Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By

Shyam Lokhande	
Abhishek Dhamdhere	
Neha Bhosale	
Vaishnav Bhujbal	

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

DATA RECOVERY USING MSSQL

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By

Abhishek Dhamdhere

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

CERTIFICATE

This is to certify that the Seminar report entitled, "**FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH"** Submitted by,

	Student Name	**Exam Seat No.**
1.	Shyam Lokhande	
2.	Abhishek Dhamdhere	
3.	Neha Bhosale	
4.	Vaishnav Bhujbal	

is a record of bonafide work carried out by him/her, in the partial fulfillment of the Term-work of Third year in Computer Engineering of Savitribai Phule Pune University at Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune under Savitribai Phule Pune University, Pune. This work is done during the academic year 2022-23.

Date: - 19/10/2022

Place: - Pune

Prof. B.C.Julme	Prof. M.S.Pokale	Prof.D.D.Sapkal
Seminar Guide	External Examiner	H.O.D. (Computer Engg)

Acknowledgment

I would like to express my gratitude towards my guide, Prof.B.C.Julme, Assistant Professor in the Computer Engineering Department, who has been very concerned and has aided for all the help essential for the preparation of this work. He has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.

I am also thankful to Prof. D.D.Sapkal, Head of Department, Computer Engineering, for the motivation and inspiration that triggered me for the seminar Work.

Shyam Lokhande

T1

(T.E.Computer Engineering)

Abstract

Database forensics is becoming more important for investigators with the increased use of the information system. Although various database forensic methods such as log analysis and investigation model development have been studied, among the database forensic methods, recovering deleted data is a key technique in database investigation for DB tampering and anti-forensics. Previous studies mainly focused on transaction or journal log to recover deleted data, but if logs are set to be deleted periodically or logs containing critical evidence are overwritten by new logs, the log-based recovery method can not be used practically. For this reason, an engine-based recovery method that analyzes data file at a raw level has been also introduced. There is research to recover small-sized databases such as SQLite and EDB, but there is no prior work describing the structure of data file and technology to recover deleted data of large databases used by enterprises or large organizations. In this context, investigate Microsoft SQL Server (MSSQL), which is one of the most used large databases. Our method focuses on a storage engine of MSSQL. Through analyzing the storage engine, identify the internal structure of MSSQL data files and the storage mechanism. Based on these findings, a method to recover tables and records is presented by empirical examination. It is compatible with various versions of MSSQL because it accesses data at the raw level. Our proposed method is verified by a comparative experiment with forensic tools implemented to recover deleted MSSQL data. The experimental results show that our method recovers all deleted records from the unallocated area. It recovers all data types including multimedia data, called Large Objects (LOB) in the database

field. To contribute digital forensic community, also provide the source code of the implementation; it facilitates the knowledge sharing of database forensics.

INDEX

LIST OF FIGURES

CHAPTER 1 : INTRODUCTION

Database forensics is a branch of digital forensics that analyzes the structure of database and examines contents. It has motivated forensic researchers since database manages important data of personal or enterprise. With the increased use of Internet of Things (IoT) devices, database allows countless users to access a variety of applications and stores the user's data and log; database forensics is becoming more important for a forensic investigator.

Various forensic methods have been studied to find critical evidence in database, such as log analysis and investigation model development.

Among the database forensic methods, recovering deleted records plays an important role in finding the evidence, because it can recover data that existed in the past or that was intentionally deleted for anti-forensics. Especially, attackers try to the database tampering to take away sensitive information or try to delete the information by recovering deleted records, an investigator can detect and deal with these behaviors. The recovery of deleted records is also used to solve the financial fraud cases in forensic accounting.
The techniques to recover deleted records are divided into two main categories: the log-based method and engine-based method. Many researchers proposed the log-based methods that analyze the transaction log or journal log. The approach showed a notable achievement, but there is a limit that logs containing critical evidence may not exist on the database. The logs are recorded by the administrator's security settings, so if the logs are set to be deleted periodically or to be small fixed-size, it is impossible to recover deleted data by analyzing the logs. For this reason, engine-based methods that analyze the structure of the data file and then recover deleted data were also studied. The methods approach raw data directly, so it can improve the drawback of the log-based method. Previous studies mainly analyzed small-sized databases such as SQLite and EDB, used in endpoint devices (desktop computers, smartphones, and tablets). On the other hand, there is no research describing an engine-based recovery method for large databases, mainly used in enterprises or large organizations, such as MSSQL and Oracle. This is

because the Database Management Systems (DBMS) structure of the large databases is complex and the source code of the DBMS is not open to the public, making it difficult to identify the structure of data file and storage mechanism.

Here propose an engine-based recovery method to recover deleted data in MSSQL that is one of the most used DBMS globally. research the internal structure of the MSSQL storage file and then describe the system tables used to recover deleted records. The storage engine of MSSQL remains unchanged when the MSSQL version is updated, so this approach is compatible with various versions from MSSQL 2008 R2 to 2019. Based on the MSSQL data file format and system table, reconstruct all types of records including LOB. Lastly, implement the proposed method as a tool. The developed tool is evaluated for performance with existing forensic tools that are developed to recover deleted data in MSSQL.

The main contribution of this is summarized as follows:

• This report presents the detailed internal structure of the MSSQL data file.

• This report introduces MSSQL data recovery algorithm based on the storage engine.

• The proposed method recovers all deleted records on the unallocated area and it is compatible with various MSSQL versions.

• This report provides a comparative performance evaluation of our method and well-known forensic tools developed to recover deleted data of MSSQL.

The remainder of this report is organized as follows:

Related works are presented in Section II. In Section III, it introduces an internal structure of the MSSQL storage. Next, it describes several system tables used to recover deleted records in Section IV. In Section V and VI, describe and evaluate the proposed method

CHAPTER 2 : RELATED WORK

2.1 DATABASE FORENSICS MODEL:

Several database forensic models have been researched steadily. The models focused on how an investigator performs tasks to discover information on database. The researchers tried to propose universal database forensic models.

Khanuj a and Adane proposed a framework for analyzing and reconstructing the activity of the suspect's behavior. They described their framework in general digital forensic procedures: identification, collection, analysis, validation, interpretation, report, and preservation. Beyers et al. focused on the data model of the DBMS and proposed a method to transform the data model into a clean state. Flores and Jhumka presented a database investigation model for audit records. They tried to satisfy the chain of custody requirements by implementing triggers and stored procedures. Bria et al. proposed five stages of database forensic analysis: identification, investigation, artifacts collection, analysis, and documentation. The stages are similar to general forensic procedures, so they attempted to match database forensics with digital forensics. Al-Dhaqm et al., proposed a forensic model by reviewing 54 investigation processes from 18 forensic models of databases. They presented database forensics with four phases: identification, artifact collection & preservation, artifact analysis, and documentation & presentation.

Wagner et al. proposed architectures in the context of database reconstruction model: `DBDetective' and `DBCarver'. The DBDetective works on capturing snapshots from RAM and hard drive. The DBCarver was developed to reconstruct the database from a forensic image. It carves individual pages using three parameters: the page header, row directory, and the row data. Wagner et al. also designed a tool, named 'DICE' , to recover deleted data. It analyzes data files directly and identifies the remnant data, marked as deleted but not actually deleted. The studies contributed to the digital forensic community, however, it is difficult to fully grasp various DBMS; more explanations about the detailed parameters and recovery algorithm are needed. Especially, commercial DBMSes such as MSSQL have complex algorithms to store LOB, therefore enhance the knowledge of metadata and structure of LOB in this paper.

2.2 DBMS-DEPENDENT FORENSIC ANALYSIS :

Previous researches have tried to create DBMS-independent models, but, owing to the complexity and multidimensional nature of DBMS, the proposed database forensic models have little practical application. For this reason, DBMS-dependent forensic models have been studied.

Fowler proposed a method for investigating MSSQL. The method consists of 4 investigation phases: preparation, incident verification, artifact collection, and analysis. Khanuja and Adane presented artifact collection of MSSQL log files, data files, cache, and so on. Based on the collected artifacts, it detects whether the database has been tampered with by an attacker. Toombs explored MSSQL forensics through standard forensic processes. Toombs also demonstrated that the size and distributed nature of the database should be considered in database forensics.
Previous studies show practical methodologies when MSSQL is attacked by hackers. The methodologies are available when the forensic investigator can access transaction logs or use DBMS functions provided by MSSQL. This article has a different point of view about the research subject and method to analyze MSSQL. doesn't research transaction log but data file, so access MSSQL directly without using DBMS services. By this approach, our method can be applied even if MSSQL is crashed or the transaction logs are overwritten or damaged.

CHAPTER 3 : INTERNAL STRUCTURE OF THE MSSQL DATABASE

The MSSQL consists of two kinds of files. One is a data file (.mdf,.ndf) that stores actual data, and the other is a transaction log file (.ldf) that stores log data. The transaction log may not be acquired during data collection and it may be modified or deleted by the attacker [25]. In addition, it may not be possible to obtain enough data to investigate incidents due to a policy for log size limitation. Given the situation, focus on the data file where the raw data is stored. When the table or record is stored in the data file, MSSQL uses a storage engine that has not been changed even if the MSSQL version was updated. To analyze the storage engine, reverse-engineered MSSQL. Based on our findings, this section presents the internal structure of the data file and how MSSQL stores the data.

3.1 Page:

The page is the smallest unit of MSSQL data file and its default size is 8,192 (0x 2000) bytes. Fig. 1 shows the overall structure of the page; it consists of the page header, data row, and the row offset array. This subsection describes the page header and row offset array. The data row is represented in Section III-B.

The page header is 96 bytes in size. The first 64 bytes, described in Fig. 2, contain the page metadata and the remaining 32 bytes are filled with data. Among the fields of the page header, 7 fields are used to recover records: Type, Flag Bits, SlotCnt, Page Object ID, Page ID, File ID, and Checksum (TornBits).

The Flag Bits field indicates how the Checksum field is used. If the Flag Bits value is 0 x 200, the Checksum field is used for validating the page. However, if the value is 0x100, the Tornbit is stored in the Checksum field. The Tornbit is described in Section III-D.

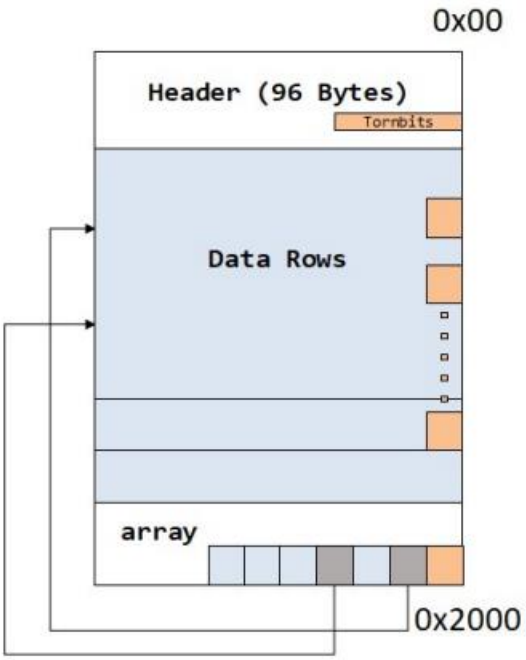


FIGURE 1. Page structure (applied with Tornbits).

The SlotCnt field represents the number of normal records on the page. The Page Object ID field is used to store an identifier of the object such as a table, index, and so on. Therefore, by analyzing the Page Object ID, it is determined to which page the data of the table is allocated. The Page ID indicates the page number. By calculating the 'Page ID' x 'size of page (0 x 2000)', the page's location is identified.

The File ID represents the file identification number. In MSSQL, table data can be stored in multiple data files. The primary data file has .mdf extension and the secondary data file has .ndf extension.

The row offset array indicates the location where records are stored on the page. It consists of 2 bytes values that represent the start position of each record. As seen in Fig. 1, 000 the value at the end of the array points to the first record on the page; the row offset array is stored in reverse order

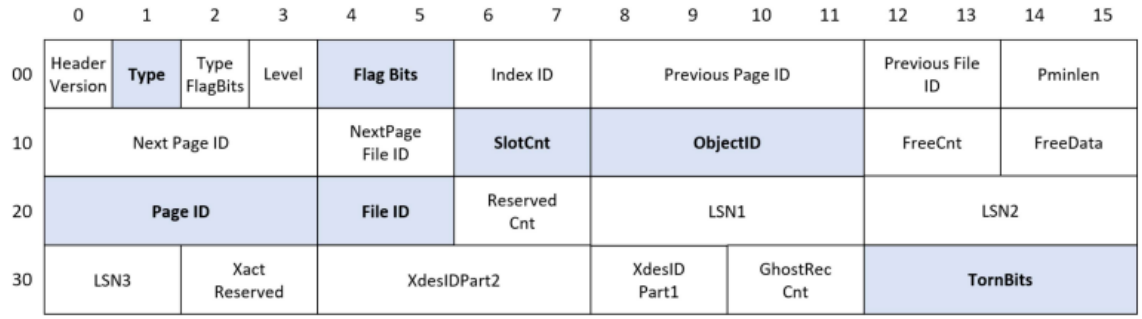


FIGURE 2. **Page header structure.**

The Type field indicates the type of the page. It is known that there are 14 types of the page; this report focuses on data page (Type 1), text mixed page (Type 3), and text page (Type 4). The Type 1 page stores data records. The data value can be classified into normal value and large object value; the Type 1 page can store the normal value. The Type 3 and 4 pages are used to manage the LOB data, which is described in Section III-C.

3.2 Record:

In MSSQL, a record of the table is stored in a data row of the Type 1 page. The way to store the record is determined by the size of the record. If the size of the record is less than 8,060 bytes, the record is stored with in Row Data method. Fig. 3 shows the structure of record stored with the In Row Data. As seen in the figure, at the front of the record, fixed-length columns such as int, float, and date are stored. And then, the variable-length columns such as varchar, nvarchar, and varbinary are followed. The description of each field is as follows:

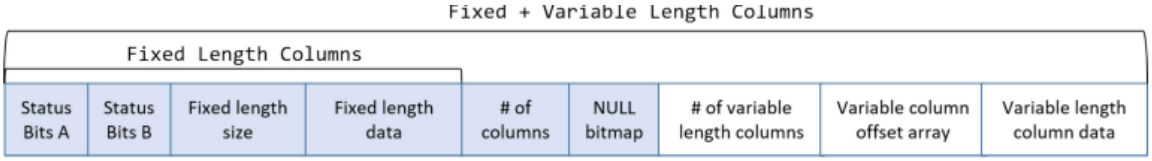


FIGURE 3. **Record structure: In row data.**

The first 2 bytes of the row, called Status Bits A and Status Bits B, are the bitmaps containing the information about the row. The bitmaps indicate row type (ghosted or NULL), MSSQL version, and the presence of variable-length columns. The next two bytes indicate the size of the fixed-length data. After the fixed-length data portion, the number of columns and the NULL bitmap is stored. Following the NULL bitmap, there is the variable-length data portion. The first two bytes indicate the number of the variable-length columns. MSSQL stores two bytes offset values per each variable-length column in the variable column offset array. It is followed by the actual variable-length portion of the data.

3.3 Large Data:

When the size of the record exceeds 8,060 bytes, MSSQL stores the record on multiple pages. In this case, the value of the record in the Type 1 page indicates the location of another page where the large data is stored. The page that stores the large data is Type 3 and Type 4 pages. Fig. 4 shows how large data is stored. As seen in the figure, the data of the Type 1 page is used to indicate the address of Type 3 or Type 4 pages.

There are three different ways to store large data, depending on the data type and length: Row-overflow Data, LOB Data, and MAX-length Data.

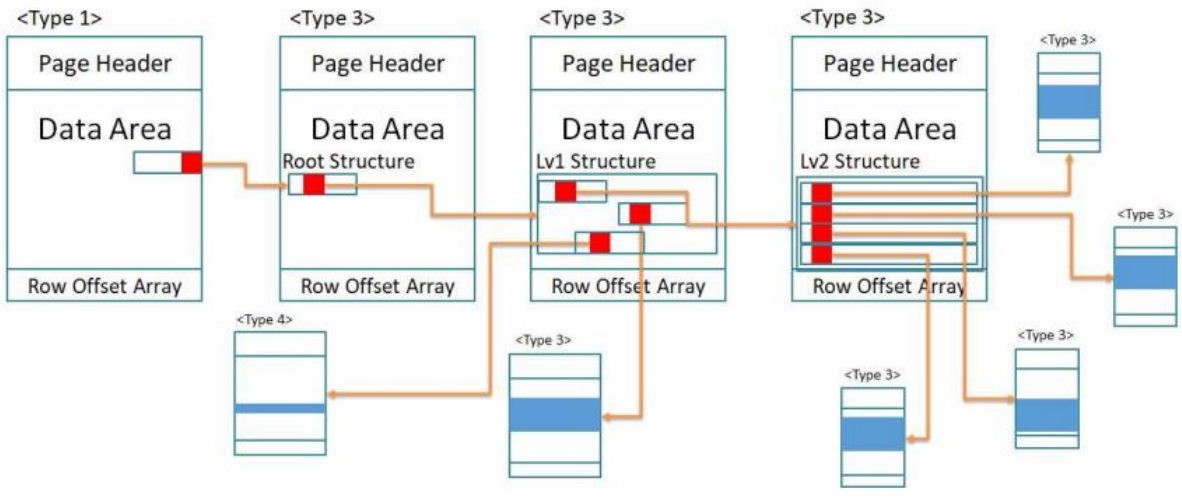


FIGURE 4. Large data structure (B-tree).

Row-overflow Data called restricted-length Large object data is used when the data type is varchar(n), varbinary(n), or nvarchar(n). Row-overflow data refers to the large data whose record size exceeds 8,060 bytes, but the size of each column data is less than 8,060 Bytes. For example, if the schema of a table consists of varchar(8000) and nvarchar(4000), the size of each column data is less than 8,060 bytes but the total record size is 16,000 bytes. Therefore MSSQL stores the record in Row-overflow Data form. However, the record can be stored as In Row Data when the size of the actual record data does not exceed 8,060 bytes even if the predefined size is more than 8,060 bytes.

