Security Checking in Relational Database Management Systems Augmented with Inference Engines

M.B. Thuraisingham
Honeywell, Corporate Systems Development Division, Golden Valley, Minnesota, USA

In this paper we will discuss the notion of multilevel security and the difficulties encountered in designing an implementation scheme for a security policy for a multilevel secure database management system (MLS/DBMS). We will then describe how these difficulties may be overcome in augmenting a database with an inference engine so that it functions like a knowledge based system.

Keywords: Multilevel security, Multilevel secure database management systems, Inference engine, Knowledge based systems, Security policy, Logic and databases

Dr. M.B. Thuraisingham is a Principal Research Scientist at Honeywell Corporate Systems Development Division 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. She has also served as a faculty member in the Department of Computer Science, New Mexico Institute of Mining and Technology, and in the Department of Mathematics, University of Minnesota. Dr. Thuraisingham received the B.Sc. degree in Mathematics and Physics from the University of Sri-Lanka, M.Sc. degree in Mathematical Logic from the University of Bristol, U.K., and Ph.D. degree in Recursion Theory and Theory of Computation from the University of Wales, U.K. in 1975, 1977 and 1979 respectively. She also has the M.S. degree in Computer Science from the University of Minnesota. She has published articles in Computers & Security, Journal of Computer and many System Sciences, Notre Dame Journal of Formal Logic, and the Zeitschrift fur Mathematische und Grundlagen der Logik. She is a member of ACM, IEEE, and Sigma-Xi.

North-Holland
Computers & Security 6 (1987) 479–492

1. Multilevel Security in Database Management Systems

Security is the control of the flow of data and information through the system and its interfaces with the outside world. This is done to prevent the unauthorized disclosure of information or modification of data protected by the system. This can be achieved by assigning data to sensitivity classes and restricting the types of access individuals and code acting on their behalf can have to each of the sensitivity classes. A security policy describes the set of restrictions. One reasonable approach is to associate an element from a partially ordered set of security levels with each of the sensitivity classes and with each individual, and then to restrict accesses so that information and data can only flow upward in level [7]. Such a policy is called a multilevel security policy, and a system which enforces one is said to be multilevel secure.

When there is a database on a multilevel secure system, it is possible that not all of the data contained in the database is equally sensitive. However, present-day database management systems are not built with adequate controls and mechanisms to assure that users are allowed to access only the data for which they have been granted a clearance and for which they have a legitimate need to know, but at the same time provide for the sharing of data by these users. Thus an (MLS/DBMS) is different from a conventional DBMS in at least the following ways:

- Every data item in the database has associated with it one of several classifications or sensitivities, that may need to change dynamically over time;
- A user's access to data must be controlled based upon the user's authorization with respect to these data classifications.

Providing a MLS/DBMS service on current computing systems presents a new set of problems than those encountered in developing secure operating systems. The most obvious of these problems is that the granularity of classification in a DBMS is generally finer than a file, and may be as fine as a single data element. Another problem that is unique to databases is the necessity to classify data based on content, time, aggregation and context. Furthermore, DBMSs are also vulnerable to (1) inference attacks where a user infers unauthorized data from the knowledge that he has accumulated and/or the context in which data is displayed, and (2) Trojan Horse programs which are hostile code residing in the system. Any security policy for MLS/DBMS must not only protect against direct disclosure of data, but it must also attempt to limit the attacks from hostile users. The aim of the mandatory security policies is to confine Trojan Horse programs in such a manner that their attempts at hostile behaviour are frustrated.

A formulation of a mandatory security policy that handles inference threats in a read only relational database management system is stated as follows: A subject has read access to an object only if the following conditions are satisfied:

1. Its classification level (referred to as clearance) is greater than that of the object, and
2. By combining the information in the object with the information that it has obtained previously by reading other objects and with the knowledge that it has gleaned from the outside world, the subject will not acquire any new information to which it has no authorization.

The formalization of this policy will be described progressively. For any classification level L, there is a corresponding environment $E(L)$ which consists of all responses that have been properly classified at a level less than or equal to L, i.e. all the responses that a user at level L is authorized to know. The following relationship holds among the environments. If $L_1 > L_2$, then $E(L_2)$ is a subset of $E(L_1)$. A response R is properly classified at a level L if R combined with information already in $E(L)$ will not result in the inference of any information at a level not less than or equal to L, i.e. the following condition holds

$$\forall L_1 \geqslant L, \text{CLASSIFICATION}(\text{INFER}(R, E(L_1)) \leqslant L_1$$

where CLASSIFICATION(X) is the level assigned to X. $(R, E(L))$ is $E(L)$ union the response R, and INFER(R, $E(L)$) is the information that can be inferred from R and $E(L)$. Then the policy states that for any query Q posed by a subject at a level L, the response R that is released by the DBMS should be properly classified at level L.

Partial solutions to the implementation of this policy in a relational DBMS using standard languages such as relational algebra have been proposed [10]. However, the techniques employed by standard relational database management systems are not sufficient to resolve the major compromise to security in databases due to the inference of unauthorized information from the information that users can legitimately acquire. Also this approach does not facilitate the expression and management of mechanisms which classify the data.

Any complete implementation of this policy requires a suitable definition for the INFER predicate described in the policy. In [8], a definition of inference has been given based on the uncertainty of acquiring a particular information. It states that X can be inferred from Y if by acquiring Y all uncertainty on X has been removed. The disadvantage of this definition of inference is the lack of a suitable formalization of the notion of uncertainty. In the past, uncertainty has been related to the program size complexity [6]. Whether this can be applied to the programs which compute queries to databases remains to be investigated further.

Our formulation of the definition of inference and consequently a solution to the design of an implementation scheme for the policy is described in detail in Section 3. The essence of our approach is to apply artificial intelligence techniques by means of formal logic. A previous approach to partially implement the policy is described briefly in Section 2. For the database theory terminology used in this paper, we refer the reader to [18]. An introduction to concepts in Mathematical Logic can be obtained in [14]. A survey of mandatory security policies has been presented in [2].

2. Previous Approach to Implement the Security Policy

Security constrains have been used in the past to associate classification levels with all data in the relational database [10]. 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 provide for the classification of the entire database, as well as the classification by relation and by attribute. Constraints that classify by content provide the mechanism for classification by tuple and by element. Context based constraints classify relationships between data. In addition, the results of applying functions to an attribute or a subset of an attribute, such as 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.

