SECURE QUERY PROCESSING USING AI TECHNIQUES

M.B. Thuraisingham

Honeywell
Corporate Systems Development Division
Golden Valley, MN 55427

W.T. Tsai

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

T.F. Keefe

Department of Computer Science
University of Minnesota
Minneapolis, MN 55455

Abstract - This paper describes the use of AI techniques for query processing in a multilevel secure database management system. The techniques ensure that a query is modified in such a way that if the modified query is posed, then the response generated will not violate security requirements. These techniques also handle constraints which classify data by context, time, and aggregation. Furthermore, DBMSs are not built with adequate controls and mechanisms to ensure that users are allowed to access only the data to which they have clearance. Thus an MLS/DBMS is different from a conventional DBMS in at least the following ways:

1. every data item has associated with it one of several classification levels that may need to change with time,
2. a user's access to data must be controlled based upon the user's authorization with respect to these data classifications.

Providing an MLS/DBMS on current secure computing systems presents a new set of problems. 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 in a file. Another problem is unique to databases is the necessity to classify data based upon context, time, and aggregation. Furthermore, DBMSs are also vulnerable to inference attacks where a user infers unauthorized data from the knowledge that he has accumulated.

The Air Force Summer Study of 1982 proposed various designs for Multilevel Secure Database Management Systems, (MLS/DBMS). One, a near set of requirements incorporating off-the-shelf concepts, and two a longer term set of requirements which include content, context, dynamic classifications and solutions to the inference problem. Two groups have taken up the challenge of seeking a long term solution with the purpose of incorporating features of multilevel security to prevent security violations. What has since become apparent is that present strategies employed in MLS/DBMSs are neither wholly sufficient nor efficient in processing queries while ensuring that users do not acquire information to which they are not authorized. To overcome the lack of suitable techniques, the database security researchers at SR1 and Honeywell are applying AI techniques to solve problems in database security. At SR1, Morgenstern and co-workers have focussed their attention on the foundations of the inference problem in database security. They have since established a general framework within which the inference problem in MLS/DBMS can be overcome. In contrast, Thuraisingham at Honeywell has been directing her efforts at particular aspects of the inference problem. She has designed an architecture for a MLS/DBMS which augments a DBMS with an inference engine and then suggests a query processing strategy which handles the inference problem arising from logical inference. Thuraisingham's results have been illustrated mainly with examples. Efficient techniques for secure query processing are yet to be formulated.

This paper will discuss the problems encountered in designing Multilevel Secure Database Management Systems, (MLS/DBMS) and propose solutions utilizing AI techniques for some of these problems. We will explore the suitability of techniques used in AI applications for secure database query retrieval. More precisely we discuss the use of the relational model and logic for representing security relevant knowledge and we propose two graph factoring methods, one based on rewrite rules and the other based on graph factoring. Such techniques will not only overcome certain security violations in DBMS but will also eventually lead toward security in knowledge-based systems.

The organization of this paper is as follows: In Section 2, we will describe some previously proposed techniques for retrieving data from a MLS/DBMS and discuss their limitations. In Section 3 we propose new techniques for query processing in MLS/DBMS augmented with inference engines which overcome some of these limitations. Finally in Section 4, we conclude with future considerations.

2. 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. For example, queries against views have been transformed into queries against base relations. In deductive databases, queries against virtual relations 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 through query modification. In this section we will describe this query modification mechanism and then state some of its drawbacks. 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. Before illustrating this technique with examples, we will first give an exposition on 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 the classification of the entire database, as well as the classification by relation and by attribute. Constraints that classify by context provide...
attacks. This is check the history information necessitates the JOIN of the relations EMP and DEPT. Such constraints modification algorithm is for a query.

Moreover, for each additional query that is posed some efficient computation technique is required to see if the response generated combined with the previous responses will cause a breach in security.

In the next section we will explore the representation of constraints, representation of history of released information and their manipulation to process queries securely.

3. STRATEGIES FOR SECURE QUERY PROCESSING

In this section we describe strategies for query processing in an MLS/DBMS. The underlying mechanism used in all strategies is "query modification". It should be noted however, that even if the query is modified correctly, the response generated may still violate security if the DBMS contains a Trojan horse (or hostile code) which will signal information downward in level. To ensure that no Trojan horse exists, specific portions of the code have to be verified. In this paper we do not consider the verification issue. Instead we concentrate on different strategies that can be used to modify the query.