LOB Data called unrestricted-length large object data is used when the type is text, ntext, image, varchar, nvarchar, or varbinary. Since MSSQL 2005, the text, ntext, and image data type are deprecated and varchar, nvarchar, and varbinary data type with MAX specifier are substituted for them. name records using the deprecated data types (text, ntext, image) LOB Data.

MAX-length Data is also unrestricted-length large object data like LOB Data. When MAX specifier is used as a parameter of the data type varchar, varbinary, or nvarchar, the record is stored as MAX-length Data. It means that the size of the record can exceed 8,060 bytes like LOB Data, but there is a difference from LOB Data. When data of 8,060 bytes or less is stored in a record defined as a data type using MAX specifier, it is stored in the same manner as In Row Data. When more than 8,060 bytes of data are stored, the data is stored as MAX-length Data.

3.4 Tornbits:

There are two ways to verify the page's integrity in MSSQL; One is Checksum and the other is TomPageDetection. When creating a database, the user can choose three options: Checksum, TornPageDetection, and None. In particular, the TornPageDetection uses the byte substitution. It is applied to the pages of data files such as .mdf and .ndf, so interpreting the Tornbits is important when parsing records. The Tornbits is similar to Fixup Arrays of NTFS.

When the Tornbits option is applied, the last two bits of particular bytes are replaced by the last two bits of the Tornbits.

Fig. 5 shows an example of the Tornbits. As seen in the figure, the last two bits of bytes from 0 x 3FF to 0 x 1FFF at 0 x 200 interval are replaced 102, the last two bits of the Tornbits. And then the original binary data of the bytes are recorded in the TornBits. When the page is loaded into memory, the original data stored in the TornBits are also loaded; in this way, MSSQL verifies the integrity of the page.

CHAPTER 4 : SYSTEM TABLES

This section describes the system tables that are required to recover the deleted records. Among many system tables, five system tables are used for recovery in this paper: syscolpars, sysschobjs, sysiscols, sysrowsets, sysallocunits.

Analyzed the schema structure of system tables based on MSSQL 2017. Some versions have different schema structures of system tables, but the columns required for record recovery remain unchanged regardless of the MSSQL version. So, the recovery method proposed in Section 5 is independent of versions because system tables are analyzed before data in user tables are accessed.

4.1 SYSCOLPARS:

In the syscolpars, column information of all tables of the database, including system tables and user tables, is stored. By analyzing the syscolpars, column's name, the order in which columns are stored, size, and type are identified.

4.2 SYSSCHOBJS:

The sysschobjs stores all table information used in the database. It contains the table name, the number of columns, the table type (system table or user table), and the Table Object ID.

4.3 SYSISCOLS:

The sysiscols stores the index column information of the table. Generally, the column data of a record is stored in the data file in the order specified by the user, which is identified in the syscolpars. However, when an index is created on the table (e.g. designating a primary key), column data of records are stored in a different order (See Table 1). This characteristic is an important issue when recovering deleted records; this issue is discussed in detail in subsection 5.3.

4.4 SYSROWSETS:

The sysrowsets stores Partition ID for each table. Partition ID is used to obtain the Allocation Unit ID from the sysallocunits.

Among the columns of sysrowsets, the idmajor column represents the Table Object ID and the rowsetid column represents the Partition ID. The Partition ID of the table to

be analyzed can be obtained by comparing the Table Object ID obtained from the sysrowsets with the Table Object ID obtained from the sysschobjs.

4.5 SYSCOLUNITS:

The sysallocunits manages Allocation Unit IDs. The Allocation Unit ID of the table can be obtained via the Partition.

ID obtained from the sysrowsets. The Allocation Unit ID is used to acquire the Page Object ID of the table. Among the columns of sysallocunits, the ownerid column represents the Partition ID and the auid column represents the Allocation Unit ID

CHAPTER 5 : RECOVERY OF DATA

This section introduces a method to recover deleted records by analyzing the internal structure of the data file and system tables described in Section III and IV. The proposed method is version-independent because it is based on the storage mechanism that is constant even if the version of MSSQL is updated. There are four phases in the method: scanning pages, collecting system table information, collecting user table information, and recovering deleted records. Implement the proposed method in this paper as open-source tool. It can be downloaded from GitHub as mentioned in Section I.

5.1 SCANNING PAGE.

In this phase, all pages of the data files are scanned in page units. By scanning the files, all Page IDs and Page Object IDs are identified.

5.2 COLLECTING SYSTEM TABLE INFORMATION

This phase aims to collect the column information of the system tables such as sysschobjs, sysiscols, sysrowsets, and sysallocunits by parsing syscolpars. Although the schema of the system tables varies by MSSQL version, the column names that are needed to recover deleted records are constant. Therefore, by identifying the column names stored in syscolpars, columns required for recovery are acquired regardless of the MSSQL version.

5.3 COLLECTING USER TABLE INFORMATION

In this phase, the information of the user table to be recovered is obtained by parsing the system table analysed. This phase is further divided into four phases.

5.3.1 TABLE INFORMATION IN SYSSCHOBJS

By using the columns information identified in the previous phase, sysschobjs can be parsed and analyzed. In this phase, the user table's information such as table name, Table Object ID, and the number of columns is identified.

5.3.2 COLUMN INFORMATION IN SYSCOLPARS

Through matching syscolpars with sysschobjs, the column information of the user table is identified. The data type, length, name, the order in which columns data are stored, parameters of specific data types (time, numeric, and date-time) are obtained in this phase. Based on the information, a temporary table is created to store the recovered records, by using CREATE TABLE query.

5.3.3 CLUSTERED INDEX COLUMN INFORMATION IN SYSISCOLS

After obtaining the column information, the index column information is collected from the syscolpars and sysiscols. In this phase, the order in which the column data are stored is identified. Particularly, in the sysiscols, the clustered index information should be checked. As the indexing is developed to speed-up the query process in MSSQL, many databases are using this function, so identifying whether the indexing is used or not is an important procedure. For example, by setting the primary key, which is a typical function of the indexing, the order is completely changed (See Table 1).

5.4 LOCATION INFORMATION IN SYSROWSETS AND SYSALLOCUNITS

To identify Allocation Unit ID, sysrowsets and sysallocunits tables are parsed. The Page Object ID is calculated by Equation 1.

indexid = allocUnitId >> 48

pageObjectId = (allocUnitId — (indexid << 48)) >> 16

To obtain the Page ID to which user table is allocated, the Page Object ID of each page obtained in Section 5.1 is compared with the Page Object ID obtained in Equation 1. Finally, the start offset of the page is calculated by a formula, Page ID x page unit size(0 x 2000).

RECOVERING DELETED RECORDS

In this phase, records of the user table are recovered. It is further divided into three phases.

5.4.1 ACCESSING DATA PAGE

The page (Type 1) to which the data is allocated is identified through the location information of the user table acquired in Section V-C4

5.4.2 IDENTIFYING UNALLOCATED AREA

Fig. 6 shows three cases where the unallocated area can exist in a data page. In the first case, the unallocated area exists when more than one record below the page header was deleted. The second is that the unallocated area exists between the records. The third is that the unallocated area exists between the last record and the row offset array. The following is a procedure for determining the unallocated area according to these cases.

1) After storing the row offset array values in array (Slot) and arranging it in ascending order, the record length is calculated using column information from Section V-C2

2) If the value of the (Slot[i] + record length + 1) is equal to the value of Slot[i + 1], it is judged that there is no unallocated area between the records

3) If the value of Slot[i + 1] is larger than (Slot[i] + record length + 1), it is determined that there is the unallocated area between the i-th record and the (i + 1)-th record

5.4.3 RECONSTRUCTING DELETED RECORDS

Based on the record structure shown in Fig. 3, the data is analyzed by separating the fixed-length column and the variable-length column, and then an INSERT query is generated.

The algorithm to reconstruct deleted records is shown in Algorithm 1.

1] First, it calculates the size of the unallocated data area (LUnalloc)

2] Next, the length of the record to be recovered in the unallocated area (LRecord) is calculated. Then, to get the data of each column, the starting position of the divided fixed-length (Oststic) (Ovariabie) and variable-length column data area is calculated with reference to the Fig. 3.

When accessing column data, data is acquired according to the column order and data size of each column through schema information (Sschema) obtained from Section V-C2 and V-C3. Finally, the acquired data (D) is decoded according to the data type and used to make an INSERT query (q). Since there may be more than one record in the unallocated area, the length of the recovered record (LRecord) is stored in LTotal and compared to the size of the unallocated area (I,Unalloc)• If the value LTotal is greater than or equal to the value Lunaiioc, recovery of the record in the input unallocated area is finished.

In this section, present a method to recover deleted data in four stages. This approach may be applicable to other databases. For example, in Oracle, 'block' that is the smallest unit of the data file is mapped to the page of MSSQL. There are also system tables and user tables in the Oracle data file. If the identification of the unallocated area and the storage mechanisms of data types are researched, deleted data in Oracle can be also recovered by our proposed methodology

CHAPTER 6 : CONCLUSION

Database is used by many users to store and manage sen sitive data of personal or enterprise. Furthermore, with the increased use of Internet of Things devices, database allows countless users to access a variety of applications and stores the users data and log: database forensics is becoming more important for a forensic investigator. Although some previous researches proposed an universal investigation method to DBMSes, there is insufficient information on how to inves tigate DBMS and recover deleted records practically

In this paper, I have researched MSSQL, which is the most used DBMS globally. I have described the internal structure of MSSQL including large data, tornbits. To recover deleted records regardless of the MSSQL version, several sys tem tables have been analyzed. I have proposed a method to recover deleted records and implemented the method as open-source tool. Finally, the performance of our method has been verified by comparing the commercial recovery tools. Though I have focused on one DBMS, MSSQL, there are many DBMSCs that need to be analyzed and new DBMSes are still being developed. In database forensics, there is a lack of practical researches to investigate these DBMSes.

CHAPTER 7 : REFERENCES

1] HOYONG CHOI , SANGJIN LEE , AND DOOWON JEONG on “Forensic recovery of SQL Server Database : A Practical Approach” published on January 4,2021.

2] “Ten years of critical review on database forensics research” by Rupali Chopade , V.K. Pachghare, 2019.

3] “Categorization and Organization of Database Forensic Investigation Processes” by Shukor Abd Razak, (Member, IEEE), David A. Dampior (Senior Member, IEEE), KIM-KWANG RAYMOND CHOO , (Senior Member, IEEE), KAMRAN SIDDIQUE (Member, IEEE), RICHARD ADEYEMI IKUESAN, ABDULHADI ALQARNI , AND VICTOR R. KEBANDE

4] M. S. Olivier. "On metadata context in database forensics," Digit Invest.. vol. 5, nos. 34. pp. 115-123, Mar. 2019.

5] M. H. Bhagwani, R. V. Dharaskar, and V. Thakare. "Comparative analysis of database forensic algorithms," in Proc. IJCA Nat. Conf. Knowl, Innov Technol. Eng. (NCKITE), Jul. 2015, vol NCKITE 2015, no. 3, pp. 33-36 19] A. Al-Dhaqm. S. Abd Razak. D. A. Dampier, K.-K. R. Choo, K. Siddique. R. A. Ikucsan, A. Alqarni, and V. R. Kebande, "Categorization and organi zation of database forensic investigation processes," IEEE Access, vol. 8. pp. 112846-112858, 2020.

Annexure A: Glossary

1. Forensics : scientific tests or techniques used in connection with the detection of crime.
2. Database : a large amount of data that is stored in a computer and can be easily used, added to, etc.
3. RDBMS: Relational Database Management System
4. Servers : a computer that stores information that a number of computers can share
5. SE : Storage Engine
6. PGA: Program Global Area
7. RMAN: oracle recovery manager

Annexure B: Reference Paper

1. Title: Forensic Recovery of SQL Server Database: Practical Approach

2. Published In: Journal IEEE Access Volume 9
ISSN: 14564-14575

Annexure C: Plagiarism Report

Page Count: 36
Word Count: 5654
Character Count: 30207
Unique: 78%

Plagiarism: 22%

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By

Student Name	**Exam Seat No.**
Abhishek Dhamdhere	
Shyam Lokhande	
Neha Bhosale	
Vaishnav Bhujbal	

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

DATA RECOVERY USING ORACLE

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By

Abhishek Dhamdhere

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

CERTIFICATE

This is to certify that the Seminar report entitled, "**FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH"** Submitted by,

Student Name	**Exam Seat No.**
Abhishek Dhamdhere	
Shyam Lokhande	
Neha Bhosale	
Vaishnav Bhujbal	

is a record of bonafide work carried out by him/her, in the partial fulfillment of the Term-work of Third year in Computer Engineering of Savitribai Phule Pune University at Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune under Savitribai Phule Pune University, Pune. This work is done during the academic year 2022-23.

Date: - 19/10/2022

Place: - Pune

Prof. B.C.Julme	Prof. M.S.Pokale	Prof.D.D.Sapkal
Seminar Guide	External Examiner	H.O.D. (Computer Engg)

Acknowledgment

I would like to express my gratitude towards my guide, Prof.B.C.Julme, Assistant Professor in the Computer Engineering Department, who has been very concerned and has aided for all the help essential for the preparation of this work. He has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.

I am also thankful to Prof. D.D.Sapkal, Head of Department, Computer Engineering, for the motivation and inspiration that triggered me for the seminar Work.

Abhishek Dhamdhere

T1

(T.E.Computer Engineering)

Abstract

Database forensics is becoming more important for investigators with the increased use of the information system. Although various database forensic methods such as log analysis and investigation model development have been studied, among the database forensic methods, recovering deleted data is a key technique in database investigation for DB tampering and anti-forensics. There is research to recover small-sized databases such as SQLite and EDB, but there is no prior work describing the structure of data files and technology to recover deleted data of large databases used by enterprises or large organizations. In this context, I investigate ORACLE SQL Server, which is one of the most used large databases. I look for evidence in those places and technologies designed by Oracle for disaster recovery purposes – namely Undo segments, Flashback and the Recycle Bin - of a compromise and the actions an attacker may have taken. To contribute to the digital forensic community, I have also provide the source code of the implementation; it facilitates the knowledge sharing of database forensics.

INDEX

LIST OF FIGURES

CHAPTER 1 : INTRODUCTION

Database forensics is a branch of digital forensics that analyzes the structure of database and examines contents. It has motivated forensic researchers since database manages important data of personal or enterprise. With the increased use of Internet of Things (IoT) devices, database allows countless users to access a variety of applications and stores the users' data and log; database forensics is becoming more important for a forensic investigator.

Various forensic methods have been studied to find critical evidence in database, such as log analysis and investigation model development. Among the database, the associate editor coordinating the review of this manuscript and approving it for publication was Shaohua Wan. Forensic methods, recovering deleted records plays an important role in finding the evidence, because it can recover data that existed in the past or that was intentionally deleted for anti-forensics. Especially, attackers try to the database tampering to take away sensitive information or try to delete the information; by recovering deleted records, an investigator can detect and deal with these behaviors. The recovery of deleted records is also used to solve the financial fraud cases in forensic accounting.

The techniques to recover deleted records are divided into two main categories: the log-based method and engine-based method. Many researchers proposed the log-based methods that analyze the transaction log or journal log. The approach showed a notable achievement, but there is a limit that logs containing critical evidence may not exist on the database.

CHAPTER 2 : MOTIVATION

Data is one of the most powerful weapons a business has. It provides key insights about employees, customers, products and competitors. It's collected from a whole host of sources and can make or break a firm's success.

Losing important and sensitive data can have a significant impact on the way our business operates. First, losing important data can put a halt in our operations. After all, how can proceed with business when important, and probably sensitive, files and data are missing.

CHAPTER 3 : INTRODUCTION TO ORACLE

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components.

Figure 1 : Logo Of Oracle

3.1 ORACLE DATABASE SERVICES

i. Data Analysis: The in-database analytical functions and features that are embedded inside the Oracle Database can be used to answer a wide variety of business problems. Developers and business users can access a wide range of

analytic features and combine their results with other SQL queries and analytical pipelines to gain deeper insights.

ii. Data Administration: Each database requires at least one database administrator (DBA). An Oracle Database system can be large and can have many users. Therefore, database administration is sometimes not a one-person job, but a job for a group of DBAs who share responsibility.