A constraint consists of a data specification and a classification. The data specification defines any subset of the database using relational algebra and the classification defines the classification level of this subset. For example, consider a database which consists of a relation EMP(NAME, SALARY, SOC_SEC#) with SOC_SEC# as the key.

The content based constraint which classifies all names who earn more than 50K as secret is expressed as

LEVEL(PROJECT[NAME]
  (SELECT[SALARY > 50K] EMP)) = SECRET

and the context based constraint which classifies all names and salaries taken together as secret is expressed as

LEVEL(PROJECT[NAME, SALARY] EMP) = SECRET

The simple constraint which classifies all names and salaries taken individually as secret is expressed as

LEVEL(PROJECT[NAME] EMP) = SECRET

LEVEL(PROJECT[SALARY] EMP) = SECRET

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 be only applied in the computation of the result which is to be output in response to a user's query. For example, the content based constraint which classifies all names who earn more than 50K as secret will result in the storage of the relation EMP in secret and unclassified files in our design. The salaries and ss will be stored in an unclassified file and the names will be placed either in a secret file or an unclassified file depending on its sensitivity. Instead, if the context based constraint which assigns secret classification to names and salaries taken together is enforced then the relation is stored in unclassified files, and the classification level is assigned to the response only when it is released.

The security constraints are used in evaluating queries in a technique called query modification. Variations of this technique have been used to solve other database problems in the past [17]. The user's query is first modified according to the security constraints so that the response can be assigned a classification level which will make it observable to the user. The modified query is compiled and executed. We will illustrate this technique with an example. Suppose an unclassified user poses the query 'Retrieve all names from EMP' and there is a constraint which classifies all names who earn more than 50K at a secret level. Then this query is modified to 'Retrieve all names from EMP who earn less than or equal to 50K'. This modified query is posed and the response returned will not contain any secret values.

This approach is inadequate to handle threats due to inference attacks. To overcome such attacks, the history of all the queries that have been posed, and the responses that have been released by the DBMS has to be maintained. Furthermore, for each additional query that is posed certain constraints have to be examined to determine whether the release of any information by the DBMS may lead to the inference of unauthorized information. Standard relational database management systems do not provide any mechanism to accomplish these tasks. Although the concept of a relation is well defined and any object can be modelled as a relation, the manipulation of relations used to model complex objects can be extremely intricate.

In addition to the inability to handle multiple queries, another disadvantage with this method is that it is not possible to include any information which is not present in the database. For example, one cannot define a senior-emp as a person who earns more than 40K unless this information is in the database. In the next section we shall describe a formalism that is powerful to represent constraints as well as real world information, and at the same time be flexible so that the data can be manipulated easily.

3. Enhancing Database Management Systems with Inferencing Capability

The desire to integrate database technology, artificial intelligence technology and logic programming technology resulted in the investigation of extending database systems to provide them with the functionality of expert systems, thus creating knowledge based systems [1]. A solution that has been proposed to extend the power and capability of database management systems is to augment a database with a knowledge base which consists of all information for which the relational model is not an appropriate representation, and a processing system separate from the DBMS to process the information in the knowledge base [4]. Since standard relational databases are not powerful enough to overcome inference threats, the augmented database approach can be taken to design a MLS/DBMS. The database consists of only those information which can be manipulated easily by the relational DBMS. The knowledge base consists of all security constraints, integrity constraints, the environments at the various classification levels, and the real world information. The information in the knowledge base is more complex than the information in the database, and therefore a model other than the relational model is needed to represent it. Furthermore, it is also desirable to have a uniform model to represent all the data in the knowledge base. Since the constraints are easily expressed as rules, 'Logic' is an extremely appropriate candidate to model the knowledge base.

Logic is powerful enough to be used to formalize programs, specifications, databases, legislation and natural language among others [12]. All security constraints, real world information, environments, and integrity constraints are expressed as formulas in first order logic. This set of formulas which constitutes the knowledge base will also be referred to as the universe or the rule base. The system which processes the formulas in the rule base to deduce new formulas by some rule of inference is an inference engine. A rule of inference that is used in first order logic is 'Modus Ponens' where B is deduced from the formulas 'A' and 'If A then B' [14]. Consequently most logic programming systems which deal with Horn clause logic, a subset of first order logic, use 'Resolution' as the rule of inference [16]. Subsequently variations to the resolution principle have been formulated by many researchers [13]. In addition to the ease of expression and manipulation of the constraints expressed in logic, another advantage in using logic is that logical deduction of formulas fits in naturally with the deduction of information which may lead to the violation of security by inference. Consequently a definition of 'violation of security by inference' can be formulated as follows. Let R be the response released by the DBMS to a query posed by a user at a level L. Let S be the set of formulas which represents all the information in $(E(L), R)$. Let A be a formula that can be deduced from S by a logical rule of inference, i.e. there is a sequence of formulas $B_1, B_2, \dots, B_n$ where $B_n = A$, each B_i $(1 \leqslant i \leqslant \text{n})$ either belongs to S or there exist j, k where $(1 \leqslant j,\ k \leqslant n)$ such that B_i results from B_j and B_k by applying some rule of inference. Then a violation of security by inference has occurred if the sensitivity of A is greater than L.

A high level architecture of standard relational databases and relational databases augmented with inference engines are shown in *Fig. 1*. This approach in augmenting database systems with an inference engine originates in the study of Logic and Databases [11]. In this study, there are three possible approaches to view a database using logic as a framework. The first approach which is commonly used in question-answering systems represents the database as a first order theory, i.e. the proper axioms of this theory are the facts in the database and the general laws. The general laws are any rules that are enforced on the relations in

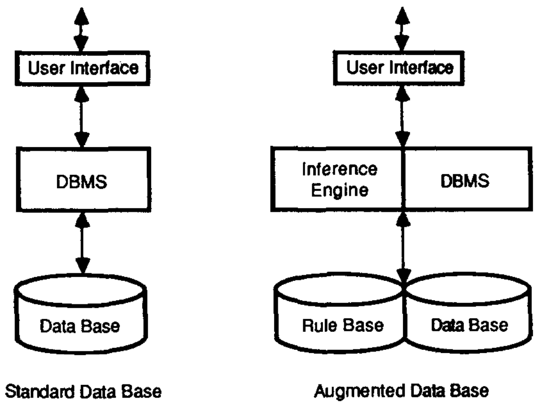

Fig. 1. High level architecture of standard relational databases and relational databases augmented with inference engines.

the database. Queries are expressed as formulas in first order logic. Responses to queries are those instances of the corresponding formulas which can be proved as theorems of this first order theory. The second approach considers the set of elementary information which constitutes the database as an interpretation of a first order theory whose proper axioms are the general laws. The general laws here are all used as integrity rules. Responses to queries are those instances which evaluate to true under this interpretation. Standard relational databases implicitly follow this approach. The third approach is similar to the second one except that the general laws consist of the integrity rules as well as the derivation rules. These derivation rules are used to generate extensions to the relations in the database. We will illustrate the three approaches with examples.

