ELSEVIER Information Sciences 113 (1999) 131–146

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

Auditing user queries in dynamic statistical databases

Shiuh-Pyng Shieh *, Chern-Tang Lin

Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan

Received 1 March 1997; received in revised form 21 December 1997; accepted 12 April 1998

Communicated by Ben Wah

Abstract

Chin proposed an audit scheme for inference control in statistical databases (SDBs) which can determine whether or not a query will lead to the compromise of an SDB. As Chin points out that the dynamic updates of an SDB are prohibited in this scheme because, otherwise, the time and storage requirements will become infinite. The restriction limits the use of this scheme since many SDBs need to be dynamically updated. In this paper, we propose an algorithm to remove this restriction so that updates can be allowed. We also propose an efficient audit scheme for dynamic SDBs which requires less time and storage requirements, and does not have the space explosion problem that appears in Chin's scheme. © 1999 Elsevier Science Inc. All rights reserved.

Keywords: Statistical database; Inference control; Security

1. Introduction

A statistical database (SDB) is a database that contains sensitive records describing individuals but only statistical information is available. SDBs are mainly used for statistical analysis where only statistical queries, such as SUM, AVERAGE, COUNT are available and information of individuals cannot be disclosed. SDBs are used in many applications, such as census data,

* Corresponding author. Tel.: 886 3573 1876; fax: 886 3572 4176; e-mail: ssp@csie.nctu.edu.tw

PII: S0020-0255(98)10044-0

mortality data, and economic planning. A typical example of SDB is illustrated in Fig. 1. In the SDB, the scores of individuals should not be disclosed, and therefore AVERAGE(ID = 1, Score), the average score of students with ID 1, is an illegal query. But statistical queries, such as COUNT(ALL) and AVERAGE(Address = "New York", Score) are legal. Although users are only allowed to access the statistical information from an SDB, they can infer the confidential individual information by invoking a series of legal queries. When any confidential information is disclosed, the SDB is *compromised.* For example, both AVERAGE(Address = "New York", Score) and AVERAGE (Dept. = "C.S.", Score) are legal queries. A user can infer the confidential information (the score of ID 3) by computing the difference between these two queries. If both queries are answered, the SDB will be compromised. Therefore, the SDB should deny one of the two queries to protect the individual information.

In practice, many SDBs are *dynamic*. That is, the individual records of an SDB need to be inserted, deleted and updated dynamically to keep statistical information fresh. A user may infer confidential information from the updates of a dynamic SDB. For example, when invoking the query AVERAGE(Gender = "M", Score) before and after inserting a new record with gender "M" into the SDB shown in Fig. 1, the invoker can infer the new record's score from the change of the answers. Therefore, not only the old and the new values of an individual, but also the change of an SDB should be protected.

There are many inference control methods proposed to protect various database systems, such as multilevel security database [1–3]. Those methods for SDBs can be classified into three classes: *conception*, *perturbation*, and *query restriction*. The conceptual model provides a framework for investigating the security problem at the conceptual-data-model level [4]. A popular approach for the conceptual model is the lattice model [5,6]. This model presents a framework for better understanding and investigating the security problem of SDBs, but gives too many constraints for users. Perturbation approaches [7–13] introduce noise in the data, or perturb the answer to user queries while leaving the data in the SDB unchanged. These approaches cannot provide precise answers to users. Query restriction methods impose extra restriction on queries which includes restricting the query set size [14], controlling the overlap among suc-

ID	Gender	Address	Dept.	Score
1	F	New York	C.S.	82
2	M	Washington	M.E.	75
3	M	Washington	C.S.	71
4	F	New York	C.S.	83

Fig. 1. A statistical database.

cessive queries [15], auditing [16], partitioning [17,18] and suppressing cells [19]. Some of them cannot guarantee high security assurance, while others limit the usefulness of the SDBs.

Chin et al. proposed an inference control scheme, Audit Expert [16], which uses the query restriction approach. Audit Expert maintains a matrix to audit the history of user's queries and check if a new query will lead to the compromise of an SDB. Audit Expert can provide high assurance of security of SDBs, and need not impose extra restriction on user queries. Chin points out that Audit Expert is only applicable to static SDBs. In a dynamic SDB where individual data is dynamically updated, the audit matrix will be full of garbage columns and rows and its size may become infinite. Consequently, the time and storage requirements for the analysis of the audit matrix are quite high. Audit Expert suffers from the time and storage space explosion problem and thus is not applicable to dynamic SDBs. Since many SDBs are dynamic, this restriction limits the use of Audit Expert. In this paper, we investigate how to remove the restriction on the use of Chin's Audit Expert, and then propose an efficient audit scheme which requires less storage and time for the statistical analysis. This new audit scheme not only provides high security assurance and imposes no extra restriction on user queries, but also is applicable to dynamic SDBs.

