Secure Query-Processing Strategies

T.F. Keefe, University of Minnesota
M.B. Thuraisingham, Honeywell
W.T. Tsai, University of Minnesota

A multilevel secure database management system is a system that is secure when shared by users from more than one clearance level and contains data of more than one sensitivity level. MLS/DBMSs evolved from multilevel secure computing systems. Present-day DBMSs are not built with adequate controls and mechanisms to enforce a multilevel security policy. Thus, an MLS/DBMS is different from a conventional DBMS in at least the following ways:

1. Every data item controlled by an MLS/DBMS is classified in one of several sensitivity levels that may need to change with time.
2. Access to data must be controlled on the basis of each user's authorization to data at each sensitivity level.

Providing an MLS/DBMS on current secure computing systems presents a new set of problems. The most obvious 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 unique to databases is the necessity to classify data according to context, time, and aggregation. Furthermore, DBMSs are vulnerable to inference attacks—that is, users can infer unauthorized data from the knowledge they have accumulated.

The Air Force Summer Study of 1982 proposed various designs for multilevel secure relational database management systems. One MLS/RDBMS proposal is a near-term set of requirements incorporating off-the-shelf concepts; another is a longer-term set of requirements, including content, context, dynamic classifications, and solutions to the inference problem.

There has been a lot of interest in producing designs for MLS/RDBMSs. Much attention is also being paid to the foundations of the inference problem in database security. Other work has dealt with particular aspects of the inference problem, such as architectures for MLS/DBMSs and query-processing strategies.

The strategies for secure query processing that we propose here are carried out by query modification, a technique that has been used for enforcing integrity constraints and providing view mechanisms. The technique consists of replacing the query the user presents with one that, when evaluated, will perform the desired function. In the case of a view mechanism, the names of views referenced in the query are replaced by the definitions of the views in terms of base relations.

Security models for MLS/DBMSs are still a research issue. Once an appropriate security model is chosen, a DBMS that adheres to it must be built. The correspondence between the model and the DBMS is demonstrated by verification. To allow an effective verification of the security properties of the DBMS, the security-relevant portion of the code must be small. Query modification allows the security function to be separated from the normal functions of a DBMS. It allows one to concentrate on the security properties of the
A previous approach to secure query processing

Query modification techniques have been used extensively in the past to transform queries into a more appropriate format for a particular application. In deductive databases, queries against virtual relations, or relations that are not computed from rules in the rule base, have been transformed into queries against base relations. Query modification has been used in a discretionary access control mechanism. A variation of this technique was proposed for mandatory access control.

In this section we will describe this query modification mechanism and some of its limitations. In essence, the user's query is modified by applying the relevant security constraints for the particular query in such a way that if the modified query is posed, then the response generated by the DBMS will not result in any security violation. An alternate approach to secure query processing would be not to modify the query but to check whether the response to be released will violate security. If it will, the response is not released. For this approach, methods of database integrity maintenance such as Sadri and Kowalski's could be adapted.

Before illustrating the query modification technique with examples, we must first describe security constraints. Security constraints assign classification levels to all data in the database. They provide a 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 classification of the entire database, as well as classification by relation and by attribute. Content-based constraints provide a classification based on the contents of the database and allow for classification by tuple and by element. Context-based constraints classify relationships between data. Finally, the classification levels of the data can change dynamically with changes in content or context.

A constraint consists of a data specification and a classification. The data specification defines a subset of the database that can be expressed with relational algebra. The classification defines the classification level of this subset. For example, consider a database that consists of a relation Emp(SS#, Name, Salary, Dept#) with SS# as the key. The content-based constraint that classifies the names of all employees who earn more than 100K ($100,000 a year) as Secret is expressed as

Level(PROJECT[Name] Emp) = Secret

and the context-based constraint that classifies all names and salaries taken together as Secret is expressed as

Level(PROJECT[Name, Salary] Emp) = Secret

The simple constraints that classify all names and salaries in Emp taken individually as Unclassified are expressed as

Level(PROJECT[Name] Emp) = Unclassified

Level(PROJECT[Salary] Emp) = Unclassified

Simple and content-based constraints classify elements of tuples in the database and therefore can be applied to data as it is actually stored in the database. In this case, each element of data is associated with its sensitivity level. Context-based constraints classify sets of elements in the database and can only be applied in the computation of the result that is to be output in response to a user's query. For example, if the content-based constraint that classifies all names of employees who earn more than 100K as Secret is enforced, the Secret names are stored in a Secret file and the Unclassified names are stored in an Unclassified file. If the context-based constraint that names and salaries taken together are Secret is enforced, the names and salaries are stored in Secret files and the constraint allows the information to be downgraded in appropriate situations.

