On the Completeness of Reconstructed Data for Database Forensics

Oluwasola Mary Adedayo and Martin S. Olivier

ICSA, Department of Computer Science,
University of Pretoria, South Africa
{mfasan,molivier}@cs.up.ac.za

Abstract. Databases are often used to store critical and sensitive information in various organizations and this has led to an increase in the rate at which databases are exploited in computer crimes. Even though various investigations involving databases have been explored, very little amount of research has been done on database forensics. This paper briefly describes a database reconstruction algorithm presented in an earlier work and shows the limitation that can be encountered when the algorithm has to deal with partially reconstructed relations or the deletion of tuples in a relation. Since reconstructed data can often be used as the evidence to support or refute claims about the data in a database, the inability to reconstruct necessary data may imply the absence of evidence. However, according to an axiom from forensic science, this does not mean an evidence of absence. As such, this paper presents two different techniques that can be used in reconstructing more tuples in a relation and provide corroborating evidence to claims about the data on a database. A typical example is used to describe the limitation of the database reconstruction algorithm and how the limitation can be overcomed by using the techniques described in the paper.

Keywords: Digital forensics, Database forensics, Database reconstruction algorithm, Digital evidence, Forensic science.

1 Introduction

The use of databases in todays commercial systems cannot be over-emphasized as databases have become a core component of many computing systems and are often used to store critical and sensitive information to an organization or her clients. Unfortunately, the increased usage of databases in storing volumes of information together with the increased relevance of the data on many databases in solving various crimes have led to an increase in the number of attacks directed towards databases and interests investigating databases for artifacts that may assist in solving various different crimes.

Database forensics is an emerging branch of digital forensics [16,1] that deals with the identification, preservation, analysis and presentation of evidence from databases [7]. Even though digital forensics has grown over the last decade from a

M. Rogers and K.C. Seigfried-Spellar (Eds.): ICDF2C 2012, LNICST 114, pp. 220–238, 2013.

relatively obscure trade-craft to an important part of many investigations [8], the same cannot be said of database forensics, despite the importance of databases. Although a large amount of research has been done on digital forensics, database theory and database security, very little has been done on database forensics [15] even though investigations involving databases have been explored in theory and in practice. Similar to other branches of digital forensics, database forensics helps in determining the root cause of an attack and finds out what was done. An important aspect of database forensics deals with the ability to revert data manipulation operations and determine values contained in a database at an earlier time.

Although various data restoration techniques such as rollback and incremental backups have been explored over the years, these techniques are sometimes inadequate for database forensics. For example, a rollback operation can only be used provided that the transaction has not been committed and the use of incremental backups is dependent on the availability of viable backups from which data can be restored. Database forensics requires the ability to revert data manipulation operations even when a transaction has been long committed or when there are no viable backups.

In our earlier work [6], we presented an algorithm that can be used for reconstructing the information in a database at an earlier time of interest. The algorithm makes use of the inverse functions of the relational algebra [4] and incorporates the notion of value blocks (a group of queries whose evaluation does not change the information in a particular relation). The inverse of a query is found by taking the database schema and the log of modifying queries performed on the database into consideration. In another work [5], we prove that the relation generated from the algorithm is always correct. That is, even though the reconstructed relation may be incomplete if compared with the original relation, it is at least a subset of the original relation.

The generation of incomplete relations or inability to reconstruct values of interest in a relation when using the database reconstruction algorithm [6] stems from the fact that the inverse generated from some of the inverse operators of the relational algebra may be missing one or more tuples or values in a column of the original relations. It also implies that the evidence needed from a database during an investigation may not be found. However, this does not imply that such evidence does not exist. The objective of this paper is to discuss the limitation of the database reconstruction algorithm and describe some of the techniques that can be applied in conjunction with the algorithm in order to generate more complete relations or tuples of a relations as well as provide corroborating evidence regarding claims about the information on a database at an earlier time. The paper describes a typical application of the reconstruction algorithm that reflects its limitation. It also discusses two different techniques of reconstructing more information from a database using the reconstruction algorithm.

2 Background and Notation

This section gives a brief background on database forensics and introduces the relational model of database management systems (DBMS) and its operators. It also describes the notation used in the rest of the paper.

2.1 Database Forensics

As earlier mentioned, database forensics often requires the determination of the information in a database at an earlier time. Although the information in a database at any instance can be determined by querying the database, much more effort is required in order to determine the information contained at an earlier time since various modifications might have occurred. Some of the little work that has been done in database forensics include the series of papers by Litchfield [9,10,11,12,13,14] all of which focus on Oracle forensics. Wright [18] published a book that also explains Oracle forensics and investigates the possibility of using Oracle LogMiner as a forensics tool [17]. Another book by Fowler [7] focuses on SQL server database forensics and discusses the effect of rootkits on data collection and analysis during the forensics investigation of an SQL server database. None of these works describes the process of reconstructing the information in a database at an earlier time.

