Aggregation and Inference:
Facts and Fallacies

Teresa F. Lunt
Computer Science Laboratory
SRI International
Menlo Park, California

Abstract

This paper examines inference and aggregation problems that can arise in multilevel relational database systems and points out some fallacies in our thinking about these problems that may hinder real progress from being made toward their solution. Although others have done some initial research toward solving inference problems, aggregation has been treated only superficially in the literature. This paper attempts to lay a firmer foundation for a theory of these problems. Several types of problem are identified and approaches toward their solution suggested.

1 Introduction

A relational database consists of a collection of atomic facts. Each fact represents either an entity (e.g., an employee number), a property of an entity (e.g., an employee’s address), or an association among entities (e.g., between an employee and occupation). A relation represents an entity type. An entity is typically represented as a tuple, or row, in a relation. Thus, entities of different types are typically represented as tuples in different relations. Entity properties are represented as the attributes, or columns, of a relation. Associations are represented by join attributes—that is, data that are stored in two entities that can be matched, or joined, to obtain the association. In a multilevel database system, each of these atomic facts is assigned a classification. The challenge is to design these classifications in such a way that they accurately reflect “real-world’ classifications for the information they represent, and that classified information cannot be inferred from the data assigned lower classifications.

The inference problem arises whenever some data z can be used to derive partial or complete information about some other data y, where y is classified higher than z. In some cases, even learning of the existence of the information may be unacceptable. The aggregation problem also arises from an attempt to protect a sensitive relationship among otherwise nonsensitive data.

Only a few of the database security projects have addressed inference and aggregation. These are SeaView [1,2,3], LOCK Data Views (LDV) [4,5], and TRW’s A1 Prototype [6,7]. This paper discusses the approaches to inference and aggregation that have been taken in these projects. However, I attach the caveat that these projects are not unambiguously documented in the literature and that, because the project ideas naturally evolve over time, so that some approaches originally proposed may later be abandoned, some of the approaches attributed herein to the various projects may be inaccurate. Nevertheless, the contribution of this paper lies not in an assessment of which project has the best approach, but in its clarifying the problems and its analysis of various approaches toward their amelioration.

This paper discusses some of the common misconceptions about database inference and aggregation problems. In the SeaView project, we have devoted a significant amount of study to inference and aggregation problems in multilevel secure database systems [1,8,9,10] and discovered that many inference and aggregation problems can be avoided by proper database design. Many such problems can be detected at data design (or redesign) time, using a static analyzer tool. This paper is not so much concerned with detecting inference problems as how to deal with such problems once it has been determined that a problem exists. Others [6,11] are working on methods for detecting inference problems.

2 What is Aggregation?

The aggregation problem arises whenever some collection of facts has a classification strictly greater than that of the individual facts forming the aggregate. To qualify as an aggregation problem, it must be the case that the aggregate class strictly dominates the class of every subset of the aggregate; otherwise, the simple mandatory rules would suffice to protect the collection of information. This is our first “fallacy”:

• FALLACY: Where an aggregation problem exists, some proper subsets of the aggregate may have the aggregate (high) classification.

To illustrate why this is indeed a fallacy, suppose that the aggregate $A$, classified high, is composed of the $n$ data items $a_1, \ldots, a_n$. Then if some $a_i \in A$ is also classified high, simple mandatory security will not allow a low subject to retrieve any set containing $a_i$, and thus would also trivially protect the aggregate $A$. Now suppose that the set $B$ is a proper
subset of $A$: $B \subseteq A$. Now if $B$ contains a high element
$a$, then again simple mandatory security would not allow a
low subject to retrieve $B$, and thus $B$ is classified high by
definition. If, however, $B$ does not contain a high element
and still must be considered to be classified high, then the
set $B$, rather than $A$, actually represents the aggregation
problem. Thus, if we were able to protect each such $B$, then
$A$ would be trivially protected by the simple mandatory
security rules. The next section illustrates how not using
this logically simple definition of aggregation can lead us astray.