This paper is organized as follows. In next section, Chin's scheme is introduced and a new method for reducing its time and space requirements is proposed. With the proposed method, Chin's scheme can be extended so that it can be used in dynamic SDBs. In Section 3, we propose a new audit scheme which can protect dynamic SDBs in a more efficient way. Section 4 discusses the updates of a dynamic SDB in our scheme. In Sections 5 and 6, we analyze the complexity of the proposed scheme, and give the conclusions.

2. Chin's scheme and the enhancement

In Chin's scheme, the SDB consists of n individuals x_i, $1 \leqslant i \leqslant n$. For notational simplicity, each individual x_i is assumed to have a single protected numerical attribute value, and each answered query reveals a set of individual records $\{x_i, x_j, x_k, \ldots\}$. Hence, each answered query can be represented by a vector $(a_1, a_2, \ldots, a_n)$, where $a_i = 1$, if x_i is accessed in this query. The user's knowledge space KS is the vector space spanned by the set of vectors of answered queries AQ. Formally, KS has the following properties.

1. If $\overline{q} \in AQ$, then $\overline{q} \in KS$.
2. If $\overline{q} \in KS$, then $b\overline{q} \in KS; b$ is a real number.
3. If $\overline{q_1}, \overline{q_2} \in KS$, then $\overline{q_1} + \overline{q_2} \in KS$.
4. Nothing else is in KS.

KS can be represented by a maximal set of nonredundant vectors of AQ. For example in Fig. 1,

$\overline{q_1} = (1,0,0,1)$ (SUM of the scores of the people living in NewYork),
$\overline{q_2} = (1,0,1,1)$ (SUM of the scores of the people majoring in C.S.).
We have

$$KS = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array},$$

where c_i represents the column associated with individuals x_i. Notice that the vectors in KS are linear independent. Therefore, the number of rows cannot exceed the number of columns in KS. The SDB is compromised if there exists a vector of the form $(0,\ldots,0,1,0,\ldots,0)$ in KS. Unfortunately, Chin's scheme suffers from space explosion problems if the SDB is dynamically updated. In Chin's scheme, when an individual of an SDB is inserted, a new column corresponding to this individual is inserted to KS. Since the new individual has not yet been queried, all entries of the new column are zeros. On the other hand, when an individual is deleted, the corresponding column, called the *dangling column*, cannot be directly removed from the KS matrix for the protection of individual information.

If we directly delete the dangling columns to reduce the size of KS, the deletion may cause both false alarms and security disclosure. A false alarm is raised when a vector with a single "1" is found in the audit matrix but the corresponding individual is not disclosed. On the other hand, security disclosure occurs when the audit matrix does not have any vector with a single 1, but the secret of an individual is disclosed. For example in Fig. 2, the individual x_2 is deleted from SDB. If we remove the corresponding column c_2 in KS, the audit matrix will report that x_4 is disclosed and the SDB is compromised. (That is, according to the second row recorded in new KS, the vector (0, 0, 1, 0, 0, 0) contains a single 1 at the position of x_4.) In fact, x_4 is still undisclosed at this time. Thus, a false alarm has been raised.

Another example illustrated for security disclosure is shown in Fig. 3. In this example, the individual x_1 is deleted from the SDB. It seems reasonable to delete the column c_1. However, the deletion of the column will cause disclosure of secret information. Assume that a new answered query, (0,1,0,1,1,1), is invoked in KS after the deletion. The audit scheme will check KS and consider it as a

$$KS = \begin{array}{c} \begin{matrix} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \xrightarrow{\substack{x_2 \text{ is deleted from the} \\ \text{SDB and the } c_2 \text{ column} \\ \text{is removed}}} KS = \begin{array}{c} \begin{matrix} c_1 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Fig. 2. Deletion that causes a false alarm.

$$KS = \begin{array}{c} \begin{array}{ccccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \xrightarrow{\substack{x_1 \text{ is deleted from the SDB} \\ \text{and the corresponding col-} \\ \text{umn } c_1 \text{ is removed from } KS}} KS = \begin{array}{c} \begin{array}{cccccc} c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$\xrightarrow{(0,1,0,1,1,1) \text{ is invoked}} KS = \begin{array}{c} \begin{array}{cccccc} c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{l} \leftarrow r_1 \\ \\ \\ \\ \end{array}$$

Fig. 3. Deletion that causes disclosure of secret information.

redundant answerable query, which is the same as r_1. As a result, *KS* remains unchanged and the query is answered. Thus, the secret information of the deleted x_1 is disclosed.