The ability to reconstruct the information on a database at earlier time is an important aspect of database forensics. An illustration of this fact, which requires forensics investigation is a situation where a sales representative claims to have sold a large quantity of a certain good at the selling price on the database at a particular date even though the price presents a huge loss to the organization. Verifying the representative's claim requires that the selling price of the good at that particular date can be determined even though several modifications/updates might have been performed on the database which might have affected the price of the good since the date of interest.

In our earlier work, we present an algorithm [6] that can be used to determine the information contained in a database at an earlier time. Although our focus in this paper is to discuss the completeness of the result generated from the algorithm and how this can be improved, it is important to introduce the notion of inverse relational algebra and value blocks which are a major component of the algorithm. The relational algebra [4] is employed since it represents a fundamental aspect of databases and gives a formal description of how the information stored in a database relate with each other.

2.2 Inverse Relational Algebra

The relational model for DBMS was developed by Codd [4] and works on the relational theory of mathematics. The model is composed of one type of compound data known as a relation. Given a set of domains, $D = D_1, D_2, \ldots, D_n$ over which attributes $A = A_1, A_2, \ldots, A_n$ are defined respectively, a relation R (also called an *R-table* or $R(A)$) is a subset of the Cartesian product of the

Table 1. Inverse Operators of the Relational Algebra

Operators	**Query**	**Inverse Operators**
Rename (ρ)	$R \leftarrow \rho_{A_i=B_j}(R)$	$\rho^{-1}(R) = \rho_{B_j=A_i}(R)$
Cartesian product ($\times$)	$T \leftarrow R(A) \times S(B)$	$\times^{-1}(T) = (R, S)$ where $R = \pi_A(T)$ and $S = \pi_B(T)$
Union ($\cup$)	$T \leftarrow R \cup S$	$\cup^{-1}(T) = (R^*, S)$ where $R^* = T - S$ provided that S is known and vice versa
Intersection ($\cap$)	$T \leftarrow R \cap S$	$\cap^{-1}(T) = (R^*, S^*)$ where $R^* = S^* = T$
Difference ($-$)	$T \leftarrow R - S$	$-^{-1}(T) = R^* = T$. If R is known, $S^* = R - T$.
Division ($/$)	$T \leftarrow R/S$	$/^{-1}(T) = (R^*, S^*)$ where $R^* = RM$ and RM is the remainder of the division
Join ($\bowtie$)	$T \leftarrow R(A) \bowtie_{p(A,B)} S(B)$ $T \leftarrow R \bowtie_{p(A,B)} S$	$\bowtie^{-1}(T) = (R^*, S^*)$ where $R^* = \pi_A(T)$ and $S^* = \pi_B(T)$
Projection (π)	$T \leftarrow \pi_{A_1,A_2,A_3}(R)$ $T \leftarrow R[A_1, A_2, A_3]$	$\pi^{-1}(T) = S^* = T$
Selection (σ)	$T \leftarrow \sigma_{p(A)}(R)$ $T \leftarrow R[p(A)]$	$\sigma^{-1}_{p(A)}(T) = S^* = T$

domains. A relation can be conceived as a table where the columns are the attributes, the rows are referred to as tuples and the domains define the data types of the attributes.

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. Such transformations are known as relation-valued expressions (*rve*). A query is defined in the form $T \leftarrow rve$, where T is the name of the relation obtained when the rve is evaluated. The basic operators as defined by Codd [4] and the corresponding notation often used are shown in the second column of table 1, where R, S, and T are relations and A, B and C are attributes of these 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.

The inverse operators of the relational algebra work on the assumption that the database schema is known. The aim of the inverse operators is to find the value of one or more attributes of a relation at a specific time t by finding the inverse of the most recent query performed on the current relation R_t sequentially until the desired time t_i is reached. The output generated by an inverse operator may either be partial or complete when compared to the original relation. That is, the inverse of a query Q can be defined as Q^{-1} such that,

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

where R_t^* is a subset of R_t. That is, some tuples or values that should be in some columns of R_t^* may be missing[1]. In cases where $R_t^* = R_t$, we refer to the inverse found as a complete inverse. Otherwise, we refer to the inverse found as a partial inverse. A partial inverse can be a partial tuples inverse and/or a partial columns inverse depending on whether some of the tuples or values in some columns of the original relation are missing, respectively. However, regardless of the classification of the inverse operators, there are often instances where a complete inverse can be found with some of the operators grouped as partial inverse. A summary of the inverse operators of the relational algebra and how inverses are computed is given in table 1. More details about relational algebra, the inverse operators and instances where a complete inverse can be generated from the inverse operators classified as partial inverses can be found in earlier works [6,5].

