Reconstruction in Database Forensics

by

Oluwasola Mary Adedayo

Submitted in partial fulfilment of the requirements for the degree of Philosophiae Doctor in Computer Science

In the Faculty of Engineering, Built Environment and Information Technology,
University of Pretoria

February 2015

Reconstruction in Database Forensics

by

Oluwasola Mary Adedayo

E-mail: madedayo@cs.up.ac.za

Abstract

The increasing usage of databases in the storage of critical and sensitive information in many organizations has led to an increase in the rate at which databases are exploited in computer crimes. Databases are often manipulated to facilitate crimes and as such are usually of interest during many investigations as useful information relevant to the investigation can be found therein.

A branch of digital forensics that deals with the identification, preservation, analysis and presentation of digital evidence from databases is known as database forensics. Despite the large amount of information that can be retrieved from databases and the amount of research that has been done on various aspects of databases, database security and digital forensics in general, very little has been done on database forensics. Databases have also been excluded from traditional digital investigations until very recently. This can be attributed to the inherent complexities of databases and the lack of knowledge on how the information contained in the database can be retrieved, especially in cases where such information have been modified or existed in the past.

This thesis addresses one major part of the challenges in database forensics, which is the reconstruction of the information stored in the database at some earlier time. The dimensions involved in a database forensics analysis problem are identified and the thesis focuses on one of these dimensions. Concepts such as the relational algebra log and the inverse relational algebra are introduced as tools in the definition of a theoretical framework that can be used for database forensics.

The thesis provides an algorithm for database reconstruction and outlines the correctness proof of the algorithm. Various techniques for a complete regeneration of deleted or lost data during a database forensics analysis are also described. Due to the importance of having adequate logs in order to use the algorithm, specifications of an ideal log configuration for an effective reconstruction process are given, putting into consideration the various dimensions of the database forensics problem space. Throughout the thesis, practical situations that illustrate the application of the algorithms and techniques described are given.

The thesis provides a scientific approach that can be used for handling database forensics analysis practice and research, particularly in the aspect of reconstructing the data in a database. It also adds to the field of digital forensics by providing insights into the field of database forensics reconstruction.

Keywords: Digital Forensics, Database Forensics, Reconstruction, Forensic Analysis, Database Management System.

Supervisor: Prof. Martin S. Olivier
Department: Department of Computer Science
Degree: Philosophiae Doctor

Acknowledgements

Words would not express my gratitude to the faithful and true God, who speaks and brings to pass, who knows the end of a thing from the beginning and who orchestrates all things and decisions to work for my good. You have been the lifter up of my head and I am forever grateful to You, the lover of my soul.

My sincere gratitude also goes to my supervisor, Professor Martin S. Olivier, you are a mentor worthy of emulation and I could not have asked for a better supervisor. Thank you very much for the time, money and effort that you invested in this research and my career. I appreciate all your kind words of encouragement from the very beginning to the end of this research.

I am grateful for the financial support that I received from various organizations in order to complete this research. Thanks to everyone who recognized the potential in me and gave me a chance. Thanks to the Organization for Women in Science for the Developing World (OWSDW) for the OWSD fellowship award, the African Network of Scientific and Technological Institutions (ANSTI) and the German Academic Exchange Service (DAAD) for the ANSTI/DAAD fellowship, L'ORÉAL and UNESCO for the L'ORÉAL/UNESCO fellowship for Women in Science in Sub-Saharan Africa, Google for the Google Anita Borg Memorial fellowship and lastly to the Heidelberg Laureate Forum for giving me the opportunity to meet some of the great minds in Mathematics and Computer science.

I would also like to take this opportunity to say a big thank you to all the members of

the ICSA research group that I met during my studies. You all played a part in the success story of this research.

Thanks to all my friends for your encouragement, support and understanding during this time. Thanks to the members of The Bridge Church, who always asked me how far along I was. I appreciate you all. God bless you.

This acknowledgement would not be complete without mentioning my family. To my parents, Johnson Kolawole Fasan and Eunice Modupe Fasan, thank you for giving me the opportunity to begin small. We are here now! To my siblings, Femi, Bros Ay, Bros Jhyde, Sis T, I'll always be grateful for having you in my life. To my awesome husband, Adebayomi, thank you for your support and love. Indeed love conquers all, I love you dear. To the latest addition to what I call family, Inioluwami, this had to finish because of you. I pray that you will be greater than your parents, I love you.

To everyone that has played a part in the success of this research, thank you very much.

To my maker and closest friend, I am nothing without HIS grace.

To Mum and Dad, thank you for being ever caring and always believing in me.

To Adeorimi, Adebayomi Oluwaseunfunmi, You are the best husband ever!

To Inioluwami, You rock my world babe, and I know you will be greater than me!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"Known to God from eternity are all his works."

- Acts 15:18

With the growing use of computers and technology in various aspects of everyday life, it has become almost impossible to imagine a crime that does not involve a computer or a digital aspect at some point. Criminals use technology to facilitate crimes and avoid apprehension. This has created new challenges for law enforcement agencies, security professionals and forensic examiners [29]. The increasing use of computers in facilitating crimes has also led to an increased volume of digital evidence that can be used to hold offenders accountable. This current nature of crimes has resulted in a new branch of forensic science known as *digital forensics*. Although it involves the application of science to investigations and prosecution as does the field of forensic science in general, digital forensics deals with the collection and analysis of data produced, transmitted or stored by digital devices and is aimed at clarifying the events that occurred during an incident and identifying the perpetrators [64].

Database systems are a core component of many computing systems and have a significant impact on the growing use of computers in various areas in the modern society. It is fair to say that databases play a critical role in almost all fields in which computers are used, including business, commerce, medicine, engineering, law, education and sciences [51]. The use of databases in many systems cannot be overemphasized as databases are often used to store critical and valuable information relating to an organization, her

clients, or the operations of the organization [53].

The important role played by databases in many computing systems has led to an increase in the rate at which databases are exploited in computer crimes. Databases are often manipulated in order to facilitate criminal acts and as such they are of interest during many digital forensics investigations as useful information relevant to an investigation can usually be found therein [53]. The branch of digital forensics that deals with the information found on databases is referred to as *database forensics*. As with other branches of digital forensics, database forensics is aimed at determining the root cause of an incident and finding out what was done by the perpetrator.

Due to the fact that the information stored in a database can be retrieved, updated and manipulated in various ways, database forensics often requires the recovery of lost data or the previous values of data which have been modified because of normal business operations, modifications or updates performed on the database. This requires that such updates or modifications can be reversed and is comparable to the process of reconstruction in digital forensics in that it assists an investigator to have information about both the final and the previous states of the data in a database by examining the operations performed on the database [25].

However, despite the importance of databases and the increasing amount of research in digital forensics and database security in general, only a little amount of research has been done in database forensics, even though the field is gradually gaining attention. This can be attributed to the inherent complexity of databases that is not yet completely understood in a forensic sense [100]. It is hoped that this thesis will contribute to a better understanding of database forensics, particularly the process of reconstruction in database forensics analysis.

1.1 Motivation

The need to perform a forensic analysis on a database may arise either because the database has been involved in a crime or because it contains information that may

assist in resolving a crime. In many investigations, the information stored in a database may be the required piece of evidence or information that provides more insight into the incident. Until recently, traditional digital investigations often excluded databases even though evidence can often be found in them [59]. One of the main concerns in conducting a database forensics analysis involves the lack of adequate knowledge on how the information stored in the database can be retrieved, considering the fact that the information might have existed a long time ago, updated in various ways or even deleted from the database. Many of the publications that are available on database forensics [84, 85, 86, 87, 88, 89, 90, 59, 60, 139] are either focused on the technical details of a specific database system or fail to give a scientific approach that can be used to reconstruct the information in a database.

In order to determine the data in a database at a particular time, prior to various updates or modifications, it is required that the data of interest can be reconstructed. Thus, there is need for an approach that can be applied for the reconstruction of data in different database systems.

1.2 Problem Statement

The main goal of this thesis is to describe techniques that can be used for reconstruction in database forensics. Although the term *reconstruction* is often used to refer to the process of determining the events that happened during an incident [64, 25], in the context of this thesis, it refers to the process of determining the values of the actual data stored in a database at some earlier time of interest. This thesis explores techniques that can be used for reconstructing the information in a database at an earlier time prior to various, updates, modifications or even deletion. The thesis addresses the following questions:

- What are the different categories of databases that may be encountered during a database forensics analysis?

- What are the techniques that can be applied for reversing data manipulation operations performed on a database?
- How can an algorithm for database forensics reconstruction be developed?
- Can such an algorithm be proved to be correct?
- What approaches should be used to ensure that the algorithm is capable of reconstructing as much information as possible?
- What information is required for an efficient database forensic analysis or reconstruction?

By addressing these questions, this thesis lays a foundation upon which future research in database forensics can be built and defines various techniques for the reconstruction of the data stored in a database at some time of interest.

1.3 Scope of the Research

The title of the thesis makes reference to the term, *reconstruction* which may have different meanings in different contexts. As mentioned earlier, reconstruction may be referred to as the process of identifying and analyzing the events that occurred during an incident being investigated. While this is a possible aspect of database forensics that may be considered, it is not the focus of this thesis. In the context of this thesis, reconstruction refers to the process of determining the initial values of data stored in a database, which might have been lost or updated due to normal processes, updates or other manipulations. Although the thesis involves the exploration of the operations (or queries) performed on a database, which may be viewed as the events leading to an incident, the main focus is to reconstruct the data stored in the database and not the queries or events involved.

Due to the widespread use of the relational database model in most of the current and popular database systems today, the work presented in the thesis relates mostly to relational databases. The use of relations for data representation in relational databases

is explored. Although it may possible that the ideas presented can be adapted to other database models, this possibility is not investigated in this thesis.

As will be later discussed in the thesis, database forensics may deal with the analysis of databases that have gone through several changes. These changes may range from normal modifications or updates on the database to several levels of malicious updates or changes to the database. Although the thesis later considers some of the applications and effects of the techniques described in various types of databases that can be investigated, the main focus of the thesis is on the analysis of databases that have undergone normal business changes or modifications. Other types of databases that may be investigated are considered but not explored in much detail.

Since only a little amount of work has been done on database forensics and due to the proprietary nature of many database management systems, the focus of this thesis is to add to knowledge on database forensics and to give a theoretical understanding of the process of reconstruction in database forensics. As such, most of the techniques described in the thesis are not automated and are left for future work outside this thesis.

1.4 Methodology

To address the questions raised in the problem statement, the thesis first presents an overview of the concepts involved in digital forensics. Aspects of the literature that are considered necessary for an understanding of the focus of the thesis and the ideas later presented are discussed. An overview of the concepts of database systems is also presented in order to highlight some of the characteristics of databases that distinguish database forensics from other branches of digital forensics, and reflect the need for a more specialized approach for database forensics analysis. Emphasis is placed on describing the relational data model and relational algebra since a core part of the thesis relies on an understanding of both topics.

In order to position this thesis in the right context, an overview of the literature on database forensics is also presented. Although there is only a little amount of literature

on database forensics, the analysis of the available literature leads to the solution of the first question raised in the problem statement and lays the necessary foundation for the remainder of the thesis.

A large part of this thesis uses a mathematical approach in developing the database reconstruction algorithm and handling various aspects of the algorithm such as proving its correctness and investigating its completeness. The main part of the thesis consists of the development of an algorithm that allows the reversal of queries used to manipulate a database. The algorithm uses the information available in the database log and analyses them in a way that enables the earlier values of data to be determined. Using mathematical techniques of proving correctness, the algorithm is also proven to be correct and yielding correct results. However, one limitation of the algorithm is that results generated may be incomplete due to certain instances of irreversible operations that may occur in a database manipulation. Various approaches of ensuring the completeness of the results generated from the algorithm are presented in order to address this limitation.

Since it is common knowledge that a database is not only composed of the raw data but also of the metadata that describes the data, the reconstruction algorithm is extended for the reconstruction of the database schema. This leads into the analysis of databases that might not only have been changed or manipulated under normal circumstances but which might have involved malicious changes. Techniques that can be employed for the reconstruction of the schema are explored.

The techniques described in the thesis are considered in a real life situation so as to address the last question pointed out in the problem statement. An analysis of the information required for an efficient database forensics investigation is carried out by considering six popular database management systems in use today and how their logging preferences may affect the volume of information available for an investigation. The required or ideal logging preferences for ensuring the availability of the information usually needed for a database forensics analysis are then identified and considered in the various categories of databases that can be investigated. Each of the questions

highlighted in the problem statement are answered and compiled into this thesis.

1.5 Terminology

Most of the terms used in this thesis are defined in the chapters dedicated to or related to the topics being discussed. Various definitions are given throughout the literature as required in order to aid the readability of the thesis and provide a contextual understanding of the terms. However, it is important to mention that the terms *incident* and *crime* are sometimes used interchangeable throughout the text. While it is true that an incident may not necessarily be a crime, both terms are used to refer to events that violate a law or some policies, or to events that need to be investigated. A glossary of the acronyms and symbols used in the thesis is included in Appendix B.

1.6 Thesis Layout

This thesis consists of 10 chapters with appendices to aid readability and lastly a bibliography of the literatures cited in the thesis. The details of the chapters are as follows:

Chapter 1, which is the current chapter introduces the thesis and gives a brief overview of the importance of the research conducted therein. The problem statement and the methodology describe the focus of this research and the approach used in solving the identified problems. A summary of the contributions made in this thesis is also given in this chapter.

In order to lay the necessary foundations for understanding the concepts of digital forensics, **Chapter 2** provides a discussion of the concepts of digital evidence and digital forensics. The nature of digital evidence and the processes involved in a digital forensics analysis process are described, the related literatures are also discussed.

Chapter 3 presents the background work necessary for an understanding of the properties of databases and database systems. Since a large part of this thesis deals with the use of the logs on databases, a conceptual description of the concept of logging is given

in this chapter. The relational model and the relational algebra are also discussed in detail since the remaining parts of the thesis rely on these topics.

Chapter 4 deals with database forensics and gives an overview of the importance of database forensics in digital investigations. Based on a comprehensive review of the available literature in database forensics, the dimensions involved in a database forensics investigation are identified. This provides solution for the first problem identified in the problem statement and places the rest of the thesis in perspective, as the remainder of the thesis focuses particularly on one of the dimensions identified. An overview of the processes, tools and challenges involved in database forensics research and practice is also provided in the chapter.

Chapter 5 presents one of the main contributions of this thesis which is an algorithm for reconstructing the values of data in a database at an earlier time prior to various modifications, updates or deletions. Concepts such as the inverse operators of the relational algebra, and the notion of relational algebra log and value blocks which are important aspects of the algorithm are introduced. A detailed description of the algorithm, together with application examples are also given.

Chapter 6 builds on Chapter 5 by presenting the correctness proof of the algorithm presented in Chapter 5. Using a mathematical approach, the proof of the partial and the total correctness of the algorithm is presented by building on several lemmas relating to different aspects of the algorithm and proving theorems that can be drawn from the lemmas.

Chapter 7 is another addition to the reconstruction algorithm as it considers the issue of completeness of the results generated when using the algorithm. The limitation of the reconstruction algorithm is described with typical examples and various techniques to ensure that a more complete set of data can be reconstructed in many cases are described.

Since a database is composed of both the raw data and the metadata describing the data, it is important to consider how the metadata can be retrieved and this is the

focus of **Chapter 8** of the thesis. Various ways in which a database schema can be compromised are discussed with a few examples. Different ways of reconstructing the schema by applying the reconstruction algorithm and other techniques are described with examples of how it can be done.

Chapter 9 considers real life situations relating to the techniques described in the earlier chapters of the thesis by considering the effectiveness of database logs in database forensics. This is done through a study of popular database systems in use today and their various logging preferences. The ideal set of information that should be present in a database log in order to ensure that a database forensics analysis can be effectively conducted is identified and the required logging preferences to enable the availability of the information are also considered.

Chapter 10 concludes the thesis and reflects on the future work identified during the completion of the research.

1.7 Contribution of the Thesis

The main focus of this thesis is to develop algorithms to assist digital forensics examiners in reconstructing the information in a database at a particular time, prior to various updates, modifications or deletions. The first contribution of this work is the identification of the various dimensions that may be involved in a database forensics analysis. The ability to reconstruct the data in a database at some earlier time of interest is an important aspect of forensic investigations as databases often contain information that may assist in many investigations, but which might have been modified in various ways. To the best of our knowledge, this has not been considered in any previous work. This thesis provides a new way of reconstructing the data in a database at some earlier time of interest by presenting an algorithm that can be used for the reconstruction process.

One of the requirements that enable the presentation of digital evidence in a court of law involves proving the technique used to arrive at the evidence. The second contribution of this thesis is the presentation of the correctness proof of the database reconstruction

algorithm which ensures that this requirement is met and allows us to know that results presented are right if they have to be presented in a court of law. In addition, since some data may not be easily reconstructed due to certain modifications performed on them, it is important to find alternative ways of determining such data when required. The thesis describes various techniques in which this can be done and show examples that illustrate the various approaches.

The third contribution of this thesis is the extension of the reconstruction algorithm for reconstructing the database schema. Since a database contains both the raw data and the data describing it, it is important that both sets of data can be reconstructed when necessary. This thesis is the first work that illustrates how the schema can be reconstructed using the algorithm described.

Lastly, this thesis contributes to the identification of the information required for database forensics analysis in the different dimensions of database forensics. It reveals the inadequacy of the logging preferences on some databases when it comes to the information required for a forensic analysis or reconstruction in a database. It also gives a view of the information that a database administrator should ensure is in the log for an effective database forensics analysis, should the need arise.

1.8 Summary

This chapter provides an introduction to this thesis and explains the importance of the research. It gives the problem statement and outlines the methodology describing how the objectives of the research will be accomplished. The layout of the thesis and the contributions of the thesis are also given in the chapter.

Chapter 2

Concepts of Digital Evidence and Digital Forensics

> "Study to show thyself approved unto God, a workman that needeth not to be ashamed, rightly dividing the word of truth."
>
> - 2 Timothy 2:15

The extensive use of technology in various aspects of our daily living today has led to an increased usage of computers in facilitating crimes all over the world. Over the past few years, this has created new challenges for law enforcement agencies, forensic examiners and security practitioners and has resulted in significant advances in the field of digital forensics. The aim of this chapter is to describe the terms and concepts in the field of digital forensics and also give an overview of digital evidence and the digital forensics process. Section 2.1 gives a brief history of digital forensics and describes some of the events that resulted in the emergence of the field. Section 2.2 describes the nature of digital evidence in detail. An overview of digital forensics process is given in Section 2.3. A detailed explanation of the examination and analysis stage of the digital forensics process, which involves the process of reconstruction is given in Section 2.4.

2.1 A Brief History of Digital Forensics

Over the past decade, the digital forensics field has grown from a relatively obscure tradecraft to an important part of many investigations [63]. It originated from attempts to

acquire and analyze chains of evidence relating to investigations, by computer professionals who worked with the law enforcement and/or by law enforcement personnel with very little background in computing. The early days of digital forensics were characterized by the diversity of hardware, software and applications as well as the increasing number of different file formats, many of which were poorly documented [63]. The lack of a formal process, tools and training in digital forensics was also a major issue in the early days. Although the field has progressed over the years with a rapid growth in research and professionalism, many researchers have pointed out the need for standardization and a unified process and training for digital forensics [10, 63, 93].

One of the driving forces of digital forensics research (which is also a challenge in the field) is the progressive nature of technological advancement and this has an impact on any attempt to describe the origin of digital forensics. The remainder of this section examines the fundamental attribute of the digital forensics field by reflecting on the history of the practice as well as the definitions and perspectives that have evolved over the years.

Donn Parker is perhaps the first person to perceive the emergence of computer related crimes. He described the use of digital information for investigation and prosecution of computer related crimes in the earliest one of his books [108, 110, 109] in 1976. During this time, system administrators were often responsible for the security of their own system, most of which were not networked to the outside world. System audits (which constituted the first approach to information security) were designed for ensuring the accuracy and efficiency of data processing and the information collected could be used to investigate wrongdoings [114, 108].

The need to handle computer related evidence, especially in mini-systems and mainframe computers became a major issue in the 1980s. Organizations such as the New Scotland Yard in the UK and the Department of Defense and Federal Bureau of Investigation in the US established groups of law enforcement agents that were given basic mainframe and minicomputer training and were able to assist case investigators in obtaining digital

information usable as evidence [114, 7]. However, these units were the exception because many investigations were still carried out by individual officers with minimal training, little or no equipment and no supervision. The late 80s into the early 90s, experienced a growth in the use of personal computers, an increasing awareness of digital evidence and the recognition of the need for techniques for preserving digital evidence [119].

A tremendous growth in size and maturity of digital forensics was witnessed in the next decade starting from the late 90s and was driven by the explosion of technology and an increased number of child pornography cases [114]. Digital forensics became an important part of many investigations and was useful in solving various crimes. Although very little amount of academic or research work were done on digital forensics up to the late 90s, the new millennium has experienced an increased number of journals and conferences targeted specifically at digital forensics. Some of these include the Digital Forensics Research Workshop (DFRWS) that was established in 2001, the International Journal of Digital Evidence, established in 2002 , the International Journal of Digital Investigation which was established in 2004, the IFIP Working Group 11.9 International Conference on Digital Forensics, which was established in 2005, and many more that have evolved since then.

Until the late 90s, terms such as *computer forensics* and *forensic computing* remained informally defined even though they were used in publications. In 1999, McKemmish defined forensic computing as "the process of identifying, preserving, analyzing and presenting digital evidence in a manner that is legally acceptable" [92]. The Scientific Working Group on Digital Evidence (SWGDE) defined computer forensics as "the scientific examination, analysis and/or evaluation of digital evidence in legal matters" [120]. The first broad and widely accepted definition of digital forensics was given by researchers at the first Digital Forensics Research Workshop (DRFWS) in 2001. They defined digital forensic science as "the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of

facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations" [107].

Today, the terms digital forensics, computer forensics and forensic computing are still arguably used interchangeably. However, one unifying aspect of the various definitions is the concept of digital evidence. Also, even though there has been recent concerns about the future of digital forensics due to technological advancements, fundamental changes in the computer industry and the current lack of standardization [63, 10], an increasing number of research is being done on digital forensics in general and on various branches of digital forensics such as cloud forensics, mobile forensics, network forensics, and database forensics, all of which involve the analysis of digital evidence and rely on the same underlining principle of digital forensics, despite their specific challenges. The following sections describe the concept of digital evidence and digital forensics in detail and lay a foundation for the subsequent chapters in this thesis.

2.2 Nature of Digital Evidence

There is an increasing number of evidence that can be found on computers during various investigations, even in investigations that are not necessarily electronic in nature. The law enforcement agencies and investigators have come to terms with the fact that digital data should be collected in any investigation.

The term *digital evidence* is defined as any data stored or transmitted using a computer, and that support or refute a theory of how an offense occurred or that address critical elements of the offense such as intent or alibi [29]. The data can include files stored on a computer, including text, images, video or audio files, network packets, audit trails, application logs, application metadata, internet service provider logs, or database contents or transaction logs. Although digital evidence can be used to support or refute a theory about a crime, it may also be simply used to facilitate the investigation of a crime, reconstruct a crime or incident, understand criminal motivations, identify suspects or to defend the innocent. Also, even though digital evidence may be used as a form of physical

evidence, it possesses some unique properties that define the way it should be handled and sometimes create problems that can complicate the use of digital evidence. Some of these properties include the context and volume of digital data, the completeness of digital information, anonymity of digital information, and the fragility and reproduction of digital data.

Context and volume of digital data: The volume of data that can be retrieved from a digital device is one of the challenges involved in handling digital evidence. Digital evidence is often a mixture of various pieces of information, layered on top of each other over time. Only a small portion of these data is usually relevant in many investigations and it is important to be able to extract only the useful information. However, even when the relevant data has been retrieved, the content and meaning of the information highly depends on the context in which the data was produced. For example, a hard drive may contain lots of data that may relate to hundreds of users or events. It is also possible that the relevant information on the hard drive had been generated for use by some specific device or computer program. Thus, it is necessary to retrieve relevant pieces of information, understand their context and be able to fit them together in order to correctly interpret it.

Completeness of digital information: The digital information retrieved during an investigation is often an abstraction of the actual events. Although the layers of abstraction that exists in a digital data can often be used to analyze the data [20], the record of activities available on a computer (for example, emails and log files) are often only capable of providing a partial view of the events that occurred. Thus, the absence of some data does not proof whether or not an event occurred.

Anonymity of digital information: Digital evidence is usually suggestive. That is, even though it may be used to prove that some activities was performed using a particular computer, it may be difficult to attribute the activities to an individual. For example, if a computer log contains record that a person's account was involved in an incident, it is necessary to prove that no one else could have used the person's account. Thus, a piece

of digital evidence is only one component of a solid investigation [29] and it often relies on other digital evidence.

Fragility and reproduction of digital data: Digital data is highly sensitive and can be easily manipulated or destroyed. It can be changed accidentally during collection, through a damaged storage media or by a malicious offender and this creates a huge concern during many investigations. However, a positive aspect of handling digital data is that it can be easily duplicated and the copy can be treated as original. Thus making it harder for evidence to be destroyed in only one attempt since there may be multiple copies in different places. In addition, by using the appropriate tools and techniques (for example, checksumming [79]), it is easy to determine when digital data has been tampered with or modified.

2.2.1 Admissibility of Digital Evidence

The properties of digital evidence described above raise questions about the authenticity of data and its acceptance in a court of law. A piece of evidence needs to be considered *admissible* before it can be accepted as evidence in a court of law. The conditions relating to the admissibility of digital evidence in a court of law is often determined by the law in a jurisdiction and the decision is sometimes made by the judge [29]. The admissibility of digital evidence is usually assessed based on three major characteristics of digital evidence: *relevance*, *authenticity*, and *weight*.

Relevance refers to the tendency of a piece of evidence to make the fact in question more or less probable than it would be without the evidence [43]. Authenticating a piece of evidence generally requires a testimony which reliably identifies the evidence (that is, which shows that a complete and accurate copy of the digital evidence was acquired), establish a chain of custody and integrity documentation, as well as the location of the evidence since its copying. In order for an evidence to pass the admissibility test, it must also be proven to have weight. The weight of evidence is a measurement of how much the evidence changes the probability of the fact in question. It depends on how

probable the evidence is if the fact is false [64]. Thus, to preserve the weight of digital evidence, the possibility of tampering with it must be minimized since evidence that is likely to originate from tampering as much as from the fact being proved, has no weight in proving the fact.

2.2.2 Forensic Science and Digital Evidence

Forensic science is the application of science to investigation and prosecution of crime, or the just resolution of conflict [29]. There are various disciplines in forensic science that have been accepted in courts to help in different investigations. When the scientific study of digital data (or digital evidence) relates to the investigation and/or prosecution of a crime or the resolution of a conflict, it becomes a forensic discipline. Forensic science provides scientific techniques and theories that allow digital investigators to analyze and process digital data, reconstruct a crime or incident, and test their hypotheses in order to generate strong possibilities about an incident.

In the same way that the admissibility of digital evidence can be questioned, the technique or scientific theory used in processing digital evidence can also be challenged by categorizing them as scientific evidence. If scientific evidence is based on a theory that lacks adequate experimental support or that involves a questionable process, the admissibility of the evidence may be affected. In various countries of the world, different criteria are often used to establish scientific evidence and its underlying theory. For example, the United States makes use of some criteria specified in the Daubert vs Meriell Dow Pharmaceutical Inc. case [47] for evaluating the validity of scientific evidence. These criteria include [29]:

- whether the theory or technique used can be (and has been) tested.
- whether there is a high known or potential rate of error associated with the theory or technique.
- whether there is an existence and maintenance of standard controlling the technique's operation.

- whether the theory or technique has been subjected to peer review and publication.
- whether the theory or technique enjoys a widespread or general acceptance within the relevant scientific community.

According to Olivier [101], these criteria continue to reverberate in the minds of those trying to establish a forensic discipline such as digital forensics. He described scientific correctness from the perspective of algorithmics and argues that rather than just following a set of protocols, correctness should be based on scientific facts and a formal proof of correctness of an algorithm with a quantifiable rate of error [43]. Other researchers [102, 130] have also discussed the issue of hypothesis in digital forensics and pointed out the need for a higher level of scientificness in digital forensics than what is done currently. One approach described by Olivier [101] to handle the issue of correctness is by reconstructing information and checking the validity of the reconstructed data with what is known. It is important to note that even when an algorithm is proven, correctness still depends on its correct implementation.

2.2.3 Absence of Evidence and Exchange of Evidence

Due to the ubiquitous nature of digital evidence, it is often rare to find a crime that does not have some related data that are stored or transmitted with a computer system. These data can be the evidence required for a prosecution[1], or it may be used to support other evidence or get more information during an investigation. However, even when no data can be found, it is important to remember an axiom from forensic science that states that, "the absence of evidence is not the evidence of absence"[57]. For example, 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 not used to access the site. It is important to base all assertions on solid supporting evidence and not on an absence of evidence [29]. Thus, it is necessary for an investigator to find corroborating evidence [27] that clearly demonstrates the falsity or truth of a claim when the required evidence

[1]subject to admissibility in court

cannot be found.

The principle underlying the process of finding corroborating evidence is a principle that is usually applied in a physical crime scene which is known as Locard's exchange principle, that contact between two items will result in an exchange of matter between them [34]. That is, there will always be some trace evidence with every interaction even though it may not be easily detected. For example, hair, fiber or fingerprints from a criminal are often found at a physical crime scene.

According to Casey [29] and Carrier [22], this principle applies in both the physical and digital realms because the same effect often occurs in a digital crime scene. The data entering or leaving a computer leave traces of digital evidence. The principle can also provide link between the physical crime scene and the digital crime scene as shown in figure 2.1 [29]. 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.

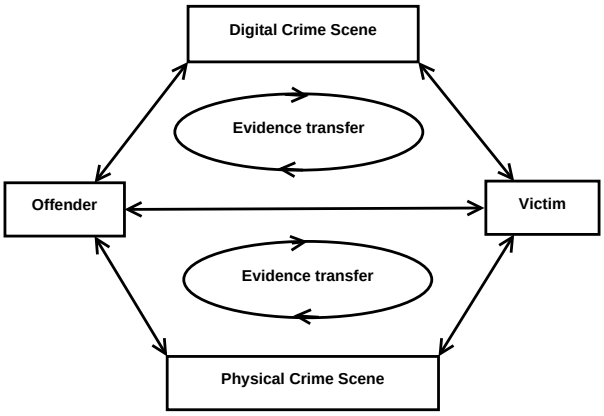

Figure 2.1: Evidence Transfer in Physical and Digital Dimensions [29].

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. However, such records may have to rely on the operating system and/or the application developer since the evidence that will be stored depends on both. The traces left on a digital crime scene (or a computer) may

also depend on the role played by the computer in the crime being investigated. In the following section, some of the roles played by computer in a crime are briefly described.

2.2.4 The Role of Computers in Crimes

Before describing the process involved in digital forensics investigation, it is important to highlight some of the roles that computers can play in a crime since it may affect the type of evidence that can be found on the computer. Over the years, several sources of digital evidence have evolved. Apart from computer systems, sources of digital evidence have expanded into areas including networks, mobile devices, media devices and databases. For simplicity, these devices are generally referred to as computers in this section. The type of evidence that can be collected from a computer and their possible uses depend on the specific role that the computer played in an incident. There are three major categories of the use of computers in a crime [109, 26]:

- Computer as the target (or object): a computer is the target of a crime when the computer is affected by the criminal act. This may include cases involving the destruction of a computer and/or the misuse of the data contained in it. For example, a computer is the target of a crime if it is used to steal intellectual property, personnel data, government records or destroy programs on a computer with the intent of creating havoc to a business or procedure.
- Computer as the instrumentality of a crime: this occurs when a computer is used as an instrument or tool for conducting or facilitating a crime. It involves the use of the computer processes and not necessarily the data stored in files. An example is when a computer is used to forge documents or commit other frauds that can be done through computer transactions.
- Computer is incidental to other crimes: a computer is incidental to a crime when the computer is related to but not compulsory for the crime to occur. This often includes the use of computers to speed up a crime or to enable multiple instances of the crime. Examples include, the usage of a computer in money laundering process,

or a situation where murder is committed by changing a patience's medication information and dosage in a hospital computer [26].

Other categories of the role of computers in a crime include Parker's [109] categories of:

- Computer as the subject: a computer is the subject of a crime when it is the site or the environment in which the crime is committed. For example, the presence of a rootkit or computer virus on a computer, which impairs its use makes the computer the subject of the incident.
- Computer as the symbol: this occurs when a computer is used to intimidate or deceive. An example is a situation where a merchant makes people to believe that they can make money from the internet through the use of some computer program. And even though the merchant has no such program, lots of people were convinced to pay for the supposed program.

In the three major categories described above, a computer may act as the source of both physical evidence (computer components) and digital evidence (data stored or transmitted using a computer). The US Department of Justice's [134] created a different set of categories that makes this distinction. However, it is important to note that evidence that proves or disproves a claim about an incident can often also be found on a computer even though the computer did not play any role in the crime or incident. For example, a computer that contains records of people whose tags were used in accessing a building may become a source of digital evidence when investigating a crime committed in the building.

Regardless of the role played (or not played) by a computer in an incident, various procedures and techniques have been developed for investigating digital evidence. In the next section, the investigative process of digital forensics is described.

2.3 Investigative Process for Digital Forensics

This section summarizes the various process models that have been developed for digital forensics investigations and describes the major stages involved in an investigation. Some of the process models that have evolved over the years include the linear process models such as [96, 107, 43, 91, 116] and other methodical approaches such as [9, 11, 22, 36, 29]. Many of these models were developed to handle different circumstances and as such may not be adequate for all types of investigations. A comparison of the different process models, shown in table 2.1 highlights four major stages in the digital investigation process: Preparation, Collection, Examination and analysis, and Presentation. The first column of the table shows different process models while the other columns show the term used to refer to each of the major stages identified in the respective process model.

Preparation: the preparation stage involves the generation of an action plan and preparation of tools and equipment necessary to perform a digital investigation in the event of an incident. It also involves getting the necessary authorizations or approval to collect data for an investigation. This stage ensures that the best evidence can be collected when the need arises.

Preservation and collection: Preservation involves the steps taken in order to maintain the integrity and authenticity of data during an investigation. It involves the prevention of activities that can damage the data available during an investigation. Operations such as making duplicate copies of data, avoidance of deletion processes and early collection of volatile data that may be lost when a system is turned off are often carried out as part of the preservation stage. Failure to ensure that data is preserved may lead to inadmissibility of evidence. The preservation stage is usually followed by the collection of data. The collection stage may involve different technologies and techniques depending on the circumstance [43]. It consists of finding and collection of digital data that may be relevant to an investigation. The collection stage may also involve the removal of the hardware or personal computers

Stages	Preparation	Collection	Examination and analysis	Presentation
Equivalent in [96]	Preparation	Assessment, Acquisition	Examination	Documenting and reporting
Equivalent in [107]		Identification, Preservation, Collection	Examination, Analysis	Presentation, Decision
Equivalent in [43]		Identify, Collect, Preserve, Transport, Store	Analyze, Interpret, Attribute, Reconstruct	Present, Destroy
Equivalent in [91]	Pre-incident preparation, Detection of incident, Initial response, Formulate response strategy	Data collection	Data analysis	Reporting
Equivalent in [116]	Identification, Preparation, Approach strategy	Preservation, Collection	Examination, Analysis	Presentation, Returning evidence
Equivalent in [11]	Preparation, Incident response	Data collection	Data analysis	Findings presentation, Incident closure

Table 2.1: Comparison of Digital Investigative Process Models.

from the physical crime scene, copying of all files on a computer or some necessary files and gathering of network traffic.