A database administrator's responsibilities can include the following tasks:

- Installing and upgrading the Oracle Database server and application tools
- Allocating system storage and planning future storage requirements for the database system
- Creating primary database storage structures (tablespaces) after application developers have designed an application

iii. Data Replication: Replication is the process of copying and maintaining database objects, such as tables, in multiple databases that make up a distributed database system. Changes applied at one site are captured and stored locally before being forwarded and applied at each of the remote locations. Oracle replication is a fully integrated feature of the Oracle server; it is not a separate server.

iv. Data Warehousing: Oracle Autonomous Data Warehouse is an easy-to-use, fully autonomous data warehouse that scales elastically, delivers fast query performance, and requires no database administration. The setup for Oracle Autonomous Data Warehouse is very simple and fast.

v. System Migration: Oracle migration processes and procedures transform existing versions or releases of Oracle databases (including their applications) into different versions or releases. All Oracle7 Server releases are upwardly compatible with all earlier Oracle versions and releases. Therefore, databases transformed using the migration processes described in this book work in the same manner as in earlier versions and, optionally, permit the use of functionality available with the new release.

vi. Backup & Recovery: Oracle provides a powerful utility to back up and restore databases, which is known as Oracle Recovery Manager (RMAN). We can use RMAN to back up databases as a complete backup or incremental backup. Since RMAN is a feature of the Oracle Database server, there is no need to separately install it.

vii. Data Modeling: Oracle SQL Developer Data Modeler is a free graphical tool that enhances productivity and simplifies data modeling tasks. Using Oracle SQL Developer Data Modeler users can create, browse and edit, logical, relational, physical, multi-dimensional, and data type models

viii. Data Architecture: The database has logical structures and physical structures. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

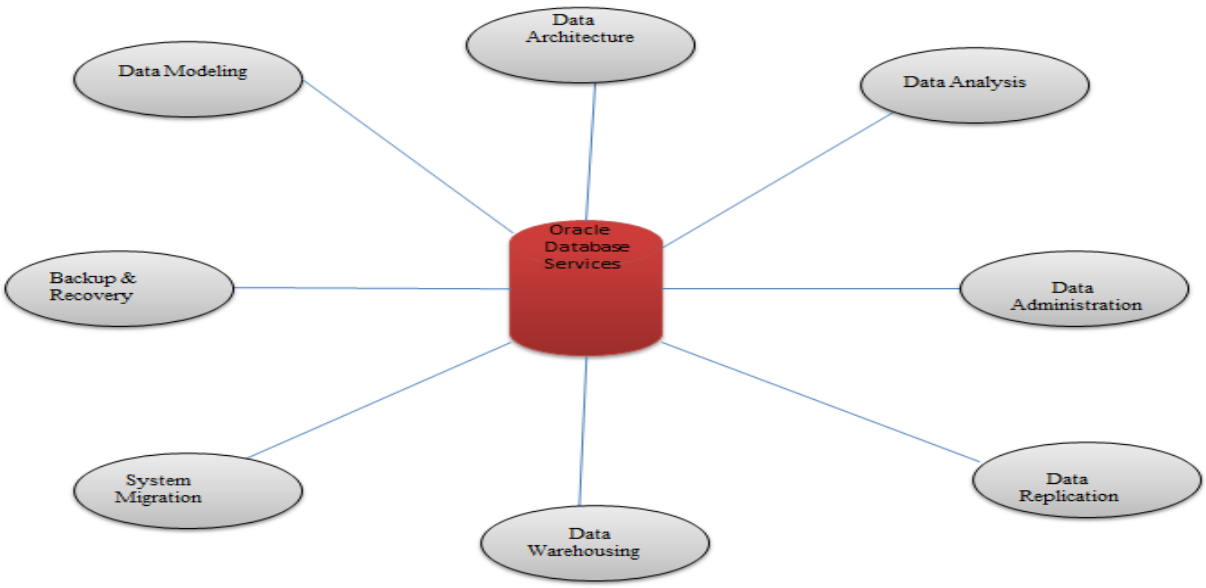


Figure 2: Oracle Services

3.2 ORACLE HISTORY

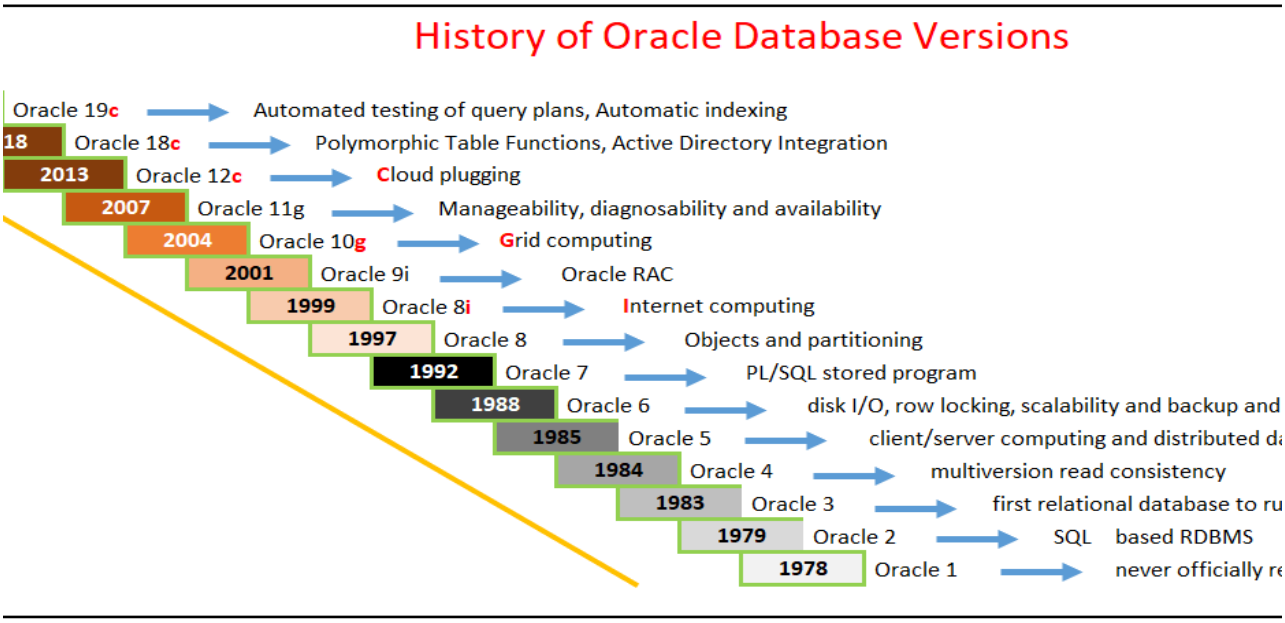


Figure 3: Oracle Database Versions

Larry Ellison and his two friends and former co-workers, Bob Miner and Ed Oates started a consultancy called Software Development Laboratories (SDL) in 1977. SDL developed the original version of Oracle Software.

3.3 RELATIONAL DATABASE MANAGEMENT SYSTEM

A relational database is a type of database that stores and provides access to data points that are related to one another. Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points. A Database Management System (DBMS) handles the way data is stored, maintained, and retrieved. In the case of

a relational database, a Relational Database Management System (RDBMS) performs these tasks. DBMS as used in this book is a general term that includes RDBMS.

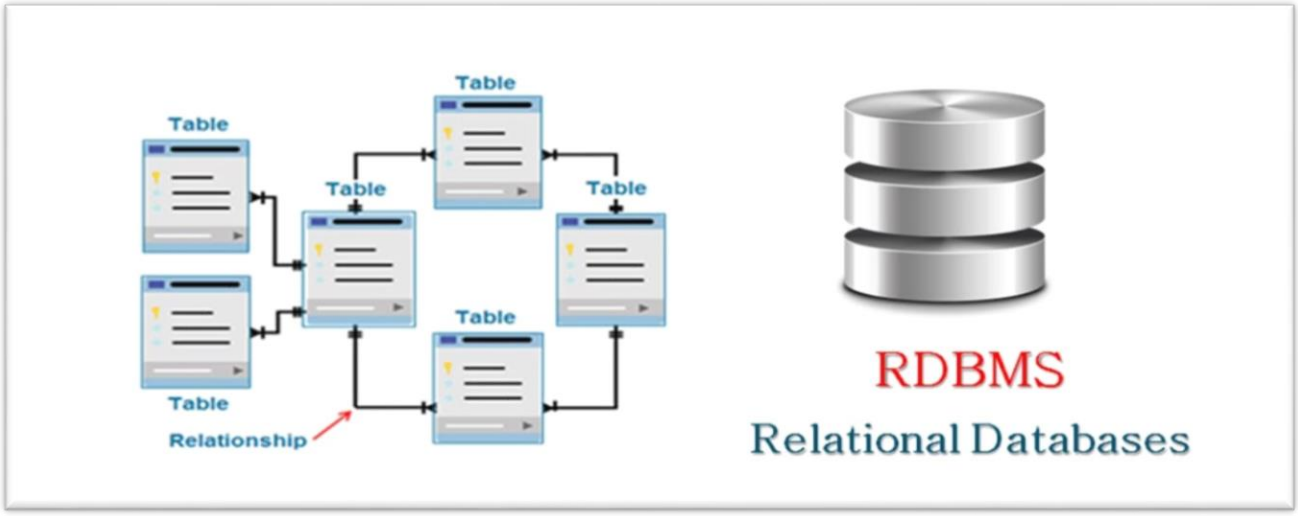


Figure 4 : Relational Database Management System

Advantages of RDBMS

- RDBMS is based on rows and columns (table). So it is easier to understand RDBMS work.
- RDBMS supports more than one user.
- As it is advanced it can easily handle huge amounts of data.
- Security is pretty good for RDBMS.

Disadvantages of RDBMS

- It is very costly.
- Special software is required for RDBMS

3.4 ORACLE FAMILY

Oracle Database is available in five editions, each suitable for different development and deployment scenarios. Oracle also offers several database options, packs, and other products that enhance the capabilities of Oracle Database for specific purposes.

- Oracle Database Standard Edition One : Oracle Database Standard Edition One delivers unprecedented ease of use, power, and performance for workgroup, department-level, and Web applications. From single-server environments for small business to highly distributed branch environments, Oracle Database Standard Edition One includes all the facilities necessary to build business-critical applications.
- Oracle Database Standard Edition : Oracle Database Standard Edition delivers the unprecedented ease of use, power, and performance of Standard Edition One, with support for larger machines and clustering of services with Oracle Real Application Clusters (Oracle RAC). Oracle RAC is not included in the Standard Edition of releases prior to Oracle Database 10g, nor is it an available option with those earlier releases.
- Oracle Database Enterprise Edition : Oracle Database Enterprise Edition provides the performance, availability, scalability, and security required for mission-critical applications such as high-volume online transaction processing (OLTP) applications, query-intensive data warehouses, and demanding Internet applications.
- Oracle Database Express Edition : Oracle Database Express Edition (Oracle Database XE) is an entry-level edition of Oracle Database that is quick to download, simple to install and manage, and is free to develop, deploy, and distribute. Oracle Database XE makes it easy to upgrade to the other editions of Oracle without costly and complex migrations. Oracle Database XE can be installed on any size machine with any number of CPUs, stores up to 11 GB of user data, using up to 1 GB of memory, and using only one CPU on the host machine. Support is provided by an online forum.

- Oracle Database Personal Edition : Oracle Database Personal Edition supports single-user development and deployment environments that require full compatibility with Oracle Database Standard Edition One, Oracle Database Standard Edition, and Oracle Database Enterprise Edition. Personal Edition includes all of the components that are included with Enterprise Edition, as well as all of the options that are available with Enterprise Edition, with the exception of the Oracle Real Application Clusters option, which cannot be used with Personal Edition.

CHAPTER 4 : STRUCTURE OF ORACLE

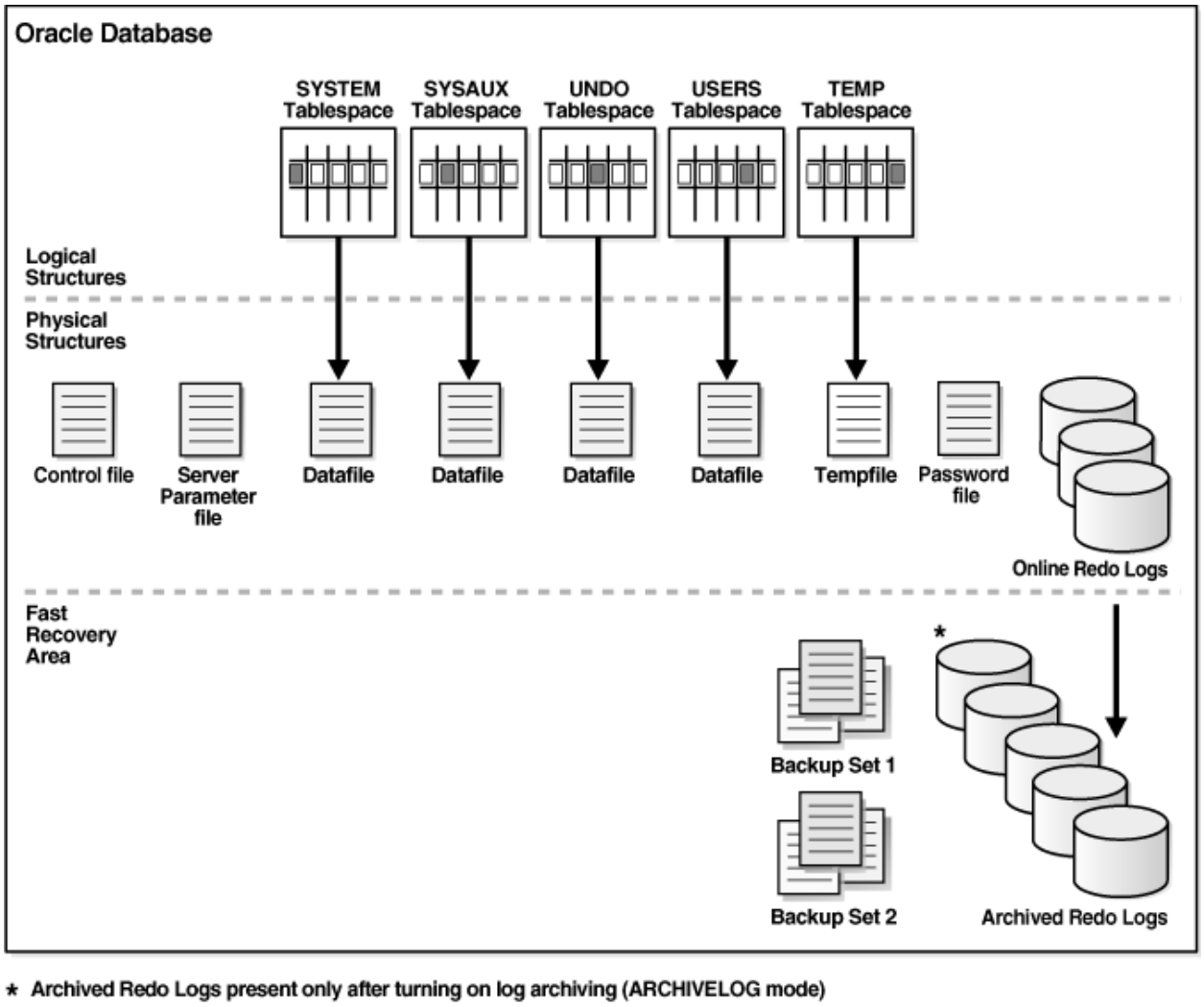


Figure 5 : Architecture of Oracle

An Oracle Database consists of at least one database instance and one database. The database instance handles memory and processes. The database consists of physical files called data files, and can be a non-container database or a multitenant container database.

Oracle Structure consists of :

- Physical Strucutres
- Logical Structures

- Oracle Instance
- SGA
- PGA

4.1 Physical Structure :

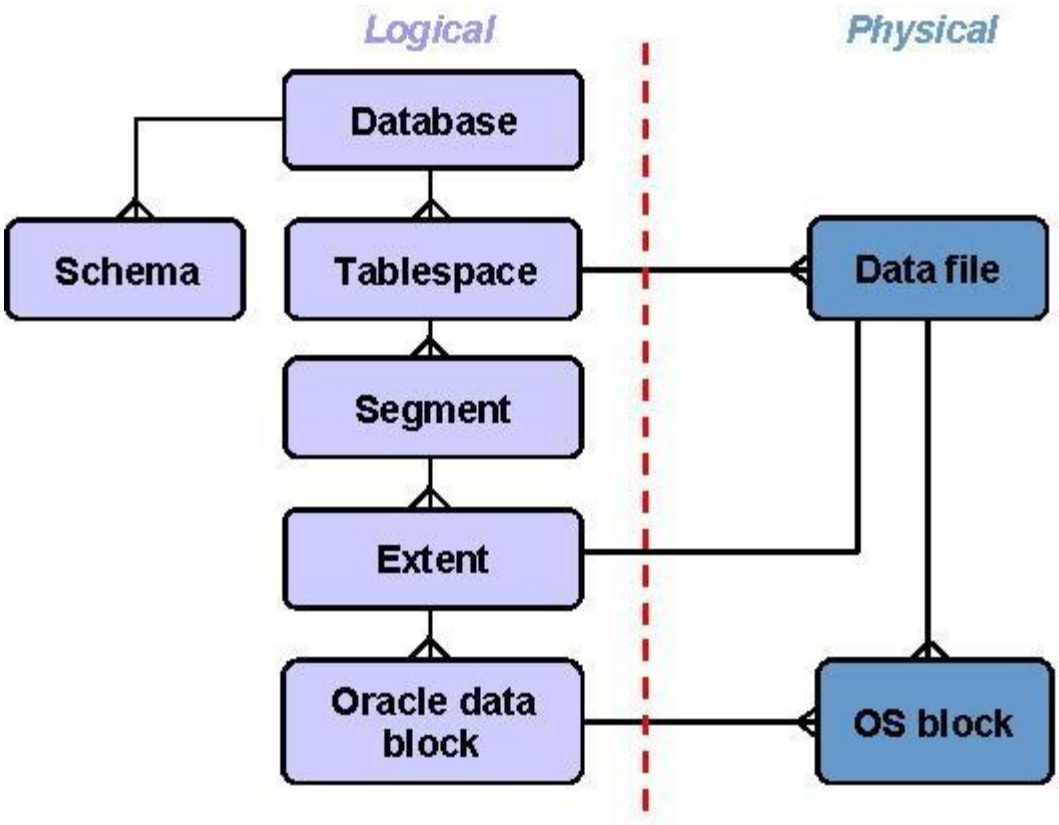


Figure 6 : Logical and Physical Structure Of Oracle

Physical Structure consists of :

- Datafiles (*.dbf) : The datafiles contain all the database data. The data of logical database structures, such as tables and indexes, is physically stored in the datafiles allocated for a database.

- Control Files (.ctl) : Every Oracle database has a control file. A control file contains entries that specify the physical structure of the database such as Database name and the Names and locations of datafiles and redo log files.
- Redo Log Files (*.log) : The primary function of the redo log is to record all changes made to data. If a failure prevents modified data from being permanently written to the datafiles, then the changes can be obtained from the redo log, so work is never lost.
- Archive Log Files (*.log) : Oracle automatically archives log files when the database is in ARCHIVELOG mode. This prevents oracle from overwriting the redo log files before they have been safely archived to another location.
- Parameter Files (initSID.ora) : Parameter files contain a list of configuration parameters for that instance and database.
- Alert and Trace Log Files (*.trc) : Each server and background process can write to an associated trace file. When an internal error is detected by a process, it dumps information about the error to its trace file. The alert log of a database is a chronological log of messages and errors.