3 Some Aggregation Problems are Really Inference Problems

As the preceding section suggests, many so-called aggregation
problems are not true aggregation problems and can readily be
solved through appropriate design of the data
structures (relations) in the database. This idea was first
suggested in the SeaView security policy report [1]. For
many perceived aggregation problems, some of the essential
facts are more appropriately classified at the higher, aggregate
classification, as is typically the case when the aggregate
represents an association among different entities (i.e.,
among data stored in two or more relations). Such problems are
readily solved by classifying the associations among entities separately and at a classification strictly greater than that of the entities themselves. Thus our second “fallacy”:

- FALLACY: Special, trusted, mechanisms are needed
to detect “context-dependent” aggregation problems: that is, when two entities, such as names and salaries, are classified low when seen in isolation, but classified high when combined. Such mechanisms must detect when a user (or a group of potentially colluding users) asks for one of the entities and has previously retrieved the other entity, and must either upgrade or withhold the result.

For example, consider a database containing personnel infor-
mation, including names, addresses, and salaries of em-
ployees. Suppose that names, addresses, and salaries are all
UNCLASSIFIED, but that names and salaries taken together
are SECRET—that is, the association of a salary with an individual is SECRET.

LDV includes a mechanism involving a set of history
files and constraints to detect just such apparent problems [5]. LDV’s mechanism can either upgrade both the
security level of the process handling the user query and the query result when SALARY is asked for and NAME
has already been seen, or else can lock the NAME file to any uncleared users when one uncleared user has accessed the SALARY file. The former alternative introduces a po-
tential covert channel, whereas the latter leads to serious
denial-of-service problems. Moreover, such solutions, be-
cause they enforce part of the mandatory security policy, have to be verified to the desired degree of assurance—a
potentially difficult proposition, since they rely on much of the database functionality.

LDV’s solution may have been driven by a misinter-
pretation of the sensitive relationship. Specifically, their
solution treats the association of the entire list of names with the entire list of salaries as sensitive, when what in
actuality is sensitive is the association of individual names with the individual salaries.

LDV’s solution is also motivated by a desire to provide protection against inferences that can be made by observing
the changing responses to the same query over time. We will
say more about this later in this section.

Hinke’s method [6], used in TRW’s prototype, correctly
recognizes the nature of the problem (that the sensitive
relationship is that between an individual entry in one re-
lation and an individual entry, or set of entries, in another relation). His method uses an inference detection tool to
detect potential such inference problems in the database,
based on the database schema. However, rather than modify
the data structures to remove the problem, Hinke’s
method merely runs queries periodically to detect whether
there are in fact tuples in the relations in question that can
be joined to give a sensitive association. For example, in
the names and salaries example, his system would period-
ically run a query to check to see whether there were any
names in the EMPLOYEES relation who also had salary in-
formation recorded in the SALARIES relation. If any data
are returned, an inference problem exists. In the TRW pro-
totype, the results of such queries are recorded in an audit
trail for subsequent analysis by a security officer, who can
then take steps to reclassify some of the data in an attempt
to remove the problem.

TRW’s inference detection queries run at system-high and require global access to the database. Thus, if
TRW’s inference detection queries were to be enhanced to
provide real-time access control, a large portion of the
database mechanism would then be participating in enforcing
mandatory access control, introducing assurance ques-
tions similar to LDV’s.

These “context-dependent” aggregation problems, as
Honeywell calls them, are not really aggregation problems
at all, but are solved through appropriate design and clas-
sification of the relations in the database. This is generally
the case when the “aggregation” involves the association
between entities of different types. For instance, the above
element is easily solved by the following data design

$$\begin{align*}
\text{EMPLOYEES} & \text{(EMP\# , NAME, ADDRESS)} \\
\text{SALARIES} & \text{(S\#, SALARY)} \\
\text{EMP-SALARIES} & \text{(EMP\#, S\#)}
\end{align*}$$

where the EMPLOYEES and SALARIES relations are both
UNCLASSIFIED but the EMP-SALARIES relation is SECRET.

Because there is no way to meaningfully join EMPLOYEES and SALARIES to match a salary with a name, the sensitive
relation, represented by the relation EMP-SALARIES, is pro-
tected from uncleared users. Thus, there is no aggregation
problem, and a simple reference monitor can allow unre-
restricted access to names and salaries while protecting the sensitive names-salaries relationship—no special machinery is needed to detect “aggregation violations.”

Now suppose that we have the above “safe” set of relations, and we want to add a new attribute, employee start date (which is not sensitive), to the database. It seems to make sense to add it to the SALARIES relation, as follows:

\[ \text{SALARIES}(S\#, \text{SALARY}, \text{START-DATE}) \]

However, an employee’s start date is an easily observable or discoverable attribute of an employee. Thus, an employee’s relation. Another way is to create a separate (unclassified) relation HIRE-DATES(EMP#, START-DATE). Note that EMP# rather than S# must be used as the key for the new relation, otherwise the inference is not removed.

