Aggregation in Relational Databases: Controlled Disclosure of Sensitive Information

Amihai Motro, Donald G. Marks, and Sushil Jajodia

Department of Information and Software Systems Engineering
George Mason University
Fairfax, VA 22030-4444

Abstract. It has been observed that often the release of a limited part of an information resource poses no security risks, but the relase of a sufficiently large part of that resource might pose such risks. This problem of controlled disclosure of sensitive information is an example of what is known as the *aggregation* problem. In this paper we argue that it should be possible to articulate specific secrets within a database that should be protected against overdisclosure, and we provide a general framework in which such controlled disclosure can be achieved. Our methods foil any attempt to attack these predefined secrets by disguising queries as queries whose definitions do not resemble secrets, but whose answers nevertheless "nibble" at secrets. Our methods also foil attempts to attack secrets by breaking queries into sequences of smaller requests that extract information less conspicuously. The accounting methods we employ to thwart such attempts are shown to be both accurate and economical.

1 Introduction

The most common approach to secrets is to specify the information that must be protected, and to devise mechanisms that forbid disclosure of *any* of this information to unauthorized users. In the environment of relational databases, secret information is often defined via *views*. Models for *multi-level security* are used to classify both users and information according to a variety of secrecy levels, and *authorization* algorithms ensure that classified information is made available only to users with the appropriate classification [1].

However, it has been observed that often the release of a *limited* part of an information resource poses no security risks, but the relase of a sufficiently large part of that resource might pose such risks. In the environment of relational databases, this implies that the release of a limited number of tuples would be permitted, but once this number exceeds a predetermined threshold, security might be breached. This problem of controlled disclosure of sensitive information is an example of what is known as the *aggregation* problem [5, 3, 6, 7, 2, 4].

A typical example is that of the Secret Government Agency (SGA) Phonebook. In this example, the entire phonebook is a classified document that is not

The work of Motro was supported in part by NSF Grant No. IRI-9007106 and by ARPA grant, administered by the Office of Naval Research under Grant No. N0014-92-J-4038. The work of Jajodia was supported in part by NSF Grant No. IRI-9303416.

available without the appropriate clearance; yet, individual phonebook entries are available to inquiring callers. In theory this may appear to be contradictory, because with a sufficient number of queries it should be possible to extract all the information in the phonebook. In practice, however, these queries are handled by operators who can recognize repetitive querying; also, the low bandwidth of these extractions serve to protect the phonebook from substantial disclosures. A small example of such a phonebook is shown in Fig. 1.

Name	Tel	Div	Mail	Bldg	Room
A. Long	x1234	A	m404	1	307
P. Smith	x1111	B	m303	2	610
E. Brown	x2345	B	m101	3	455
C. Jones	x1234	A	m202	1	307
M. Johnson	x1234	B	m101	3	103
B. Stevenson	x2222	A	m202	1	305
S. Quinn	x2222	C	m606	3	101
R. Helmick	x1234	A	m404	1	307
A. Facey	x1122	C	m505	2	400
S. Sheets	x2345	B	m101	3	103

Fig. 1. The Phonebook Example

An approach for preventing overdisclosure of such a database is to *monitor* the number of database tuples that are being given away [2]; in this example, where the total number of tuples is 10, it could be determined that only 3 tuples should be disclosed to the same user.

Yet, it is conceivable that we would be willing to disclose an *unlimited* number of telephone numbers, but only a limited number of division affiliations (say, because these affiliations disclose the amount of effort devoted to particular projects). Similarly, we might be willing to disclose any number of telephone numbers, but limit the disclosure of telephone numbers of employees in division A (because we would like to prevent users from estimating the size of this division). In other words, rather than protect the entire database from overdisclosure, we argue that it should be possible to articulate those specific aggregates that are sensitive, and protect only these aggregates from overdisclosure.