The two examples above demonstrate that we cannot arbitrarily remove a column in a *KS* when the corresponding individual is deleted from the SDB. Therefore, the size of *KS* will only be expanded without any upper bound when the individuals of a finite-size SDB are dynamically inserted, deleted or updated. It is possible to have a large *KS* for a small SDB. Substantial memory and CPU time are wasted in handling these columns. It is not efficient to check the entire *KS* matrix for every query, when the number of the rows and the columns in *KS* is large. To cope with the problem, Chin imposes the restriction on the scheme that it can only be used in static SDBs. The restriction limits the use of the scheme. A method for the reduction of *KS* is desirable.

2.1. The enhancement

In this section, we propose an algorithm for the reduction of *KS* size. With this algorithm, Chin's scheme can be enhanced so that it can be used in a dynamic SDB. As described above, in order to guarantee the security of an SDB, all dangling columns cannot be arbitrarily deleted from the *KS*. However, it is possible to delete part of the dangling columns if the deletion will not cause the false alarm or the disclosure of any individual information. In the proposed algorithm, we assume that *KS* contains m rows and n columns, and the corresponding individuals are $x_1, x_2, \ldots, x_n$. Assume that the SDB is secure, that is, no individual has been disclosed. When k individuals are deleted from the SDB, the corresponding k columns of *KS* are marked as *dangling*.

In an audit matrix, an entry can only be either 1 or 0. A column and a row are *directly related* if their shared entry is 1. Indirectly related relation can be defined recursively. A column/row is *indirectly related* to a column/row if a directly related column/row of the former is directly/indirectly related to the latter. If a column/row is directly or indirectly related to another column/row, then they are *related.* Otherwise, they are *unrelated.* All related columns and rows form a *related set.* All elements of a related set are related to each other, and no element outside of the related set can be related to any element of the set. For example, in Fig. 4, r_1 and r_4 are directly related to c_1; r_1 are indirectly related to r_4; $\{c_1, c_3, c_5, c_6, c_7, r_1, r_3, r_4\}$ is a related set.

Definition 1. Let $c_1, c_2, \ldots, c_k, r_1, r_2, \ldots, r_l$ represent all elements of a related set in the audit matrix. If $c_1, c_2, \ldots, c_k$ are dangling columns, then

1. $c_1, c_2, \ldots, c_k$ and $r_1, r_2, \ldots, r_l$ *are garbage columns* and *rows*, respectively, and
2. the related set is called a *related garbage set.*

Since the garbage columns and rows of a related set are unrelated to other columns and rows, they can be removed without affecting the subsequent security analysis of the audit matrix. The idea is formalized as Theorem 1.

Theorem 1. *Removing a related garbage set of columns and rows from KS will not affect the subsequent security analysis of the SDB.*

Proof. Without loss of generality, assume that KS is an $m \times n$ matrix and has a related garbage set of k garbage columns and l garbage rows. Move all garbage columns to the first k columns and move all garbage rows to the first l rows. In the new matrix, by Definition 1, both the last $(n-k)$ entries of a garbage row and the last $(m-l)$ entries of a garbage column must be zeros. Hence, KS can be transformed into a *block-diagonal* matrix

$$\begin{bmatrix} A & O_1 \\ O_2 & B \end{bmatrix},$$

where $[A]$ is an $l \times k$ matrix, $[B]$ is an $(m-l) \times (n-k)$ matrix, $[O_1]$ is an $l \times (n-k)$ null-matrix, and $[O_2]$ is an $(m-l) \times k$ null-matrix. Assume that we do not remove any column or row from this matrix and give a new query

$$KS = \begin{array}{c} \begin{array}{ccccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{array} \\ \left[\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \end{array}$$

$\{c_1, c_5, c_6, c_7, r_1, r_3, r_4\}$
is a related set

Fig. 4. A related set.

which is not contained in *KS*. Since the first k individuals have been deleted from the SDB, the first k entries of the query vector must be all zeros. Clearly, we only need check this query against the rows in $[B]$ to determine its legality. $[A\ O_1]$ and $[O_2]$ will not affect the security analysis of the SDB anymore. Therefore, the submatrices $[A]$, $[O_1]$, and $[O_2]$ can be removed. □

Transforming an audit matrix to a block-diagonal matrix needs to move many columns and rows. In practice, it is not necessary to move the related columns and rows to determine whether they are removable. Instead, we can use the proposed algorithm in Fig. 5, which is based on the concept of Corollary 1.