The security constraints are used in evaluating queries as follows: Suppose an unclassified user wants to retrieve all the names from the relation Emp, and the content-based constraint that classifies all names of employees who earn more than 100K is enforced. Then the query is modified to retrieve all names of which the corresponding salaries are less than or equal to 100K. This modified query is then evaluated and all the Unclassified names will be retrieved. The formal description of this query modification algorithm appears in an article by Dwyer, Jelatis, and Thuraisingham.

Although the query modification algorithm modifies the query correctly, it does not specify how to extract the relevant constraints. In some cases, a relation may have numerous constraints enforced on it. In addition, constraints enforced on Join operations are rather complex. For example, consider the following relations:

Emp(SS#, Name, Salary, Dept#)

Dept(Dept#, Project, DName, Mgr)

SS# is the key of Emp, and Dept# is the key of Dept. We enforce the constraint that classifies all names in Emp who work in a particular project—say Stars—in Dept as Secret. This constraint necessitates the Join of the relations Emp and Dept. Such constraints necessitate an efficient search procedure to extract the relevant constraints for a query.

Another drawback of the query modification algorithm is that it takes into consideration only the response to the current query. It does not maintain the history information, making it impossible to determine whether the response to be generated, taken together with previously released responses, might lead to security violations. In other words, the algorithm does not handle inference attacks. To overcome such attacks, the information in all responses released must be maintained. Furthermore, for each additional query some efficient computation technique is required to see if the response generated, combined with previous responses, will cause a breach in security.

In the next section we will explore the representation of constraints and the representation of histories of released information, along with methods for manipulating them to process queries securely.

Strategies for secure query processing

There are two important concerns about secure query-processing strategies. The first is the representation of the information required for processing the query. This information consists of security constraints, including simple, content-based, and context-based types, and environmen-
tal state information concerning what data has been released and at what level. The second concern is efficient methods of manipulating this information—for example, methods for finding constraints relevant to a given query and methods for constructing the modified query.

**Basic strategy.** Most existing relational database management systems use the relational model to represent metadata. Therefore, we will first examine the suitability of such a model for the representation of metadata for an MLS/DBMS. The metadata of an MLS/DBMS includes the external schemas that describe the views, the conceptual schemas that describe the relations and the attributes, the internal schemas that describe the physical files, the security constraints, and the integrity constraints.

We will illustrate our design of the data dictionary, which contains the metadata, with an example. Suppose the database consists of the relations Emp and Dept described earlier. Let the following security constraints be enforced:

<table>
<thead>
<tr>
<th>Rname</th>
<th>Aname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>SS#</td>
</tr>
<tr>
<td>Dept</td>
<td>Dept#</td>
</tr>
</tbody>
</table>

Table 1. Relation definitions.

<table>
<thead>
<tr>
<th>Rname</th>
<th>Arity</th>
<th>Cardinality</th>
<th>Primary Key</th>
<th>Foreign Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>4</td>
<td>0</td>
<td>SS#</td>
<td>Dept#</td>
</tr>
<tr>
<td>Dept</td>
<td>4</td>
<td>0</td>
<td>Dept#</td>
<td>Null</td>
</tr>
</tbody>
</table>

Table 2. Attribute definitions.

<table>
<thead>
<tr>
<th>View Name</th>
<th>View Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>PROJECT [SS#, Name, Salary, Dept#, Project, Dname, Mgr] (Emp JOIN Dept)</td>
</tr>
<tr>
<td>V2</td>
<td>PROJECT [Name, Salary] (Emp)</td>
</tr>
<tr>
<td>V3</td>
<td>PROJECT [Dname, Mgr] (Dept)</td>
</tr>
</tbody>
</table>

Table 3. View definitions.