There are two important concerns related to a strategy for secure query processing. First is the representation of the knowledge required for processing the query, this includes the constraints, environmental knowledge, and real-world knowledge. Second is the efficient methods of manipulating this knowledge. For example methods for finding constraints relevant to a given query or methods for constructing the modified query. The next two sections deal with these issues. Section 3.1 discusses knowledge representation and Section 3.2 deals with methods for modifying queries using one of the knowledge representations.

3.1 KNOWLEDGE REPRESENTATION

There is basically three kinds of knowledge which must be represented. First and most important is security constraints, including simple, content, and context types. Second is environmental information concerning what data has been released at what level. Finally there is real-world knowledge describing virtual relations and miscellaneous facts. These representation problems are discussed in the following sections.

The organization of this section is as follows: In Section 3.1.1, we describe how a standard relational MLS/DBMS processes queries. Here we assume that the security constraints are represented by the relational model and then point out the inability of this model to handle inference. In Section 3.1.2 we will describe an MLS/DBMS augmented with an Inference Engine and demonstrate the suitability of logic for representing the necessary knowledge.

3.1.1 RELATIONAL MODEL. Most existing relational database management systems are not sufficient to resolve the major compromise to security due to the inference of unauthorized information from the information that users can legitimately acquire. To overcome such attacks, the information on all responses that have been released must be maintained. Furthermore, for each additional query that is posed some efficient computation technique is required to see if the response generated combined with the previous responses will cause a breach in security.

In the next section we will explore the representation of constraints, representation of history of released information and their manipulation to process queries securely.

...
C1: NAME in EMP where SALARY > 100K is SECRET
C2: NAME and SALARY taken together are SECRET
C3: NAME in EMP when PROJECT in DEPT is "STARS" is SECRET
C4: DNAME and MGR taken together when DEPT = "D1" is SECRET
C5: PROJECT in DEPT is SECRET.

The representation of this meta-data is shown in Tables 1 through 6. Table 1 describes the relations, the arity of each relation, the number of tuples in each relation and one key attribute. Table 2 describes the attributes in each relation. Table 3 describes the views. Table 4 describes the name and text of each security constraint. Table 5 describes the context and simple constraints. Note that a simple constraint is a content-based constraint with no condition. Table 6 describes the context-based constraints. We have not described the internal schemas as they are not required for the query modification process.

### Table 1

<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

<table>
<thead>
<tr>
<th>Rname</th>
<th>Anname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>SS#</td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Salary</td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td></td>
</tr>
<tr>
<td>Dept</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3

<table>
<thead>
<tr>
<th>View Name</th>
<th>View Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>project (Emp join Dept) [SS#_Name, Salary, Dept#_Dept, Project, Dname, Mgr]</td>
</tr>
</tbody>
</table>

### Table 4

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>text1</td>
</tr>
<tr>
<td>C2</td>
<td>text2</td>
</tr>
<tr>
<td>C3</td>
<td>text3</td>
</tr>
<tr>
<td>C4</td>
<td>text4</td>
</tr>
<tr>
<td>C5</td>
<td>text5</td>
</tr>
</tbody>
</table>

### Table 5

<table>
<thead>
<tr>
<th>Rname</th>
<th>Cname</th>
<th>Target</th>
<th>Condition</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emp</td>
<td>C1</td>
<td>Name</td>
<td>Salary &gt; 100K</td>
<td>Secret</td>
</tr>
<tr>
<td>V1</td>
<td>C3</td>
<td>Name</td>
<td>Project = &quot;STARS&quot;</td>
<td>Secret</td>
</tr>
<tr>
<td>Dept</td>
<td>C5</td>
<td>Project</td>
<td>NULL</td>
<td>Secret</td>
</tr>
</tbody>
</table>

### Table 6

<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>Emp</td>
<td>C2</td>
<td>NULL</td>
<td>Salary</td>
<td>Secret</td>
</tr>
<tr>
<td>Dept</td>
<td>C4</td>
<td>Dept = &quot;D1&quot;</td>
<td>Dname</td>
<td>Secret</td>
</tr>
<tr>
<td>Dept</td>
<td>C4</td>
<td>Dept = &quot;D1&quot;</td>
<td>Mgr</td>
<td>Secret</td>
</tr>
</tbody>
</table>

The relational model has three major weaknesses as a means for knowledge representation for secure query processing. First, suppose an unclassified user wants to obtain all names. Then the Tables 5 and 6 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 will be first modified to retrieve all names where the 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 our model. Although it is possible to represent such real-world information using the relational model, the manipulation of such information is quite complex.