Examination and analysis: The nature and extent of the examination and analysis stage depends on the circumstances of the crime and the constraints placed on the digital investigator [29]. Digital investigation can involve two main types of analysis techniques: *live* analysis and *dead* analysis (also referred to as live forensics and dead forensics, respectively) [24]. In a live analysis, software that exist on the system being investigated are used in the analysis and as such the system is

kept running during the analysis. This technique is often used in critical systems where it may be too expensive to switch the system off. On the other hand, dead analysis does not require the use of any software on the system being investigated because it involves switching the system off and making a copy of the disk for further analysis with a trusted operating system and/or applications. Unlike the live analysis technique, dead analysis enables an investigator to repeat the steps taken during an analysis, since copies of the disk at the time of the incident can be easily made.

The examination stage entails searching of collected information for relevant data or the recovery of relevant data. The purpose of the examination stage is to find trace elements such as files, timestamps and other data that might have generated the data being examined. The analysis of these trace elements allows an investigator to draw conclusions based on the evidence gathered. The four major steps completed in this stage include, examination of collected data, interpretation, attribution and reconstruction.

Presentation: This is the last stage of the digital investigation process and it involves a summarization of the findings and conclusions of an investigation into a report that conveys the findings to other people and which may be presented in a court. The report contains an outline of the examination process as well as the data recovered during an investigation.

The preparation, preservation and collection, and presentation stages of the digital investigation process are not explored further in this thesis since the focus of the thesis (that is, reconstruction) falls into the examination and analysis stage. The rest of this thesis assumes that all the necessary preparations for data collection were made and that the steps necessary to ensure that data is preserved were followed during data collection. For interested readers, more information about the preparation, preservation and collection, and presentation stages, as well as a set of steps that should be employed by an investigator during these stages can be found in practice guides such as the ACPO guide

[5] and guides from the US Department of Justice [96, 97]. Various tools that can assist in the preservation and collection of digital evidence have also been developed and these include, EnCase [68] and Forensic Toolkit (FTK) [1].

2.4 Examination and Analysis

The aim of this section is to describe the tools and techniques used in the examination and analysis stage of the digital forensics process. The four major steps involved in the examination and analysis stage are described in the following subsections.

2.4.1 Examination of Data

The examination of collected data focuses on the extraction and preparation of data for analysis [29]. This step involves data translation, reduction, recovery and organization of relevant pieces of information in order to facilitate detailed analysis.

At its simplest level, digital data exists on physical media such as wires, magnetic disk or in form of signals. It is required that these forms of data can be translated into a form that is humanly understandable in order to examine them [28]. However, the abstraction layer that usually exists in all forms of digital data is a major challenge in data translation since it can introduce errors or loss of important information. Carrier [20] discussed these layers of abstraction, their properties and the error types that can be encountered in digital forensics analysis. A technique for mitigating the risk of errors caused by data translation is to validate data using multiple forensic tools [28]. Forensic tools should also satisfy requirements such as usability, accuracy and verifiability [20] in order to ensure that data is correctly translated.

Another aspect of the examination step deals with the reduction or filtering of data. Due to the large volume of information that may be collected during an investigation, it is important that an investigator focuses only on the relevant information. The reduction of data often include the elimination of valid system files and other known entities that are of no relevance, identification of most probable user-created data, identification of

files created within a restricted time frame, managing of duplicate files and identification of discrepancies between different forensic tools [29]. In addition, various search techniques can be used to identify the types of objects or data that are relevant to an investigation. Based on the level of search automation, Gladyshev [64] categorized current search techniques into groups including, manual browsing, keyword search, regular expression search, approximate matching search, custom search, and search of modification. Although data reduction allows an efficient and thorough examination of data, the risk of missing important information or clues makes it important for an investigator to identify the appropriate data reduction technique in any given situation.

Data examination also involves the recovery of data from various data or file types, or from data hidden amongst other collected information. The recovery process depends on the type of data, its state, the type of hardware and/or software involved, their configuration and the operating system being used. Data can often be recovered from binary files of various applications, from slack space or unallocated space containing parts of previously deleted files, from encrypted data and from other hard-to-find places such as files/file systems that are not normally used [43]. Some of the various techniques that have been developed for the recovery of data include the extraction of document headers and fragments for grafting, finding the paraphrase protecting private keys or finding unencrypted versions of data in unallocated space or swap files [29], and searching for indicators that represent potential presence of hiding methods [45].

The final part of the data examination stage involves the organization of retrieved relevant data into classes of objects with similar characteristics or features such as file formats, hardware model, software versions, combinations of words in a document or number of pixels in a picture. To organize data, it is necessary to ascertain what the digital object is, determine its distinguishing characteristics, and evaluate its source [29]. The ability to organize data facilitates the easier location of data fragments on disks and enables an investigator to analyze data more thoroughly.

2.4.2 Interpretation

Interpretation deals with the application of context to information available during an investigation. Despite the fact that data relevant to an investigation may be found on a computer, there may also be many possible sequences of events that might have resulted in the presence of such data. For example, data might have been maliciously placed on a computer by an intruder or a virus program, or it might have been produced by another program that looks similar to the one being considered [43]. The presence of a particular piece of information does not always indicate that the event indicated really took place. The purpose of interpretation is to consider alternative explanations that fit the complete collection of data and not just a single piece of information. This assists an investigator in explaining the events that might have occurred and provides a level of certainty for assumptions made [43]. Explanations which prove to be inconsistent with available information may be discarded in order to arrive at reasonable conclusions.

Interpretation involves the use of logic and other reasoning techniques and tools. However, even though tools such as EnCase [68] and FTK [1] may be employed, the process of interpretation is mainly a decision making process in which the investigator considers steps to be taken and those not to be taken in order to reach a conclusion or generate a hypothesis. Thus, it is important that an investigator does not over-interpret or mis-interpret data and must ensure that tools used also do not. Apart from the interpretation of available data, the absence of some data (that should probably be present) in a collection can also lead to various interpretations [41]. Missing data can facilitate the detection of alterations or spoliation.

A major challenge of the interpretation stage is the possibility of arriving at false positives or false negatives due to the mis-interpretation of data, over-interpretation of data, missed content, context, meaning, process, relationship, ordering, time, location, corroboration, and consistency faults. Thus creating a level of uncertainty in conclusions made. Casey [27] discusses the issue of uncertainty in digital evidence in detail. As mentioned in Section 2.2.3, a technique for handling this problem is to find corroborating evidence,

which can reduce the number of alternative explanations and evaluate the probability of falsity or truth of the different explanations. Other techniques that have been explored for mitigating errors that can arise from interpretation include the use of the redundancy in tool use, application of information physics, and substitutions and similarity comparison mechanisms [43].

2.4.3 Attribution

Attribution refers to the question of who or what caused certain effects on a piece of digital evidence [80, 33]. To identify the relationships between events, suspects and victims in a digital environment, one of the techniques used is to represent the digital environment as a finite state automaton. The states of the automaton represent places where the suspect has been, email or IP addresses he has accessed, telephone numbers called, and so on. The objective of the automaton is to establish connections between the states, and thus enabling an investigator to get an overview of an incident and locate sources of digital evidence that may be initially unknown. Examples of analysis tools used to connect events during investigation include NetMap [99] and Analyst Netbook [73]. In many situations there is a relatively small number of inputs, program interactions or events that make up the finite state automaton, thus allowing the easy construction of the automaton for attribution purpose. However, a problem arises when the complexity of the automaton is too high for a precise modelling of connections between states of the automaton [43].

Other techniques that can be used for attribution include simulation of the events, use of complexity argument, and the use of redundant information that may be available from the environment. For example, the information retrieved from a CCTV camera may be useful in asserting who was in an office at a particular time. These techniques often provide an independent support for hypotheses made about the attribution of some events to someone.

Although the purpose of authentication mechanisms (for example, biometric devices, file

ownership and/or authorship and passwords) is to confirm an identification, it is also often used as a basis for attribution and the certainty of the attribution relies on the additional or corroborating evidence that can be found. Cohen [40] explains in more detail, the concept of attribution, mechanisms used, their limits and the complexity issues with tools used for attribution.

2.4.4 Reconstruction

This section defines the concept of reconstruction in digital forensics process and gives a survey of the different techniques that have been employed for reconstruction. It also lays a foundation for Chapter 4 of this thesis, which discusses the database forensics field and the process of reconstruction in database forensics.

All the examination and analysis steps discussed earlier, as well as many of the available digital forensics tools are focused on finding evidence and identifying who might be responsible for an incident that is being investigated. However, the fact that a digital object exists on someone's computer does not prove that he is responsible for its presence. It is required that all the examination, interpretation and attribution together with the hypothesis made in these steps can be tested in order to confirm or refute the hypothesis [29]. And this is achieved through reconstruction.

Reconstruction involves the examination of digital evidence with the aim of identifying why it has its characteristics and the events that might have caused the occurrence of the evidence. The process of reconstruction assists an investigator to not only have information about the final state of an object but to also deduce its previous states by examining the events that might have involved the object [25]. It also questions the source and the time of creation of an object. Reconstruction differs from the re-enactment or recreation of an incident and criminal profiling in the sense that it is more comprehensive and directed towards a final resolution [83].

Based on the available literatures, this thesis groups reconstruction techniques into two major approaches. The first approach, *experimental reconstruction*, is focused on testing

hypotheses about an investigation in order to confirm, refute or reveal more information about the hypotheses. The second approach, *retrospective reconstruction*, involves an attempt to go backwards in time in order to determine the previous inputs and states that could have generated the available evidence [3]. A survey of the literatures relating to these two approaches is discussed below.

Experimental Reconstruction

In this approach, the process of reconstruction generally involves four main steps:

1. the creation of hypothesis based on available evidence,
2. reproduction of the evidence based on the hypothesis made,
3. comparison of reproduced evidence and the original evidence,
4. Evaluation or refining of hypothesis for further reconstruction.

Some of the work that have been done on experimental reconstruction of a physical crime scene include that of Miller in James et al. [77], Rynearson [118], Bevel and Gardner [12] and that of Chisum [133]. Miller [77] describes a process of reconstruction which consists of five phases:

- collection of evidence from the crime scene,
- formulation of an initial conjecture about the events at the crime scene,
- formulation of a reconstruction hypothesis about the incident, based on the examination of the physical scene and evidence,
- testing of the hypothesis to confirm or refute all or some aspects of the hypothesis,
- and lastly the formulation of a reconstruction theory.

Rynearson [118] incorporates the "common sense reasoning" and its applicability in forensics science to the examination of evidence at a crime scene. His method of reconstruction concentrates on the evaluation of a crime scene and the recognition of individual objects, relationships between objects, or environmental observations. His method is focused on

revealing relational clues (that come from an object's location and orientation with respect to other objects), functional clues (that come from the operational conditions of the object), and temporal clues (that come from the interaction of time and environment on the evidence) [118] and determining how an object happen to be. The process involves getting an initial idea about the crime scene, followed by a reconstruction of events that might have occurred and the development of a hypothesis. The hypothesis can then either be confirmed or refuted depending on whether or not evidence is found to support it. If it is refuted, the reconstruction process retracts to the point of the contradiction in order to show why the hypothesis is not valid.

A formal procedure for reconstruction was developed by Bevel and Gardner [12]. They describe the term *event* as an occurrence at the macro level and the term *event segments* as the micro level events that make up an event. For example, a login and logout on a computer that has been used to commit a crime may be viewed as event segments. Their reconstruction procedure involves the collection and examination of evidence, followed by the creation of event segments, and the sequencing and grouping of event segments into larger events. Once the sequence of events and event segments is determined, a flowchart representing the incident can then be created in order to get a complete picture of the incident.

In relation to a digital crime scene, Stephenson [126, 127] proposed the use of a colored Petri net model for testing event reconstruction hypotheses. This approach involves the definition of a petri net model for the system being investigated and the simulation of events to determine if they could have occurred. The stages in the reconstruction process include, collection of evidence, analysis of individual events, preliminary correlation of events, event normalization, event deconfliction, second level correlation, timeline analysis, chain of evidence construction, and corroboration [126]. This model of reconstruction is more applicable in large and complex investigations.

Based on the principle of relational, functional and temporal information that can be found in a piece of digital evidence [133], Casey and Turvey [29] describe three analysis

techniques for reconstruction:

1. Relational analysis: involves the use of the relationships between suspects, victims and crime scene in identifying which event could have occurred and where additional evidence can be located.
2. Functional analysis: assesses how a computer functions and determines if a computer or an individual is capable of performing the event that is believed to have occurred at a particular time.
3. Temporal analysis: creates a chronological list of events by exploring the time information available on a computer. This information may include MAC times[2], date-time stamps on log files or databases showing times of various activities. The objective of temporal analysis is to develop a timeline [70] or a histogram of times which may be useful in recognizing patterns and understanding deviations from regular events [29].

Carrier and Spafford [25] present a more formalized approach to experimental reconstruction which is similar to that of Bevel and Gardner [12]. Their reconstruction model consists of five phases. The evidence examination phase examines each piece of digital evidence in an attempt to identify and individualize it. The reliability and credibility of the evidence is also examined. The second phase is the role classification phase which examines each object and identifies what type of information it has, based on the functional, relational, and temporal categories [29]. The event construction and testing phase entails the correlation of cause and effect objects using either a forward or a backward search technique [25]. The fourth phase is the event sequencing phase during which events are ordered based on the time at which they occurred. The last phase involves the testing of hypothesis in order to attach a confidence level to each hypothesis and determine which ones can be confirmed or refuted. The Petri net model [126] of reconstruction can also be applied in the testing phase of their reconstruction model.

[2]Time of last Modification, time of last Access, and time of Creation.

Retrospective Reconstruction

The retrospective (or backward) reconstruction is another approach to reconstruction in the digital forensics process. Many of the techniques that have been proposed in this regard involve the definition of the system to be investigated as a finite state machine (FSM). The reconstruction of events is done by going backwards in time (or in transitions) from the final state (or the state in which the system was discovered) to determine the previous inputs and states that must have or could have produced the evidence. Scenarios that disagree with the available evidence are then discarded.

Gladyshev and Patel [65] proposed a reconstruction technique that uses FSMs. The generic algorithm presented in their work computes the set of all possible explanations for the evidence, with respect to the FSM generated for the system involved. The steps involved in their reconstruction algorithm includes a procedure for computing observation sequences, meaning of observation sequences, and how these can be combined into the meaning of the evidence [65]. Due to the exponential complexity of the algorithm, it is more suited for small systems [64].

Another backward reconstruction mechanism similar to that of Gladyshev and Patel [65] is the hypothesis based approach proposed by Carrier [23] in that it also uses a FSM to model the system being investigated. He presents a history model that can be used either for reconstruction or to identify assumptions made during an investigation. The formulation and testing of hypotheses is done through the application of the scientific process [49, 83] in the investigation process.

Cohen [43] outlines some of the hindrances that may arise (based on the concept of information physics) when using the retrospective reconstruction approach. For example, it is possible that there are several inputs that could have generated a piece of evidence and this possibility may increase the time and space complexity of a retrospective reconstruction technique. However, it is important to note that these hindrances may be negligible in certain situations, so that the retrospective reconstruction technique is adequate in

such cases. Also, regardless of the reconstruction approach used, there are various challenges that must be overcome in order for the result of the reconstruction process to be admissible. Some of the questions that should be asked after a reconstruction in order for it to be admissible are discussed in [43].

2.5 Summary

This chapter describes the concept of digital evidence and digital forensics. A brief history of digital forensics is given to show how technological advancement and the need to analyze data from digital equipment during an investigation have caused a tremendous growth in the digital forensics field over the last decade. The nature of digital evidence is described in order to highlight the properties and challenges involved in handling digital evidence. An important aspect of digital evidence deals with the exchange of evidence between a physical and a digital crime scene. And even though evidence may sometimes be absent (or cannot be found), this does not prove that it does not exist and it is important to find corroborating evidence to support or refute a claim.

The chapter also discusses some of the digital forensics process models and gives attention to the examination and analysis stage during which reconstruction takes place. The two general approaches of reconstruction are described as these lay a foundation for many of the concepts which are discussed in the subsequent chapters of this thesis.

Chapter 3

Database Systems

> "For precept must be upon precept, precept upon precept, line upon line, line upon line, here a little, there a little."
>
> - Isaiah 28:10

The focus of this thesis is on the reconstruction process of the forensics analysis of a database system. In Chapter 2, the concepts of digital forensics and digital evidence have been explained. In order to understand the application of digital forensics techniques in databases and highlight some of the relevant aspects of databases in relation to forensics analysis, this chapter discusses some of the concepts and architecture of databases and database management systems (DBMSs). The chapter positions the thesis within the context of database forensics and introduces some of the basic terminology used in the rest of the thesis. Section 3.1 gives a brief introduction to the concepts of database systems. In Section 3.2 and Section 3.3, the characteristics and advantages of database systems are described. Section 3.4 discusses the concepts of logging and the use of logs in ensuring database consistency and recovery. A brief description of database models is given in Section 3.5. Section 3.6 explains the relational database model and explains some of the concepts used in the rest of the thesis.

3.1 Concepts of Database Systems

Databases play an important role in various organizations where computers are used. Databases are often used to store large amounts of sensitive and critical information

relating to an organization and/or her clients, and assist in running the organization's business. In order to adequately describe the concept of database systems, a general definition of a database is required.

A *database* can be defined as a logically coherent collection of data, which is designed for some specific purpose and an intended group of users [51]. Databases have a major impact on the increasing usage of computers and are among the most important applications of computer technology. The main purpose of a database is to provide multiple users with a structured way of handling real world data using various application programs that enable interaction with the database or a *database management system*.

A database management system (DBMS) is a collection of programs that enables users to create and manage a database. That is, a DBMS is a software system that facilitates the processes of creating, storing, retrieving, editing and updating of the data in a database. A database together with the software used to manipulate (execution of operations such as querying, updating, and report generation) the data in the database is referred to as a *database system*. Figure 3.1 [122] illustrates the components of a simplified database system environment and shows their interaction. As shown in the figure, a DBMS has three basic components that provide the following facilities [122]:

1. Data definition language (DDL): this is used by the database users to define data types, structures and constraints on the data to be stored in the database. The DDL compiler in the DBMS processes the DDL statements from the user and generates an object schema that is stored in the DBMS catalog.
2. Data manipulation language and query facility: Manipulations that can be done on data include retrieval, insertion, deletion, and updates. The DBMS provides a data manipulation language for these purposes. It also provides general query facility through the use of a query language such as SQL.
3. Software for controlled access to the database: The DBMS provides controlled access to the database. This includes the prevention of unauthorized access to the database, provision of a concurrency control system to allow shared access of the

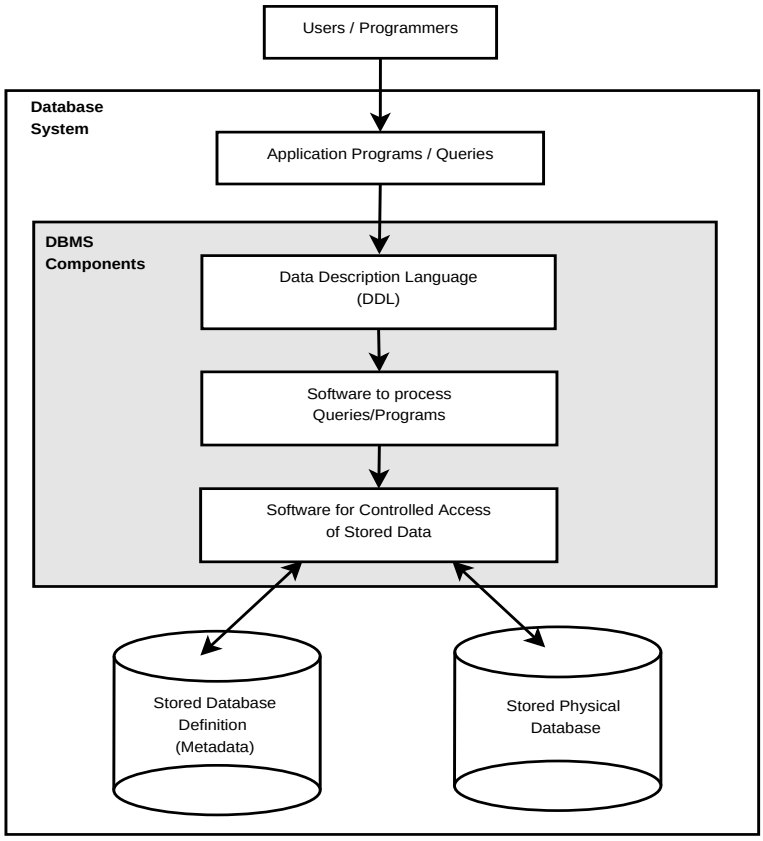

Figure 3.1: A Simplified Database System Environment [122].

database, and the activation of a recovery system to ensure that a database can be restored to a previous consistent state in the event of a hardware, software or any other failure.

Although a database is really a set of related files, there are various characteristics that distinguish a database system from a traditional file management system. For example, in traditional file management, each user determines the content of the files needed for a specific application and files are not easily linked to one another. This results in redundancy in the creation and storage of data records when one or more users require the same set of data. On the other hand, a database system maintains a single repository of data that can be accessed by different users. A detailed description of the main characteristics of a database system is discussed in the next section.

3.2 Characteristics of Database Systems

Database systems are used in various organizations and application areas. Although the size and the complexity of a database vary depending on different factors, including the amount of information contained in the database and the number of users, there are several fundamental characteristics of database systems that unify different databases and make data management more efficient. Some of these characteristics are described in the following sections.

3.2.1 Structured and Self Describing

One of the fundamental characteristics of a database system is that it does not only contain the database but also contains a complete definition and description of the database and the data contained. This description is known as the *metadata* or *database schema* and is stored in the system *catalog*. The metadata contains details about the structure, type and the format of each file and data item, as well as the relationship between the data and any other constraints [51, 98]. The metadata is used by the DBMS to provide information to database users or application programs that need certain information about the database structure. Since the DBMS software is not written for one specific database application, the metadata provides the information needed by a DBMS in order to understand the structure, type and format of files and data that it will access.

One important consequence of this characteristic is that there may be many database instances (that is, the data in the database at a particular time) that correspond to a particular database schema. The DBMS is partly responsible for ensuring that every instance of the database satisfies the structure and constraints specified in the schema or metadata.

3.2.2 Concurrent Use and Multiple Views of Data

A typical database allows multiple users to access the database at the same time, particularly in cases where the data for different applications or purposes are merged and maintained in a single database. The DBMS is required to put in place adequate measures to ensure that several users do not try to update the same data at the same time, so that the integrity of data can be maintained. One of the mechanisms used to facilitate the sharing of data involves allowing database users to have different *views* of the database depending on their access rights and data need. A data view may consist of a subset of the data stored in the database or it may consist of data that is derived from the database files but which is not explicitly stored in the database [51].

3.2.3 Data Abstraction

Another fundamental characteristic of a database system is that it provides some level of data abstraction by providing users with a conceptual representation of data that hides details of how data is stored [51]. This abstraction is achieved by using a *data model*. A data model is a set of logical concepts used to describe data types, data relationships and constraints that should hold on the data stored in a database. Most data models also include a set of operations for modifying the database. Database users refer to this conceptual representation of files when an information is required and the DBMS extracts the details of the file storage from the catalog on behalf of the users. Over the years, several data models have been proposed and can be categorized based on the type of concept used to describe the data structure:

1. High-level or conceptual data models: these data models provide concepts that are similar to the way users perceive data.
2. Low-level or physical data models: these data models provide concepts that describe the details of how data is stored in the computer.

3. Implementation or representational data models: these are data models that provide concepts that may be understandable to the user but which also reflects how data is organized within the computer [51].

Regardless of the data model used, the description of the database (database schema or metadata) is always separated from the database itself.

3.2.4 Program-Data Independence

Application programs do not require a knowledge of the structure, format or type of data and data files in order to get access to the data stored in a database. This is because the metadata, which is stored in the database catalog, is separated from the application programs trying to access data. Programs communicate with the DBMS through standardized interfaces and languages (for example, SQL) in order to understand the database structure or access data. This property is often referred to as program-data independence and it implies that applications can be totally separated from the data in a database. The DBMS has an exclusive right of access to data and the metadata. The property also implies that database internal reorganizations or improvement of efficiency would have no effect on the operation of application programs [98].

In order to help in achieving this characteristic of database systems, a three-schema architecture also known as the ANSI/SPARC[1] architecture [132, 78] was proposed. Although from a user perspective, most database systems have a similar basic architecture [98], the ANSI/SPARC architecture is described to provide insight into some of the interesting aspects of a database and have a comparable abstract picture of a database. The main goal of the three-schema architecture is to separate user applications and the physical database. The logical structure of the architecture is shown in figure 3.2 [19, 51].

[1] American National Standards Institute/Standards Planning and Requirements Committee.

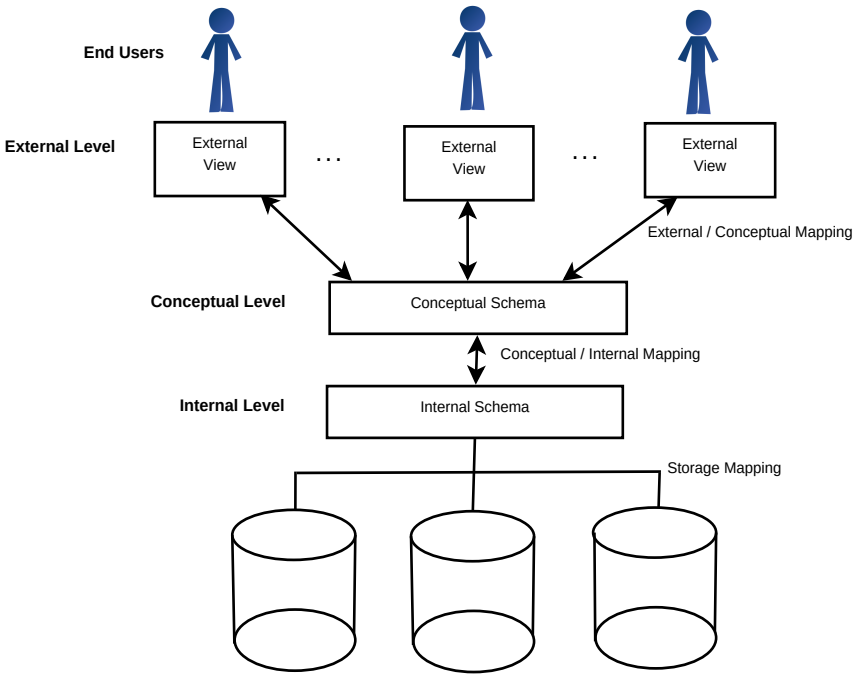

Figure 3.2: The Three-Schema Architecture [19, 51].

The Three-Schema Architecture

The ANSI/SPARC architecture defines a database schema at the following three levels [51]:

1. Internal level: the internal level consists of an internal schema that describes the structure of the database from a view that is closest to the physical storage. Thus, it describes the complete detail of data storage methods, access paths and other technical aspects of the way the data is physically stored. A low-level data model can be used for internal schemas.

2. External level: the external level or the view level contains the external schemas or the external views of the database. Each of the external schemas is a collection of data representing the properties and relationships in the database that affects a specific user or group of users. The external level is the view closest to the users and is concerned with how users view data. Thus, it describes the information about the user view (the part of the database that a particular user or group is

interested in), and the methods and constraints associated with this information but hides the rest of the database from the user. A high-level data model or an implementation data model can be used for external schemas.

3. Conceptual level: the conceptual level is concerned with a community of user views. That is, the conceptual schema describes the structure of the whole database. It gives a global description of the database by describing the entities that make up the data, together with their properties, data types, interrelationships and constraints. However, the details of the physical storage are hidden so that the conceptual level acts as an intermediary between the internal level and the users. A conceptual schema can be based on a high-level data model or an implementation data model.

The three-schema architecture gives a description of the data in the database. However, the data is only existent at the physical level. User requests to access data are processed by mappings between any two of the three levels. This assists in ensuring program-data independence in a database system.

Three-Schema Architecture and Data Independence

This section gives a brief description of how the three-schema architecture assists in achieving data independence. The concept of data independence can be viewed as the ability to change the schema at one level of a database system without changing the schema at the next higher level [51]. Thus, there are two types of data independence:

1. Physical data independence: this is the ability to change the internal schema without having to change the conceptual or the external schema [51]. This property assists in ensuring that the reorganization of files in a database does not affect the structure of the whole database or how the users view the database.

2. Logical data independence: this is the ability to change the conceptual schema without having to change the external schema or modify the application programs. This property assists in ensuring that modifications to the database do not call for changes to the application programs.

The DBMS catalog contains the mapping information between any two levels and facilitates data independence by modifying the mapping information whenever any of the schemas involved is changed, instead of having to change all the connected higher level schema. However, a consequent disadvantage of having these mappings is that they create an overhead when compiling or executing a query or program, and sometimes cause inefficiencies in the DBMS. Techniques such query optimization [66, 69, 51] are used in many DBMSs to speed up the execution of queries.

3.2.5 Backup and Recovery

The DBMS provides facilities for recovering from failures, including system crash, disk failure, power failure and transaction[2] errors. The DBMS ensures that transactions submitted are either entirely completed with their effect recorded in the database or the transaction has no effect whatsoever on the database or on any other transaction [51]. A transaction is expected to have properties, often referred to as the ACID (Atomicity, Consistency, Isolation and Durability) properties in other ensure that the database is recoverable. These properties are enforced by the concurrency control and recovery method of the DBMS. The ACID properties are described as follows:

- **Atomicity** implies that a transaction is an atomic unit of processing and it should either be performed in its entirety or not at all.
- **Consistency** implies that if all the operations of a transaction are completely executed, the database is transformed from one consistent state to another [75].
- **Isolation** implies that a transaction should appear as if it is executed in isolation from other concurrently running transactions.
- **Durability** or **Permanency** means that once a transaction is successfully completed, the changes made by the transaction must persist in the database and not be lost even if a system failure occurs.

[2]A transaction is the execution of a program or query(ies) that accesses or changes the content of the database.

These properties allow the recovery of a database to be possible in the event of a failure. To recover from failures, some of the concepts used in the recovery process include the use of logs, checkpoints, and the use of commit points. A large part of this thesis deals with using the log files in databases. As such a detailed description of the concepts of logging in databases is discussed in Section 3.4. The commit point of a transaction is reached when all the operations that access the database in the transaction have been executed successfully and the effect of all the transaction operations on the database has been recorded in the log file. Beyond this point, a transaction is said to be committed and its effect must be recorded permanently in the database [51]. The recovery concepts are used in conjunction with techniques such as; Transaction rollback, immediate update and deferred update.

- The deferred update technique do not perform any update on a database until after the transaction reaches its commit point or is committed. If a transaction fails before reaching its commit point, it would not have modified the database in anyway. Before the commit point is reached, transaction updates are usually recorded in the main memory buffers maintained by the DBMS. Before a commit, updates are permanently recorded in the log and are then written to the database on disk after commit. Although nothing would have to be undone if a transaction fails before reaching its commit point, it may be necessary to redo the effect of the operations of a committed transaction from the log, because their effect may not yet have been recorded in the database on disk [51].
- The immediate update technique may allow some operations of a transaction to update the database before the transaction reaches its commit point. However, these operations are typically recorded in the log and forced to disk before they are applied to the database. If a transaction fails after updating the database and do not reach its commit point, the effect of its operation is undone through a *transaction rollback*. Also in the event of certain failures, it may be necessary to redo the effect of the operations of a committed transaction from the log because

their effect may not yet have been recorded in the database [51].

- Transaction rollback technique is used to ensure that any data item values changed by a transaction that eventually fails after updating the database are returned to their previous values prior to the update. In the event of a failure, the log is searched for all transactions that have a record in the log that they have been started but with no record of their completion. These transactions are rolled back to undo their effect on the database. Transactions that are recorded to have been completed in the log, must also have written all the changes made in the log, so that their effect on the database can be redone from the log records if necessary. The transaction rollback technique involves the use of records stored in the log to ensure that the database is in a consistent state.

It is important to note that the transaction rollback technique is fundamentally different from the concept of reconstruction discussed in Section 2.4.4. Transaction rollback deals with retrieving values that are not yet permanently stored in the database from the log records while the concept of reconstruction deals with finding events, data or states of a system that could have generated or led to the current state of the system. The difference is of utmost importance in understanding the focus of the remaining chapters of this thesis.

Database backups also provide a way for databases to recover from events such as a disk crash or other physical failures that affect a large part of a database. A backup involves a periodical copying of the whole database and log unto a storage medium. Critical applications are often backed up and moved to safe locations. In the event of a failure, a past consistent version of the database can be loaded onto the disk from the backup and the available log records may then be reapplied in some cases.

3.3 Other Advantages of Database Systems

Apart from the characteristics of database systems discussed in the last section, there are other advantages of using database systems which also define how databases are handled.

Some of these are described below.

1. Redundancy control: database systems integrate the views of different users during database design and allow several users to access views of the database at the same time. This removes the redundancy that would have been involved with storing the same data several times.

2. Enforcing data integrity and restricting unauthorized access: Database applications often have integrity constraints that must hold on the data [51]. Integrity constraints are specified in a database and are expected to hold on every instance of the database. There are different types of constraints that can be defined on database, some of which include, key constraints, entity integrity and referential integrity constraints. The database administrator is responsible for specifying the constraints that should apply on a database during the designing of the database.

   In addition to the data integrity constraints, the database catalog also contains authorization constraints that are used to specify which information in the database some users are not authorized to access. A DBMS typically includes a security and authorization subsystem that is used by the administrator to create user accounts and specify the restrictions that apply to the account. The DBMS automatically enforces these restrictions and ensures that information access privileges are given only to the authorized users.

3.4 Logging in Databases

This section gives a conceptual description of the concept of logging in database systems. It is important to note that some of the concepts discussed may be subject to the logging preference enabled on any particular system.

Database systems usually maintain a log (also called transaction log[3]) in order to enable recovery from failures that affect transactions and keep track of operations that change

[3]A transaction log may also be called a different name depending on the database system.

the value of database items. Apart from the transaction log, a database may also contain other types of logs that are used to store information that may be useful for recovery from failures [51]. A log is a sequence of log records, which is kept on disk so that it is not affected by any type of failure or other physical problems such as fire, theft, or overwrites [75].

Typically, and for performance reasons, the last part of a log is first held in a main memory buffer until the log buffer is filled or some other conditions occur. The buffer is then added to the end of the log file. The log file is periodically backed up to archival storage or may be handled according to the logging options enabled on a system. In the event of an extensive damage to the database or some other physical failures, the recovery method restores as much information as possible from the archival storage and reapplies operations of committed transaction from the log.