In this paper we describe a scheme for aggregation control that provides this flexibility. Our scheme uses general *views* to define the secrets that can be disclosed to a limited degree, and associates a threshold value with each such view. We shall refer to these views as *sensitive concepts* (or simply *concepts*). Clearly, if a view is defined to be a concept, then any view that incorporates this concept (a “larger” concept) should be protected as well.

This approach allows to define a variety of sensitive concepts. For example, with the database of Fig. 1:

1. To limit the disclosure of the total number of employees to five, the concept π_{Name} is assigned the threshold 5.
2. To prevent disclosure of information regarding the division to which employees belong, the concept $\pi_{Name,Div}$ is assigned the threshold 0. Note that any view that includes the attributes *Name,Div* would be protected as well!
3. To limit the disclosure of occupants in Building 1 to at most three employees, the concept $\pi_{Name}\sigma_{Bldg=1}$ is assigned the threshold 3.

Attacks on sensitive concepts (i.e., secrets that may be disclosed partially) may be disguised by either of two strategies (or a combination of both):

1. Queries are broken down into sequences of *smaller* requests that extract information less conspicuously.
2. Queries are disguised as other queries whose definitions do not resemble secrets, but whose answers nevertheless extract information covered by secrets.

Any system that permits controlled disclosure of secrets must be able to recognize both of these strategies.

Consider the above concept limiting to 3 the disclosure of occupants of building 1. A query to list the names and buildings of all employees at mailstop m202 retrieves 2 employees in Building 1 (Jones and Stevenson). A second query to list the names, telephone numbers and buildings of employees at room 307 retrieves 3 employees (Long, Jones and Hemlick). Any mechanism for protecting concepts must recognize that these apparently dissimilar queries in effect might be "nibbling" at the same sensitive concept.

The first strategy can be foiled by keeping track of the total number of tuples from a concept that have already been disclosed to every user. Thus, it is mostly a question of continuous accounting. Yet, this accounting is often subject to gross inaccuracies. When queries Q_1 and Q_2 retrieve n_1 and n_2 tuples, respectively, the number of tuples disclosed would be taken as $n_1 + n_2$. In effect, if the queries overlap, the number is smaller; for example, if $Q_1 = Q_2$ (the same query is repeated), the number of tuples disclosed is only n_1. In the previous example, the number of building 1 occupants disclosed by both queries is not 5, but 4, because Jones was included in both queries. This inaccuracy may be corrected by keeping record of the *actual* tuples that have been disclosed; for example, by maintaining a set of tuple identifiers. The disadvantage is that with very large databases such record keeping becomes very costly.

The method we describe foils the first strategy by continuous accounting, which is precise yet economical: it keeps accurate account of the number of tuples disclosed without maintaining the entire set of these tuples.[1]

The second attack strategy poses a more serious challenge. As an example, assume that the set of employees working in division A is a sensitive concept and consider a query on the employees in building 1. Apparently, this query is unrelated to the concept, and therefore should be allowed. Yet, the set of

[1] Incidentally, since we wish to protect arbitrary views, it would not have been even possible to use tuple identifiers to keep track of the tuples that have been disclosed.

employees in building 1 is identical to the set of employees in division A. By answering this query, the system will be giving away the *contents* of a sensitive concept, although the *definition* of this set of values (as formulated by the user) is quite different from its definition as a sensitive concept.

Consider a database view V and its materialization v in a specific database instance. V and v are often called, respectively, the *intensional definition* (or simply intension) and the *extensional definition* (or simply extension) of a view.[2] Given V (and assuming the proper permissions), v is easily and uniquely determined. It is much more difficult, however, to determine V from v. Indeed, there may be numerous intensional definitions of a view that evaluate to the same view extension. The problem of finding for a given view intension V another view intension V' that shares the same extension v is known as *intensional answering* [9]. In that context, the finding of alternative intensional definitions to a query is considered *cooperative* behavior: provide users with additional characterizations of their answers. In this context, finding alternative intensional definitions to a query is intended to be *uncooperative*: given a query, the system would search for alternative intensional definitions that "cover" the same information as the query, but correspond to sensitive concepts; if found, the user may be attempting to attack a concept using the second strategy.