2.3 Relational Algebra Log and Value Blocks

A relational algebra log (RA log) is a log of queries expressed as relational algebra operations instead of the traditional SQL notation. The use of the RA log allows us to easily determine when a relation has been modified. Using the relational algebra notation, a relation is changed only when a new assignment operation is made into the relation. This knowledge allows us to group the RA log into a set of overlapping *value blocks*. Another advantage of using 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. Thus, the log file is more readable. For example, a typical select statement in a SQL log file can take several forms; however, the use of the RA log eliminates ambiguities that may arise in defining an inverse for select statements since any select statement can be expressed with relational algebra operators.

A value block is defined as a set of queries within which a particular relation remains unchanged. Value blocks are named based on the relation that remains the same in the block and subscripts are used to signify which value block occurs first. A value block starts with an assignment operation into the relation and ends just before another assignment operation into the relation is encountered. For example, the value block of a relation R is denoted as V_{R_i} where $i = 1, 2, 3, \ldots$. 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}. Thus, 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 [6]. The time stamps usually associated with the traditional query log is preserved in the RA log in order to group the value blocks into appropriate sequences. An example of a RA log generated from a traditional log file and divided into value blocks is shown in figure 1. Subsequent examples in the paper will refer to this RA log.

[1] The mapping generated by queries are not usually a bijection. However, this does not mean that some inverses cannot be found.

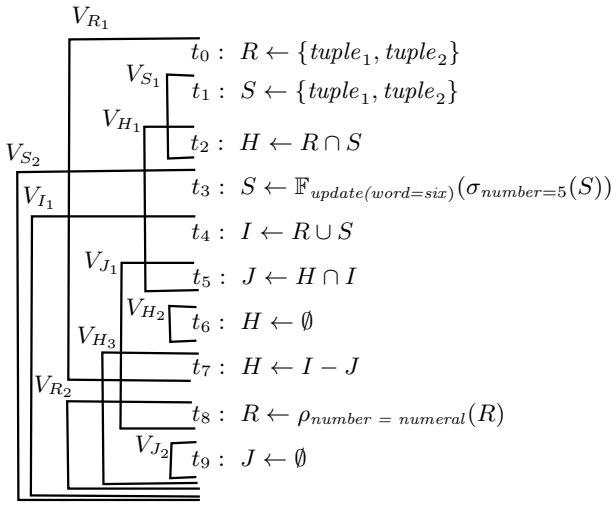

Fig. 1. A Relational Algebra Log Grouped into Value Blocks

2.4 Database Reconstruction Algorithm

The database reconstruction algorithm employs the notion of RA logs and value blocks as well as the inverse operators of the relational algebra. In this section, we give a brief description of the reconstruction algorithm necessary to understand the content of this paper, a more detailed explanation of the algorithm can be found in [6].

```
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: }
```

Fig. 2. The INVERSE(Relation D, RA Query $V_{D_i}[1]$) function

Although the algorithm is aimed at reconstructing specific values in a relation on a database at some earlier time, it can also be applied to generate tables in a database. The algorithm assumes that the query log exists, and contains the complete set of modifying queries that have been performed on the database from at least the particular time of interest (or earlier) to the present time.

The reconstruction algorithm consists mainly of two functions: the `inverse` and the `solve` functions. The `solve` function makes use of the `inverse` function (shown in figure 2) which takes as input the name of the relation to be reconstructed (D) together with a query, specifically the first line of a value block of D (denoted as $V_{D_i}[1]$) and finds the inverse of the query in order to determine

```
SOLVE(Relation D, Value Block V_{D_i}, RA Log log, Set S)
OUTPUT: Reconstructed relation D in value block V_{D_i} (RD)
Let Q = Set of queries involving relation D in value block V_{D_i};
Let R = Set to reconstructed D from different approaches;
If (D, V_{D_i}, RD) ∈ S: return RD
else:
 S = S ∪ (D, V_{D_i}, RD); //RD is initialized as an empty relation
 for each element e in Q:
  switch(e) {
   case (D ← op D):
    if (∄ V_{D_{i+1}}): return D;
    else:
     D ← SOLVE(D, V_{D_{i+1}}, log, S); T ← INVERSE(D, V_{D_{i+1}}[1]);
     Insert T into R
     OR
     D ← SOLVE(D, V_{D_{i-1}}, log, S); T ← op D;
     Insert T into R
   case (D ← op A): //Assume is in V_{A_i}
    if (∄ V_{D_{i+1}}): return D;
    else:
     if (∄ V_{A_{i+1}}):
      D ← op A; return D;
      else:
       A ← SOLVE(A, V_{A_{i+1}}, log, S); A ← INVERSE(A, V_{A_{i+1}}[1]);
       D ← op A; return D;
   case (D ← A op D):
   case (D ← D op A): //Assume A is in V_{A_i}
    if (∄ V_{D_{i+1}}): return D;
    else:
     D ← SOLVE(D, V_{D_{i+1}}, log, S); T ← INVERSE(D, V_{D_{i+1}}[1]);
     Insert T into R;
     if (∄ V_{A_{i+1}}):
      D ← SOLVE(D, V_{D_{i-1}}, log, S);
      T ← A op D or (D op A); //depending on case
      Insert T into R;
     else:
      D ← SOLVE(D, V_{D_{i-1}}, log, S);
      A ← SOLVE(A, V_{A_i}, log, S);
      T ← A op D or (D op A) //depending on case
      Insert T into R;
      OR
      D ← SOLVE(D, V_{D_{i-1}}, log, S);
      A ← SOLVE(A, V_{A_{i+1}}, log, S); A ← INVERSE(A, V_{A_{i+1}}[1]);
      T ← A op D or (D op A); //depending on case
      Insert T into R;
```