Log records or entries are often dependent on the logging preferences enabled on a system. However, the various types of entries that can be stored include [75]:

- Start record: A log record $[T_i, start]$ is used to show that a transaction T_i has been started.
- Update record: The log record $[T_i, X, V_o, V_n]$ is used to show that a transaction T_i has performed an update operation on the data item X, with the old value being V_o and new value being V_n.
- Read record: A log record $[T_i, X]$ is used to show that a transaction T_i has read the data item X from the database.
- Commit record: A log record $[T_i, commit]$ is used to show that a transaction T_i has been committed.
- Abort Record: The log record $[T_i, abort]$ is used to show that a transaction T_i has been aborted, and needs to be rolled back.

Two types of log entries known as the *undo-type log entry* and the *redo-type log entry* are

stored whenever a transaction needs to update the value of a database item. The undo-type log entry includes the copy of the data item before the update is performed and is used to undo the effect of an operation on the database when necessary. The redo-type log entry includes the copy of the data item after the update has been performed and is used to redo the effect of a database operation that was not recorded in the database before a system failure occurs [75]. A technique known as *Write-Ahead Logging* (WAL) is often used to ensure that all log records are forced-write to disk or a stable storage before any changes are made to the database. This ensures that a consistent version of the database can be recovered by examining the log and using the appropriate recovery technique (Section 3.2.5), should a system failure occur.

Another type of entry that may be found in a log is known as a *checkpoint*. When a database system is restarted after a failure, the entire log needs to be scanned in order to know what operations should be undone or redone to get the database into a consistent state. Checkpointing is an approach that is used to reduce the time overhead involved with scanning the logs before determining what needs to be done. A checkpoint is a point of synchronization between the database and the log. It is used to find a point that is sufficiently far back enough in the log to ensure that any item written to the log before that time has been correctly stored in the database [122]. The checkpoint approach is an additional component of the logging scheme that may be used to limit the volume of log records and as such reduce the amount of searching to be carried out on the log. A checkpoint is periodically written to the log, typically when all modified buffers are written to the database. This implies that all the transactions that have their commit records in the log before a checkpoint entry is written do not require that their update records be redone during recovery since all the updates would have been stored to the database during checkpointing. The recovery manager of a DBMS is responsible for deciding the intervals at which checkpoints should occur [51].

Despite the importance of logs in ensuring that a database can be recovered, a disadvantage of logs in database systems involves the overhead incurred from writing to and

reading from logs to maintain database consistency. There is usually a trade-off between the volume of log records and the performance of the database. This may be seen as the major reason why many database systems allow various levels of logging and provide different log preservation options. Unless necessary precautions are taken, a log file may grow indefinitely, making it difficult to handle the log or even causing the system to run out of space in extreme cases. *Log rotation* techniques are often used to put a bound on how large a log file may become. It involves a regular (usually daily or weekly) moving of an existing log file into a different file (with a timestamp or some ordering) and opening of a new log file. Some of the techniques used for log rotation include:

- Circular logging: works by overwriting the oldest log file once the specified size of the log is reached. Logs in these cases are usually used and retained only to the point of ensuring the integrity of the current transactions [72].
- Archive logging: involves the archiving of logs when they become full, without overwriting any of the logs.

Similar to the concept of log rotation for limiting log sizes, techniques such *incremental backup* (storage of only the data items that have changed since the last full backup of a database) [135] are also used to reduce the time and space required for database backups.

It is important to note that the level of logging enabled on a system may have some implications. An implication on the amount of information available in the logs and how logs are preserved for future use is that it determines how much of the database and how far back in time the database can be recovered when necessary.

3.5 Database Models

As mentioned earlier, a database model (or a data model) is a set of logical concepts used to describe how data is represented, used and manipulated. And depending on the concept used, data models can be categorized into three types, namely, high-level or conceptual data models, implementation or representational data models, and low-level

or physical data models. Although the representational data models hide some of the information about data storage from the user, they can be implemented directly on a computer system and as such they are most frequently used in all traditional commercial DBMSs [75]. In the history of database design, the most often used database models (representational data models) are the hierarchical data model, network data model, and the relational data model. Object-based data models have also recently become popular. A brief description of each of these models is given below.

- **Hierarchical Data Model**: The hierarchical model is the oldest data model, developed by IBM in 1968 [75]. In this model, data is organized in a tree-like structure where each entity has only one parent but can have several children (or dependents). A database based on the hierarchical data model consists of records that are connected through links that associate two or more records. The top of the tree consists of a single record called the *root node* and which has no parent.

- **Network Data Model**: The network model was proposed by the Data Base Task Group (DBTG) of the Conference on Data Systems Language (CODASYL) in 1969 [46, 129]. In the network model, the entities are in the form of graphs, where the links represent an association between precisely two records [75] and there is no hierarchy. Each record represents a node in the model.

  Although the hierarchical and the network data models have their unique advantages, there are several issues that limit their usage. One of these is the complexity of implementing the system. It is also difficult to make changes to the database system without a substantial redesigning effort since the physical links between the records are hard-coded into the data structure.

- **Relational Data Model**: The relational model, which was developed by E. F. Codd [38], does not require physical links between records as in the hierarchical and network models. In the relational model, the user view of the database is simplified through the organization of data into two-dimensional tables called *relations*. Each row of a table (also called a tuple) represents an entity in the table while each

column (or field) represents an attribute of the entities. The relationship between any two relations is shown through a common attribute in the tables. The data type describing the type of values that can appear in each column is represented by a domain of possible values [75].

Over the years, the relational database model has become more programmer friendly, popular and dominant both in academia and in the industry. Most of the leading database systems, including Oracle, Sybase, DB2, Postgres,MySQL and Microsoft SQL Server are based on the relational database model. Some of the advantages of the model include [122]:

1. Simplicity: this allows designers to focus on the logical view of the database and not the details of the physical storage.
2. Structural independence: the model does not rely on physical links and as such changes to the database structure do not affect data access.
3. Ease of design, implementation, maintenance and use of relational database systems.
4. Powerful, flexible and easy-to-use query facility through the use of the Structured Query language (SQL).

Due to the high relevance of the relational database model, this thesis focuses on relational databases. A detailed discussion of relational databases and the relational algebra is given in Section 3.6.

- **Object-based Data Model**: In recent years, the object-oriented programming paradigm has been applied to database systems. The model enables the flexibility of data structuring capabilities and the explicit specification of integrity constraints. Two new data models that have been developed based on the object-oriented paradigm are the object-oriented data model and the object-relational data model. The object-oriented data model extends the concepts of object-oriented programming language [48] with persistence, versioning, concurrency control and other database capabilities [75]. The object-relational data model extends

the relational data model by combining features of the relational data model with that of the object-oriented data model. Although the object-based data model is capable of improving the productivity of database application developers and also data access performance, there is still no precise definition of what constitutes an object-oriented DBMS because some of the available products and prototypes differ significantly from one another. The model is also difficult to maintain [122].

- **NoSQL and NewSQL Databases**: The advances in web technology and the proliferation of devices that are connected to the Internet have resulted in the generation of a massive amount of data that need to be stored and processed. This poses a challenge for the relational database management system and have led to the development of new database technologies such as NoSQL and NewSQL [67]. These technologies provide data processing alternatives that can handle the huge volume of data and also provide scalability. As the name implies, NoSQL (Not Only SQL) technology does not make use of SQL and is able to handle unstructured data unlike relational databases [82]. The NewSQL technology is aimed at bringing the relational data model into a NoSQL environment.

  Although these technologies are relatively new, they are gradually becoming the main data store for large enterprises. However, one of the issues with the use of these technologies is the growing number of different systems that are based on the technology and the large amount of discrepancies among them. This makes it difficult to formulate a perspective on the domain or select an appropriate system for a particular situation [67].

3.6 Relational Data Model and Relational Algebra

Based on the importance and the widespread use of the relational data model in most of the current and popular database systems, this thesis focuses on the concept of reconstruction in relational databases. In this section, a detailed description of the relational database model as well as the concept of *relational algebra* is given. This section lays a

foundation for the remaining parts of the thesis. Also, due to the mathematical nature of the relational model and relational algebra upon which the model is based, different mathematical notation such as $\subset, \subseteq, \in, \notin, \exists, \nexists$ and $\forall$ will be used in the remaining parts of the thesis. The usual meanings of the notation are implied throughout the thesis.

3.6.1 Relational Data Model

The relational data model was introduced by E. F. Codd of IBM Research in 1970 [37] and is characterized by its simplicity and mathematical foundation. The model originated from the concept of mathematical relation [52] and is composed of only one type of compound data known as a *relation*. It also has its theoretical basis in set theory and first order predicate logic [51]. In the relational data model, a database is represented as a collection of relations (also called tables), where each row is called a tuple and each column header is referred to as an attribute, and contains values drawn from a particular *domain*. A domain is used to define the data type of an attribute. The number of tuples in a relation is called its *cardinality* while the number of attributes is called the *degree*.

In formal terms, a domain can be defined as a set of atomic values, which are indivisible to the relational model. For a given n number of domains $D = (D_1, D_2, D_3, \ldots, D_n)$, a relation R consists of an un-ordered set of tuples with attributes $A = (A_1, A_2, A_3, \ldots, A_n)$, where each value of A_i is drawn from the elements of the corresponding domain D_i, such that, $A_1 \in D_1, A_2 \in D_2, \ldots, A_n \in D_n$ or is a special NULL value. The Relation R (also expressed as $R(A)$) is a subset of the Cartesian product (denoted as $\times$) of the domains that define R [38]. That is,

$$R(A) \subseteq (D(A_1) \times D(A_2) \times \ldots D(A_n)).$$

Characteristics of a Relation

It is important to note that a relation differs from a table and from a file because of certain characteristics of a relation. These include:

1. No two tuples in a relation are identical: since a relation is a set of tuples, two

tuples with the same values for all the attributes cannot exist in a relation.

2. Ordering of tuples in a relation: since a set has no particular ordering of its elements, tuples in a relation have no particular order. Although tuples can be ordered by the values of one or more attributes, the information in a relation remains the same regardless of the ordering.

3. Ordering of values in a tuple: Each tuple in a relation is an ordered set of attributes values $(v_1, v_2, v_3, \ldots, v_n)$ belonging to the domain $D_1, D_2, D_3, \ldots, D_n$, respectively. As such, the order in which the values of a tuple appear is significant.

4. Values and Nulls in a tuple: since a domain is a set of atomic values, each tuple in a relation contains a single (non-composite) value for each of its attributes. In addition, the value of an attribute can be denoted as *null* if the value of the attribute does not exist or the value is unknown.

3.6.2 Relational Algebra

The Structured Query Language (SQL) has become the universal and standard language for relational databases and is supported by nearly every DBMS [103]. SQL may be considered as one of the reasons for the commercial success of relational databases as it allows the migration of database applications across different types of relational DBMSs. It is a non-procedural language that can be used for defining the structure of data, modifying the data in a database and specifying constraints. The first version of SQL (also known as SEQUEL) was developed by Donald D. Chamberlin and Raymond F. Boyce as part of the SYSTEM R project at IBM in the 1970s [31, 6] and has since been standardized and expanded to include more features [30].

Although SQL provides an interactive query interface that allows users to execute complex queries, the user is only able to specify what the result should be and the decision on how to execute the query is left solely to the DBMS. The formal language associated with the relational model, which can be used to specify basic retrieval requests is known

as the *Relational Algebra* [75]. It is an abstract language with operations for manipulating and accessing relations. Relational Algebra (sometimes referred to as RA in the thesis) consists of a set of relational operators that are of importance in understanding the types of requests that may be specified on a relational database [51]. In relational algebra, a query is written as a sequence of operations that generates the required result when executed. It can be shown that SQL queries can be expressed in relational algebra notation as relation databases make use of the relational algebra for internal representation of queries for query optimization and execution [66, 69]. This thesis exploits the expressiveness and mathematical nature of the relational algebra as well as the ability to transform SQL queries into relational algebra operations.

The relational algebra operators[4] include 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 referred to as a relation-valued expression (*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 relational algebra operators

Operators	**Notation**
Projection (π)	$T \leftarrow R[A_1, A_2, A_3]$ $T \leftarrow \pi_{A_1,A_2,A_3}(R)$
Selection (σ)	$T \leftarrow R[p(A)]$ $T \leftarrow \sigma_{p(A)}(R)$
Join ($\bowtie$)	$T \leftarrow R[p(A,B)]S$ $T \leftarrow R \bowtie_{p(A,B)} S$
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 ($/$)	$T \leftarrow R[A, B/C]S$
Rename (ρ)	$R \leftarrow \rho_{A_i=B_j}(R)$

Table 3.1: Basic Operators of the Relational Algebra.

as defined by Codd [38] and the notation often used for each operator are shown table

[4]This section of the thesis is an excerpt from previously published papers [56, 53].

3.1, 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. A description of each of the relational algebra operator as well as the corresponding expressions and SQL queries are described below. These definitions are widely used in the rest of the thesis.

Project operator (π): The project operator takes a single relation as its operand and generates a relation containing columns of the operand which are listed as part of the command. As with all relational operators, the operator removes re-occurrence of any tuple in the relation generated. The notation used for the operator is as follows:

$$T \leftarrow R[A_1, A_2, A_3] \text{ or } T \leftarrow \pi_{A_1,A_2,A_3}(R)$$

In SQL: SELECT A_1, A_2, A_3 FROM R

Select operator (σ): The select operator employs a single Relation R as its operand and generates a relation that contains some of the rows in the operand so that such rows satisfy the condition specified by a logical predicate p on the attribute(s). The notation used for the select operator is as follows:

$$T \leftarrow R[p(A)] \text{ or } T \leftarrow \sigma_{p(A)}(R)$$

In SQL: SELECT * FROM R WHERE $p(A)$

Join operator ($\bowtie$): The join operator takes two relations $R(A)$ and $S(B)$ as its operands and generates a relation containing rows of one operand concatenated with rows of the second operand but only where the condition specified by the logical predicate $p(A, B)$ is found to hold true. The notation used for the operator is as follows:

$$T \leftarrow R[p(A, B)]S \text{ or } T \leftarrow R \bowtie_{p(A,B)} S$$

In SQL: SELECT * FROM (R JOIN S ON $p(A, B)$)

It is important to note that the join operator described above refers to the inner join. However, there are several variations of the join operation that can be performed. These

include *natural join*, *left outer join*, *right outer join*, *full outer join* and *semi join*. Although each of the join operators have specifics ways in which the rows are concatenated, the notation used are the same (but with the appropriate join type specified).

Cartesian product ($\times$): The Cartesian product is a special case of the join operator where the predicate is simply *true*. That is, each row of the operand $R(A)$ is concatenated with each row of $S(B)$ without any condition specified. The operator is expressed as follows:

$$T \leftarrow R \times S$$

In SQL: SELECT * FROM R, S

Union operator ($\cup$): The union operator takes two relations, R and S that are union-compatible (that is, they have the same number of columns and the i^{th} attribute in both relations draw values from the same domain) as its operands and generates a relation containing all the rows of S together with all the rows of T but with duplicate rows removed. The notation used is as follows:

$$T \leftarrow R \cup S$$

In SQL: (SELECT * FROM R) UNION (SELECT * FROM S)

Intersection operator ($\cap$): The intersection operator takes two relations, R and S that are union-compatible as input and produces a relation containing only those rows of R that also appear as rows of T. The notation used for the operator is as follows:

$$T \leftarrow R \cap S$$

In SQL: (SELECT * FROM R) INTERSECT (SELECT * FROM S)

Difference operator ($-$): The difference operator takes two relations R and S that are union-compatible and generates a relation containing only those rows of R that do not appear as rows of S. The operator is expressed as:

$$T \leftarrow R - S$$

In SQL: (SELECT * FROM R) EXCEPT (SELECT * FROM S)

Division operator (/): The relational division operator works in a similar way to arithmetic division with a divisor, dividend, quotient and remainder except that instead of numbers, these operands are all relations. Suppose column B of a Relation R is divided by column C of a Relation S and column A of Relation R is the source of values for the quotient, then the quotient T is written as[5]:

$$T \leftarrow R[A, B/C]S$$

Rename operator (ρ): The operator renames an existing domain, relation or the column of an existing relation. Both the old name and the new name are supplied as part of the command. All references to the old name are updated to the specified new name. The operator is expressed as follows:

$$R \leftarrow \rho_{A_i=B_j}(R)$$

In SQL: ALTER R RENAME A_i TO B_j

3.7 Summary

This chapter explains the underlying concepts of database systems and describes the architecture of database management systems. The characteristics that unify different database systems and which make data management efficient in databases are also described. The three-schema architecture of database systems is described in conjunction with how the architecture assists in achieving data independence. In order to lay a foundation for subsequent chapters of the thesis and differentiate between the concept of reconstruction and database backup or recovery, a brief description of database transactions as well as the techniques used for backup and recovery is given. The ACID properties expected of transactions are also explained. Since a major part of this thesis deals with the applications of log files in databases (as will be explained in the next chapter), the concepts of logging in databases, the different types of log entries, log rotation techniques and the importance of logs in databases are also discussed. Although

[5]There is no DIVIDE operator defined in SQL commands, but the operator can be expressed using a combination of SQL equivalents of other operators.

this thesis relates mainly to the relational database model, due to its high relevance and popularity, a description of the different data models is given. The relational data model and the relational algebra are also explained in detail. A description of the relational algebra operators and the notation used for each is given as these are of importance throughout the rest of the thesis.

Chapter 4

Concepts of Database Forensics

> "Who art you, O great mountain? Before Zerubbabel you shall become a plain!"
>
> - Zechariah 4:7

In the previous chapter, the concepts of database systems were described. This chapter builds on the two previous chapters and explains the need for the forensic analysis of databases. The chapter describes the concepts of database forensics since the process of reconstruction in databases is really a subset of the steps involved in the forensics analysis of a database. A description of the processes, tools and methods involved in database forensics is also given. The first contribution of this thesis to the field of digital forensics, which is the definition of the dimensions of reconstruction and/or research in database forensics, is discussed in this chapter. The chapter builds on a paper that was previously published in [55].

In Section 4.1, an introduction of the database forensics research field as well as its importance is given. In Section 4.2, the dimensions in which research in database forensic has been focused as well as the relationship of these dimensions and the research that has been done in each of the dimensions are discussed. In Section 4.3 the database forensics analysis process and the need for an effective reconstruction process is discussed. Section 4.4 describes some of the tools that are currently being used in database forensic analysis. Some of the challenges involved in database forensics research and practice are discussed in Section 4.5.

4.1 Database Forensics

Over the years, database systems have become a core component of many computing systems in our society and are used to store critical and sensitive information relating to an organization or her clients. Unfortunately, the increase in the usage of databases to store volumes of information has led to an increase in the number of attacks directed towards databases and the rate at which databases are manipulated to facilitate crimes [55]. Several security breaches have been reported in the recent years, many of which involve the theft of personal data or financial information stored within a database. The fact that many organizations combine several databases onto fewer database servers in order to cut costs and simplify their management [59] also compounds this problem and presents databases as prime targets for attackers. However, even when a database is not the target of an attack, information relevant to an investigation are usually found in databases and can be retrieved to facilitate such investigations. As such, databases have become of interest in finding artifacts that may assist in solving various kinds of investigations [55].

Examples of organizations that have been targets of some of the largest security breaches in history include CardSystems and TJ Maxx. CardSystems experienced a security breach that allowed the exposure of 200,000 credit card numbers and the TJ Maxx breach led to the exposure of over 45 million credit and debit card numbers [59]. These incidents reflect the enormous amount of information that may be lost from a single attack on a database system. Another example is where an unauthorized user (or an authorized user with illegal motives) gains access to an organization's database and modifies their production database server (for example, by changing order dates, delivery address or production prices), resulting in erroneous product shipment and financial loss to the company [58]. It is also possible that a database was not manipulated to commit a crime but the database contains information that may be used to resolve the crime. For example, in a situation where every access into a building is stored in a database, investigating a murder case in which there was no break-in may require querying the

database to know who was in the building at the time. A forensic analysis of the database may also be required if the database was manipulated to hide certain information.

Database forensics is an emerging branch of digital forensics that deals with the identification, preservation, analysis and presentation of evidence from databases [59]. It involves the application of digital forensics [20, 116, 107] techniques in gathering evidence admissible in a court of law from databases.

Although digital forensics has grown over the past decade to an essential part of many investigations [63], the same cannot be said of database forensics. Despite the large amount of research that has been done on digital forensics and other related fields such as database systems and database security, very little has been done on database forensics [100] even though investigations involving databases have been explored in theory and practice. Due to this lack of research, traditional digital investigations often excluded databases despite that evidence can usually be found in them. Research in the database forensics field only started taking root in the last three to five years. Although the field is still in its early years, it is quickly becoming an important part of many investigations due to the increased volume of information that may be helpful in solving different crimes and the large number of risks associated with the information stored on many databases. In addition, the increasing number of crimes involving databases and the ongoing changes in cybercrime and digital investigations now require digital forensics practitioners to augment traditional forensic skills with database forensics techniques in order to enhance digital investigations. Some of the advantages of incorporating database forensics techniques into digital investigations include the following [59]:

- Prove or disprove the occurrence of a data security breach.
- Retrace queries executed on a database.
- Identify data pre- and post-transactions.
- Reconstruct previously deleted database data.

Of major importance in database forensics is the ability to retrace the operations performed on a database and reconstruct previously deleted or modified information in the database, and this is the main focus of this thesis. This requirement affects how data is collected and analyzed during the forensic analysis of a database, thus making it necessary that certain guidelines are followed during the process. In order to explain the processes, tools and techniques involved in the forensic analysis of a database, the next section considers some of the research that has been done on database forensics and presents the dimensions that exist in database forensic investigations.

4.2 Dimensions of Database Forensics

Although different aspects of database forensics have been explored by researchers over the past few years, research work have taken directions that define various dimensions of reconstruction and investigation in database forensics. The various research seem divergent, often focusing on aspects that may only be relevant in a fraction of database forensics investigations. An analysis of these dimensions shows the research that has been done in relation to each dimension and reflects some of the aspects of database forensics that is yet to receive any research attention. This section presents the idea that the apparently diverging strands of research in database forensics are, in fact related. And much more, that they form different dimensions of a problem space, where most of the work done has focused on only a specific dimension of the space [55].

Considering most of the research work that have been done in database forensics, three main dimensions of databases that can be investigated emerge. These are:

1. Compromised databases.
2. Damaged or Destroyed databases.
3. Modified databases.

In the following sections, these dimensions are considered in more detail and previous database forensics research (by different authors) are positioned into a dimension.

4.2.1 Compromised Databases

A compromised database can be defined as a database in which some of the metadata or some software of the DBMS have been modified by an attacker even though the database is still operational. It is a known fact that the result of a database query is a function of both the metadata of the database and the data stored in the database. As such, a major concern of database forensics in this situation is that an investigator cannot trust the information provided by the database being investigated [55]. For example, a criminal may corrupt a database or hide certain data from the user by storing the data in a column (of a relational database) with a column name that does not reflect its purpose or even store the data in several relations with foreign keys that link to them. A view (which is stored as metadata) may enable the criminal to have an easy access to the data. However, once the metadata is deleted, the logical relationship between the data is lost and the data may not be retrievable even though the raw data may still exist [100].

Olivier [100] pointed out that although a database itself seems to be the best tool for collecting data for a forensic analysis, the integrity of the data contained or the results obtained from queries cannot be trusted since the database might have been coerced into giving false information, for example, if the data dictionary has been destroyed or modified to yield query results that are different from the correct ones. This problem is somewhat similar to the problem faced during a *live* investigation of an operating system [24, 125], in which the data apparently collected from the system may not be the actual data on it, but what a rootkit or other malware [76, 81] on the system wants to be seen. In the case of a compromised database, the retrieved data is imbued by the metadata. Unfortunately a *dead* analysis [24] does not solve the problem in this case because it is likely that the same metadata will still be used to interpret the raw data collected from the database [100].

Kornbrust [81] identified the similarities in the architecture of database systems and operating systems, and posits that a database can also be affected by a malware (like

viruses or rootkits) in the same way as an operating system. He illustrated how an Oracle database can be attacked by a rootkit. The presence of a database user, as well as the jobs and processes executed by the user can be hidden by implementing a rootkit on the system. As such, to ensure that the evidence collection process is not hampered by a rootkit, it is important to ensure that the data model has not been compromised.

The problem with investigating compromised databases was also identified by Litchfield [87] while discussing steps to perform in a live response to attack on an Oracle database. He pointed out that even though the knee-jerk reaction in the event of an incident is to pull the plug or disconnect a system from the network in order to avoid additional incursions or data theft, this may lead to the loss of useful evidence such as volatile, in-memory data [87]. For example, an attacker that compromises an Oracle database with administrator privileges could create a trigger that clears the audit trail (or do much worse), making it difficult for the investigator to perform an analysis.

In other recent work by Beyers et al. [13, 14], the authors also identify that the metadata affects the view of data and as such presents complications during the forensic analysis of a database. In their paper [13], the authors describe four abstract layers of a DBMS: the data model layer, the data dictionary layer, the application schema layer, and the application data layer; which separates the various levels of DBMS metadata and raw data. Various techniques that can be used in the collection phase of the database forensics process when one or more of the abstract layers have been compromised are also described. In a subsequent paper [14], the authors describe the notion of a *clean* data model environment and a *found* data model environment. This differentiation is necessary since the data model can be considered as a code or underlying software that controls the metadata and the data, as a rootkit [76] or as a toolkit during a forensic analysis. Various techniques on how to achieve different states of data model environment was described by the authors.

It is important to note that the major decision that has to be made when investigating compromised databases is whether to use the metadata as it exists on the database or

to try and get a clean copy of the DBMS [13]. This makes it important that the DBMS used in the investigation of this category of databases is specified by the investigator.

4.2.2 Damaged/Destroyed Databases

Another dimension of database forensics deals with the investigation of damaged or destroyed database. This category of databases refers to databases where the data contained or other data files have been modified, deleted or copied from their original (or expected) locations into other places. These databases may or may no longer be operational depending on the extent of the damage done. Most of the research that have been done in database forensics falls into this dimension of database forensics. A practical example of a damaged database is a situation where an attacker deletes or moves information from a particular location in the database in order to hide traces of what he had done.

The series of papers by Litchfield [84, 85, 86, 87, 88, 89, 90] seem to be the most detailed and practical resource available on database forensics in this category. The papers focus on the forensic analysis of Oracle databases (specifically, Oracle 10g Release 2 server running on Windows). The first two papers [84, 85] address how information or potential evidence can be retrieved from the redo logs and how dropped objects can be located. The author described the contents of the redo logs by dissecting the log into different sections. He also demonstrated how the data manipulation language (DML) operations performed on a database can be determined by reading the binary format of the redo log. Litchfield [85] also described the structure of Oracle data blocks and discussed how dropped tables, functions and deleted tuples can be located. Anti-forensics techniques that can be used by an attacker to cover his footprints are also mentioned in the papers [84, 85].

In the subsequent series of papers [86, 88, 89, 90], Litchfield discussed various ways of finding evidence of attacks on the authentication system, evidence of data theft, or other evidence in different places in an Oracle database. He described how failed login attempts

usually stored in the audit trails as well as records in the TNS Listener's log file of Oracle databases can be used in a forensic analysis. In the absence of auditing, other sources of evidence in the database server and web server's log file are also identified. Other work that has been done on the forensic analysis of Oracle databases includes that of Wright [139, 138]. He explains technical methods that can be used to identify when the data in an Oracle database has been modified and how to recognize vulnerabilities in the database. In addition, Wright [138] also investigates the possibility of using Oracle LogMiner as a forensic tool. However, even though many of these work describe practical techniques that can be employed in the forensic analysis of an Oracle database, they do not represent a generic underlying model that can serve as a basis for database forensics in general.

Another database that has been considered in this dimension of database forensics is the SQL Server. In his book on SQL Server Forensic Analysis, Fowler [59] discusses techniques that can be used for the collection and preservation of database artifacts, and how they can be of benefit during an investigation. Specific methods that can be used to configure an SQL Server in a state of preparedness for forensic analysis as well as ways of verifying security incidents and analyzing the artifacts collected from a SQL Server database are also described. In addition, the author discusses the effects that rootkits can have on the collection or analysis of data and describes ways of detecting and handling database rootkits during a SQL Server forensic analysis.

Other research that fall into this dimension of database forensics deals with the detection of database tamper, and data hiding in a database. Snodgrass et al. [123] proposed a technique which relies on cryptographically strong one-way hash functions for detecting when the data in a database has been tampered with. The idea presented in the paper was extended to deal with the forensic analysis of a data security breach in a subsequent paper [111], where the notion of corruption diagrams was introduced as a way of visualizing attacks and forensic analysis algorithms. In [112], another algorithm for detecting tampering and determining what and when a database was tampered with was

also presented. Although the work of Stahlberg et al. [124] and that of Pieterse and Olivier [113] seem to be anti-forensics, they both expose the areas of database systems that should be checked for previously deleted data or any form of information hiding during the forensic analysis of a database.

In order to investigate a damaged database, it may be necessary to employ techniques for reconstructing files in which the metadata have been lost or damaged (a process referred to as *file carving* [62]). This may assist in regenerating damaged or destroyed underlying files in a database that may be helpful in retrieving data of interest during a forensic analysis.

4.2.3 Modified Databases

In contrast to compromised and damaged or destroyed databases, a modified database is a database where the data, metadata and other data files of the systems have not been compromised or damaged, but which has undergone changes due to normal business processes since the event of interest occurred. For example, proving a shop attendant's claim that he sold a good at the price on the database on a particular day involves getting values from a modified database. This dimension of database forensics deals mostly with the investigation of databases that are not directly involved in the incident being investigated but is used to store information that may assist in resolving the incident [55]. Research in this dimension are of importance because it is possible that a database is modified prior to or after a damage or compromise is done. This thesis, as well as most of the work that have been published during the completion of the thesis [56, 53, 2] (details of which are in subsequent chapters of the thesis) fall into this dimension of database forensics.

Other research that have been done in this category includes the work of Frühwirt et al. [60, 61]. The authors describe the file format of MySQL database with InnoDB storage engine and give practical examples of how to reconstruct and interpret the data in the file system. The content of the redo logs, often used for crash recovery in InnoDB database

is also explored for the forensic analysis of the database. The authors also describe and demonstrate how data manipulation queries executed on an InnoDB database can be reconstructed from the log files during an investigation.

Although there are various locations where forensic data or artifacts can be found on databases (as will be shown in Section 4.3), most of the work that have been done in all the dimensions of database forensics, especially on modified databases rely on the use of log files found in databases. As discussed in Section 3.4, log files serve as a rich source of information that can be used to gain knowledge of changes made to a database and the queries executed on the database. Also, the use of log files in database forensics is one of the unifying elements for the three dimensions of database forensics described earlier [3]. Another aspect of the database forensics dimensions that is worth mentioning is the seemingly orthogonal nature of the three dimensions; this is explained in detail in the following section.

4.2.4 Orthogonality of the Dimensions

At the moment, most of the research in database forensics treats the dimensions discussed above as being orthogonal (in mathematics terminology) or independent of one another[1]. Figure 4.1 shows the positioning of most of the previous research in database forensics into a single dimension of the database forensics problem space.

However, even though it is possible for research to be directed in a single dimension, this is not always the case in practice. A database being investigated may belong to one or dimensions of database forensics in varying degrees. For example, an investigation may be positioned at point A on the problem space as shown in figure 4.1, such that the database involved in the forensic analysis has been compromised at A_1 degrees, damaged at A_2 degrees and modified at A_3 degrees. Thus, it is possible that a database was compromised by modifying the metadata and some of the data in the database was later destroyed in order to hide traces of the compromise. Also, it is possible that a damaged database is

[1]This section of the thesis has been published in [55].

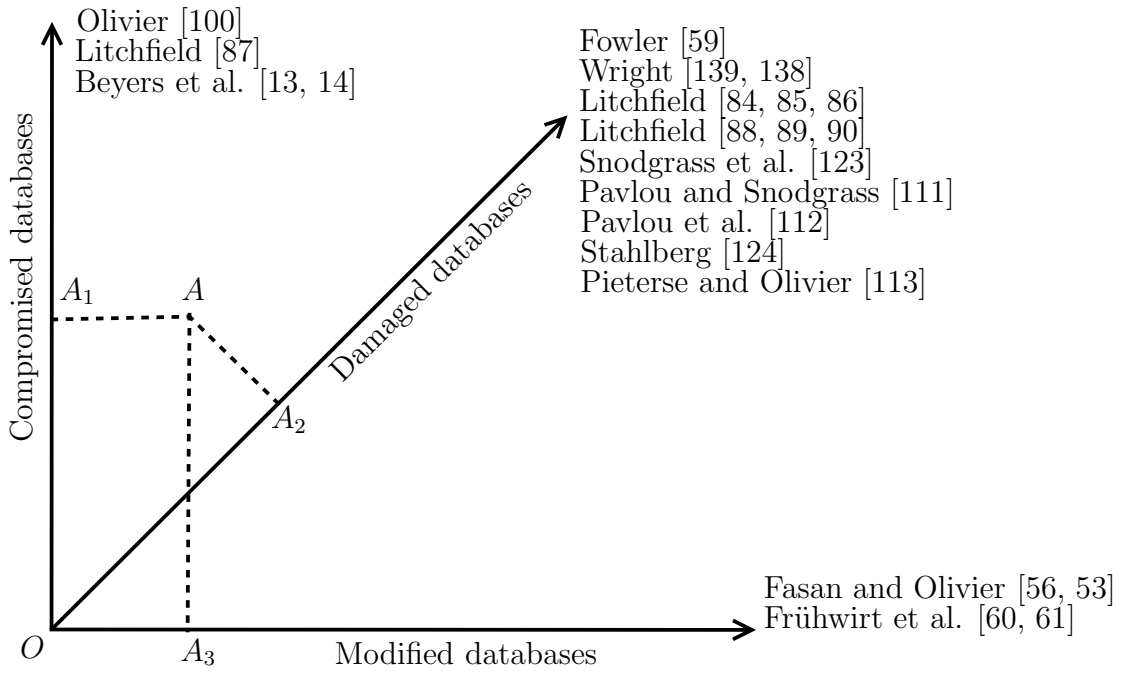

Figure 4.1: Dimensions of Database Forensics.

still operational and the damage is only discovered after much normal processing. The database forensics problem space is at least three dimensional, where any given forensic investigation can be positioned somewhere in this three dimensional space. This poses challenges both for database forensics research and practice. In practice, heuristics that can be used to place an investigation into a position in this space may be required. The implication of a research conducted in one dimension may also have to be considered in the other dimensions. In addition, it also raises questions about whether or not it is possible to quantify the degree of each dimension that occurs during the forensic analysis of a database and the order or approach to be followed in investigating each of the dimensions involved in an investigation.

In deciding the technique to be used during an investigation involving more than one dimension of database forensics, questions relating to the trustworthiness of the database, the authenticity of the data it contains, or if there is any known modification of the database should first be asked. The investigation of each dimension involved should be conducted putting into account the degree of that dimension in the analysis. For example,

if an investigation is place at the origin (point O in figure 4.1) of the three dimensions, that is where the database has not been compromised, damaged or modified, then the required data can be collected from the database by simply querying the database. However, the same approach will not be applicable if the database has been compromised, damaged and/or modified. In addition, since there is no defined method for identifying a compromised database, it may be necessary to assume that an investigation is at the origin of the dimensions when conducting the analysis of time or mission critical database systems. Educated guesses can then be made from the set of possibly contradicting information gathered based on this initial assumption.