Given a query, our method will detect whether it trespasses a sensitive concept, regardless of the statement of the query. Thus, our method thwarts the second strategy as well.

In summary, we provide a general framework in which sensitive information can be defined flexibly. To protect sensitive concepts from excessive disclosure, for each concept and for each user our method keeps track of the number of tuples that has already been disclosed. Each incoming query is compared with each predefined concept to determine whether it might trespass that concept. If so, the number of *new* tuples thus disclosed is computed and added to the number of tuples already disclosed, and, depending on the relationship of this counter to the threshold, the query is either permitted or rejected. Our accounting is both accurate and economical.

Our discussion is limited to databases that are single relations and to queries and concepts that are projection-selection views, where all selections are conjunctions of simple clauses of the form *attribute* = *value*. We conjecture that our method can be extended to overcome these limitations.

Section 2 establishes the formal framework with definitions of queries, concepts and concept disclosure by queries. Section 3 describes a basic algorithm for disclosure control, and Section 4 describes two improvements. Section 5 concludes this paper with a brief summary and discussion of future research directions.

[2] The terms are derived from the notions of *intensive* and *extensive* descriptions of information [10].

2 The Model

We adopt the usual definition of relational databases, but restrict our attention to databases that are *single* relations, and to *projection-selection* views, where all selections are conjunctions of simple clauses of the form *attribute* = *value*. We denote the database scheme $R = (A_1, \ldots, A_n)$.

2.1 Queries, Concepts and Patterns

A *query* is a view. Its extension in the present database instance is the *answer* to the query. Queries are defined by users and describe the information they are seeking. A *concept* is also a view. Concepts are defined in the system and describe the information that needs to be protected.

Views (queries or concepts) may be syntactically different, but yet describe the same information. Consider the example database scheme *Emp = (Name, Tel, Div, Mail, Bldg, Room)* and these two views:

1. $\pi_{Name,Room}\sigma_{(Room=103)\wedge(Div=\mathrm{B})}$
2. $\pi_{Name}\sigma_{(Room=103)\wedge(Div=\mathrm{B})}$

Both view definitions are identical, except that the latter view does not project a selection attribute which is projected by the former (*Room*). Nevertheless, because the values of selection attributes are known (in this case, the constant value 103), there is no difference in the information these views describe. Consequently, we shall always assume that views are defined in their *expanded* form, where the projection attributes include all the selection attributes. Thus, in the above example, both views would be interpreted as

$$\pi_{Name,Div,Room}\sigma_{(Room=103)\wedge(Div=\mathrm{B})}$$

A *pattern* is a formal notation for views. A pattern is an n-tuple $p_1, \ldots, p_n$, where n is the number of attributes in the database scheme, and each p_i is defined as follows

$$p_i = \begin{cases} a \text{ if the selection formula includes } A_i = a \\ * \text{ if } A_i \text{ is a projection attribute which is not a selection attribute} \\ - \text{otherwise} \end{cases}$$

Because the projection attributes are assumed to include the selection attributes, patterns record only the projection attributes that are *not* selection attributes.

For example, all three view definitions above are represented by this pattern

$$(*, -, \mathrm{B}, -, -, 103,)$$

Note that $*$ indicates an attribute of the database which is *unaffected* by the view: it is neither restricted nor removed. This notation resembles the notation for meta-tuples used in [8].

2.2 Concept Disclosure

Let U and V be views of database scheme D.

The selection condition of U is *at least as restrictive* as the selection restriction of V, if every clause $A_i = a$ in V's selection condition also appears in U's selection condition. The selection conditions of U and V are *contradictory*, if U's selection condition includes the clause $A_i = a$ and V's selection condition includes the clause $A_i = b$, for some attribute A_i and two different constants a and b.