Corollary 1. *Removing a column and all its related columns and rows from KS will not affect the subsequent security analysis of the SDB if all these columns are dangling.*

Proof. All these columns and rows are related. If all these columns are dangling, then these columns and rows form a related garbage set. Removing a related garbage set of columns and rows from *KS* will not affect the subsequent security analysis of the SDB. □

The proposed algorithm FINDING_GARBAGE in Fig. 5 is based on the concept that garbage columns and rows are related. Whenever an individual is deleted, the algorithm is able to find all the columns and rows related the new dangling column. If these columns are also dangling, then these columns and rows are all garbage and can be removed. At the end of the algorithm, *G_Set* contains the garbage columns and rows. Consequently, these columns and rows can be removed accordingly.

In order to illustrate the use of the algorithm, we will use the same example shown in Fig. 4, where *KS* is a 4×7 audit matrix. Columns c_1, c_5, c_6, c_7 are dangling columns in *KS* associated with the deleted individuals x_1, x_5, x_6, x_7. If x_3 is also deleted, then column c_3 is marked as dangling accordingly (see Fig. 6(a)). For the reduction of knowledge space, we need to find the related set which contains c_3. Because the rows r_1 and r_3 are directly related to c_3, we mark these two rows, as shown in Fig. 6(b). Then, all columns directly related to r_1 and r_3 are marked (see Fig. 6(c)), that is, columns c_1, c_5, c_6, and c_7. Because they are dangling columns, the process of finding related set is continued. Otherwise, the process should be stopped. Since r_4 is related to c_1, c_5, and c_6, the same process repeats (see Fig. 6(d)). Consequently, with the FINDING_GARBAGE algorithm, the related garbage set $G\_Set = \{c_1, c_3, c_5, c_6, c_7, r_1, r_3, r_4\}$ is found. To represent clearly, we can move the columns in *G_Set* to the first five columns and the rows in *G_set* to the first three rows, then the matrix becomes a block-diagonal matrix. As shown in Fig. 6(e), the left bottom and right top blocks of the matrix are all 0's. Fig. 6(f) shows the

```
Procedure FINDING_GARBAGE(KS: the knowledge space of the SDB;
                          c_new: the new dangling column;)
begin
   G_Set := {c_new};                  /* G_Set contains the candidates for garbage
                                         columns and rows */
   TEMP_R := {r_i | r_i is a row directly related to c_new};
                                      /* TEMP_R contains the rows to be checked */
   TEMP_C := ∅;                       /* TEMP_C contains the columns to be checked */
   ROW := {all rows in KS} - TEMP_R;  /* ROW contains rows that haven't been checked */
   For each r_i in TEMP_R             /* finding garbage columns and rows */
   begin
      G_Set := G_Set ∪ {r_i};         /* r_i is a candidate for garbage rows */
      TEMP_C := {c_i | c_i is a column directly related to r_i};
      For each c_i in TEMP_C          /* checking whether c_i is dangling and finding its
                                         directly related rows */
         If c_i is a dangling column then   /* c_i is a candidate for garbage columns */
            If c_i is not in G_Set then     /* c_i hasn't been checked */
               G_Set := G_Set ∪ {c_i};
               For each row r_j directly related to c_i
                  If r_j is in ROW then        /* r_j hasn't been checked */
                  begin
                     ROW := ROW - {r_j};
                     TEMP_R := TEMP_R ∪ {r_j};   /* r_j needs to be checked */
                  end
         else                         /* c_i is not a dangling column. That is, the related set
                                         is not a related garbage set. */
         begin
            G_Set := ∅;
            return G_Set;             /* no garbage exists */
         end
   end
   return G_Set;          /* garbage columns and garbage rows are recorded in G_Set */
end
```

Fig. 5. The algorithm for finding garbage columns and rows.

KS after the removal of garbage columns and rows. As a result, the size of the audit matrix is reduced.

Although we can use this method to reduce the memory requirement and improve the performance of Chin's Audit Expert, FINDING_GARBAGE itself also introduce overhead for the deletion of individuals from an SDB. In next section, we present a new scheme to construct the knowledge space of the SDB. It uses less space to maintain the audit matrix, and its garbage infor-

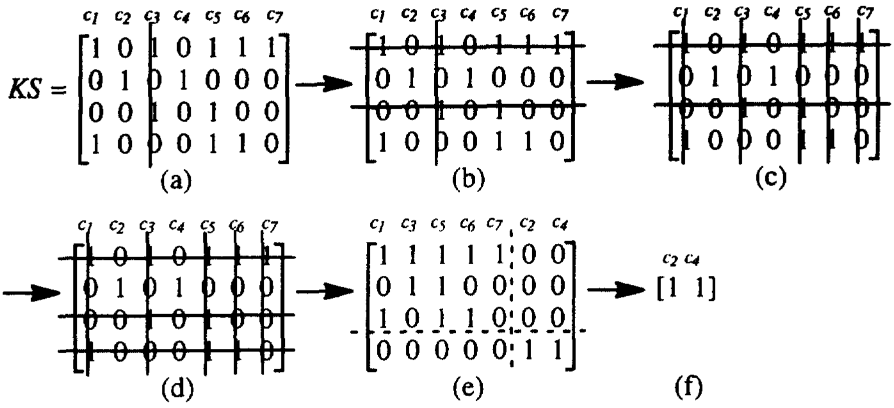

Fig. 6. Reduction of a knowledge space KS.

mation in the knowledge space can be easily found and removed without the need of invoking the FINDING_GARBAGE procedure.