Another difficulty is encountered with the relational model in representing constraints when the level of one attribute depends on the level of another. In the previous example, suppose we have another constraint which classifies salary to be secret only if the department number in EMP is secret. If an unclassified user wants to retrieve the salaries, then the query is first modified to retrieve all salaries where the predicate DEPARTMENT_UNCLASSIFIED is true. But this modified query cannot yet be evaluated against the relational database. The tables have to be searched again to get all the constraints which classify the department number as secret to provide a definition for the DEPARTMENT_UNCLASSIFIED predicate.

The third difficulty is inferring over time. Users can accumulate various sets of information retrieved over many sessions and infer unauthorized information. This scenario can be described with a simple example. Suppose some information R is released at a level L (i.e. to users at level L). This information is also available to all users at levels L1 ≥ L. However, a security violation occurs if users at level L1 ≥ L can combine R with the information (or history) (L1) already available to them and infer some information at level L2 > L1. This example shows that not only should information be maintained regarding the constraints, information should be maintained on the access history of the data for each security level. In addition, users may have other sources of information apart from the database. Such real world information should also be taken into account.

The relational model is capable of storing the necessary types of security information for the examples above and algorithms could be designed to manipulate the information in the necessary ways. If the relational model is viewed as an abstract data type, as data and operations defined on it, the problems become apparent. The types of operations naturally defined for the relational data model is the relational algebra perhaps with some extensions. It is clearly not practical to do the kinds of inference necessary to modify queries using these types of operations. Therefore if the knowledge representation problem is viewed in terms of abstract data types, clearly relational algebra is inappropriate. This is evidenced by the complex algorithms necessary to carry out even simple query modifications.

If the database security problem is simply stated using logic, then a knowledge representation that supports inference seems an appropriate choice. In the ensuing sections we will describe a relational database management system augmented with an inference engine which not only facilitates the representation and manipulation of constraints for query modification but also lays the foundation for solving the inference problem arising from logical inferences.

### 3.1.2 LOGIC FOR KNOWLEDGE REPRESENTATION

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.14 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 a suitable representation, and a processing system called the inference engine separate from the DBMS to process the information in the knowledge base.15 A high level architecture of an augmented DBMS is shown in Figure 1. Since standard relational databases are not powerful enough to overcome the inference problem, the augmented database approach can be taken to design an MLS/DBMS. The database consists of only the information that can be manipulated easily by the relational DBMS. The knowledge base consists of all the security constraints, integrity constraints, and the real-world information. Since the constraints can be expressed as rules, logic is an appropriate candidate for modelling the knowledge base.
In addition to the ease of expression and manipulation of constraints expressed in logic, another advantage of using logic is that logical deduction of formulas fits in naturally with the deduction of information that 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 level L. Let S be the set of formulas which represents all the information in (E(L),R) where E(L) is all the information that is already known to this user at level L. Let A be a formula that can be deduced from S by a logical rule of inference. That is, there is a sequence of formulas B1,...,Bn where Bn=A, and each Bj (1 ≤ j ≤ n) either belongs to S or there exists j,k (where 1 ≤ j,k < i) such that Bj results from Bj and Bj by applying some rule of inference. Then a violation of security by inference has occurred if the sensitivity of A is greater than L.

We will now describe how the constraints defined in the previous section can be expressed in logic. A logical formula is of the form A → B meaning A implies B and B the conclusion. Each base relation (a relation in the database) or a virtual relation (a relation not in the database) is represented by a predicate. Therefore the constraints C1 to C5 are expressed by the following rules R1 to R5 respectively.

R1: EMP(X,Y,Z,W) → RELEASE(Y,UNCLASSIFIED) → LEVEL(Y,SECRET)
R2: EMP(X,Y,Z,W) → LEVEL(Y,SECRET)
R3: EMP(X,Y,Z,W) → DEPT(P,Q,R,S) → W=P ∧ Q="STARS" → LEVEL(Y,SECRET)
R4: DEPT(P,Q,R,S) ∧ P="DI" → LEVEL(RS,SECRET)
R5: DEPT(P,Q,R,S) → LEVEL(Q,SECRET)

The rules R2 and R4 are context-based constraints. They classify attributes taken 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 using the relational model it was not easy to express such constraints, whereas using logic such rules can be expressed easily as follows:

R6: EMP(X,Y,Z,W) ∧ RELEASE(Y,UNCLASSIFIED) → LEVEL(Z,SECRET)
R7: EMP(X,Y,Z,W) ∧ RELEASE(Z,UNCLASSIFIED) → LEVEL(Y,SECRET)
R8: DEPT(P,Q,R,S) ∧ P="DI" ∧ RELEASE(R,UNCLASSIFIED) → LEVEL(S,SECRET)
R9: DEPT(P,Q,R,S) ∧ P="DI" ∧ RELEASE(S,UNCLASSIFIED) → LEVEL(R,UNCLASSIFIED)