<table>
<thead>
<tr>
<th>Rname</th>
<th>Cname</th>
<th>Condition</th>
<th>Target</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>C2</td>
<td>Null</td>
<td>Name</td>
<td>Secret</td>
</tr>
<tr>
<td>Dept</td>
<td>C2</td>
<td>Null</td>
<td>Salary</td>
<td>Secret</td>
</tr>
<tr>
<td>Dept</td>
<td>C4</td>
<td>Dept# = D1</td>
<td>Dname</td>
<td>Secret</td>
</tr>
<tr>
<td>Dept</td>
<td>C4</td>
<td>Dept# = D1</td>
<td>Mgr</td>
<td>Secret</td>
</tr>
</tbody>
</table>

Table 4. Simple and content-based constraints.

The representation of this metadata is shown in Tables 1 through 5. Table 1 describes the relations, the arity of each relation, the number of tuples in each relation, one primary key, and one foreign key. Table 2 describes the attributes in each relation. Table 3 describes the views. Table 4 describes the content-based and simple constraints. Note that a simple constraint is a content-based constraint with no condition. Table 5 describes the context-based constraints. We have not described the internal schemas as they are not required for the query modification process.

The proposed strategy has two major weaknesses. The first is that it doesn’t maintain any environmental state information. To examine this weakness, let’s suppose an unclassified user wants to obtain all names. Tables 4 and 5 will have to be searched for all the relevant constraints. The relevant constraints in this case are C1, C2, and C3. Furthermore, C1 and C3 are directly relevant, and the query first will be modified to retrieve all names of employees whose salary is less than 100K and who do not work in the project Stars. The constraint C2 becomes directly relevant only if the salaries have been released earlier. But this information is not represented in the model.

The second weakness of this strategy is its difficulty in expressing complex security classifications. We encounter this difficulty in representing constraints in which the sensitivity level of one attribute depends on the sensitivity level of another. In the previous example, suppose we have another constraint, which classifies salary as Secret only if Dept# in Emp is Secret. If an unclassified user wants to retrieve the
salaries, then the query is first modified to retrieve all salaries for which the predicate Dept#,Secret is False. But this modified query cannot yet be evaluated against the relational database. The tables have to be searched again to get all the constraints that classify the department number as Secret to provide a definition for the Dept#,Secret predicate.

**Strategy 2: adding environmental information.** Here we will build on the basic strategy and add the environmental information it lacked. We will express this strategy using logic. Logic can be used in two ways: to express a computation and to imply a method of execution. We use logic in both ways. First, we express the computation performed by the strategy. Second, we analyze the performance of the strategy, assuming the implied method of execution. We assume a method of execution based on a technique proposed by Chang for translating queries against virtual relations to queries against base relations.

We will now describe how the constraints defined in the preceding subsection can be expressed in logic. A logical formula is of the form A → B, meaning A implies B where A is called the premise and B the conclusion. Each base relation (a relation in the database) or virtual relation (a relation not stored in the database) is represented by a predicate. Therefore, the constraints C1 through C5 are expressed by the following rules:

- **R1:** Emp(X,Y,Z,W) \( A \times Z > 100K \rightarrow \) Level(Y, Secret)
- **R2:** Emp(X,Y,Z,W) \( \rightarrow \) Level([Y,Z], Secret)
- **R3:** Emp(X,Y,Z,W) \& DepP(Q,R,S) \& P \( = \) W \& Q \( = \) Stars \( \rightarrow \) Level(Y, Secret)
- **R4:** DepP(Q,R,S) \& P \( = \) DI \( \rightarrow \) Level([R,S], Secret)
- **R5:** DepP(Q,R,S) \& W \( \rightarrow \) Level(Q, Secret)

The rules R2 and R4 are context-based constraints. They classify sets of attributes released together. Therefore, if one of the attributes has already been released, then the other should be classified at the Secret level. We have found that with the relational model it was not easy to express such constraints. We will now examine how they can be expressed with logic. First we will examine the case where only one variable is released in a query. For this case the constraints R2 and R4 can be expressed as follows:

- **R6:** Emp(X,Y,Z,W) \& Release(Y, Unclassified) \( \rightarrow \) Level(Z, Secret)

R7: Emp(X,Y,Z,W) \& Release(Z, Unclassified) \( \rightarrow \) Level(Y, Secret)

R8: DepP(Q,R,S) \& P \( = \) DI \& Release(R, Unclassified) \( \rightarrow \) Level(S, Secret)

R9: DepP(Q,R,S) \& P \( = \) DI \& Release(S, Unclassified) \( \rightarrow \) Level(R, Secret)

where Release(X,L) implies X has been released at level L.