Suppose, the database consists of the two relations FATHER(father, child) and GF(grandfather, grandchild), and the two laws 'The age of the father is greater than that of the child', and 'the father of the father is a grandfather'. In the first approach the facts in the database as well as two laws are considered to be proper axioms of a first order theory. Then a query to retrieve all the (grandfather, grandson) pairs results in all pairs (X, Y) such that GF(X, Y), can be proved to be a theorem of this theory from the axioms. In the second approach the database is an interpretation of a first order theory whose proper axioms are the two laws which are integrity rules. Therefore, if a tuple (X, Y) is in the GF relation, then there should be tuples (X, Z), and (Z, Y) already present in the FATHER relation. In the third approach the first law is considered to be an integrity rule and the second law a derivation rule. This derivation rule generates new tuples which belong to the extension of the GF relation. Therefore, although the pair (X, Y) may not be present in the GF relation in the database, it will be returned as part of the response to the query to retrieve all pairs (grandfather, grandchild) if there is a Z such that (X, Z) and (Z, Y) are in the FATHER relation.

Our approach closely follows the third view of the database, but incorporates security as an additional feature. Throughout this section, we will illustrate our augmented database approach with examples. The main issues that will be addressed are the expression and management of the universe and the evaluation of queries.

3.1 Expression and Management of the Universe

As shown in *Fig. 2*, the universe, whose elements are expressed as formulas in first order logic, consists of the integrity constraints, security constraints, the real world information, and the environments corresponding to the various classification levels. The classification levels under consideration in increasing order of sensitivity are unclassified, secret and top-secret. This section will be devoted to a description of this universe.

3.1.1 Integrity Constraints

Integrity constraints which are laws enforced on the data have been studied extensively in the past [18]. Whenever data is updated, these constraints are checked to ensure that the update is valid. In addition to enforcing integrity constraints on the data in the database, we will also enforce on the security constraints and the real world information. However the techniques utilized to enforce integrity on security constraints are identical to those used for the data in the database. These techniques will be described when we discuss security constraints.

3.1.2 Security Constraints

Security constraints, which we will also refer to as constraints, are the rules which assign classification levels to data either directly by a user who is authorized to do so, or by deduction from the other rules. These constraints are expressed as formulas of the form $A \rightarrow B$(read 'IF A THEN B') where the formula A specifies conditions imposed and the formula B specifies the level assigned to the data to be classified.

Consider the following constraints enforced on the relation EMP:

1. Salary is secret.

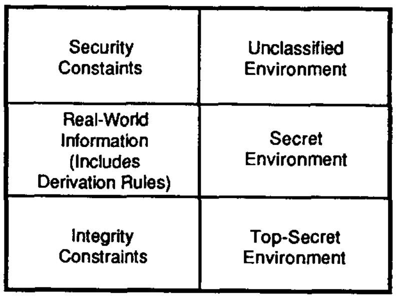

Fig. 2. The universe (or the rule base).

2. Name where salary > 50K is secret.
3. If Reagan is the president, then Name is secret.
4. Name and Salary taken together are secret.

These constraints are expressed by the following respective formulas.

$C1$: $\text{EMP}(X, Y, Z) \rightarrow \text{LEVEL}(Y, \text{SECRET})$

$C2$: $\text{EMP}(X, Y, Z) \wedge \text{GREATER}(Y, 50K)$
$\rightarrow \text{LEVEL}(X, \text{SECRET})$

$C3$: $\text{EMP}(X, Y, Z) \wedge \text{PRESIDENT}(\text{REAGAN})$
$\rightarrow \text{LEVEL}(X, \text{SECRET})$

$C4$: $\text{EMP}(X, Y, Z) \rightarrow \text{LEVEL}(XY, \text{SECRET})$

where LEVEL(X, L) is the level assigned to X.

In the case of constraint $C4$, if we assume that if one of name or salary is released by the DBMS at an unclassified level, then the other is classified at a secret level, then $C4$ can be expressed by the following two constraints.

$C5$: $\text{EMP}(X, Y, Z) \wedge \text{RELEASE}(X, \text{UNCLASSIFIED})$
$\rightarrow \text{LEVEL}(Y, \text{SECRET})$

$C6$: $\text{EMP}(X, Y, Z) \wedge \text{RELEASE}(Y, \text{UNCLASSIFIED})$
$\rightarrow \text{LEVEL}(X, \text{SECRET})$

where RELEASE(X, L) implies that X is released into the environment $E(L)$.

Whenever a security constraint is updated (i.e. inserted, deleted, or modified) then the integrity of the data (in the universe or database) has to be maintained. Therefore, the insertion, deletion or modification of security constraints can be treated in the same way as the update of data in the database. When data is updated certain semantic integrity constraints may be checked for the maintenance of consistency. If the data is not updatable, then recovery mechanisms are needed. Concurrency control is essential when more than one user is simultaneously modifying the data. A detailed treatment of handling integrity in databases has been investigated in [15].

A similar argument can be applied when constraints are updated. Some of the essential points in this treatment are explained here. When inserting a security constraint, one must first check whether any integrity constraints enforced are violated. Then through a deduction process all the information that can be deduced from the added information must be found and made explicit. When deleting information, due to the fact that the deduced information was made explicit in the add phase, this deduced information should also be deleted unless it can be deduced from other information. Modifying information can be treated as a deletion followed by an insertion. Therefore the operations performed during both the insert and delete phases have to be applied. We will illustrate this process with examples. Suppose, in addition to the relation EMP, the database also contains another relation DEPT(NAME, DEPT#, PROJECT) with DEPT# as the key. The NAME attribute specifies the names of employees working on a project in a department. Then, if the constraint $C2$ is enforced, the names in DEPT who are earning more than 50K have to be assigned a secret classification level. This information has to be made explicit. Therefore the following constraint $C5$ should also be inserted.