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

Suppose an unclassified user wants to retrieve all names from EMP. Then the general technique used to modify the query:

\[ Y: \text{EMP}(X,Y,Z,W) \]

is as follows: The knowledge base will be searched for all rules which will classify the names in EMP. These rules are R1, R3, and R7. The conditions in these rules will be negated and the resulting query is

\[ Y: \text{EMP}(X,Y,Z,W) ∧ Z \leq 100K ∧ \text{DEPT}(P,Q,R,S) ∧ (P=W ∧ Q="STARS") ∧ \neg\text{RELEASE}(Z,\text{SECRET}) \]

This query cannot be evaluated still as RELEASE is not a base relation. Therefore the knowledge base will be searched to see if the salaries have been released. Since the salaries have not yet been released, \( \neg\text{RELEASE}(Z,\text{UNCLASSIFIED}) \) is TRUE. Therefore the query that will be evaluated is:

\[ Y: \text{EMP}(X,Y,Z,W) ∧ Z < 100K ∧ \text{DEPT}(P,Q,R,S) ∧ (P=W ∧ Q="STARS") \]

This query will be translated into a language appropriate to the DBMS. This query will be evaluated against the relational DBMS and the response will be released. Since processing this query releases names at the unclassified level, this information has to be recorded as real world information. This is done by adding the rule

R10: \( \rightarrow\text{RELEASE}(Y,\text{UNCLASSIFIED}) \)

to the knowledge base. Now if the unclassified user wants to obtain the salaries in EMP, the query posed will be:

\[ Z: \text{EMP}(X,Y,Z,W) \]

This query will be modified to

\[ Z:\text{EMP}(X,Y,Z,W) ∧ \neg\text{RELEASE}(Y,\text{UNCLASSIFIED}) \]

However, by rule R10, \( \neg\text{RELEASE}(Y,\text{UNCLASSIFIED}) \) is FALSE. Since we arrive at a contradiction, no query will be posed.

The examples described above show that unlike the relational model, logic can be used fairly easily to represent constraints, history and real world information.

### 3.2 Query Transformation Methods

The last section developed methods for representing the knowledge necessary for query processing. This section describes efficient methods for manipulating the logic-based knowledge representation described there.

In Sections 3.2.1 and 3.2.2 we describe two strategies which use standard AI techniques, rewrite rules and a graph factoring mechanism, for query modification.

In recent years prototype DBMS augmented with inference engines and Knowledge Base Systems have been developed which utilize these techniques and other AI techniques for query processing. These query processing techniques have been shown to be fairly efficient. However, the performance of these techniques when applied to Multilevel Security is yet to be determined. Nevertheless we show how easily these techniques can be modified to provide Multilevel Security.

#### 3.2.1 Rewrite Rules for Query Transformation

The query transformation technique that we will describe here is similar to the technique proposed by Chang in the DEDUCE 2 project\cite{15} for rewriting virtual relations used in queries in terms of base relations. We will show that this technique can also be applied to secure query

---

**(Figure 1 - Relational DBMS Augmented with an Inference Engine)**

In addition to the ease of expression and manipulation of constraints expressed in logic, another advantage of using logic is that logical deduction of formulas fits in naturally with the deduction of information that 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 level L. Let S be the set of formulas which represents all the information in (E(L),R) where E(L) is all the information that is already known to this user at level L. Let A be a formula that can be deduced from S by a logical rule of inference. That is, there is a sequence of formulas B1,...,Bn where Bn=A, and each Bj (1 ≤ j ≤ n) either belongs to S or there exists j,k (where 1 ≤ j,k < i) such that Bj results from Bj and Bj by applying some rule of inference. Then a violation of security by inference has occurred if the sensitivity of A is greater than L.

We will now describe how the constraints defined in the previous section can be expressed in logic. A logical formula is of the form A → B meaning A implies B and B the conclusion. Each base relation (a relation in the database) or a virtual relation (a relation not in the database) is represented by a predicate. Therefore the constraints C1 to C5 are expressed by the following rules R1 to R5 respectively.