U *overlaps* V, if their selection conditions are not contradictory, and U's projection attributes contain V's projection attributes. When U overlaps V, then the extension of U could be processed by another view that will remove the extra attributes. Some of the resulting tuples may be in the extension of V.

U *overlays* V, if U's selection condition is at least as restrictive as the selection condition of V, and U's projection attributes contain V's projection attributes. Obviously, when U overlays V, it also overlaps V. However, when the extension of U is processed by another view that removes the extra attributes, *all* the resulting tuples will be in the extension of V.

The overlap and overlay relationships are illustrated schematically in Fig. 2, in which U_1 overlaps V and U_2 overlays V.

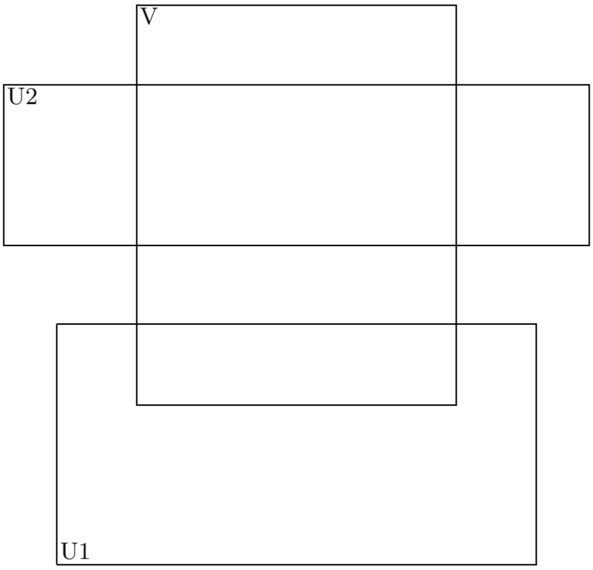

Fig. 2. Overlapping and Overlaying

Assume that U overlaps V. The *restriction* of V to U, denoted $V \mid U$, is the view obtained from V by appending to its selection condition the selection condition of U. The *exclusion* of U from V, denoted $V \mid \neg U$, is the view obtained from V by appending to its selection condition the *negation* of the selection condition of U.[3] Obviously, $V = (V \mid U) \cup (V \mid \neg U)$.

Let C be a concept view and let Q be a query view. Q *discloses* C, if Q overlaps C. Intuitively, a query discloses a concept, if its result could be processed by another query, to possibly derive tuples from the protected concept.

As an example, with the previous database scheme, consider this concept

$$C = \pi_{Name,Div,Room}\sigma_{(Room=103)\wedge(Div=\mathrm{B})}$$
(names of those in division B and in room 103)

and these three queries

1. $Q_1 = \pi_{Name,Tel,Div,Room}\sigma_{(Room=103)\wedge(Div=\mathrm{B})\wedge(Tel=\mathrm{x2345})}$
(names of those in room 103, in division B, and with telephone x2345)
2. $Q_2 = \pi_{Name,Div,Room}\sigma_{Div=\mathrm{B}}$
(names and rooms of those in division B)
3. $Q_3 = \pi_{Name,Div,Room}\sigma_{Room=102}$
(names and divisions of those in room 102)

Q_1 discloses C, because applying the query $\pi_{Name,Div,Room}$ to the result of Q_1 may yield some tuples in C. Q_2 discloses C in its entirety, because applying the query $\sigma_{Room=103}$ to the result of Q_2 yields all the tuples of C. Q_3 does not disclose any tuples of C because their selection conditions contradict.

The disclosure relationship between a query and a concept is illustrated schematically in Fig. 3. Notice that a concept protects its tuples, but not its subtuples; i.e., a query on a *subset* of the concept's projection attributes does not disclose the concept. On the other hand, a query on a *superset* of the attributes would disclose the concept (unless their selection conditions are contradictory).