Fig. 3. The SOLVE function

```
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;
```

Fig. 3. (*Continued.*)

D in its previous value block ($V_{D_{i-1}}$). The `solve` function takes as input the name of the relation to be reconstructed D, its value block in which it is to be reconstructed V_{D_i}, a relational algebra log *log*, and a set S which is used to store tuples of relation and value block (and the corresponding result) which have been considered during the reconstruction. The reconstructed relation D in the specified value block is returned from the algorithm. A listing of the `solve` function is shown in figure 3.

3 Limitation of the Reconstruction Algorithm

In section 2.2, we mentioned that the output generated from the inverse operators of the relational algebra may either be complete or partial when compared with the original relation. Even though every reconstructed relation is always correct, that is, it is at least a subset of the original relation [5], partial inverses

sometimes affects the amount of information that can be reconstructed using the database reconstruction algorithm. Since the algorithm depends on the inverse operators of the relational algebra, the generation of partial inverses from some of these operators sometimes result in the generation of significantly incomplete (or empty) relation when using the algorithm.

This section gives a brief description of how the values in a relation at an earlier time can be reconstructed using the database reconstruction algorithm and reveals the limitation of the algorithm when dealing with inverse operators that generate partial inverses. In figure 1, we show a typical example of a RA log generated from a traditional SQL log file. For simplicity and further explanations in subsequent sections of the paper, we assume that the content of each relation after executing the queries in the RA log at each timestamp are as computed in figure 4.

Our aim is to reconstruct the tuples in relation H at time t_3 since several modifications of the relation has occurred. The relations R ans S both have attributes `word` and `number` and contains two tuples each, which are assigned at time t_0 and t_1, respectively.

The relation H at time t_3 in figure 1 is the same as H at any time between t_2 and t_5 inclusively, since the queries executed between these times are in the same value block of H, that is, V_{H_1}. Using the reconstruction algorithm, there are three different ways in which the relation H at t_3 can be reconstructed:

1. By reversing the query performed on the first line of value block V_{H_2} at time t_6. Unfortunately, this cannot be achieved since the relation H was dropped (or all its contents were deleted) at this point.
2. Another alternative is to find the inverse of the intersection operation performed at time t_5 in order to obtain a partial reconstruction of H. However, since the relation J was also subsequently deleted at time t_9, this inverse cannot be found since the inverse of the intersection operation is given as $\cap^{-1}(J) = (H^*, I^*)$ where $H^* = I^* = J$.
3. The last possible way of reconstructing H at t_3 is to re-execute the query at time t_2. This requires that the relations R an S at time t_2 are known (or reconstructed first). Since relations R and S are in value blocks V_{R_1} and V_{S_1}, respectively at time t_2 and they both have subsequent value blocks, the relations must first be reconstructed in their respective value blocks at t_2 before the query at t_2 can be re-executed. The relation R at time t_2 (or in V_{R_1}) can be found by finding the inverse of the rename operation performed at time t_8, which is given as $\rho^{-1}(R) = \rho_{numeral\ =\ number}(R)$. Since an inverse rename operation always generates a complete relation, the relation R at t_2 is successfully reconstructed from the inverse rename operation. The relation S at time t_2 (or in V_{S_1}) can be found by getting the inverse of the update operation performed on S at t_3. The inverse of the update can be represented as:

$$\mathbb{F}^{-1}_{update(word=six)}(\sigma^{-1}_{number=5}(S)) = \mathbb{F}_{update(word=null)}(\sigma_{number=5}(S)).$$

This generates the partial relation S^* shown in table 2. Since relations R and S at t_3 are now known, the query at t_3 ($H \leftarrow R \cap S$)can be re-executed