4.2 Logical Structures :

Physical Structure consists of :

- Tablespaces : A database is divided into logical storage units called tablespaces, which group related logical structures together. One or more datafiles are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace.
- Oracle Data Blocks : At the finest level of granularity, Oracle database data is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on disk. The standard block size is specified by the DB_BLOCK_SIZE initialization parameter.

- Extents : The next level of logical database space is an extent. An extent is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.
- Segments : Above extents, the level of logical database storage is a segment. A segment is a set of extents allocated for a certain logical structure. The different types of segments are:
    - Data segment - stores table data
    - Index segment - stores index data
    - Temporary segment - temporary space used during SQL execution
    - Rollback Segment - stores undo information

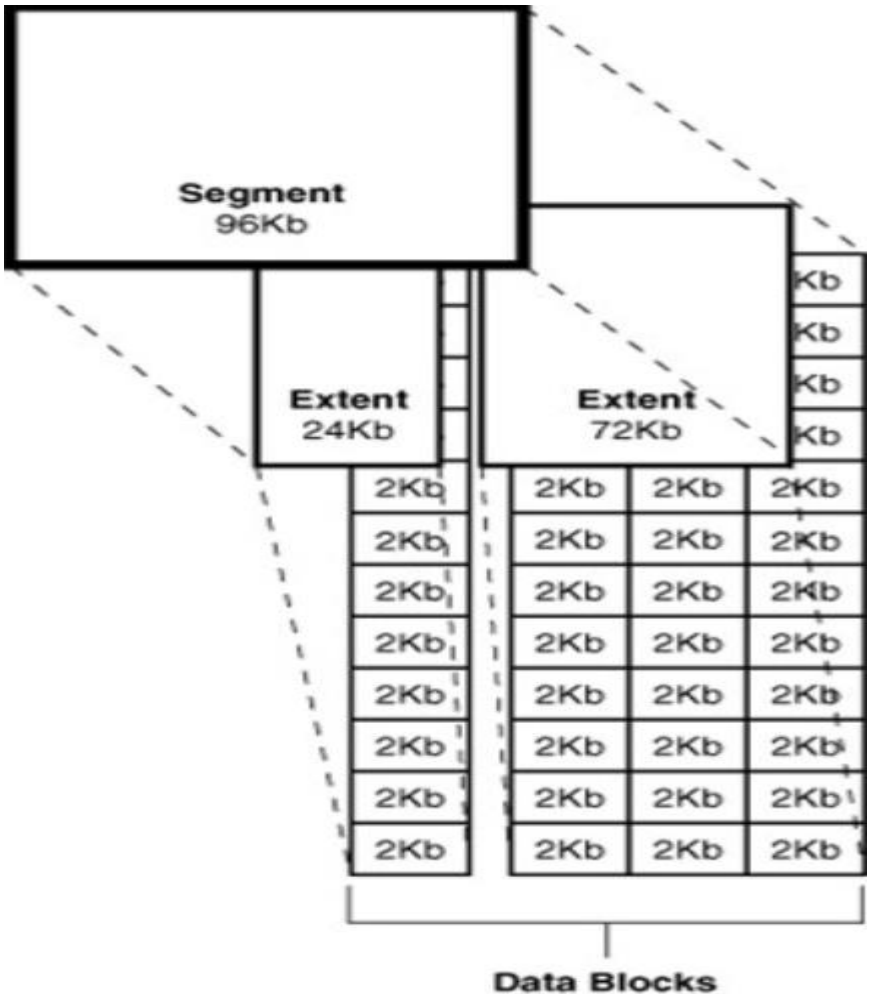


Figure 7 : Components of Logical Structure

4.3 ORACLE INSTANCE :

An Oracle database server consists of an Oracle database and an Oracle instance. Every time a database is started, a system global area (SGA) is allocated and Oracle background processes are started. The combination of the background processes and memory buffers is called an Oracle instance.

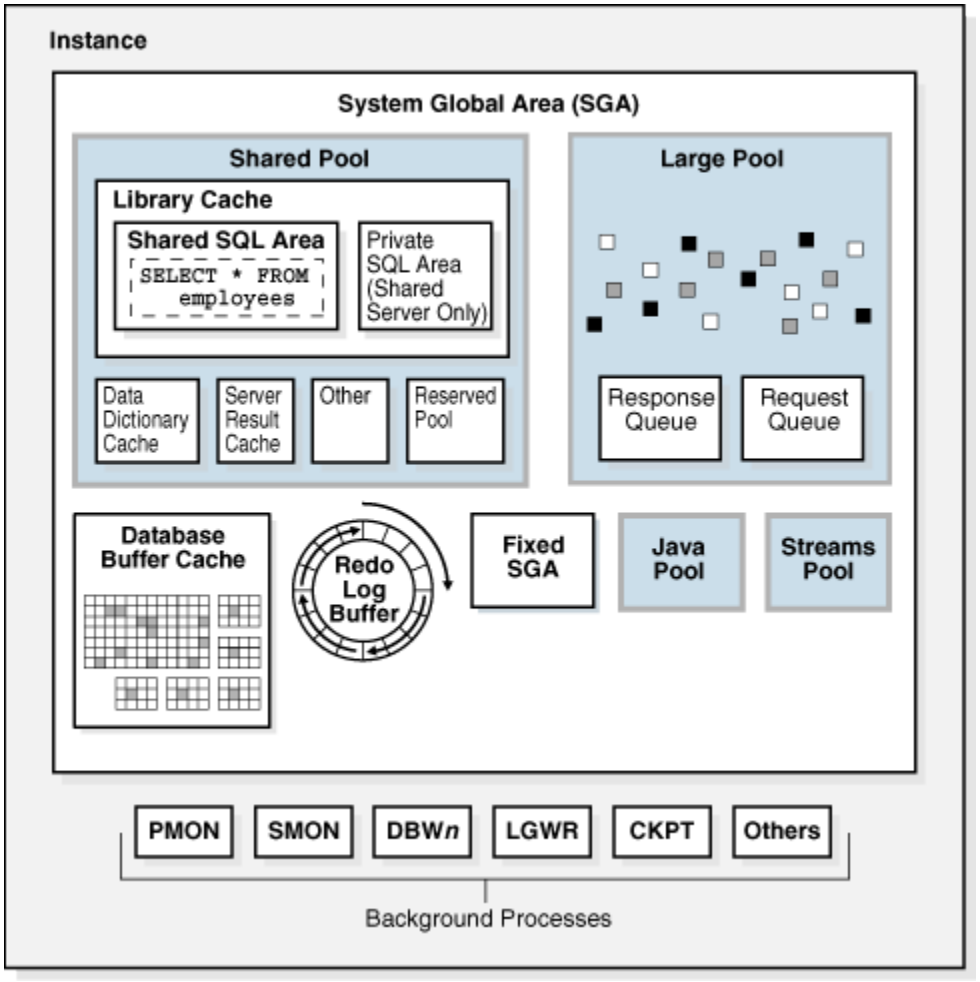


Figure 8 : Database Instance

4.4 SYSTEM GLOBAL AREA:

The System Global Area (SGA) is a shared memory region that contains data and control information for one Oracle instance. Users currently connected to an Oracle database share the data in the SGA. The SGA contains the following memory structures:

- Database Buffer Cache : Database buffers store the most recently used blocks of data. The set of database buffers in an instance is the database buffer cache. The buffer cache contains modified as well as unmodified blocks. Because the most recently (and often, the most frequently) used data is kept in memory, less disk I/O is necessary, and performance is improved.
- Redo Log Buffer of the SGA : The redo log buffer stores redo entries-a log of changes made to the database. The redo entries stored in the redo log buffers are written to an online redo log, which is used if database recovery is necessary. The size of the redo log is static.
- Shared Pool of the SGA : The shared pool contains shared memory constructs, such as shared SQL areas. A shared SQL area is required to process every unique SQL statement submitted to a database. A shared SQL area contains information such as the parse tree and execution plan for the corresponding statement.

4.5 PROGRAM GLOBAL AREA:

PGA is a memory buffer that contains data and control information for a server process. A server process is a process that services a client's requests. A PGA is created by oracle when a server process is started. The information in a PGA depends on the oracle configuration. The PGA area is a non-shared area of memory created by oracle when a server process is started.

The basic difference between SGA and PGA is that PGA cannot be shared between multiple processes in the sense that it is used only for requirements of a particular process whereas the SGA is used for the whole instance and it is shared.

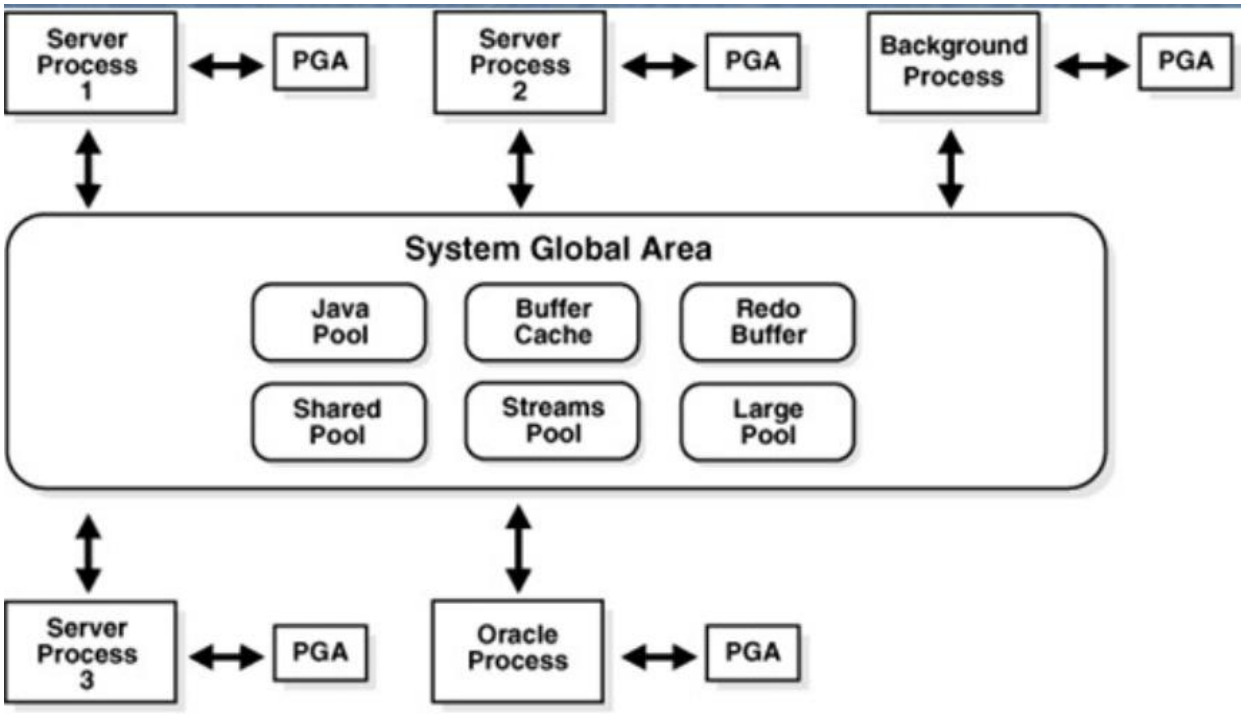


Figure 9 : SGA and PGA Relation

CHAPTER 5 : RECOVERY OF DATA

Recovery of data means rebuilding a database or table space if problems occur. It could be because of power interruption, application or storage failure.

To restore a physical backup of a datafile or control file is to reconstruct it and make it available to the Oracle database server. To recover a restored datafile is to update it by applying archived redo logs and online redo logs, that is, records of changes made to the database after the backup was taken. If you use RMAN, then you can also recover datafiles with incremental backups, which are backups of a datafile that contain only blocks that changed after a previous incremental backup.

After the necessary files are restored, media recovery must be initiated by the user. Media recovery involves various operations to restore, roll forward, and roll back a backup of database files.

Media recovery applies archived redo logs and online redo logs to recover the datafiles. Whenever a change is made to a datafile, the change is first recorded in the online redo logs. Media recovery selectively applies the changes recorded in the online and archived redo logs to the restored datafile to roll it forward.

To correct problems caused by logical data corruptions or user errors, you can use Oracle Flashback. Oracle Flashback Database and Oracle Flashback Table let you quickly recover to a previous time.

Types of recovery in Oracle include :

- Media Recovery
- RMAN and User-Managed Restore and Recovery
- Oracle Flashback Technology Recovery
- Instance and Crash Recovery

5.1 MEDIA RECOVERY:

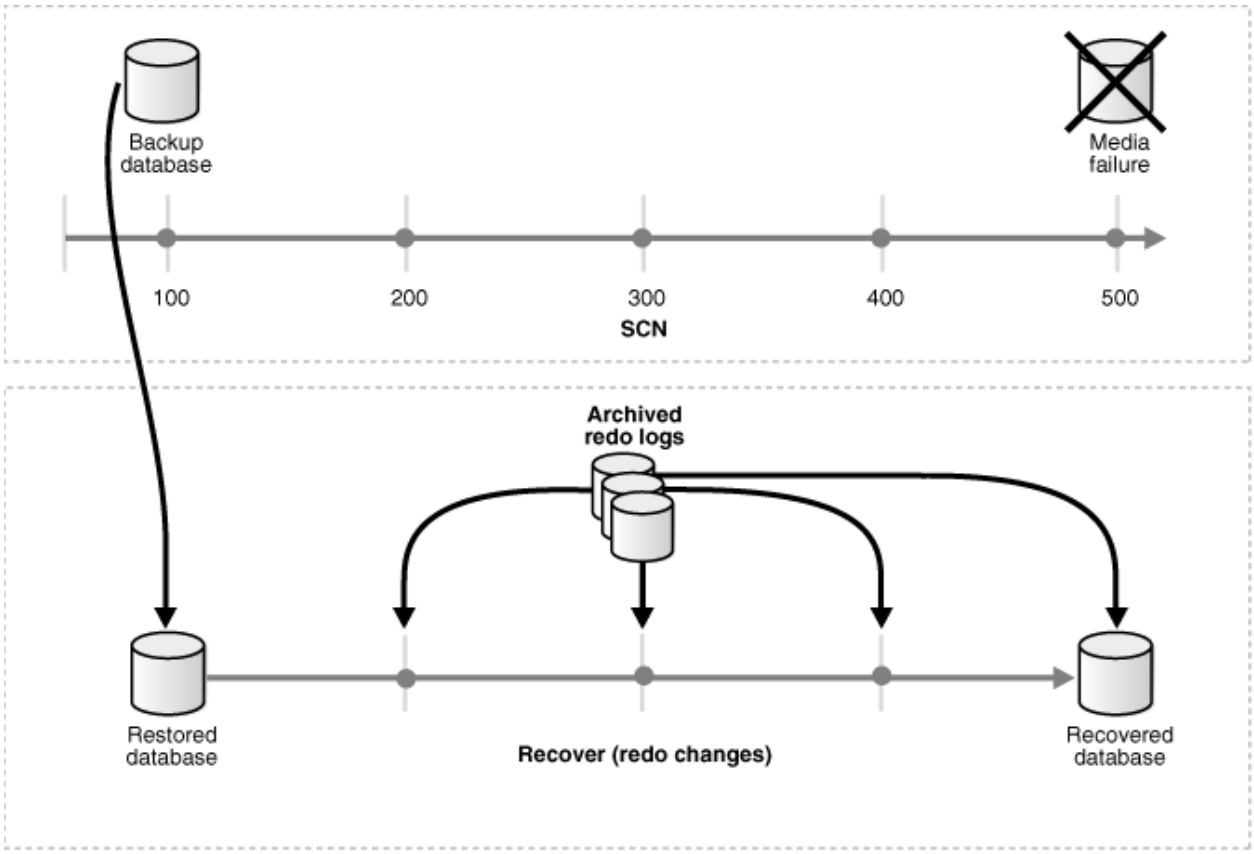


Figure 10 : Media Recovery

- The type of recovery that takes a backup and applies redo is called media recovery. Media recovery updates a backup to either to the current or to a specified prior time. Typically, the term "media recovery" refers to recovery of datafiles. Block media recovery is a more specialized operation that you use when just a few blocks in one or more files have been corrupted
- It contains :
    - Complete Recovery - Complete recovery involves using redo data or incremental backups combined with a backup of a database, tablespace, or datafile to update it to the most current point in time. It is called complete because Oracle applies all of the redo changes contained in the archived and online logs to the backup. Typically, you

perform complete media recovery after a media failure damages datafiles or the control file.

- Incomplete Recovery - Incomplete recovery, or point-in-time recovery, uses a backup to produce a noncurrent version of the database. In other words, you do not apply all of the redo records generated after the most recent backup. You usually perform incomplete recovery of the whole database in the following situations:
  - Media failure destroys some or all of the online redo logs.
  - A user error causes data loss, for example, a user inadvertently drops a table
  - You cannot perform complete recovery because an archived redo log is missing
  - You lose your current control file and must use a backup control file to open the database.

  To perform incomplete media recovery, you must restore all datafiles from backups created prior to the time to which you want to recover and then open the database with the RESETLOGS option when recovery completes.
  The RESETLOGS operation creates a new incarnation of the database—in other words, a database with a new stream of log sequence numbers starting with log sequence 1.

- Datafile Media Recovery - Datafile media recovery is used to recover from a lost or damaged current datafile or control file. It is also used to recover changes that were lost when a tablespace went offline without the OFFLINE NORMAL option. Both datafile media recovery and instance recovery must repair database integrity. However, these types of recovery differ with respect to their additional features. Media recovery has the following characteristics:

  - Applies changes to restored backups of damaged datafiles.

- Can use archived logs as well as online logs.
- Requires explicit invocation by a user.

- Block Media Recovery - Block media recovery is a technique for restoring and recovering individual data blocks while all database files remain online and available. If corruption is limited to only a few blocks among a subset of database files, then block media recovery might be preferable to datafile recovery.

  The interface to block media recovery is provided by RMAN. If you do not already use RMAN as your principal backup and recovery solution, then you can still perform block media recovery by cataloging into the RMAN repository the necessary user-managed datafile and archived redo log backups.