As mentioned earlier, disclosure control requires that the number of tuples disclosed from a given concept does not exceed a certain predetermined number. For each concept C we define three integer values called *concept total*, *concept threshold* and *concept counter*, and denoted respectively, N, T and D. N denotes the total number of tuples in the extension of this concept, T denotes the maximal number of tuples that may be disclosed from this concept, and D denotes the number of tuples from this concept that have already been disclosed. If $T \geq N$, then the concept is *unrestricted*; we shall assume that none of the concepts are unrestricted. As queries are processed, the database system must keep track of D to ensure that $D \leq T$. The number of tuples in the extension of a view V will be denoted $\|V\|$; e.g., $\|C\| = N$.

[3] Note that the resulting selection condition is no longer a simple conjunction.

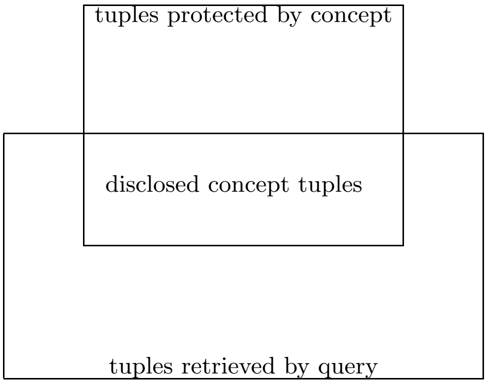

Fig. 3. Disclosure relationships between a query and a concept

3 Aggregation Control

Given a concept C with its three counters and given a query Q, our goal is to determine whether the request should be satisfied or not, and update the counters as appropriate. The main problem is to determine *how many tuples of C does Q disclose?* Once this question is answered, the rest is mostly bookkeeping.

To answer this question, we consider the patterns that represent Q and C. Let

$$C = (c_1, \ldots, c_n)$$
$$Q = (q_1, \ldots, q_n)$$

We establish a relationship between each element of the query pattern q_i and the corresponding element of the concept pattern c_i. Recall that a pattern element could be a constant or $*$ or $-$. A constant a indicates that A_i is a selection attribute and the selection condition is $A_i = a$, $*$ indicates that A_i is a projection attribute which is not a selection attribute, and $-$ indicates that A_i is neither a selection attribute nor a projection attribute.

We now define overlapping and restriction at the level of pattern elements. An element q_i of the query pattern *overlaps* the corresponding element c_i of the concept pattern, if either

1. q_i is a constant and c_i is either the same constant or $*$ or $-$.
2. q_i is $*$.
3. q_i and c_i are both $-$.

Thus, the query element q_i overlaps the concept element c_i in all but two situations: (1) when the query and the concept have contradictory selection conditions, or (2) when the attribute A_i is protected by the concept, but not requested by the query. Intuitively, if q_i overlaps c_i, and all the other pattern elements in both the query and the concept are $*$ (reflecting unaffected attributes), then the query Q overlaps the concept C.

Assume that q_i overlaps c_i. The *restriction* of c_i to q_i is defined as follows:

$$r_i = \begin{cases} q_i \text{ if } q_i \text{ is a constant} \\ c_i \text{ otherwise} \end{cases}$$

Intuitively, if the query element q_i overlaps the concept element c_i, and all the other pattern elements in both the query and the concept are $*$ (reflecting unaffected attributes), then the pattern with r_i in position i and $*$ everywhere else describes the restriction of concept C to the query Q.

In general, a query Q overlaps a concept C, if all their corresponding pattern elements overlap. Similarly, the restriction of a concept C to a query Q is obtained from the restrictions of the corresponding pattern elements. Thus, by considering "in parallel" all the pattern elements, we can determine whether Q discloses C, and define the precise subview of C that is disclosed by Q. This discussion is summarized in the following theorem.

