Reconstruction in Database Forensics

Oluwasola Mary Fasan, Martin Olivier

Chapter 19

RECONSTRUCTION IN DATABASE FORENSICS

Oluwasola Mary Fasan and Martin Olivier

Abstract Despite the ubiquity of databases and their importance in digital forensic investigations, the area of database forensics has received very little research attention. This paper presents an algorithm for reconstructing a database for forensic purposes. Given the current instance of a database and the log of modifying queries executed on the database over time, the database reconstruction algorithm determines the data that was present in the database at an earlier time. The algorithm employs inverse relational algebra operators along with a relational algebra log and value blocks of relations to perform database reconstruction. Illustrative examples are provided to demonstrate the application of the algorithm and its utility in forensic investigations.

Keywords: Database forensics, database reconstruction, inverse relational algebra

1. Introduction

Databases often contain information vital to forensic investigations. A typical example is a database that has been manipulated to facilitate a criminal act. Consider a case where a retail business discovers a huge loss because a large volume of a certain item was sold at a highly discounted price. The salesperson under suspicion claims that the items were sold at the official price listed in the database on the date in question. Verifying the salesperson's claim requires that the sales price of the item at that particular date be determined. However, since numerous updates of the database may have occurred since that date, it is necessary for the investigator to somehow reverse all the database transactions (queries) that affected the sales price of the item.

A promising approach to reverse queries executed on a database is to compute the inverses of the queries and proceed to systematically

reconstruct the database. However, despite extensive research on query processing, little research has focused on reverse query processing or computing query inverses. In fact, the work in this area focuses on test database generation, performance evaluation, and debugging of database applications and SQL queries [1–3, 17]. While these approaches generate good test databases, they cannot be used for forensic purposes because the databases they produce are non-deterministic in nature.

This paper defines the inverse operators of the relational algebra [4] and investigates how they can be used for the purpose of database reconstruction during forensic investigations [5]. The paper also explores the division of a query log into a set of "value blocks" – groups of queries whose evaluation does not change the information in a particular relation. The results are formalized as an algorithm that traverses the query log and value blocks, and applies inverse relational algebra operators to reconstruct database relations. Illustrative examples are provided to demonstrate the application of the database reconstruction algorithm and its utility in forensic investigations.

2. Background

This section briefly describes the relational database model and its basic operators. Also, it discusses related research in database forensics.

2.1 Relational Database Model

The relational database model developed by Codd [4] describes how data items stored in a database relate with each other. The model stores and manipulates data based on set theory and relations.

Relations. The relational model engages only one type of compound data called a relation. Given a set of domains $\{D_1, D_2, \ldots, D_n\}$ associated with the set of attributes $A = \{A_1, A_2, \ldots, A_n\}$, a relation R (also called an R-table or $R(A)$) is defined as a subset of the Cartesian product of the domains [4]. A relation may be conceived as a table whose columns are the attributes. The rows of the table are referred to as tuples, and the domains define the data types of the attributes.

Basic Operators. The relational algebra consists of basic operators used to manipulate relations and a relational assignment operator ($\leftarrow$). The basic operators transform either one or two relations into a new relation. These transformations are referred to as relation-valued expressions (*rve*). A query is defined as $T \leftarrow rve$ where T is the relation obtained when the *rve* is evaluated. The basic relational operators [4]

Table 1. Basic relational algebra operators.

Operators	**Notation**
Cartesian Product ($\times$)	$T \leftarrow R \times S$
Union ($\cup$)	$T \leftarrow R \cup S$
Intersection ($\cap$)	$T \leftarrow R \cap S$
Difference ($-$)	$T \leftarrow R - S$
Division ($/$)	$T \leftarrow R[A, B/C]S$
Join ($\bowtie$)	$T \leftarrow R[p(A, B)]S$
	$T \leftarrow R \bowtie_{p(A,B)} S$
Projection (π)	$T \leftarrow R[A_1, A_2, A_3]$
	$T \leftarrow \pi_{A_1,A_2,A_3}(R)$
Selection (σ)	$T \leftarrow R[p(A)]$
	$T \leftarrow \sigma_{p(A)}(R)$

are listed in Table 1, where R, S and T denote relations and A, B and C are attributes of relations. The notation $p(attributes)$ is a logical predicate on one or more attributes representing a condition that must be satisfied by a row before the specified operation can be performed on it.