R1: EMP(X,Y,Z,W) → RELEASE(Y,UNCLASSIFIED) → LEVEL(Y,SECRET)
R2: EMP(X,Y,Z,W) → LEVEL(Y,SECRET)
R3: EMP(X,Y,Z,W) → DEPT(P,Q,R,S) → W=P ∧ Q="STARS" → LEVEL(Y,SECRET)
R4: DEPT(P,Q,R,S) ∧ P="DI" → LEVEL(RS,SECRET)
R5: DEPT(P,Q,R,S) → LEVEL(Q,SECRET)

The rules R2 and R4 are context-based constraints. They classify attributes taken 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 using the relational model it was not easy to express such constraints, whereas using logic such rules can be expressed easily as follows:

R6: EMP(X,Y,Z,W) ∧ RELEASE(Y,UNCLASSIFIED) → LEVEL(Z,SECRET)
R7: EMP(X,Y,Z,W) ∧ RELEASE(Z,UNCLASSIFIED) → LEVEL(Y,SECRET)
R8: DEPT(P,Q,R,S) ∧ P="DI" ∧ RELEASE(R,UNCLASSIFIED) → LEVEL(S,SECRET)
R9: DEPT(P,Q,R,S) ∧ P="DI" ∧ RELEASE(S,UNCLASSIFIED) → LEVEL(R,UNCLASSIFIED)

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

Suppose an unclassified user wants to retrieve all names from EMP. Then the general technique used to modify the query:

\[ Y: \text{EMP}(X,Y,Z,W) \]

is as follows: The knowledge base will be searched for all rules which will classify the names in EMP. These rules are R1, R3, and R7. The conditions in these rules will be negated and the resulting query is

\[ Y: \text{EMP}(X,Y,Z,W) ∧ Z ≤ 100K ∧ \text{DEPT}(P,Q,R,S) ∧ (P=W ∧ Q="STARS") ∧ \neg\text{RELEASE}(Z,\text{SECRET}) \]

This query cannot be evaluated still as RELEASE is not a base relation. Therefore the knowledge base will be searched to see if the salaries have been released. Since the salaries have not yet been released, \( \neg\text{RELEASE}(Z,\text{UNCLASSIFIED}) \) is TRUE. Therefore the query that will be evaluated is:

\[ Y: \text{EMP}(X,Y,Z,W) ∧ Z < 100K ∧ \text{DEPT}(P,Q,R,S) ∧ (P=W ∧ Q="STARS") \]

This query will be translated into a language appropriate to the DBMS. This query will be evaluated against the relational DBMS and the response will be released. Since processing this query releases names at the unclassified level, this information has to be recorded as real world information. This is done by adding the rule

R10: \( \rightarrow\text{RELEASE}(Y,\text{UNCLASSIFIED}) \)

to the knowledge base. Now if the unclassified user wants to obtain the salaries in EMP, the query posed will be:

\[ Z: \text{EMP}(X,Y,Z,W) \]

This query will be modified to

\[ Z:\text{EMP}(X,Y,Z,W) ∧ \neg\text{RELEASE}(Y,\text{UNCLASSIFIED}) \]

However, by rule R10, \( \neg\text{RELEASE}(Y,\text{UNCLASSIFIED}) \) is FALSE. Since we arrive at a contradiction, no query will be posed.

The examples described above show that unlike the relational model, logic can be used fairly easily to represent constraints, history and real world information.
processing. The language DEDUCE 2 is a query language based on logic. Using DEDUCE 2, for example, the rule R1 can be expressed by:

\[ \forall Y, X, Z, W: \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \land Z > 100K \rightarrow \text{LEVEL}(Name = Y, \text{SECRET}) \]

This rule defines a virtual relation named LEVEL which relates values of the attribute Name with a security classification of SECRET.

The query transformation method includes the following steps:

1. Create a connection graph using the query, the constraints, and environmental knowledge.
2. Generate rewrite rules from the connection graph.
3. Use rewrite rules to create a set of plans.
4. For each plan generate a query.
5. Create modified query from original query and queries generated in 4.

We will now illustrate the rewriting rule technique with an example. The query to be modified:

\[ \forall Y, X, Z, W: \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \land Z > 100K \rightarrow \text{LEVEL}(Name = Y, \text{SECRET}) \]

is posed by an unclassified user. The * by Name implies that the query is to retrieve the attribute "Name". In this example the constraints C1, C2, and C3 are enforced, and salaries have not yet been released to the user. The constraints are shown below:

\[ C_1: \forall Y, X, Z, W: \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \land Z > 100K \rightarrow \text{LEVEL}(Name = Y, \text{SECRET}) \]

\[ C_{2a}: \forall Y, X, Z, W: \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \land \text{Release}(Y, \text{Unclassified}) \rightarrow \text{LEVEL}(Name = Y, \text{SECRET}) \]

\[ C_{2b}: \forall Y, X, Z, W: \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \land \text{Release}(Y, \text{Unclassified}) \rightarrow \text{LEVEL}(Salary = Z, \text{SECRET}) \]

\[ C_3: \forall Y, X, Z, W, P, Q, R, S: \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \land \text{Dept}(Dept# = P, \text{Project} = Q, \text{Name} = R, \text{Mgr} = S) \land P \neq W \land Q = \text{"STARS"} \rightarrow \text{LEVEL}(Name = Y, \text{SECRET}) \]

The following sections illustrate each step in the method using the above example.

