Multi-level Security in Database Management Systems

Patricia A. Dwyer, George D. Jelatis and Bhavani M. Thuraisingham
Honeywell Computer Sciences Center, 1000 Boone Avenue North, Golden Valley, Minnesota 55427, USA

Multi-level secure database management system (MLS-DBMS) security requirements are defined in terms of the view of the database presented to users with different authorizations. These security requirements are intended to be consistent with DoD secure computing system requirements. An informal security policy for a multi-level secure database management system is outlined, and mechanisms are introduced that support the policy. Security constraints are the mechanism for defining classification rules, and query modification is the mechanism for implementing the classification policy. These mechanisms ensure that responses to users' queries can be assigned classifications which will make them observable to the querying users.

Keywords: Database security, Database management systems, Security policy, Security constraints, Query modification.

Patricia A. Dwyer is a principal research scientist at the Honeywell Computer Sciences Center. Her research interests include distributed systems, database systems, and security. She is currently investigating secure database management system issues, and is participating in the design, development, and application of Honeywell's Distributed Database Testbed System (DDTS). She has also investigated algorithms for transaction and query optimization using the testbed system. She previously worked at Philip Morris where she investigated the use of computer modeling and simulation in cigarette product design and development. Dwyer received the B.S. degree in Chemistry from Northern Illinois University in 1977, the M.S. degree in Chemistry from the University of Minnesota in 1980, and the M.S. degree in Computer Science from the University of Minnesota in 1981. She is a member of the ACM and the IEEE.

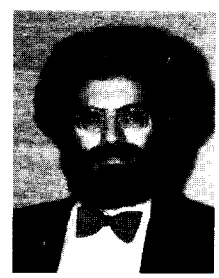
This effort has been supported by U.S. Government Contract F30602-86-C-0003.

North-Holland
Computers & Security 6 (1987) 252–260

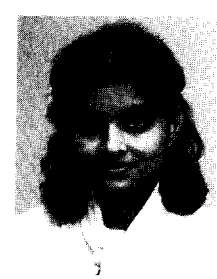
1. MLS-DBMS Security Requirements

A multi-level secure database management system (MLS-DBMS) is different from a conventional DBMS in at least three ways: (1) every data item in the database has associated with it one of several classifications or sensitivities, that may change dynamically; (2) control of users' access to data must be based upon these classifications; and (3) the classification based access controls cannot be avoided or subverted, that is, they are mandatory.

In such a multi-level secure database system, the critical factor which distinguishes one user



George D. Jelatis is a senior principal research scientist at the Honeywell Computer Sciences Center. He is currently investigating secure database management system issues. He has also investigated local area network issues ranging from protocol design, validation and evaluation, to communication architectures and application and use of LANs in distributed computing environments. He has also done research in distributed computing and participated in the design and implementation of a distributed systems testbed. He previously worked at University of Minnesota Hospitals where he was involved in computerized Electrocardiography and computer networking. He earned a BS in Physics from the University of Minnesota and is currently pursuing an MS in Computer Science. He is a member, and past chairman of the IEEE P802.4 Token-bus Working Group, and is a member of the ACM and the IEEE.



Bhavani M. Thuraisingham is a principal research scientist at the Honeywell Computer Sciences Center and an adjunct professor of computer science at the University of Minnesota. Her research interests include database security, distributed processing, and applications of logic and recursion theory in computer science. Previously she worked at Control Data designing and developing computer networks, and was also a member of faculty at the Department of Computer Science, New Mexico Tech. and at the Department of Mathematics, University of Minnesota. Thuraisingham received a B.S. degree in Mathematics and Physics from the University of Sri-Lanka, M.S. degree in Mathematical Logic from the University of Bristol U.K., and Ph.D. degree in Theory of Computation from University of Wales U.K. in 1975, 1977 and 1979 respectively. She also has an M.S. in computer science from the University of Minnesota. She is a member of ACM and IEEE.

from another is that each is authorized to access only particular sub-sets of the data within the DBMS. An MLS-DBMS addresses the rather natural expectation that users at different levels should be able to use the same database, with each seeing only that data for which s/he has appropriate authorization, and users with different authorizations sharing some data.