$t_0: R \leftarrow \{tuple_1, tuple_2\}$

R	Word	Number
	five	5
	six	6

$t_1: S \leftarrow \{tuple_1, tuple_2\}$

S	Word	Number
	four	4
	five	5

$t_2: H \leftarrow R \cap S$

H	Word	Number
	five	5

$t_3: S \leftarrow \mathbb{F}_{update(word=seven)}(\sigma_{number=5}(S))$

S	Word	Number
	four	4
	seven	5

$t_4: I \leftarrow R \cup S$

I	Word	Number
	four	4
	five	5
	six	6
	seven	5

$t_5: J \leftarrow H \cap I$

J	Word	Number
	five	5

$t_6: H \leftarrow \emptyset$

H	Word	Number

$t_7: H \leftarrow I - J$

H	Word	Number
	four	4
	six	6
	seven	5

$t_8: R \leftarrow \rho_{number\ =\ numeral}(R)$

R	Word	Numeral
	four	4
	five	5
	six	6

$t_9: J \leftarrow \emptyset$

J	Word	Number

Fig. 4. Original Relations Obtained from Queries Executed

Table 2. Reconstructed relation S^*

S^*	Word	Number
	four	4
	null	5

in order to reconstruct the tuples in H at time t_3. However, because the reconstructed relation S^* is a partial inverse, the tuple in H at t_3 cannot be reconstructed from the re-execution of this query and an empty relation H^* with the same attributes as the original relation H is generated (table 3).

Table 3. An Empty Reconstructed relation H^*

H^*	Word	Number

This example reflects a major limitation of the database reconstruction algorithm that can be encountered when dealing with partial inverses. In the rest of this paper, we discuss some of the techniques that can be applied in conjunction with the reconstruction algorithm in order to generate more complete reconstructed relations and/or find corroborating evidence regarding the data on a database during database forensics.

4 Absence of Evidence

The database reconstruction algorithm [6] can be used to find the information in the database at an earlier time. In this same way as data collected in different branches of digital forensics can be the required evidence or assist in carrying out an investigation, reconstructed relations may often be used as the evidence[2], provide support for other evidence during an investigation, or to provide more information about an investigation. Unfortunately, the fact that a reconstructed relation may be incomplete implies that some evidence may not be found. In situations where the evidence to refute or support a claim cannot be found in a reconstructed relation, it is important to remember an axiom from Forensics Science that says that, "absence of evidence is not evidence of absence" [2]. For example, if no evidence (or reconstructed data) could be found to support the sales representative's claim about the price of good sold on a particular date, it does not mean that the representative is lying. If no evidence could be found on a computer to determine whether or not it accessed a particular web page, it does not mean that the computer was used to access the site. It is important to base all assertions on solid supporting evidence and not on an absence of evidence [2]. Thus, it is necessary for an investigator to find corroborating evidence that clearly demonstrates the falsity or truth of a claim about the information on a database at an earlier time.

In this paper, we present two techniques of finding corroborating evidence about claims on the data in a database. The first technique works based on Locard's exchange principle that contact between two items will always result

[2] Evidence may or may not be admissible in a court of law.

in an exchange [3]. That is, there will always be some trace evidence with every interaction even though it may not be easily detected. According to Casey [2], this principle applies in both the physical and digital realms and can provide links between them. For example, in a case involving email harassment, the act of sending messages over a web-based email service can leave traces such as files and links on the sender's hard disk and/or web browser as well as some date-time related information. Other information may also possibly be obtained from the email service provider [2]. Although this principle may not be true for all systems in general, it is true for systems that keep record of their actions or activities. In database reconstruction, the items involved in an interaction are the relations on a database while the interaction is the operation performed on such relations. This technique works on the fact that if there is a claim that some data was in a relation at an earlier time, then there should be some trace evidence that can be gathered from the interaction of the relation with other relations on the database.

The second technique for finding corroborating evidence involves the reconstruction of more complete relations through the iteration of the database reconstruction algorithm presented in [6] and inferences from reconstructed relations. The technique works on the fact that the data created when an investigator reenacts the events in a crime should resemble the original evidence collected as close as possible. That is, given a reconstructed relation, if an investigator re-executes the queries performed on the database (using the log record), the recreated database instance should be the same as the current instance of the database. If this is not the case, then it implies that some information is missing in the reconstructed relation since we have already proved that any data in a reconstructed relation is indeed correct and contained in the original relation [5].

In the following sections we describe how these techniques can be applied in finding corroborating evidence regarding claims about the information in a database at an earlier time and how the database reconstruction algorithm can be used to get more complete reconstructed relations. The techniques are currently not automated as this paper is focused on describing the logical steps to be followed during reconstruction.

5 Reconstruction from Interaction

According to Locard's exchange principle [3], the interaction or contact between two items will always result in an exchange. The technique of reconstructing data from interaction works on this principle and is synonymous to the collection of trace evidence at a crime scene.