SQL queries can be expressed in relational algebra because relational databases use the algebra for the internal representation of queries for query optimization and execution [6, 7]. The relational algebra operators can also be used independently, i.e., one or more operators can be used to express another operation. For example, $R \cap S = R - (R - S)$ and $R \bowtie_{p(A,B)} S = \sigma_{p(A,B)}(R \times S)$. This paper exploits these characteristics of the relational algebra by expressing the query log on a database as a sequence of relational operations, which we refer to as a relational algebra log (RA log).

2.2 Relational Algebra Log and Value Blocks

The relational algebra log (RA log) is a log of queries expressed as operations involving relational algebra operators instead of the traditional SQL notation. The RA log helps determine when a relation has changed. Based on the relational algebra, a relation is changed only when a new assignment operation is made into the relation. This knowledge enables the RA log to be grouped into a set of overlapping value blocks. Another motivation for the use of the RA log instead of the usual SQL log file is that relational algebra allows queries to be represented as a sequence of unary and binary operations involving relational algebra operators; this makes the log file more readable. In addition, since a typical `select` statement in an SQL log file can take several forms, the use of the RA log eliminates ambiguities that may arise in defining an

inverse for `select` statements; the ambiguity is eliminated because any `select` statement can be expressed using relational algebra operators.

A value block is defined as a set of queries within which a particular relation remains the same. Value blocks are named based on the relation that remains the same in the blocks and subscripts are used to signify the chronological sequence of blocks. A value block starts with an assignment or a rename operation and ends just before another assignment or rename is performed on the relation. For example, the value block of a relation R is denoted as V_{R_i} $(i = 1, 2, \dots)$. The relation R remains the same throughout the execution of block V_{R_1} until it is updated by the execution of the first query of block V_{R_2}. The value block of a relation can be contained in or overlap that of another relation, so that V_{R_1} and V_{S_2} can have a number of queries in common. However, two value blocks of the same relation, V_{R_1} and V_{R_2}, cannot overlap or be a subset of the other. The time stamps associated with each query are preserved in the RA log to ensure the appropriate order of the value blocks. Figure 1 in Section 4.2 shows an example of an RA log divided into value blocks.

2.3 Database Forensics

Despite the interest in digital forensics and database theory, little research has been conducted in the area of database forensics. One of the principal goals of database forensics is to revert data manipulation operations and determine the values contained in a database at an earlier time [14]. Litchfield [8–13] has published several papers on Oracle database forensics. Wright [15, 16] has also conducted research on Oracle database forensics, including the possibility of using Oracle LogMiner as a database forensic tool. However, none of these research efforts specifically address the underlying theory of database forensics or the reverting of data manipulation operations for forensic purposes.

3. Inverse Relational Algebra

The inverse operators of the relational algebra can be used to obtain the value of an attribute A of a tuple in relation R at time t. This is accomplished by computing the inverse of the most recent query performed on the current relation R_t sequentially until the desired time t is reached. The operators work on the assumption that the database schema is known and generate a result that is a partial or complete inverse of the query. Formally, we define the inverse a query Q as Q^{-1} such that:

$$Q^{-1}(Q(R_t)) = R_t^* \tag{1}$$

where $R_t^* \subseteq R_t$, i.e., R_t^* is contained in R_t. In other words, R_t^* may contain some missing tuples or missing values in some columns. A complete inverse has the property $R_t^* = R_t$; otherwise, the inverse is partial. A partial inverse is either a partial tuples inverse or a partial columns inverse depending on whether it has missing tuples or missing values in some columns, respectively. There are also cases where an inverse is both a partial tuples inverse and a partial columns inverse.

3.1 Complete Inverse Operators

Only two inverse operators of the relational algebra generate an output that is always a complete inverse. The first is the inverse rename (ρ^{-1}) operator, which only changes the name of a relation, not the data in the relation. The inverse is computed by changing the name of the renamed relation back to its previous name. If the query $A \leftarrow \rho_{A_1=B_2}(A)$ is issued to change the name of column A_1 in relation A to B_2, then the inverse of the operation is $\rho^{-1}(A) = \rho_{B_2=A_1}(A)$.