5.2 RMAN AND USER-MANAGED RESTORE AND RECOVERY:

You have a choice between two basic methods for recovering physical files. You can:

- Use the RMAN utility to restore and recover the database
- Restore backups by means of operating system utilities, and then recover by running the SQL*Plus RECOVER command

Whichever method you choose, you can recover a database, tablespace, or datafile. Before performing media recovery, you need to determine which datafiles to recover. Often you can use the fixed view V$RECOVER_FILE. This view lists all files that require recovery and explains the error that necessitates recovery.

- RMAN Restore And Recovery - The basic RMAN recovery commands are RESTORE and RECOVER. Use RESTORE to restore datafiles from backup sets

or from image copies on disk, either to their current location or to a new location. You can also restore backup sets containing archived redo logs, but this is usually unnecessary, because RMAN automatically restores the archived logs that are needed for recovery and deletes them after the recovery is finished. Use the RMAN RECOVER command to perform media recovery and apply archived logs or incremental backups. RMAN automates the procedure for recovering and restoring your backups and copies.

- User-Managed Restore And Recovery - If you do not use RMAN, then you can restore backups with operating system utilities and then run the SQL*Plus RECOVER command to recover the database. You should follow these basic steps:

    - After identifying which files are damaged, place the database in the appropriate state for restore and recovery. For example, if some but not all datafiles are damaged, then take the affected tablespaces offline while the database is open.
    - Restore the files with an operating system utility. If you do not have a backup, it is sometimes possible to perform recovery if you have the necessary redo logs dating from the time when the datafiles were first created and the control file contains the name of the damaged file.
      If you cannot restore a datafile to its original location, then relocate the restored datafile and change the location in the control file.
    - Restore any necessary archived redo log files.
    - Use the SQL*Plus RECOVER command to recover the datafile backups.

5.3 ORACLE FLASHBACK TECHNOLOGY RECOVERY:

To correct problems caused by logical data corruptions or user errors, you can use Oracle Flashback. Flashback Database and Flashback Table let you quickly recover to a previous time.

It consists of:

- Oracle Flashback Database - Oracle Flashback Database lets you quickly recover an Oracle database to a previous time to correct problems caused by logical data corruptions or user errors.

  If an Oracle managed disk area, called a flash recovery area is configured, and if you have enabled the Flashback functionality, then you can use the RMAN and SQL FLASHBACK DATABASE commands to return the database to a prior time. Flashback Database is not true media recovery, because it does not involve restoring physical files. However, Flashback is preferable to using the RESTORE and RECOVER commands in some cases, because it is faster and easier, and does not require restoring the whole database.

  To flashback a database, Oracle uses past block images to back out changes to the database. During normal database operation, Oracle occasionally logs these block images in Flashback logs. Flashback logs are written sequentially, and they are not archived. Oracle automatically creates, deletes, and resizes Flashback logs in the flash recovery area. You only need to be aware of Flashback logs for monitoring performance and deciding how much disk space to allocate to the flash recovery area for Flashback logs.

  The amount of time it takes to Flashback a database is proportional to how far back you need to revert the database, rather than the time it would take to restore and recover the whole database, which could be much longer. The before images in the Flashback logs are only used to restore the database to a point in the past, and forward recovery is used to bring the database to a consistent state at some time in the past. Oracle returns datafiles to the previous point-in-time, but not auxiliary files, such as initialization parameter files.

- Oracle Flashback Table - Oracle Flashback Table lets you recover tables to a specified point in time with a single statement. You can restore table data along with associated indexes, triggers, and constraints, while the database is online, undoing changes to only the specified tables. Flashback Table does not address physical corruption; for example, bad disks or data segment and index inconsistencies.

  Flashback Table works like a self-service repair tool. Suppose a user accidentally deletes some important rows from a table and wants to recover the deleted rows. You can restore the table to the time before the deletion and see the missing rows in the table with the FLASHBACK TABLE statement.

  For Flashback Table to succeed, the system must retain enough undo information to satisfy the specified SCN or timestamp, and the integrity constraints specified on the tables cannot be violated. Also, row movement must be enabled.

  The point of time in the past that you use Flashback Table to go to is controlled by the undo retention of the system. Oracle Database 10g automatically tunes a parameter called the undo retention period. The undo retention period indicates the amount of time that must pass before old undo information—that is, undo information for committed transactions—can be overwritten. The database collects usage statistics and tunes the undo retention period based on these statistics and on undo tablespace size.

5.4 INSTANCE AND CRASH RECOVERY:

Crash recovery is used to recover from a failure either when a single-instance database fails or all instances of an Oracle Real Application Clusters database fail. Instance recovery refers to the case where a surviving instance recovers a failed instance in an Oracle Real Application Clusters database.

The goal of crash and instance recovery is to restore the data block changes located in the cache of the terminated instance and to close the redo thread that was left open. Instance and crash recovery use only online redo log files and current online datafiles. Oracle recovers redo threads of the terminated instances together.

Crash and instance recovery involve two distinct operations: rolling forward the current, online datafiles by applying both committed and uncommitted transactions contained in online redo records, and then rolling back changes made in uncommitted transactions to their original state.

Crash and instance recovery have the following shared characteristics:

- Redo the changes using the current online datafiles (as left on disk after the failure or SHUTDOWN ABORT)
- Use only the online redo logs and never require the use of the archived logs
- Have a recovery time governed by the number of terminated instances, amount of redo generated in each terminated redo thread since the last checkpoint, and by user-configurable factors such as the number and size of redo log files, checkpoint frequency, and the parallel recovery setting

Oracle performs this recovery automatically on two occasions:

- At the first database open after the failure of a single-instance database or all instances of an Oracle Real Applications Cluster database (crash recovery).
- When some but not all instances of an Oracle Real Application Clusters configuration fail (instance recovery). The recovery is performed automatically by a surviving instance in the configuration.

The important point is that in both crash and instance recovery, Oracle applies the redo automatically: no user intervention is required to supply redo logs. However, you can set parameters in the database server that can tune the duration of instance and crash recovery performance. Also, you can tune the rolling forward and rolling back phases of instance recovery separately.

CHAPTER 6 : CONCLUSION

Database is used by many users to store and manage sensitive data of personal or enterprise. Furthermore, with the increased use of Internet of Things devices, database allows countless users to access a variety of applications and stores the users' data and log; database forensics is becoming more important for a forensic investigator. Although some previous researches proposed an universal investigation method to DBMSes, there is insufficient information on how to investigate DBMS and recover deleted records practically.

The redo logs can be a rich source of evidence for a forensic examiner when they are investigating a compromised Oracle database server. I can potentially find evidence of authentication attacks in both the TNS Listener's log file and the audit trail.

In this seminar, I have researched Oracle, which is the most used DBMS globally. I have described the internal structure of Oracle including. To recover deleted records I have seen different methods. I have proposed a method to recover deleted records and implemented the method as open-source tool.

Though I have focused on one DBMS, Oracle, there are many DBMSes that need to be analyzed and new DBMSes are still being developed. In database forensics, there is a lack of practical researches to investigate these DBMSes.

REFERENCES

[1]D. Litchfield, ‘‘Oracle forensics part 1: Dissecting the redo logs,’’ NGSSoftware Insight Secur. Res. (NISR), Next Gener. Secur. Softw. Ltd., Sutton, U.K., Tech. Rep., 2007.

[2] D. Litchfield, ‘‘Oracle forensics part 2: Locating dropped objects,’’ NGSSoftware Insight Secur. Res. (NISR), Next Gener. Secur. Softw. Ltd., Sutton, U.K., Tech. Rep., 2007.

[3] D. Litchfield, ‘‘Oracle forensics part 3: Isolating evidence of attacks against the authentication mechanism,’’ NGSSoftware Insight Secur. Res. (NISR), Next Gener. Secur. Softw. Ltd., Sutton, U.K., Tech. Rep., 2007.

[4] D. Litchfield, ‘‘Oracle forensics part 4: Live response,’’ NGSSoftware Insight Secur. Res. (NISR), Next Gener. Secur. Softw. Ltd., Sutton, U.K., Tech. Rep., 2007.

[5] D. Litchfield, ‘‘Oracle forensics part 5: Finding evidence of data theft in the absence of auditing,’’ NGSSoftware Insight Secur. Res. (NISR), Next Gener. Secur. Softw. Ltd, Sutton, U.K., Tech. Rep., 2007.

[6] D. Litchfield, ‘‘Oracle forensics part 6: Examining undo segments, flashback and the oracle recycle bin,’’ NGSSoftware Insight Secur. Res. (NISR), Next Gener. Secur. Softw. Ltd, Sutton, U.K., Tech. Rep., 2007.

[7] D. Litchfield, ‘‘Oracle forensics part 7: Using the oracle system change number in forensic investigations,’’ NGSSoftware Insight Secur. Res. (NISR), Next Gener. Secur. Softw. Ltd, Sutton, U.K., Tech. Rep., 2008.

Annexure A: Glossary

8. Forensics : scientific tests or techniques used in connection with the detection of crime.
9. Database : a large amount of data that is stored in a computer and can be easily used, added to, etc.
10. RDBMS: Relational Database Management System
11. Servers : a computer that stores information that a number of computers can share
12. SGA: System Global Area
13. PGA: Program Global Area
14. RMAN: oracle recovery manager

Annexure B: Reference Paper

1. Title: Forensic Recovery of SQL Server Database: Practical Approach

2. Published In: Journal IEEE Access Volume 9

ISSN: 14564-14575

Annexure C: Plagiarism Report

Page Count: 37

Word Count: 5720

Character Count: 30603

Unique: 78%

Plagiarism: 22%

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By

Student Name	**Exam Seat No.**
Abhishek Dhamdhere	
Shyam Lokhande	
Neha Bhosale	
Vaishnav Bhujbal	

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

DATA RECOVERY USING MySQL AND SQLite

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By

Vaishnav Bhujbal

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

CERTIFICATE

This is to certify that the Seminar report entitled, "**FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH"** Submitted by,

Student Name	**Exam Seat No.**
Abhishek Dhamdhere	
Shyam Lokhande	
Neha Bhosale	
Vaishnav Bhujbal	

is a record of bonafide work carried out by him/her, in the partial fulfillment of the Term-work of Third year in Computer Engineering of Savitribai Phule Pune University at Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune under Savitribai Phule Pune University, Pune. This work is done during the academic year 2022-23.

Date: - 19/10/2022

Place: - Pune

Prof. B.C.Julme	Prof. M.S.Pokale	Prof.D.D.Sapkal
Seminar Guide	External Examiner	H.O.D. (Computer Engg)

Acknowledgment

I would like to express my gratitude towards my guide, Prof.B.C.Julme, Assistant Professor in the Computer Engineering Department, who has been very concerned and has aided for all the help essential for the preparation of this work. He has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.

I am also thankful to Prof. D.D.Sapkal, Head of Department, Computer Engineering, for the motivation and inspiration that triggered me for the seminar Work.

Vaishnav Bhujbal

T3

(T.E.Computer Engineering)

Abstract for MySQL

For various reasons, circumstances might arise in which an investigator, enlisting the help of a system administrator, needs access to an instance of MySQL that is password protected by an individual system user. If this password is unknown and the user is uncooperative or unavailable, alternative means must be utilized to gain access to the data stored within the program. Two main approaches will be explored, each with its pros and cons. In one case, the password can be bypassed entirely, granting the investigator unfettered access to the program data. The second method allows a narrower look at only some of the data. Either method will give the investigator, at least to some extent, the opportunity to overcome the problem of a missing or uncooperative system user.

INDEX

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Problem

Since MySQL employs a password to protect the information stored by the user, problems can arise if said user forgets the password, becomes unavailable, or uses the program for nefarious purposes and thus becomes uncooperative in providing the password to his or her superiors for investigation. As such, it is prudent to search for ways in which a user with higher system permissions can bypass the password. This would allow for an investigator to engage in his or her investigation or data recovery effort without the need to elicit the cooperation of the user.

1.2 Objective

In response to the aforementioned problem, two methods will be explored that allow an investigator to bypass the password requirement imposed by lower level system users running the database management system MySQL.

1.3 Approach

Two main approaches will be explored in this thesis. Either way will result in access to at least some of the stored data as well as a history file that shows the command history of MySQL. The first method requires that the program files be copied into a new instance of MySQL on another machine (or virtual machine) where the password is known. This will reveal the databases and their entries exactly as they would appear should the password be entered on the original instance of MySQL.

The second technique involves looking at the raw program data files and sifting through the encoding to obtain information of interest. This method reveals table structure and some of the entry data. However, it does not provide a complete view of the database as not all data types are encoded in the program data files in a manner easily readable by unaided human eyes.

1.4 Organization of this Thesis

Chapter 2 covers background information concerning different user roles in a Computer system as well as the security applied by the MySQL program itself. Chapter 3 describes in detail the two methods by which an investigator can bypass the password prompt in MySQL and demonstrates that these methods are effective. Chapter 4 provides a concluding overview of the thesis.

CHAPTER 2: BACKGROUND

2.1 Permission Levels

When dealing with computer security, one should realize that there are different levels of access. An individual user typically has access only to his or her files on the computer and often is not allowed to access system files or make substantial changes to the system. This responsibility and privilege falls to the system administrator who has a “master key” of sorts in the form of an administrative password. This password grants the administrator access to nearly every aspect of the system, including both user and system files. In some ways, this makes sense. The system administrator is typically employed by the entity that owns the system. As such, the administrator should have full access to all files on the system to perform duties ranging from simple maintenance to investigations to intrusion detection and everything in between. On the other hand, this lends itself to a clear lack of privacy on the part of the individual user. Though the user has a password for his or her account and may even have a password to access certain programs or files within that account, the system administrator can often bypass such safeguards with the aforementioned “master key.” Such a situation arises with one of the most widely used open-source relational database management systems – MySQL.

2.2 MySQL Security

Upon initial setup, MySQL prompts the user to set a password that is intended to prevent unauthorized access to the program or, more accurately, to the information stored by the program. Failure to enter the correct password results in “access denied” errors that prevent the user from accessing the databases or the information stored within them.
This is good in order to promote security and confidentiality, but problems arise should the password be forgotten or the password holder hide unscrupulous behavior within the MySQL program and refuse to divulge the password upon questioning. If there is a way to bypass the password requirement, then the data can be recovered from either scenario, but it could also be recovered, potentially, by unauthorized persons. Some systems, such as those used in this thesis, require system administrator privileges in order to install MySQL. As such, the system administrator could set the password for MySQL. However, this thesis rests on the idea that the system user is allowed to set the password.

CHAPTER 3: APPROACH

3.1 Two Approaches

Investigation into the MySQL database reveals that there are at least two ways to recover the password-protected information, and the two known ways will be discussed in the following pages of this thesis. The first way, henceforth referred to as the “Copy Method,” involves invoking administrator access in order to manually copy program data files, while the second requires sifting through these files in a command line environment.
The Copy Method is preferred as it produces the information in an easily readable format and appears just as it would if the password to the program were known. However, the second way, henceforth termed the “Plainview Method,” has the potential to show entries that have been deleted from the table, allowing for historical look at the

data. On the other hand, certain data types are not readable, or at the very least not easily, when looking directly at the program files, which appear “garbled” without the MySQL program to decode and display the information they contain.
Regardless of which method is employed, the relevant system files are the same. While MySQL creates a number of files and folders, the only ones of interest are those associated with the stored data and data structures. Which files these are and where they are located depend on a number of factors concerning the version of MySQL being run and some of its configuration settings. The version of MySQL being used for this thesis was 5.5.32, meaning that by default the storage engine is InnoDB and the innodb_file_per_table option is turned off.
Given these facts, the files of interest include the following: ibdata1, ib_logfile0, ib_logfile1, the db.opt file associated with each individual database, and all .frm files associated with each individual table within the databases.Additionally, the file .mysql_history can be recovered to show a history of the command line inputs within the MySQL program.
The .mysql_history file is found in the home directory. The location for the other files can be found in the my.cnf file located in the /etc/mysql/ folder. By default on Linux, MySQL’s data and folder structure are found at / var /lib / mysql as indicated by the value of the “datadir” variable.

```
acl@zero:/etc/mysql$ ls -al
total 24
drwxr-xr-x  3 root root 4096 Oct 28 17:29 .
drwxr-xr-x 87 root root 4096 Oct 28 17:30 ..
drwxr-xr-x  2 root root 4096 Oct 28 17:29 conf.d
-rw-------  1 root root  333 Oct 28 17:29 debian.cnf
-rwxr-xr-x  1 root root 1220 Oct 22 13:25 debian-start
-rw-r--r--  1 root root 3505 Oct 22 15:09 my.cnf
acl@zero:/etc/mysql$ cat my.cnf
#
# The MySQL database server configuration file.
#
# You can copy this to one of:
# - "/etc/mysql/my.cnf" to set global options,
# - "~/.my.cnf" to set user-specific options.
#
# One can use all long options that the program supports.
# Run program with --help to get a list of available options and with
# --print-defaults to see which it would actually understand and use.
#
# For explanations see
# http://dev.mysql.com/doc/mysql/en/server-system-variables.html

# This will be passed to all mysql clients
# It has been reported that passwords should be enclosed with ticks/quotes
# escpecially if they contain "#" chars...
# Remember to edit /etc/mysql/debian.cnf when changing the socket location.
[client]
port            = 3306
socket          = /var/run/mysqld/mysqld.sock

# Here is entries for some specific programs
# The following values assume you have at least 32M ram

# This was formally known as [safe_mysqld]. Both versions are currently parsed.
[mysqld_safe]
socket          = /var/run/mysqld/mysqld.sock
nice            = 0

[mysqld]
#
# * Basic Settings
#
user            = mysql
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
port            = 3306
basedir         = /usr
datadir         = /var/lib/mysql
tmpdir          = /tmp
lc-messages-dir = /usr/share/mysql
```

Figure 1: Partial display of my.cnf file showing that the data directory (datadir) is/var/lib/mysql

The files listed contain the necessary data to reconstruct the databases and their associated entries. The metadata and the table data are stored in the ibdata1 file. It also contains the indexes for the tables that keep the data organized and available for retrieval.
As their name suggests, ib_logfile0 and ib_logfile1 are log files that help MySQL recovery from crashes by keeping record of commands that made or attempted to make changes to the tables.
Each database has one db.opt file and, assuming it has any tables, about the database. There will exist one .frm file for every table created within a database. This .frm file stores metadata about its table. Because the InnoDB engine also stores this metadata in the aforementioned ibdata1 file, it is not always necessary to include the .frm files just to view the database as it was. It is recommended, however, as having these files may be required in order to do a backup at a later date, should that prove to be necessary.
Including them in the move with the other files is the safest option since it keeps the file structure as similar to the original instance of MySQL as possible.