3. The proposed audit scheme

In the statistical queries of SDBs, individuals with the same characteristics tend to be queried together, and individuals with different characteristics tend not to be queried together. It is possible to speed up the security analysis process by taking advantage of the characteristics. In this section, we will propose a new audit scheme which is able to efficiently distinguish illegal queries. Let G_j represent the set of individuals that were always queried together. The knowledge space in our scheme is represented as a set of vector spaces, VS_1, $VS_2, \ldots, VS_m$, and an untouched set Z of never accessed individuals. VS_i provides the knowledge regarding to the individuals that were accessed at least in a query. VS_i is represented in the matrix form where the columns are associated with the groups, the rows are linearly independent answered query vectors, and its entry indicates the status of the groups. A 1 entry indicates that all individuals of a group are accessed. A 0 entry indicates that all individuals of a group are not accessed.

There are three operations that are used to reconstruct the VS set: creating new VSs and new groups, splitting groups, and merging independent VSs into a new one. We will discuss them in the following.

3.1. Creating new VSs and new groups

If some individuals in Z are queried by a new answered query, they will form a new group. If all the queried individuals belongs to Z, a new vector space VS_i

is created which only contains a single column associated with the new group since the new group are never queried together with other groups. If only a subset of the queried individuals belongs to Z, a new column associated with the subset will be added to the VS_i whose groups are also queried in the new answered query. As an example, assume that initially the VS set are empty, and the set Z contains all individuals $x_1, x_2, \ldots, x_7$. When the first answered vector is invoked, the individuals which are accessed in this query should be grouped together and the others should remain in the Z set. If the first vector is $\overline{q_1} = (1,0,1,0,1,1,1)$, then

$$\begin{array}{c} G_1 \\ VS_1 = [1] \end{array} \quad \begin{array}{l} G_1 = \{x_1, x_3, x_5, x_6, x_7\}, \\ Z = \{x_2, x_4\}. \end{array}$$

Assume the second vector $\overline{q_2} = (0,1,0,1,0,0,0)$ is invoked. The accessed individuals in $\overline{q_1}$ and $\overline{q_2}$ are totally unrelated, and therefore a new vector space VS_2 and a new group G_2 are created. At the same time, the Z set must also be changed. Therefore, the VSs become

$$\begin{array}{c} G_1 \\ VS_1 = [1] \end{array} \qquad \begin{array}{c} G_2 \\ VS_2 = [1], \end{array}$$

where $G_1 = \{x_1, x_3, x_5, x_6, x_7\}$, $G_2 = \{x_2, x_4\}$, and $Z = \varnothing$.

3.2. Splitting groups

When individuals that have been always queried together and included in the same group are not queried together in the new answered query, the group must be split. Because there are only two possible values, 0 or 1, in an answered vector, the group must be split into two new groups: one group associated with the 1's and the other group associated with the 0's in the new answered vector. The two new columns associated with the two new groups have the same values as the old column associated with the original group. A new row will also be inserted into the new vector space VS_i. With the same example above, assume that the third answered query is $\overline{q_3} = (0,0,1,0,1,0,0)$, where only x_3 and x_5 are queried together. Thus, G_1 is split into two new groups, G_1 and G_3. The new VS_1 and groups are listed as follows:

$$VS_1 = \begin{array}{c} \begin{array}{cc} G_1 & G_3 \end{array} \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \end{array} \qquad VS_2 = \begin{array}{c} G_2 \\ [1] \end{array} \qquad \begin{array}{l} G_1 = \{x_1, x_6, x_7\}, \\ G_2 = \{x_2, x_4\}, \\ G_3 = \{x_3, x_5\}, \\ Z = \varnothing. \end{array}$$

Notice that, except the second row in the new VS_1, the two new columns associated with the new G_1 and G_3 have the same values as the old one associated with the old G_1.