Theorem (disclosure). *Let C be a concept with pattern $(c_1, \ldots, c_n)$ and Q a query with pattern $(q_1, \ldots, q_n)$. Then*

1. *Q possibly discloses tuples of C, if q_i overlaps c_i, for all $1 \leq i \leq n$.*
2. *The set of C tuples disclosed by Q is given by the pattern $(r_1, \ldots, r_n)$, where r_i is the restriction of c_i to q_i.*

This theorem suggests a basic algorithm for disclosure control, shown in Fig. 4. The input to this algorithm is a set $C_1, \ldots, C_m$ of protected concepts, each with its associated counters N_i, T_i and D_i, and a query Q. The algorithm also uses a temporary counter M_i for each concept. When it terminates, the value of *permit* indicates whether the answer to Q should be presented to the user or not.

Essentially, the overhead incurred in authorizing Q is the determination whether Q overlaps C_i and, if it does, the derivation of $C_i \mid Q$ from Q. As the theorem suggests, overlapping is discovered by a simple comparison of the patterns. The derivation of the disclosed tuples $C_i \mid Q$ from the answer set Q is also quite simple. Altogether, the complexity of this algorithm (excluding the cost of materializing Q) is $O(m \cdot n \cdot k)$, where m is the number of concepts, n is the number of attributes, and k is the size of the answer.

Referring to the example in Fig. 1, there are 4 employees in division A. Assume that only 3 employees in this division may be disclosed, and that one such employee has already been disclosed. The concept pattern and counters are

$$\begin{aligned} C_1 &= (*, *, \mathrm{A}, *, *, *) \\ N_1 &= 4 \\ T_1 &= 3 \\ D_1 &= 1 \end{aligned}$$

```
Algorithm (disclosure)
permit := true
materialize $Q$
$i := 0$
while permit and $i < m$
do
    $i := i + 1$
    $M_i := 0$
    if $Q$ overlaps $C_i$
    then
        $M_i := \|C_i \mid Q\|$
        if $D_i + M_i > T_i$
        then
            permit := false
            break
        endif
    endif
done
if permit
then
    for $i = 1, \ldots m$
    do
        $D_i := D_i + M_i$
    done
endif
```

Fig. 4. Basic algorithm for disclosure control

Consider now the query that requests complete information on the employees whose telephone number is x1234 and whose mail stop is m404. The query and query pattern are

$$\sigma_{(Tel=\text{x1234}) \wedge (Mail=\text{m404})}$$
$$Q = (*, \text{x1234}, *, \text{m404}, *, *)$$

The restriction of C_1 to Q is described by the pattern

$$R = (*, \text{x1234}, \text{A}, \text{m404}, *, *)$$

R's extension has two tuples, so the query is accepted and D_1 is updated to 3.

As another example, the telephone number x1234 is assigned to 4 employees. Assume that only 3 employees with this number may be disclosed, and that two have already been disclosed. The concept pattern and counters are

$$C_2 = (*, \text{x1234}, -, -, -, -)$$
$$N_2 = 4$$
$$T_2 = 3$$
$$D_2 = 2$$

Consider now the query that requests the location of the telephone whose number is x1234. The query and query pattern are

$$\pi_{Tel,Bldg,Room}\sigma_{Tel=\text{x1234}}$$
$$Q = (-, \text{x1234}, -, -, *, *)$$

Q does not overlap C_2, so it is permitted.

4 Improvements

Assume a concept C. First, consider a query Q_1 that overlaps C by m_1 tuples. Algorithm **disclosure** increments the counter D of disclosed tuples by m_1. Consider now a second query Q_2 that overlaps C by m_2 tuples. The algorithm will increment D by m_2. This continues until D reaches the threshold value T, when further queries that overlap C would be denied.

Yet, it is entirely possible that some of (or all) the tuples disclosed by Q_2 have already been disclosed by Q_1. In other words, possibly the user is being "charged" twice for the same tuples, and is thus approaching the threshold faster than warranted.