Although there have been several attempts at designing a general purpose MLS-DBMS [3,7], the problems encountered in designing and building such a system are quite difficult, and have not yet been overcome

It is only recently and with considerable difficulty that the simpler case, that of providing multi-level security for operating systems and for their associated resources, has been solved. Such a multi-level computing system is generally referred to as a trusted computer base (TCB) [6] within the computer security community. The TCB generally controls the access of user processes to system resources, at the file or device granularity, according to security classification levels.

Providing a multi-level secure DBMS service on current computing systems, even on a TCB, presents a new set of problems. The most obvious of these problems is that the granularity of classification in a useful DBMS will generally be much finer than a file, and may be as fine as single data elements within a record. It is not, however, sufficient to classify data statically as it is stored in the database, using only fixed labels or structural information to determine the data's classification on access. It must also be possible to classify the results of database queries based upon individual data values (the content of the data returned), particular combinations of data elements or values (the context in which data is presented) or the potential for inference, on the user's part, of unauthorized information from otherwise authorized data returned.

There are two fairly direct, known approaches to this MLS-DBMS problem, each of which has serious drawbacks. The first of these is that a "Trusted" DBMS development could be undertaken; this would require formal verification of all DBMS software which is probably an extremely difficult and costly effort. The second approach is that an untrusted DBMS could be hosted on an existing TCB; this would rely upon the TCB's existing security mechanisms for DBMS security which is likely to result in an operationally complex and inconvenient system, probably with rather poor performance. This second approach may also involve such compromises as duplication of parts of the database (resulting in consistency problems and storage of large amounts of redundant data) or extremely awkward access control techniques.

The MLS-DMS design approach we will discuss here is a hybrid of sorts, in that it will require formal verification of limited portions of the DBMS software, and will rely heavily upon the security enforcement mechanisms of a new TCB, the Honeywell Secure Ada Target (SAT) [2].

Our approach uses security constraints as the mechanism for defining classification rules, and query modification as the mechanism for implementing the classification policy. Using these mechanisms, the MLS-DBMS can ensure that responses to users' queries are classified so that they are observable to the users.

This approach will allow users who are cleared at different levels to share a single, integrated, multi-level database. It also eliminates the risk of direct disclosure of data to users who should not see it, protects against direct Trojan Horse attacks and will attempt to limit the information a user can gain through inference and covert channel attacks. Since our design encapsulates the entire DBMS within a special, restricted access-domain, it should be possible to base the system on a conventional, univerified database manager of which some small components or modules may need to be trusted and/or verified. This approach should also provide a foundation upon which more sophisticated inference control mechanisms could be built.

2. Related Work

The MLS-DBMS security techniques we will describe in the following sections have strong antecedents in conventional DBMS technology. Specifically, the techniques of query modification, constraints, views and access restrictions have all been applied to other database problems in the past. This section discusses each of these techniques in their original application, in order to put their use in later sections into perspective.

Query modification was used in the INGRES relational DBMS [10] to implement integrity constraints, views [9] and discretionary access control

[8]. The QUEL query language was used to specify these integrity constraints, views and access control restrictions. When users entered QUEL data retrieval or update requests, the DBMS would modify each request, based upon the QUEL constraints, and then apply the modified request to the database.

Integrity constraints are enforced by modifying every update request into a new update request that is guaranteed not to change the data in ways which would violate those integrity constraints. Similarly, all retrieval and update requests against views are modified into new requests against the base relations. Discretionary access control constraints are enforced by modifying all requests into new requests which contain no access violations; note that no further access violation checking is done.

Our approach uses query modification to enforce mandatory access controls, specified by means of constraints, on every database access request. It also goes beyond simply modifying the query, in that all data returned is checked for access violations, using TCB type enforcement mechanisms, before being released to the user.

3. Basic Assumptions

In order to restrict the scope of the security related design problem, we have made a number of assumptions about the MLS-DBMS environment and operation. These assumptions and their consequences are discussed in this section. (The relational data model is discussed briefly in the next section.)

We have assumed that both the database and its access language conform to the relational model of data [4]. This provides us with a well defined, regular language for defining the database structure and operations.

We have assumed that it is necessary to provide data classification at a fine granularity; that is, at the tuple, attribute or even element level. We have also assumed that the results of functions applied to sets of data may need to be classified independently of the data itself. Further, we have assumed that the classification of data may be required to change, dynamically, over time.