3.3. Merging the VSs

When the individuals of different VSs are queried together, these VSs must be merged into a new one. Thus, a $h \times m$ VS_i and a $k \times n$ VS_j will be merged into a $(k+h) \times (n+m) VS_k$. The $k \times n VS_j$ must be expanded by padding with m all-'0' columns before merging with a $h \times m VS_i$. Similarly, the $h \times m VS_i$ also need to be expanded by padding with n all-'0' columns. Using the previous example, assume that the fourth query $\overline{q_4} = (0,1,1,1,1,0,0)$ is invoked, then VS_1 and VS_2 are merged because that G_2 and G_3 are queried together. Note that the new query vector will not be inserted into the new VS_1 because it can be computed as $r_2 + r_3$, and thus is not linearly independent of the rows of VS_1. The merging process is shown as follows.

$$\left\{\begin{array}{l} VS_1 = \begin{array}{c} G_1\ G_3 \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \end{array} \\ VS_2 = \begin{array}{c} G_2 \\ [1] \end{array} \end{array}\right. \xrightarrow{\text{expand and pad}} \left\{\begin{array}{l} VS_1{}' = \begin{array}{c} G_1\ O_3\ G2 \\ \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \\ VS_2{}' = \begin{array}{c} G_1\ G_3\ G2 \\ [0\ \ 0\ \ 1] \end{array} \end{array}\right.$$

$$\xrightarrow[\text{with } VS_2{}']{\text{merge } VS_1{}'} VS_1 = \begin{array}{c} G_1\ G_3\ G_2 \\ \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{array}$$

The VS set is reconstructed if any of the three operations described above is invoked by a new answerable query. The algorithm for reconstructing the VSs is summarized in Fig. 7. The algorithm itself is self-explanatory. Its input parameters are the new answerable query $\overline{q}$, the untouched set Z, and the set of vector spaces VSs. The output is the modified knowledge space, including the VS set and Z. The reconstruction of the VS set is scarcely needed if individuals with similar characteristics tend to be queried together, and individuals with the different characteristics tend not to be queried together in the answered queries. In this case, the time spent on reconstructing the VSs can be ignored.

With the VS set presented in our scheme, we are able to distinguish illegal queries. The checking process within a VS is similar to that within a KS in Chin's scheme. An SDB is compromised if there exists a row containing a single '1'-entry in its VSs and the corresponding group contains only a single individual. Otherwise, the SDB is still secure after answering the query.

```
Procedure Reconstruct_VS (q̄, Z, VSs)
Begin
   I := the set of individuals that are queried by q̄;
   if (I ⊆ Z) then /*all of the queried individuals are accessed in Z*/
      begin
         Combine the individuals into a new group;
         Create a new VS which only contains the new group;
         Z = Z \ I;
      end
   else if (I ∩ Z = ∅) then /*none of the queried individuals are in Z*/
      begin
         if only the subset of a group is queried then
            split the group;
         if the groups in different VSs are queried together then
            merge these VSs;
      end
   else    /*I ∩ Z ≠ ∅*/
      begin
         Create a new group for the queried individuals taken from Z;
         Add a new column, associated with the new group, to the VS;
         Pad the column with 0's.
         Z = Z \ I;
         if only the subset of a group is queried then
            split the group;
         If the groups in different VSs are queried together then
            merge these VSs;
      end
End
```

Fig. 7. Algorithm for reconstructing VSs.

4. Updates in a dynamic SDB

The insertion and deletion of individuals in our scheme is easy. In a dynamic SDB, whenever a new individual is inserted into the SDB, the individual is directly inserted into the set Z because it is never accessed before. On the other hand, when an individual is deleted from the database, it can be removed from Z without modifying VSs if it belongs to the set Z. Otherwise, we must consider two cases. Assume that the deleted individual x_i belongs to the group G_j which is contained in VS_k. In the first case, G_j contains at least one individual, excluding x_i, that has not yet been deleted. All we have to do is to mark x_i as deleted. In the second case that all individuals, except x_i, contained in G_j have been marked as deleted, G_j must be deleted and the corresponding column of G_j is considered as a *dangling column*, which is similar to what we introduced in the enhanced Audit Expert. These dangling columns cannot arbitrarily removed from *VS*s. The proposed scheme has some interesting characteristics that can help reduce the space requirement of *VS*s.

Definition 2. Two groups G_i and G_j are *related*, if

(i) G_i and G_j were queried in the same query, or

(ii) there exists another group G_k' such that (a) G_k' and G_i were queried together; (b) G_k' and G_j are related.

This is a recursive definition. In condition (ii), we can continue expanding the relationship between G_k' and G_j. The recursive expansion is stopped when condition (ii) (a) is reached. Thus, G_i, G_j, and the expanded groups G_k' are all related.