To rectify this, we offer the following improvement. With each concept C we associate a *predicate* P that describes the concept tuples that have already been disclosed. P is initialized to *true*. Assume that $Q_1, \ldots, Q_p$ have already been processed when Q_{p+1} is received, and let $\alpha_1, \ldots, \alpha_p$ denote their respective selection conditions. The present value of P would be $\alpha_1 \vee \cdots \vee \alpha_p$. After computing the restriction of C to Q_{p+1}, we exclude from it the view σ_P. The tuples in this new query are those that have *not* been delivered already. This improvement is incorporated into a new algorithm, shown in Fig. 5.

The input to this algorithm is a set $C_1, \ldots, C_m$ of protected concepts, each with its associated predicate P_i and counters N_i, T_i and D_i, and the query Q whose selection predicate is α. When the algorithm terminates, the value of *permit* indicates whether the answer to Q should be presented to the user or not.

Again, the overhead incurred in authorizing Q is the determination whether Q overlaps C_i and, if it does, the derivation of $(C_i \mid Q) \mid \neg\sigma_{P_i}$ from Q. Again, these are simple procedures, and the complexity of the algorithm is $O(m \cdot n \cdot k \cdot p)$, where n, m and k are as before, and p is the number of queries already processed.

We have assumed that the collection of protected concepts $C_1, \ldots, C_m$ is essentially *unstructured* and have ignored any possible relationships among these concepts. At times, the concepts to be protected form specific structures; recognizing these structures could help improve the performance of the disclosure control algorithms.

Assume an organization with three divisions called A, B and C, and the following limitations on disclosure: 20 employees in division A, 15 in division B, 10 in division C, but not more than 30 employees in total. These limitations are described in four concepts:

```
**Algorithm (improve1)**
$permit := true$
materialize $Q$
$i := 0$
while $permit$ and $i < m$
do
    $i := i + 1$
    $M_i := 0$
    if $Q$ overlaps $C_i$
    then
        $M_i := \|(C_i \mid Q) \mid \neg\sigma_{P_i}\|$
        if $D_i + M_i > T_i$
        then
            $permit := false$
            break
        endif
    endif
done
if $permit$
then
    for $i = 1, \ldots m$
    do
        $P_i := P_i \wedge \alpha$
        $D_i := D_i + M_i$
    done
endif
```

Fig. 5. Disclosure control algorithm with accurate bookkeeping

1. $C_1 = \pi_{Name,Div}\sigma_{Div=\mathrm{A}}$
   $T_1 = 20$
2. $C_2 = \pi_{Name,Div}\sigma_{Div=\mathrm{B}}$
   $T_2 = 15$
3. $C_3 = \pi_{Name,Div}\sigma_{Div=\mathrm{C}}$
   $T_3 = 10$
4. $C_4 = \pi_{Name}$
   $T_4 = 30$

With respect to the mutual relationships of these four views, we note that

1. Each of the concepts C_1, C_2, and C_3 *overlays* the concept C_4.
2. The restrictions $C_4 \mid C_1$, $C_4 \mid C_2$, $C_4 \mid C_3$ *partition* C_4.

Thus, every disclosure from one of first three concepts corresponds to a disclosure from C_4 (and disclosure from C_4 corresponds to a disclosure from exactly one of the first three concepts). Consequently, the disclosure control algorithm only needs to compare a query Q against the first three concepts. Increments of D_1,

```
Algorithm (improve2)
materialize $Q$
$permit := true$
for every leaf concept $C_i$
do
    $M_i := 0$
    if $Q$ overlaps $C_i$
    then
        $M_i := \|(C_i \mid Q) \mid \neg\sigma_{P_i}\|$
        if $D_i + M_i > T_i$
        then
            $permit := false$
            break
        else
            for every ancestor $C_j$ of $C_i$
            do
                $M_j := M_i$
                if $D_j + M_j > T_j$
                then
                    $permit := false$
                    break
                endif
            done
        endif
    endif
    if $\neg permit$
    then
        break
    endif
done
if $permit$
then
    for $i = 1, \dots m$
    do
        $D_i := D_i + M_i$
        if $C_i$ is leaf concept
        then
            $P_i := P_i \wedge \alpha$
        endif
    done
endif
```

Fig. 6. Disclosure control algorithm with concept hierarchy

D_2 or D_3 should also trigger identical increments to D_4 (and a comparison of D_4 with T_4).