We have assumed that besides simply classifying each (arbitrary) unit of data, it will be necessary to classify data depending upon its value or content, and also the context in which it is seen. This need generalizes into the need to control the user's inference of more sensitive information from less sensitive information.

Finally, we have assumed that it is more expensive to verify that all of the DBMS software satisfies the security requirements, than it is to encapsulate the DBMS and provide verified software which filters all incoming requests and correctly classifies all outgoing results.

4. Relational Database Definitions

The MLS-DBMS uses the relational data model and a query language based upon the relational algebra [4]. In a relational database, the data may be thought of as being structured into tables (Fig. 1). The columns of a relation are referred to as attributes. The degree of a relation is the number of attributes defined for that relation. The rows of a particular relation (table) are referred to as tuples. The cardinality of a relation is just the number of (unordered) tuples it contains. In a particular tuple, the field which corresponds to a particular attribute may contain any one of the values in the domain of that attribute. For an introduction to these relational database concepts see [5]. For a more theoretical discussion of the relational model and the relational algebra, see [11].

4.1 Relational Algebra Operators

The operators of relational algebra act on operands which are relations of a fixed degree. There are five basic operators that serve to define relational algebra: selection, projection, union, set difference and cartesian-product. Two other commonly used operators, intersection and join, can be derived from them [11]. These seven operators are shown in Fig. 2 and described below.

The five basic relational operators are:

1. *Selection*. The selection operator constructs a horizontal subset of a relation. The result of the selection operation is the subset of tuples within a relation, R, for which a specified predicate, P, is true.
2. *Projection*. The projection operator constructs a vertical subset of a relation. The result of the projection operation is the subset of R ob-

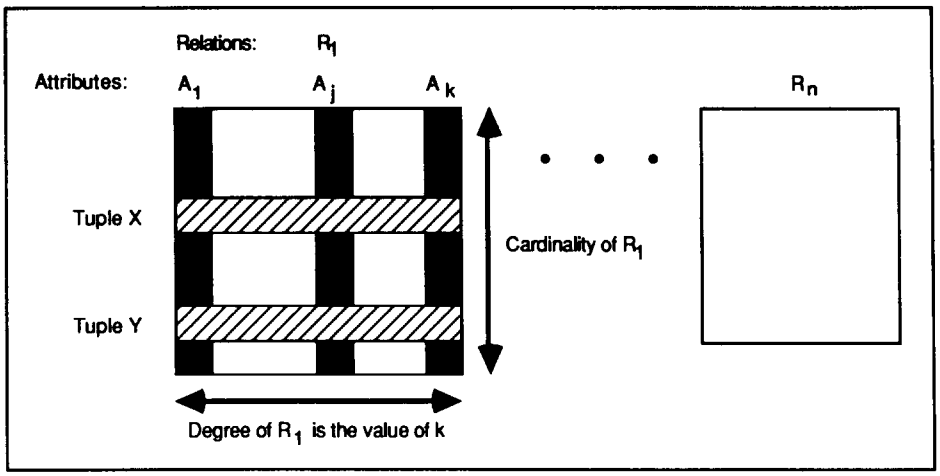


Fig. 1. Relational Database definitions.

Selection	SL [P]
Projection	PJ [A]
Union	R UN S
Set Difference	R DF S
Cartesian Product	R CP S
Intersection	R IN S
Join	R JN [jp] S

Fig. 2. Relational algebra operators.

tained by selecting specified attributes (A), and eliminating others (and also eliminating duplicate tuples within the attributes selected).

3. *Union*. The union of two relations of the same degree, R and S, is the set of tuples that are in R or S or both.
4. *Set difference*. The difference of two relations of the same degree, R and S, is the set of tuples in R but not in S.
5. *Cartesian product*. The cartesian product of two relations of degree m and n, R and S, is the set of $(m \times n)$-tuples whose first m attributes form a tuple in R and whose last n attributes form a tuple in S.

Two additional, commonly used relational operators are:

6. *Intersection*. The intersection of two relations of the same degree, R and S, is the set of tuples in both R and S.
7. *Join*. The join of R and S is those tuples in the cartesian product of R and S for which a specified join predicate, jp, is true.