The other operator that generates a complete inverse is the inverse Cartesian product ($\times^{-1}$). Given a relation T representing the Cartesian product of two relations $R(A)$ and $S(B)$, the result of $\times^{-1}(T)$ (i.e., R and S) can be completely determined by performing a projection on their respective attributes and removing redundant tuples. That is, $\times^{-1}(T) = (R, S)$ where $R = \pi_A(T)$ and $S = \pi_B(T)$. The trivial case is when one of the operands of the Cartesian product is the empty relation. In this case, the second operand cannot be determined from the inverse operation, but this rarely occurs in practice.

3.2 Partial Inverse Operators

Most of the inverse operators are classified as partial inverses. However, regardless of this classification, there are often instances when a complete inverse can be found. The remaining inverse operators are defined below and the situations in which they yield a complete inverse are highlighted.

Inverse Projection. Given the result R of a projection operation, the inverse projection (π^{-1}) generates a partial columns inverse. The result is a relation with the expected columns (determined from the schema), but with null values in the columns not included in the projection. The columns included in the projection contain the corresponding data in R. That is, if $R \leftarrow \pi_{A_1,A_2}(S)$, then $\pi^{-1}(R) = S^*$ where S and S^* have exactly the same columns, the values of the attributes A_1 and A_2 of S and S^* are the same, and the values of the other attributes in S^* are null.

A complete inverse projection is obtained when the original projection was performed on all the columns of a relation.

Inverse Selection. The inverse selection (σ^{-1}) generates a partial tuples inverse. It is similar to the inverse projection except that it contains missing tuples instead of columns with missing data. For the result R of a selection operation $R \leftarrow \sigma_{p(A)}(S)$, the inverse selection is given by $\sigma^{-1}_{p(A)}(R) = S^*$ where $S^* = R$. The inverse selection yields a complete inverse when all the tuples in the operand of the original selection operator satisfied the selection condition.

Inverse Join. The inverse join ($\bowtie^{-1}$) is similar to the inverse Cartesian product except that the output generated may contain missing tuples depending on which tuples in the operands satisfied the condition specified in the original join operator. The output of the inverse join operator is computed by performing a projection on the columns of the expected outputs. In general, a complete inverse join is obtained if all the tuples in the original join operands satisfied the join condition, or if the join type is a full outer join. If the join type is a left outer join, then the right operand of the join operation can be completely determined and vice versa. Otherwise, a partial tuples inverse is generated.

Inverse Intersection. Given a query $T \leftarrow R \cap S$, the inverse intersection ($\cap^{-1}$) generates a partial tuples inverse containing all the tuples in T. A complete inverse is obtained when R and S are known to be the same, in which case the three relations R, S and T are equal.

Inverse Divide. Given the quotient Q and the remainder RM of a divide operation ($Q \leftarrow R/S$), the inverse divide ($/^{-1}$) generates two relations R^* and S^*. The relation R^* is readily known because all the tuples in RM are also in R^* ($RM \subseteq R^*$). A complete inverse divide is obtained only when one of the outputs is known. If R is previously known, then $S = R/T$; if S is previously known, then $R = (S \times T) \cup RM$.

Inverse Union. The inverse ($\cup^{-1}$) of a union operation $T \leftarrow R \cup S$ can only be determined if one of the outputs is known. Even so, the output generated may be a partial inverse. If relation S is known, then $R^* = T - S$; if R is known, then $S^* = T - R$. A complete inverse union is obtained only when R and S have no tuples in common. The trivial case of the inverse union is when T contains no tuples, which implies that R and S also contain no tuples.

Inverse Difference. Given a difference operation $T \leftarrow R - S$, the left operand of the operation is readily determined by the inverse difference operator $(-^{-1})$ as $R^* = T$ since $T \subseteq R$. A complete relation R can be determined only if relation S is known and all the tuples in S are also known to be in R (i.e., $S \subseteq R$) so that $R = T \cup S$. The relation S^* with partial tuples can also be determined if R is known, in which case $S^* = R - T$. If $S \subseteq R$, then a complete relation S is found from the inverse as $S = R - T$.

