base single level relation containing all keys at classification level *u*) and Rname,*c* - are referred to as base relations. Suppose we want to recover the output of the query *"Select name, devby, length from R where range<>35"*. The following is a description of the steps of the BVA associated with examples from the above the relation (*Ai* denotes any attribute and *A1* denotes the apparent key).

1- For every relation R*Ai*,*x*,*y* (single level relation containing all entries form multilevel relation having keys at classification level *x* and *Ai* attribute values at classification level *y*), excluding base relations, create a bit vector BV.R*Ai*,*x*,*y* denoting the presence or absence of the keys at level *x*. BV.R*Ai*,*x*,*y* should have the same number entries as the number keys at level *x* (can be found in relation Rkey,*x*).

Keys at level *u* = {AS30L, AS-9 Kyle, Al Hussein, Aspide}
Keys a level *c* = {Roland 1/2/3}

BV.Rdevby,*u*,*u* = 1110
BV.Rdevby,*u*,*c* = 0001
BV.Rdevby,*c*,*c* = 1
BV.Rdevby,*c*,*s* = 1

BV.Rlength,*u*,*u* = 1100
BV.Rlength,*u*,*c* = 0011
BV.Rlength,*c*,*c* = 1

BV.Rrange,*u*,*u* = 1000
BV.Rrange,*u*,*c* = 0111
BV.Rrange,*c*,*c* = 1
BV.Rrange,*c*,*s* = 1

2- For every R*Ai*,*x*,*y* excluding base relations create a Mapping Vector Index, MVI.R*Ai*,*x*,*y*, mapping the position of the keys in R*Ai*,*x*,*y* to the position of the keys in R*A1*,*x*. The entries in the MVIs are of the form (*pib, pit*) where *pib*, Position In Base relation, is the position of the key in the base relation; and *pit*, Position In This relation, is the position of the key in this relation (see figure 3).

MV.Rdevby,*u*,*u*	
pib	Pit
1 2 3	1 2 3

MV.Rdevby,*u*,*c*	
Pib	Pit
4	1

MV.Rdevby,*c*,*c*	
pib	Pit
1	1

MV.Rdevby,*c*,*s*	
Pib	Pit
1	1

MV.Rlength,*u*,*u*	
pib	Pit
1 2	1 2

MV.Rlength,*u*,*c*	
pib	Pit
3 4	1 2

MV.Rlength,*c*,*c*	
pib	Pit
1	1

MV.Rrange,*u*,*u*	
pib	Pit
1	1

MV.Rrange,*u*,*c*	
pib	Pit
2 3 4	1 2 3

MV.Rrange,*c*,*c*	
pib	Pit
1	1

MV.Rrange,*c*,*s*	
Pib	pit
1	1

Figure 3: The mapping vector indices of the single-level relations

3- Create an Output Keys Vector OKV*x* (contains all keys with classification *x* that will appear in the output table) for all *x*:

   a. As having a number of bits equal to the number of entries in Rkey,*x*.
   b. Read all relations having an attribute participating in the selection criteria of the query (Rrange,*u*,*u* and Rrange,*u*,*c* at level *u* and Rrange,*c*,*c* and Rrange,*c*,*s* at level *c*).
   c. Get all entries from those relations satisfying the selection criteria at each level *x* (at level *u* we have AS30L, AS-9 Kyle, and Al Hussein; and at level *c* we have Roland1/2/3).
   d. For every entry that succeeds, set its position in OKV*x* to 1. The position of an entry in OKV*x* can be found by matching the key of this entry to the key in Rkey,*x* and getting its position in Rkey,*x*.