R

a	b	c
10	Green	16
22	Green	12
17	Purple	14
25	Purple	11
18	Green	15

S

d	e
11	St. Paul
12	Chicago
13	Chicago
14	Cleveland
15	Paris
16	St. Paul

Fig. 3. Example relations R(a, b, c) and S(d, e), a, d are keys; c, d have same domain for join.

4.2 A Relational Database Example

The two relations in Fig. 3 comprise a small sample database which will be used in the subsequent sections to illustrate the operation of query modification.

5. MLS-DBMS Informal Security Policy

The result of a user's query against a relational database is always a relation. Since the database contains data which is classified at various levels, the result relation could contain data items with several different classifications; this is a multi-level response. If security policy enforcement measures were not taken, some of the data in this result relation might not be classified at or below the user's clearance level. Our security policy can be

stated most simply in terms of the difference between the result relations, returned in response to a query, for a non-secure DBMS and a secure DBMS. The non-secure DBMS simply returns all tuples satisfying the query. The multi-level secure DBMS returns only those complete tuples which are classified at or below the user's level, thus all tuples that contain one or more elements above the querying user's level are eliminated from the secure DBMS's results.

Our policy can be described in terms of classification rules and a classification policy. The classification rules are used to associate classification levels with all data in the database, and the classification policy determines the result of a user's query for a user at a particular security level. Security constraints are the mechanism for defining classification rules, and query modification is the mechanism for implementing the classification policy. These mechanisms are discussed in the following two sections.

6. Security Constraints

Security constraints are used in our approach to associate classification levels with all data in the relational database. They provide the basis for a versatile, powerful classification policy because any subset of data can be specified and assigned a level statically or dynamically.

Simple constraints allow for classification of an entire database, as well as classifying by relation or by attribute. Constraints that classify by content provide the mechanism for classification by tuple and by element. Constraints that classify by context are the mechanism for classifying relationships between data. Any subset of the database can be classified based upon content or context. In addition, the results of applying functions to an attribute or a subset of an attribute, for example, sum, average, and count, can be assigned different classification levels than the underlying data. Finally, the classification levels of the data can change dynamically based upon changes in time, content, or context.

It is important to note that "simple" and "content" based constraints can be applied to data as it is actually stored in the database, while "context," "functional," and "dynamically" based constraints can only be applied in the computation of the result relation which is to be output in response to a user's query.

A constraint consists of a data specification and a classification. The data specification defines any subset of the database using the relational algebra and the classification defines its classification level. A constraint has the following standard form:

$$L\big(f_n\big(PJ[a_{ij},\ldots](SL[P](R_1\ OP\ (R_2\ OP\ \ldots R_n)))\big)\big) = \text{classification level}$$

where

R_i $(1 \leqslant = i \leqslant = n)$ is either a base relation or an expression in the standard form;

f_n is either null, average, sum, count, max, or min;

a_{ij} is the jth attribute of R_i;

P is either TRUE or a predicate of the attributes of $R_1, \ldots, R_n$; and

OP is either CP, UN, DF, $JN[jp]$, IN, or null.

Using the relational algebra, the data specification can define any subset of the database. However, in practice a limited number of types of data specifications will actually be used. The following examples illustrate classification policies for a relational database and the constraints that must exist to support them.

Each set of examples consists of an abstract and a concrete example. The abstract example is based on the following database:

$R_1(a_{11}, \ldots, a_{1j})$,
$R_2(a_{21}, \ldots, a_{2k}) \ldots$
$R_n(a_{n1}, \ldots, a_{nl})$

In this database description R_i is the i-th relation and a_{ij} is the j-th attribute of R_i.
The concrete example is based on the database in Fig. 3.
Each example presents an informal, English language definition of a classification rule (on the left) paired with the relational algebra constraints (on the right) needed to implement or enforce that rule through query modification.

The constraints in the example are a subset of

the standard form given above:

$$L\big(f_n\big(PJ[a_{ij},\ldots](SL[P](R_1JN[a_{1k}=a_{2l}](R_2JN[jp]\ldots)))\big)\big) = \text{classification level}$$

The first set of constraints define classification levels for the entire database.