4. Database Reconstruction

This section describes the steps involved in reconstructing the information in a database at an earlier time using the inverse operators and RA logs divided into value blocks. Although the focus is on determining specific values in a relation at some earlier time, the approach can be applied to generate relations in a database.

4.1 Motivation

Forensic investigations often require the discovery or inference of the data contained in a database at an earlier time. Although the data currently in a database can be determined by querying the database, answers to queries posed during a forensic investigation require more than just the current instance of a database. This is because the current database content may be different from what it was at the time of interest due to modifications and updates that have occurred since the time of interest.

Earlier, we discussed the issue of proving a shop attendant's claim about the sales price of an item in a fraud investigation. Many other types of questions may be posed in forensic investigations. For example: If a relation in an organization's database was deleted by a criminal, can it be proven that a customer's record was in the deleted relation based on previously executed queries? Another example question is: Can it be proven that a patient died because the Prescribed Drugs column of the patient's record some weeks before his death was not what it was supposed to be? These questions and others call for the ability to reconstruct values in a database at an earlier time.

The next two sections provide examples of how a database can be reconstructed and present a database reconstruction algorithm.

4.2 Reconstruction

Figure 1 shows an RA log generated from the complete query log of a database by transforming queries into operations involving rela-

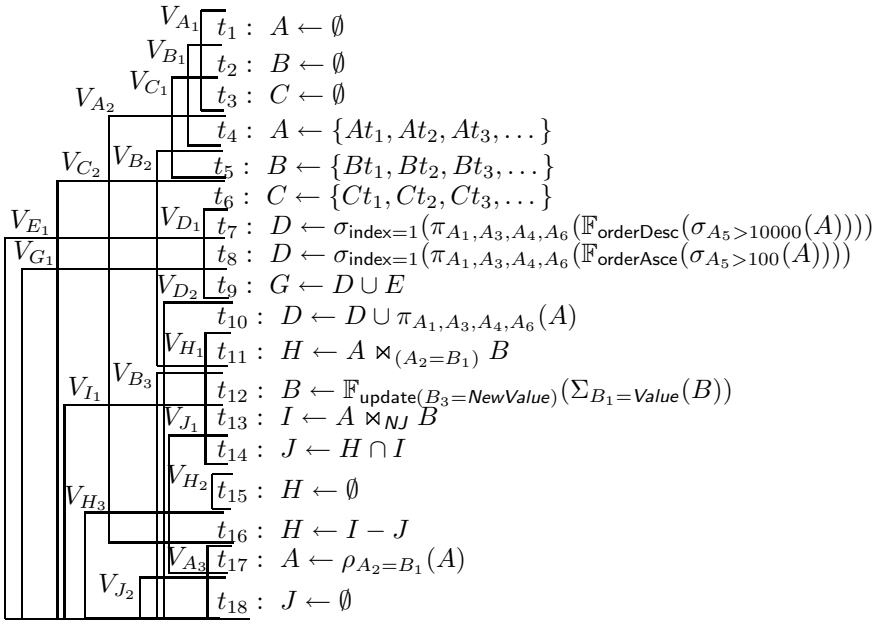

Figure 1. RA log grouped into value blocks.

tional algebra operators. The RA log is also grouped into value blocks, representing blocks of queries in which a particular relation remains unchanged. The notation V_{A_1} represents the first value block of a relation R while $t_1, t_2, \dots$ represent the time stamps at which a particular query is executed. With reference to this RA log, some of the questions that might be asked in a forensic investigation include:

- **Case 1:** Was a particular value in column D_1 of relation D at time t_8?

- **Case 2:** Is the claim that a value was in column H_2 of relation H at t_{13} true?