$C7$: $\text{EMP}(X, Y, Z) \wedge \text{DEPT}(X, U, V)$
$\wedge \text{GREATER}(Y, 50K) \rightarrow \text{LEVEL}(X, \text{SECRET})$

The same argument applies when $C2$ is already enforced and DEPT is entered as a new relation into the database. Again $C7$ has to be made explicit. If $C2$ is deleted, then since $C7$ cannot be deduced any other way, it also has to be deleted. But the converse does not hold.

In certain cases, the insertion of a constraint may cause data to have conflicting levels. Suppose the following constraint is enforced in addition to $C2$.

$C8$: $\text{DEPT}(X, Y, Z) \wedge \text{EQUAL}(Z, P1)$
$\rightarrow \text{LEVEL}(X, \text{TOP-SECRET})$

That is, all those who work in project $P1$ are classified at a top-secret level. If there exists a person who earns more than 50K and who works in project P_1, then this person has two levels assigned to him. The resulting level assigned to this data depends on the integrity constraints that are enforced. There are two solutions to this problem. One solution is to enforce the integrity constraint that a data cannot have conflicting levels. Another solution is to enforce the integrity constraint that if the two levels assigned to the data are comparable, then the data is assigned the higher level, and the security constraints which caused the data to be classified at the lower level should be updated. If the levels are incomparable, then the constraint that caused the inconsistency should not be inserted.

In the previous example, if we adopt the first solution, then the constraint $C8$ cannot be inserted. If we adopt the second solution, then since the top-secret level dominates the secret level in the lattice, the name is assigned a top-secret classification, the constraints $C2$ and $C5$ are deleted, and the following constraints are inserted.

$C9$: EMP(X, Y, Z) GREATER(Y, 50K)
NOT(LEVEL(X, TOP-SECRET))
$\rightarrow$ LEVEL(X, SECRET)

$C10$: EMP(X, Y, Z) GREATER(Y, 50K)
DEPT(X, U, V) NOT(LEVEL(X, TOP-SECRET))
$\rightarrow$ LEVEL(X, SECRET)

In addition to maintaining the consistency of a set of constraints, we also want the set to be complete. That is each data in the database should be assigned a classification level. However, if the database is very large, this will require a great number of constraints. One way to overcome this is to assume that any data that is not explicitly classified takes a default classification level. Suppose the classification levels that are considered are top-secret, secret and unclassified, then this assumption can be expressed by the rule

$C11$: NOT(LEVEL(X, SECRET))
$\wedge$ NOT(LEVEL(X, TOP_SECRET))
$\rightarrow$ LEVEL(X, UNCLASSIFIED)

3.1.3 Environment

The environment for a classification level contains all the information that has been released by the DBMS at its level, and also it contains all the environments whose classification levels are less than its level. For example, the top-secret environment includes all the information that has been released by the DBMS at a top-secret level, and it also includes all the information in the secret and unclassified environments. The format of the information in an environment will be explained in the next section when we discuss the evaluation of queries. In our treatment of the environment, we do not distinguish between the users at a particular clearance level. Therefore, in our discussion we accommodate only three different users. It should be noted that our results can be easily extended to differentiate between the various users at a level by incorporating the user-id as an argument.

Any information released into an environment is visible to the users at levels greater than or equal to that of the environment. Furthermore information classified at level L can be released into the environment $E(L)$ if other information at level $L1 > L$ cannot be deduced from a user at level L. This is represented by the following rules.

$C12$: RELEASE(X, L) $\wedge$ DOMINATE(L_1, L)
$\rightarrow$ RELEASE(X, L_1)

$C13$: LEVEL(X, L) $\wedge$ LEVEL(Y, L_1)
$\wedge$ DOMINATE(L_1, L)
$\wedge$ NOT(RELEASE(X, L) $\rightarrow$ LEVEL(Y, L))
$\rightarrow$ RELEASE(X, L)

The rules $C11$, $C12$, and $C13$ are examples of derivation rules. In our discussion we shall include such rules as part of the real world information.

3.1.4 Real World Information

Real world information is any information which is either a derivation rule which defines extensions to the relations in the database or it is not present in the database. Some examples of real world information in the form of derivation rules have been presented in the previous sections. In the database consisting of the FATHER and GF relation, the rule that 'the father of a father is a grandfather' is real world information. So is the rule which specifies the default classification level that is assigned to the data. We can also include information which has no connection to the data in the database as part of the real world information. For example, in our sample database which consists of the relation EMP and DEPT, the fact that Reagan is the president has no relevance, but we can include this information in the universe by the rule

$C14$: $\rightarrow$ PRESIDENT(REAGAN)

We can also define new relations not in the database from relations in the database. The fact that an employee is a senior employee if he earns more than 40K can be expressed by the rule

$C15$: EMP(X, Y, Z) $\wedge$ GREATER(Y, 40K)
$\rightarrow$ SEN_EMP(X)

Since NAME is a common attribute to both EMP and DEPT, the names in DEPT whose corresponding salaries in EMP are greater than 40K are also senior employees. This information is made explicit by the following rule.

$C16$: DEPT$(X, Y, Z) \wedge$ EMP(X, U, V)
$\wedge$ GREATER$(Y, 40K) \rightarrow$ SEN_EMP(X)

Alternative definitions can be given for senior employees. For instance, senior employees can also be defined to be those employees who work in project $P2$. This is expressed by the rule

$C17$: DEPT$(X, Y, Z) \wedge$ EQUAL(Z, P_2)
$\rightarrow$ SEN_EMP(X)

Since NAME is a common attribute to both EMP and DEPT, the names in EMP whose project in DEPT is $P2$ are also senior employees. Therefore the insertion of the rule $C17$ necessitates the insertion of the following rule.

$C18$: DEPT$(X, Y, Z) \wedge$ EMP(X, U, V)
$\wedge$ EQUAL$(Z, P_2) \rightarrow$ SEN_EMP(X)

Classification levels can also be assigned to data not in the database. The following rule $C19$ assigns a top-secret classification level to all senior employees.

$C19$: SEN_EMP$(X) \rightarrow$ LEVEL$(X$, TOP-SECRET$)$

This means that all employees who earn more than 40K or who work in project P_2 are assigned top secret classification levels. This results in the generation of the following rules $C20$–$C23$

$C20$: EMP$(X, Y, Z) \wedge$ GREATER$(Y, 40K)$
$\rightarrow$ LEVEL$(X$, TOP_SECRET$)$

$C21$: EMP$(X, Y, Z) \wedge$ DEPT(X, U, V)
$\wedge$ GREATER$(Y, 40K)$
$\rightarrow$ LEVEL$(X$, TOP_SECRET$)$

$C22$: DEPT$(X, U, V) \wedge$ EQUAL(U, P_2)
$\rightarrow$ LEVEL$(X$, TOP_SECRET$)$

$C23$: EMP$(X, Y, Z) \wedge$ DEPT(X, U, V)
$\wedge$ EQUAL$(U, P_2) \rightarrow$ LEVEL$(X$, TOP_SECRET$)$

The insertion of the above rules may cause data to have conflicting classification levels. For example, along with the above rules, if rule $C2$ is enforced, then an employee may have secret as well as top-secret classification levels. As explained earlier, there are two possible options. One is to prohibit the insertion of the constraint $C19$, and the other option is to assign the top-secret classification to the data, and to update the constraints that resulted in the lower classification level.