Suppose an unclassified user wants to retrieve all names from Emp:

- **[Y]:** Emp(X,Y,Z,W)

Instead of modifying the query to obtain all Unclassified names, we pose a query that will first obtain all names, then obtain Secret names, and finally subtract the Secret names from the set of all names. In general, this will give only a subset of the Unclassified names because Name is not a key attribute. The situation could arise in which two different employees share the same name and one makes less than 100K and the other makes more than 100K. According to R1, the name the two employees share is both Secret and Unclassified. However, the subtraction of the two sets would remove the name. The user will get only the Unclassified names.

More specifically, in processing the query, the system searches the rule base for all rules that classify the names in Emp with a sensitivity level higher than the clearance of the user. These rules are R1, R3, and R7. The premises in these rules are conjuncted and the result is used to form a query Q'. This query is as follows:

- **[Y]:** (Emp(X,Y,Z,W) \& A \times Z > 100K) \& Level([Y,Z], Secret)

The query Q' is subtracted from the original to give the Unclassified names. The result is as follows:

- **[Y]:** Emp(X,Y,Z,W) \& Level([A,C], Secret)

This query still cannot be evaluated because Release is not a base relation. Therefore, the rule base will be searched to see if the salaries have been released. Since the salaries have not yet been released, Release(Z, Unclassified) is False. Therefore, the query that will be evaluated is

- **[Y]:** Emp(X,Y,Z,W) \& Level([A,B,C], Secret)
The number of rules is exponential with respect to the number of elements in the classified set. This is a significant weakness of this strategy if there are large sets of attributes involved in context-based security constraints.

The next problem we need to address is how to compute the sensitivity level of the list of attributes being released by a query. The answer is that the level for the set of attributes dominates the level assigned to each subset. For example, if the constraints E1 through E7 are in force, any set of attributes that contains the set \{A, B, C\} will be classified as Secret. If C has been released Unclassified, then any set of attributes that includes \{A, B\} is also Secret. Since each subset must be classified, the process takes exponential time with respect to the number of attributes released by the query.

The examples described above show that unlike the basic strategy, this one can be used fairly easily to represent constraints and environmental information. It allows a natural expression of context-based constraints and classification of elements based on the classification of other elements. However, the method is inefficient in representing context-based constraints and computing the classifications of sets of output attributes.

**Strategy 3: using graphs.** This strategy builds on the previous one in two ways. First, it uses graphs to index security constraints and make access more efficient. Second, it processes each output attribute independently, reducing the difficulty in processing queries with multiple output attributes and reducing the number of rules needed to express context-based constraints. This technique is a variation of the factoring mechanism used for query processing. The essence of the factoring mechanism is to extract the common information present in all the rules to avoid representation of redundant information. For example, if more than one constraint is enforced on the relation Emp, then it is not necessary to repeat the name Emp and its attributes in each constraint. Just one point represents Emp, and the constraints enforced on it originate at this point.

![Figure 1](image-url)

**Figure 1. Representation of constraints with a graph.**

March 1989
relation Dept, and an edge from Project(Q) all meet at the node P = W ∧ Q = Stars. Since the names in project Stars are Secret, there is an edge from Level(Y, Secret) to the node P = W ∧ Q = Stars.

The query is processed in a manner similar to before. For example, if an unclassified user wants to retrieve all names, a search is made for a node that classifies all names at a level greater than or incomparable to the Unclassified level. In this example the node found will be Level(Y, Secret). Then all edges originating from this node will be traversed. The two nodes that will specify the conditions to classify the names are Z > 100K and P = W ∧ Q = Stars. Then these conditions will be negated and the query will be modified to retrieve all names for whom the salary is less than or equal to 100K and who do not work in Stars.