1a. database classified T.S. (all data in the entire database is classified T.S. – equivalent to a T.S. label on each element in database)	$L(PJ[a_{ij}]R_i) = \text{T.S.}$ for all i, j
1b. database is classified T.S.	$L(PJ[a]R) = \text{T.S.}$ $L(PJ[b]R) = \text{T.S.}$ $L(PJ[c]R) = \text{T.S.}$ $L(PJ[d]S) = \text{T.S.}$ $L(PJ[e]S) = \text{T.S.}$

The next set of constraints define classification levels for a relation.

2a. relation R_i classified T.S. (all data in the entire relation is classified T.S. – equivalent to a T.S. label on each element in relation)	$L(PJ[a_{ij}]R_i) = \text{T.S.}$ for some i for all j
2b. R is classified T.S.	$L(PJ[a]R) = \text{T.S.}$ $L(PJ[b]R) = \text{T.S.}$ $L(PJ[c]R) = \text{T.S.}$

The next set of constraints define classification levels for an attribute.

3a. attribute a_{ij} classified T.S. (all data in the entire attribute is classified T.S. – equivalent to a T.S. label on each element in attribute)	$L(PJ[a_{ij}]R_i) = \text{T.S.}$ for some i for some j
3b. a is classified T.S.	$L(PJ[a]R) = \text{T.S.}$

The next two sets of constraints illustrate classification by context. The first set define classification levels for 2 attributes in the same relation by using the $PJ[A]$ operator.

4a. relationship between attributes in the same relation classified T.S. (they are classified T.S. when read together, but not necessarily when read individually)	$L(PJ[a_{ij},\ldots]R_i) = \text{T.S.}$ for some i for 2 or more j's
4b. relationship between a and b is classified, i.e. that 10, 22, and 18 are Green and 17 and 25 are Purple is classified T.S.	$L(PJ[a, b]R) = \text{T.S.}$

The second set define classification levels for 2 attributes in different relations by using the $PJ[A]$ and $JN[jp]$ operators.

5a. relationship between attributes in different relations classified T.S.	$L(PJ[a_{ij},\ldots](R_iJN[jp]\ldots)) = \text{T.S.}$ for 2 or more i's for 1 or more j's
5b. relationship between a and e is classified T.S.	$L(PJ[a, e](R\,JN[c = d]\,S)) = \text{T.S.}$

The next two sets of constraints illustrate classification by content. The first set defines classification levels for all attributes in a relation that satisfies a particular predicate by using the $SL[P]$ operator.

6a. tuples classified T.S. (all data in the tuples is classified T.S.)	$L(PJ[a_{ij}](SL[P]R_i)) = \text{T.S.}$ for some i for all j
6b. everything when b = Purple is classified T.S.	$L(PJ[a](SL[b = \text{Purple}]R)) = \text{T.S.}$ $L(PJ[b](SL[b = \text{Purple}]R)) = \text{T.S.}$ $L(PJ[c](SL[\text{b} = \text{Purple}]R)) = \text{T.S.}$

The second set define classification levels for a single attribute in a relation that satisfies a particular predicate by using the $SL[P]$ operator.

7a. elements classified T.S.	$L(PJ[a_{ij}](SL[P]R_i)) = \text{T.S.}$ for some i for some j
7b. c when b = Purple is classified T.S.	$L(PJ[c](SL[b = \text{Purple}]R)) = \text{T.S.}$

The next set of constraints illustrate the classification of a function of an attribute.

8a. function of an attribute classified T.S. f = average, count, sum, max, min	$L(f_n(PJ[a_{ij}]R_i)) = \text{T.S.}$ for some i for some j
8b. average of c is classified T.S.	$L(\text{average}(PJ[c]R)) = \text{T.S.}$

The operators can be combined to provide a very flexible classification policy. The following set of constraints combine classification by context and content.

9a. relationship between elements in different relations classified T.S.	$L(PJ[a_{ij}, \ldots](SL[P](R_iJN[jp]\ldots))) = \text{T.S.}$ for 2 or more i's for 1 or more j's
9b. relationship between a and e when b = Purple is classified T.S.	$L(PJ[a, e](SL[b = \text{Purple}](R\,JN[c = d]\,S))) = \text{T.S.}$

Note that query modification prevents one individual query from violating a constraint, but on multiple queries. Enforcing constraints across multiple queries would require use of a history file or equivalent mechanism. Such a mechanism would enable the DBMS to treat a single query in the context of all previous queries made by a given user.