3.2 Query Evaluation

The difference between non-secure DBMS and secure DBMS in processing queries is the modification of the queries according to the security constraints. Once the query is modified, standard techniques can be used to evaluate queries expressed in relational algebra. However, relational algebra itself is not sufficient to express all the queries. For example, the query to retrieve all senior employees cannot be expressed in relational algebra. Furthermore, since all the rules are expressed in logic, the expression of queries in relational algebra would necessitate the translation of these queries into logic. Therefore, in our approach we express queries in first order logic and use the rules to modify the query. This modification is performed by the inference engine. Then techniques such as those investigated in [5] can be applied to translate the modified queries into relational algebra and evaluate them against the relational database.

There are two types of queries that can be evaluated. The first type are those queries whose evaluation does not necessitate any database accesses, i.e. the responses to these queries can be generated by accessing only the rule base. An example of such a query is 'retrieve the father of John' and the rule base contains the rule 'Peter is the father of John'. The answer to this query is 'PETER' and will be released provided there is no security violation. This is the technique that is used in the first approach to view databases through logic where all the facts in the database are encoded as rules. The second type are those queries whose evaluation involves accesses to the database. Since our main concern is the design of a secure database management system, we shall only consider queries of the second type in the ensuing discussion.

The graphical representation of the query processing strategy in an augmented database is depicted in *Fig. 3*. The query posed in first order logic is modified in such a way that if this modified query is posed and there is no Trojan Horse in the system, then the response that is released by the DBMS will be properly classified at the level of the user who posed the query. The modified query

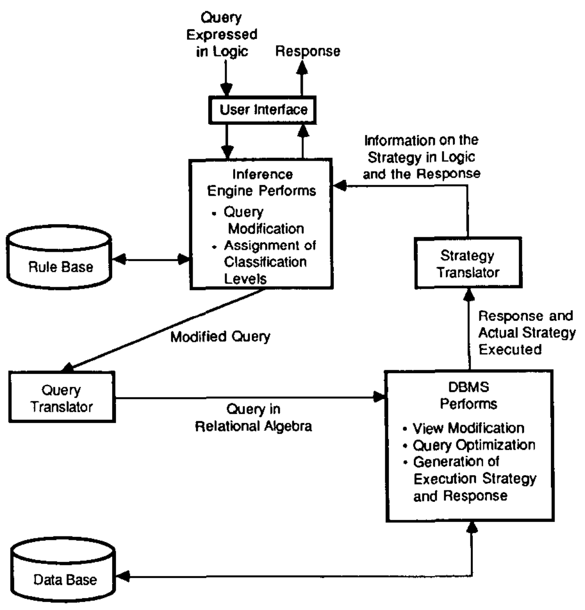
Query Expressed in Logic
Response
User Interface
Information on the Strategy in Logic and the Response
Inference Engine Performs
• Query Modification
• Assignment of Classification Levels
Rule Base
Strategy Translator
Response and Actual Strategy Executed
Modified Query
Query Translator
Query in Relational Algebra
DBMS Performs
• View Modification
• Query Optimization
• Generation of Execution Strategy and Response
Data Base

Fig. 3. Query processing in an augmented database.

is translated into relational algebra using some of the existing techniques. The translated query is evaluated against the relational database. While generating the response to the query, the DBMS also assembles the actual strategy that was executed which consists of the files that were opened and read, and the operations performed on these files. Due to Trojan Horse threats, this actual strategy may not be the correct strategy that should be executed for the corresponding query. We actual strategy along with the attribute names corresponding to the files in the strategy, retrieved from the data dictionary, is passed to the response translator which translates this information into formulas in first order logic. The inference engine by accessing the rule base eventually assigns a classification level to the response released by the DBMS. This response is released into the appropriate environment.

A crucial issue that has to be addressed in the design of the query processing strategy, is the amount of code that has to be verified to protect against the operation of Trojan Horses. Our objective is to minimize the expensive process of verification. The verification issues involved in the design of a standard MLS/DBMS has been investigated in [3]. In this design, when a query is evaluated, the verification process is deliberately delayed because once the verification process is terminated, the remaining events should not contain any hostile code that could damage the results. If this philosophy is adapted to augmented databases, then it is sufficient to commence the verification only when the actual strategy that is executed by the DBMS is assembled. Then the remaining events of translating the strategy into logic, and the assignment of classification level to the response by the inference engine should also be verified. It should be noted that not verifying the query modification process may result in the incorrect modification of the query and consequently cause incorrect information to be present in the response. However, the verification of the assembly of the actual strategy, the translation of the strategy, and the assignment of classification level to response will ensure that the information in the response will not lead to any violation of security. The two additional algorithms that have been devised due to the augmented database approach to security are the query modification algorithm and the classification level assignment algorithm. These algorithms will be described below.

3.2.1 *Query Modification Algorithm*

Step 1. Let a query Q be posed by a user at a level L. Let $X_1, X_2, \ldots, X_n$ be the attributes that are requested to be released in the query. At the user interface this query will be extended and will be expressed as

$$Q \wedge \text{RELEASE}(X_1, L) \wedge \text{RELEASE}(X_2, L) \wedge \ldots \wedge \text{RELEASE}(X_n, L).$$

Let the above expression be P.

Step 2. Examine the rules in the universe which lowers the classification level of any information by releasing one or more of the X_i's at the level L. In other words, search for the existence of the following two rules

$$A(X_1, X_2, \ldots, X_n, Y) \wedge B(T_1, T_2, \ldots, T_k, Y) \wedge \text{RELEASE}(Z_1, L) \wedge \text{RELEASE}(Z_2, L) \wedge \ldots \wedge \text{RELEASE}(Z_m, L) \rightarrow \text{LEVEL}(Y, L)$$

$$B(T_1, T_2, \ldots, T_k, Y) \rightarrow \text{LEVEL}(Y, L_1) \wedge \text{DOMINATE}(L_1, L)$$

where $m \leqslant n$, each $Z_i (1 \leqslant i \leqslant m)$ belongs to $\{X_1, X_2, X_3, \ldots, X_n\}$, A is a relationship between $X_1, X_2, \ldots, X_n, Y$ and B is a relationship between $T_1, T_2, \ldots, T_k, Y$. If there are such rules, then the query will not be posed.