The decision regarding the dimensions that will be considered in a database forensics investigation, as well as the degree or extent of each dimension determines the tools that should be used, as well as the process that should be followed during a database forensics analysis. An overview of the techniques, processes and tools that have been employed in database forensics analysis is given in the sections 4.3 and 4.4.

4.3 Database Forensics Process

Although research in digital forensics has led to the development of various techniques and process models, many of these techniques are not completely transferable to database forensics due to some of the unique characteristics of database systems which requires the techniques to be adapted for database forensics [55]. Given the limited amount of research in database forensics, it is necessary to find a foundation on which the techniques and processes of database forensics may be built. In this section, the similarities and differences of database forensics and file system forensics are explored in order to derive some of the principles that apply to database forensics. In addition, some of the steps involved in the general digital forensics process are considered from a database perspective. The steps that should be involved in a database forensics investigation process are also proposed.

4.3.1 Database Forensics and File System Forensics

One method of exploring database forensics is to consider the similarities and differences between file systems and database systems, and then build on the principles that apply to file system forensics in order to derive principles for database forensics [100]. This is because, even though both fields are applied in different environments, they both focus on the retrieval of stored data. As such, since the field of file system forensics is well developed and often used in practice, some of the approaches from this field may be transferable to database forensics.

File systems provide a method for the storage and retrieval of data from a computer system. In many operating systems, data is usually stored on a computer in a hierarchy of files and directories (files that contain other files), with the metadata. A file system makes use of the files and directories to organize data in a way that enables the computer to know where to find them. Although the attributes of a files vary depending on the operating system, various attributes such as file name, file type, location, size, access information, MAC time and user information are usually associated with each file [21, 18]. This information is commonly referred to as the metadata. The metadata is used to map files to their physical location on disk because it provides information on how to locate and assemble a file or directory. According to Olivier [100], some of the aspects of files systems that are of relevance from a forensics perspective include:

- The contents of the file [18];
- Metadata about a file [18, 50];
- File carving techniques [62]; and
- Recovery of data or files stored in slack space or on free parts of the disk [50].

On the other hand, as discussed in Chapter 3, databases provide a structured way of handling collections of data. The metadata in a database describes the structure of the database as well as the structure and relationship of the data it contains. One of the main similarities of database forensics and file system forensics is that both the data in

the database and the data describing the data (metadata) are required to recover lost or deleted information, just as it is also the case in file system forensics. Techniques for reconstructing files when the metadata has been lost or damaged (file carving) will also be required for damaged databases.

However, the differences arise from the fact that database systems have a significantly more complex structure than file systems. The complexity of databases is described in more detail by Olivier [100]. Another important difference in database forensics and file system forensics is the high level of logging that occurs in databases, in contrast to file systems. As described in Section 3.4, database logs often contain enough information that can be used to roll back or forward a database in order to get the database into a consistent state. As such, log files play a very important role in the forensic analysis of a database, especially in the reconstruction of deleted data.

The fundamental elements that may be derived from file system forensics process deals with integrity, searchability, and restoration of files, as well as the extraction of metadata and attribution [100]. These elements can be combined with the characteristics of database systems in order to gain more insight into database forensics. The elements are considered below, in relation to database systems.

Searchability: In digital forensics analysis process, one of the important phases deals with the location of evidence among the volumes of data collected from the computer. This is usually done by compiling data from different sources and using different search techniques to locate data items that might be of interest in an investigation. In database forensics, the database itself seems to be the best forensic tool for searching for evidence because it allows an investigator to perform a search using powerful queries on the database [100].

Integrity: In file system forensics, data is usually collected from a computer through a process known as *imaging*, where a copy of the disk is made. Various techniques to ensure the copy is provably an exact copy of the original are often employed.

However, in the case of database forensics, even if an exact copy of the database is retrieved, the database may still be forced into giving false result, depending on whether or not the data dictionary has been modified [100]. This gives rise to the question of whether the data model on a database system should be used as it is found or whether a "clean" data model should be used. This concern is discussed in more detail in [13, 100].

Metadata extraction: As mentioned earlier, a file is usually associated with metadata including details such as the file name, MAC time, and size. The metadata is useful during a forensic analysis as it provides more information about the file. According to Olivier [100], it is of forensic use in database forensics to indicate that a particular version of the data model, data dictionary, application schema, application data or segment of the metadata as they existed at some point in time is to be used in the analysis. He also suggested notation that can be used to differentiate different versions of the metadata.

Reconstruction: Similar to the concept of file carving in file system forensics, different forms of carving can be achieved in database forensics. For example, tuples, tables, and database schema can be recovered from the data collected during the analysis of a database. In this thesis, the process of carving data is referred to as *reconstruction*. Apart from the application data, data from the lower levels of a database architecture may also be reconstructed or guessed from available or other reconstructed information. For example, if a deleted table can be entirely reconstructed, then the schema of the table may be derived. However, it may be necessary to differentiate between the reconstructed and the actual data during an investigation. This is because, if a table is carved or reconstructed, it is possible that not all records with a similar structure necessarily formed part of the table and one might need to use other hints - such as links in other tables, to confirm whether records do or do not belong to the table. Also from the perspective of

admissibility, hypothesis may be tested by reconstructing a scenario with a hypothesis and comparing the reconstructed instance with the original instance in order to check the validity of the hypothesis.

Attribution: Attribution refers to the task of determining who is responsible for certain effects in a file (or other evidence), or who caused it to have certain characteristics [80, 33]. In file system forensics, attribution may be based on the textual feature of a file or on the metadata. In the context of database forensics, attribution may be addressed two ways. One is to make use of the database logs, if available; while the other way is to make use of the metadata to determine who has the privilege to perform certain actions. Although it is possible that a perpetrator might have modified the logs, or bypassed the logging facility or authorization system, a combination of the two methods may be useful in correlating findings in an investigation [100].

This thesis relates directly to the reconstruction of data during the forensic analysis of a modified database, where the data of interest might have occurred at some earlier time prior to various modifications and/or deletion of the data in the database. Issues relating to the integrity of collected data, searchability, metadata extraction and attribution are not discussed in detail even though they may be mentioned where necessary. An overview of the database forensics investigation process and the steps involved in the analysis of a database are discussed in the following sections.

4.3.2 Database Forensics Investigation Process

One of the pointers to the need for further research in database forensics is the fact that there is currently no defined underlying process model for database forensics. The available methods are focused on a few specific DBMSs. Wong and Edwards [137] presented a patent method for the forensic analysis of an Oracle database. The method consists of generic steps that a forensic investigator may try to follow to discover more information about an operation that was performed on a database [100]. Another methodology

focused on a damaged SQL Server database was presented by Fowler [59]. The methodology consists of four major steps: investigation preparedness, incident verification, artifact collection and artifact analysis.

Although the exact steps to be taken during a database forensic analysis will depend on the specific situation and the DBMS being investigated, the database forensics investigation process should in general include the following steps [55].

- Determining whether the database has been compromised, damaged or modified, or if an investigation involves more than one dimension.
- Determining which acquisition method is most applicable to the situation.
- Collection of volatile artifacts.
- Collection of non-volatile artifacts.
- Preservation and authentication of collected artifacts.
- Analysis of collected data and determination of the intruders activities.
- Reconstruction of required data.

4.3.3 Database Forensics Analysis Techniques

As with digital forensics analysis in general, database forensics analysis can involve live or dead analysis techniques [24]. A live analysis involves the use of the resources and software of the database system. Considering the fact that databases are usually a critical component of many systems and organizations, a live analysis can provide a means of investigating the database without disrupting all activities or causing a significant loss to the organization. In addition, it allows a large amount of information to be identified, stored, or manipulated using the database itself. Live analysis can be used to retrieve both volatile and non-volatile data (that would be lost if the database was switched off) from the database. However, an important point to note in a live investigation of a database is that any action performed changes the state of the database and may lead to a potential loss of evidence in the logs or system memory. The risk of the system data

being altered due to the presence of a rootkit on a database system is another issue in the live analysis of a database as this may cause the database to give false answers to queries.

On the other hand, a dead analysis of a database requires that the database system is inactive. This may be done by abruptly switching the system off (or pulling the plug) in order to prevent the execution of system and application shutdown routines [59]. However, a dead analysis results in the loss of volatile and in-memory data which may contain evidence. Unlike other branches of digital forensics where a dead analysis may be used to prevent the interference of rootkits or malware, it does not solve the problem in database forensics because the same database metadata may still be used to interpret the raw data collected. Also, depending on the logging technique used in a database system, it is possible that data modifications recorded in the log have not yet been written to disk. An abrupt shutdown of the database may cause the image collected not to reflect the true state of data during an analysis. Although a dead analysis may allow the reproduction of results obtained during an analysis, it may introduce several complexities in the forensic analysis of a database and as such, it is not considered to be the best approach for database forensics [59, 87].

Another analysis technique for database forensics, known as the *hybrid* analysis was described by Fowler [59]. The hybrid analysis involves the key elements of both the live and the dead analysis techniques. Hybrid analysis can be viewed as a typical dead analysis which is performed after the live acquisition of volatile data and selected non-volatile data. One advantage of this analysis technique is that it allows an investigator to align the ratio of live versus dead acquisition to specific needs in an investigation.

Although an investigator may decide to use any of the analysis techniques, it is important that the benefits of using any of the techniques are weighed with the possibility of losing potential evidence if the technique is used. This will assist in determining which technique is most appropriate for a given investigation. Regardless of the technique used, it is necessary to ensure that evidence is preserved and data is not unintentionally altered

or destroyed.

4.3.4 Preservation, Collection and Analysis of Artifacts

The objective of preservation in database forensics is to reduce the amount of evidence that may be overwritten. Data must be collected in a forensically sound manner that generates a correct image of database data and extreme care must be taken to guarantee that actions performed do not unintentionally alter data [55]. This may be achieved by using a MD5 algorithm (or any other trusted digital algorithm) to ensure that the image corresponds to the original copy of data. A write blocker may also be used to prevent modifying data accidentally. Although data can be acquired from a modified database by querying the database, the execution of any query that can delete the information in the database must be avoided. In the case of a compromised or damaged database, no SQL statements should be executed as this may modify the data stored in memory, cause splits in the internal data pages, or lead to the storage of new data in the caches, and thus complicating the investigation process [59]. As with other branches of digital forensics, the preservation of data is an important aspect of database forensics because it may be necessary of prove the integrity of the data used as evidence in a court of law, if required.

Although the logs often provide a vast amount of information for database forensics, forensic data also exists in several places in a database. It is important to know which data is important for an investigation and to prioritize evidence collection due to the volatility of some data. Apart from using the database itself as a source of data in an investigation, data can also be found in tools used by the database system; for example in the execution plan cache. An execution plan cache is a documentation of the most efficient way of executing data requests issued by database users and are stored in the plan cache for possible reuse. The information in the plan cache can be used to identify database misuse by an insider or data damage in a database [59]. Other sources of forensic data in a database consist of files that are used to store histories relating to the

database. Some of these file (for example, log files and data files) may be specifically reserved for the database while others (such as web server logs or system event logs of the operating system) are not explicitly reserved for database server usage. Records of logins to a database will also be useful in identifying patterns that may be useful. It is important that the level of volatility of a file is considered in deciding which information should be collected first during an investigation. A summary of the volatility level of various areas in which data can be found in a database is given by Fowler [59].

The analysis of artifacts (or collected data) involves the consolidation and evaluation of the data. The specific steps taken in this phase may depend on the type of data, the specific DBMS and the specific situation being investigated [55]. The analysis should take into account the dimensions involved in an investigation and where relevant information can be found. Another important criterion for the analysis phase in database forensics is that previously deleted or modified data should be reconstructed where necessary and the actions performed by an intruder must be determined. Events such as login attempts and irregular database activities should be identified and included in the investigation timeline as this may assist in recognizing patterns and identifying activities that may not be logged sequentially in the database log file.

4.4 Database Forensics Tools

The decision regarding the dimensions involved in a database forensic analysis, the degree of the dimension(s), and the analysis technique to be used plays an important role in deciding the tool to use during an analysis. However, the current unavailability of enough tools for database forensics also reflects the need for research in the field [55]. Database forensics tools should allow information from different sources in a database to be gathered and consolidated to provide more insight into an investigation. It should assist the investigation process and the recovery of information regardless of the dimensions involved in the investigation. In addition, a database forensics tool should be capable of recognizing the dimensions of database forensics involved in any investigation.

Currently, database forensics is done with inbuilt database tools that are not specifically designed for database forensic analysis and with the database itself (by executing queries to retrieve data). As such, most of the tools being used are only applicable to specific DBMSs and are focused only on damaged databases. Although these tools have been helpful in collecting data and identifying fraudulent activities, they are not accurate or precise enough to be used as a forensic tool. For example, the Oracle LogMiner was investigated for possible use as a forensic tool by Wright [138]. He reported that there are anomalies with the way it works which makes it inadequate for a forensic analysis. Other tools that have been used include audit features such as SQL Server Audit [59, 115] and the Oracle Audit [104, 139].

Although the use of the database itself as a forensic tool allows an investigator to search the database using powerful queries, the database cannot be used as a tool in the investigation of a damaged or compromised database. As mentioned earlier, one is immediately faced with questions regarding the integrity of the database since the data is clouded by the metadata [100]. In addition, the query processor on many databases may optimize queries in ways that cannot be controlled by the user. An investigator has to be certain that an optimized query is an exact representation of the original query, especially in cases where changes might have been made to the metadata by an intruder or the presence of a database rootkit is suspected. These constraints restrict the use of databases as a forensic tool for itself in many cases [55].

As the field of database forensics develops further, it is hoped that more tools that are specifically designed for database forensics will be developed. For example, Litchfield [136] announced that he is in the process of developing an open source tool called Forensic Examiners Database Scalpel (FEDS) for database forensics a few years ago. One of the main concerns that may be faced in the development of tools includes the fact that the data model is typically hardcoded into the DBMS. Since such a data model that can be used as a forensic tool for itself does not currently exist, new models will have to be created [100]. Further research is still required in order to develop tools that can be

used for database forensics. The development of these tools calls for an understanding of databases as well as the processes involved in database forensics analysis [55].

4.5 Challenges in Database Forensics

Sections 4.3 and 4.4 have described some of the pointers to the need for extensive scientific research on many aspects of database forensics. Although the field is slowly becoming popularly, the limited amount of research can be attributed to the different challenges involved in database forensics investigations. Olivier [100] identified three facets in which a database has to be considered during a forensic analysis. This inherent multidimensional structure and complexity of databases, which is not yet completely understood in a forensic sense is a major contribution to the lack of research in the field.

In addition, there are various challenges that are encountered in the data collection and analysis stage of database forensics. First, is how to determine whether a database has been compromised, damaged, modified, or if there are more than one dimension involved. There is no heuristic on how to determine the dimensions involved or where to start an investigation in this three dimensional space. Moreover, deciding about the degree of the dimensions involved in an investigation as well as the most appropriate data acquisition method to use in investigations with more than one dimension is another challenge since no research that offers any guideline has been done. Similar to other branches of digital forensics, the volume of data that can be collected from a database is another issue. An examiner must determine which data are pertinent to an investigation and the order in which to collect them. The process of eliminating some data sources to reduce the volume of data poses challenges such as misinterpretation or over-interpretation of data. Coupled with the fact that there are usually different file formats for files in various databases, some of which may be encrypted, this results in the potential dismissal of valuable content [42].

Another challenge in database forensics deals with the reconstruction of data. Since reconstruction may involve the restoration of data from proprietary formats, collaboration

between researchers and vendors is also required in order to have a good knowledge of the proprietary formats in individual DBMSs and assist in the development of database forensics tools.

4.6 Summary

This chapter focuses on the concepts, processes, tools and challenges involved in database forensics. The need for database forensics analysis as well as the benefits of incorporating database forensics into traditional digital investigation process are highlighted in this chapter. One of the contributions of this thesis, which is the definition and description of the three-dimensional nature of the database forensics field, is also discussed in detail. Moreover, the notion that database forensics investigations may involve more than one of these dimensions in varying degrees is emphasized.

In order to present a detailed overview of the database forensics process, the concepts involved in database systems are compared to those of file systems since both fields deal with the retrieval of stored data and the use of metadata in the process. Although the fields apply to different environments, the similarities and differences are explored in the discussion of various aspects of database forensics. An explanation of the database forensics process, analysis techniques, and the techniques used in the preservation, collection and analysis of artifacts from databases is given in order to position the specific focus of this thesis, which is the reconstruction of data that might have been modified or deleted at some earlier time of interest in a modified database. An overview of the tools and challenges involved in carrying out the processes involved in database forensics analysis are also described.

The ideas and concepts described up to this point in the thesis lay the necessary foundation for the subsequent chapters and are explored in the remaining parts of this thesis.

Chapter 5

Database Reconstruction Algorithm

> "He reveals the deep and secret things; He knows what is in the darkness, and light dwells with him."
>
> - Daniel 2:22

Although the information currently stored in a database can be determined by simply querying the database, much more effort is required in order to determine the information in a database at an earlier time. As mentioned in Chapter 4, an important aspect of database forensics involves the ability to reverse data manipulation operations and determine the values in a database at an earlier time, prior to various modifications of the data in the database. This process is referred to as reconstruction (of data) in database forensics in this thesis. In this chapter, an algorithm for the reconstruction of data in a database for forensic purposes is presented. The chapter is based on a previously published paper [56]. The first part of the chapter describes the need for reconstruction in database forensics analysis and highlights a possible way of reconstructing data. The notion of *relational algebra log* and *value blocks* are introduced together with the concept of *inverse relational algebra*, as these are of utmost importance in the formalization of the reconstruction algorithm. The database reconstruction algorithm is presented in conjunction with a few examples of its application in typical instances.

In Section 5.1, the need for database forensics reconstruction is described using typical instances that require reconstruction. The approach taken in this thesis to solve the problem is also highlighted. Section 5.2 gives the definition of the inverse operators of

the relational algebra and categorizes the operators into groups. In Section 5.3, the notion of relational algebra log and value blocks are discussed in detail. The steps taken in typical examples of the database reconstruction process are shown in Section 5.4. These examples reflect the ideas formalized into the database reconstruction algorithm presented in Section 5.5.

5.1 Database Forensics Reconstruction

Although various data restoration techniques such as database 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.

The reconstruction of the data in a database is important in digital forensics investigations since databases are usually of interest in many investigations and useful information relevant to an investigation are often found therein. A typical instance is where a database has been manipulated in order to facilitate a suspicious act. It is a known fact that output produced by queries on a database is dependent on the raw data it contains. However, forensic investigations often require finding the data contained in a database at an earlier time. Although the data stored in a database at a point in time can be determined by executing a query on the database (with the appropriate privileges), answers to queries arising from a forensic investigation may require more than just the current instance of the database. This is because the current information in the database may be different from what it was at the time of interest to the investigation, since various modifications or updates of the database might have taken place since then.

An illustration of this fact is a situation where a shop attendant 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 shop. Verifying the shop attendant's claim requires that the selling price of the good at that particular date be determined. However, since various updates of the database might have occurred prior to the investigation date, it is necessary to find ways of reversing queries that have been performed on the database which have affected the price of the good since the particular date of interest. Other examples of questions that call for the ability to reconstruct values in a database at an earlier time during a forensic investigation include the following instances.

- If a table in an organization's database was deleted by a criminal, can it be proved that a customer's record was actually in the deleted table based on previously executed queries involving the table?
- Can it proved that an hospital patient died because the 'Prescribed Drugs'column of the patient's record some weeks before his death is not what it was supposed to be?

As mentioned earlier, the information about queries performed on a database is usually stored in the database log (also referred to as query log or transaction log) of the database[1]. And this information is often used to recover from failures. However, the query log also constitutes a rich source of information that can be used for reconstruction. Although the queries executed on a relational database are usually expressed in SQL notation, the log files may sometimes be stored in the binary format. Various work [84, 85, 60, 61] have been done on how the corresponding SQL queries can be generated in these cases. Thus, the assumption in this thesis is that a log of queries executed in the traditional SQL format is available or can be generated.

In Section 2.4.4, the concept of a retrospective (or backward) reconstruction is described. From a database forensics perspective, retrospective reconstruction can be performed on a database by reversing the queries performed on the database since a particular time of interest. The constraint of increasing time and space complexity pointed out by Cohen [43] when using this reconstruction approach can be overlooked in database

[1]The extent of information logged is usually dependent on the logging preference of the database.

forensics reconstruction because database systems maintain a log of queries executed on the system and queries can be reversed in accordance to the possibilities reflected in the log.

A possible way of reversing queries performed on a database since a particular date is to find the inverse of such queries. In order to do this, it is necessary to convert queries expressed in the SQL notation into a more mathematical notation such as the relational algebra so that inverse functions can be applied. This concept is described in Section 5.3. Although an extensive research and industrial effort has been invested into query processing and database efficiency over the last few decades, very little attention has been given to reverse query processing or finding the inverse of a query despite its several applications. The little amount of work that has been done on reversing of queries [15, 16, 121, 17] focus specifically on the generation of test databases, testing of DBMS performance and debugging of SQL queries. However, even though these techniques can be used to generate good test databases, they cannot be used for forensic purposes as the databases often generated are non-deterministic in nature. That is, it is possible to generate more than one instance of the database with the same set of input and the decision procedure is left for a model checker in order to guess the best result [15]. This chapter focuses on describing how inverse queries can be used for database forensics reconstruction.

In the following sections, the definition of the inverse operators for relational algebra as well as the concept of relational algebra log is introduced. The process of dividing the relational algebra log into value blocks is also discussed. By traversing the relational algebra log and value blocks, and applying the inverse relational algebra, an algorithm that can be used for reconstruction in database forensics is given in Section 5.5.

5.2 Inverse Relational Algebra

This section gives a definition of inverse functions for the operators of the relational algebra [38]. The inverse functions are defined such that they can be used for the

reconstruction [39] of a database during an investigation and determine values in the database at an earlier time. The inverse operators rely on both the queries (from the query log file) executed on the database and the database schema. The goal of the inverse operators is to find the value of an attribute A (for a tuple in Relation R) 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 operators work on the assumption that a complete set of queries that have modified the database from a particular time of interest in an investigation (or earlier) to the present time is available and that the database schema is known (both for the input and the expected output). The operators generate a result which is either a *partial* or a *complete inverse* of the query. More formally, the inverse of a query Q is defined as Q^{-1} such that:

$$Q^{-1}(Q(R_t)) = R_t^*, \tag{5.1}$$

where $R_t^* \subseteq R_t$ and the notation, $\subseteq$ means that R_t^* is contained in R_t. That is, R_t^* may contain some missing tuples or missing values in some columns. In cases where $R_t^* = R_t$, the inverse is referred to as a *complete inverse*. Otherwise, the inverse found is referred to as a *partial inverse*. A partial inverse can either be 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 and partial columns inverse. The definition of the inverse operators for each operator of the relational algebra and a specification of which operators produce a complete, partial tuples or partial columns inverse is given below. Table 5.1 gives a summary of the inverse operators and the type of inverse that can be produced when each is used. A more detailed description of the information provided in the table is given in sections 5.2.1 and 5.2.2.

5.2.1 Complete Inverse Operators

Only two inverse operators of the relational algebra generate outputs that are always a complete inverse. The first operator which is the inverse rename (ρ^{-1}) operator is

Complete Inverses	**Partial Columns Inverses**	**Partial Tuples Inverses**
Inverse rename		
Inverse Cartesian product - Except when one empty relation is involved.		
Inverse projection - If projection was done on all the columns of the operand.	**Inverse projection**	
Inverse selection - If all the tuples of the selection operand satisfied the condition.		**Inverse selection** - If some tuples did not satisfy the selection condition.
Inverse join - If all tuples of the join operands satisfied the join condition. - If the join is a full outer join.		**Inverse join** - If some tuples did not satisfy the join condition.
Inverse intersection - If the operands of the intersection are equal.		**Inverse intersection** - If the operands are not equal.
Inverse divide - If one of the divide operands is known.		**Inverse divide** - Of left operand if both operands are not known.
Inverse union - If one of the union operands is known and both operands had no tuples in common.		**Inverse union** - If one of the union operands is known.
Inverse difference - If one of the difference operands is known. - For either operands if the other is known and the right operand is a subset of the left operand.		**Inverse difference** - Of left operand if both operands are not known.

Table 5.1: Summary of Output Generated by Inverse Operators.

the simplest inverse operator since a rename operation does not change the data in a database in any way but only changes a name. The inverse is found by changing the name of the renamed object to its previous name. That is, if the query $A \leftarrow \rho_{A_1=B_2}(A)$ is performed to change the name of column A_1 in Relation A to B_2, then the inverse of the operation is, $\rho^{-1}(A) = \rho_{B_2=A_1}(A)$.

Another 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 $\times^{-1}(T)$ (that is, R and S) can be completely determined by doing 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 of the inverse Cartesian product is when one of the operands of the Cartesian product was an empty relation. In this case, the second operand cannot be determined from the inverse operation, but this rarely happens in practice.

5.2.2 Partial Inverse Operators

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

Inverse projection (π^{-1})

Given the result R of a projection operation, the inverse projection generates a partial columns inverse. The result is a relation having the expected columns (determined from the schema) but with values in the columns not included in the projection being null. The columns included in the projection contain the data in R. That is, if $R \leftarrow \pi_{A_1,A_2}(S)$, then $\pi^{-1}(R) = S^*$ where both S and S^* have exactly the same columns; values in attributes A_1 and A_2 of both S and S^* are exactly the same and values of other attributes in S^* are null. A complete inverse projection can be found when the projection was done on all the columns of a relation.

Inverse selection (σ^{-1})

The inverse selection generates a partial tuples inverse. It is similar to the inverse projection except that it contains missing tuples instead of columns with missing data. Given 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 if all the tuples in the operand of the selection operator satisfied the condition that was specified.

Inverse join ($\bowtie^{-1}$)

The inverse join is similar to the inverse Cartesian product except that the output generated may contain missing tuples depending on which of the tuples in the operands satisfied the condition specified in the join performed. Similar to the inverse Cartesian product, the output of the inverse join operator is found by doing a projection on the columns of the expected outputs. In general, a complete inverse join can be found in the following cases. Otherwise, the inverse generated is a partial tuple inverse.

1. If all the tuples in the join operands satisfied the join condition.
2. If the join type is a full outer join.
3. The left (or right) operand of the join can be completely determined by the inverse join operator if the join type is a left (or right) outer join.

Inverse intersection ($\cap^{-1}$)

Given a query $T \leftarrow R \cap S$, the inverse intersection generates partial tuples inverses containing all the tuples in T. A complete inverse intersection can be found if and only if R and S are known to be the same, in which case the three relations R, S and T are equal. In addition, if either R or S is known, then from set theory it is known that:

1. if $\exists\ x \in S$ and $x \notin T$ then $x \notin R$ and
2. if $\exists\ x \in R$ and $x \notin T$ then $x \notin S$.

Inverse divide ($/^{-1}$)

Given the quotient Q and the remainder RM of a divide operation ($Q \leftarrow R/S$), the inverse divide operator generates two relations R^* and S^*. The Relation R^* is readily known since all the tuples in RM are also in R^* ($RM \subseteq R^*$). A complete inverse divide can be determined if and only if one of the outputs is known.

1. If R is previously known, then $S = R/T$ and
2. if S is previously known, then $R = (S \times T) \cup RM$.

Inverse union ($\cup^{-1}$)

The inverse 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 $R^* = T - S$. On the other hand, if R is known, then $S^* = T - R$. A complete inverse union is found only when both R and S have no tuples in common. The trivial case of the inverse union is when T contains no tuples which implies both R and S also contain no tuples. In cases where neither R nor S is known, working with the knowledge from set theory may be useful in order to determine at least a partial inverse. That is;

1. if $\exists\ x \in T$ then $x \in R$ or $x \in S$.
2. If $\exists\ x \in T$ and $x \notin R$ ($x \notin S$) then $x \in S$ ($x \in R$).

Inverse difference ($-^{-1}$)

Given a difference operation $T \leftarrow R - S$, the left operand of the operation is readily determined by the inverse difference operator as $R^* = T$ since $T \subseteq R$. A complete R can be determined only if the Relation S is known and all the tuples in S are also known to be in R (that is, $S \subseteq R$) 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 it is known that $S \subseteq R$, then a complete Relation S is found from the inverse as $S = R - T$.

5.3 Relational Algebra Log and Value Blocks

As a definition, a relational algebra log (also referred to as RA log) is a log of queries expressed as operations involving relational algebra operators instead of the traditional SQL notation. The concept of converting a query expressed in SQL into a relational algebra expression is an important aspect of query processing, optimization and execution [66, 69, 51] which has been explored over the years. Relational databases use relational algebra for internal representation of queries for query optimization and execution. Although SQL is the query language that is used in most relational databases [51], a SQL query first has to be converted into its relational algebra equivalent in order for it to be optimized and executed. One of the properties of relational algebra that is usually explored in this regard involves the knowledge of various transformation and equivalence rules that allow relational algebra expressions to be transformed into equivalent ones. As such, the operators of the relational algebra can be used independently, that is, one or more operators can be used to express another operation. For example,

$$R \cap S = R - (R - S)$$
$$\pi_A(R \cup S) = (\pi_A(R) \cup (\pi_A(S)))$$
$$R \bowtie_{p(A,B)} S = \sigma_{p(A,B)}(R \times S)$$

This research exploits these characteristics of the relational algebra by expressing the query log in a database as a sequence of relational operations The resulting log is referred to as a Relational Algebra Log (RA Log).

The use of the RA log allows us to easily determine when a relation has changed. In relational algebra, 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 motivation for the use of 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, making the log file more readable. In addition, a typical `select` statement in a 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 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 the same. Value blocks are named based on the relation that remains the same in the block and subscripts are used to signify which block occurs first. 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} 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}. Typically, 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 usually associated with each query is preserved in the RA log in order to group the value blocks into appropriate sequences. An example of an RA log divided into value blocks is shown in figure 5.1. The SQL equivalent of the RA log is listed in appendix A.

Using the notion of value blocks, and RA log and the inverse relational algebra operators, Section 5.4 gives typical examples of how the value in a database at an earlier time can be reconstructed.

5.4 Concept of Database Reconstruction

In this section, typical examples of the steps involved in reconstructing the information in a database are given. Although the main focus is to determine specific values in a table at some earlier time, the concept can be applied to generate tables in a database. Figure 5.1 shows an example of a relational algebra log generated from the complete query log of a typical database by transforming queries into operations involving relational 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 $t_1, t_2, \ldots$ represent the time stamps at which a particular query is executed. Considering this RA log, some of the questions that can be asked as part of a forensic investigation include the following. The reconstruction of the desired values in the different cases are discussed thereafter.

- Case 1: Was a particular value present in column D_1 of table D at time t_8?
- Case 2: Is the claim that a value is in column H_2 of table H at time t_{13} true?
- Case 3: Is the value in column J_1 of Relation J the same as expected at time t_{15}?

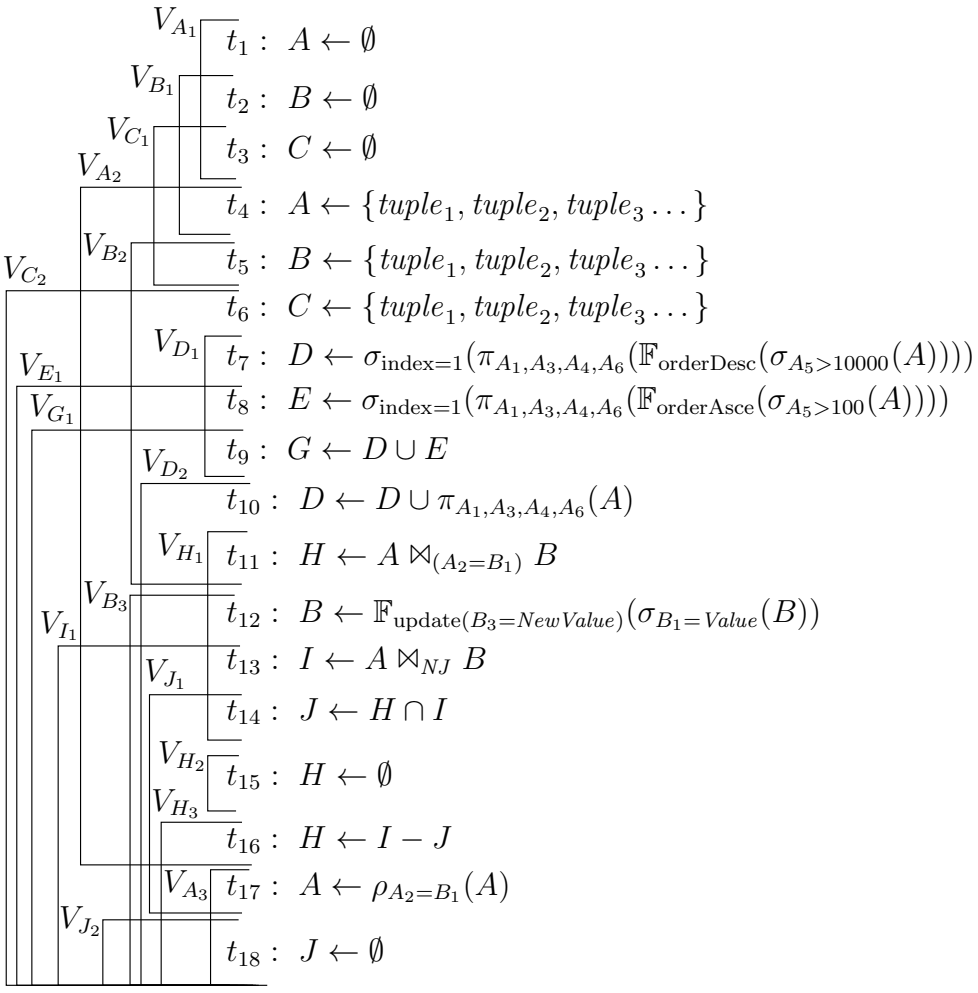

Figure 5.1: A Relational Algebra Log Grouped into Value Blocks.

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 which were in column D_1 at time t_8 and check for the desired value. From the RA log in figure 5.1, 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 the value block V_{D_1}, then the check for the desired value can be performed. This can be achieved by finding the inverse of the union operation between relations D and E at time t_9. Since the second operand of the union (E) as well as its result (G) have not been changed since t_9, a partial or complete tuple inverse D (depending on if D and E had any tuple in common) can be found as:

$$\cup^{-1}(G) = (D^*,\ E) \text{ where } D^* = G - E \tag{5.2}$$

An easier alternative to the reconstruction of values in D_1 is to perform the actual query which resulted in D at time t_7. This requires that the inverse of operations that has changed the data in Relation A between t_7 and the current time be found. Since the queries at $t_7, t_8 \ldots t_{16}$ are in the same value block of A, that is V_{A_2}, the Relation A is only modified at t_{17}. The inverse of the rename operation at t_{17} ($A \leftarrow \rho_{A_2=B_1}(A)$) is found by simply changing the name of column B_1 to its previous name, A_2, as shown in equation 5.3. The query at t_7 can then be performed again to generate the complete Relation D.

$$\rho^{-1}_{A_2=B_1}(A) = \rho_{B_1=A_2}(A) = A \tag{5.3}$$

Since t_7 is in V_{D_1}, another alternative to reconstructing the values in D is to find the inverse of the first query in value block V_{D_2} (that is, $D \leftarrow D \cup \pi_{A_1,A_3,A_4,A_6}(A)$). This is done in the same way as in equation 5.2, that is:

$$\cup^{-1}(D) = (D^*,\ \pi_{A_1,A_3,A_4,A_6}(A)) \text{ where } D^* = D - \pi_{A_1,A_3,A_4,A_6}(A) \tag{5.4}$$

Even though the Relation A has been updated at t_{17}, the update does not affect the projection involved in equation 5.4 since the renamed column is not projected. And 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 various ways may either yield the same outputs or some outputs may be more complete than others. In cases where the various approaches to the reconstruction of a value generate minimal tuples (or columns), the union of the different approaches should be taken in order to generate a relation with more data.