From the reconstruction example in section 3, it is obvious that the tuples in relation H at t_3 could not be reconstructed because of two reasons:

1. the database reconstruction algorithm depends on the inverse of the update performed on relation S, which results in the generation of a partial relation S^* with missing values in one of its columns.

2. The inverse of the first query of the subsequent value block of H after time t_3, that is, the query at t_6 in value block V_{H_2} cannot be found since H was either dropped or all of its tuples were deleted at this point.

In general, a particular situation in which the database reconstruction algorithm may be unable to reconstruct required data during an investigation is when the relation to be reconstructed is deleted in the subsequent value block of the relation; one or more relations which the relation being reconstructed interacted with have been deleted; or where the re-execution of the actual query that led to the relation being reconstructed cannot be done due to the inability to determine or reconstruct a complete version of other relations involved in the query.

An alternative way of reconstructing data in these cases is to explore the interaction of the relation to be reconstructed with other relations (using the RA log) and making inferences based on the operations performed during the interaction. A summary of inferences that can be made when considering different operations in an interaction are given below:

1. **Cartesian product**: if $H \leftarrow I(A) \times J(B)$, where A and B are attributes of the relations, then:
   (a) $x \in \pi_A(H) \Leftrightarrow x \in I$
   (b) $x \in \pi_B(H) \Leftrightarrow x \in J$.
2. **Union**: if $H \leftarrow I \cup J$, then:
   (a) $x \in H \Leftrightarrow x \in I$ or $x \in J$, and this means that,
   (b) $x \in H$ and $x \notin I \Rightarrow x \in J$ and
   (c) $x \in H$ and $x \notin J \Rightarrow x \in I$.
3. **Intersection**: if $H \leftarrow I \cap J$, then:
   (a) $x \in H \Leftrightarrow x \in I$ and $x \in J$.
4. **Difference**: if $H \leftarrow I - J$, then:
   (a) $x \in H \Leftrightarrow x \in I$ and $x \notin J$
   (b) $x \in J \Rightarrow x \notin H$.
5. **Division**: if $H \leftarrow I/J$, then
   (a) $x \in H \times J \Rightarrow x \in I$. That is $H \times J \subseteq I$.
6. **Projection**: if $H \leftarrow \pi_A(J)$, then:
   (a) $H \subseteq J$, that is, $y \in H \Rightarrow y \in J$ where y are values in similar columns of H and J.
7. **Selection**: if $H \leftarrow \sigma_A(J)$, then:
   (a) $H \subseteq J$, that is, $x \in H \Rightarrow x \in J$.
8. **Rename**: if $H \leftarrow \rho_{A=B}(J)$, where A and B are attributes, then:
   (a) $J = \rho_{B=A}(H)$ and $x \in H \Leftrightarrow x \in J$.

Considering the reconstruction example in section 3, this technique can be applied to reconstruct the tuple in H instead of the empty relation H^* generated from the reconstruction algorithm. It is important to note that the technique of reconstruction from interaction is not independent and requires the usage of the inverse operators of the relational algebra or the use of the reconstruction algorithm in regenerating other relations that might be involved in an interaction. This technique can be used in reconstructing the tuples in a relation by taking the following steps. The reconstruction of the tuples in relation H at t_3 (problem from section 3) is used to provide an example of the process at each step.

1. Identify all the interactions involving the relation to be reconstructed from the RA log. There should be at least one interaction before and after the deletion of the relation. For example, the interactions of H in figure 4 include the query at t_5 (that is, $J \leftarrow H \cap I$) and at t_7 (that is, $H \leftarrow I - J$).
2. Determine the tuples in the other relations involved in the interaction(s) that occurred after the deletion of the relation of interest either through the reconstruction algorithm or the inverse operators of the relational algebra. For example, we need to find the inverse of the query $H \leftarrow I - J$ in order to determine the tuples in J since relation I is known. Thus, we have[3]:

$$-^{-1}(H) = J^* = I - H$$

which is as shown in table 4.

Table 4. Relation J^* from the inverse difference operation

J^*	Word	Number
	five	5

3. The last step involves making inferences from the other relations that have been reconstructed in step 2, and which were also involved in an interactions with the relation being reconstructed before its deletion. For example, the relation J was involved in an interaction with H at t_5 (that is the query, $J \leftarrow H \cap I$) and since this involves an intersection operation, the inferences described earlier implies that every tuple in J must also be in H. That is,

Table 5. H through Reconstruction from Interaction

H	Word	Number
	five	5

we have the relation H which is given as table 5 instead of the earlier empty relation in table 3.

6 Reconstruction through Iteration

Another technique that can be used in reconstructing the information in a database is through the iteration of the database reconstruction algorithm and the queries in the RA log, and making inferences from tuples generated and queries performed during the process.

[3] Although the resulting J^* is complete when compared with the original J at t_7, this is not always the case with inverse difference operator.