Example:

Consider constraint number 5b:
$L(PJ[a, e](R\,JN[c = d]\,S)) = \text{T.S.}$

Suppose the user makes the following two queries:

1. $PJ[a, c]R$
2. $PJ[d, e]S$

These are equivalent to the single query:

3. $PJ[a, e](R\,JN[c = d]\,S)$,

which would have been rejected by query modification due to constraint 5b. The first 2 queries are equivalent to the 3rd query, because the foreign key in relation R (c) is displayed in the 1st query,

and the key of relation S(d) is displayed in the 2nd query. The user can compute the result of the join between R and S manually, and thus discover the relationship between attributes a and e.

7. Query Modification

Query modification is the mechanism for supporting the classification policy. The user's query is first modified according to the security constraints so that the response can be assigned a classification which will make it observable to the user. The modified query is then compiled and executed. We will first state the essential points in the query modification algorithm briefly, and then describe this algorithm in greater detail.

When a user poses a query, this query is modified as follows: Select those constraints from the set of all constraints, which have classified at least one attribute with a classification level not less than or equal to that of the user. Then modify the query by applying these selected constraints in such a way that when the modified query is posed, only the information in the database which is classified at an equal or lower level than that of the user is returned.

In the algorithm we will only consider a subset of the queries in the standard form given in the previous section. The queries in this subset are of the form:

$$f_n\big(PJ[a_{ij},\ldots](SL[P]E)\big),$$

where E is either a base relation or a relation derived from base relations using the operators CP, UN, and DF. The essential points of the argument can be clearly exhibited by considering this subset of queries, and we can extend our argument to include all types of queries.

Most common and useful queries belong to this subset. For example, it can be shown by induction on the number of operators that any query which uses only the operators PJ, SL, JN, and CP belongs to this subset.

Before we state the algorithm, we will clarify the special terms that we have used. By an "empty query' is meant a non-existent query. By the 'qualification in a query or a constraint' is meant the condition specified in the 'where' clause in the operation SL.

The algorithm proceeds in stages:

STAGE 1: SELECT THE CONSTRAINTS RELEVANT TO THE QUERY

Let C_1, C_2, $C_3,\ldots, C_n$ $(n \geqslant 1)$ be the constraints where for each i $(1 \leqslant i \leqslant n)$ the level assigned in C_i is denoted by $L(C_i)$. Assume that the classification levels form a lattice where '$\leqslant$' is the partial ordering. Therefore, for any two levels L_1 and L_2, either L_1 AND L_2 are incomparable, $L_1 \leqslant L_2$, or $L_1 > L_2$.

Let Q be the query posed by a user whose clearance is L. Let the qualification in the query be G. Note that G may be NULL. Let the attributes used in the qualification be B_1, $B_2,\ldots B_t$ and those used in the result be A_1, $A_2 \ldots A_m$. Let $\{P_1, P_2,\ldots, P_r\}$ $(r \geqslant 1)$ be the largest subset of $\{C_1, C_2,\ldots, C_n\}$ satisfying the following conditions:

1. $L(P_i)$ not $\leqslant L$ where $1 \leqslant i \leqslant r$.
2. For each $i(1 \leqslant i \leqslant r)$, the set of attributes classified in P_i is a subset of $\{A_1, A_2,\ldots A_m, B_1, B_2,\ldots B_t\}$.

P_1, $P_2,\ldots P_r$ are the constraints relevant to the query. Let the qualification associated with each P_i be F_i $(1 \leqslant i \leqslant r)$. Note that F_i may be NULL.