Step 3. If the condition in Step 2 is not satisfied, from the information in the rule base derive a formula of the form $R \rightarrow P$ where R consists of only relations in the database.

Step 4. If such a formula can be derived, then R is the modified query. This modified query is translated and evaluated. Also insert the following rule into the environment at the level L

$$\rightarrow R \wedge \text{RELEASE}(X_1, L) \wedge \text{RELEASE}(X_2, L) \wedge \ldots \wedge \text{RELEASE}(X_n, L)$$

where $\text{RELEASE}(X_i, L)$ implies that attribute X_i can be released at level L.

Step 5. If such formula cannot be derived, i.e. if a derivation of

$$\rightarrow (\text{NOT } P)$$

can be obtained, then no response should be released to the user, and therefore the query will not be evaluated. It should be noted that since each data in the database or real world is assigned a classification level, either a formula of the form $R \rightarrow P$, or $\rightarrow (\text{NOT } P)$ can be derived.

3.2.2 Classification Level Assignment Algorithm

Step 1. The rule that is passed to the inference engine from the response translator will be of the form

$$\rightarrow R(Y_1, Y_2, \ldots, Y_m) \wedge \text{RESPONSE}(X_1, X_2, X_3, \ldots, X_n)$$

where R is a conjunction of some relations in the database, $Y_1, Y_2, \ldots, Y_m$ are the attributes in these relations, $m \geqslant n$, each X_i $(1 \leqslant i \leqslant m)$ belongs to $\{Y_1, Y_2, \ldots, Y_m\}$, and $\text{RESPONSE}(X_1, X_2, \ldots, X_n)$ implies that $X_1, X_2, \ldots, X_n$ are all the attributes in the response generated by the DBMS.

Step 2. The inference engine will attempt a derivation of the following formula

$$\rightarrow R(Y_1, Y_2, \ldots, Y_m) \wedge \text{RELEASE}(X_1, L) \wedge \text{RELEASE}(X_2, L) \wedge \ldots \wedge \text{RELEASE}(X_n, L)$$

for some classification level L.

Step 3. Such a formula can be derived because any data that is not assigned an explicit classification level will default to the unclassified level. The response is then assigned the classification level L, and the following rule is inserted into $E(L)$.

$$\rightarrow R(Y_1, Y_2, \ldots, Y_m) \wedge \text{RELEASE}(X_1, L) \wedge \text{RELEASE}(X_2, L) \wedge \ldots \wedge \text{RELEASE}(X_n, L)$$

Therefore any user at level L will be able to see the response.

We will now state and prove a theorem which guarantees the correctness of the query modification algorithm.

Theorem:

For a given query Q posed at a level L, the query modification algorithm will produce a modified query Q^* in such a way that if this modified query is posed, the DBMS will release a response that is properly classified at a level L.

Proof:

To make the proof simpler, we will assume that there is no Trojan Horse in the system. The proof is by contradiction. Suppose the response S is not properly classified at level L. Then the following condition holds.

$$\exists L_1 > L, \text{CLASSIFICATION}(\text{INFER}(S, E(L))) = L_1$$

We will prove that this condition is never true. Let Y be the attribute (or attributes) inferred from $(S, E(L))$. Let the attributes in $(S, E(L))$ that resulted in Y be $X_1, X_2, \ldots, X_n$. Then $X_1, X_2, \ldots, X_n, Y$ are in some relationship which is denoted say by A. Also the universe would have had the following two rules before the query Q was posed.

$$A(X_1, X_2, \ldots, X_n, Y) \wedge B(T_1, T_2, \ldots, T_k, Y) \wedge \text{RELEASE}(X_1, L) \wedge \text{RELEASE}(X_2, L) \wedge \ldots \wedge \text{RELEASE}(X_n, L) \rightarrow \text{LEVEL}(Y, L)$$

$B(T_1, T_2, \ldots, T_k, Y) \rightarrow \text{LEVEL}(Y, L_1) \wedge \text{DOMINATE}(L_1, L)$

In Step 2 of the algorithm, it would have been ensured that the query is never posed. This contradicts the assumption that the query is not properly classified at level L.

We will now illustrate the techniques with examples. First we will assume that there is no Trojan Horse in the system and consider the evaluation of the following four queries all posed by unclassified users.

$Q1$. Retrieve names in EMP and the constraint $C2$ is enforced.

$Q2$. Retrieve all senior employees and the constraint $C2$ is enforced.

$Q3$. Retrieve all salaries from EMP and the constraints that are enforced are 'The names in EMP are secret' and 'If the salaries in EMP are released, then the names are unclassified'.

$Q4$. First retrieve all names, and then retrieve all salaries, and the constraint $C4$ is enforced.

The queries $Q1$, $Q2$ and $Q3$ are single queries. $Q1$ involves only relations in the database, whereas $Q2$ does not. No query will be evaluated in processing the query $Q3$ because the condition in Step 2 of the query modification algorithm will be satisfied. The query $Q4$ is an example of a multiple query. It should be noted that due to the fact that there is no Trojan Horse in the system, it suffices to explain only the query modification algorithm for these four examples. We will then relax our assumption and evaluate the following query.

$Q5$. Retrieve all social security numbers from EMP and the constraint C is enforced.

In evaluating the query $Q5$, the query modification algorithm as well as the classification level assignment algorithm will be applied.