Case 2

To determine whether a value was in column H_2 of Relation H at time t_{13} (in value block V_{H_1}), both value blocks V_{H_1} and V_{H_2} need to be checked. It is obvious that the inverse of the first line of V_{H_2} cannot be found since the data in H was deleted at this point (t_{15}). From value block V_{H_1}, there are two alternatives for the reconstruction of column H_2 of Relation H. An inverse of the intersection operation at t_{14} will generate a partial tuples H^* containing all the data in Relation J. Unfortunately, the data in J have been deleted at t_{18}. Thus, the only feasible option to reconstruct a value in H is to redo the query which 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 found. The Relation B can also be determined by finding the inverse of the natural join operation at t_{13}.

The inverse of the rename operation at t_{17} is found as shown in equation 5.3. To determine B using the inverse of the update at t_{12}, the values in column B_3 of B need to be replaced 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 generated from the inverse of the update operation contains partial tuples of B with missing values in column B_3. That is, the inverse of the query at t_{12} is given as:

$$\mathbb{F}^{-1}_{\text{update}(B_3=NewValue)}(\sigma^{-1}_{B_1=Value}(B)) = \mathbb{F}_{\text{update}(B_3=null)}(B^*) = B^* \tag{5.5}$$

The Relation B, with partial tuples, can also be found by taking the inverse of the natural join operation at t_{13}. Since both relations I and A (from equation 5.3) are known, B

can be found as:

$$\bowtie_{NJ}^{-1}(I) = (A,\ B^*) \text{ where } B^* = \pi_{B_1,B_2 \ldots B_n}(I) \tag{5.6}$$

As mentioned earlier, the union of the two relations generated from equations 5.5 and 5.6 can be taken to generate a more complete B^*. For example, the actual values of the nulls in column B_3 of B^* generated from equation 5.5 may be determined from those of B^* generated from equation 5.6. 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 find H^* most likely containing 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 when reconstruction is done. 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 it is actually not in the relation of interest (H).

Case 3

The data in column J_1 of Relation J at time t_{15} can be reconstructed in two ways. Since the difference operation $(H \leftarrow I - J)$ at t_{16} is in the same J value block as at time t_{15} and both relations H and I are known, the easier way to reconstruct J (with partial tuples) is to find the inverse of the difference operation. That is:

$$-^{-1}(H) = (J^*,\ I) \text{ where } J^* = I - H.$$

Alternatively, J can be reconstructed by executing the query at t_{14} again. This requires that the relations H and I be reconstructed in the associated value blocks, V_{H_1} and V_{I_1}. The reconstruction of the data in Relation H in value block V_{H_1} is exactly the same situation as in case 2 above. The reconstruction of values in I in value block V_{I_1} is done

by finding the inverse of the rename operation on A (equation 5.3) and executing the query at t_{13} again. With at least some tuples in the reconstructed relations H and I, the Relation J (probably with partial tuples) can be generated by executing the query at t_{13} again. The values in column J_1 can then be checked to determine if it contains the expected values.

5.5 Database Reconstruction Algorithm

In Section 5.4, typical examples of how values in a database can be reconstructed have been given. This section generalizes the reconstruction process by providing an algorithm that can be used to reconstruct values in a database. The algorithm, defined as SOLVE(Relation D, value block V_{D_i}, RA log log, Set S) 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 as Relation RD.

```
01: INVERSE(Relation D, RA Query V_Di[1]) {
02: OUTPUT: Inverse of the assignment into D from query q
03: Let q = the query at V_Di[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_Ai
11:     if (op = ∩): T = D; return T;
12:     if ((op = ∪) and (∃ V_Ai+1)): T = ∅; return T;
13:     else:
14:       A ← SOLVE(A, V_Ai, log, S);
15:       T = op^-1(D)|A; return T;
16:   }
17: }
```

Figure 5.2: The INVERSE(Relation D, RA Query $V_{D_i}[1]$) Function.

The `SOLVE` function makes use of the `INVERSE` function (shown in figure 5.2) 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 D in its previous value block ($V_{D_{i-1}}$). Calls to the `INVERSE` function occur only when a value block $V_{D_{i-1}}$ exists and its output depends on the operation performed by the query. Since updates on a relation involves taking the relation itself as an operand, except when the tuples in the relation are deleted ($D \leftarrow \emptyset$), the `INVERSE` function considers the different instances that may occur in a query. The inverse of a query involving the deletion of all the tuples in a relation cannot be determined and the function simply returns an empty Relation D^* in this case (line 6). This situation also applies to a case in which a constant assignment was made into a relation. For example, $D \leftarrow M$, where M is a relation. However, this case is not included in the `INVERSE` function since it does not apply to any typical SQL command. In all other cases of a query, the inverse of the query is found based on the inverse operators of the relational algebra defined in Section 5.2. The notation $\text{op}^{-1}(D)|A$ (in line 15 of the `INVERSE` function) means the inverse of an operation with two operands, of which one (that is, A) is known.

The `SOLVE` function (shown in figure 5.3) starts 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 will be stored. The parameter S of the `SOLVE` function is empty the first time the function is called and it stores tuples of relation and value block (with the corresponding result) that have already been considered in a reconstruction process 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 `SOLVE` are stored as a tuple in S with an associated reconstructed relation RD, which is initially empty. The algorithm then considers the various possible combinations of D in a query and outlines the steps to be followed in

```
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;
```

Figure 5.3: The SOLVE(Relation D, Value Block V_{D_i}, RA Log log, Set S) 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;
```

Figure 5.3: The SOLVE(Relation D, Value Block V_{D_i}, RA Log log, Set S) Function (contd.).

reconstructing D from the different approaches listed in Q. In all the approaches, the first check is to confirm whether there is a value block of D after the one in which it is to be reconstructed (that is, $V_{D_{i+1}}$). If there is none, it implies that D has not been modified and the current Relation D is returned as in lines 9, 17, 26, 45 and 57 of figure 5.3. If D has been modified, the algorithm considers the different ways in which D can be reconstructed based on the operation in which it was involved and stores the reconstructed Relation D in the set R. For example, if an operation ($D \leftarrow D$ op A) is in set Q, and it is confirmed that there is a subsequent value block of D (in line 27). The Relation D can be determined by reconstructing it in the subsequent value block, $V_{D_{i+1}}$ and finding the inverse of the operation leading to the generation of this value block. The resulting Relation T is a possible reconstruction of D in V_{D_i} (lines 28 – 29). Alternatively, D can be reconstructed by checking if the Relation A has also changed. If A has not been modified, then D can be found by reconstructing the relation in its earlier value block ($V_{D_{i-1}}$) and performing the query being considered again (lines 31 – 33). In a case where both relations D and A have been modified (from line 34), D can be reconstructed by finding D in its earlier value block $V_{D_{i-1}}$ and also reconstructing A either from its current value block (as in line 36 of figure 5.3) or from its subsequent one (as in line 41). The query being considered can then be performed again (as in lines 37 or 42 respectively). Other queries in the set Q are examined in a similar way based on the suitable case in the `SOLVE` function and all possible reconstructions of D are included in the set R. Once all the queries in Q have been considered, the union of all the possible reconstructions is stored as the reconstructed Relation RD and returned as the final result.

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 done to determine or check a particular value in or claim about 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 can then be terminated when the value of interest or desired information

has been determined.

In very few cases, it is possible that the reconstruction process results in an empty relation. For example, this might happen when all the tuples in a relation were deleted before the relation was used in any way. However, in general, the algorithm presented above can be used in reconstructing values in a relation especially for forensic purposes. The typical examples discussed in Section 5.4 are solved using the reconstruction algorithm. It can be proven that tuples generated in a reconstructed relation are indeed in the relation and that the algorithm terminates and does not result in an infinite loop. These proofs are presented in Chapter 6 of the thesis.

5.6 Summary

This chapter describes some of the main contributions of this thesis. It presents the concepts involved in database forensic reconstruction and an algorithm that can be used for the reconstruction. The chapter describes the purpose of reconstruction in database forensics analysis and highlights instances where the reconstruction of the values stored in a database is required. In order to describe the reconstruction algorithm, the concepts of inverse relational algebra, Relational algebra (RA) log, and value blocks are presented. A definition of the inverse operators of the relational algebra is given together with the conditions in which complete or partial inverses can be found. The concept of translating entries (SQL queries) in a log file into the equivalent relational algebra notation (called an RA log) is also presented. The process of dividing the RA log into value blocks is also described in detail. Lastly, the chapter combines the concepts described therein in the formalization of the database reconstruction algorithm presented. Typical examples of situations involving the reconstruction of values in a database at an earlier time using the database reconstruction algorithm are also shown. The reconstruction algorithm works on the assumption that the database being investigated works correctly and has not been compromised or damaged in any way (that is, the modified databases category). It is also assumed that the query log is maintained in a certain order.

The remaining chapters of this thesis build on the ideas presented in this chapter, particularly the database reconstruction algorithm. For example, as mentioned earlier, it can be proved that the reconstruction algorithm is correct and that it always terminates. This proof is presented in Chapter 6 of the thesis. The following chapters also describe various aspects of the reconstruction algorithm for database forensics.

Chapter 6

Correctness Proof of Algorithm

> "Correct your son, and he will give you rest; yes, he will give delight to your soul."
>
> - Proverb 29:17

In Chapter 5, one of the main contributions of this thesis, an algorithm for reconstruction in database forensics, is presented. The reconstruction algorithm can be applied for the regeneration of data that may be used as evidence or in support of findings in a database forensics investigation. However, since legal requirements that apply to the presentation of evidence in a legal proceeding require that tools and techniques used must be proven to be reliable [94], this chapter builds on the previous chapter by presenting a correctness proof of the algorithm. It gives an assurance that the values generated using the reconstruction algorithm are indeed contained in an earlier instance of the database. The chapter is based on a previously published paper [53].

The correctness proof consists of the proof of partial correctness as well as the total correctness [44] of the algorithm. This is done by first showing that the output generated (which is a reconstructed relation) when the algorithm terminates is correct. Correctness in this sense means that the output is at least a subset of the original relation desired. We then prove that the algorithm always terminates regardless of whether there is a recursive call to the function or not.

In Section 6.1 the partial correctness is described by showing that if the algorithm terminates then the output generated is indeed correct. Section 6.2 presents the total

correctness proof by showing that the algorithm always terminates regardless of the recursive calls to the `SOLVE` function. The correctness proofs are established through a sequence of lemmas leading to each conclusion.

6.1 Partial Correctness

The correctness of the output of the reconstruction algorithm is proved by using the principle of mathematical induction [52] on the entries of the RA log. To establish the proof, the following lemmas showing the correctness proof for each of the inverse operators are presented. In all the notation used, Relation D is the relation to be reconstructed. And the inverse operations refer to the definitions in Section 5.2.

Lemma 6.1.1. *Any query used as parameter of the* `INVERSE` *function is of the form* $D \leftarrow rve$ *and the Relation* D *is a parameter of the operation in the rve (relation-valued expression).*

Proof. We know that any update on a relation always includes the relation as a parameter when expressed in relational algebra. That is, an update on a Relation D can only be of the forms: $D \leftarrow \text{op } D$, $D \leftarrow D \text{ op } A$ or $D \leftarrow A \text{ op } D$, where A is another relation. The only exception to this is when a relation is created for the first time, in which case, operations on any operand(s) can be assigned into the relation. We also know that the `INVERSE` function in the reconstruction algorithm always takes a query which is the first line of a value block of the relation to be reconstructed as one of its parameters and the function is called only when the relation exists in an earlier value block. Thus, since the first query in a value block is an update of the associated relation, it implies that any query used as parameter of the `INVERSE` function always contain an assignment operation into the relation to be reconstructed and the relation is also an operand of the operation performed in the query. That is, to find the inverse of the first query in a value block of Relation D, the query used as a parameter of the `INVERSE` function can only be of the forms, $D \leftarrow \text{op } D$, $D \leftarrow D \text{ op } A$ or $D \leftarrow A \text{ op } D$. This justifies the different cases considered in proving the correctness of each of the `INVERSE` operators in the subsequent

lemmas in this chapter. □

Lemma 6.1.2. *The* `INVERSE` *function yields a correct output for queries with the rename operation as its parameter.*

Proof. Given a query involving a rename operation as a parameter in the `INVERSE` function, from lemma 6.1.1 we know that the query is of the form $D \leftarrow \rho_{A_i=B_j}(D)$ where A_i is an attribute of D. The inverse returned from the `INVERSE` function $(\rho^{-1}(D))$ is defined as $\rho^{-1}(D) = \rho_{B_j=A_i}(D)$ from Section 5.2. This implies that the previous name of the renamed attribute is replaced and thus the relation generated is the same as before the rename operation was performed since the operation did not modify the data in the table.

More formally, in equation 5.1, we have defined the inverse of a query Q as Q^{-1} such that $Q^{-1}(Q(R_t)) = R_t^*$, where R_t and R_t^* are relations and $R_t^* \subseteq R_t$. Since the inverse rename operation usually generates a complete inverse, we now show below that $R_t^* = R_t$ when Q is a rename operation. That is, $Q^{-1}(Q(D)) = D$. If we assume that the original Relation D (before the rename operation) has n attributes given as $A = \{A_1, A_2, \ldots, A_i, \ldots, A_n\}$ and D' is the resulting relation after the rename operation, then we have:

$$\begin{aligned} Q^{-1}(Q(D)) &= \rho_{B_j=A_i}(\rho_{A_i=B_j}(D)) \\ &= \rho_{B_j=A_i}(D') \text{ where } A = \{A_1, A_2, \ldots, B_j, \ldots A_n\} \\ &= D'' \text{ where } A = \{A_1, A_2, \ldots, A_i, \ldots, A_n\} \end{aligned}$$

Since the data in D'' has not been modified in any way and it has the same attributes $(A = \{A_1, A_2, \ldots, A_i, \ldots, A_n\})$ as D, it implies that $D'' = D$. □

Lemma 6.1.3. *The* `INVERSE` *function yields a correct output for queries with the selection operation as its parameter.*

Proof. If the query parameter of the `INVERSE` function involves a selection operation, from lemma 6.1.1, the query is of the form $D \leftarrow \sigma_{p(A)}(D)$. Since a selection operation

generates a relation that contains some of the rows in its operand satisfying the condition specified as $p(A)$, it implies that the result is a subset of the operand. Also, since the output of the inverse selection is defined as the result of the selection performed (in Section 5.2), it implies that the output is a correct partial inverse. Formally, we want to show that

$$Q^{-1}(Q(D)) = D^* \text{ where } D^* \subseteq D. \tag{6.1}$$

If D' is the resulting relation from the selection operation, we have,

$$\begin{aligned} Q^{-1}(Q(D)) &= \sigma^{-1}_{p(A)}(\sigma_{p(A)}(D)) \\ &= \sigma^{-1}_{p(A)}(D') \text{ where } D' \subseteq D \\ &= D' \end{aligned}$$

Since $D' \subseteq D$ and D' is unique (since we always get the same result from the same query), it implies that equation 6.1 is true. □

Lemma 6.1.4. *The* `INVERSE` *function yields a correct output for queries with the intersection operation as its parameter.*

Proof. Given a query with an intersection operation as a parameter of the `INVERSE` function, the query is of the form $D \leftarrow D \cap A$ or $D \leftarrow A \cap D$ (from lemma 6.1.1). We know that all the tuples in the resulting D are also in the initial D. Since we have defined the relation generated from an inverse intersection operation to contain all the tuples in the result of the actual intersection, it implies that the output generated from the `INVERSE` function when a query parameter involves an intersection operation is correct. We now show that $Q^{-1}(Q(D)) = D^*$, where $D^* \subseteq D$. If we denote the resulting D of the intersection operation as D', then we have;

$$\begin{aligned} Q^{-1}(Q(D)) &= \cap^{-1}(D \cap A) \\ &= \cap^{-1}(D') \text{ where } D' \subseteq D \\ &= D' \end{aligned}$$

Since $D' \subseteq D$, it implies that the output of the `INVERSE` function is correct when the query parameter is an intersection operation. □

Lemma 6.1.5. *The* `INVERSE` *function yields a correct output for queries with the union operation as its parameter.*

Proof. From lemma 6.1.1, we know that any query used as a parameter of the `INVERSE` function, and which contains a union operation is of the form $D \leftarrow D \cup A$ or $D \leftarrow A \cup D$. The resulting D (denoted as D') contains all the tuples in both the initial D and A. To find only the tuples in the initial D, we would have to remove the tuples in A from D'. That is, $\cup^{-1}(D') = (D, A)$ where $D = D' - A$. However, this requires that all the tuples in A are known, otherwise the result is incorrect. Since the reconstruction algorithm does not guarantee that a complete relation can be reconstructed, it implies that a reconstructed Relation A cannot be used to compute the initial D. Thus, the Relation A must not have been modified. The inverse algorithm handles this on line 12 of figure 5.2. We now show that if A is known, then $Q^{-1}(Q(D)) = D^*$, where $D^* \subseteq D$:

$$\begin{aligned} Q^{-1}(Q(D)) &= \cup^{-1}(D \cup A) = \cup^{-1}(A \cup D) \\ &= \cup^{-1}(D') \\ &= \cup^{-1}(D')|A \\ &= D' - A \\ &= D^* \text{ where } D^* \subseteq D \end{aligned}$$

Since the `INVERSE` function ensures that A is known before finding the inverse of a union operation, it implies that the function generates a correct result when the query parameter contains a union operation. □

Lemma 6.1.6. *The* `INVERSE` *function yields a correct output for queries with the difference operation as its parameter.*

Proof. If a query involving a difference operation is used as a parameter of the `INVERSE` function, we know from lemma 6.1.1 that the query is of the form, $D \leftarrow D - A$ or $D \leftarrow A - D$. The first case implies that the resulting D (denoted as D') is a subset of the initial D (with tuples in A removed). If A is known and A is a subset of D, then A

can be added back to D' to regenerate D completely. To show that $Q^{-1}(Q(D)) = D^*$, where $D^* \subseteq D$, we have:

$$\begin{aligned} Q^{-1}(Q(D)) &= -^{-1}(D - A) \\ &= -^{-1}(D') \\ &= \begin{cases} D' \cup A & \text{if } A \text{ is known and } A \subseteq D \\ D' & \text{otherwise.} \end{cases} \end{aligned}$$

We know that if $A \subseteq D$ then $(D' \cup A)$ is also a subset of D and so also is D'. Thus, it implies that the `INVERSE` function generates a correct inverse in the first case.

The second case of the difference operation queries $(D \leftarrow A - D)$ implies that, any tuple in A is either in the initial D or the resulting D' after the query is executed. If the tuples in D' are removed from A, the result is D^*, where $D^* \subseteq D$. That is, we have:

$$\begin{aligned} Q^{-1}(Q(D)) &= -^{-1}(A - D) \\ &= -^{-1}(D') \\ &= A - D' \\ &= D^* \end{aligned}$$

$D^* \subseteq D$ since there might have been some tuples in D which were not in A when the difference operation was performed. Thus, the inverse is also correct in this case. □

Lemma 6.1.7. *A query with a Cartesian product operation cannot be a parameter of the* `INVERSE` *function but the inverse Cartesian product can be found correctly.*

Proof. From 6.1.1, if a Cartesian product is the operation performed in a query used as parameter of `INVERSE` function, then the query should be of the form $D \leftarrow D \times A$ or $D \leftarrow A \times D$. However, considering database schemas, these queries cannot occur in practice as the resulting Relation D' would have a different schema from Relation D that was used as the operand. Thus, queries with Cartesian product cannot be used as parameter of the `INVERSE` function.

Regardless of this, the inverse of a Cartesian product can be found correctly (as in line 63 of figure 5.2). We know that given two relations $D(A)$ and $E(B)$ where attributes $A = \{A_1, A_2, \ldots, A_n\}$ and attributes $B = \{B_1, B_2, \ldots, B_m\}$, the relation resulting from their Cartesian product ($G \leftarrow D \times E$) has attributes $AB = \{A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m\}$. If the sets of attributes can be separated from each other in G and redundant tuples is removed the relations generated are exactly D and E. That is, $D = \pi_A(G)$ and $E = \pi_B(G)$ which is the inverse Cartesian product definition in Section 5.2. We now show that $Q^{-1}(Q(D)) = D$ since inverse Cartesian product generates a complete inverse:

$$\begin{aligned} Q^{-1}(Q(D)) &= \times^{-1}(D \times E) \\ &= \times^{-1}(G) \text{ where } G \text{ has attributes } \{A, B\} \quad (6.2) \\ &= \pi_A(G) \quad (6.3) \\ &= D' \text{ with attributes } A_1, A_2, \ldots, A_n \end{aligned}$$

Since the projection operation does not modify the data in the columns projected in equation 6.3, and D' has exactly the same attributes as D, it implies that the inverse of the Cartesian product, $D' = D$. Note that, the inverse Cartesian product has two outputs (D' and E') and E' can be proven to be correct in the same manner as above. □

Lemma 6.1.8. *A query with a join operation cannot be a parameter of the* `INVERSE` *function but the inverse of a join can be found correctly.*

Proof. For the same reasons as in lemma 6.1.7 above, it is impossible to have a query of the form ($D \leftarrow D \bowtie_{p(A,B)} A$) or ($D \leftarrow A \bowtie_{p(A,B)} D$). Thus, queries with a join operation cannot be a parameter of the `INVERSE` function. In addition, since a join operation is the same as a Cartesian product with some conditions that must be satisfied by tuples that will be joined, the proof that the inverse of a join operation can be found correctly follows from lemma 6.1.7 as well. However, since it is possible that not all tuples of D and E are joined, the Relation G in equation 6.2 may not contain all the joining of tuples in D and/or E, thus the projection in equation 6.3 may not contain all the tuples that

was previously in D. The resulting relation of the inverse join operation D' is therefore a subset of D. □

Lemma 6.1.9. *A query with a projection operation cannot be a parameter of the* `INVERSE` *function but the inverse of a projection can be found correctly.*

Proof. From lemma 6.1.1, we know that if a query used as a parameter of the `INVERSE` function involves a projection operation, then the query is of the form $D \leftarrow \pi_A(D)$. On the contrary, this query cannot occur in practice as the schema of the resulting D' and the initial D are different. The exception to this however is when A represents all the attributes of D, in which case the `INVERSE` function can generate a correct result.

Now we show that the inverse of a projection operation can be found correctly. Given a projection operation $G \leftarrow \pi_A(D)$, we know that G contains all the data in some of the columns in D. Thus, a correct inverse projection can always be found by copying the data in G back into D and leaving columns that were not specified in the projection as columns with missing values. Assume that, D as attributes $\{A, B\}$, to show that $Q^{-1}(Q(D)) = D'$ where $D' \subseteq D$, we have,

$$\begin{aligned} Q^{-1}(Q(D)) &= \pi_A^{-1}(\pi_A(D)) \\ &= \pi_A^{-1}(G) \\ &= D' \text{ where } D' \subseteq D \end{aligned}$$

The Relation D' has exactly the same attributes as D but with values in attributes B set to null since they cannot be determined from the inverse projection. Since $D' \subseteq D$, it implies that the inverse of a projection can be found correctly. The inverse projection is the only operation that generates partial column inverses. □

Lemma 6.1.10. *The inverse of a selection operation can always be found correctly.*

Proof. The proof of lemma 6.1.3 shows that the `INVERSE` function yields a correct result for queries with selection operation. The same proof applies when the queries are of the

form $D \leftarrow \sigma_{p(A)}(G)$ or $A \leftarrow \sigma_{p(A)}(D)$. That is, queries (with selection operation) which cannot be a parameter of the `INVERSE` function according to lemma 6.1.1. □

Using the various lemmas proven above, we now show that the output from the reconstruction algorithm is correct. As mentioned earlier, correctness in these proofs means that the reconstructed relation is at least a subset of the original relation and may be missing some tuples or values in some columns. The correctness proof of the reconstruction algorithm is based on the principle of mathematical induction [52] which is applied on the entries of the relational algebra log.

Theorem 6.1.11. *Any relation in a database can be reconstructed at any earlier time.*

Proof. Given a relational algebra log, we show that an earlier instance of a relation can be correctly regenerated using the reconstruction algorithm presented. We assume that the log entries are labelled L_0, L_1, ..., L_n where L_0 is the last query executed in the log file and L_n is the first query in the RA log. For simplicity, the proof is divided into two parts. From figure 5.2, the algorithm involves series of reverse operations on queries performed (for example, on lines 11, 22, 28), as well as re-execution of queries with known operands (for example, on lines 14, 20, 42). The first part of the proof shows that relations can be reconstructed from the reverse operations on queries using the principle of mathematical induction. The second part of the proof, which is a consequence of the first part, shows that queries can be re-executed in order to regenerate a relation.

For the base case of the inductive proof, it is trivial that we can always reconstruct any relation after the execution of L_0 since L_0 represents the current instance of the database. In reality, this does not require any reconstruction since all the relations in the current database instance are known.

Now we assume that the database instance after the execution of RA log entry L_k where $0 < k < n$ is known. That is, the information contained in all the relations in the database after the execution of L_k are known.

We can now show that any relation in the database after the execution of L_{k+1} can be reconstructed. From the definition of a value block, we know that a relation remains unchanged until a new value block of the relation is encountered in the RA log and a new value block of a relation is characterized by an assignment operation into the relation in the first query of the value block. Thus, to prove that a relation can be reconstructed after the execution of entry L_{k+1} of the RA log, it suffices to show whether the relation was modified at L_k or not. This can be done by checking the query executed at L_k. If the query at L_k belongs to a new value block of the relation to be reconstructed, then the relation at L_{k+1} is different from what we currently know at L_k. Otherwise, the relation at L_{k+1} is the same as what is known at L_k.

If D is the relation to be reconstructed, we now consider the different query cases that can occur at L_k as follows:

1. The query does not involve D: this is a query of the form $(A \leftarrow B \text{ op } C)$ where A, B and C are relations. The Relation D at L_{k+1} in this case can be reconstructed since we know D after the execution of L_k and D was not generated from the execution of the query at L_k. That is, both log entries at L_{k+1} and L_k are in the same value block of D. Thus, since D is known at L_k then D is known at L_{k+1}.

2. The query involves D but does not include an assignment operation into D: this includes queries of the form $(A \leftarrow \text{ op } D)$, $(A \leftarrow A \text{ op } D)$ or $(A \leftarrow D \text{ op } A)$. Although these queries involve the Relation D as an operand of the operations performed, we know that in relational algebra notation, a relation is only modified if there is an assignment into it. Thus, if any of these queries occur at L_k, it implies that both L_{k+1} and L_k are still in the same value block of D and since D is known at L_k then D is known at L_{k+1}.

3. The query modifies D: queries modifying a Relation D can be of the form $(D \leftarrow \text{ op } D)$, $(D \leftarrow A \text{ op } D)$, $(D \leftarrow D \text{ op } A)$, $(D \leftarrow \text{ op } A)$ or $(D \leftarrow A \text{ op } B)$. From lemma 6.1.1, we have shown that a query modifying a Relation D must have D as an operand of the operation performed except if D is being created for the first

time. This implies that if either of the queries ($D \leftarrow$ op A) or ($D \leftarrow A$ op B) is performed at L_k then the Relation D does not exist at L_{k+1}. In the algorithm, a reverse operation to regenerate a relation that has just been created never occurs since calls to the `INVERSE` function requires that there is an earlier value block of the relation being considered. On the other hand, if a query of the form ($D \leftarrow$ op D), ($D \leftarrow A$ op D) or ($D \leftarrow D$ op A) is performed at L_k, it implies that the Relation D at L_{k+1} is different from the one at L_k. To determine the Relation D after the execution of L_{k+1}, we have to be able to reverse the query at L_k. From lemma 6.1.1 through lemma 6.1.10 above, we have shown that the `INVERSE` function can be used to find the inverse of queries used as its parameter and also shown that inverses of queries which cannot be used as parameter of the `INVERSE` function can be determined if needed. This implies that the reconstruction algorithm can be used to determine the value of Relation D at L_{k+1}. That is, it can regenerate an earlier instance of a relation by reversing the queries performed on such relations.

For the second part of this proof, we show that a relation can also be reconstructed by re-executing the queries that were used to generate the relation initially. As shown in the reconstruction algorithm (for example, lines 14, 20, 37, 42), this requires that the relations involved in the operation can be reconstructed in a particular (usually earlier) value block. In the first part of the proof, we have shown that a relation can be reconstructed at an earlier log entry (or value block). Thus, if a Relation D was generated at L_{k+1} as $D \leftarrow A$ op B and both A and B can be reconstructed (or are known) in their associated value blocks, then it is trivial that D can also be regenerated by executing the query again.

Thus, from the base case (L_0) and the induction step ($L_k \Rightarrow L_{k+1}$), it can be concluded that the reconstruction algorithm can be used to generate an earlier instance of a relation in a database. □

Theorem 6.1.12. *Any relation generated from the reconstruction algorithm is correct.*

Proof. From theorem 6.1.11, we know that an earlier instance of a relation in a database can be generated using the reconstruction algorithm. Also, from the definition of the inverse relational algebra operators in Section 5.2, we know that having a reconstructed relation does not imply that all the information in the original relation were successfully reconstructed. That is, a reconstructed relation may contain missing tuples or missing values in column(s). However, we now prove that the tuples in any reconstructed relation are indeed in the original relation, meaning that the reconstructed relation is correct.

From figure 5.3, we know that there are three major ways to regenerate a relation using the algorithm. First, is when there is no subsequent value block of the relation to be reconstructed. In this case, the correctness of the relation returned from the algorithm is trivial since it implies that the relation has not been modified and can be simply returned as in the algorithm. The other cases involve finding the inverse of queries and re-executing of queries based on relations that are known or also reconstructed from inverses. In lemma 6.1.1 through lemma 6.1.10, we have shown that the results generated from the `INVERSE` function and the inverse operators of the relational algebra are correct. Since reconstructed relations are based on these inverse operators and the `INVERSE` function in figure 5.2, it implies that any output from the reconstruction algorithm is correct. The term "correct" in this case means that the reconstructed relation is at least a subset of the original relation. □

6.2 Total Correctness

In Section 6.1, we have shown that if the reconstruction algorithm terminates, then the output generated is correct. In order to complete the correctness proof of the algorithm, this section presents the proof that the algorithm always terminates. Similar to Section 6.1, the proof is established based on the sequence of lemmas which we prove below.

Lemma 6.2.1. *There is at least one query in the set Q.*

Proof. The set Q is defined in line 1 of figure 5.3 as the set of queries involving the

relation to be reconstructed (D) in the value block in which it is to be reconstructed (V_{D_i}). Showing that Q is a non-empty set is trivial since every value block starts with an assignment operation into the relation involved. Thus, Q contains at least one query which is the first line of the value block V_{D_i} and the query contains an assignment operation into D. □

Lemma 6.2.2. *The number of log entries in the relational algebra log is finite.*

Proof. We have assumed that the relational algebra log used in the reconstruction algorithm consists of a complete set of modifying queries that have been performed on a database from the time in which a relation is to be reconstructed (or earlier) to the present time. The number of operations that can be performed on a database in a bounded time interval is finite. Thus, the number of log entries that can be in a query log is finite. Since a relational algebra log is generated from traditional query logs, it implies that the number of log entries that can be in a RA log is always finite. □

Lemma 6.2.3. *The number of value blocks in a relational algebra log is finite.*

Proof. We know that a database consists of a finite number of relations [32]. Lemma 6.2.2 also shows that number of log entries in an RA log is finite. Since value blocks are defined using the entries of the relational algebra log (by putting into consideration relations that are modified by any query), and they represent groupings of the log entries, it implies that the number of value blocks that can exist in a relational algebra log is also finite. □

Lemma 6.2.4. *The number of possible relation and value block tuples are finite.*

Proof. As mentioned earlier, we know that the number of relations in a database is finite. Lemma 6.2.3 also proves that the number of value blocks into which a RA log can be divided is finite. If we represent the number of relations in a database and the number of value blocks in a RA log as S and T respectively, we can show that the set representing the Cartesian product of S and T is finite.

Since S and T are finite sets, it implies that the cardinality of $S \times T$ is $|S| \times |T|$ which is also finite. $S \times T$ represents the total number of relation and value blocks tuples that can exist. Since $|S \times T|$ is finite, it implies that the number of relations and value block tuples that can be encountered in the reconstruction algorithm is finite. □

Theorem 6.2.5. *The database reconstruction algorithm always terminates.*

Proof. There are various conditions in the reconstruction algorithm that ensure its termination. The simplest condition checks whether or not there is a subsequent value block of the relation being reconstructed after the value block specified in the call to the reconstruction algorithm. This step occurs on lines 9, 17, 26, 45 and 57 of figure 5.3. It is obvious that none of the instances leading to these lines requires a recursive call to the `SOLVE` function. Thus, the termination of the reconstruction algorithm can be said to be trivial in these cases.

In the non-trivial cases of the reconstruction algorithm, at least one subsequent or earlier value block of the relation being reconstructed exists. There are two conditions in the algorithm that ensure its termination in these cases. The first one is the set Q defined in the algorithm (on line 1 of figure 5.3), which contains a list of all the queries involving the relation to be reconstructed in the specified value block. The algorithm works by examining each of the queries in Q and attempting a reconstruction based on each query. Lemma 6.2.1 shows that there is at least one query in Q whenever the `SOLVE` function is called. In lemma 6.2.2, we have also shown that the number of log entries in the RA log is finite. Since the set Q is a subset of the RA log, it implies that Q is also a finite set. Since the reconstruction algorithm considers each element of Q only once, it implies that the algorithm eventually considers all the elements of Q and terminates when there are no more elements in Q to consider.

However, we note that the analysis of each query in Q can lead to a recursive call to the `SOLVE` function. The algorithm ensures termination in this case by keeping track of the set S (a parameter of the `SOLVE` function) which stores tuples of relations and value blocks

(and the associated result) already considered in a particular reconstruction process. If a relation and value block tuple which has been stored in S is used as parameters `SOLVE` function, the algorithm simply returns the result associated with such parameter combination. Otherwise, the tuple is included in the set S and the corresponding result is initially set to an empty relation which is replaced by the reconstructed relation after the successful completion (line 81 of figure 5.3) of the reconstruction process. Thus, if an attempt to reconstruct a relation in a value block leads to a recursive call to reconstruct the relation in the same value block, it implies that the relation cannot be reconstructed and the algorithm return an empty relation and terminates. On the other hand, if a relation has already been reconstructed in a particular value block, there can never be an infinite loop to continue reconstructing the same relation in the same value block since the associated result is always returned (as in line 3 of figure 5.3). Even though the algorithm contains recursive calls to the `SOLVE` function, we have proved in lemma 6.2.4 that the number of relation and value block tuples that can be encountered in a reconstruction process is finite. The worst case scenario that can be encountered in the reconstruction algorithm is when every relation is reconstructed in every value block (an almost impossible case). Since the number of such tuples is finite, then it implies that the reconstruction algorithm always eventually terminates. □

The proof of partial correctness of the database reconstruction algorithm presented in Section 6.1 and the proof of its termination presented in Section 6.2 both show that the reconstruction algorithm is correct.

6.3 Summary

This chapter builds on the database reconstruction algorithm presented in Chapter 5 by proving the algorithm to be correct. This is in support of the requirement that the tools and techniques used in the analysis and presentation of evidence in digital forensics must be proven to be reliable. Due to the mathematical nature of the relational algebra, which is a core aspect of the techniques presented in this thesis, the proof of correctness is done

by applying mathematical methods of proof such as proof by contradiction and proof by induction. The correctness proof of the database reconstruction algorithm is described in two sections where the first deals with the partial correctness of the algorithm and the second deals with the total correctness of the algorithm. The proof of partial correctness shows that the output of the algorithm is correct by presenting several lemmas and building on them to reach the conclusion. The proof of total correctness shows that the algorithm will always terminates. The conclusion is also reached by building on several lemmas presented in the chapter.

This chapter proves the applicability of the algorithm for the reconstruction process in database forensics. In the following chapter, the completeness of the reconstructed data that can be generated from this algorithm is considered as it is also important that as much information as possible can be reconstructed if required during the forensic analysis of a database.

Chapter 7

Completeness of Reconstructed Data

> "Put on the whole armor of God, that you may be able to stand against the wiles of the devil."
>
> - Ephesians 6:11

In the previous chapter, the proof of correctness of the database reconstruction algorithm is presented. The term *correctness* implies that any reconstructed relation is at least a subset of the original relation, even though it may be incomplete if compared (in terms of the tuples contained) with the original relation. The generation of incomplete relations or inability to reconstruct values of interest in a relation when using the database reconstruction algorithm 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. This 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 chapter 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 relation as well as provide corroborating evidence regarding claims about the information on a database at an earlier time. A typical application of the reconstruction algorithm which reflects its limitation is given and two different techniques of reconstructing more information from a database using the reconstruction

algorithm are described in the chapter.

Section 7.1 describes the limitation of the database reconstruction algorithm in detail and gives a typical example of a situation in which the limitation can be encountered. In Section 7.2, the absence of evidence in terms of data reconstruction in database forensics is discussed. Sections 7.3 and 7.4 describe techniques that can be applied for the generation of complete inverses by showing how the missing values of the example given in Section 7.1 can be reconstructed.

7.1 Limitation of the Reconstruction Algorithm

As mentioned earlier, 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, partial inverses sometimes affect 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 the operators sometimes result in the generation of incomplete (or empty) relations when using the algorithm.

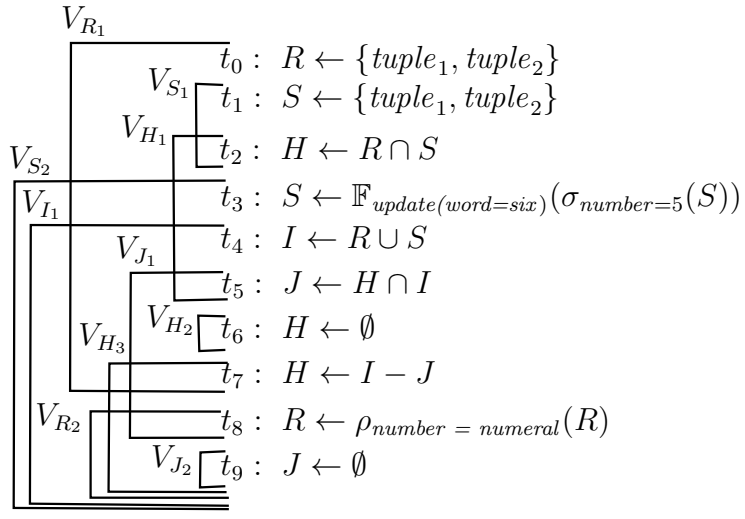

Figure 7.1: A Relational Algebra Log Grouped into Value Blocks.

This section gives an example that reveals the limitation of the algorithm when dealing

with inverse operators that generate partial inverses. In figure 7.1, an example of a RA log generated from a traditional SQL log file is given. For simplicity and further explanations in subsequent sections of this chapter, we assume that the content of each relation after executing the queries in the RA log at each timestamp are as computed in figure 7.2.

The aim of using the reconstruction algorithm is to reconstruct the tuples in Relation H at time t_3 since several modifications of the relation has occurred. The relations R and S both have attributes `word` and `number` and contains two tuples each, which were stored at time t_0 and t_1, respectively.

From figure 7.1, the Relation H at time t_3 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 $\rho^{-1}(R) =$

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

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
	five	5
	six	6

$t_9: J \leftarrow \emptyset$

J	Word	Number

Figure 7.2: Original Relations Obtained from Queries Executed.

$\rho_{numeral\ =\ number}(R)$. 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 7.1. 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 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 7.2).

S^*	Word	Number
	four	4
	null	5

Table 7.1: Reconstructed Relation S^*.

H^*	Word	Number

Table 7.2: An Empty Reconstructed Relation H^*.

This example reflects a limitation of the database reconstruction algorithm that can be encountered when dealing with partial inverses. In the rest of this chapter, 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 in a database during database forensics.

7.2 Absence of Evidence

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[1], to 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" [29]. 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. Also, 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 [29]. Thus, it is necessary for an investigator to find corroborating evidence that clearly demonstrates the falsity or truth of a claim about the information in a database at an earlier time.

There are two techniques that can be used for 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 in an exchange [34]. That is, there will always be some trace evidence with every interaction even though it may not be easily detected. As discussed in Section 2.2.3, this principle applies in both the physical and digital realms and can provide links between them [29]. 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, for example a database. In database reconstruction, the items involved in an interaction are the relations in a database while the interactions are the operations performed on such relations. This technique works on the fact that if

[1]evidence may or may not be admissible in a court of law.

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 in the database.

The second technique for finding corroborating evidence involves the reconstruction of more complete relations through the iteration of the database reconstruction algorithm and making inferences from reconstructed relations. This technique works based 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 [53].

The following sections 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.

7.3 Reconstruction from Interaction

According to Locard's exchange principle [34], 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 from a crime scene.

From the reconstruction example in Section 7.1, 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 the 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 7.1, 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 7.1) 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 7.2 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 using the reconstruction algorithm. 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[2]:

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

which is as shown in table 7.3.

J^*	Word	Number
	five	5

Table 7.3: Relation J^* from Inverse Difference Operation.

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 interaction 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, we have the Relation H which is given

H	Word	Number
	five	5

Table 7.4: H through Reconstruction from Interaction.

as table 7.4 instead of the earlier empty relation in table 7.2.

7.4 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.

[2]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.

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 (that is, as it was found before the investigation). Since it has been proved in 6 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 7.1, 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 7.1) 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 need to be reconstructed. For example, our attempt to reconstruct Relation H at t_3 in figure 7.2 required the reconstruction of relations R and S. For simplicity, the subscript r is used 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 7.1) given in figure 7.3.

	Word	Number
R_r	five	5
	six	6

	Word	Number
S_r	four	4
	null	5

Figure 7.3: Reconstructed Relations R_r and S_r.

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 7.2 using the relations R_r and S_r. The re-execution process is shown in figure 7.4.

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

Figure 7.4: Re-execution of Queries Using 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

Figure 7.5: Reconstructed Relation I_r and Current Relation I.

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 7.5) 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 7.5. Since the Relation S_r generated from the inferences are exactly the

S_r	Word	Number
	four	4
	seven	5

Table 7.5: Table S_r Generated from Re-execution and Inferences.

same as the current 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 7.6. The Relation H_r generated from the re-execution process at time t_7 should be

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.

Figure 7.6: Re-execution of Queries Using Reconstructed Relations.

the same as the current instance of H on the database. But, this is not the case (as shown in figure 7.7). 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

	Word	Number
H_r	four	4
	five	5
	six	6
	null	5

	Word	Number
H	four	4
	six	6
	seven	5

Figure 7.7: Reconstructed Relation H_r and Current Relation H.

it should have been in the Relation J_r since 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 7.6.

H	Word	Number
	five	5

Table 7.6: H from Reconstruction Through Iteration.

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. Both techniques can be used in reconstructing data when dealing with situations involving incomplete reconstruction of some other relations or the deletion of the 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 in any particular situation.

7.5 Summary

One limitation of the database reconstruction algorithm deals with the fact that reconstructed relations may be incomplete due to information that may be missing in the

output generated by the inverse operators of the relational algebra. Unfortunately, this may imply missing evidence during an investigation that requires the ability to reconstruct missing parts of a reconstructed relation. However, since the evidence may still exist, it is important to find ways to reconstructing the missing the data or find corroborating evidence that can be used to support claims about the data on a database. This chapter describes techniques that can be used for ensuring that more complete relations can be reconstructed using the reconstruction algorithm earlier described in Chapter 5. The techniques explore axioms and principles in forensic science and show with a typical example how missing data in a reconstructed relation can be determined.

As mentioned in Section 3.2, one of the characteristics of a database system is that the database contains both the data stored in it together with a complete description of the database and the data contained, called the metadata or schema. In the next chapter, the concepts described for the reconstruction of data contained in a database is extended for the reconstruction of the schema as well. This handles the earlier assumption that the database schema is always known and describes techniques that can be used for reconstructing the schema when it is unknown or not trusted, which may be the case of a compromised database.

Chapter 8

Reconstruction of a Database Schema

> "Neither do men put new wine into old bottles; else the bottles break, and the wine is spilled, and the bottles perish. But they put new wine into new bottles, and both are preserved."
>
> - Matthew 9:17

The focus of this chapter is to describe techniques that can be used for the reconstruction of a database schema. Until this chapter, it has been assumed that the schema is known and can be trusted, since our focus is on the dimension of modified databases. However, in practice this is not always the case. One of the main challenges of database forensics analysis deals with the fact that the results obtained from an analysis may actually differ from the raw data stored in the database due to changes that may have been made to the metadata. Although a situation involving a metadata change would be categorized as a compromised database forensics dimension (Section 4.2), it is interesting to consider how the database reconstruction algorithm and other techniques described in the thesis till now can be applied in the reconstruction of the database schema.

This chapter considers how the actual schema of a database can be reconstructed after the database has been compromised or some information have been lost or deleted. The categories of changes that can be made to a database schema by an attacker are described and typical examples are used to show that metadata changes can truly affect

results obtained from queries executed on a database. Techniques for reconstructing the schema are then discussed. Similar to the reconstruction of the data in a database relation, the techniques described work by considering the operations performed on the relations involved and applying the inverse operators of the relational algebra. Section 8.1 highlights the fact that a database schema can be compromised using typical examples and Section 8.2 describes various techniques that can be used for schema reconstruction in database forensics. This chapter is based on a previously published paper [4].

8.1 Compromising a Database Schema

In Section 4.2, the various dimensions that exist in database forensics have been described. The category of compromised databases deals with databases where some of the metadata of the DBMS have been modified, even though the database is still operational. This section describes some of the various ways in which a database schema can be compromised and illustrate the fact that this causes a database to give false answers to queries. The study presented in this section was done using Postgres 9.2 running on a Windows 8 operating system.

There are several activities that can be carried out on a database schema to compromise the database. Usually an attacker may compromise a database with the intention of damaging the database so that it is not useful for the intended purpose or users. An attacker may also compromise a database in an attempt to hide some data (stored legitimately by a user or by the attacker) or to cover his tracks. In general, we can categorize the activities that can be carried out by an attacker to achieve his objectives into the following groups:

- Localized changes to data,
- Changes to blocks of data, and
- Changes to links between blocks of data.

8.1.1 Localized Changes to Data

Within a database schema, an attacker may make changes to specific columns of a relation. This may be done to hide the information contained in such columns or make the data unavailable to the database user. In practice, there are several ways in which this can be accomplished. A compromise such as simply swapping two columns can have severe effects on the operation of a database [100]. A typical example is where the "Purchase price" and the "Selling price" of a store's database are swapped by tampering with the schema. This obviously causes the store to sell their goods at a loss to the store. Figure 8.1 shows a code segment that can be used to compromise a database by swapping the column names through the schema. This swap causes a `select` statement on one of the columns to return values from the second column.