      OKVu = 1110 (the first three keys in Rname,*u* will appear in output table)
      OKVc = 1 (the first and only key in Rname,*u* will appear in output table)

4- Create a Polyinstantiated Keys Vector, PKV*x*,*y* that contains all polyinstantiated keys at level x by subjects at level y, for all *x*,*y* such that *x*<*y*. For all attributes *Ai* requested in the output of the query (name, devby and length) except for the key (name) do the following:

   a. Create PKV*Ai*,*x*,*y* = the ORing of all BV.R*Ai*,*x*,*z* where *x*<= *z*<*y* for all *z*

PkVdevby,*u*,*c* = BV.Rdevby,*u*,*u* = 1110
PkVdevby,*u*,*s* = BV.Rdevby,*u*,*u* OR BV.Rdevby,*u*,*c* = 1110 OR 0001 = 1111
PkVdevby,*c*,*s* = BV.Rdevby,*c*,*c* = 1
PkVlength,*u*,*c* = BV.Rlength,*u*,*u* = 1100
PkVlength,*u*,*s* = BV.Rlength,*u*,*u* OR BV.Rlength,*u*,*c* = 1100 OR 0011 = 1111
PkVlength,*c*,*s* = BV.Rlength,*c*,*c* = 1

b. Set PKV*Ai*,*x*,*y* = PKV*Ai*,*x*,*y* AND BV.R*Ai*,*x*,*y*

PkVdevby,*u*,*c* = PkVdevby,*u*,*c* AND BV.Rdevby,*u*,*c* = 1110 AND 0001 = 0000
PkVdevby,*u*,*s* = PkVdevby,*u*,*s* AND BV.Rdevby,*u*,*s* = 1111 AND 0000 = 0000
PkVdevby,*c*,*s* = PkVdevby,*c*,*s* AND BV.Rdevby,*c*,*s* = 1 AND 1 = 1

PkVlength,*u*,*c* = PkVlength,*u*,*c* AND BV.Rlength,*u*,*c* = 1100 AND 0011 = 0000
PkVlength,*u*,*s* = PkVlength,*u*,*s* AND BV.Rlength,*u*,*s* = 1111 AND 0000 = 0000
PkVlength,*c*,*s* = PkVlength,*c*,*s* AND BV.Rlength,*c*,*s* = 1 AND 0 = 0

c. Create PKV*x*,*y* as the ORing of all PKV*Ai*,*x*,*y*

PKVu,*c* = PKVdevby,*u*,*c* OR PKVlength,*u*,*c* = 0000 OR 0000 = 0000
PKVu,*s* = PKVdevby,*u*,*s* OR PKVlength,*u*,*s* = 0000 OR 0000 = 0000
PKVc,*s* = PKVdevby,*c*,*s* OR PKVlength,*c*,*s* = 1 OR 0 = 1

A 1-bit in position n in any vector PKV*x*,*y* signifies that the nth entry in Rkey,*x* is polyinstantiated . Therefore, entry 1 in Rname,*c* that is *Roland1/2/3* is polyinstantiated.

5- Create POKV*x*,*y* (polyinstantiated output keys vector) as the ANDing of PKV*x*,*y* and OKV*x*

POKVu,*c* = OKVu AND PKVu,*c* = 1110 AND 0000 = 0000
POKVu,*s* = OKVu AND PKVu,*s* = 1110 AND 0000 = 0000
POKVc,*s* = OKVc AND PKVc,*s* = 1 AND 1 = 1

6- Create an Output table, as shown in figure 4, as follows:

a. Having a number of columns equal to the number of fields, *Ai*, requested in the output of the query (name, devby and length)➔3 columns.
b. Scan OKV*x* for 1-bit entries (appears in the output). If a 1 bit appears in position n do the following for all *Ai* attributes requested in the output:

i. If *Ai* is the key (i = 1) then get the nth record from R*A1*,*x* and store it under *A1* column in output table. This entry has classification *x*.
ii. Else, go to the nth entry in BV.R*Ai*,*x*,*z* where *z* = *x* initially . If a 1 bit is found in position n then get the *pit* value of the entry in MVI.R*Ai*,*x*,*z* that has *pib* = n. Let p = *pit,* get the p'th entry from R*Ai*,*x*,*z* and store it under *Ai* column in output table. This entry has classification *z*. Else (if 1 bit is not found in position n of BV.R*Ai*,*x*,*z* then) increment *z* to the next higher level and repeats this step.