3.2.3 Evaluation of the Query Q1

This query is expressed by the rule

$C24{:} \rightarrow \text{EMP}(X, Y, Z) \wedge \text{RELEASE}(X, \text{UNCLASSIFIED})$

Note that since only the names have to be retrieved, we assume that the variables Y and Z are existentially quantified although we have not explicitly represented it here. In other words the response that is returned is the set

$\{X{:}\exists Y, Z\ \text{EMP}(X, Y, Z) \wedge \text{RELEASE}(X, \text{UNCLASSIFIED})\}$

Note also that the sub-formula RELEASE(X, UNCLASSIFIED) will be inserted by the user interface to the query $\{X{:}\exists Y, Z\ \text{EMP}(X, Y, Z)\}$. The formula in $C24$ is the expression P. The condition in Step 2 of the algorithm is not satisfied and therefore a derivation of the formula $R \rightarrow P$ will be attempted as follows:

$C25{:}\ \text{EMP}(X, Y, Z) \wedge \text{NOT}(\text{GREATER}(Y, 50\text{K})) \rightarrow \text{LEVEL}(X, \text{UNCLASSIFIED}) \times \{C2, C12\}$

$C26{:}\ \text{EMP}(X, Y, Z) \wedge \text{NOT}(\text{GREATER}(Y, 50\text{K})) \rightarrow \text{EMP}(X, Y, Z) \wedge \text{LEVEL}(X, \text{UNCLASSIFIED}) \times \{C26\}$

$C27{:}\ \text{EMP}(X, Y, Z) \wedge \text{NOT}(\text{GREATER}(Y, 50\text{K})) \rightarrow \text{EMP}(X, Y, Z) \wedge \text{RELEASE}(X, \text{UNCLASSIFIED}) \times \{C27, C14\}$

Therefore the modified query is

$C28{:} \rightarrow \text{EMP}(X, Y, Z) \wedge \text{NOT}(\text{GREATER}(Y, 50\text{K}))$

and the following rule will be inserted into the unclassified environment.

$C29{:} \rightarrow \text{EMP}(X, Y, Z) \wedge \text{NOT}(\text{GREATER}(Y, 50\text{K})) \wedge \text{RELEASE}(X, \text{UNCLASSIFIED})$

3.2.4 Evaluation of Query Q2

This query is expressed by the rule

$C30{:} \rightarrow \text{SEN\_EMP}(X) \wedge \text{RELEASE}(X, \text{UNCLASSIFIED})$

Assume that senior employees are defined by the rule $C12$. Then the query is equivalent to:

$C31{:} \rightarrow \text{EMP}(X, Y, Z) \wedge \text{GREATER}(Y, 40K) \wedge \text{RELEASE}(X, \text{UNCLASSIFIED})$

The derivation of the query to be posed so that the information returned is observable to the user is as follows:

$C32{:}\ \text{EMP}(X, Y, Z) \wedge \text{NOT}(\text{GREATER}(Y, 50\text{K})) \rightarrow \text{LEVEL}(X, \text{UNCLASSIFIED})\{C2, C12\}$

$C33$: EMP$(X, Y, Z) \wedge$ GREATER$(Y, 40\mathrm{K})$
$\wedge$ NOT(GREATER$(Y, 50\mathrm{K})$)
$\rightarrow$ EMP(X, Y, Z)
$\wedge$ GREATER$(Y, 40K)$
$\wedge$ LEVEL$(X$, UNCLASSIFIED$)\{C33\}$

$C34$: EMP$(X, Y, Z) \wedge$ GREATER$(Y, 40\mathrm{K})$
$\wedge$ NOT(GREATER$(Y, 50\mathrm{K})$)
$\rightarrow$ SEN_EMP(X)
$\wedge$ RELEASE$(X$, UNCLASSIFIED$)\{C34, C16\}$

Therefore the modified query is

$C35$: $\rightarrow$ EMP$(X, Y, Z) \wedge$ GREATER$(Y, 40\mathrm{K})$
$\wedge$ NOT(GREATER$(Y, 50K)$)

and the following rule will be inserted into the unclassified environment.

$C36$: $\rightarrow$ EMP$(X, Y, Z) \wedge$ GREATER$(Y, 40K)$
$\wedge$ NOT(GREATER$(Y, 50\mathrm{K})$)
$\wedge$ RELEASE$(X$, UNCLASSIFIED$)$

3.2.5 Evaluation of Query Q3

The query will be expressed as

$C37$: $\rightarrow$ EMP(X, Y, Z)
$\wedge$ RELEASE$(Y$, UNCLASSIFIED$)$

The condition in step 2 of the algorithm will be satisfied due to the two constraints expressed by the following two rules.

$C38$: $\rightarrow$ EMP$(X, Y, Z) \wedge$ LEVEL$(X$, SECRET$)$

$C39$: EMP$(X, Y, Z) \wedge$ RELEASE$(Y$, UNCLASSIFIED$)$
$\rightarrow$ LEVEL$(X$, UNCLASSIFIED$)$

Therefore the query will not be posed.

3.2.6. Evaluation of Query Q4

Since names and salaries only when taken together are secret, if one of them has been released to an unclassified user, then the other will be assigned a secret level. This is represented by the rules C5 and C6.

Suppose an unclassified user wants to retrieve all names. The query posed will be expressed by the rule $C24$. The derivation of the formula $R \rightarrow P$ is as follows:

$C40$: EMP(X, Y, Z)
$\rightarrow$ EMP$(X, Y, Z) \wedge$ LEVEL$(X$, UNCLASSIFIED$)$
$\times\{C12\}$

$C41$: EMP(X, Y, Z)
$\rightarrow$ EMP(X, Y, Z)
$\wedge$ RELEASE$(X$, UNCLASSIFIED$)$
$\times\{C39, C13\}$

Therefore the query that will be evaluated is

$C42$: $\rightarrow$ EMP(X, Y, Z)

and all the names will be released. The rule $C24$ will be inserted into the unclassified environment.

Suppose at a later time the unclassified user requests all salaries from EMP. The query that will be posed is expressed by rule $C37$.

Rule $R \rightarrow P$ cannot be derived as (NOT P) can be deduced as follows:

$C43$: $\rightarrow$ EMP(X, Y, Z)
$\wedge$ RELEASE$(Y$, SECRET$)\{C41, C5\}$

$C44$: $\rightarrow$ NOT(EMP(X, Y, Z)
$\wedge$ RELEASE$(Y$, UNCLASSIFIED$)$)$\{C43\}$

Therefore the query to retrieve the salaries will not be evaluated.

Note that if a secret user had requested for the salaries, the query posed will be expressed by rule $C43$ and the salaries will be released as a derivation of a rule of the form $R \rightarrow P$ can be obtained as follows:

$C45$: EMP(X, Y, Z)
$\rightarrow$ EMP(X, Y, Z)
$\wedge$ LEVEL$(Y$, SECRET$)\{C43\}$

$C46$: EMP(X, Y, Z)
$\rightarrow$ EMP(X, Y, Z)
$\wedge$ RELEASE$(Y$, SECRET$)\{C45, C13\}$

The rule $C43$ will be inserted into the secret environment.