The first problem above (that the SECRET relationship between employee name and salary is inferable) can be derived through analysis of the data structures and security constraints that would normally be part of a multilevel database. However, inference problems cannot always be detected using only the information stored in the database. For example, to solve the second problem above, we need to know that employee name can be inferred from START-DATE, a piece of information that would not normally be stored in the database.

In general, in a relational database system, an association among entities of different types (i.e., among information stored in different relations) can be represented either by a separate relation that contains join attributes for the entity relations, or by join attributes that are stored in the entity relations themselves. If, as in the above example for employee name and salary, the association is represented entirely by a separate relation (i.e., no meaningful associations can be made by joining the relations for the separate entities), the entire relation may be classified at the higher level. If the join attributes are stored in the entity relations, then the association can be protected by classifying at least one of the join attributes at the higher classification. For example, another solution to the above problem would be as follows:

\[ \text{EMP-SALARIES}(\text{EMP#}, S\#) \]

where attributes EMP#, NAME, and ADDRESS of the EMPLOYEES relation are UNCLASSIFIED, and attribute S# of the SALARIES relation is classified SECRET. The entire SALARIES relation is UNCLASSIFIED.

It is worth pointing out that even with intelligent data design, inference problems can still arise. For example, in the above example, there may be a small enough number of employees that the association between an employee’s name and salary may be guessed or narrowed down. Moreover, if the user can assume a piece of information that is probably correct, such as that the president of the company probably has the highest salary, then some of the sensitive associations will be revealed. In addition, there are well-known methods of statistical attack that can be employed to obtain sensitive information (Denning’s book [12] gives a nice overview of these). The sensitive association will also be compromised if the employee and salary records are both sorted in the same order, and this can be determined (for example, by an employee finding the location of his or her own employee and salary records and making the inference). None of the methods proposed in LDV or SeaView or those of Hinke [6], Morgenstern [11], or Thuraisingham [13] address these sorts of problems, and more work needs to be done in this area.

In addition, as has been pointed out by LDV, inferences can be drawn from observing the system’s changing response to the same query over time. In the example above, whenever a new employee is added, new tuples will be added to both the EMPLOYEES and the SALARIES relations, thus enabling one to infer the S# for the new employee. LDV’s mechanism addresses this problem, but there may be other, simpler, solutions as well. For example, the data for new employees could be classified SECRET and then declassified whenever their numbers had grown sufficiently large. Or, it may be more appropriate to classify the entire SALARIES relation SECRET if the inference threat is deemed to be too high. Although LDV’s approach may make sense on a system like LOCK, in this paper I seek to provide guidance toward general solutions that will be useful to anyone facing these problems.

As another example, a project’s mission could be inferred by knowing, say, the project leader, budget, start and end dates, client name, deliverables, etc. Those attributes (e.g., project leader, client name, deliverables) that reveal the most information about a project’s sensitive mission should be segregated from one another, as in the above example, so that the sensitive association can be coded separately and classified high. In this way, low subjects can have unrestricted access to the data without being able to derive the sensitive association, and the mandatory reference monitor automatically allows only appropriately cleared subjects to access the sensitive association—all special mechanism is needed. By contrast, LDV uses “aggregation constraints” that define the sensitive relationship, and they employ a mechanism to detect when the offending attributes are retrieved together and to reclassify or withhold the result. This mechanism is much more elaborate than the label comparison mechanism of a reference monitor.

Thus, we see that many of the so-called aggregation problems that LDV is designed to detect can be largely solved through appropriate data design. Many so-called aggregation problems really intend to hide sensitive relation-

1 This example is due to Tom Haigh and Paul Stachour.
ships among data, when the individual data items themselves are not sensitive. In a database system, such problems can be solved by storing the data low and storing the sensitive relationships high. Then the data are available to low subjects while the sensitive relationships are automatically protected by the mandatory security mechanisms, without relying on complicated trusted software to detect violations. With adequate database design tools (such as the inference control tool proposed by Morgenstern [11]), a proposed database design can be analyzed for such problems and restructured so that the problems are eliminated or minimized. Below, we outline the general approach for doing this.