Case 1. In order to determine if a particular value was in column D_1 of relation D at time t_8, it is necessary to reconstruct at least the values that were in column D_1 at time t_8 and check for the value. According to Figure 1, the data in D remained unchanged between t_7 and t_9 (value block V_{D_1}). Thus, if the data in D_1 can be determined anywhere in value block V_{D_1}, then it is possible to check for the desired value. This can be accomplished by computing the inverse of the union of relations D

and E at time t_9. Because the second operand of the union (E) and its result (G) have not been changed since t_9, a partial or complete tuples inverse D (depending on whether D and E had any tuples in common) can be computed as:

$$\cup^{-1}(G) = (D^*,\ E) \text{ where } D^* = G - E \tag{2}$$

An easier alternative to the reconstruction of values in D_1 is to perform the actual query that resulted in D at time t_7. This requires the computation of the inverse of the operations that changed the data in relation A between t_7 and the current time. Since queries at $t_7, t_8, \ldots, t_{16}$ are in the same value block of A (i.e., V_{A_2}), the relation A was only modified at t_{17}. The inverse of the rename operation at t_{17} ($A \leftarrow \rho_{A_2=B_1}(A)$) is computed by simply changing the name of column B_1 to its previous name A_2 using the equation:

$$\rho^{-1}_{A_2=B_1}(A) = \rho_{B_1=A_2}(A) = A \tag{3}$$

The query at t_7 can then be performed again to generate the complete relation D.

Since t_7 is in V_{D_1}, another alternative to reconstructing the values in D is to compute the inverse of the first query in value block V_{D_2} (i.e., $D \leftarrow D \cup \pi_{A_1,A_3,A_4,A_6}(A)$) as in Equation (2):

$$\cup^{-1}(D) = (D^*,\ \pi_{A_1,A_3,A_4,A_6}(A)) \text{ where } D^* = D - \pi_{A_1,A_3,A_4,A_6}(A) \tag{4}$$

Although the relation A was updated at t_{17}, the update does not affect the projection involved in Equation (4) because the renamed column is not projected. Thus, no inverse rename is required.

It is important to note that the reconstruction of a value in a database can often be done in several ways. The different approaches may yield the same outputs or some outputs may be more complete than others. In cases where the approaches for reconstructing a value generate minimal tuples (or columns), the union of the different approaches should be computed in order to generate a relation with more data.

Case 2. To determine if a value was in column H_2 of relation H at time t_{13} (in value block V_{H_1}), it is necessary to check the value blocks V_{H_1} and V_{H_2}. It is obvious that the inverse of the first line of V_{H_2} cannot be computed because all the data in H was deleted at this point (t_{15}). From value block V_{H_1}, there are two alternatives for reconstructing column H_2 of relation H. An inverse of the intersection operation at t_{14} generates a partial tuples inverse H^* that contains all the data in

relation J. Unfortunately, the data in J was deleted at t_{18}. Thus, the only feasible option to reconstruct a value in H is to re-execute the query that resulted in H at t_{11} ($H \leftarrow A \bowtie_{(A_2=B_1)} B$). In order to do this, the inverse of the rename operation on A at t_{17} and the inverse of the update on B at t_{12} must be computed. The relation B can also be determined by computing the inverse of the natural join operation at t_{13}.

The inverse of the rename operation at t_{17} is computed according to Equation (3). To determine B using the inverse of the update at t_{12}, it is necessary to replace the values in column B_3 of B with the previous values prior to the update. Since these values are not known from the query log, the values in column B_3 are replaced with nulls. In addition, since the update contains a selection from B first, the result of the inverse of the update operation contains partial tuples of B with missing values in column B_3. Thus, the inverse of the query at t_{12} is given by:

$$\mathbb{F}^{-1}_{\text{update}(B_3=NewValue)}(\sigma^{-1}_{B_1=Value}(B)) = \mathbb{F}_{\text{update}(B_3=null)}(B^*) = B^* \quad (5)$$

The relation B with partial tuples can also be obtained by computing the inverse of the natural join operation at t_{13}. Since both relations I and A (from Equation (3)) are known, relation B can be obtained by:

$$\bowtie^{-1}_{NJ}(I) = (A,\ B^*) \text{ where } B^* = \pi_{B_1,B_2,\ldots,B_n}(I) \quad (6)$$

As mentioned earlier, the union of the two relations generated from Equations (5) and (6) can be computed to generate a more complete B^*. For example, the actual values of the nulls in column B_3 of B^* generated by Equation (5) may be determined from those of B^* generated by Equation (6). Finally, since both relations A and B (with partial tuples and probably some null values) can be determined, the query at t_{11} ($H \leftarrow A \bowtie_{(A_2=B_1)} B$) can be executed again in order to obtain H^*, which most likely contains partial tuples as well. The claim that H_2 contains a particular value can then be ascertained by checking the data in H_2.