Theorem 2. *Groups are related, if and only if they are contained in the same VS.*

Proof. ($\Rightarrow$) Without loss of generality, assume groups G_i and G_j are related. We need to consider two cases with respect to the two conditions of Definition 2. The first case is that G_i and G_j are queried by the same query $\overline{q}$ and they originally belong to two different *VS*s. As a result of the query $\overline{q}$, these two *VS*s must be merged so that G_i and G_j are contained in the new *VS*. In the second case, assume that G_i was queried together with G_1', G_1' was queried together with $G'_2, \ldots$, and G_n' was queried with G_j. In the same way as above, we know G_i and G_1' are in the same *VS*, G_1' and G_2' are in the same *VS*, ..., and G_n' and G_j are in the same *VS*. Thus G_i and G_j are in the same *VS*.

($\Leftarrow$) We can prove this by a simple induction.

(i) It is trivial when the *VS* contains only one group.

(ii) We hypothesize that all n groups in the *VS* are related.

Based on the above hypothesis, we add a new group G_{n+1} to this *VS* and verify whether the $n + 1$ groups are still related or not. By the operations of the *VS*s, only the following three cases will generate new groups.

Case I: Creating new *VS*s and new groups. In this case, the new group G_{n+1} is extracted from the set Z and is first queried together with some group G_i in the *VS*, that is, G_{n+1} must be related to G_i. Therefore, G_{n+1} is related to the other n groups.

Case II: Splitting the group. By the definition of this operation, the old group G_i is split into two groups G_i' and G_{n+1}. So G_i' and G_{n+1} are related, and as G_i they are also related to the other $n - 1$ groups in the *VS*.

Case III: Merging the *VS*s. If G_{n+1} is contained in another *VS* and merged into this *VS*, by the definition of this operation, G_{n+1} must be queried together with some group G_i in the *VS*. Therefore, in the merged *VS*, G_{n+1} is related to all other groups.

Consequently, all groups in the *VS* are still related while new groups are added to the *VS*. Thus, we induce that the groups in the same *VS* must be related. □

Since the groups in different *VS*s are unrelated to each other, security analysis of a query can usually be done in a *VS* rather than all *VS*s, and the deletion of the groups in a *VS* will not affect the security analysis of the groups in the other *VS*s. In the proposed scheme, the reduction of *VS*s is simple. The extra overhead for invoking FINDING_GARBAGE to find garbage columns and rows is not needed. The idea can be described in Theorem 3.

Theorem 3. *VS_i contains a related set of garbage columns and rows, if and only if all groups in VS_i are deleted.*

Proof. (⇒) Assume that G_j is an undeleted group associated with column c_j in VS_i. By Theorem 1, we can transform the matrix of VS_i into a block-diagonal matrix

$$\begin{bmatrix} A & O1 \\ O2 & B \end{bmatrix},$$

where c_j is contained in

$$\begin{bmatrix} O1 \\ B \end{bmatrix}.$$

Obviously, VS_i can be divided into two independent vector spaces VS_i' and VS_i'' for $[A]$ and $[B]$, respectively. This result conflicts with the characteristic of the vector space that all groups of a *VS* are related.

(⇐) It is trivial. If all groups belonging to VS_i are deleted, all columns of VS_i are dangling. Since the columns and rows of VS_i are related, according to Theorem 2, they form a related set of garbage columns and rows. □

Because groups in different *VS*s are unrelated, removing the entire VS_i will not affect the security analysis of other *VS*s. Therefore, by Theorem 3, the size of the knowledge space can be reduced by removing the entire *VS* and its groups. Unlike the enhanced Audit Expert, our scheme does not need to analyze the *related* relation between columns and rows in a *VS*, that is, the FINDING_GARBAGE algorithm in Section 2 is not needed in our scheme. Instead, our scheme only need to check whether all groups in a *VS* are deleted. If all groups of a *VS* are deleted, then the *VS* and its groups can be removed. Otherwise, no columns or rows can be removed. It is clear that the proposed scheme is more efficient than the enhanced Audit Expert.

5. Complexity

In Chin's scheme, it takes no more than O(KN) steps to check the security of the SDB and determine whether a new query vector $\overline{q} \in K \times N$ KS [13], where N is equal to the sum of the total number of individuals in an SDB (n_a) and that of the deleted ones (n_d). In our scheme, all groups of individuals are partitioned, and the groups in a partition only corresponds to the columns of a VS. The knowledge space is split into v different $k_i \times n_i$ vector spaces VS_i, where $i = 1, \ldots, v, \sum_i^v n_i \leqslant N$ and $\sum_i^v k_i \leqslant K$. It takes O($k_i \cdot n_i$) steps in our audit scheme to check the security of the SDB and determine whether the new query vector $\overline{q} \in VS_i$. In general, n_i is far smaller than N and k_i is far smaller than K. Consider an average case where each group contains u individuals and each vector space contains equal number of columns (n) and rows (k). The complexity of our scheme for checking the security of the SDB becomes O(kn), which can also be represented as O $((K/v)(N/uv))$. Furthermore, in a dynamic SDB where ninety percent of the individuals ($n_d/N = 90\%$). are deleted and their corresponding columns are garbage, the complexity can be further reduced to O $((K/10v)(N/10uv))$. Comparing with the O(KN) of Chin's scheme, our scheme performs better.