```
update pg_attribute set attnum = '5' where attrelid = '16432'
 and attname = 'purchasePrice';
update pg_attribute set attnum = '2' where attrelid = '16432'
 and attname = 'sellingPrice';
update pg_attribute set attnum = '3' where attrelid = '16432'
 and attname = 'purchasePrice';
```

Figure 8.1: Modifying Schema to Swap two Column Names.

A possible way of manipulating the database schema by changing specific values in the schema in order to hide information is to change the identification of the attribute (within the schema) to something unknown to the schema. Although the raw data may still be present in the database, it becomes impossible to query the database for that information. It is possible that information initially thought to have been deleted or non-existent, turns out to be irretrievable because it now appears hidden from the DBMS through a modification of the schema. Figure 8.2 shows the commands that can be used to hide the attribute `purchasePrice` from the authorized users by changing the identification used by the DBMS to access the column into a different one.

Details such as the data type of the attributes of a relation can also be modified by

```
update pg_attribute set attnum = '5' where attrelid = '24587'
  and attname = 'purchasePrice';
```

Figure 8.2: Modifying Schema to Hide a Column.

changing specific values in the schema. This information affects both the type of information that can be inserted into a column and the information that will be retrieved when the column is queried. Modifying an attribute's data type in the schema causes data (whether being inserted or retrieved) to be formatted according to the new data type. The implication of this is that it may become impossible to comprehend previously stored information due to changes in the data type. Figure 8.3 displays the command that can be used to change the data type of a column to another recognized data type in a database. An example of the original data and the result of a query executed after modifying the data type in the schema are shown in figures 8.4(a) and 8.4(b), respectively.

```
update pg_attribute set atttypid = '18' where attrelid = '16432'
  and attname = 'firstName'
```

Figure 8.3: Modifying Schema to Change Attribute's Datatype.

Depending on the DBMS, it is possible to compromise various parts of a database schema, including the column names, data types, constraints and other details in the schema by changing specific values in the schema. An attacker with an understanding of the structure of a DBMS can execute a SQL query that affects the schema and the result obtained from subsequent queries on the database.

8.1.2 Changes to Blocks of Data

Apart from changes to specific values, blocks of data such as complete tables or set of columns can be modified through the schema of a database. A typical instance is where an attacker changes similar column names in different tables to a different name or changes the name of several tables to different names that are not understandable

id integer	lastname character varying(45)	firstname character varying(45)
21	Kirsten	Paltrow
22	Elvis	Marx
23	Sandra	Kilmer
24	Cameron	Streep
25	Kevin	Bloom
26	Rip	Crawford
27	Julia	Mcqueen
28	Woody	Hoffman
29	Alec	Wayne
30	Sandra	Peck
31	Sissy	Sobieski
32	Tim	Hackman
33	Milla	Peck
34	Audrey	Olivier
35	Judy	Dean
36	Burt	Dukakis
37	Val	Bolger
38	Tom	Mckellen
39	Goldie	Brody
40	Johnny	Cage

(a) Original Data

id integer	lastname character varying(45)	firstname "char"(49)
21	Kirsten	
22	Elvis	
23	Sandra	{
24	Cameron	D
25	Kevin	↕
26	Rip	
27	Julia	
28	Woody	j
29	Alec	9
30	Sandra	♂
31	Sissy	
32	Tim	
33	Milla	r
34	Audrey	;
35	Judy	
36	Burt	
37	Val	
38	Tom	h
39	Goldie	3
40	Johnny	∟

(b) Retrieved Data after Schema Change

Figure 8.4: View of a Relation Before and After Data type Compromise in Schema.

to the user. For example, executing the code in figure 8.5, changes the name of all the tables with names starting with letter `s` into a different name. As such, it becomes impossible to retrieve the tables with the original name. In the same way as changing the name of several blocks of data, blocks of data can also be removed or even combined to compromise the database.

```
update pg_class set relname = 's-unknown'||nextval('serial')
where relname in
select relname from pg_class where relnamespace=2200 and
 reltype!=0 and relkind ='r' and relname like 's%'
```

Figure 8.5: Changes to Blocks of Data in Schema.

8.1.3 Changes to Links Between Blocks of Data

Another category of changes that can be made to a database schema to compromise the database involves changes that affect the links between data. This type of changes often involves a modification of the foreign keys and/or the primary keys of the relations in the database. It can involve a localized change of specific information in the schema or changes to groups of data in the schema or a combination of both. A typical instance is where an attacker modifies the constraint specified on a table (within the schema) by removing the foreign key, so that it becomes impossible to link the information within two different tables. It is also possible to modify the links such that a table is linked to a wrong table. For example, executing the code in figure 8.6 causes a primary foreign key to become a primary key in a table. The trigger controlling the constraint is also modified. The effect of this is that values that do not have a corresponding value in the referenced table can now be entered into table with the modified schema.

```
update pg_constraint set contype = 'p' where conname = 'DEPT_FK';
update pg_trigger set tgtype = 17 where oid = 16460;
```

Figure 8.6: Changes to Link Between Blocks of Data in Schema.

Regardless of the category of changes that can be made by an attacker, an important aspect of database forensics is the reconstruction of information. This requires that the schema of a table can be reconstructed along with the data in the table. In the following section, we describe techniques that can be used to reconstruct the schema by considering operations that have previously been performed on the database.

8.2 Schema Reconstruction

In this section, we again explore the simplicity and mathematical nature of relational algebra in the reconstruction of database schema. Section 8.2.1 describes how the schema of a relation can be reconstructed by looking at the conditions associated with each of the operators of the relational algebra. Section 8.2.2 investigates the application the

concept of the inverse relational algebra previously used for the reconstruction of data in a database for reconstructing schemas. Section 8.2.3 describes how the schema can be reconstructed by checking the consistency of the information in the database.

8.2.1 Reconstruction from Previous Manipulations

As discussed in Section 3.4, database logs provide a rich source of information that can be used for reconstruction in the forensic analysis of a database. Although, the logs may be stored in different formats, translating the log into a relational algebra log (RA log) offers several benefits described in Section 5.1 and allows an investigator to exploit the characteristics of the relational algebra. Given a RA log, the attributes of the operations performed on a relation can be used to determine the schema of the relation. In addition, the operations performed on a database to compromise the schema can also be represented in relational algebra notation since they are usually SQL operations.

Many of the operators of the relational algebra that require two operands have the characteristics that the structures of the two operands are the same. The Union, Intersection and Difference operators fall into this category. Given a query expressed in relational algebra,

$$C \leftarrow A \text{ op } B$$

where A, B and C are relations and `op` is a union ($\cup$), intersection ($\cap$) or difference operation ($-$), the three relations have exactly the same structure. Thus, if the structure of one of the relations is known, it can be used as the structure for any of the other two relations and the data in the relation can be reconstructed [56, 53] based on this structure.

This is also true with the selection operator. Given select query, $B \leftarrow \sigma_{p(attr)}(A)$, where some tuples in A are selected as B given the condition expressed as $p(attr)$, both relations A and B have exactly the same structure. And the structure of one can be used for the other during reconstruction of data or to infer the structure of other relations.

An interesting aspect of applying this approach to schema reconstruction deals with the fact that it makes it possible to combine different operations and analyze what the likely

schema for the resulting relation would be.

8.2.2 Reconstruction Using Inverse Relational Algebra

A more formalized approach to reconstructing the schema of a relation is to apply the database reconstruction algorithm earlier described. Since a major part of the algorithm is based on the inverse relational algebra, we describe how the inverse operators of the relational algebra can be used for schema reconstruction.

Transposing Schemas into Relations

Since the inverse operators of the relational algebra require a relation as their operand, it is necessary that the schema of a relation can also be expressed as a relation in order to use the inverse operators in schema reconstruction. Many DBMSs provide the ability to retrieve the structure of a relation on the database as a table. In Postgres for example, the command shown in figure 8.7 can be used to retrieve the structure of a relation into a table where each attribute of the relation becomes a tuple in the retrieved table. Figure 8.8 shows a typical result of the application of this command.