3.2 The Copy Method

3.2.1 Procedure of the Copy Method

The essence of the Copy Method is to use system administrator privileges to access the relevant program files where MySQL stores the data and the table structures and references. These files are then copied into another instance of MySQL the password of which is known. The source files (those being copied) will replace any identically named existing files in the new instance of MySQL. It is important to note, then, that this new instance of MySQL must be a clean instance and not have any databases, tables, or entries associated with it. It is vital that the folder structure and the permissions for each file and folder be observed as well. This technique is dependent on putting the files where
MySQL will expect them to be when it is run and commanded to use a certain database and table. In a sense, the second instance of MySQL is being tricked into reading files that it never created but that somehow just showed up in its file structure. The following example should be followed in order to illuminate the details of the Copy Method along with its associated commands.
The first virtual machine is the source; that is, it has the files the investigator is interested and is the instance of MySQL protected by the password unknown to the investigator. It was set up with the password “password00” and a few databases, tables, and entries, as demonstrated in part below.

mysql> create database Dogs;
Query OK, 1 row affected (0.00 sec)

mysql> use Dogs
Database changed

mysql> create table Boarders (Namevarchar(20), Breed varchar(20),Sex char(1), Age int, Birthday DATE, Owner varchar(20));
Query OK, 0 rows affected (0.01 sec)

mysql> describe Boarders;

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| Name     | varchar(20) | YES  |     | NULL    |       |
| Breed    | varchar(20) | YES  |     | NULL    |       |
| Sex      | char(1)     | YES  |     | NULL    |       |
| Age      | int(11)     | YES  |     | NULL    |       |
| Birthday | date        | YES  |     | NULL    |       |
| Owner    | varchar(20) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Figure 2:Description of the Boarders table in database Dogs

mysql> select * from Boarders;
Empty set (0.00 sec)

Mysql >insert into Boarders values ('Apollo', 'Golden Retriver Mix', 'M', '12', '2001`02`12', 'Sally');

Query OK, 1row affected (0.00 sec)

Similar statements were used to insert two other entries into the table. There result is Shown in Figure 3.

```
mysql> select * from Boarders;
+--------+----------------------+------+------+------------+-------+
| Name   | Breed                | Sex  | Age  | Birthday   | Owner |
+--------+----------------------+------+------+------------+-------+
| Apollo | Golden Retriver Mix | M    |   12 | 2001-02-12 | Sally |
| Samson | German Shepherd Dog | M    |   13 | 0000-00-00 | Sally |
| Chloe  | Pomeranian           | F    |    3 | 2010-05-14 | Jack  |
+--------+----------------------+------+------+------------+-------+
3 rows in set (0.00 sec)
```

Figure 3: Data entries for the Boarders table in database Dogs

The above process was repeated to create and populate a table named Strays, also in The database Dogs, resulting in Figure 4.An UPDATE command was given to demonstrate how it appears in each method. The effects of the UPDATE command are given in Figure 5.

```
mysql> select * from Strays;
+-----------+-----------+------+------------+-------------+--------------+----------------+
| Name      | Breed     | Sex  | Age        | WeightInLbs | DateCheckedIn | DateCheckedOut |
+-----------+-----------+------+------------+-------------+--------------+----------------+
| JaneDoe   | Pitbull   | F    | Adult      |          42 | 2013-10-10    | 0000-00-00     |
| Lightning | Greyhound | M    | Adolescent |          85 | 2013-10-08    | 2013-10-09     |
+-----------+-----------+------+------------+-------------+--------------+----------------+
2 rows in set (0.00 sec)
```

Figure 4: Data entries for the Strays table in database Dogs prior to the UPDATE command

mysql> update Strays set Name= 'Sweetie'where Name= 'Jane Doe' and Breed='Pitbull' and DateCheckedIn='2013`10`10';

Query OK, 1row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

```
mysql> select * from Strays;
+-----------+-----------+------+------------+-------------+--------------+----------------+
| Name      | Breed     | Sex  | Age        | WeightInLbs | DateCheckedIn | DateCheckedOut |
+-----------+-----------+------+------------+-------------+--------------+----------------+
| Sweetie   | Pitbull   | F    | Adult      |          42 | 2013-10-10    | 0000-00-00     |
| Lightning | Greyhound | M    | Adolescent |          85 | 2013-10-08    | 2013-10-09     |
+-----------+-----------+------+------------+-------------+--------------+----------------+
2 rows in set (0.00 sec)
```

Figure 5: Data entries for the Strays table in database Dogs after the UPDATE command

A second database, Cats, was created. Similar to database Dogs, it also has two tables Named Boarders and Strays. These were populated using INSERT statements. A DELETE command was also input to show how it appears in each method. The final Forms of these two tables are shown in Figure 6 and Figure 7.

```
mysql> select * from Boarders;
+-----------+------+------+--------------+---------+
| Name      | Sex  | Age  | Birthday     | Owner   |
+-----------+------+------+--------------+---------+
| Speckles  | F    |    6 | 2007-01-17   | Junior  |
| DC        | M    |   10 | 2003-04-21   | Missy   |
+-----------+------+------+--------------+---------+
2 rows in set (0.00 sec)
```

Figure 6: Data entries for the Boarders table in database Cats

mysql>delete from Strays where Name='Jackie';

&
Query OK, 1 row affected(0.00 sec)

```
mysql> select * from Strays;
+---------+------+--------+-------------+--------------+----------------+
| Name    | Sex  | Age    | WeightInLbs | DateCheckedIn | DateCheckedOut |
+---------+------+--------+-------------+--------------+----------------+
| JaneDoe | F    | Kitten |          12 | 2013-10-05    | 0000-00-00     |
+---------+------+--------+-------------+--------------+----------------+
1 row in set (0.00 sec)
```

Figure 7: Data entry for the Strays table in database Cats

The data is now shown to be stored and password protected by the source machine's instance of MySQL. Using the command "sudos " and inputting the
1 Administrative password allows a look at the files MySQL created in the above process. These files are located at the file path/var/lib/mysq . Prior to the data creation, it appeared as shown in Figure 8.

```
total 28696
drwx------  5 mysql mysql     4096 Oct 27 16:31 .
drwxr-xr-x 35 root  root      4096 Oct 24 22:41 ..
-rw-r--r--  1 root  root         0 Oct 24 22:41 debian-5.5.flag
-rw-rw----  1 mysql mysql 18874368 Oct 27 16:29 ibdata1
-rw-rw----  1 mysql mysql  5242880 Oct 27 16:29 ib_logfile0
-rw-rw----  1 mysql mysql  5242880 Oct 24 22:41 ib_logfile1
drwx------  2 mysql root      4096 Oct 24 22:41 mysql
-rw-rw----  1 root  root         6 Oct 24 22:41 mysql_upgrade_info
drwx------  2 mysql mysql     4096 Oct 24 22:41 performance_schema
drwx------  2 mysql root      4096 Oct 24 22:41 test
```

Figure 8: File structure before data creation

But it now looks like Figure 9.

```
total 28704
drwx------  7 mysql mysql     4096 Oct 27 16:27 .
drwxr-xr-x 35 root  root      4096 Oct 24 22:41 ..
drwx------  2 mysql mysql     4096 Oct 27 16:28 Cats
-rw-r--r--  1 root  root         0 Oct 24 22:41 debian-5.5.flag
drwx------  2 mysql mysql     4096 Oct 27 16:26 Dogs
-rw-rw----  1 mysql mysql 18874368 Oct 27 16:29 ibdata1
-rw-rw----  1 mysql mysql  5242880 Oct 27 16:29 ib_logfile0
-rw-rw----  1 mysql mysql  5242880 Oct 24 22:41 ib_logfile1
drwx------  2 mysql root      4096 Oct 24 22:41 mysql
-rw-rw----  1 root  root         6 Oct 24 22:41 mysql_upgrade_info
drwx------  2 mysql mysql     4096 Oct 24 22:41 performance_schema
drwx------  2 mysql root      4096 Oct 24 22:41 test
```

Figure 9: File structure after data creation

With subfolders Dogs and Cats appearing in Figure 10 and Figure 11.

```
root@zero:/var/lib/mysql/Dogs# ls -al
total 36
drwx------ 2 mysql mysql 4096 Oct 28 17:32 .
drwx------ 7 mysql mysql 4096 Oct 28 17:33 ..
-rw-rw---- 1 mysql mysql 8718 Oct 28 17:32 Boarders.frm
-rw-rw---- 1 mysql mysql   65 Oct 28 17:32 db.opt
-rw-rw---- 1 mysql mysql 8790 Oct 28 17:32 Strays.frm
```

Figure 10: Files created within the Dogs database

```
root@zero:/var/lib/mysql/Cats# ls -al
total 36
drwx------ 2 mysql mysql 4096 Oct 28 17:33 .
drwx------ 7 mysql mysql 4096 Oct 28 17:33 ..
-rw-rw---- 1 mysql mysql 8686 Oct 28 17:33 Boarders.frm
-rw-rw---- 1 mysql mysql   65 Oct 28 17:33 db.opt
-rw-rw---- 1 mysql mysql 8758 Oct 28 17:33 Strays.frm
```

Figure11: Files created within the Cats database

Now because sudo is required to access the MySQL program files, they must be copied to a less protected folder (one not requiring administrative privileges) that is accessible by the sftp process and moved to the “clean” virtual machine that the investigator has total control over. There are several ways to go about this, but one method is demonstrated, in part, as follows.

```
acl@zero:~$ sudo su
[sudo] password for acl:
root@zero:/home/acl# cd ../../var/lib/mysql
root@zero:/var/lib/mysql# ls `al
total 28704
drwx``````  7 mysql mysql       4096 Oct 10 20:47 .
drwxr`xr`x 35 root  root        4096 Oct 10 20:08 ..
drwxZZZZZZ&&2&mysql&mysql&&&&&4096&Oct&10&20:49&Cats&
`rw`r``r``  1 root  root           0 Oct 10 20:08 debian`5.5.flag
drwxZZZZZZ&&2&mysql&mysql&&&&&4096&Oct&10&20:40&Dogs&
ZrwZrwZZZZ&&1&mysql&mysql&18874368&Oct&10&20:52&ibdata1&
ZrwZrwZZZZ&&1&mysql&mysql&&5242880&Oct&10&20:52&ib_logfile0&
```

ZrwZrwZZZZ&&1&mysql&mysql&&5242880&Oct&10&20:08&ib_logfile1&
drwx````` 2 mysql root 4096 Oct 10 20:08 mysql

12
`rw`rw```` 1 root root 6 Oct 10 20:08 mysql_upgrade_info
drwx`````` 2 mysql mysql 4096 Oct 10 20:08 performance_schema
drwx`````` 2 mysql root 4096 Oct 10 20:08 test
root@zero:/var/lib# mkdir DataCopy
root@zero:/var/lib# cd DataCopy
root@zero:/var/lib/DataCopy# mkdir Cats
root@zero:/var/lib/DataCopy# mkdir Dogs
root@zero:/var/lib/DataCopy# cd ../mysql/Cats
root@zero:/var/lib/mysql/Cats# ls `al
total 36
drwx`````` 2 mysql mysql 4096 Oct 10 20:49 .
drwx`````` 7 mysql mysql 4096 Oct 10 20:47 ..
ZrwZrwZZZZ&1&mysql&mysql&8686&Oct&10&20:49&Boarders.frm&
ZrwZrwZZZZ&1&mysql&mysql&&&65&Oct&10&20:47&db.opt&
ZrwZrwZZZZ&1&mysql&mysql&8758&Oct&10&20:49&Strays.frm&
root@zero:/var/lib/mysql/Cats# cp Boarders.frm ../../DataCopy/Cats

The folders and files in need of copying have been bolded. The directory DataCopy was created along with subfolders Dogs and Cats to mimic the structure of MySQL's data files and keep the identically named files from the Dogs and Cats folders separate. While the above only shows copying one file from the Cats folder, the same should be done for the other two files in Cats, all three files in the Dogs folder, and the three files indicated in the mysql folder.

root@zero:/var/lib/mysql# cd ../../../home/acl
root@zero:/home/acl# ls `al
total 32
drwxr`xr`x 3 acl acl 4096 Oct 10 20:54 .
drwxr`xr`x 3 root root 4096 Oct 3 10:57 ..
`rw`````` 1 acl acl 99 Oct 3 11:08 .bash_history
`rw`r``r`` 1 acl acl 220 Oct 3 10:57 .bash_logout
`rw`r``r`` 1 acl acl 3486 Oct 3 10:57 .bashrc
drwx`````` 2 acl acl 4096 Oct 3 10:58 .cache
ZrwZZZZZZZ&1&acl&&acl&&2637&Oct&10&20:54&.mysql_history&
`rw`r``r`` 1 acl acl 675 Oct 3 10:57 .profile
root@zero:/home/acl# cp .mysql_history ../../var/lib/DataCopy
root@zero:/home/acl# cd ../../var/lib/DataCopy
root@zero:/var/lib/DataCopy# ls `al

```
total 28692
drwxr`xr`x  4 root root      4096 Oct 10 22:03 .


drwxr`xr`x 36 root root      4096 Oct 10 21:58 ..
drwxr`xr`x  2 root root      4096 Oct 10 21:59 Cats
drwxr`xr`x  2 root root      4096 Oct 10 22:00 Dogs
`rw`r`````  1 root root 18874368 Oct 10 22:01 ibdata1
`rw`r`````  1 root root  5242880 Oct 10 22:01 ib_logfile0
`rw`r`````  1 root root  5242880 Oct 10 22:01 ib_logfile1
`rw```````  1 root root      2637 Oct 10 22:03 .mysql_history
root@zero:/var/lib/DataCopy# chmod 644 ibdata1
```

The above also shows where to find the .mysql_history file, which is discussed later in this section. It should be noted that the chmod command (chmod 644) must be given on all ten files that sftp might be allowed access to the files. The administrator can then put the files onto a third machine and then onto the clean machine (useful if the source machine becomes the clean machine after a wipe) or directly onto the clean machine from the source.

```
acl@one:~$ sudo su
[sudo] password for acl:
root@one:/home/acl# cd ../../var/lib
root@one:/var/lib# mkdir DataCopy
root@one:/var/lib# chmod 777 DataCopy
root@one:/var/lib# cd DataCopy
root@one:/var/lib/DataCopy# mkdir Cats
root@one:/var/lib/DataCopy# mkdir Dogs
root@one:/var/lib/DataCopy# chmod 777 Cats
root@one:/var/lib/DataCopy# chmod 777 Dogs
```

This is setting up the folder structure to receive the source files. It should be noted that chmod 777 is executed on each directory to ensure sftp has access to them for the copy. Again, because administrator access is required to access the mysql folder, the files cannot be put directly into place but instead must be put onto the clean machine and then moved into place using the sudo command.

```
root@one:/var/lib/DataCopy#  cd  ../mysql
root@one:/var/lib/mysql#  ls  `al
total  28696


drwx``````   5  mysql  mysql       4096  Oct  10  20:09  .
drwxr`xr`x 36  root   root        4096  Oct  10  22:13  ..
`rw`r``r``   1  root   root           0  Oct  10  20:08  debian`5.5.flag
`rw`rw````   1  mysql  mysql  18874368  Oct  10  20:09  ibdata1
`rw`rw````   1  mysql  mysql   5242880  Oct  10  20:09  ib_logfile0
`rw`rw````   1  mysql  mysql   5242880  Oct  10  20:08  ib_logfile1
drwx``````   2  mysql  root        4096  Oct  10  20:09  mysql
`rw`rw````   1  root   root           6  Oct  10  20:09  mysql_upgrade_info
drwx``````   2  mysql  mysql       4096  Oct  10  20:09  performance_schema
drwx``````   2  mysql  root        4096  Oct  10  20:08  test
root@one:/var/lib/mysql#&rm&ibdata1&
root@one:/var/lib/mysql#&rm&ib_logfile0&
root@one:/var/lib/mysql#&rm&ib_logfile1&
```

The files being removed were created when MySQL was installed. However, they need to be replaced by the source files since the source files contained the data from the source machine. Now, the folders Cats and Dogs must be created in the mysql folder, and all files should be copied over into their appropriate places.

```
root@one:/var/lib/mysql# mkdir Cats
root@one:/var/lib/mysql# mkdir Dogs
root@one:/var/lib/mysql# cd ../DataCopy
root@one:/var/lib/DataCopy# cp ibdata1 ../mysql
root@one:/var/lib/DataCopy/Dogs# cd ../../mysql
root@one:/var/lib/mysql# chmod 644 ibdata1
root@one:/var/lib/mysql# chmod 744 Cats
root@one:/var/lib/mysql# cd Cats
root@one:/var/lib/mysql/Cats# chmod 644 db.opt
```

The above gives a brief example of creating the folders, copying one of the files, and then changing the permissions on the files and folders such that MySQL will be allowed access to them when it is run. All of the files should be copied and their permissions changed as demonstrated. In order for MySQL to recognize these changes and access

the files appropriately, the following commands must be given. These will effectively restart the MySQL service.

```
root@one:/var/lib/mysql/Dogs# stop mysql
mysql stop/waiting
root@one:/var/lib/mysql/Dogs# start mysql
mysql start/running, process 20202
```

The .mysql_history file was copied from the source machine and can be placed on the clean machine. It can either replace the clean machine's existing mysql_history file or be renamed and stored in addition to it. Looking at this file using the cat command in the command line allows the investigator to look back through the commands entered into MySQL to search for deletes, updates, and other commands that may not be easily determined from looking only at the final resulting data. The feature of recording commands to this history file can be disabled, and the file can be deleted.complete picture of the target's activities.