3.1 Detecting Sensitive Associations

The aggregation problems just described fall under what has been called the related data inference channel [8]. Such problems can be addressed by specifying, as part of the database schema, constraints that govern how related data are to be classified so as to prevent against inferences. SeaView uses classification constraints to address this problem [2,9,14]. For example, if information about A can be inferred from B, then B can be classified at A’s level using content-dependent or type-dependent classification constraints (see [2] and [9] for examples of such constraints).

Although classification constraints do not guarantee that all related data will be classified in such a way as to prevent inferences, they provide a means whereby a security officer who has identified potential inference problems can constrain the classifications associated with the data causing the problems.

Early on, the SeaView project demonstrated how constraints (which we originally called derivation rules) could be used to locate inferences and to suggest to the user how the database might be redesigned so as to prevent the inferences [9]. For example, suppose we have a relation \( R(A, B, C) \) with classification constraints that classify attributes \( A \) and \( B \) as SECRET and \( C \) as TOP SECRET. Now suppose that we have the following integrity constraint or derivation rule for \( C \):

\[
C = A \ast B
\]

This reveals the following inference problem: If \( C \) is TOP SECRET, then simply labeling it as TOP SECRET is not enough, because it can be derived from \( A \) and \( B \). Thus, analysis of the constraints leads us to the solution that at least one of \( A \) and \( B \) should be classified TOP SECRET.

This simple analysis gives the general idea behind the ideas of others (such as Denning and Morgenstern [10], Morgenstern [15,11], Hinke [6], and Thuraisingham [13]) who have proposed additional methods for locating inference problems arising from related data.

4 True Aggregation Problems

For true aggregation problems, when the aggregate class strictly dominates the class of every subset of the aggregate, as opposed to the above examples, we need a different approach. Below we discuss SeaView’s and LDV’s approaches to aggregation. TRW’s project does not address this problem.

4.1 Quantity-Based Aggregation

With so-called quantity-based aggregation problems, a collection of up to \( N \) items of a given type is not sensitive, but a collection of greater than \( N \) items is sensitive. For example, consider the familiar phone-book problem, in which the entries in the phone book for agency \( X \) have one classification, but the entire phone book, or even a set of more than \( N \) entries from the book, has a strictly greater classification.

- FALLACY: Quantity-based aggregation problems are addressed by mechanisms that notice when greater than \( N \) items are retrieved and that “upgrade” or withhold the result.

LDV has proposed mechanisms to detect in such cases when greater than \( N \) records have already been released, and if so, to refuse to release any more except at a higher security level. SeaView takes a somewhat different tack by requiring that all such records be classified high and that they be released in quantities of \( N \) or less only after sanitization by users authorized to perform such sanitization. LDV’s approach suffers from the same shortcoming noted above—a large amount of the database functionality must be relied upon to enforce this portion of the mandatory security policy. LDV’s approach also does not account for the fact that a user may have prior knowledge of a piece of information that completes the aggregate, although he or she has not retrieved it from the database. SeaView’s solution is awkward in that the data are not readily available in small quantities; users must request the data, which must then be sanitized by an authorized individual. (SeaView’s approach, however, follows the same procedure as the real world, where each phone number must be requested individually.) In the "paper and pencil" world, agency \( X \)’s phone book is indeed classified CONFIDENTIAL, and one cannot merely cut it up into small pieces, throw the pieces into a drawer (analogous to an unclassified storage object), and thereby satisfy security. Thus, SeaView’s approach matches more closely that mandated by DoD regulation. Although some believe that such data can be classified low in a computer system, in fact the proper classification of the data is governed by regulation, and we are required to label it at the mandated level and not lower. In SeaView, the individual entries in the phone book are protected at the higher level (their actual classification), and individual phone numbers are released to users with lower clearances only through sanitization. By storing the individual phone numbers in CONFIDENTIAL storage objects, the reference monitor automatically
protects them from unauthorized disclosure without any additional mechanism.