3.2.1.1 Connection Graph-- Instead of modifying the query to obtain all unclassified names, we pose a query which 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 since "Name" is not a key attribute. These could arise the situation where two different employees share the same name and makes less than 100K and the other makes more than 100K. By 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. Note that in this discussion we have assumed only two levels, namely the unclassified and secret levels. If there are more than two levels, the technique can be modified.

First the query is modified to Q MINUS Q:

\[ \forall Y, X, Z, W: \{ \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \} \rightarrow \text{MINUS} \]

\[ \forall Y, X, Z, W: \{ \text{EMP}(SS# = X, Name = Y, Salary = Z, Dept# = W) \} \land \text{LEVEL}(Name = Y, \text{SECRET}) \]

A connection graph is constructed for the portion of the query following the MINUS, that is the query Q'. In a connection graph, a constraint

\[ (Q_1 X_1) \ldots (Q_r X_r) (A_1 \land \ldots \land A_n \land F) \rightarrow B, \]

where Qi represents a quantifier, Ai is an atomic formula containing a base or virtual relation, F is a formula free of base or virtual relations and B is an atomic formula containing a virtual relation, is expressed by:

\[ (Q_1 X_1) \ldots (Q_r X_r) \begin{bmatrix} A_1 & A_2 & \ldots & A_n & F & B \end{bmatrix} \]

where A1, A2, ....., An are called left literals F is a left formula and B is a right literal. A query

\[ (Q_1 Y_1) \ldots (Q_s Y_s) (C_1 \land \ldots \land C_m \land G), \]

is expressed by

\[ (Q_1 Y_1) \ldots (Q_s Y_s) \begin{bmatrix} C_1 & C_2 & \ldots & C_m & G \end{bmatrix} \]

where C1, C2, ....., Cm are left literals and G is a left formula.

The graph is constructed from one node representing each constraint and one node representing the query Q'. Edges are added to the connection graph according to the following rule:

For every pair of literals L1 and L2 in the graph, if L1 and L2 contain a virtual relation and are unifiable,15 and if L2 is a right literal and L1 is a left literal, draw a directed edge from L1 to L2 and label the edge.

The connection graph for the example is shown in Figure 2. Constraint nodes which are not connected to the query node are not used in the transformation and do not appear in the figure.

3.2.1.2 Rewrite Rules-- Rewrite rules form a grammar describing a language T. Each sentence in this language is called a plan and describes a query Qi defined solely in terms of base relations such that:

\[ n = Q' = \bigcup_i Q_i \]

where Q' is constructed from the posed query which contains references to virtual relations.

Rewriting rules are obtained from the connection graph as follows:

1. For each left literal n, if m1, m2, ....., mℓ are all the right literals having edges pointing to n as shown in Figure 3, the rewriting rule obtained is:

\[ W(n) = \alpha_1 W(m_1) \cup \ldots \cup \alpha_\ell W(m_\ell) \]

2. For each constraint, if there is an edge leaving the right literal n as shown in Figure 4, the rewriting rule obtained is:

\[ W(n) = P_1 \ldots P_{\ell} \]

where Pi = W(mi) if mi is a literal containing a virtual relation; other wise.
Note in Figure 2, Emp is short for Emp(SS#=X, Name=Y, Salary=Z, Dept#W) and Dept is short for Dept(Dept#=P, Project=Q, Dname=R, Mgr=S).

Figure 2 - Connection Graph for Example

Figure 3 - Rewriting Rule Diagram for Left Literals

Figure 4 - Rewriting Rule Diagram for Right Literals

3. For each query as shown in Figure 5, the rewriting rule obtained is:

\[ T = Q_1 \ldots Q_m \]

where \( Q_i = W(C_i) \) if \( C_i \) is a literal containing a virtual relation; otherwise.

\[ = C_i \]

Figure 5 - Query Clause
The rewrite rules for the example are given below:

\[ T = Q_1 W(Q_2) \]  
(3)

\[ W(Q_2) = \alpha_1 W(A_3) \cup \alpha_2 W(B_4) \cup \alpha_3 W(C_3) \]  
(1)

\[ W(A_3) = A_1 A_2 \]  
(2)

\[ W(B_4) = B_1 B_2 B_3 \]  
(2)

\[ W(C_3) = C_1 C_2 \]  
(2)

Where the naming of literals corresponds to that in Figure 2, and the numbers at the right of each production names the rule number used in its derivation.