STAGE 2. TRANSFORM THE QUERY Q TO Q^*

```
CONTINUE := TRUE; INDEX := 1;
WHILE (INDEX ⩽ r AND CONTINUE) DO
    Consider constraint P_i.
    IF F_i is NULL, i.e. the constraint has no qualification, THEN
        Q* is the empty query.
        CONTINUE := FALSE;
    ELSE
        Consider the expression (G AND (NOT(F_i)).
        IF this expression results in a CONTRADICTION, i.e. it consists
        of a subformula of the form (A AND (NOT A)) THEN
            Set Q* to be the empty query.
            CONTINUE := FALSE;
```

```
        ELSE
            Let R1 and R2 be the relations to which a select is applied in the query Q and the constraint Pi respectively.
            Set R = R1 CP R2
            Let K be the condition that is used to JOIN R1 and R2 over
            common attributes, i.e. R1JP[K]R2 = SL[K](R1CPR2)
            The new query Q is obtained from the old query Q be substituting R for R1 and (G AND K AND (NOT(Fi))) for the
            qualification G.
            i.e. R1 := R; G := G AND K AND (NOT(Fi))
            IF INDEX = t THEN
                CONTINUE := FALSE;
                Q* := Q;
            ELSE
                INDEX := INDEX + 1;
            ENDIF;
        ENDIF;
    ENDIF;
ENDWHILE;
```

We will now trace this algorithm with an example. Consider the relations R and S defined in Fig. 3. Let the constraints enforced be

$L(PJ[a](SL[a = 22]R)) = \text{T.S.}$

$L(PJ[b](SL[a = 22]R)) = \text{T.S.}$

$L(PJ[c](SL[a = 22]R)) = \text{T.S.}$

Suppose an unclassified user wants to obtain the colors and cities when $a = 22$. The user will pose the following query:

$PJ[b, e](SL[a = 22](R\ JN[c = d]\ S))$

This query is equivalent to the following query:

$PJ[b, e](SL[a = 22 \text{ AND } c = d](R\ CP\ S))$

It can be seen that the constraints are relevant.

The expression (G AND NOT(F_i)) in stage 3 of the algorithm is ($a = 22$ AND $c = d$ AND NOT($a = 22$)). This results in a contradiction. Therefore the query is modified to the empty query and no information is returned to the user.

8. Conclusion and Future Work

We have presented a unique approach for defining classification rules and enforcing a classification policy that ensures that the response to a user's query against a multi-level secure database can be assigned a classification which will make the response observable to the querying user. This approach consists of the modification of a user's query based on the security constraints and the application of the modified query to the database. We have also devised an algorithm which successfully implements this query modification process.

We are currently in the process of investigating how the mechanisms of the SAT TCB could be utilized to minimize the amount of code that has to be verified within the DBMS based on the query modification process, extending our approach to include updates, analyzing mechanisms for inference control, and analyzing covert channels.

References

[1] D.E. Bell, and L.J. LaPadula: *Secure Computer System: Unified Exposition and Multics Interpretation*, MITRE Technical Report MTR-2997, July, 1975.

[2] W.E. Boebert, W.D. Young, R.Y. Kain, and S.A. Hansohn: *Secure Ada Target: Issues, System Design and Verification*, 1985 IEEE Symposium on Security and Privacy, Oakland, CA, April 22–24, 1985, pp. 176–183.

[3] D. Bonyun et al.: *A Model of a Protected Data Management System*, I.P. Sharp Report ESD-TR-76-289, 1976.

[4] E.F. Codd: A Relational Model of Data for Large Shared Data Banks, *Communications of the ACM*, Vol. 13, No. 6, June, 1970, pp. 377–387.

[5] C.J. Date: *An Introduction to Database Systems*, Addison-Wesley Publishing Company, 1977.

[6] DoD Computer Security Center: *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83, August 15, 1983.

[7] T.H. Hinke, and M. Schaefer: *Secure Data Management System*, Rome Air Development Center Report RADC-TR-266, November, 1975.

[8] M. Stonebraker, and E. Wong: Access Control in a Relational Data Base Management System By Query Modification, *ACM National Conference Proceedings*, 1974, pp. 180–186.

[9] M. Stonebraker: Implementation of Integrity Constraints and Views by Query Modification, *ACM SIGMOD International Symposium on Management of Data*, 1975, pp. 65–78.

[10] M. Stonebraker, E. Wong, and P. Kreps: The Design and Implementation of INGRES, *ACM Transactions on Database Systems*, Vol. 1, No. 3, September, 1976, pp. 189–222.

[11] J.D. Ullman: *Principles of Database Systems*, Computer Science Press, 1982.