Thus, in SeaView, data that form part of a given aggregate are assigned the classification of the aggregate. The rationale for classifying and storing the data at the aggregate classification is to reduce their exposure. If the data were stored at a lower classification, as in LDV and in TRW’s A1 prototype, they would be accessible to the database system subjects operating at the lower level. If some subject were able to bypass the aggregation controls, or if the aggregate controls were flawed, the aggregate data could be disclosed to uncleared users. In LDV, the software implementing the constraints used to control aggregation comprises a significant portion of the database system, and it may be difficult to obtain assurance that the system will always recognize when an aggregation violation might occur. Because the protection of aggregates is part of the mandatory security policy, a high degree of assurance is required that the aggregate cannot be released to subjects not cleared for it. By storing the data at the aggregate classification, the data can be protected from unauthorized release using only the reference monitor—no special mechanism is needed. In addition, subsets of an aggregate can be downgraded and released to users with lower clearances only through the intervention of a properly authorized user.

Quantity-based aggregation problems, although awkwardly formulated, really intend to protect a sensitive relationship among otherwise nonsensitive data. The phonebook example as used by LDV, SeaView, and all the other database security projects should really be phrased as: the identity of individuals who work for agency X is sensitive. However, we recognize that, in reality, the identities of some agency X employees are known by some uncleared people. Thus, if an uncleared person gives the name of an agency X employee and asks for his or her phone number, the phone number can be released. It is not the case, however, that a random name and phone number can be released to a person who asks to be given a random record. Thus, the phone book problem is not in essence a quantity-based aggregation problem, but intends to protect the sensitive relationship between the names of individuals and their employer, namely, agency X. The fact that at most N phone numbers are given out at any one time does not in itself constitute a classification rule meaning that more than N numbers is classified high; it is simply a device to reduce the bandwidth of information released. With SeaView’s approach, a low user wanting an agency X phone number must request a high user to perform the query and sanitize it, thereby ensuring a low bandwidth. LDV’s approach, storing the individual phone numbers low, leaves them at risk if their “detection” mechanisms are flawed or can somehow be bypassed. In addition, even if LDV’s mechanisms work correctly, they are designed to prohibit further release of phone numbers to unclassified users once N phone numbers have been released at the unclassified level. Thus, LDV’s approach suffers from too literal an interpretation of the problem, whereas SeaView’s allows a closer approximation of the intent of the rule.

4.2 Association Among Like Entities

More general aggregation problems exist than the so-called quantity-based ones. These are not the sensitive associations we discussed in Section 3 (where, for example, the relationship between an employee’s name and salary is sensitive); neither are they quantity-based. Instead, these are general associations among the entities of the same type, i.e., among the tuples in a single relation.

For example, suppose that knowing that employees Smith, Jones, and Davis are working together on a project may reveal (or partially reveal) the SECRET mission of the project, although other project and employee information may be CONFIDENTIAL. We can store employee and project information in the following relations

\[
\text{EMPLOYEES(EMP\#, NAME, DEPT, TITLE)}
\]
\[
\text{PROJECTS(PROJ\#, PROJ-NAME, BUDGET)}
\]

which are classified CONFIDENTIAL. Figures 1 and 2 show some of the data in these relations.

---

**Figure 1: Relation EMPLOYEES: CONF**

<table>
<thead>
<tr>
<th>EMP#</th>
<th>NAME</th>
<th>DEPT</th>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>smith</td>
<td>15</td>
<td>cartographer</td>
</tr>
<tr>
<td>14000</td>
<td>jones</td>
<td>9</td>
<td>driver</td>
</tr>
<tr>
<td>10800</td>
<td>davis</td>
<td>11</td>
<td>explosives expert</td>
</tr>
</tbody>
</table>

**Figure 2: Relation PROJECTS: CONF**

<table>
<thead>
<tr>
<th>PROJ#</th>
<th>PROJ-NAME</th>
<th>BUDGET</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>product-x</td>
<td>250000</td>
</tr>
<tr>
<td>19</td>
<td>special-deliveries</td>
<td>500000</td>
</tr>
</tbody>
</table>

The association of employees and the projects they are assigned to can be stored in the following relation

\[
\text{WORKS-ON(PROJ\#, EMP\#)}
\]

where classification is at the tuple level. If WORKS-ON has all three of the tuples shown in Figure 3 (indicating that Smith, Jones, and Davis are all working together on the Special-Deliveries project), then these three tuples would be classified SECRET, whereas the other tuples in WORKS-ON would be classified CONFIDENTIAL (unless there were other projects on which Smith, Jones, and Davis work together).
In the above example, it is safest to classify the three tuples in question as SECRET, because a low user may already know, say, that Jones is working on the Special Deliveries project, and thus no "aggregation detection" mechanism would be able to protect the aggregate from compromise. For the same reason, it is safer to classify all three of the tuples at SECRET, rather than just one of them. By classifying the aggregate at the higher, aggregate, classification, we do not need any special mechanism to protect the aggregate from compromise.