c. Scan POKV*x*,*y* for 1-bit entries (appears in the output and polyinstantiated). If a 1 bit appears in position n do the following for all *Ai* attributes requested in the output:

i. If *Ai* is the key (*Ai* = *A1*) then get the nth record from R*A1*,*x* and store it under *A1* column in output table. This entry has classification *x*.

ii. Else, go to the nth entry in BV.R*Ai,x,z* where *z* = *y* initially . If a 1 bit is found in position n then get the *pit* value of the entry in MVI.R*Ai,x,z* that has *pib* = n. Let p = *pit,* get the p'th entry from R*Ai,x,z* and store it under *Ai* column in output table. This entry has classification *z*. Else (if 1 bit is not found in position n of BV.R*Ai,x,z* then) decrement *z* to the next lower level and repeats this step.

*Name		Developed by(devby)		Length (M)	
AS30L	*U*	France	*U*	3.65	*U*
AS-9 Kyle	*U*	Russia	*U*	6	*U*
Al Hussein	*U*	Iraq	*U*	12.2	*C*
Roland1/2/3	*C*	France	*C*	2.4	*C*
Roland1/2/3	*C*	Germany	*S*	2.4	*C*

Figure 4: The output table of the given query

4. Conclusion

The BVA algorithm is an enhancement over the previous algorithms. Compared to the recovery algorithm of the Sea View model, which was one of the earliest and most important attempts towards multilevel database security, the BVA has the following advantages:

1. No spurious tuples in the output table due of polyinstantiation.
2. No time inefficiency because the BVA algorithm does not depend on the use of joins and unions like the Sea View model algorithm to create the output table.

The BVA algorithm has the advantages of the DVA algorithm over the recovery of the Sea View model. In addition, it has some advantages over the DVA algorithm itself. Those advantages mainly stem from the fact that the BVA reduces the temporary storage used to create the output table. The BVA algorithm eliminates the following data structures that are used by the DVA algorithm:

1. The Domain Vector Tables (referred to as DVTs in the DVA algorithm) used to map keys to their positions in the primary relations.
2. The Domain Value Indices (referred to as DVIs in the DVA algorithm) of primary relations.
3. The Domain Vectors (referred to as DVs in the DVA algorithm) of primary relations.
4. The participation of primary relations in the creation of the Polyinstantiated Domain Vectors (referred to as PDVs in the DVA algorithm).
5. Select Omit Tables (referred to as SOTs in the DVA algorithm), which are used to produce the Output table.

References

[1] Bell, D. and LaPadula, L. "Secure Computer Systems: Unified Exposition and Multics Interpretation". *Technical Report*. The Mitre Corporation. 1976.

[2] Jackson, J. "MAC & DAC Brief". www.garrison.com/html/docmacdac.html. 2001.

[3] Jajodia, S. and Sandhu, R. "Toward a Multilevel Secure Relational Data Model". *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Denver, Colorado, May 29-31, 1991, pages 50-59.

[4] Jajodia, S. and Sandhu, R. "A Novel Decomposition of Multilevel Relations into Single-Level Relations". *Proceedings of IEEE Symposium on Security and Privacy*. Oakland, California, May 20-22, 1993, pages 300-313.

[5] Lunt, T., Denning, D., Schell, R., Hechman, M., and Shockley, W. "The Sea View Security

Model". *IEEE Transactions on Software Engineering*, Volume 16, Number 6, June 1990.

[6] Perrizo, W. and Panda, B. "Query Acceleration in Multilevel Secure Distributed Database Systems". *Proceedings of the 16th National Computer Security Conference*. Baltimore, Maryland, September 20-23, 1993.

[7] Sandhu, R. and Jajodia, S. "Restricted Polyinstantiation or How to Close Signaling Channels Without Duplicity". *Proceedings of the third RADC Workshop on Multilevel Database Security*. Castile, New York, June 1990, pages 7-12.

[8] Sandhu, R. "Design and Implementation of Multilevel Databases". *Proceedings of the 6th RADC Workshop on Multilevel Database Security*. Southwest Harbor, Maine, June 1994.