3.2.2 Results of the Copy Method

The files are now in place on the clean machine, and MySQL has been Restarted – a necessary step in order to sync everything together. The password protected data can now be viewed from the clean machine. Two screenshots appear in Figures 12 and 13. Figure 12 is from the source machine, while Figure 13 is from the clean machine. It is clear that they now contain the same information, indicating that the transfer of files was a success. While a somewhat tedious process, the password protecting the data was bypassed using administrative access and a fresh instance of MySQL with a known password.

```
root@zero:/var/lib/mysql# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 44
Server version: 5.5.34-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use Dogs
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from Strays;
+-----------+-----------+------+------------+-------------+---------------+----------------+
| Name      | Breed     | Sex  | Age        | WeightInLbs | DateCheckedIn | DateCheckedOut |
+-----------+-----------+------+------------+-------------+---------------+----------------+
| Sweetie   | Pitbull   | F    | Adult      |          42 | 2013-10-10    | 0000-00-00     |
| Lightning | Greyhound | M    | Adolescent |          85 | 2013-10-08    | 2013-10-09     |
+-----------+-----------+------+------------+-------------+---------------+----------------+
2 rows in set (0.00 sec)

mysql> select * from Boarders;
+--------+---------------------+------+------+------------+-------+
| Name   | Breed               | Sex  | Age  | Birthday   | Owner |
+--------+---------------------+------+------+------------+-------+
| Apollo | Golden Retriver Mix | M    |   12 | 2001-02-12 | Sally |
| Samson | German Shepherd Dog | M    |   13 | 0000-00-00 | Sally |
| Chloe  | Pomeranian          | F    |    3 | 2010-05-14 | Jack  |
+--------+---------------------+------+------+------------+-------+
3 rows in set (0.00 sec)

mysql> use Cats
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from Strays;
+---------+------+--------+-------------+---------------+----------------+
| Name    | Sex  | Age    | WeightInLbs | DateCheckedIn | DateCheckedOut |
+---------+------+--------+-------------+---------------+----------------+
| JaneDoe | F    | Kitten |          12 | 2013-10-05    | 0000-00-00     |
+---------+------+--------+-------------+---------------+----------------+
1 row in set (0.00 sec)

mysql> select * from Boarders;
+----------+------+------+------------+--------+
| Name     | Sex  | Age  | Birthday   | Owner  |
+----------+------+------+------------+--------+
| Speckles | F    |    6 | 2007-01-17 | Junior |
| DC       | M    |   10 | 2003-04-21 | Missy  |
+----------+------+------+------------+--------+
2 rows in set (0.00 sec)
```

Figure 12: MySQL output showing the data stored on the source machine

```
root@zero:/var/lib/mysql# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 44
Server version: 5.5.34-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use Dogs
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from Strays;
+-----------+-----------+------+------------+-------------+---------------+----------------+
| Name      | Breed     | Sex  | Age        | WeightInLbs | DateCheckedIn | DateCheckedOut |
+-----------+-----------+------+------------+-------------+---------------+----------------+
| Sweetie   | Pitbull   | F    | Adult      |          42 | 2013-10-10    | 0000-00-00     |
| Lightning | Greyhound | M    | Adolescent |          85 | 2013-10-08    | 2013-10-09     |
+-----------+-----------+------+------------+-------------+---------------+----------------+
2 rows in set (0.00 sec)

mysql> select * from Boarders;
+--------+---------------------+------+------+------------+-------+
| Name   | Breed               | Sex  | Age  | Birthday   | Owner |
+--------+---------------------+------+------+------------+-------+
| Apollo | Golden Retriver Mix | M    |   12 | 2001-02-12 | Sally |
| Samson | German Shepherd Dog | M    |   13 | 0000-00-00 | Sally |
| Chloe  | Pomeranian          | F    |    3 | 2010-05-14 | Jack  |
+--------+---------------------+------+------+------------+-------+
3 rows in set (0.00 sec)

mysql> use Cats
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from Strays;
+---------+------+--------+-------------+---------------+----------------+
| Name    | Sex  | Age    | WeightInLbs | DateCheckedIn | DateCheckedOut |
+---------+------+--------+-------------+---------------+----------------+
| JaneDoe | F    | Kitten |          12 | 2013-10-05    | 0000-00-00     |
+---------+------+--------+-------------+---------------+----------------+
1 row in set (0.00 sec)

mysql> select * from Boarders;
+----------+------+------+------------+--------+
| Name     | Sex  | Age  | Birthday   | Owner  |
+----------+------+------+------------+--------+
| Speckles | F    |    6 | 2007-01-17 | Junior |
| DC       | M    |   10 | 2003-04-21 | Missy  |
+----------+------+------+------------+--------+
2 rows in set (0.00 sec)
```

Figure 13: MySQL output showing the data that has been transferred to the Clean machine

3.3 The Plainview Method

3.3.1 Procedure of the Plainview Method

The Plainview Method involves looking at the aforementioned files directly (by using the cat command, for example). While most of the file are unreadable to a Human without MySQL to decipher them, they can still yield useful information about the structure and data associated with the tables in the databases. The results of this method are more difficult to interpret and are likely far less complete than those produced by the Copy Method. However, if another instance of MySQL is not available due to limited resources or any other factor, this may be the only option together any information at all. Though it does not yield as much information as an investigator would probably desire, it is better than being left without even a glimpse into the table structure and data. Because ibdata1 is known to contain the data entries for the tables and the .frm files contain the table structure, they can be used to gain a limited look at what Was stored in the databases. If the cat command is given on a .frm file, the result will be something similar to Figure 14 below.

The methods explored in this thesis provide powerful tools to the Investigator needing to gain access to a user's MySQL database without the need for the user's co-operation. This is useful in cases of emergency where the user is unavailable or where the user is under investigation (and thusly unaware of or uncooperative with the investigation) and the data stored in MySQL is believed to be relevant.

```
root@one:/var/lib/mysql/Dogs# cat Strays.frm
?

0K     d? // ?? InnoDBV?\K>P)                                          NameBreedSexAge
                                                                              WeightInLbs  DateCheckedIn
DateCheckedOut,?-?


                B

Fp?

Ip?Name?Breed?Sex?Age?WeightInLbs?DateCheckedIn?DateCheckedOut?root@one:/var/lib/mysql/Dogs#
```

Figure 14: The output of Strays.frm from the Dogs database

It is seen that the last line corresponds to the column names for the Strays table in The Dogs database with a '?' serving as a delimiter. This provides an idea of what

Type of information is stored and gives at least a clue to the data type. Moving to look at the ibdata1 file, the cat command yields results a bit more difficult to interpret. It may be helpful to find a way to eliminate all of the question marks. One method is to copy and paste the output into Microsoft Word and use the replace feature to search for '?' and replace with nothing. The information of interest is found towards the end of the file.

3.3.2 Results of the Plainview Method

Figure 15 shows the output that relates to the table Strays in database Dogs.

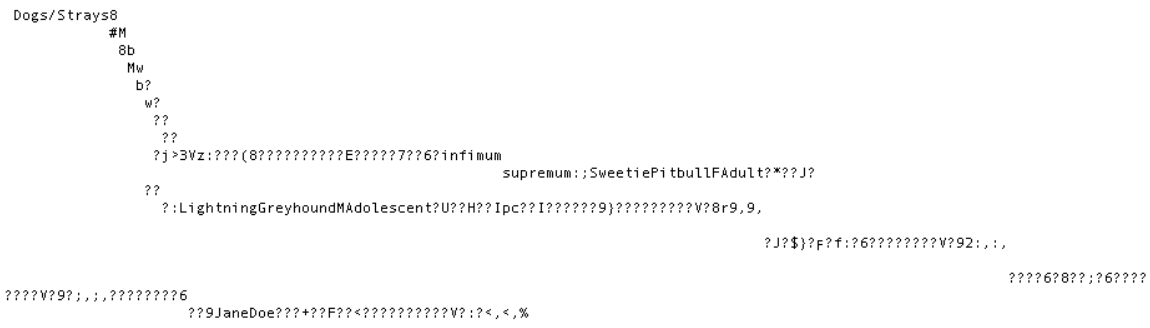


Figure15: Part of the output from the ibdata1 file that pertains to the Strays Table in database Dogs

```
mysql> select * from Strays;
+-----------+-----------+------+------------+------------+--------------+---------------+
| Name      | Breed     | Sex  | Age        | WeightInLbs | DateCheckedIn | DateCheckedOut |
+-----------+-----------+------+------------+------------+--------------+---------------+
| Sweetie   | Pitbull   | F    | Adult      |          42 | 2013-10-10    | 0000-00-00     |
| Lightning | Greyhound | M    | Adolescent |          85 | 2013-10-08    | 2013-10-09     |
+-----------+-----------+------+------------+------------+--------------+---------------+
2 rows in set (0.00 sec)
```

Figure 16: The data entries for the Strays table of the Dogs database after the UPDATE command was executed

CHAPTER 4: CONCLUSION

4.1 Summary

There are times when an investigator may need access to the data stored in MySQL but may not have the individual user's password to the program. Whether The user is simply unavailable or perhaps under investigation, the investigator may need to use the higher permission of the system administrator to bypass the user's password. This thesis explains two methods that help achieve this goal. The Copy Method copies the system files to a new instance of MySQL and totally defeats the Password protection. The Plainview Method looks at the system files through the command line and provides a narrow view that reveals some of the data.

4.2 Contributions

The methods explore provide powerful tools to the Investigator needing to gain access to a user's MySQL database without the need for the user's cooperation. This is useful in cases of emergency where the user is unavailable or where the user is under investigation (and thusly unaware of the uncooperative with the investigation) and the data stored in MySQL is believed to be relevant. It should be noted, however, that this could also be considered a vulnerability in MySQL's security as someone with administrative access could use the above methods for nefarious means without the consent of the system's owner.

4.3 Future Work

An exploration of exactly what permission levels are needed on the folders and files to be copied and accessed in the Copy Method would enhance that approach so that higher permissions than are necessary would not be set. It is generally accepted that files, like users, should be given only the level of access that is truly needed. While it at first appeared that copying the files higher into the folder structure (to a less protected folder) before transferring them to the

clean machine was a necessary step so that sftp could be used, further investigation has suggested the possibility of alternative, less tedious methods. Further study on the patterns formed by the Date data types in the ibdata1 file would prove useful if it revealed a reliable way to decipher the encoding methods. Along the same lines, testing with the rest of the data types supported by MySQL database to study their encodings would provide useful information that would also enhance the effectiveness of the Plainview Method.

CHAPTER 5: REFERENCES

[1] "MySQL." *Wikipedia.* 13 Nov. 2013. http://en.wikipedia.org/wiki/Mysql

[3] "MySQL Glossary." *MySQL.* 13 Nov. 2013.
http://dev.mysql.com/doc/refman/5.5/en/glossary.html

[4] "Introduction to InnoDB 1.1." *MySQL.* 14 Nov. 2013.
http://dev.mysql.com/doc/refman/5.5/en/innodb`introduction.html

[5] "mysql Logging." *MySQL.*-14 Nov. 2013.
http://dev.mysql.com/doc/refman/5.0/en/mysql`logging.html

[7] "CREATE DATABASE Syntax." *MySQL.* 14 Nov. 2013.
http://dev.mysql.com/doc/refman/5.0/en/create`database.html

[8] "mysql Options." *MySQL.* 14 Nov. 2013.
http://dev.mysql.com/doc/refman/5.0/en/mysql`command`options.html

[9] "ASCII Table and Description." *Ascii-Table*. 14 Nov. 2013.
http://www.asciitable.com

Abstract for SQLite

Forensic analysis and evidence collection for web browser activity is a recurring problem in digital investigation. It is not unusual for a suspect to cover his traces. Accordingly, the recovery of previously deleted data such as web cookies and browser history are important. Fortunately, many browsers and thousands of apps used the same database system to store their data: SQLite. Reason enough to take a closer look at this product. In this article, follow the question of how deleted content can be made visible again in an SQLite-database. For this purpose, the technical background of the problem will be examined first. Techniques are presented with which it is possible to carve and recover deleted data records from a database on a binary level. A novel software solution called FQLite is presented that implements the proposed algorithms. The search quality, as well as the performance of the program, is tested using the standard forensic corpus. The results of a performance study are discussed, as well.

INDEX

INDEX FOR FIGURES

CHAPTER 1: INTRODUCTION

SQLite is a widely used, lightweight database system. The self-contained SQL database engine runs on any smartphone as well as most computers. It implements a simple, compact, and file-based database management system (DBMS), which can be accessed via a program library. No other server software is required, and the database can be easily integrated into any application. It is prevalent and used in different software products. For example, Safari, Mozilla Firefox, and Google Chrome use SQLite version 3 databases to manage user data such as history, cookies, and downloaded files.According to an estimate of the domain sqlite.org, there are more than one trillion database installations in active use today1.
In digital forensics, the detailed analysis of such databases is of great importance. Whether a suspect deleted call-lists, chat protocol, browser history, or configuration settings: Today, this data is almost always stored in an SQLite DBMS. Many free programs support reading the database format .Unfortunately, these tools only show active records.Deleted information cannot be viewed or restored with most of these products. The recovery must then often be made manually.
Alternatively, someone can use commercial and costly forensic solutions.
This article will first examine how a recovery of deleted data records on a low-level is carried out in general. In this context, two algorithms for the recovery of deleted data sets within SQLite databases are presented in detail. Beyond that, a novel free, open-source tool will be presented to recover slack spaces within the database. The search quality and the performance of the study will be discussed.

CHAPTER 2: TECHNICAL BACKGROUND

Before turn to recover deleted records, let us first discuss some basics of building and managing SQLite databases. When considering how to reclaim data that have been removed or overwritten, first have to talk about the way SQLite organizes its dataset. In relational databases, all data is managed in tables. A table, in turn, is composed of several pages. Similar to blocks in a file system, a page is used to store data. It can, therefore, contain multiple records. Each page has a variable size - the default size is 4096 bytes. The maximum size for an SQLite page is 65536 bytes. The pages are numbered. The first page in the database has the number 1. It is reserved exclusively for the database header. The 100-byte header record is usually followed by a

description of the database schema. To represent a table together with their pages, the system uses a balance dtree data structure (B-tree) under the hood. This is a straightforward and highly efficient way to store data (Comer, 1979). The layout of B-tree pages in SQLite is similar to the layouts used in many other relational DBMS (Wagner et al, 2015).

As usual with trees as a data structure, a distinction is made between inner nodes and leaf nodes. The special feature is that only the leaf nodes can contain data records. The inner nodes are therefore only used for linking (see Figure 1). There is a separate tree for each table. The data records are accessed exclusively via the root node, traversing to each leaf. An active page of a database must be assigned to a node. In SQLite, a total of 4 different page types are used.

The one-byte flag at offset 0 of each page indicates the type. A value of 2 (0x02) means the page is an interior index b-tree page, whereas a value of 10 (0x0a) means the page is a leaf index b-tree page. Can ignore these values in our case since are interested in data pages only. A value of 5 (0x05) means the page is an interior table b-tree page whereas the value of 13

(0x0d) means the page is a leaf table b-tree page. The link between node and page is realized via the page number. If a page is removed, then, similarly to the deletion of a file from the filesystem, the affected page is only “marked” as deleted. More precisely, the leaf node assigned with this page must only be cut from the tree (page number is removed). If cannot find a page number in the table tree anymore, it is considered as deleted and can be filled with new data. All pages released in this way are saved in the free-page list (see next section). By inserting new content into the database, the released areas are actually overwritten. Thus, slack space within the database can emerge. These areas are of particular forensic interest.

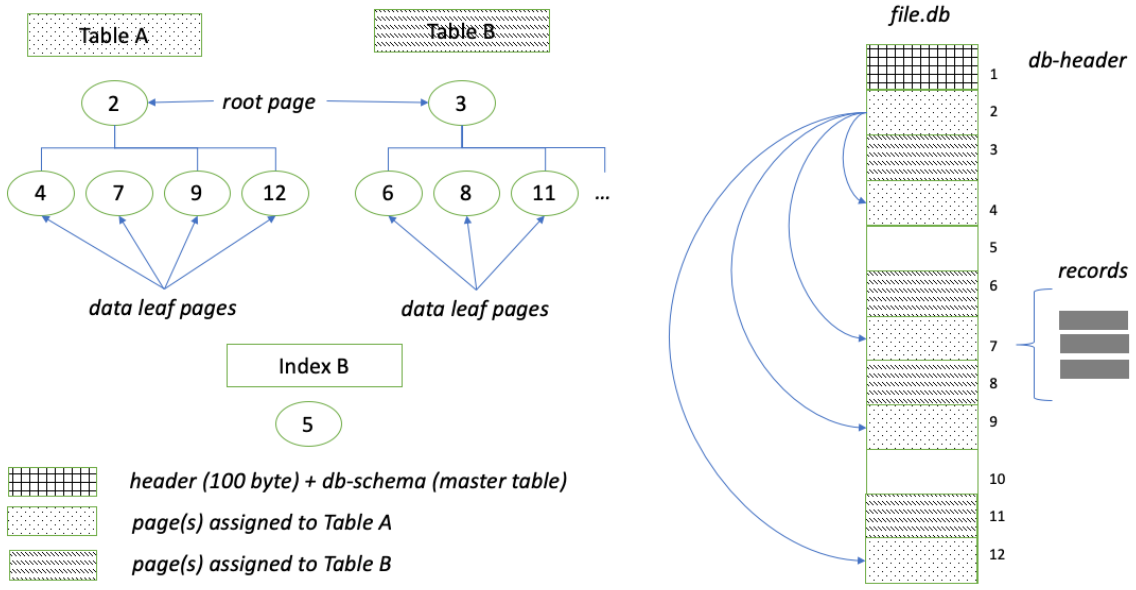


Figure 1. Structure of a SQLite3 database**.**

For an exact forensic examination, further need to take a closer look at the header information of the database. The data base is managed in a single file on disk only. All data records are stored in a unique binary format. Thus, have to identify and decode the fields that are essential for our investigation. The head of each SQLite database has a well-defined structure. Each byte has a meaning. In Table 1, can see the structure and function of each header-field within the first 40 bytes of the database file. This information is taken from the official web page of the SQLite project (sqlite.org, 2020). With this in mind, it is straightforward to identify an SQLite database. Only need to compare the beginning of the file with the header stringspecified in Table 1. However, the table contains even more interesting information that can help us to find and recover deletedrecords. Beginning at offset 16, can find a two-byte value that represents the page size of the database. It is a big-endianinteger and must be a power of two. Unused or removed pages in the database file are normally stored on a data structurecalled free-page list. At offset 32, can find a 4-byte big-endian integer, which indicates the beginning of this list. It containsthe offset of the first page of the list.