3.2.1.3 Plans-- Using the rewrite rules developed above, plans are generated from them by generating sentences in the language T described by the rewrite rules. The sentences generated in this example are:

- **K**: \( Q_1 \alpha_1 A_1 A_2 \)
- **M**: \( Q_1 \alpha_3 B_1 B_2 B_3 \)
- **N**: \( Q_1 \alpha_3 C_1 C_2 \)

In our example three plans say K', M', and N' corresponding to K, M, and N respectively will be generated. The plan diagrams differ from the sentences simply in that the literals themselves appear in the plan and not merely the name, quantification is made explicit, and the edge appears directly. The plans for this example appear below:

**PLAN K':**

\( \forall Y, X, Z, W \) \( Z > 100k \)

\( \forall Y, X, Z, W \)

**PLAN M':**

\( \forall Y, X, Z, W \) \( W = W \)

\( \forall Y, X, Z, W \) \( P = W \)

\( \forall Y, X, Z, W \) \( Q = "STARS" \)

**PLAN N':**

\( \forall Y, X, Z, W \) \( W = W \)

\( \forall Y, X, Z, W \) \( W = W \)

\( \forall Y, X, Z, W \) \( W = W \)

\( \forall Y, X, Z, W \) \( W = W \)

Where the naming of literals corresponds to that in Figure 2, and the numbers at the right of each production names the rule number used in its derivation.

3.2.1.4 Sub-queries-- A plan is transformed into a query by renaming variables such that variables in different clauses are unique and the name correspondence described by the directed edges in the plan is accounted for. Also, if a plan contains a virtual relation which is not defined in the plan, then this relation is replaced by FALSE. For example, the relation RELEASE in PLAN N' has been replaced by FALSE.

The plans from the example give rise to the following three sub-queries Q1, Q2, and Q3 where \( Q' = Q_1 \cup Q_2 \cup Q_3 \):

- **Q1**: \( \forall Y, X, Z, W \) \( (\text{EMP}(SS#=X,*Name=Y,Salary=Z,Dept#=W) \land Z > 100k) \)
- **Q2**: \( \forall Y, X, Z, W, P, Q, R, S \)
  \[ \text{EMP}(SS#=X,*Name=Y,Salary=Z,Dept#=W) \land \text{DEPT}(P=Q,\text{Projects}=Q,\text{Dname}=R,\text{Mgr}=S) \land (P=W \land Q = "Stars") \]
- **Q3**: \( \forall Y, X, Z, W \) \( (\text{EMP}(SS#=X,*Name=Y,Salary=Z,Dept#=W) \land \text{FALSE}) \)

Note that in general all three queries have to be posed to retrieve the secret names. However, in the case of query Q3, since its condition is a contradiction this query will not be posed. The query that will be evaluated is \( Q \) \( \text{MINUS} \) \( Q' \) and the unclassified names will be released.

3.2.1.5 Environmental Information-- The fact that names were released at the Unclassified level is represented by the following node in the connection graph:

**TRUE**

Release\((\text{Name}=Y,\text{Unclassified})\)

Next, suppose the unclassified user wants to retrieve all names after he obtains the salaries. The relevant part of the connection graph for this query is shown in Figure 6.

The plans that will be generated for the query now are the same as before with the exception of Plan N' which is shown below. The query that will be posed is:

\( \forall Y, X, Z, W \) \( \text{EMP}(SS#=X,*Name=Y,Salary=Z,Dept#=W) \text{MINUS} \)

Consequently, no information will be released to the user.
FACTORIZING MECHANISM FOR QUERY TRANSFORMATION: In this section we will describe another technique for secure query processing. 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 so that representation of redundant information can be avoided. 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. There is just one point representing EMP, and the constraints that are enforced on it originate at this point. The remainder of this section describes how information is represented and manipulated using this technique.