Figure 3: Relation WORKS-ON

<table>
<thead>
<tr>
<th>PROJ#</th>
<th>EMP#</th>
<th>TUPLE-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>12345</td>
<td>secret</td>
</tr>
<tr>
<td>19</td>
<td>14000</td>
<td>secret</td>
</tr>
<tr>
<td>19</td>
<td>10800</td>
<td>secret</td>
</tr>
</tbody>
</table>

In the above example, the classification rule is designed to protect a sensitive project's mission. In this example, the data designer should know (or should be told) when a project's mission is sensitive.

4.3 Named Aggregates

Authorization to downgrade subsets of aggregates for release is different from authorization to make downgrading decisions. By designating that a collection of information forms an aggregate, subsets are, in effect, pre-authorized for downgrading. Thus, it is an important service to users for the database system to be able to reliably indicate whether a set of data is part of an aggregate.

An aggregate label, which the mandatory reference monitor would associate with the data forming an aggregate, would enable a privileged user to know that a basis for downgrading exists. SeaView earlier proposed the use of classification constraints to define the aggregate; i.e., to specify (1) the elements comprising the aggregate, and (2) the aggregate name. The aggregate name would be attached to the data elements comprising the aggregate. In SeaView, we hoped that these aggregate names could be handled by the mandatory reference monitor. In fact, traditional reference monitors may need to be modified to accommodate aggregate labels. The problem is that the aggregate labels are intended to be advisory in nature and do not indicate a compartment or classification of the data. Thus, for example, data labeled CONFIDENTIAL AGG=PHONE-BOOK would be treated as confidential for release purposes, but the labels displayed to the user would allow him or her to know that an individual data item has been pre-authorized for downgrading.

5 Other Security Dimensions

Gary Smith has recently published a taxonomy of security semantics—in other words, what makes data classified [16]. He identifies three dimensions of security semantics—three ways in which data can become classified. The first dimension he calls the content dimension, which refers to the data value or sensitive association to be protected. The second dimension is called the description dimension, where the classification rule may itself be classified. The third dimension is the existence dimension, where the existence of the classified data may be sensitive. Any given data item has a basis for classification along one or more of these dimensions. We have discussed the content dimension in the preceding sections. Here we examine the other dimensions.

The fact that a specific classified data item or classified data of a specific type exists may or not itself be classified. This leads us to our next "fallacy":

- FALLACY: The fact that high data exists is always classified at the level of the data.

The fact that some flights are classified SECRET may be known to uncleared users. In this case, both SECRET and UNCLASSIFIED flight data can be merged in a single multilevel relation, and a low user gains no information by observing "holes" or blanks in the relation. However, if the fact that SECRET satellites exist is itself secret, then information about SECRET satellites cannot be stored in the same multilevel relation as unclassified data about unclassified satellites; otherwise the presence of holes in a relation could allow inference of the existence of high data. This has been referred to as the missing data inference channel [8].

The SeaView project uses classification constraints to assign labels to newly entered data [1,2,9]. Thus, in SeaView, classification constraints can be viewed as integrity constraints on the classification attributes of a multilevel relation. LDV, as we have already discussed, uses classification constraints to label outgoing data (as well as incoming data). In both LDV and SeaView, inferences can arise from the application or the reading of classification constraints. This leads us to our next "fallacy":

- FALLACY: Classification rules can be uniformly applied to low as well as to high subjects.

In both LDV and SeaView, individual classification constraints are each assigned a classification. Thus, classification constraints must be appropriately classified so as to avoid inadvertently disclosing high data to low users through an inference channel. For example, if a classification constraint for the relation

\[ \text{AGG}= \text{PHONE-BOOK} \]

is confidential, then a low user should not be able to infer that the data is about a phone book. This is achieved by classifying the constraint as a classification rule for a secret attribute.

\[ \text{AGG}= \text{PHONE-BOOK} \]

With the SeaView approach, classification constraints are used to reflect the will of the user in labeling new data; the degree of trust required for mandatory access control is not needed for the classification constraint enforcement mechanisms because any constraint that assigns a class lower than the user's clearance level requires user confirmation via a trusted path. Thus, in the SeaView approach, the classification constraints are not relied on to enforce mandatory security, whereas in LDV they are.