It is possible that the value of interest is contained in the tuples missing in H after performing the reconstruction. In a forensic investigation, the conclusion about the presence or absence of a value in a relation can be strengthened by reconstructing the value in other relations in which it is expected to be present. If the value cannot be reconstructed in any other relation in which it should be present, then it is highly probable that the value is not in the relation of interest (H).

```
01: inverse(Relation D, RA Query V_{D_i}[1]) {
02: OUTPUT: Inverse of the assignment into D from query q
03: Let q = the query at V_{D_i}[1];
04: switch(q) {
05:  case (D ← ∅):
06:   T = ∅; return T;
07:  case (D ← op D):
08:   T = op^{-1}(D); return T;
09:  case (D ← A op D):
10:  case (D ← D op A): //Assume A is in V_{A_i}
11:   if (op = ∩): T = D; return T;
12:   if ((op = ∪) and (∃ V_{A_{i+1}})): T = ∅; return T;
13:   else:
14:    A ← SOLVE(A, V_{A_i}, log, S);
15:    T = op^{-1}(D)|A; return T;
16:  }
17: }
```

Figure 2. `inverse` function.

4.3 Reconstruction Algorithm

This section generalizes the reconstruction process as an algorithm for reconstructing values in a database. The algorithm, defined as function `solve`, takes as input the name of the relation D to be reconstructed, the value block V_{D_i} in which D is to be reconstructed, an RA log *log*, and a set S for storing tuples of the relation and value block (and the corresponding result) that are considered during the reconstruction. The algorithm returns the reconstructed relation D in the specified value block.

The `solve` function makes use of the `inverse` function (Figure 2), which takes as input the name of the relation to be reconstructed (D) together with a query in the first line of a value block of D (i.e., $V_{D_i}[1]$) and computes the inverse of the query in order to determine D in its previous value block ($V_{D_{i-1}}$). Calls to the `inverse` function occur when a value block $V_{D_{i-1}}$ exists and its output depends on the operation performed by the query. The notation $\text{op}^{-1}(D)|A$ (line 15) denotes the inverse of an operation with two operands of which one operand (i.e., A) is known.

The `solve` function (Figure 3) begins by generating a set Q of queries involving the relation D in the value block V_{D_i} in which it is to be reconstructed. Each element of Q represents a different approach in which D can be reconstructed. The algorithm initializes a set R in which all possible reconstructions of D are stored. The parameter S of the `solve` function is empty the first time the function is called and it stores tuples of the relation and value block (with the corresponding result) that have already been considered in the reconstruction process

```
solve(Relation D, Value Block V_{D_i}, RA Log log, Set S)
OUTPUT: Reconstructed relation D in value block V_{D_i} (RD)
01: Let Q = Set of queries involving relation D in value block V_{D_i};
02: Let R = Set to reconstructed D from different approaches;
03: If (D, V_{D_i}, RD) ∈ S: return RD
04: else:
05: S = S ∪ (D, V_{D_i}, RD); //RD is initialized as an empty relation
06: for each element e in Q:
07:  switch(e) {
08:  case (D ← op D):
09:   if (∄ V_{D_{i+1}}): return D;
10:   else:
11:   D ← SOLVE(D, V_{D_{i+1}}, log, S); T ← INVERSE(D, V_{D_{i+1}}[1]);
12:   Insert T into R
13:   OR
14:   D ← SOLVE(D, V_{D_{i-1}}, log, S); T ← op D;
15:   Insert T into R
16:  case (D ← op A): //Assume  is in V_{A_i}
17:   if (∄ V_{D_{i+1}}): return D;
18:   else:
19:   if (∄ V_{A_{i+1}}):
20:    D ← op A; return D;
21:    else:
22:    A ← SOLVE(A, V_{A_{i+1}}, log, S); A ← INVERSE(A, V_{A_{i+1}}[1]);
23:    D ← op A; return D;
24:  case (D ← A op D):
25:  case (D ← D op A): //Assume A is in V_{A_i}
26:   if (∄ V_{D_{i+1}}): return D;
27:   else:
28:   D ← SOLVE(D, V_{D_{i+1}}, log, S); T ← INVERSE(D, V_{D_{i+1}}[1]);
29:   Insert T into R;
30:   if (∄ V_{A_{i+1}}):
31:    D ← SOLVE(D, V_{D_{i-1}}, log, S);
32:    T ← A op D or (D op A); //depending on case
33:    Insert T into R;
34:   else:
35:    D ← SOLVE(D, V_{D_{i-1}}, log, S);
36:    A ← SOLVE(A, V_{A_i}, log, S);
37:    T ← A op D or (D op A) //depending on case
38:    Insert T into R;
39:    OR
40:    D ← SOLVE(D, V_{D_{i-1}}, log, S);
41:    A ← SOLVE(A, V_{A_{i+1}}, log, S); A ← INVERSE(A, V_{A_{i+1}}[1]);
42:    T ← A op D or (D op A); //depending on case
43:    Insert T into R;
```