Figure 7 shows how the constraints C1 and C2 are represented. EMP(X,Y,Z,W) is the label of a node with in-degree 0. Since EMP has four attributes, there are four edges which originate at this node. The nodes at the end of these edges specify the corresponding attribute. For example, NAME(Y) is the label of the node which states that Y in EMP is the attribute NAME. The condition Z > 100K in constraint C1 is represented by a node which points to Salary(Z). Since neither C1 nor C2 assign levels to the attributes SS# and DEPT#, there are no edges originating at the nodes labelled SS#(X) and DEPT#(W). The constraint C1 classifies all names who earn more than 100K as secret. Therefore
there will be an arrow from LEVEL(YZ,Secret) to the condition \( W = 'D1'. \) The node labelled with this condition will be the head of the edge whose tail is DEPT(W).

When an unclassified user wants to retrieve the names and salaries together, the search is started at the node EMP(X,Y,Z,W) for the attributes NAME and SALARY. The purpose of this search is to find a node which classifies names and salaries taken together at any level greater than or incomparable to the unclassified level. In this search, first the nodes NAME(Y) and SALARY(Z) will be visited and subsequently the node LEVEL(YZ,SECRET) will be reached. Since the names and salaries taken together are secret and there is no edge from the node LEVEL(YZ,SECRET), no query will be posed. If a secret user wants to retrieve the names and salaries together, the search will be for a node which classifies names and salaries taken together at a level greater than or incomparable to the secret level. Since there is no node in the graph with this property, the query will not be modified. Suppose an unclassified user wants to retrieve the names only. Then the search will be for a node which classifies names only at a level greater than or incomparable to the unclassified level. The node LEVEL(Y,SECRET) is such a node. Then all the nodes pointed to by edges emanating from LEVEL(Y,SECRET) will be examined and the conditions in these nodes will be negated to modify the query. Therefore the resultant query will be to retrieve all names whose salaries are less than or equal to 100K. If a secret user wants to retrieve all the names, then the search will be for a node which classifies all names at a level greater than or incomparable to the secret level. Since there is no such node, the query will not be modified.

### 3.2.2.1 Handling Virtual Relations

In the above discussion, we have assumed that there are no constraints on derived relations such as the constraint C3, nor can we handle multiple queries such as retrieving the names first and then the salaries. We will now examine each of these cases. First suppose the constraint C3 is enforced in addition to the constraints C1 and C2. Then the additional information that has to be incorporated into Figure 7 is shown in Figure 8. To express the constraint C3, first a join operation has to be performed between EMP and DEPT over the attribute DEPT. Therefore we have another node DEPT(P,Q,K,S) which has in-degree 0 and with four edges originating from it to represent its four attributes. Since the join is over DEPT, there is an edge from the node DEPT(W) which corresponds to the relation EMP; there is an edge from DEPT(P) which corresponds to relation DEPT, and there is an edge from PROJECT(Q) which all meet at the node \( P=W \land Q=\text{STARS}. \) Since the names in project "STARS" are secret, there is an edge from LEVEL(Y,SECRET) to the node \( P=W \land Q=\text{STARS}. \) 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 which 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 which originate from this node will be traversed. The two nodes that will specify the conditions to classify the names are \( Z > 100K \) and \( P=W \land Q=\text{STARS}. \) Then these conditions will be negated and the query will be modified to retrieve all names where the salary is less than or equal to 100K and who do not work in STARS.

### 3.2.2.2 Handling Multiple Queries

Now let us examine how multiple queries can be handled. Because of the constraint C2, the names should not be released to an unclassified user if the 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. That is the names are secret if the salaries have been released and the salaries are secret if the names have been released. The additional information that has to be incorporated into Figure 7 to enforce the constraints R8 and R9 are shown in Figure 9. In this figure we have two more nodes. One of these nodes is labelled RELEASE(Y,Unclassified) which implies that names are released and the other node is labelled 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 the salaries after he has obtained the names, then a search will be made for a node which 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 that is pointed by this node is RELEASE(Y,Unclassified). Therefore the query will be modified to retrieve all salaries where the names have not been released. But this query still cannot be evaluated because the predicate RELEASE is a virtual predicate. Therefore a search has to be made as to whether names have been released. In order to do this we need another graph to represent the real world information. This is shown in Figure 10. As soon as the names are released at the unclassified level form the previous query, this information has to be incorporated into the real world information. Therefore the NAME(Y) in Figure 9 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.

![Figure 8 - Constraints on Joins](image-url)
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 which is labelled LEVEL(Y,SECRET) has an arrow to the node labelled LEVEL(Z,SECRET). The query is then modified to retrieve all names where the salary is not secret. This query cannot be evaluated still as it contains a virtual relation. The graph is searched again for all the conditions which make salary in EMP secret. This is quite simple as all one has to do is to follow all the edges which originate at LEVEL(Z,SECRET).

5. REFERENCES