\[ \text{AGG}= \text{PHONE-BOOK} \]
FLIGHTS(FLIGHT#, DEST, TIME)
classified all destinations as UNCLASSIFIED except for Iran, 
which it classified SECRET, then if a low subject were to 
read the constraint it could see that flights to Iran are 
SECRET. Moreover, if a low user were to try to enter a 
tuple indicating DEST = Iran, then if the constraint were 
applied, the user might be able to infer that the destina-
tion Iran is classified. In this example, the classification 
constraint concerning flights to Iran is properly classified 
as SECRET. To eliminate this potential inference channel, 
SeaView requires that no constraint can be applied whose 
classification is greater than that of the subject entering 
the data (this is automatically enforced by the mandatory 
reference monitor).

Some believe that LDV's after-the-fact application of 
the classification constraints is more secure that SeaView's 
application of the constraints to the data being entered. 
However, with LDV's approach, a low user may be able 
to infer high information from the results of his or her 
queries, because different information will be released de-
pending on the context in which the query was posed. Thus, 
the possibility exists that LDV may enable inferences to 
be made from the enforcement of the security constraints 
themselves.3 (The LDV verification is attempting to prove 
that there is no way of exploiting this as a covert channel.)

If the basis for classification is known, then it is easier to 
determine which are true inference problems. None of the 
various proposed approaches for detecting inference prob-
lems addresses these distinctions. Any inference detection 
tool needs to address the problem of inferences arising from 
improper classification of the security constraints, and also 
to distinguish when the existence of high data is classified.

6 Conclusions

This paper has distinguished several different types of ag-
gregation and inference problems and has shown that the 
different types of problems are best addressed by different 
approaches. This can be summarized as follows. 1. Sensi-
tive associations among entities of different types—that is, 
between a tuple of one relation and the tuples of another 
(others have called these "context-dependent aggregation 
problems")—are best treated by representing the sensitive 
association separately and classifying the individual enti-
ties low and the relationship high. 2. Sensitive associations 
among the various properties of an entity—that is, among 
the columns of a relation—are best treated by determining 
those properties that contribute most to the inference and 
by storing those separately at a higher classification. 3. 
Sensitive associations among entities of the same type— 
that is, among tuples in a given relation (this includes 
quantity-based aggregation problems)—are best treated by 

3 Although it is premature to discuss performance of the various 
approaches, it is worth noting that LDV's constraints are used every 
time data are accessed, whereas SeaView's are applied just once, when 
data are entered.

storing the individual data items comprising the aggregate 
at the aggregate-high classification; they must be sanitized 
for release to lower-level users.

The suggested approaches allow the mandatory reference 
monitor to protect the sensitive associations, with no 
additional trusted mechanism needed. This is not only the 
most secure approach possible, but most closely complies 
with the intent of the classification rules.

Further work is called for in extending previously pro-
posed ideas for an inference detection tool to take into ac-
count Smith's multidimensional model of security seman-
tics.

Acknowledgments

This research was supported by the U. S. Air Force, Rome 
Air Development Center (RADC), under contract F30602- 
85-C-0243. I am indebted to both RADC and to the Na-
tional Computer Security Center for making this work pos-
sible.

I also thank Tom Haigh and Paul Stachour of Honeywell 
who clarified some of my misconceptions about LDV and 
whose comments greatly improved this paper.

References

Schell, M. Heckman, and W. R. Shockley. Final Re-
port Vol. 1: Security Policy and Policy Interpretation 
for a Class A1 Multilevel Secure Relational Database 
System. Technical Report, Computer Science Labora-

SeaView Formal Security Policy Model. Technical 
Report, Computer Science Laboratory, SRI Interna-
tional, Menlo Park, California, 1989.

and D. Warren. A near-term design for the SeaView 

Database Management System. Interim Report A002, 
Honeywell Systems Research Center and Corporate 
Systems Development Division, April 1987.

Thuraisingham. Secure Distributed Data Views— 
Implementation Specifications. Interim Report A005, 
Honeywell Systems Research Center and Corporate 

[6] T. H. Hinke. Inference aggregation detection in 
database management systems. In Proceedings of the 
1988 IEEE Symposium on Security and Privacy, April 