Handling multiple queries. Now let us examine how multiple queries can be handled. Because of the constraint C2, names should not be released to an unclassified user if salaries have been released earlier, and vice versa. In other words, the context-based constraint C2 will automatically enforce the two constraints R8 and R9; names are Secret if salaries have been released, and salaries are Secret if names have been released. The additional information that has to be incorporated into Figure 1 to enforce the constraints R8 and R9 is shown in Figure 3. In this figure we have two more nodes. One is labeled Release(Y, Unclassified), which implies that names are released; the other node is labeled Release(Z, Unclassified), which implies that salaries are released. Because of the constraint R8, there will be an edge from Level(Y, Secret) to the condition Release(Z, Unclassified), and the constraint R9 will be enforced by the edge from Level(Z, Secret) to the condition Release(Y, Unclassified). Now if an unclassified user wants to obtain salaries after he has obtained names, a search will be made for a node that classifies salaries at a level greater than or incomparable to the Unclassified level. The node that will be found is Level(Z, Secret). The condition pointed to by this node is Release(Y, Unclassified). Therefore, the query will be modified to retrieve all salaries if names have not been released.

But this query still cannot be evaluated because the predicate Release is a virtual predicate. Therefore, a search must be made to determine whether names have been released. To do this, we need another graph to represent environmental information. This graph is shown in Figure 4. As soon as the names are released at the Unclassified level from the previous query, this information has to be incorporated into the environmental information. Therefore, the Name(Y) in Figure 3 has an edge whose tail is Release(Y). Since the names are already released at the Unclassified level, the negation of the condition Release(Y) is False. Therefore, the query is modified to the null query and no query will be posed to retrieve the salaries.
Multiple outputs released in a query. Strategy 2 was inefficient in dealing with context-based constraints and the release of multiple outputs in a query. Up to this point we have only sketched the method for handling these problems by graph factoring. In this section we show how sets of outputs can be classified efficiently and how context-based constraints can be represented efficiently.

In this strategy the modification of the query necessary for each output variable is considered independently. As soon as the query is modified, the proper attribute is tentatively marked released in the environmental state graph. Each output is processed in this fashion. The modified query is processed. If the query is a contradiction, then no data is released and the changes to the environmental state graph are removed. If the query does release data to the user, the changes to the environmental state graph are made permanent. For example, suppose an unclassified user attempts to retrieve name and salary in a query. First, the release of names is considered. The condition for releasing names Unclassified is Release(Salary, Unclassified). This predicate is immediately looked up in the environmental state graph and replaced with False or True. This condition is added to the query. At this point the fact that names have been released Unclassified is added tentatively to the environmental state graph. Next, the release of salaries is considered. The condition for releasing them is Release(Name, Unclassified). This is looked up and replaced by True or False. Now this condition is added to the query, and the environmental state graph is again tentatively updated.

This modified query is a contradiction. When it is processed, it will not release any data and so the environmental state graph will be returned to its previous state.

Since each output is processed individually, the classifications for context-based constraints are much simpler. For each object in a context-based constraint, there must be one node with a predicate that is the conjunction of release predicates for each of the other members of the constraint. This means that for a context-based constraint with n members, there will be a total of n + (n - 1) logical terms in all of the predicates. This is much better than the previous strategy’s exponential growth of classification rules with n.

The mechanism described here is also applicable when the level of a piece of data depends on the level of another. Suppose the names in Emp are Secret only if the salaries are Secret. Then the node labeled Level(Y, Secret) has an arrow to the node labeled Level(Z, Secret). The query is then modified to retrieve all names for which the salary is not Secret. This query cannot yet be evaluated as it contains a virtual relation. The graph is searched again for all the conditions that make salary in Emp Secret. This is quite simple—all one has to do is follow all the edges that originate at Level(Z, Secret).

Performance

The performance of a query-processing strategy depends on the modification method being used, the number of security constraints, and the number of attributes released in the query, among other factors. That is the type of performance we focus on in this section. Another measure of performance is the impact of query modification on throughput. This type of performance is affected by such factors as the DBMS architecture (distributed or centralized), the type of queries being made, and the frequency of updates. It is not affected by the particular method used to modify the query. We will not consider that type of performance here.