If a top-secret user requests salaries, the query posed will be

$C47$: $\rightarrow$ EMP$(X, Y, Z) \wedge$ LEVEL$(Y$, TOP_SECRET$)$

and the formula of the form $R \rightarrow P$ can be derived as follows.

$C48$: EMP(X, Y, Z)
$\rightarrow$ EMP(X, Y, Z)
$\wedge$ RELEASE$(Y$, TOP_SECRET$)\{C45, C13\}$

Therefore the salaries will be released and rule $C47$ will be inserted into the top-secret environment.

3.2.7 Evaluation of Query Q5

The query will be expressed as

$C49: \rightarrow \text{EMP}(X, Y, Z) \wedge \text{RELEASE}(Z, \text{UNCLASSIFIED})$

Case 1: Somewhere in the process the query is incorrectly modified to retrieve all names from EMP.

Since the names in EMP are unclassified, this query is translated and evaluated. The DBMS will generate all names in EMP. From the information maintained in the data dictionary the DBMS will retrieve the attribute name corresponding to the file that was opened, and pass this information to the response translator. The response translator will generate and pass the following rule to the inference engine.

$C50: \rightarrow \text{EMP}(X, Y, Z) \wedge \text{RESPONSE}(X)$

The inference engine will attempt a derivation of the following rule.

$C51: \rightarrow \text{EMP}(X, Y, Z) \wedge \text{RELEASE}(X, L)$

and it will succeed in deducing rule $C24$.

The names released by the DBMS will be assigned an unclassified classification level and rule $C24$ will be inserted into the unclassified environment. It should be noted that the information that is returned to the user is not what he requested. But the user will not acquire any information to which he is not authorized.

Case 2: Somewhere in the process the query is incorrectly modified to retrieve all salaries in EMP.

The DBMS will generate the salaries in the response. Although the salaries are secret, since the unclassified user posed the original query, the response translator will pass the following rule to the inference engine.

$C52: \rightarrow \text{EMP}(X, Y, Z) \wedge \text{RESPONSE}(Y)$

The inference engine will attempt a deduction of the rule

$C53: \rightarrow \text{EMP}(X, Y, Z) \wedge \text{RELEASE}(Y, L)$

and will succeed in deriving rule $C43$.

Therefore the response will be assigned a secret level and rule $C43$ will be inserted into the secret environment. The unclassified user who posed the query will not acquire any information.

3.3 Future Considerations

In this paper we have presented a design for a multilevel augmented relational DBMS. In particular we have addressed the issues on the management of the information in the rule base, and the operation of the inference engine for query processing. In addition to enhancing the techniques described in this paper, some of the other issues that are currently being addressed are the evaluation of recursive queries in MLS/DBMS and handling of updates in a database. Recursive queries are recognized as being of considerable importance to knowledge based systems. Even in non secure database management systems, although strategies have been proposed to evaluate recursive queries, no method is yet known to choose from these algorithms [1].

Our security policy for read only databases has to be extended to include the read/write databases. To accomplish this, in addition to the policy for queries, we need to formulate a policy for updates, and devise algorithms to implement the policy. The main problem that has to be overcome in the case of updates is the covert channel attacks where sensitive information is encoded in otherwise benign fields. Such attacks are prevented if the policy for update ensures that higher level information is never written into a lower classification level. However it is extremely difficult to build systems which satisfy this policy at all times [9]. We need to address the issue as to whether the benefits gained by augmenting databases with inference engines in handling queries may cause additional problems in handling updates.

The successful design and implementation of secure database management systems are crucial as computerized information is a critical resource in numerous enterprises, whether industrial, commercial or government, and compromise to information security is a serious problem that faces the world today. The ideas presented in this paper provide a sound theoretical foundation upon which practical solutions to the design of MLS/DBMS can be provided.

References

[1] F. Bancilhon and R. Ramakrishnan: "An Amateur's Introduction to Recursive Processing Strategies", *Proceedings of ACM Sigmod '86*, pp. 16–52.

[2] W.E. Boebert, B.B. Dillaway and J.T. Haigh: "Mandatory Access Control Policies", *DBMS Security Workshop*, Baltimore, MD, June 1986.

[3] W.E. Boebert, B.B. Dillaway, P.A. Dwyer, J.T. Haigh, R.Y. Kain, M.B. Thuraisingham and W.D. Young: "Design of a Multilevel Secure Database Management System Hosted on a Trusted Computing Base", To be submitted.

[4] M. Brodie, J. Mylopoulos and J. Schmidt: *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Springer-Verlag, 1984.

[5] S. Ceri, G. Gottlob and L. Vavazza: "Translation and Optimization of Logic Queries: The Algebraic Approach", *Proceedings of the 12th International Conference on Very Large Databases*, Kyoto, Japan, 1986, pp. 395–419.

[6] G. Chaitin: "A Theory of Program Size Formally Identical to Information Theory", Vol. 22, #3, 1975, pp. 329–339.

[7] D.E. Denning: "A Lattice Model of Secure Information Flow", *Communications of the ACM*, May 1976, pp. 236–243.

[8] D.E. Denning, S.K. Akl, M. Morgenstern, P.G. Neumann, R.R. Schell and M. Heckman: "Views for Multilevel Database Security", *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, April 1986, pp. 156–172.

[9] B.B. Dillaway and J.T. Haigh: "A Practical Design for a Multilevel Secure Database Management System", *Aerospace Security Conference*, December 1986.

[10] P.A. Dwyer, G.D. Jelatis and M.B. Thuraisingham: "Multilevel Security in Database Management Systems", *Computers & Security*, Vol. 6, Nr. 3, 1987.

[11] H. Gallaire and J. Minker: *Logic and Databases*, Plenum Press, 1978.

[12] R. Kowalski and M. Sergot: "A Logic-based Calculus of Events", *New Generation Computing*, Vol. 4, 1986, pp. 67–95.

[13] J. Lloyd: *Foundations of Logic Programming*, Springer Verlag, 1984.

[14] E. Mendleson: *Introduction to Mathematical Logic*, Van Nostrand, 1978.

[15] J.M. Nicholas and K. Yazdanian: "Integrity Checking in Deductive Databases", *Logic and Databases*, Plenum Press, 1978, pp. 325–344.

[16] J.A. Robinson: "A Machine-Oriented Logic Based on the Resolution Principle", *JACM*, Vol. 12, #1, 1965, pp. 23–41.

[17] M. Stonebraker: "Implementation of Integrity Constraints and Views by Query Modification", *ACM National Conference Proceedings*, 1974, pp. 180–186.

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