6. Conclusions

In the paper, we propose a method to enhance Audit Expert. This method is able to resolve the space explosion problem caused by insertion and deletion in a dynamic SDB. Our algorithm can detect the garbage columns and rows, and thus reduce the knowledge space *KS*. Furthermore, we propose a new audit scheme which is also able to determine the security of an SDB. In this scheme, we further reduce the size of the knowledge space *KS* and thus save time and storage spent on analyzing a new query against the *KS*. The reduction process in the scheme is simpler and more efficient than that in the enhanced Audit Expert. The removal of garbage columns and rows can keep *KS* small, and speed-up the security analysis in a dynamic SDB.

Our future work is to extend the audit scheme to a large distributed SDB system. Extension of the audit scheme to a distributed environment is not easy because the merging of distributed *VS*s are difficult to perform. Further investigation is needed to explore the relationship between distributed database servers. Another interesting research related to this work is regarding to the design of a common kernel for secure SDB systems. This study will try to combine the audit scheme with database access control mechanisms. With the secure common kernel, it is easier to design a secure and efficient SDB system.

Acknowledgements

This work was supported in part by the National Science Council of Taiwan under grant number NSC-85-2213-E-009-032.

References

[1] M. Mogenstern, Controlling logical inference in multilevel database systems, in: Proceedings of The IEEE CS Symposium on Security and Privacy, April 1988, pp. 245–255.
[2] H.S. Delugach, T.H. Hinke, Wizard: A database inference analysis and detection system, IEEE Trans. Knowledge and Data Eng. 8 (1) (1996) 56–66.
[3] J.C. Wortmann, N.R. Adam, Security-control methods for statistics databases: A comparative study, ACM Computing Surveys 21 (4) (1989) 515–554.
[4] F.Y. Chin, G. Ozsoyoglu, Statistical database design, ACM Trans. Database Systems 6 (1) (1981) 113–139.
[5] D.E. Denning, A security model for the statistical database problem, in: Proceedings of The Second International Workshop on Management, 1983, pp. 1–16.
[6] D.E. Denning, J. Schlorer, Inference control for statistical databases, Computer 16 (7) (1983) 69–82.
[7] D.E. Denning, Secure statistical databases under random sample queries, ACM Trans. Database Systems 5 (3) (1980) 291–315.
[8] G. Ozsoyoglu, T.A. Su, On inference control in semantic data models for statistical databases, Journal of Computer and System Sciences 40 (3) (1990) 405–443.
[9] S.B. Reiss, The practicality of data swapping, Technical Report No. CS-48, Dept. of Computer Science, Brown Univ., Providence, RI, 1979.
[10] S.B. Reiss, Practical data-swapping: The first steps, in: Proceedings of The 1980 Symposium on Security and Privacy, IEEE Computer Society, April 1980, pp. 38–45.
[11] S.B. Reiss, Practical data-swapping: The first steps, ACM Trans. Database Systems, March (1984) 20–37.
[12] J. Schlorer, Security of statistical databases: Multidimensional transformation, ACM Trans. Database Systems 6 (1) (1981) 95–112.
[13] J.F. Traub, Y. Yemini, H. Wozniakowski, The statistical security of a statistical database, ACM Trans. Database Systems 9 (4) (1984) 672–679.
[14] D.E. Denning, Cryptography and Data Security, Addison–Wesley, Reading, MA.
[15] D. Dobkin, A.K. Jones, R.J. Lipton, Secure databases: Protection against user inference, ACM Trans. Database Systems 4 (1) (1979) 97–106.
[16] F.Y. Chin, G. Ozsoyoglu, Auditing and inference control in statistical databases, IEEE Trans. Software Eng., April (1982) 574-582.
[17] F.Y. Chin and G. Ozsoyoglu, Security in partitioned dynamic statistical databases, in: Proceedings of The IEEE COMPSAC, 1979, pp. 594–601.
[18] M. McLeish, Further result on the security of partitioned dynamic statistical databases, ACM Trans. Database Systems 14 (1) (1989) 98–113.
[19] L.H. Cox, Suppression methodology and statistical disclosure control, J. Am. Statist. Assoc. 75 (370) (1980) 377–385.