Figure 3. `solve` function.

in order to avoid loops in the recursive calls to `solve`. If an attempt to reconstruct D in value block V_{D_i} has been made earlier (line 3), the `solve` function returns the associated reconstructed relation RD. Otherwise, the relation and value block parameters of the function are stored as a tuple in S with an associated reconstructed relation that is initially empty. The algorithm then considers the possible combinations

```
44:  case (G ← op D): //Assume G is in V_{G_i}
45:   if (∄ V_{D_{i+1}}): return D;
46:   else:
47:   if (∄ V_{G_{i+1}}):
48:    T ← op^{-1}(G); Insert T into R;
49:   else:
50:    D ← SOLVE(D, V_{D_{i+1}}, log, S); T ← INVERSE(D, V_{D_{i+1}}[1]);
51:    Insert T into R;
52:    OR
53:    G ← SOLVE(G, V_{G_{i+1}}, log, S); G ← INVERSE(G, V_{G_{i+1}}[1]);
54:    T ← op^{-1}(G); Insert T into R;
55:  case (G ← D op A):
56:  case (G ← A op D): //Assume G and A are in V_{G_i} and V_{A_i} respectively
57:   if (∄ V_{D_{i+1}}): return D;
58:   else:
59:   if (∄ V_{G_{i+1}}):
60:    if (op = ∩):
61:    Insert G into R;
62:    if (op ≠ ∪):
63:    T ← op^{-1}(G)[1]; //D is at index 1 in the output of op^{-1}(G)
64:    Insert T into R;
65:    if (∄ V_{A_{i+1}}):
66:    T ← op^{-1}(G)|A; Insert T into R;
67:    else:
68:    A ← SOLVE(A, V_{A_{i+1}}, log, S); A ← INVERSE(A, V_{A_{i+1}}[1]);
69:    T ← op^{-1}(G)|A; Insert T into R;
70:   else:
71:    if (∄ V_{A_{i+1}}):
72:    G ← SOLVE(G, V_{G_{i+1}}, log, S); G ← INVERSE(G, V_{G_{i+1}}[1]);
73:    T ← op^{-1}(G)|A; Insert T into R;
74:    else:
75:    G ← SOLVE(G, V_{G_{i+1}}, log, S); G ← INVERSE(G, V_{G_{i+1}}[1]);
76:    if (op = ∩): Insert G into R;
77:    else:
78:     A ← SOLVE(A, V_{A_{i+1}}, log, S); A ← INVERSE(A, V_{A_{i+1}}[1]);
79:     T ← op^{-1}(G)|A; Insert T into R;
80:  }
81: RD ← union of all the relations in R; //Reconstructed D
82: return RD;
```

Figure 3. `solve` function (continued).

of D in a query and outlines the steps to be followed in reconstructing D from the different approaches listed in Q. After all the queries in Q have been considered, the union of all the possible reconstructions is stored as the reconstructed relation RD and this result is returned.