	Word	Number
R_r	five	5
	six	6

	Word	Number
S_r	four	4
	null	5

Fig. 5. Reconstructed relations R_r and S_r

The technique works on the notion that if the queries in a log are re-executed using some reconstructed relations, then the final instance of the database generated after the re-executions should be the same as the current instance of the database. Since it was proven in an earlier work [5] that the output generated from the database reconstruction algorithm is correct, any difference between the current instance of the database and the instance generated from the re-executions implies that there are some missing data in one or more relations involved in the queries that were re-executed. The differences identified between the two database instances can be used to make inferences and reconstruct the missing data in the relations involved.

Considering the reconstruction example in section 3, this technique can be applied to reconstruct the tuple in H instead of the empty relation H^* generated from the reconstruction algorithm. The steps involved in this technique are listed below. The reconstruction of the tuples in relation H at t_3 (problem from section 3) is used to provide an example of the process at each step.

1. Attempt the reconstruction using the database reconstruction algorithm and identify other relations that needed to be reconstructed. For example, our attempt to reconstruct relation H at t_3 in figure 4 required the reconstruction of relations R ans S. For simplicity, we will use a subscript r to denote relations that were reconstructed or generated from reconstructed relation. Thus, the reconstruction of relations R and S generated the relations $R_r = R$ and $S_r = S^*$ (as explained in section 3) given in figure 5.
2. Re-execute the queries in the log using the reconstructed relations and make possible inferences whenever a reconstructed relations differs from the current instance on the database. For examples, in the reconstruction of H, we can re-execute the queries in figure 4 using the relations R_r and S_r. The re-execution process is shown in figure 6.

   At time t_4 of the re-execution, the relation I_r generated differs from the relation I in the current instance of the database. A comparison of the two relations (figure 7) shows that I contains a tuple that is not in I_r and I_r contains a tuple that is not in I. It is possible to assume that the *null* value in I_r is indeed the value "seven" since there is only one column with a missing value and the second column in both I and I_r matches. Alternatively, we can make inferences from the tuple in I which is not in I_r. That is, since I is a union of R and S, then the tuple <seven, 5> should be in either R_r or S_r. However, since we are sure that the relation R_r is complete, it implies that the tuple is in S_r. That is, S_r is given as table 6. Since the relation S_r generated from the inferences are exactly the same as the current

t_0 : Reconstructed relation R_r

R_r	Word	Number
	five	5
	six	6

t_1 : Reconstructed relation S_r

S_r	Word	Number
	four	4
	null	5

t_2 : $H_r \leftarrow R_r \cap S_r$

H_r	Word	Number

t_3 : $S \leftarrow \mathbb{F}_{update(word=seven)}(\sigma_{number=5}(S))$

Skipped since S_r was generated from its inverse.

t_4 : $I_r \leftarrow R_r \cup S_r$

I_r	Word	Number
	four	4
	five	5
	six	6
	null	5

Fig. 6. Re-execution of the query log using the reconstructed relations

I_r	Word	Number
	four	4
	five	5
	six	6
	null	5

I	Word	Number
	four	4
	five	5
	six	6
	seven	5

Fig. 7. Reconstructed relation I_r and current relation I

Table 6. Table S_r generated from re-execution and inferences

S_r	Word	Number
	four	4
	seven	5

instance of the relation S, no further inferences can be made at this point. The concluding part of the re-execution process is shown in figure 8. The relation H_r generated from the re-execution process at time t_7 should be the same as the current instance of H on the database. But, this is not the case (as shown in figure 9). Again, the differences between the two relations can be used to make inferences about the data in the database. Relation H_r contains the tuple <five, 5> which is not present in the current instance of H, this implies that some data was missing in the reconstructed relations used to compute H_r. Since the tuple, <five, 5> is not expected to be in H_r, then the only possibility is that it should have been in the relation J_r since

t_5 : $J_r \leftarrow H_r \cap I_r$

J_r	Word	Number

t_6 : $H_r \leftarrow \emptyset$

H	Word	Number

t_7 : $H_r \leftarrow I_r - J_r$

H_r	Word	Number
	four	4
	five	5
	six	6
	null	5

t_8 : $R \leftarrow \rho_{number\ =\ numeral}(R)$

Skipped since R_r was generated from its inverse.

Fig. 8. Re-execution of the query log using the reconstructed relations

H_r	Word	Number
	four	4
	five	5
	six	6
	null	5

H	Word	Number
	four	4
	six	6
	seven	5

Fig. 9. Reconstructed relation H_r and current relation H

all the tuples in J_r were removed from I_r to generate H_r (from the difference operation at t_7). If the tuple is in J_r at t_7, it implies that it was also in J_r at t_5 since both times are in the same value block of J. This further implies the tuple <five, 5> was in both H_r and I_r at time t_5. Also, since t_5 and t_2 are in the same value block of H, it implies that the tuple <five, 5> was in H at t_2 and subsequently at t_3. Thus, the relation H at t_3 can be reconstructed as shown in table 7.