```
select ordinal_position, column_name, data_type, is_nullable,
  descrip.description AS comment
from information_schema.columns columns
left join pg_class class on (columns.table_name = class.relname)
left join pg_description descrip on (class.oid = descrip.objoid)
left join pg_attribute attrib on (class.oid = attrib.attrelid
  and columns.column_name = attrib.attname
  and attrib.attnum = descrip.objsubid)
where table_name= 'actor'  /* The table name*/
group by ordinal_position, column_name, data_type,
   is_nullable, table_schema, descrip.description;
```

Figure 8.7: Retrieving Schema as a Table.

As a consequence of retrieving the schema of a relation as another relation, we also need to find the *transpose* of the operations performed on the original relations so that the transposed operation can be applied to the retrieved schemas and used for schema

	ordinal_position integer	column_name character varying	data_type character varying	is_nullable character varying(3)	comment text
1	1	id	integer	NO	
2	2	lastname	character varying	NO	
3	3	firstname	character varying	NO	
4	4	lastupdate	timestamp without time zone	NO	

Figure 8.8: Typical Schema Retrieved as a Table.

reconstruction. Below, we consider each of the relational algebra operators and identify the operation that can be considered as a transposition of each operation.

- Cartesian product ($\times$): Given a query $T \leftarrow R \times S$, we know that the resulting Relation T has all the attributes of both R and S. That is, if we can retrieve the schema of both relations R and S, then the schema of T would be the union of both schemas. Thus, the transpose of a Cartesian product operation is a union operation.
- Union ($\cup$): Given a query $T \leftarrow R \cup S$, the schema of T is the same as that of R and S. If the schema of each of the operands could be retrieved, then the schema of T would be an intersection of the two schemas. Thus, the transpose of a union operation is an intersection operation[1].
- Intersection ($\cap$): Based on the same reason as for the union operation, the transpose of an intersection operation, $T \leftarrow R \cap S$, is also an intersection operation.
- Difference ($-$): In a difference operation expressed as the query, $T \leftarrow R - S$, we know that the schema of T is the same as that of R and S. Similar to the union and intersection operation, the transpose of a difference operation is an intersection operation.
- Join ($\bowtie$): A join operation can be compared to a Cartesian product, except for the fact that a join is performed under certain specified condition and as such may not include all the tuples that might be in a Cartesian product. However, this has no effect on the attributes of the operands or the resulting relation. As such, given a

[1]The transpose of Union, Intersection and Difference operations may also be considered as a Union operation.

join query, $T \leftarrow R \bowtie_{p(A,B)} S$, Relation T has all the attributes of both R and S. Thus, if we can retrieve the schema of both R and S, then the schema of T would be the union of both schemas. This implies that the transpose of a join operation is a union operation.

- Projection (π): A projection operation selects some of the attributes of a relation as another relation. Given a query, $T \leftarrow \pi_{A_1,A_2,A_3}(R)$, the attributes of T is a subset of that of R. Thus, the transpose of the projection operation is a selection operation.
- Selection (σ): The output of a selection operation, $T \leftarrow \sigma_{p(A)}(R)$, has exactly the same set of attributes as that of the operand, since a selection only affects the tuples of a relation. Thus, the transpose of a selection operation is a selection (without any conditions) of all the tuples in the schema of the operand.
- Division ($/$): Given a query, $T \leftarrow R[A, B/C]S$, where a Relation R is divided by an attribute C of the Relation S, we know that the attribute of T will include all the attributes of A except the attribute (also in S) that was used in the division. That is, if the schema of the two relations R and S are known, then the schema of T would be the schema of R minus the schema of S. Thus, the transpose of a divide operation is a difference operation.

Operators	**Transposed Operators**
Cartesian product ($\times$)	Union ($\cup$)
Union ($\cup$)	Intersection ($\cap$)
Intersection ($\cap$)	Intersection ($\cap$)
Difference ($-$)	Intersection ($\cap$)
Join ($\bowtie$)	Union ($\cup$)
Projection (π)	Selection (σ)
Selection (σ)	Selection (σ)
Division ($/$)	Difference ($-$)

Table 8.1: Transpose of the Relational Algebra Operators.

Table 8.1 gives a summary of the *transposed* operations that can be applied in the reconstruction of a schema. As mentioned in the definition of the inverse operators of

the relational algebra, the objective of the inverse operators is to find the value of a particular attribute of a relation at a specific time t by finding the inverse of the most recent query performed on the current relation and sequentially going backwards until the desired time is reached. The same technique can be applied in schema reconstruction, with the addition that the transposed operations are used to handle the schemas involved in any of the relational algebra operations. Below, we consider the application of the inverse relational algebra in conjunction with the transposed operations and show that the inverse operators can also be used for schema reconstruction in database forensics.

Applying the Inverse RA and Transpose Operators for Reconstruction

This section describes how the transposed operators can be used together with the inverse functions defined for the operator in order to reconstruct table schemas. From table 8.1, there are four relational algebra operators that we need to consider: The union ($\cup$), Intersection ($\cap$), Selection (σ) and Difference ($-$) operators since the relational algebra operators can be transposed as one of them.

As discussed in Section 5.2, the inverse of a union operation $T \leftarrow R \cup S$ can only be determined if one of the expected outputs (that is R or S) is known. If Relation S is known, then $\cup^{-1}(T) = (R^*, S)$ where $R^* = T - S$. On the other hand, if R is known, then $\cup^{-1}(T) = (R, S^*)$ where $S^* = T - R$. A complete inverse union is found only when both R and S have no tuples in common [56, 53]. Since the transpose of the Cartesian product and the join operators is a union operation, this definition can be applied in the reconstruction of the schemas of relations involved in either operations. Given an operation $C \leftarrow A \ \texttt{op} \ B$, where `op` is a Cartesian product or a join operator, if the schema of C can be retrieved as Relation T, then the schema of A can be reconstructed using the inverse union operator, provided that the schema of B is known, or vice versa. As an example, if the schema of Relation C (S_C) and Relation B (S_B) are retrieved using the code in figure 8.7 and stored as the tables in figure 8.9, then we can reconstruct the schema of Relation A as ($S_C - S_B$), which yields the result in figure 8.10.

ordinal_position integer	column_name character varying	data_type character varying
1	id	integer
2	lastname	character varying
3	firstname	character varying
4	lastupdate	timestamp
5	emp_id	integer
6	dept	character varying
7	address	character varying

(a) Schema of Relation C (S_C)

ordinal_position integer	column_name character varying	data_type character varying
1	id	integer
2	lastname	character varying
3	firstname	character varying
4	lastupdate	timestamp

(b) Schema of Relation B (S_B)

Figure 8.9: Retrieved Schemas of Relations C and Relation B.

ordinal_position integer	column_name character varying	data_type character varying
5	emp_id	integer
6	dept	character varying
7	address	character varying

(a) Reconstructed Schema of Relation A (S_A)

emp_id	dept	address
⋮	⋮	⋮
⋮	⋮	⋮

(b) Structure of Relation A

Figure 8.10: Reconstructed Schema of Relation A Using Inverse Union Operation.

Also from Section 5.2, the inverse of an intersection operation $T \leftarrow R \cap S$, generates partial tuples inverses containing all the tuples in T, that is, $\cap^{-1}(T) = (R^*, S^*)$ where $R^* = S^* = T$. Complete inverses can be found when R and S are known to be the same [56, 53]. The inverse intersection operator can be applied for the reconstruction of the schemas of relations involved in a union, intersection or difference operation since their transpose is an intersection operation. That is, given an operation $C \leftarrow A$ `op` B, where `op` is a union, intersection or difference operator, if the schema of C can be retrieved as a Relation T, then the schema of A and B can be reconstructed using the definition of the inverse intersection operator. A unifying characteristic of the union, intersection, and difference operators is that their operands must have the same schema. Thus, if the schema of Relation C (or that of A or B) is retrieved as a table, then we can completely reconstruct the schema of the other relations by applying the inverse

intersection operator.

Given a selection operation $R \leftarrow \sigma_{p(A)}(S)$, the inverse selection is given by, $\sigma^{-1}_{p(A)}(R) = S^*$, where $S^* = R$. That is, all the tuples in R are also in S. The inverse selection yields a complete inverse if all the tuples in the operand of the selection operator (that is, S) satisfied the condition specified as $p(A)$ (Section 5.2). The inverse selection operation can be applied in the reconstruction of the schemas of relations involved in a projection or a selection operation since the transpose of both operations is a selection operation. That is, given a projection operation $C \leftarrow \pi_{A_1,A_2}(B)$ or a selection operation $C \leftarrow \sigma_{p(A)}(B)$, if the schema of Relation C is known (say S_C), then the schema of B contains all the tuples in S_C. In the case of a projection operation the reconstructed schema may be incomplete since a projection is usually a selection of some columns of the operand. In the case of a selection operation, the reconstructed schema is complete since a selection picks some of the tuples of its operand with all their attributes.

The inverse difference operation can be applied for the reconstruction of the schema of the operands of a division operation since its transpose is a difference operation. Given a difference operation $T \leftarrow R - S$, the left operand is easily determined by the inverse difference operator as $R^* = T$ since $T \subseteq R$. A complete R can be determined only if the Relation S is known and all the tuples in S are also known to be in R (that is, $S \subseteq R$) 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 we know that $S \subseteq R$, then a complete Relation S is found from the inverse as $S = R - T$ [56, 53]. This implies that if the schema of C (S_C) in the divide operation $C = A/B$ is known, then we know that the schema of A contains all the tuples in S_C. The schema may be incomplete since the Relation A may contain other attributes that were not included in the divide operation. The reconstructed schema for A is given by $S_C \cup S_B$ and is complete if we know the schema of B (S_B) and also know that all the columns in B are also in A. On the other hand, the schema of Relation B can be reconstructed as $S_A - S_C$ if the schema of A (S_A) is known. Also if all the columns in B are also in A, then the reconstructed schema for

Relation B is complete.

Although reconstructed schemas are generated as a table, the structure of the corresponding table is simply a matter of transposing the reconstructed schema so that tuples in the reconstructed table becomes the columns of the required relation. The technique described above can be applied to sequence of operations in order to arrive at the particular relation whose schema is required. This is equivalent to working with the reconstruction algorithm described in Chapter 5 and using the transposed operations and the schemas expressed as tables instead of the actual operators and operands involved in an operation.

8.2.3 Reconstruction Through Consistencies

Another approach that can be used for the reconstruction of a database schema involves checking the consistency of the information contained in the database. Typically, a DBMS provides the ability to represent complex relationships between data. The descriptions of these relationships are stored in the schema and imply that certain conditions hold between related data. An understanding of the conditions expected to hold in any relationship can be used to identify instances that fail to satisfy these conditions and this may point to actions performed by an attacker.

This approach is particularly useful for the reconstruction of the constraints associated with a relation. For example, if there is a referential integrity constraint [51] from a Relation R to a Relation S, then the attributes concerned in both relations are expected to be of the same data type. This knowledge can be used to determine the data type of the attribute involved if either of the schemas of R or S needs to be reconstructed. The referential integrity constraint also points to the fact that the referenced column is the primary key of one of the two relations. Another example of the application of database consistencies for reconstruction is the application of the entity integrity constraint [51], which specified that no primary key value can be a NULL value. This shows that an attribute that can take a null value is not the primary key of the relation whose schema

is to be reconstructed and vice versa.

A combination of the different constraints that can be specified on a database as well as the characteristics of various attributes may also be useful for identifying the characteristics relating to the schema of a relation. We note that even though it may be possible to retrieve some of the constraints in a schema using the two techniques earlier described, it may be necessary to use an additional techniques such as checking for the data consistencies in reconstructing the constraints relating to a relation.

8.3 Summary

In addition to reconstructing the data in the relations of a database, it may also be required to reconstruct the schema of the relation as well. This is mainly because the information retrieved from a database query is dependent on both the raw data contained in the database and the metadata of the database. This chapter builds on the database reconstruction algorithm earlier presented for the reconstruction of data by describing techniques that can be used in conjunction with the algorithm for reconstructing the schema of a database. This is particularly useful in the investigation of compromised database where the metadata might have been modified even though the database is still operational. The chapter discusses the different categories of changes that can be made to a database schema and then describes how the schema can be reconstructed either by considering the operations previously performed on the relation concerned or by using the inverse operator of the relational algebra and the reconstruction algorithm described in Chapter 5. The technique of reconstruction through the consistency of the information retrieved from the database with the information expected to be present on the database is also described.

The main assumption in this thesis is that log files that provide information about the operations or manipulations carried out of a database are available. Although logs are maintained by most database systems, it is possible that the information required for a forensic analysis or to use some of the techniques described in this thesis may not be

available in certain cases. In the following chapter, database log settings (or preferences) are explored in conjunction with the information required for a forensic analysis of a database system and some recommendations about the logging preferences required for an analysis are made.

Chapter 9

Effectiveness of Database logs in Reconstruction

> "When I applied my heart to know wisdom and to see the business that is done on earth, even though one sees no sleep day or night, then I saw all the work of God, that a man cannot find out the work that is done under the sun."
>
> Ecclesiastes 8:16, 17

Similar to most of the work done in this thesis, many of the research that have been done in database forensics explore the volumes of relational, functional and temporal information (Section 2.4.4) available in a database log file in describing techniques that can be used for database forensics analysis. For example, Litchfield [84, 85] explored how the redo logs in Oracle can be used for locating dropped objects and recovering data in an Oracle database. Fowler [59] describes how the transaction log in SQL Server database can be used to reconstruct data manipulation operations performed on the database. Frühwirt et al. [60, 61] explored the use of the redo logs in InnoDB for the reconstruction of data manipulation queries previously performed on the database. Although many databases maintain logs that are used for recovery from failures, and it can sometimes be assumed that the information required for a forensic analysis of a database is available in the logs, logging preferences set on a database system can usurp these facts. This is because, in practice, most DBMSs allow different levels of logging, including a default logging preference. Unfortunately, some of these logging preferences and usually the default logging preference may not be adequate for database

forensics analysis or reconstruction should the need to investigate the database arise. This is due to the fact that the logging preferences selected on a database affect the volume and relevance of the information that may be available in its log during a forensic investigation. For example, the technique used in handling log rotation or what should be done when a log becomes full depends on the logging preferences and this affects how much of past information is available during an investigation.

Another important issue about the use of log files for database forensics reconstruction, is that the use of log files for analysis seems to be a unifying factor for the various dimensions of database forensics discussed in Section 4.2. However, since the challenges faced in the investigation of the different dimensions and with the use of logs in each dimension differ, the logging preferences required in each dimension of database forensics may also differ from one another.

Since most of the techniques discussed in this thesis relate to the use of database log files for database forensics analysis and reconstruction, this chapter is dedicated to exploring the default logging preferences in database systems and how these preferences affect the information needed during a database forensics investigation. This is done by considering six of the popular DBMSs namely: MySQL, Microsoft SQL Server, Postgres, Oracle, DB2, and Sybase. The objective of the chapter is to highlight the information that are important for reconstruction and the appropriate logging preferences that will enhance reconstruction in the selected databases. In addition, the identified information requirements and logging preferences are considered in relation to each dimension of database reconstruction and the specific requirements necessary to ensure an effective forensics analysis and reconstruction in each of the dimensions are discussed. The chapter is based on a published paper [3].

9.1 Default Log

In this section, we consider six of the popular databases and describe their default logging preferences. Although the default preferences are often the same in different versions of

many of the database systems considered, the specific versions of the systems studied are MySQL 5.6, SQL Server 2008 and 2012, Postgres 9.2, Oracle 11g release 1 (11.1), DB2 10.1, and SybaseIQ 15.1. In addition, we describe only the logs that are considered most useful for forensic analysis in many of the systems. The effect of the default logging preference on the information that may be available for a forensic investigation of the database thereafter are also discussed in this section.

9.1.1 MySQL

MySQL is an open source relational database management system. The database software works as a client/server or embedded system with a multi-thread SQL server that supports various backends, client programs, administrative and other tasks. MySQL server is the main program in the DBMS that manages access to the data directory containing the databases, tables, and other information such as the log files [105]. MySQL server has five different log files, namely the error log, general query log, binary log, slow query log and the relay log. These logs store information about the activities taking place in the database.

The error log is used to store information about problems encountered when starting, running or stopping a MySQL server. Error logging is enabled by default on a MySQL database. The general query log is used to store information about the client connections that are established and SQL statements or queries received from the client. Queries written to this log are in the order in which they are received and this may be different from the order of their execution. All queries including statements that are only used to select or show data are stored when the general query log is enabled. In addition to enabling the log, the destination of the log file also needs to be set in order for the server to write queries received to the log file. However, the general query log is disabled by default. The slow query log is used to store queries that take more than a specified amount of time to execute (the default time is set as 10 microseconds). Similar to the general query log, statements are written to the slow query log file after their execution

and before the release of any locks on the database. Thus, the order of the log entries may be different from the order in which the queries are executed. The slow query log is disabled by default.

The binary log is used to store statements that change the data on a MySQL database. For example, it stores table creation operations and operations that change table data. The binary log also contains information about the duration of the queries that update data. Unlike the general query log, it does not store queries that do not modify data. Also, statements are logged after they are completed but before the database locks are released. Thus, the order of the statements in the binary log is the same as the order of execution. The format of the statements stored in the binary log is dependent on the logging format enabled (row-based, statement-based or mixed base logging) on the database [105]. The binary log is disabled by default. Although enabling the binary log on MySQL database makes performance slightly slower, it is important for two major purposes:

- replication of data changes on slaves,
- and recovery of the current instance of a database from previous backup point [105].

The Relay log is used to store data changes received from a replication server master [105]. It also stores statements that describe database changes and has the same format as the binary log file. The relay logs are automatically deleted once all the events in the file have been executed and it is no longer needed. The relay log is not enabled in the default setting of the server.

Quite a large volume of information can be collected from MySQL server logs for an investigation, if the logs are enabled. Unfortunately, the default logging preference on MySQL disables all the logs except for the error log (in the Windows operating system).In the event that the forensics analysis of a MySQL server is required, the error log contains virtually no information that can be useful for reconstruction or forensic

investigation of the database in general. It is important that a logging preference that is suitable for database forensics be enabled by the user or administrator. In Section 9.2, we describe a possible logging preference that allows an effective forensics analysis and the reconstruction of data on a MySQL database.

9.1.2 Microsoft SQL Server

The Microsoft SQL Server (henceforth referred to as SQL Server) is another relational database management system. A SQL Server installation contains a set of system databases as well as the user databases that are created as needed [115]. There are four default system databases that play distinctive roles in the operation of the SQL server database: the master database, the model database, the Tempdb and the MSDB database.

- Master: this is the heart of SQL Server. It contains information about logins, system configuration settings and data relating to other databases.
- Model: this is a database template that is used for all newly created databases on SQL Server.
- Tempdb: this is a database that is used as a preliminary storage for complex operations, intermediate results, temporary objects and stored procedures. The database is overwritten only at restart.
- MSDB: this holds a wide range of information about the database, including database backups.

The user databases are created for data storage and manipulation by the user. Similar to the system databases, a user database consists of two types of operating system files: the data files and the log files. The data files are used to store database objects such as tables and procedures while the log files keep record of the events that occur on the database. There are four main types of logs[1] that are available on the SQL Server database: the

[1]Other log files in Microsoft SQL Server include the SQL Server setup log and the SQL Server profiler log [106].

Windows event log, SQL Server agent log, SQL Server error log and the transaction log.

The Windows event log contains three useful logs that can be used for troubleshooting SQL Server errors; the application log which stores events that take place in the SQL Server and the SQL Server agent), the security log that stores authentication information, and the system log which stores service startup and shutdown information. The SQL Server agent log keeps record of any information, warning and error messages concerning the SQL Server agent and its operations. A new SQL Server agent log is created with a time stamp each time the server is started [106]. The SQL Server error log works in the same way as the SQL Server agent log and is used to store information about errors that occur during the database operations. For example, if a procedure refuses to startup.

The transaction log is arguably the most important log file and a critical component of a SQL Server database because it keeps a record of all data modification queries executed on the database in the order in which they occur. The database ensures that data is written to the transaction log files before they are effected on the database, thus transaction logs are useful for recovery from failed transactions and retrieval of a consistent version of the database in the event that the system crashes.

One advantage of the SQL Server is that, every database has a transaction log and logging is enabled by default. The size of the transaction log is also determined dynamically by the server and log truncation automatically occurs after certain events [95]. Looking at the volume of information stored in the different log files and the system databases, it is possible that data stored in a SQL Server can be reconstructed during its forensic investigation provided that the structure of the logs are understood. However, it might be an added advantage in some cases, if the system administrator can specify the size of the transaction log.

9.1.3 PostgreSQL

PostgreSQL (or Postgres) is an open source object-relational database management system that was derived from the POSTGRES package written at the University of California at Berkeley [131]. Postgres uses two main types of logs[2] both of which are helpful for forensics analysis. These are;

- Write-ahead logging (WAL) or transaction log, and
- Server log.

Write ahead logging (WAL) is a standard method in Postgres used specifically for maintaining data integrity. The WAL log is used to store changes that occur in data files before the changes are made on the data. This allows the recovery of a database from the WAL log in the event of a failure by retrieving changes that have not been applied to the data pages from the WAL log records.

WAL uses several configuration parameters that affect the volume of the information stored in the log, and as such the performance of the database [131]. One of the parameters of the WAL configuration is the *wal_level*. The *wal_level* parameter determines how much information is written to the WAL log and can be set either to *minimal*, *archive* or *hot_standby* at server startup. The *archive* setting allows the logging of information needed to recover from a crash and enables the archiving of the WAL logs. The *hot_standby* setting allows the storage of additional information that are required to reconstruct the status of running transactions in the WAL log and also allows logs to be archived. The *minimal* setting keeps only the information required to recover from a crash or immediate shutdown. It avoids the logging of bulk operations, in order to make the operations faster. The default *wal_level* setting on Postgres is the *minimal* setting. Unfortunately, the *minimal* setting of WAL does not contain enough information that can be useful for reconstructing data from the WAL logs or from the backups [131].

[2]Other log files in PostgreSQL include the event log and the server log [131].

The server log in Postgres is used mainly for error reporting, server messages, and logging of executed SQL statements. There are various parameters that determine what the database logs and when it logs in the server log. Some of the parameters that affect the volume of information that may be available in the server log for forensic investigations include the following [131]:

1. *log_min_messages*: this parameter controls the level of messages that are written to the Postgres server log. The possible settings are PANIC, FATAL, LOG, ERROR, WARNING, NOTICE, INFO, DEBUG1, DEBUG2, DEBUG3, DEBUG4, and DEBUG5. Every level logs all the information contained in the preceding level together with other additional information. The default setting is NOTICE.
2. *log_min_error_statement*: this parameter determines which SQL statements that generate an error condition will be included in the server log. The possible settings are the same as for the *log_min_messages* parameter and the default setting is ERROR, which means that only statements causing errors, fatal errors or panics will be logged.
3. *log_statement*: this parameter allows the specification of which SQL statements are logged regardless of whether or not an error occurs. The possible setting include **none** (no statement is logged), **ddl** (data definition queries are logged) and **mod** (all data definition and data modification queries are logged). The default setting is **none**, which implies that it is impossible to find queries executed in a Postgres database using a default log setting should a forensics analysis be required.

9.1.4 Oracle

The Oracle database is a relational database management system that includes some aspects of other paradigms such as hierarchical and object oriented models [104]. Although evidence can be collected from various sources in an Oracle database, for example, from the listener logs, alert logs, sqlnet logs, intelligent agent logs, and access logs [140], there are three major logs in Oracle that are of utmost importance for reconstruction in a

forensic analysis of the database; the redo logs, the archived redo logs and the alert logs.

The redo log in Oracle is the most important part of Oracle when it comes to recovery or forensics analysis. Every instance of an Oracle database has a redo log that serves as a protection of the database against failure. The log consists of two or more log files that are used to store all the changes made to the database as they occur. Records in the log files describe changes made to single blocks in the database. The data stored in the files can be used to identify all the changes made to the database as well as the undo segments [104]. The redo log is enabled by default on the Oracle database.

Groups of redo log files can be saved to offline locations by using the archived redo logs. The archived redo logs allow the recovery of a database in the event of a crash or disk failure. It also allow the updating of a standby database, and provide information about the history of a database when using the LogMiner utility in Oracle. There are two modes that can be activated to instruct the database whether or not to archive the redo logs:

1. NOARCHIVELOG: this mode disables the archiving of redo logs. Although this mode protects a database from instance failures, it does not allow the recovery of a database in the event of a system failure.
2. ARCHIVELOG: this mode enables the archiving of the logs and can be done manually or automatically.

The archiving mode is usually selected at the time of creation a database and is by default set to NOARCHIVELOG.

Alert logs and trace files are used to keep record of errors that occur in an Oracle database. The alert log keeps a chronological record of messages and errors involving data definition queries, administrative operation statements, errors relating to functions of shared servers and processes, and errors with values of initialization parameters that are different from the default values. Background processes write to a trace file when an internal error is detected by the process. The information written to a trace file may

be used by an administrator to tune applications and instances of the database. The default setting for trace files is the trace level 0, which disables archive log tracing.

9.1.5 DB2

IBM DB2 is a relational database management system that uses two main types of logs[3] in its operations:

- the database recovery log (or transaction log) and
- the diagnostic information log.

The database recovery log keeps record of all the changes made to the database, including created tables and updates on existing tables. It is useful for ensuring that the database can be recovered from a failure and that the database is in a consistent state [72]. In the event of a failure, the changes already made to the database but which are not yet committed are rolled back while the committed transactions that have not been written to disk are written by using the information available in the recovery logs, in order to preserve the integrity of the database [72]. The logging option can be set either as *circular logging* (which does not allocate more space for a log file when it becomes full, but overwrites the same file provided that the records in the file have been committed) or *infinite logging* (where full log files are not overwritten but are closed, and archived).

The diagnostic information log files consist of various log files that can be used to troubleshoot and diagnose errors and causes of errors in a DB2 database. The diagnostic information log files include dump files, trap files, administration notification log files and alert log files. These files are stored in a single directory and can be sorted or merged based on the date and time stamps included in the files. There are various parameters that can be used to control how much information is logged and how the log files are

[3]In Linux, UNIX and Windows operating systems, DB2 also offers error logs that are specific to each platform.

handled in DB2. Many of the default settings are already the best choices for reconstruction. However, we highlight some parameters of the default settings that may not be beneficial for reconstruction during a forensic investigation below:

1. *log_ddl_stmts*: this parameter is used to specify whether or not extra information regarding data definition statements should be written to the log. The default setting is NO, which implies that such statements are not logged.
2. *logarchmeth1*: this parameter is used to specify the log archiving method that will be used in the database. The possible settings include OFF and LOGRETAIN. The default setting is OFF which implies that logs are not archived and the database uses *circular logging*. Thus full logs are overwritten and the database is not roll-forward recoverable [72].
3. *logfilsiz*: this parameter is used to define the size of the log file. The possible range is from 4 to 1048572 (in kb) and the default is 1000.

9.1.6 Sybase

Sybase IQ (or Sybase) database is a relational database with a fundamental difference in that it focuses on the readers and not writers. That is, it is aimed at providing fast responses to queries from many users [128]. The two logs in the Sybase DBMS are the transaction log and the message log.

Sybase automatically handles the creation and deletion of its transaction log and the database server always requires a transaction log in order to run. A transaction log mirror (duplicate) is also maintained by the database. Although it is not required, the mirror provides an alternative source of information and may be useful in places where regular backups are not maintained [128]. The transaction log stores changes made to the database including, version information, free spaces and other information that may be useful for recovery and auditing of the database. Over time, the size of the transaction log may grow very large and it may be necessary to truncate the log. The method used to truncate the log is a decision made by the administrator, but can be done either by

stopping the database and making a copy of the log or by using the backup utility in Sybase.

A message log is created for every newly created database in Sybase. By default, Sybase logs error messages, status messages, insertion notification messages, and query plans in the message log file. The message log file grows to an unlimited size by the default setting and exists until the database is dropped. Archiving is not enabled by default in Sybase.

9.1.7 Summary of Default Log Settings

From the survey of the information logged by default in the popular databases described in Section 9.1.1 through Section 9.1.6, it is evident that the default logging configuration in many DBMSs does not generate logs containing enough information for a database forensics analysis and reconstruction. Some of the issues with the default logging configurations are summarized below.

- Not all the available logs on a DBMS are usually enabled by default. For example, among the six DBMS considered, only one DBMS (SQL Server) has all its logs enabled by default.
- In cases where some logs are enabled, the most valuable log (in terms of data used for forensic analysis) is sometimes excluded in the default configuration. For example, only the error log is enabled by default in MySQL, even though the other logs may contain more valuable data for reconstruction if they were enabled.
- The level of logging selected by default on many DBMS allows the storage of only a minimal amount of data specifically for crash recovery, or the log size is set to a default value as is the case with the SQL Server, Postgres and DB2 database systems.
- Although many DBMS support the archiving of logs or the use of an online storage, the default logging preference on most systems does not support the archiving of

logs and the logs are usually overwritten.

These issues reflect the need to specify the ideal log settings or the log data that are required for database logs to be effective for analysis and reconstruction in database forensics.

9.2 Ideal Log Settings

In this section, the information required in a database log file in order for it to be useful for reconstruction during a forensic analysis of the database are identified. A general view of the required set of information is given since the content of the log files in different databases differ and the settings on different databases are unique in most cases. The set of required information may be regarded as the ideal log requirements for reconstruction in database forensics. We also consider the identified requirements in relation to the operations of the databases discussed in Section 9.1 and determine the log settings and preferences that should be enabled in the databases in order to satisfy the requirements. It is important to note that the definition of an ideal log requirement does not imply that database log files that do not meet the requirements should not be included in the data collected for a forensic analysis. Although it may be more difficult to reconstruct information using log files that do not store all the information identified, the information available in such logs may still be useful and there may be additional techniques that can be employed for the forensics analysis of the databases.

9.2.1 Ideal Log Requirement for Database Forensics Reconstruction

In many databases, there is a trade-off between the performance of the database and the volume of information that can be stored in the log files used for recovery or for ensuring the consistency of the database in the event of a failure or other incidents. Unfortunately, the log files are often the richest source of information during the forensics analysis of a database. Ensuring that adequate logs are available on a DBMS serves as part of

a forensic readiness plan [8, 117] for future incidents that may involve the database. As such, it is important to ensure that vital information that can be used during the forensic analysis of a database exists in its log files, particularly in the transaction logs (or its equivalent) of the database. Other log files, such as the error logs and server logs should also contain as much information as possible that may be useful in the event of an incident.

The required information that should be stored in a database log can be determined by considering the objective of a database forensic analysis. According to the frameworks for digital forensic investigation process that have been developed by various researchers [11, 22, 74], the objective of the reconstruction phase of a digital forensic analysis is to draw conclusions from various pieces of information and provide answers to the what?, who?, when?, where?, why? and how? questions. Ieong [74] describes the FORZA framework which gives a technical overview of various roles in an investigation and how they are interrelated through these questions. In order to identify the set of information required for database forensics reconstruction, we also consider these questions and highlight four aspects that are usually of interest in database forensic analysis based on the questions. These aspects are described below:

- Identifying changes made to a database: this aspect corresponds to the what? question of the reconstruction process and involves the recognition of the information required to achieve this objective.
- Identifying who may be responsible for the changes: this aspect of database forensics deals with the who? and where? questions together with the recognition of the information necessary to achieve this objective.
- Confirming what we expect to see in the database: this aspect corresponds to the why? and how? questions of the reconstruction process. Information about why a database (or piece of information) is the way it is on the database and how this might have happened assists in confirming the information in the database.
- Determining the timeline of events on the database: the when? question of the

reconstruction process is handled in this category. Information about when an event occurred in a database is required to determine a timeline of events that occurred in the database.

As mentioned earlier, much of the previous work in describing the process of reconstruction or the forensics analysis of a database assume that the required information is available in the log files. Unfortunately, this is not always the case. In Section 9.1, some of the popular DBMS have been discussed and various default settings that do not allow a database to store the required information was highlighted. In order to determine the logging preferences or settings that would allow an effective database forensics reconstruction to take place in a database, we identify the information that should be stored in a database log by considering each of the aspects described above.

Identifying Changes made to a Database

Identifying the changes made to a database is arguably the most important aspect of a database forensic analysis. The reconstruction phase of database forensics relies mainly on the ability to identify what was done to the database. In order to identify changes to a database, the database log should contain the information below.

Data modification queries: The fact that many of the work that has been done on the forensic analysis of a database [84, 85, 60, 61, 59, 56, 53] rely on the log of queries that was executed on a database reflects its importance in database forensics. Since a database is typically usually modified by executing a query, information about queries that modify the data in a database should always be logged. While it is true that quite a number of databases maintain a transaction log, the volume of data stored is often minimal with the default setting. Logs of data modification queries should contain all the information needed to be able to reconstruct such queries. This is particularly useful for an investigator when there are no backups or when a value of interest occurred in between two backups during an investigation. In such a case, the inverse of the queries performed may be found

in order to reconstruct data in these cases [56, 53]. The information needed to maintain the consistency of a database and perform recovery should also be stored in the appropriate log files on a database.

Data definition queries: Unlike data modification queries, most databases do not keep record of data definition queries by default. This information is of utmost importance when reconstructing data because it gives an insight into what tables should or should not be present in a database and why. This functionality should be enabled by an administrator and information about the time of execution, the user who issued the query, and other details of data definition queries such as CREATE, ALTER, DROP, RENAME, TRUNCATE should be included in a log file. Details of unsuccessful queries in this case should also be included in the error logs. This information is also required in confirming the consistency of the information on the database with what is expected to be in the database.

Metadata changes: As mentioned earlier, queries executed on a database may give a false result due to changes made to the metadata, making it difficult for an investigator to comprehend some of the changes that might have been made to a database. Since the output from a database is dependent on the data stored in it and the metadata describing the data, it is possible to get incorrect information from queries if the metadata has been tampered with. As such, the changes made to a database metadata or other database files should be logged regardless of whether or not the changes were carried out by the administrator, a program or a malicious user. The logging of metadata changes will enable an investigator to get information about the previous state of the metadata, especially where a database rootkit is involved. In a paper by Beyers et al. [13], the authors describe techniques for obtaining a clean investigation environment of a database where one or more layers of the database metadata have been compromised. The logging of metadata changes in a database log file will also be a useful approach in identifying changes and obtaining a clean (or uncompromised) version of the metadata during

a database forensics analysis.

Identifying Who may be Responsible for Changes

Another aspect of database forensics analysis involves an attempt to identify who may be responsible for the changes made to a database. In order to achieve this objective, it is important that the following information are available in a database log.

Access information: In order to identify the users that are logged on or that may be responsible for an event that occur within a particular time frame, details of user logins and logouts of a database should be logged in the server log file (or its equivalent) of a database. Analyzing previous logins and logouts of a database may reflect certain patterns and assist an investigator in identifying anomalies in such patterns. The identification of anomalies can be useful in reconstructing events that occurred or the data that was stored in a database.

Data access queries: Although database access information may show the users that are logged in on a database at some time, it does not reflect who might have accessed a particular information. As discussed in Section 9.1, the default logging preference on most databases do not enable the logging of accesses to data on the database. This is an important information that may be useful in identifying intents when investigating an incident and as such should be logged. It may also be useful in recognizing when a piece of data was last accessed and who accessed it. Log records about data accesses should include at least the name of the table that was accessed, the user that issued the query, and the date and time stamp of when the query was performed. Details of unsuccessful accesses should be included in the error logs.

Changes in privileges: To understand log records about data access queries, it may necessary to know more about the privileges given to a certain user. Information regarding changes in the privilege given to anyone with access to a database should

be logged with the date and time of such changes. This enables an administrator to occasionally review privileges given to users and assist an investigator to identify possible culprits in the event of certain crimes, such as data theft.

Database administrator operations: Since the database administrator has access to all the information on a database, it is necessary to know the operations performed on the database by the administrator. Information about operations performed on a database by an administrator or anyone with similar access privilege should be logged, especially in cases where the operation may have a critical effect on the database. The definition of what is critical should be known to the database administrator and the database. This may be useful in identifying the operations performed by anyone or anything pretending to be the administrator when the database has been compromised.

Confirming the Information in the Database

This aspect of database forensics analysis deals with checking the consistency of the database with what the investigator or administrator expects to see on the database logs. Some of the information that should be available in the logs for this purpose are described below.

Server information: In order to confirm the information in a database, it is important that the state of the database can be determined. The log should contain information that will be useful in understanding failures, identifying why a particular log may be missing certain information, and any other issue that may arise during a forensics investigation of the database. Thus, events that occur during database start-ups, stops or restarts should be logged in a database log file. This information should reflect the conditions or configurations under which a database was started or restarted, as well as information about how the database was last stopped. Errors and messages displayed during these stages should also be logged.

Operational messages such as errors that occur during a backup operation or log archiving process (when enabled) should also be included in the database log files.

System failures and errors: Errors logs are the most popular log files that are maintained on almost every DBMS. Error logs can sometimes reflect abnormal conditions in data and show evidence of tampering. It can also be used for time analysis during an investigation. However, the volume of information stored in the error logs differs depending on what the database has been configured to log. Error logs should consist of details about system failures and other failed internal processes. This information may become useful in knowing the evidence to look for and where to look when conducting a forensic analysis of the database. An error should contain important pieces of information about each error, such as, the cause, the date and time of occurrence, and any failed attempt by the DBMS to correct the error.

Relocation of log files Many databases contain a default directory used for storing log files but also allow a different directory to be specified. Records of events that cause the relocation of log files will be useful for reconstruction and forensic analysis in cases where a malicious user has copied files from their original location to other places (or deleted them) to hide their activities. Therefore, events that cause changes in the default directory for storing log files, or the copying and/or deletion of log files should be logged in one of the database log files. This type of logging should also apply to the relocation of files that are expected to be in a particular location in a database.

Determining the Timeline of Events

As with digital forensics in general, creating timeline of events can assist an investigator to gain insight into the events that occurred and the people involved. It also assists in identifying patterns and anomalies that may reveal other sources of evidence in a system. All the ideal log information described earlier are useful in the creation of a timeline of

events. An additional information that is required for the creation of a timeline of events is the archive of all the information identified above.

Archived logs: In many databases, the size of the log files is kept to a certain maximum by overwriting the file from the top when it becomes full (for example, with circular logging in Oracle). Archive of database log files should be maintained rather than allowing overwrites. Archives are very useful in reconstructing data since a single log file may not contain all the information of interest. Log archives provide a rich source of information for an investigator regardless of how long ago the data of interest might have existed or have been manipulated on the database.

9.2.2 Challenges and Solution for Ideal Logging Preferences

Logging of all the information identified above doubtlessly creates various challenges that affect the operations and the investigative process of a database. Some of the challenges that may be faced when logging all of the required information are highlighted below.

- Effect on database performance: in most databases, logging causes an overhead that affects query performance. Although error logs and server logs (or their equivalents) have little effect on a database, transaction logs and/or audit logs can significantly reduce the speed of a database in completing queries due to the number of records that may need to be written to such logs before it is completed or updated on the database. Despite this, the increasing need to ensure data security and provide a means of reconstructing data requires that adequate log records are kept at all times.
- Volume of information in log files: One of the challenges of handling digital evidence is the volume of information that can be retrieved during an investigation. Logging all the information mentioned earlier implies that the same problem is encountered in database forensics. Tools and techniques that can be used to extract useful and relevant information from the logs will be required in order for the logging process to be worthwhile. Even in the absence of analysis tools, database log records are

very useful for forensics analysis, although more effort will be required for a manual analysis.

- Log file formats and contents: Another challenge with handling database log files is the various file formats used in different databases. In addition, some databases also provide the option of logging into tables on the database itself. Thus, it is required that an investigator understands the format used by the database being investigated and how to handle data in such format. Also, data records stored in the log files of different DBMSs differ both in organization and in length. For example, certain information about a similar query may be logged in one DBMS and not logged in another even when similar settings are used. This makes the automation of log analysis an issue that has to be handled with a specific DBMS in mind.

In order to handle these challenges or minimize their effects on a database forensics analysis, some of the techniques that can be employed are described below:

- Storing log files on separate physical mediums: a possible way of handling the volume of log files and its effect on database performance is to store archived log files on a different physical medium from the database itself. This is beneficial particularly in systems that use different data access formats for the logs and the data files. For example, in Sybase, the catalog and the IQ store (which contains the data files and data dictionary) are randomly accessed while the transaction log is sequentially accessed. An additional advantage of this technique is that it also preserves the logs against loss or attack on the database and provides a way of retrieving the logs for analysis without affecting the operation of the database. A relatively similar alternative to this approach is to take regular backups of the log files before it is overwritten by default.
- Use of log analysis and/or log management tools: to enhance the analysis of the volume of information that may be retrieved from log files during database forensics, the analysis process should be automated. Many databases provide tools that

can be used for analyzing the log files. For example, the Log Analysis tool in DB2 [71], the Oracle LogMiner [138], and the Microsoft SQL Server Profiler [115]. Very few external tools are also available for the analysis of specific databases, for example, the pgFouine[4] tool that can be used for analyzing Postgres logs. These tools provide an efficient way of analyzing logs, even though they can only be used in the specific DBMS for which they were designed. Tools and techniques used in the general field of digital forensics may also be useful for analyzing logs in specific formats in some cases. Chuvakin [35] discussed the need for a more advanced database log management tool that is capable of working across various DBMSs and automating not only the process of log analysis, but also the process of collection, transferring and storing of log files. The availability of an advanced log management tool will alleviate the problem of handling and combining logs in tables and in files. It is also important that such tools have the capability to reconstruct data based on the information in the log files.

- Occasional review of logs and auditing: an occasional review of logs is required in order to provide insight about what information should be excluded from the logs or needs to be included. This may assist in reducing the volume of irrelevant data in logs during an investigation. In addition, the auditing feature (when available) of a database should be enabled as it offers a level of protection for log files and other database files, and enhances the early detection of suspicious activities on a database.

9.2.3 Identification of Possible Ideal Logging Preferences

In this section, possible logging preferences that may be used to ensure that the required information is present in the log files of the popular databases considered in Section 9.1 are identified. These preferences are meant to explain possible settings that can be enabled (which are not enabled by default) and do not imply that the settings are the only ideal settings. A definition of which information (among the log requirements discussed

[4]http://pgfouine.projects.pgfoundry.org/

above) is of utmost importance in a specific database application will be required in realizing the ideal logging preferences in any particular DBMS. The possible settings for the databases are considered below.

MySQL: MySQL server consists of various logging options (all disabled by default) that can provide a lot of information. To explore the capability of these logs for forensic analysis, the logs should be enabled. The slow query log may be viewed as a subset of the general query log and may be excluded. However, the General query log, the Binary log and the Relay log should be enabled on a MySQL server. The server automatically deletes relay logs once they are no longer needed, so the space required for relay logs and their effect performance is minimal. The size of the general query log and the binary log may grow large over time; some of the techniques described earlier (Section 9.2.2) may be employed to combat the space requirement problem.

Microsoft SQL Server: As mentioned earlier, logging is enabled by default on SQL server. However, a default size is specified for the log file. It may be necessary to increase the size of the log, especially the transaction log in some cases. The *autoshrink* option, which allows the shrinking of specified files on the database, is also set to false by default. While it is not advisable to shrink data files on a database, the log files can be configured to shrink when they become big. To handle the problem of transaction logs growing big over time, the use of backups or a different physical medium for logs can be explored to ensure that performance is not significantly hampered.

Postgres: In order to exploit the potential capability of the WAL logs in reconstructing the information on a Postgres database, the *wal_level* setting for the WAL log should be set either to the *archive* or the *hot_standby* setting before starting up the database. In addition, the *log_min_messages*, *log_min_error_statement* and the *log_statement* parameters of Postgres should be given values that allow the logging of enough information about queries. For example, the *log_statement* parameter

should be set to **mod** so that all data definition and data modification queries can be logged.

Oracle: The ARCHIVELOG setting in Oracle should be enabled to exploit the rich source of information that archived logs provide. The auditing capability of the database should also be turned on. The default archive logs trace level of 0 (which disables archive log tracing) should also be changed to 1 (by setting the parameter LOG_ARCHIVE_TRACE = 1) so that information or errors about the process of archiving log files can be retrieved.

DB2: In order to be able to reconstruct data definition statements, the *log_ddl_stmts* parameter in DB2 should be changed from the default NO value to YES. The *logarchmeth1* parameter which is used to specify the log archiving method is also set to OFF by default. This makes it impossible to access earlier log records as they are overwritten when using this default setting. A better alternative value for this parameter is the LOGRETAIN setting, which specifies that log files should be retained and become online archived log files which can be subsequently used for roll-forward recovery [72]. If performance is an issue, and the LOGRETAIN option cannot be used, the default size of the log should be increased as much as possible in order to ensure that much information can be retrieved from the logs when required. The use of a different physical medium for log archives can also be explored.

Sybase: Using the default settings in Sybase, message logging stops if the disk becomes full while writing a message to the log and a record of this is made in the server log. Although message logging also resumes by default when the error condition is resolved (by creating a new log file), it would be better to avoid log files growing indefinitely and instead archive relatively shorter log files. This makes it easier to recognize relevant log files and is also beneficial in the event of a system crash or disk failure. In addition, rather than stopping the database to truncate the transaction log, the backup facility in Sybase should be employed. This allows

the availability of more information in the event that a live forensics [24] of the database needs to be conducted.

9.3 Database Forensics Reconstruction Dimensions and Log Files

In this section, each of the required set of information identified in Section 9.2.1 is considered in relation to the three dimensions of reconstruction in database forensics [55]. This is to ensure that the required information is applicable in the various dimensions and that additional requirements for any of the information are identified in each dimension.

9.3.1 Modified Databases and the Ideal Log Setting

Amongst the three dimensions of database forensics reconstruction, the modified database category is able to provide the most readily usable log files, since the database (and therefore, the log files) have not be compromised or damaged, and requires no additional steps in order to obtain the log files. As such, the required set of information described in Section 9.2.1 completely applies to a modified database. One of the advantages of storing these required set of information in the log of a modified database is that the information in the log may reflect inconsistencies that expose a compromise or damage to the database during an investigation, such that a database initially considered to be a modified database may eventually have to be investigated as a compromised and/or a damaged database. When the objective of the forensics analysis of a modified database is to retrieve data that existed as some earlier time, and an archive of logs is available, it may be necessary to identify and analyze only the relevant logs, so as to reduce the volume of information that needs to be processed.

9.3.2 Compromised Database and the Ideal Log Setting

The main concern during the investigation of a compromised database is that the information provided by the database cannot be trusted. This concern may also affect the

use of some or all of the information in the log files on the compromised database. In order to ensure that the data in the logs have not been compromised, it is important that an investigator checks the consistency (or inconsistencies) of the logs and also the hypotheses made based on the log records [43]. To check that a log is consistent, the investigator may need to confirm that the log reflects only the information that was selected to be logged in the configuration, and that the data formats of the files are as expected. Checking the consistency of hypotheses made may involve an attempt to re-generate the current instance of a database from a reconstructed version of the database. If the regenerated instance differs from the actual database instance, then it shows that some information might not have been logged or there might have been some compromise of the database. This process may also be useful in identifying the events or steps taken by an attacker when investigating an incident.

On the other hand, there is a large amount of data to support the investigation of a compromised database if the required set of information described in Section 9.2.1 are logged. For example, if changes to a database metadata are logged, then it is possible for an investigator to identify the changes made to compromise a database and when the changes were made during an analysis. Also, the logging of data definition queries is useful for identifying changes made to the tables in a database. Other information in the log also provide bits and pieces of data that can be used by an investigator.

9.3.3 Damaged/Destroyed Databases and the Ideal Log Setting

One of the challenges of investigating a damaged or destroyed database is the fact that the data files or log files of the database may have been modified, deleted or moved into other locations different from the expected locations. Similar to the use of log files in the investigation of a compromised database, it is important for an investigator to check for any inconsistencies in the hypotheses made based on the log records [43] in cases where the database is still operational. Differences in the reconstructed database instance and the actual instance of the database may be useful in identifying missing information in

the log file or sections of the log file that have been modified. In cases where the database is no longer operational or where the log file has been deleted, file carving techniques [21] may be used in regenerating the log file (or other files) of the database for forensics purposes.

Some of the advantages of having the required set of information described in Section 9.2.1 during the investigation of a damaged database include the fact that if the information about the relocation of log files are stored, then the log file can be easily located when it is moved to a different location. Information about events relating to database startups, stops and restarts (server information) is also a good pointer to the possible intents of a database attacker. Access information and metadata changes recorded in the log file serves as a valuable source of information for identifying what was changed and by who during database forensics. The log of changes in privileges allows an investigator to identify privileges that should not have been given, and this may be useful in identifying the intent of possible culprits. It is possible that the damage to a database was carried out by an attacker pretending to be an administrator, the log of database administrator operations will be particularly useful in this case.

In summary, the required set of information for database forensics described in Section 9.2.1 applies to all the dimensions of database forensics. Although additional steps may be required to obtain or use the log files from compromised and/or damages databases, the information ensures that as much information as possible is available to assist an investigator in carrying out the forensics analysis of a database and reconstructing data.

9.4 Summary

Although many database systems contain logs that can be very valuable for recovery and forensic analysis of the database, the effectiveness of the logs is affected by the amount of information stored in it, which is often determined by the logging preferences enabled on the database. This chapter examines the application of the various techniques described throughout the thesis by considering the effective of database logs for forensic analysis,

since it constitutes a major part of the thesis. The chapter summarizes a detailed study of the default logging preferences on six popular databases and reveals the inadequacy of the logs generated through the default logging preferences in many DBMSs. It shows that the information required for a database forensics analysis and reconstruction is not always stored in logs generated using these preferences. In order to identify an ideal set of log requirements for database forensics analysis and reconstruction, the various aspects that are usually of interest in the forensic analysis of a database are considered. The information required to handle the different aspects compose the ideal log requirement for the forensic analysis and reconstruction of a database. In addition, the chapter discusses the challenges and solutions for obtaining an ideal log for forensic analysis from a database and describes the possible preferences that can be used to obtain the ideal log in the popular DBMSs considered.

Lastly, even though the focus of this thesis and many of the techniques discussed therein is on the process of reconstruction in the modified databases dimension of database forensics, this chapter considers the identified set of required information in the three dimensions of reconstruction described earlier (Section 4.2). Where necessary, additional requirements for each of the dimensions are identified. It is important to note that in certain instances of forensic analysis involving a compromised or a damaged database, additional steps or techniques other than those described in this thesis may be required to retrieve or use the database logs.

Chapter 10

Conclusion

> "Of making many books there is no end, and much study is wearisome to the flesh. Let us hear the conclusion of the whole matter: Fear God and keep His commandments, for this is the whole duty of man."
>
> Ecclesiastes 12:12-13

The work presented in this thesis deals with the reconstruction of the information stored in a database during the forensic analysis of the database. Although such information might have existed at some earlier time, deleted or updated in various ways, this thesis presents various concepts and techniques that can be used for reconstructing the data (and metadata) in a database during a forensic analysis of the database. The thesis is focused on the modified databases category of database forensics but also gives some directions relating to other dimensions of database forensics.

This chapter concludes the thesis by evaluating the extent to which the objectives stated in the problem statement have been accomplished throughout the thesis. It also highlights the main contributions of the thesis and lastly gives directions for future research that can be done in relation to the thesis.

10.1 Thesis Summary

The goal of this thesis was to address one of the main concerns of database forensics analysis, which is the lack of knowledge on how the information stored in a database can be reconstructed, particularly after several modifications of the information might

have occurred. In order to achieve this goal, the problem statement given in Section 1.2 describes the questions that need to be answered. A summary of the thesis, as well as an indication of how the questions in the problem statement have been answered are given in this section.

Chapters 2 and 3 give the necessary foundation for the remainder of the thesis and discuss the necessary literatures on digital forensics examination and database management systems, respectively. Chapter 2 discusses the nature of digital evidence and describes the general concept of reconstruction in digital forensics. In Chapter 3, a discussion of database systems is presented with emphasis on the characteristics of databases, the relational model of database systems and the relational algebra which is a core part of the thesis.

The answer to the first question of the problem statement, **What are the different categories of databases that may be encountered during a database forensics analysis?**, is presented in Chapter 4 of the thesis. The chapter gives an overview of the concepts of database forensics and describes the three dimensions that exist in database forensics analysis by considering existing literatures. The literatures relating to each dimension are discussed and an analysis of the database forensics problem space is given. The chapter also describes the database forensics process and the challenges involved in detail.

Chapter 5 provides the answer to the next two questions mentioned in the problem statement. That is, **What are the techniques that can be applied for reversing data manipulation operations performed on a database?** And **how can an algorithm for database forensics reconstruction be developed?** The chapter introduces three main techniques (Inverse relational algebra, relational algebra log and value blocks) that exploits the mathematical foundation of relational algebra and which can be used in describing an algorithm for reconstruction in database forensics. The reconstruction algorithm is then described and its use illustrated with examples.

In Chapter 6, the developed algorithm is proven to be correct using mathematical proof

techniques. This answers the fourth question stated in the problem statement, that is, **Can such an algorithm be proved to be correct?** The chapter highlights the fact that a log in SQL notations can be converted in a relational algebra format and that using such log, the output generated when using the reconstructed algorithm described in the thesis is correct. The proof is divided into series of lemmas that address different portions of the algorithm, each of which are proven to be correct.

Chapters 7 and 8 of the thesis deals with ways of ensuring that information can still be reconstructed from a database even in some "difficult" situations such as instances where some information might have been deleted or where the schema has been modified to disrupt the operation of the database. Chapter 7 explains the limitation of the reconstruction algorithm and describes various approaches of ensuring the information that is as complete as possible can be reconstructed despite the generation of partial inverses due to deletion or any other operations performed on the database. Chapter 8 shows examples of how a database schema can be compromised and discuss the different approaches that can be used to reconstruct the schema in such conditions so that the data stored in the database can be retrieved or reconstructed as necessary.

The last question of the problem statement, **What information are required for an efficient database forensic analysis or reconstruction?** is answered in Chapter 9 of the thesis. The required log information as well as the necessary logging preferences to ensure that an effective reconstruction can be carried out are identified. The requirements were based on an analysis of the default log settings of some popular DBMSs and an analysis of the information required to address recognized aspects of a database forensics analysis. The set of required information are considered in all the dimensions of database forensics and additional requirements or steps that may be necessary for each dimension are identified. The analysis in Chapter 9 reflects the condition of real life systems and as such gives a good indication of the information that should be present in a log for reconstruction purposes.

To conclude the thesis, the main contributions are described in the next section.

10.2 Main Contributions

The main contribution of this thesis, which also addresses the overall goal of the thesis is the development of an algorithm for reconstruction in database forensics. To achieve this objective, the thesis first considers the various dimensions of reconstruction that exist in database forensics. The identification of the three dimensions of database forensics analysis and how they form part of a three-dimensional problem space is a novel contribution to the literature which has not been previously addressed.

Another contribution of this thesis is the definition of the inverse operators of the relational algebra, as well as the concepts of relational algebra log and value blocks. These concepts provide a way of dealing with the complexity of handling and reversing data manipulations operations performed on a database. It contributes to the field of database systems by adding to knowledge in relational algebra.

In addition, a major contribution of this thesis is that it presents a proven algorithm for reconstructing the data in a modified database and also introduces various techniques for reconstructing the schema of a database. To the best of our knowledge, this is the first algorithm for database forensics reconstruction that exists. Although some literatures have considered technical aspects of individual DBMSs, this thesis is the first to investigate a significant number of DBMSs (relational databases) and formulate an algorithm that can be used for reconstruction during their analysis.

The final contribution of this thesis is the identification of the ideal log requirements and log setting that is necessary for an efficient reconstruction process in the various dimensions of database forensics analysis. The identified requirements is new to literature and can be useful in ensuring that a database is in a state that allows a forensic analysis and/or reconstruction to be carried out should the need arise.

Most of the main contributions highlighted above have been individually compiled as sub-research reports and submitted to either a relevant journal or conference. The title of most of the work that have been accepted or with sections that are included verbatim

in this thesis are given below:

- On dimensions of reconstruction in database forensics [55]
- Reconstruction in database forensics [56]
- Database forensics [54]
- Correctness proof for database reconstruction algorithm [53]
- On completeness of reconstructed data for database forensics [2]
- Schema reconstruction in database forensics [4]
- Ideal log setting for database forensics reconstruction [3]

10.3 Future Research

The field of database forensics can be considered to be relatively new. As such, there are still different aspects of research that can be considered in this field. In relation to the process of reconstruction in database forensics, some of the future work that can be considered are described below.

- As mentioned in Section 4.2, the problem of reconstruction in database forensics consists of three dimensions. The reconstruction approaches presented in this thesis focus mainly on one of these dimensions - that of modified databases. Research is still required on how to reconstruct information in the compromised and the damaged databases dimension of database forensics in order to give a complete description of reconstruction in all dimensions of database forensics. Although Section 8.1 briefly describes how a database can be compromised, this was done with a view to address the problem of schema reconstruction. Future research can involve adapting the reconstruction algorithm presented in this thesis to other dimensions of database forensics or developing a new algorithm that is more suited for these dimensions.
- In Section 5.1, we highlighted some techniques that can be used for data restoration

in databases. Although these techniques may not be adequate for generating all the information often required during a database forensics analysis, it may be possible that there are specific conditions in which using a particular restoration technique may be more beneficial. An empirical research on which data restoration technique or whether database forensics analysis is more suited for a particular problem can be conducted. Intuitively speaking, data restoration techniques would be inadequate when investigating a compromised or damaged database, but this can also be studied.

- In Chapter 7, we discussed the limitation of the database reconstruction algorithm, which arises from the fact that the inverse operators of the relational algebra may generate incomplete data. We also described various techniques that can be applied to generate more complete data during a reconstruction process. However, the research does not investigate whether or not the data generated by incorporating the techniques described is maximal, that is, if the log contains no further information that can be used to reconstruct values. Further research can be conducted in this regard to determine whether there are additional techniques that can be used to reconstruct more complete data and if or when such data are maximal. Conditions in which a particular approach may be a better option can be also be investigated.
- One of the approaches for schema reconstruction discussed in Section 8.2.3 involves checking the consistency of the information stored in the database. Although we have highlighted situations in which this approach may be used and given high-level examples to illustrate its use, empirical research can be conducted to measure the effectiveness of this approach. Detailed real life examples that illustrate the application of the approach can also be examined. The effectiveness of the other approaches described in Section 8.2.3 can also be measured against each other by considering practical scenarios requiring the reconstruction of a database schema.
- Although the problem of reconstruction in database forensics as well as the need to better understand database forensics analysis process spans across different models of database systems, this thesis is specifically focused on the relational model

of database systems due to their popularity and mathematical foundation. The research can be extended to other models of database systems. In addition, the problem of reconstruction and database forensics in general can also be considered in other environments or technologies such as NoSQL, NewSQL, Big Data, cloud or distributed databases environment.

- In Section 9.2.2, we identified the use of log management or log analysis tools as a way of enhancing the analysis of database logs and automating the analysis process. A survey of the capabilities of currently available database log management tools can be conducted to measure the effectiveness of such tool across different DBMSs. This can also be beneficial in determining the capabilities that should be incorporated into new log management tools and enhancing the development of more advanced tools.
- Lastly, although this thesis is focused on the theoretical detail of how reconstruction can be done in database forensics, an implementation of the different techniques discussed in the thesis is still required. Future research will investigate the implementation and usage of the algorithm on a live database.

Appendix A

SQL Log Conversion to RA Log

A listing of the SQL log converted to the relational algebra log in figure 5.1 is given below. The notation used in the conversion is also given thereafter.