It is important to note that some of the reconstructed relations in R may contain more information than others and might be adequate for the purpose of the reconstruction process. In situations where reconstruction is performed to determine or check a particular value or claim in a relation, the reconstruction algorithm can be improved by searching each of the possible reconstructed relations before inserting it in the set

R. The `solve` algorithm is terminated when the value of interest or desired information has been determined.

In a few cases, it is possible that the reconstruction process results in an empty relation. This could occur if all the tuples in a relation were deleted before the relation was used in any way. Nevertheless, the algorithm is useful for reconstructing values in relations for forensic purposes. We conjecture that it can be proved that the tuples generated in a reconstructed relation are indeed in the relation and that the algorithm does not result in an infinite loop. Developing these proofs will be the subject of our future research.

5. Conclusions

The database reconstruction algorithm presented in this paper enables forensic investigators to determine whether or not data of interest was present in a database at an earlier time despite the fact that several database modifications may have been performed since that time. The algorithm, which is based on the formal model of relational databases, employs inverse relational algebra operators along with a relational algebra log and value blocks of relations to determine whether or not data of interest was present in a database at an earlier time.

Our future work will investigate the conditions under which the reconstruction algorithm may not yield adequate results. Also, it will attempt to enhance the algorithm to ensure that the reconstructed relations preserve the integrity constraints imposed on the original relations.

Acknowledgements

This research was supported by the Organization for Women in Science for the Developing World (OWSD).

References

[1] C. Binnig, D. Kossmann and E. Lo, Reverse query processing, *Proceedings of the Twenty-Third IEEE International Conference on Data Engineering*, pp. 506–515, 2007.

[2] C. Binnig, D. Kossmann and E. Lo, Towards automatic test database generation, *IEEE Data Engineering Bulletin*, vol. 31(1), pp. 28–35, 2008.

[3] N. Bruno and S. Chaudhuri, Flexible database generators, *Proceedings of the Thirty-First International Conference on Very Large Databases*, pp. 1097–1107, 2005.

[4] E. Codd, *The Relational Model for Database Management: Version 2*, Addison-Wesley, Reading, Massachusetts, 1990.

[5] F. Cohen, *Digital Forensic Evidence Examination*, ASP Press, Livermore, California, 2010.

[6] G. Graefe, Query evaluation techniques for large databases, *ACM Computing Surveys*, vol. 25(2), pp. 73–170, 1993.

[7] L. Haas, J. Freytag, G. Lohman and H. Pirahesh, Extensible query processing in Starburst, *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pp. 377–388, 1989.

[8] D. Litchfield, Oracle Forensics Part 1: Dissecting the Redo Logs, NGSSoftware, Manchester, United Kingdom, 2007.

[9] D. Litchfield, Oracle Forensics Part 2: Locating Dropped Objects, NGSSoftware, Manchester, United Kingdom, 2007.

[10] D. Litchfield, Oracle Forensics Part 3: Isolating Evidence of Attacks Against the Authentication Mechanism, NGSSoftware, Manchester, United Kingdom, 2007.

[11] D. Litchfield, Oracle Forensics Part 4: Live Response, NGSSoftware, Manchester, United Kingdom, 2007.

[12] D. Litchfield, Oracle Forensics Part 5: Finding Evidence of Data Theft in the Absence of Auditing, NGSSoftware, Manchester, United Kingdom, 2007.

[13] D. Litchfield, Oracle Forensics Part 6: Examining Undo Segments, Flashback and the Oracle Recycle Bin, NGSSoftware, Manchester, United Kingdom, 2007.

[14] M. Olivier, On metadata context in database forensics, *Digital Investigation*, vol. 5(3-4), pp. 115–123, 2009.

[15] P. Wright, Oracle Database Forensics using LogMiner, NGSSoftware, Manchester, United Kingdom, 2005.

[16] P. Wright, *Oracle Forensics: Oracle Security Best Practices*, Rampant Techpress, Kittrell, North Carolina, 2010.

[17] S. Xu, S. Wang and M. Hong, Application of SQL RAT translation, *International Journal of Intelligent Systems and Applications*, vol. 3(5), pp. 48–55, 2011.