Table 7. H from Reconstruction through Iteration

H	Word	Number
	five	5

As with the technique of reconstruction from interaction and as shown in the example above, the technique of reconstruction data through iteration also rely on the use of the database reconstruction algorithm, inverse relational algebra and value blocks. Also, the techniques are currently not automated as this paper is focused on describing the logical steps to be followed during reconstruction. Both techniques can be used in reconstructing data when dealing situations

involving incomplete reconstruction of some other relations or the deletion of required relation at some point in the log file. The decision about which of the techniques to use will depend on the content of the log file and/or an intuitive decision of which technique is likely to enable the reconstruction of more data.

7 Conclusion and Future Work

This paper discusses an algorithm for reconstructing the information in a database at an earlier time and presents the limitation of the algorithm using a typical example. The limitation of the algorithm arises mainly because of the possibility of generating incomplete inverses when using the inverse operators of the relational algebra. Since the reconstructed relation or tuples of a relation may often be used as evidence in an investigation; to refute or support claims about the content of a database at an earlier time; or to simply get for information about an investigation, the reconstruction of incomplete data may imply that some evidence are missing.

The paper describes two different techniques that can be used in conjunction with the database reconstruction algorithm and the inverse operators of the relational algebra to generate more complete relations or provide corroborating evidence for claims about the data on a database at an earlier time. The first technique works based on Locard's exchange principle while the other rely on the iteration of the reconstruction algorithm and re-execution of the queries in the log file. Both techniques are described using a typical example.

Future work will entail investigating if these techniques can be used to reconstruct all the tuples in a relation always and if not, describe the conditions under which complete relations can be reconstructed. In addition, we will determine whether the information recovered from a database using the reconstruction algorithm and these techniques is "maximal" in that one determines that the log contains no further information that may be used to reconstruct values.

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

References

1. Carrier, B.: Defining digital forensic examination and analysis tools using abstraction layers. International Journal of Digital Evidence 1, 2003 (2002)
2. Casey, E.: Digital Evidence and Computer Crime - Forensic Science, Computers and the Internet, 3rd edn. Academic Press (2011)
3. Chisum, W.J., Turvey, B.: Evidence dynamics: Locard's exchange principle & crime reconstruction. Journal of Behavioural Profiling 1(1) (January 2000)
4. Codd, E.F.: The Relational Model for Database Management, Version 2. Addison-Wesley (1990)
5. Fasan, O.M., Olivier, M.S.: Correctness proof for database reconstruction algorithm. Digital Investigations (2012)

6. Fasan, O.M., Olivier, M.S.: Reconstruction in database forensics. In: Peterson, G., Shenoi, S. (eds.) Advances in Digital Forensics VIII. IFIP AICT, vol. 383, pp. 273–287. Springer, Heidelberg (2012)
7. Fowler, K.: SQL Server Forensic Analysis. Addison Wesley Professional (2008)
8. Garfinkel, S.L.: Digital forensics research: The next 10 years. Digital Investigation 7, S64 – S73 (2010); The Proceedings of the Tenth Annual DFRWS Conference
9. Litchfield, D.: Oracle forensics part 1: Dissecting the redo logs. NGSSoftware Insight Security Research (NISR) Publication (March 2007)
10. Litchfield, D.: Oracle forensics part 2: Locating dropped objects. NGSSoftware Insight Security Research (NISR) Publication (March 2007)
11. Litchfield, D.: Oracle forensics part 3: Isolating evidence of attacks against the authentication mechanism. NGSSoftware Insight Security Research (NISR) Publication (March 2007)
12. Litchfield, D.: Oracle forensics part 4: Live response. NGSSoftware Insight Security Research (NISR) Publication (April 2007)
13. Litchfield, D.: Oracle forensics part 5: Finding evidence of data theft in the absence of auditing. NGSSoftware Insight Security Research (NISR) Publication (August 2007)
14. Litchfield, D.: Oracle forensics part 6: Examining undo segments, flashback and the oracle recycle bin. NGSSoftware Insight Security Research (NISR) Publication (August 2007)
15. Olivier, M.S.: On metadata context in database forensics. Digital Investigation 5(3-4), 115–123 (2009)
16. Palmer, G.: A road map for digital forensic research. Technical report. In: First Digital Forensic Research Workshop (DFRWS), Utica, New York (August 2001)
17. Wright, P.M.: Oracle database forensics using logminer. Next Generation Security Software (January 2005)
18. Wright, P.M., Burleson, D.K.: Oracle Forensics: Oracle Security Best Practices. Rampant Techpress (2010)