Strategy 2. Security constraints in the second strategy are represented as rules. The method described by Chang for identifying relevant queries requires that each literal in the query be unified with each security constraint. Performance is directly related to the number of security constraints. Thus, performance may be poor because there can be a large number of security constraints. Performance is degraded further due to the way that context-based constraints are represented. As discussed earlier, the number of security constraint rules necessary to represent a context-based constraint is exponential to the number of objects classified by the constraint.

The performance of constraint location is determined by the method of classification used for sets of output variables. The number of conditions added to the modified query is exponential to the number of variables being simultaneously queried. For the query

\{A,B,C: (Emp(A,B,C,D))\}

The augmented query Q' will be

\{A,B,C: (Emp(A,B,C,D))
\land Level([A,B,C], Unclassified)
\land Level([A], Unclassified)
\land Level([B,C], Unclassified)
\land Level([A,B], Unclassified)
\land Level([B], Unclassified)
\land Level([C], Unclassified) \}

and this query will determine the number of literals that will be resolved with the security constraints. In the best case, with no context-based constraints and only single outputs released simultaneously, the time complexity for location of constraints
is $O(n)$ where $n$ is the number of security constraints.

**Strategy 3.** In the third strategy each classification constraint is associated with one attribute of a relation. In effect the constraints are indexed by the relation and the attribute. Since the attributes used in processing a query are evident from inspecting the query, this indexing makes the search for the relevant security constraints efficient. As pointed out earlier, the way the query modification is done allows each output to be handled independently. This means that graphs representing context-based constraints with groups of $n$ members have space complexity of $O(n^2)$. The performance of the query modification process depends directly on the number of constraints that apply to the output attributes and not on the number of security constraints overall.

In this article we have described strategies for query processing in an MLS/DBMS: a basic strategy making use of query modification and two extensions of this technique, Strategy 2 and Strategy 3. Strategy 2 is based on logic and adds environmental state information to deal with context-based constraints. Strategy 3 uses graphs to index security constraints and processes output variables independently. It adds no new capabilities but provides increased performance. Both techniques ensure that a query is modified in such a way that if the modified query is posed, the response generated will not violate security requirements.

We have examined performance issues for the two proposed strategies. The indexing of security constraints in graph-based Strategy 3 makes it on the whole more efficient. However, logic-based Strategy 2 can support more flexible classifications of data. Strategy 3 is clearly superior in dealing with context-based constraints, generating fewer classification rules per constraint and processing queries with multiple outputs more efficiently.

There are many problems in the development of an MLS/DBMS. Among them are developing a security model, defining strategies for query processing, verifying the security properties of the system, and ensuring adequate system performance. Secure query processing can form an essential part of an MLS/DBMS.

### References


### Authors

**T. F. Keefe** is a graduate student in the Department of Computer Science at the University of Minnesota. His research interests include database security and software engineering. He received a BS in electrical engineering and an MS in computer science from the University of Minnesota in 1980 and 1985 respectively.

**Bhavani Thuraisingham** is a lead engineer at Mitre Corporation, doing research in database security and integrity. Previously she was at Honeywell, Inc., and before that at Control Data Corporation where she worked on database security, knowledge-based systems, and networks. She was also an adjunct professor and member of the graduate faculty in the Department of Computer Science at the University of Minnesota.

Thuraisingham received the BS in mathematics and physics from the University of Sri-Lanka, the MS in computer science from the University of Minnesota, the MS in mathematical logic from the University of Bristol, United Kingdom, and the PhD in recursive functions and computability theory from the University of Wales, Swansea, United Kingdom. She has published more than 30 technical papers on data/knowledge-based systems and computability theory. She is a member of the Computer Society, ACM, and Sigma-Xi.

**W. T. Tsai** is an assistant professor in the Department of Computer Science at the University of Minnesota. His areas of interest are software engineering, artificial intelligence, computer security, computer systems, and neural networks. Tsai received his SB degree in computer science and engineering from MIT in 1979 and his MS and PhD degrees in computer science from the University of California, Berkeley, in 1982 and 1986, respectively. He is a member of the IEEE.

Readers may write Thuraisingham at Mitre Corporation, K325, Burlington Road, Bedford, MA 01730.