This improvement is incorporated into a new algorithm, shown in Fig. 6. The input to this algorithm is a *hierarchy* $C_1, \ldots, C_m$ of protected concepts, each with its associated predicate P_i and counters N_i, T_i and D_i, and the query Q whose selection predicate is α. When the algorithm terminates, the value of *permit* indicates whether the answer to Q should be presented to the user or not. The complexity of the algorithm is $O(m \cdot n \cdot k \cdot p + m^2)$.

Note that it is not necessary to maintain the predicates P_i for non-leaf concepts: tuples newly disclosed from leaf concepts are guaranteed to be newly disclosed from ancestor concepts.

5 Conclusion

We addressed the problem of controlled disclosure of sensitive information. We defined a model in which any view of the database can be defined as a sensitive concept, and we offered simple and efficient algorithms that accurately monitor the disclosure of these predefined concepts. With these algorithms, any query by a user of the database is noted for its effect on the set of predefined concepts; any "nibble" into a concept is recorded, and once these "nibbles" add up to a substantial part of a concept (as defined by a threshold), future queries are rejected. Even queries that are apparently unrelated to sensitive concepts are monitored for their effect on these concepts, thus foiling any strategy of disguising queries through alternative formulations.

Much work remains to be done, and we mention here several directions. First, we are interested in extending this work to remove the simplifying assumptions that have been made on the relations and on the definitions of concepts and queries. Also, the selection of concepts and thresholds needs to be considered more carefully. For example, thresholds must be assigned *consistently*; e.g., the threshold for a "broader" concept must be larger than the threshold for any of its "subconcepts".

Our discussion has been limited to "static" databases; i.e., when considering a sequence of queries by the same user, we assumed that the extensions of concepts do not change via insertions or deletions of tuples. Further research is required to extend this work to "dynamic" databases.

Finally, we assumed that all sensitive information has been *predefined* as concepts, and challenged every attack against these concepts. Hence, we can only detect attacks on information that has already been recognized as sensitive. A more challenging direction is to conclude from users queries whether they are attempting to "converge" on a concept which so far has been unclassified, thus alerting the system to the possibility of security "holes".

Acknowledgement. The authors are grateful to the anonymous referees for their important corrections and suggestions.

References

1. D. E. Denning. *Cryptography and Data Security.* Addison Wesley, Reading, Massachusetts, 1982.
2. J.T. Haigh, R.C. O'Brian, P.D. Stachour, and D.L. Toups. The LDV approach to security. In D.L. Spooner and C. Landwehr, editors, *Database Security III: Status and Prospects*, pages 323–339. North Holland, Amsterdam, 1990.
3. T.N. Hinke. Inference aggregation deduction in database management systems. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 96–106, April 1988.
4. S. Jajodia. Inference problems in secure database management systems. Technical Report MTR 92W0000052, The MITRE Corporation, McLean, Virginia, June 1992.
5. T.Y. Lin. Database, aggregation and security algebra. In *Proceedings of the 4th IFIP Working Conference on Database Security*, September 1990.
6. T.F. Lunt. Aggregation and inference: Facts and fallacies. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 102–109, May 1989.
7. T.F. Lunt and R.A. Whitehurst. The Sea View formal top level specifications. Technical report, Computer Science Laboratory, SRI International, February 1988.
8. A. Motro. Integrity = validity + completeness. *ACM Transactions on Database Systems*, 14(4):480–502, December 1989.
9. A. Motro. Intensional answers to database queries. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):444–454, June 1994.
10. D. C. Tsichritzis and F. H. Lochovsky. *Data Models.* Prentice Hall, Englewood Cllifs, New Jersey, 1982.