```
CREATE TABLE Employees (EmpNo INTEGER NOT NULL primary key, DeptNo varchar(10) not null ,
LastName varchar(30), Initials varchar(5), DateOfBirth DATE NOT NULL, salary integer);

insert into Employees values
(11334330, 'CS1', 'FASAN', 'O M', TO_DATE( '21-06-1995', 'DD-MM-YYYY' ),20000),
(11322560, 'GEO1',  'ADEOLU', 'P Q', TO_DATE( '01-09-1965', 'DD-MM-YYYY' ),12000),
(13234010, 'PHY1', 'MATTHEW', 'K M', TO_DATE( '31-12-1987', 'DD-MM-YYYY' ),18000),
(24143130, 'MATHS1', 'CELLIERS', 'M K', TO_DATE( '13-07-1991', 'DD-MM-YYYY' ),8000),
(13456170, 'CS1', 'NUCKLARD', 'A S',  TO_DATE( '21-08-1990', 'DD-MM-YYYY' ),10000);

select * from Employees;
DROP TABLE Departments CASCADE;
CREATE TABLE Departments (DeptNr varchar(10) Primary key, DeptName Char(40), BuildingName varchar(30));

DROP TABLE Dept_Emp CASCADE;
CREATE  TABLE DEPT_EMP
(EmpNo integer not null, DeptNo varchar(10) not null, FromDate date, ToDate date, primary key(EmpNo));

insert into Departments values ('CS1', 'Computer Science', 'IT Building'),
('GEO1','Geography','Earth building'),('PHY1','Physics','Physics lab'),
('MATHS1','Maths science','Sceince building');

insert into Dept_Emp values
(11334330,'CS1',TO_DATE( '02-08-2011', 'DD-MM-YYYY' ), TO_DATE( '31-12-2014', 'DD-MM-YYYY' )),
(11322560,'GEO1',TO_DATE( '02-07-2010', 'DD-MM-YYYY' ), TO_DATE( '31-12-2012', 'DD-MM-YYYY' ));
select * from Departments;

DROP TABLE Emp_salary1 CASCADE;
```

```
select EmpNo,LastName, Initials, Salary into Emp_salary1
from Employees where salary>10000 order by salary desc limit 1;

DROP TABLE Emp_Salary2 CASCADE;
select EmpNo,LastName, Initials, Salary into Emp_salary2
from Employees where salary>10 order by salary limit 1;
select * from Emp_Salary2;

DROP TABLE BestWorseSalary CASCADE;
create table BestWorseSalary as(select * from Emp_Salary1 union select * from Emp_Salary2);
select * from BestWorseSalary;

insert into Emp_salary1 (select EmpNo,LastName, Initials, Salary from Employees);
select * from Emp_Salary1;

drop table DeptNoName cascade;
create table DeptNoName as SELECT * FROM Employees AS a INNER JOIN Departments AS b ON a.DeptNo = b.DeptNr;
select * from DeptNoName;

update departments set BuildingName = 'GeoScience Building' where DeptNr = 'GEO1';
select * from departments;

drop table DeptJoinEmp;
create table DeptJoinEmp as select * from Employees natural join Departments;
select * from DeptJoinEmp;

drop table DeptEmpIntersect;
create table DeptEmpIntersect as (select * from DeptJoinEmp intersect select * from DeptNoName);
select * from DeptEmpIntersect;

delete from DeptNoName;

insert into DeptNoName (select * from DeptJoinEmp EXCEPT select * from DeptEmpIntersect);
select * from DeptNoName;

alter table Employees rename column DeptNo to DeptNr;
delete from DeptEmpIntersect;
select * from DeptEmpIntersect;
```

Notation used in Conversion to RA Log

Employees - A

Departments - B

Dept_Emp - C

Emp_Salary1 - D

Emp_Salary2 - E

BestWorseSalary - G

DeptNoName - H

DeptJoinEmp - I

DeptEmpIntersect - J

Appendix B

Acronyms and Symbols

DBMS Database Management System

RA Relational Algebra

US United States

UK United Kingdom

$p(A)$	Logical predicate involving attribute A
$R(A)$	Relation R with attribute A
R_r	Reconstructed relation R
R_t	Relation R at a particular time
R^*	Incomplete relation R
V_{D_i}	The i^{th} value block of relation D
$V_{D_i}[1]$	The first query of the i^{th} value block of relation D
$\times$	Cartesian product operator
$\cup$	Union operator
$\cap$	Intersection operator
$-$	Difference operator
$/$	Division operator
$\bowtie$	Join operator
ϕ	Projection operator
σ	Selection operator
ρ	Rename operator

Bibliography

[1] Access Data. Forensic toolkit. `www.accessdata.com`. Accessed 25 February, 2013.

[2] O. M. Adedayo and M. S. Olivier. On the completeness of reconstructed data for database forensics. In *Digital Forensics and Cyber Crime*, volume 114 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 220–238. 2013.

[3] O. M. Adedayo and M. S. Olivier. Ideal log setting for database forensics reconstruction. *Digital Investigation*, 2014. Manuscript submitted.

[4] O. M. Adedayo and M. S. Olivier. Schema reconstruction in database forensics. In *Tenth Annual IFIP WG 11.9 International Conference on Digital Forensics*, Vienna, Austria, January 2014. (In press).

[5] Association of Chief Police Officers of England (ACPO), Wales and Northern Ireland. *Good Practice Guide for Computer-based electronic evidence, Version 4*. ACPO E-Crime working Group, April 2008.

[6] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, J. Gray, W. F. K. III, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, P. Tiberio, I. L. Traiger, B. W. Wade, and R. A. Yost. System R: A relational data base management system. *IEEE Computer*, 12(5):42–48, 1979.

[7] J. Austen. Some stepping stones in computer forensics. *Information Security Technical Report*, 8(2):37 – 41, 2003.

[8] D. Barske, A. Stander, and J. Jordaan. A digital forensic readiness framework for South African SME's. In *Information Security for South Africa (ISSA)*, pages 1–6, 2010.

[9] M. V. Baryamureeba and F. Tushabe. The enhanced digital investigation process. In *Digital Forensic Research Workshop*, Balltimore, Maryland, August 2004.

[10] N. Beebe. Digital forensic research: The good, the bad and the unaddressed. In *Advances in Digital Forensics V*, volume 306 of *IFIP Advances in Information and Communication Technology*, pages 17 – 36. Springer Berlin Heidelberg, 2009.

[11] N. L. Beebe and J. G. Clark. A hierarchical, objectives-based framework for the digital investigations process. *Digital Investigation*, 2(2):147 – 167, 2005.

[12] T. Bevel and R. Gardner. *Bloodstain pattern analysis: with an introduction to crime scene reconstruction. 2nd Edition.* CRC Press, Boca Raton, FL, 2002.

[13] H. Beyers, M. Olivier, and G. Hancke. Assembling metadata for database forensics. In *Advances in Digital Forensics VII*, volume 383 of *IFIP Advances in Information and Communication Technology*, pages 89 – 99. Springer Berlin Heidelberg, 2011.

[14] H. Beyers, M. Olivier, and G. Hancke. Arguments and methods for database data model forensics. In *Proceedings of the 7th International Workshop on Digital Forensics and Incident Analysis (WDFIA 2012)*, pages 139 – 149. Plymouth University, 2012.

[15] C. Binnig, D. Kossmann, and E. Lo. Reverse query processing. In *IEEE 23rd International Conference on Data Engineering (ICDE)*, pages 506–515, April 2007.

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

[17] N. Bruno and S. Chaudhuri. Flexible database generators. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1097–1107, 2005.

[18] F. Buchholz and E. Spafford. On the role of file system metadata in digital forensics. *Digital Investigation*, 1(4):298 – 309, 2004.

[19] T. Burns, E. Fong, D. Jefferson, R. Knox, L. Mark, C. Reedy, L. Reich, N. Roussopoulos, and W. Truszkowski. Reference model for DBMS standardization. *SIGMOD Record*, 15(1):19–58, March 1986. `http://doi.acm.org/10.1145/16342.16343`.

[20] B. Carrier. Defining digital forensic examination and analysis tools using abstraction layers. *International Journal of Digital Evidence*, 1(4), 2003.

[21] B. Carrier. *File system forensic analysis*. Addison-Wesley Professional, Upper Saddle River, NJ, 2005.

[22] B. Carrier and E. H. Spafford. Getting physical with the digital investigation process. *International Journal of Digital Evidence*, 2(2):1 – 20, 2003.

[23] B. D. Carrier. *A Hypothesis-based Approach to Digital Forensic Investigations*. PhD thesis, Purdue University, West Lafayette, Indiana, May 2006.

[24] B. D. Carrier. Risks of live digital forensic analysis. *Communications of the ACM*, 49(2):56–61, February 2006.

[25] B. D. Carrier and E. H. Spafford. Defining event reconstruction of a digital crime scene. *Journal of Forensic Sciences (JFS)*, 49(6):1291–1298, November 2004.

[26] D. L. Carter. Computer crime categories: How techno-criminals operate. *FBI Law Enforcement Bulletin*, 64(7):21 – 27, July 1995.

[27] E. Casey. Error, Uncertainty and Loss in Digital Evidence. *International Journal of Digital Evidence*, 1(2), 2002.

[28] E. Casey. *Handbook of Digital Forensics and Investigation*. Elsevier Academic Press, 2009.

[29] E. Casey. *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*. Academic Press, 3rd edition, 2011.

[30] D. D. Chamberlin. Early history of SQL. *IEEE Annals of the History of Computing*, 34(4):78–82, 2012.

[31] D. D. Chamberlin and R. F. Boyce. SEQUEL: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, SIGFIDET '74, pages 249–264, 1974.

[32] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC, pages 77–90. ACM, 1977.

[33] C. Chaski. The keyboard dilemma and authorship identification. In P. Craiger and S. Shenoi, editors, *Advances in Digital Forensics III*, volume 242 of *IFIP The International Federation for Information Processing*, pages 133–146. Springer New York, 2007.

[34] W. J. Chisum and B. Turvey. Evidence dynamics: Locard's exchange principle & crime reconstruction. *Journal of Behavioural Profiling*, 1(1), January 2000.

[35] A. Chuvakin. Introduction to database log management. Technical report, LogLogic Inc, August 2007. `http://www.infosecwriters.com/text_resources/pdf/AChuvakin_DB_Logging.pdf`. Accessed, 19 March 2013.

[36] S. O. Ciardhuáin. An extended model of cybercrime investigations. *International Journal of Digital Evidence*, 3(1), 2004.

[37] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377 – 387, June 1970.

[38] E. F. Codd. *The Relational Model for Database Management, Version 2*. Addison-Wesley, 1990.

[39] F. Cohen. *Digital Forensic Evidence Examination - 2nd Ed.* Fred Cohen & Associates, 2010.

[40] F. Cohen. Attribution of messages to sources in digital forensics cases. In *43rd Hawaii International Conference on System Sciences (HICSS)*, volume 0, pages 1 – 10, January 2010.

[41] F. Cohen. A note on detecting tampering with audit trails, 1995. Fred Cohen and Associates.

[42] F. Cohen. Challenges to digital forensic evidence. `http://all.net/Talks/CyberCrimeSummit06.pdf`, October 2006. Fred Cohen and Associates.

[43] F. Cohen. *Digital Forensic Evidence Examination - 3rd Edition*. Fred Cohen & Associates, 2011.

[44] P. Cousot. Methods and logics for proving programs. In *Formal Models and Semantics*, volume B of *Handbook of Theoretical Computer Science*, pages 843–993. 1990.

[45] J. P. Craiger, J. Swauger, and C. Marberry. Digital evidence obfuscation: recovery techniques. In *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense IV*, volume 5778, pages 587–594, August 2005.

[46] Data Base Task Group. A survey of generalized data base management systems. Technical report, Conference on Data Systems Languages (CODASYL), May 1969.

[47] Daubert v. Merrell Dow Pharmaceuticals Inc. 509 U.S. 579, 1993. `http://supct.law.cornell.edu/supct/html/92-102.ZO.html`. Accessed 25 February, 2013.

[48] P. J. Deitel and H. M. Deitel. *C: How to Program*. How to program series. Pearson Education, Inc., Upper Saddle River, NJ, 6th edition, 2009.

[49] P. J. Denning. Is computer science science? *Communications of the ACM*, 48(4):27–31, April 2005.

[50] K. Eckstein. Forensics for advanced UNIX file systems. In *Fifth Annual IEEE SMC Information Assurance Workshop*, pages 377 – 385. IEEE, 2004.

[51] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 6th edition, 2010.

[52] S. S. Epp. *Discrete mathematics with applications*. PWS Publishing Company, 1993.

[53] O. M. Fasan and M. S. Olivier. Correctness proof for database reconstruction algorithm. *Digital Investigation*, 9(2):138–150, 2012.

[54] O. M. Fasan and M. S. Olivier. Database forensics. In *Proceedings of the 3rd Research Consortium on Information Technology Innovations (RECITI)*. Nigeria Computer Society (NCS), 2012.

[55] O. M. Fasan and M. S. Olivier. On dimensions of reconstruction in database forensics. In *Proceedings of the 7th International Workshop on Digital Forensics and Incident Analysis (WDFIA 2012)*, pages 97 – 106. Plymouth University, 2012.

[56] O. M. Fasan and M. S. Olivier. Reconstruction in database forensics. In *Advances in Digital Forensics VIII*, volume 383 of *IFIP Advances in Information and Communication Technology*, pages 273–287. Springer Berlin Heidelberg, 2012.

[57] B. A. J. Fisher, W. J. Tilstone, and C. Woytowicz. *Introduction to Criminalistics: The Foundation of Forensic Science*. Elsevier Limited, Oxford, 2009.

[58] K. Fowler. Forensics analysis of a SQL server 2005 database server, April 2007. SANS Institute.

[59] K. Fowler. *SQL Server Forensic Analysis*. Addison Wesley Professional, NJ, 2008.

[60] P. Frühwirt, M. Huber, M. Mulazzan, and E. R. Weippl. InnoDB database forensics. In *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 1028–1036, 2010.

[61] P. Frühwirt, P. Kieseberg, S. Schrittwieser, M. Huber, and E. Weippl. Innodb database forensics: Reconstructing data manipulation queries from redo logs. In *Seventh International Conference on Availability, Reliability and Security (ARES)*, pages 625–633, 2012.

[62] S. L. Garfinkel. Carving contiguous and fragmented files with fast object validation. *Digital Investigations*, 4:2–12, September 2007.

[63] S. L. Garfinkel. Digital forensics research: The next 10 years. *Digital Investigation*, 7, Supplement(0):S64 – S73, 2010. The Proceedings of the Tenth Annual DFRWS Conference.

[64] P. Gladyshev. *Formalising Event Reconstruction in Digital Investigations*. PhD thesis, University College, Dublin, 2004.

[65] P. Gladyshev and A. Patel. Finite state machine approach to digital event reconstruction. *Digital Investigation*, 1(2):130–149, 2004.

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

[67] K. Grolinger, W. A. Higashino, A. Tiwari, and M. A. Capretz. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1):22, 2013.

[68] Guidance Software. EnCase forensic. `www.encase.com`. Accessed 25 February, 2013.

[69] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh. Extensible query processing in starburst. In *ACM SIGMOD*, pages 377–388. ACM Press, 1989.

[70] C. Hargreaves and J. Patterson. An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, 9, Supplement(0):S69 – S79, 2012. The Proceedings of the Twelfth Annual DFRWS Conference.

[71] IBM. *IBM DB2 Log Analysis Tool for z/OS, User's Guide*. IBM, version 3 release 2 edition. `http://pic.dhe.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db2tools.ala33.doc.ug/alaugc30.pdf?noframes=true`. Accessed 19 August 2013.

[72] IBM. *IBM DB2 10.1 for Linux, Unix and Windows: Database administration concepts and configuration*. IBM, 2012.

[73] IBM Software. i2 analyst notebook. `http://www.ibm.com/software/products/us/en/analysts-notebook-premium/`. Accessed 25 February, 2013.

[74] R. S. C. Ieong. FORZA - digital forensics investigation framework that incorporate legal issues. *Digital Investigation*, 3:29–36, 2006.

[75] ITL Education Solutions Limited. *Introduction to Database systems*. Pearson Education, India, November 2008.

[76] M. Jakobsson and Z. Ramzan. *Crimeware: Understanding New attacks and Defences*. Pearson Education, 2008.

[77] S. H. James and J. J. Nordby. *Forensic Science: An Introduction To Scientific And Investigative Techniques*. CRC Press, Boca Raton, FL, 2003.

[78] D. A. Jardine, editor. *The ANSI/SPARC DBMS Model*, Proceeding of the Second SHARE Working Conference on Date Base Management Systems, Montreal, Canada, April 1976. North-Holland.

[79] T. A. Johnson. *Forensic Computer Crime Investigation*. International Forensic Science and Investigation. Taylor & Francis, 2005.

[80] P. Juola. Future trends in authorship attribution. In P. Craiger and S. Shenoi, editors, *Advances in Digital Forensics III*, volume 242 of *IFIP The International Federation for Information Processing*, pages 119–132. Springer New York, 2007.

[81] A. Kornbrust. Oracle rootkits 2.0. `www.red-database-security.com/wp/oracle_rootkits_2.0.pdf`, August 2006. Black Hat USA.

[82] N. Leavitt. Will NoSQL databases live up to their promise? *Computer*, 43(2):12–14, February 2010.

[83] H. C. Lee, T. M. Palmbach, and M. T. Miller. *Henry Lee's Crime Scene Handbook*. Academic Press, London, UK, 2001.

[84] D. Litchfield. Oracle forensics part 1: Dissecting the redo logs, March 2007. NGSSoftware Insight Security Research (NISR) Publication.

[85] D. Litchfield. Oracle forensics part 2: Locating dropped objects, March 2007. NGSSoftware Insight Security Research (NISR) Publication.

[86] D. Litchfield. Oracle forensics part 3: Isolating evidence of attacks against the authentication mechanism, March 2007. NGSSoftware Insight Security Research (NISR) Publication.

[87] D. Litchfield. Oracle forensics part 4: Live response, April 2007. NGSSoftware Insight Security Research (NISR) Publication.

[88] D. Litchfield. Oracle forensics part 5: Finding evidence of data theft in the absence of auditing, August 2007. NGSSoftware Insight Security Research (NISR) Publication.

[89] D. Litchfield. Oracle forensics part 6: Examining undo segments, flashback and the oracle recycle bin, August 2007. NGSSoftware Insight Security Research (NISR) Publication.

[90] D. Litchfield. Oracle forensics part 7: Using the oracle system change number in forensic investigations, November 2008. NGSSoftware Insight Security Research (NISR) Publication.

[91] K. Mandia and C. Prosise. *Incident Response & Computer Forensics, 2nd Ed.* Security Series. McGraw-Hill Education, 2003.

[92] R. McKemmish. What is forensic computing. In *Australian Institute of Criminology, Trends and Issues in crime and criminal justice*, number 118. June 1999. `http://www.aic.gov.au/publications/current%20series/tandi.aspx`. Accessed: November 2012.

[93] M. Meyers and M. Rogers. Computer forensics: The need for standardization and certification. *International Journal of Digital Evidence*, 3(2), 2004.

[94] M. Meyers and M. Rogers. Digital forensics: Meeting the challenges of scientific evidence. In M. Pollitt and S. Shenoi, editors, *Advances in Digital Forensics*, volume 194 of *IFIP The International Federation for Information Processing*, pages 43–50. Springer US, 2005.

[95] Microsoft Developer Network. SQL Server 2012, 2012. `http://www.msdn.microsoft.com/library`. Accessed 4 April, 2013.

[96] National Institute of Justice (U.S.). *Forensic examination of digital evidence: a guide for law enforcement.* NIJ special report. U.S. Dept. of Justice, Office of Justice Programs, National Institute of Justice, 2004.

[97] National Institute of Justice (U.S.). *Electronic Crime Scene Investigation: A Guide for First Responders, Second Edition.* NIJ special report. U.S. Dept. of Justice, Office of Justice Programs, National Institute of Justice, April 2008.

[98] S. Nebiker and S. Bleisch. *Introduction to Database Systems*. Geographic Information Technology Training Alliance (GITTA), June 2010. `http://www.gitta.info`. Accessed 4 June, 2013.

[99] Netmap Analytics. Netmap. `www.netmap.com`. Accessed 25 February, 2013.

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

[101] M. S. Olivier. On complex crimes and digital forensics. Technishe Berichte 63, Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam, 2013.

[102] M. S. Olivier and S. Gruner. On the scientific maturity of digital forensics research. In *Advances in Digital Forensics IX*, IFIP Advances in Information and Communication Technology - Advances in Digital Forensics, pages 33–49. Springer, 2013.

[103] A. J. Oppel. *Databases: A Beginner's Guide*. McGraw-Hill, 2009.

[104] Oracle. Oracle database administrator guide 11g release 1(11.1). `http://docs.oracle.com/cd/B28359_01/server.111/b28310/toc.htm`. Accessed 19 March 2013.

[105] Oracle. Mysql 5.6 reference manual. oracle, 2013, 2013. `http://dev.mysql.com/doc/refman/5.6/en/server-logs.html`. Accessed 19 March 2013.

[106] M. Otey. SQL Server log file. SQL Server Pro Community. `http://www.sqlmag.com/article/log-files/sql-server-log-files-94131`. Accessed 19 March 2013.

[107] G. Palmer. A road map for digital forensic research. Technical report, First Digital Forensic Research Workshop (DFRWS), Utica, New York, August 2001.

[108] D. B. Parker. *Crime by Computer*. Scribner, New York, 1976.

[109] D. B. Parker. Computer abuse research update. *Computer/Law Journal*, 2:329 – 352, 1980.

[110] D. B. Parker. *Fighting computer crime*. Scribner, New York, 1983.

[111] K. Pavlou and R. T. Snodgrass. Forensic analysis of database tampering. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 109–120, 2006.

[112] K. Pavlou and R. T. Snodgrass. The tiled bitmap forensic analysis algorithm. *IEEE Transaction on Knowledge and Data Engineering*, 22:590–601, April 2010.

[113] H. Pieterse and M. S. Olivier. Data hiding techniques for database environments. In *Advances in Digital Forensics VIII*, volume 383 of *IFIP Advances in Information and Communication Technology*, pages 289 – 301. Springer Berlin Heidelberg, 2012.

[114] M. Pollitt. A history of digital forensics. In *Advances in Digital Forensics VI*, volume 337 of *IFIP Advances in Information and Communication Technology*, pages 3 – 15. Springer Berlin Heidelberg, 2010.

[115] R. Rankins, P. Bertucci, C. Gallelli, and A. T. Silverstein. *Microsoft SQL Server 2008 R2 Unleashed*. Sams, 2010.

[116] M. Reith, C. Carr, and G. H. Gunsch. An examination of digital forensic models. *International Journal of Digital Evidence*, 1(3), 2002.

[117] R. Rowlingson. Abstract a ten step process for forensic readiness. *International Journal of Digital Evidence*, 2(3), 2004.

[118] J. Rynearson. *Evidence and Crime Scene Reconstruction*. National Crime Investigation and Training, Redding, CA, 2002.

[119] B. Schatz. *Digital evidence: representation and assurance*. PhD thesis, Queensland University of Technology, Australia, October 2007.

[120] Scientific Working Group on Digital Evidence (SWGDE). SWGDE and SWGIT digital and multimedia evidence glossary, May 2009. Version 2.3.

[121] X. Silao, W. Song, and H. Mei. Application of SQL RAT translation: A statement of RQP/RMP with an object-oriented solution. *Inter. Journal of Intelligent Systems and Applications*, 3(5):48 – 55, August 2011.

[122] S. K. Singh. *Database Systems: Concepts, Design & Applications*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2009.

[123] R. T. Snodgrass, S. S. Yao, and C. Collberg. Tamper detection in audit logs. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 504–515, 2004.

[124] P. Stahlberg, G. Miklau, and B. N. Levine. Threats to privacy in the forensic analysis of database systems. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 91–102, 2007.

[125] W. Stallings. *Operating Systems: Internals and Design Principles*. Pearson Prentice Hall, 5th edition, 2005.

[126] P. Stephenson. Modeling of post-incident root cause analysis. *International Journal of Digital Evidence*, 2(2), 2003.

[127] P. Stephenson. The application of formal methods to root cause analysis of digital incidents. *International Journal of Digital Evidence*, 3(1), 2004.

[128] Sybase. *SybaseIQ 15.1.2 documentation*. Sybase, 2010. `http://infocenter.sybase.com/help/index.jsp`. Accessed, 19 Mar. 2013.

[129] R. W. Taylor and R. L. Frank. CODASYL Data-Base management systems. *ACM Computing Surveys*, 8(1):67 – 103, March 1976.

[130] S. Tewelde, M. S. Olivier, and S. Gruner. Notions of "Hypothesis" in digital forensics. In *Eleventh Annual IFIP WG 11.9 International Conference on Digital Forensics*, Orlando, Florida, USA, January 2015. Accepted for presentation.

[131] The PostgreSQL Global Development Group. Postgresql 9.2.3 documentation, 2013. `http://www.postgresql.org/docs/9.2/static/`. Accessed 19 March 2013.

[132] D. C. Tsichiritzis and A. Klug. The ANSI/X3/SPARC DBMS framework: Report of the study group on database management systems. *Information Systems*, 3(3):173–191, 1978.

[133] B. Turvey. *Criminal Profiling: An Introduction to Behavioral Evidence Analysis*. Academic Press, 2002.

[134] U.S. Department of Justice. Federal guidelines for searching and seizing computers, 1994.

[135] B. Ward. *How Linux Works: What Every Superuser Should Know*. No Starch Press, San Francisco, CA, 2004.

[136] R. Westervelt. Black hat 2007: New database forensics tool could aid data breach cases. `http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1266525,00.html`, August 2007. TechTarget.

[137] D. M. Wong and K. B. Edwards. System and method for investigating a data operation performed on a database. `www.freepatentsonline.com/20050289187.pdf`, December 2005. United States patent application publication.

[138] P. M. Wright. Oracle database forensics using LogMiner, January 2005. Next Generation Security Software. `http://www.databasesecurity.com/dbsec/OracleForensicsUsingLogminer.pdf`.

[139] P. M. Wright. *Oracle Forensics: Oracle Security Best Practices*. Rampant Techpress, 2010.

[140] P. M. Wright. Oracle forensics in a nutshell, 2007. `http://www.databasesecurity.com/dbsec/OracleForensicsInANutshell.pdf`. Accessed 19 March 2013.