Table 1. Header fields of the on-disk database file format (sqlite.org, 2020).

offset	length	field content
0	16	The header string: "SQLite format 3\000"
16	2	The database page size in bytes. Must be a power of two between
18	1	File format write version. 1 for legacy; 2 for WAL.
19	1	File format read version. 1 for legacy; 2 for WAL.
20	1	Bytes of unused "reserved" space at the end of each page.
21	1	Maximum embedded payload fraction. Must be 64.
22	1	Minimum embedded payload fraction. Must be 32.
23	1	Leaf payload fraction. Must be 32.
24	4	File change counter.
28	4	Size of the database file in pages.
32	4	Page number of the first freelist trunk page.
36	4	Total number of freelist pages.

If the value is zero, the list is empty. The 4-byte big-endian integer at offset 36 represents the total number of entries on the free-page list. Together with the page size, now have everything need for an analysis of the slack spaces.

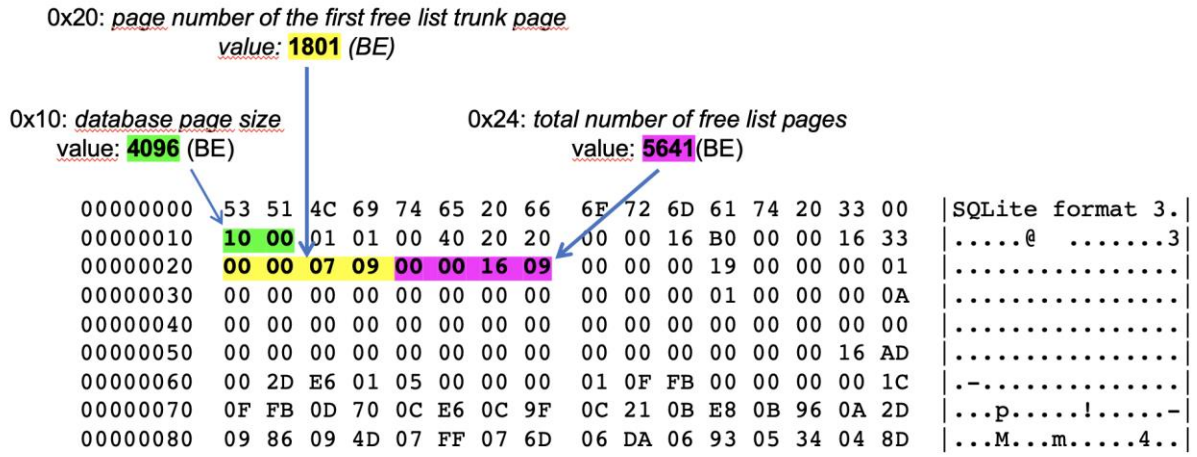


Figure 2. Example of the database header with free list information.

CHAPTER 3: METHODS AND ALGORITHM

3.1. Examine the free-page list

Let us first look at the case where one or more pages have been completely deleted from the database. To determine whetherthere are any deleted pages in the database, have to search for the free page list (Haldar, 2015), (sqlite.org, 2020). From a technical point of view, all the database's free memory pages are linked together in a list. As already mentioned, some of thesepages may contain old, actually deleted data records. Starting from the beginning of the database file, first have to find thefree page list's jump address. If the start address for the list has found, as shown in

Figure 2, the entry address of the first freememory page can then easily be determined using the following equation:

Jump to address = ((4 Byte BE in offset 32)-1) * page size (1)

In the above example, the list, therefore, starts at the beginning:

jump to hex = (0x0709 - 1) * 0x1000 = 0x708000
jump to decimal = (1801 - 1) * 4096 = 7372899

The offset value calculated in this way leads directly to the first element in the free list. From here, you can iterate over the
Sub sequent entries.

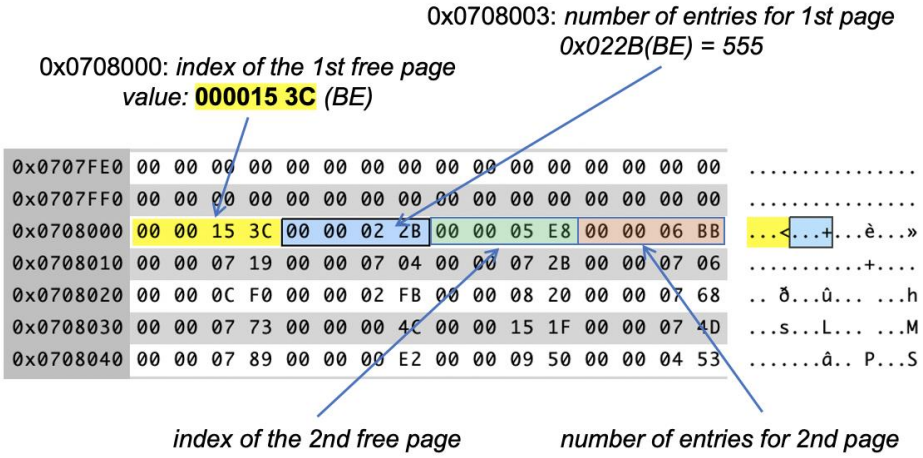


Figure 3. Start of the free page-list

As before, this value must be multiplied by the page size to determine the start address of a particular page. Now can move the file pointer to the first page. Each page begins with a header of between 8 and 12 bytes (see Figure 4). The b-tree page header is 8 bytes in size for leaf pages and 12 bytes for interior pages. All multi byte values in the page header are big-endian. The actual cell pointer array with the offset values of the data records follows the header directly. The offset value is a two-byte integer value and is given relative to the free page's beginning. Can again iterate through the field witha loop. The number of cells can be found at offset 3 - 4. Active data records are stored in the cell content region.

It should be noted that this area behaves similarly to a stack within the memory allocation. It grows "downwards" in the direction of lower addresses. Thus, a record added last to the page resides at a lower page address than the older entries.

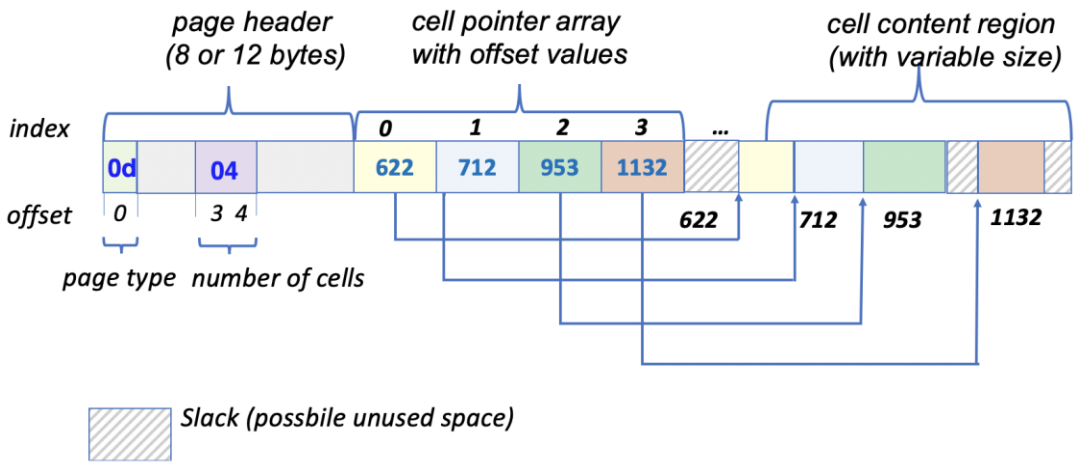


Figure 4. Structure of a data page in SQLite

When dealing with a particular record, must analyze the structure of a record more closely. Figure 5 shows an example of a data record from a free page. In this case, it is a browser cache entry from the table *moz_bookmarks*.

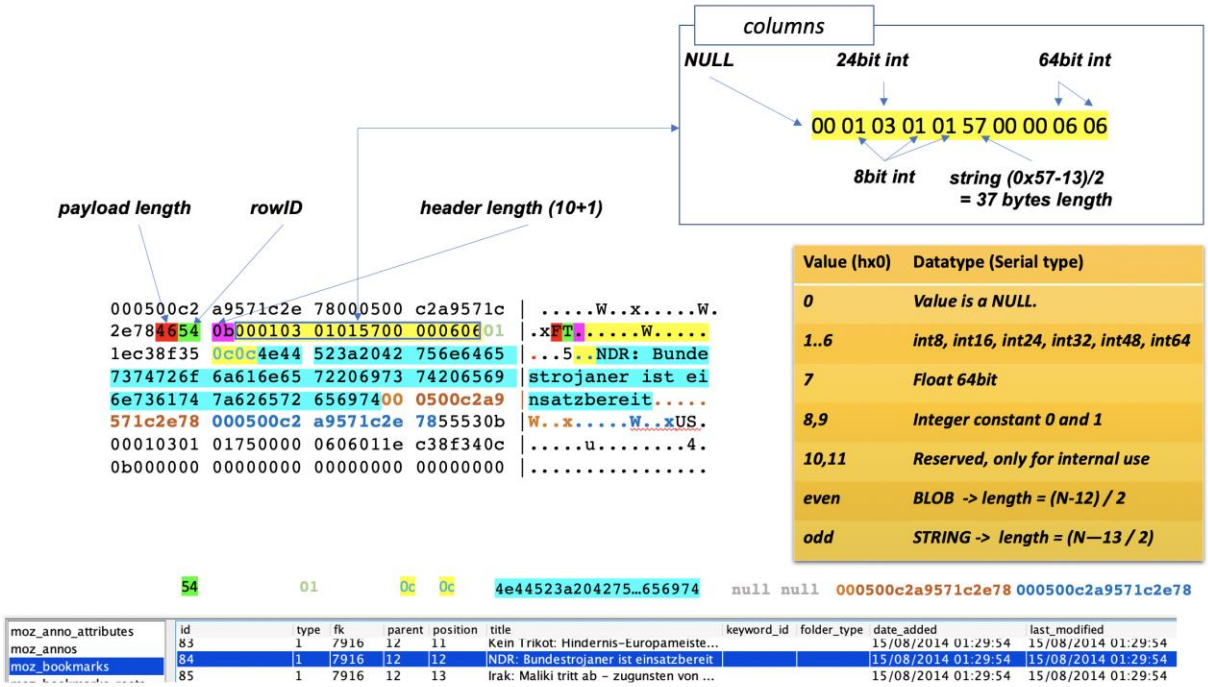


Figure 5. Structure of a data record in SQLite including serial types

The format of a cell record depends on the page the cell is located. In the above example, a leaf cell (header 0x0d) is shown.
The record is split into a header and a body. The header begins with a single var int-value. SQLite makes intensive use of such base 128 encoded numbers on binary level (sqlite.org, 2020). A var int is a way of compressing down integers into a smaller space than is usually needed. By default, computers use fixed-length integers. When storing small numbers, much memory is wasted because of this behaviour. This is different for var int values. The lower 7 bits of each byte are used to encode the actual number. The most significant bit is used to indicate, whether or not more bytes are coming. In our example, the header cell has a binary value [0]0001101. The header length is the size of the header in bytes including the size var int itself. Accordingly, have 12 (13-1) header bytes to follow. If the leftmost byte were set to one, would have to consider one or more bytes to determine the actual length. Hence, the magnitude is the total number of bytes in the header. There are two more interesting values in the header. The first one is again a var int and holds the total number of bytes of payload. In our example, the payload has a length of 0x46 (70) bytes, the initial portion of the payload does not spill to overflow pages. This value is then followed by a field that is known as rowID. However, these are of secondary importance for the understanding of the following explanations.
The header size var int and serial type var ints will usually consist of a single byte (Table 2).
The serial type var ints for large text and BLOBs might extend to two or more-byte var ints. However, that is not the rule. The actual column values immediately follow the header. Columns with values 0,8,9,10 and 11 are NULL. The string column size can be calculated using the following formula: length = (N – 13) / 2. In our example, the string length is 37 because of (0x57-13)/2. The string terminator char \0 is not stored. It should be noted that SQLite does not have an explicit data type for representing time and date information. Very often, timestamps are represented in standard UNIX time, i.e., with a 64bit long number.
To summarize, an algorithm can be derived from what has been said so far (see Figure 6). It consists of two main loops -the first one iterates the free-page list. Go through the cell pointer array for the respective page to recover data cells within the inner-loop. Accordingly, the worst-case time complexity is O (n*m), whereas n represents the number of free pages, and m represents the maximum number of cells within a page. Since the size of a page is limited, the number of cells is also limited from above. The algorithm has linear time complexity. At this point, it should be mentioned that the SQLite Auto-VACUUM function must be disabled. Otherwise, the database would be cleaned up regularly, and the slack areas would be released to save space.

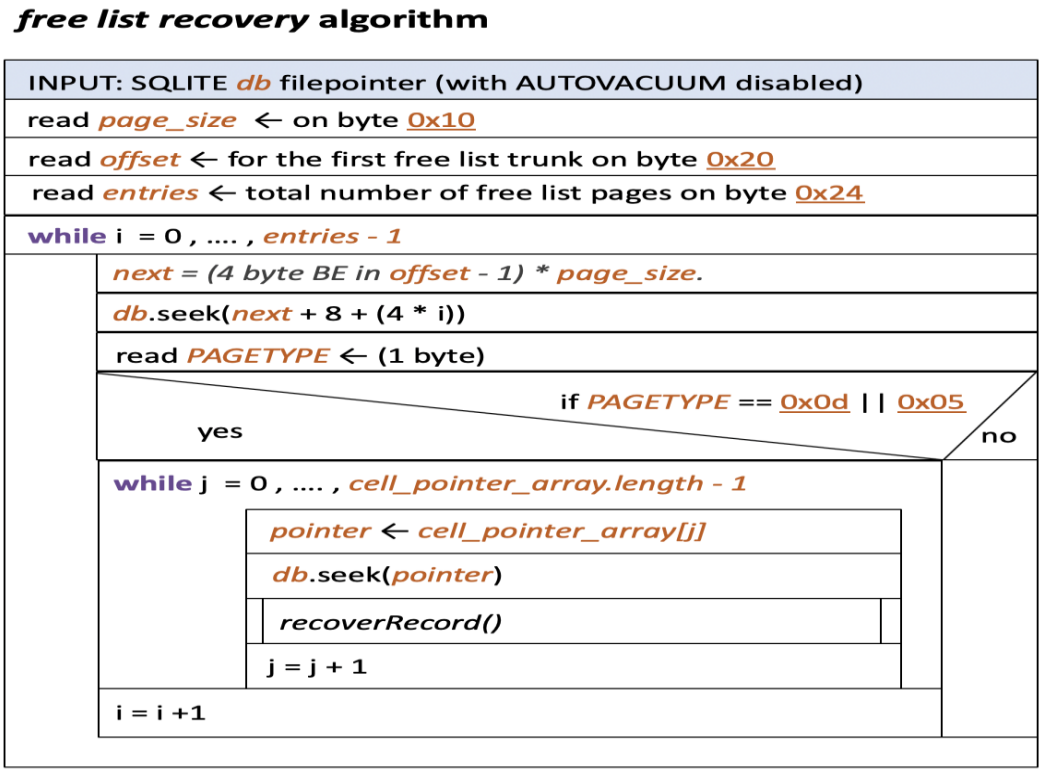


Figure 6. The free page list recovery algorithm**.**

3.2. Carving for more artefacts

So far, I have tried to grab the low hanging fruits. However, what happens if no free-page list exists? Other places in the database also could contain slack space. Whenever individual records are deleted from a table, they can occur. Hence, I have to carve for the hidden records inside all data pages. Unfortunately, this results in another problem. Before can recover those data, I first have to find the deleted records. Thus, I need to examine the slack areas more closely.
However, from the beginning: Dropping a table or deleting some rows from the database naturally creates gaps, which are replaced when new records are inserted. When a record is deleted, parts of its header are overwritten. More precisely, the first 4 bytes are filled with new information The first two bytes store the offset of the next free block within the current page. The next 2 bytes provide information about the length of the free block and thus indirectly the size of the formerly deleted data. Similar to a free page list, free blocks within a page are managed as a simple linked list. The offset of the first list element is on offset 3 and 4 in the page header. From here, I can follow the link to the next data fragment. If the value stored in the next-free block field is 0, then there are no more free blocks within this page. Unfortunately, the information on free blocks is

not always reliable. Sometimes a table is deleted with a SQL-DROP statement. In this case, the specification for free blocks is overwritten too. Hence, no cell pointer tells us where to find all the deleted records.
As pointed out, a record is overwritten only partially. Since all length specifications are stored as var int values without a fixed length, sometimes more and sometimes fewer header fields are overwritten. When carving for hidden entries, I have to consider this. Not in every case, a complete recovery is possible, as I can see in Figure 7. If only the payload length byte sand row id bytes are missing a complete recovery is always possible. A missing length specification for the header length byte is also not a big problem, since I can derive this information indirectly from the remaining header bytes. However, it becomes difficult if the length specification for the first column has been overwritten too. Sometimes it is possible to deduce the length of the first column from neighboring records. Most tables in a typical SQLite database schema are row id tables, (sqlite.org, 2020). Moreover, they start with a var int key in the first column. So, I could make an educated guess in this case. If more bytes of the header get replaced, there is no chance for an automatic recovery, due to the vast of possible length values.

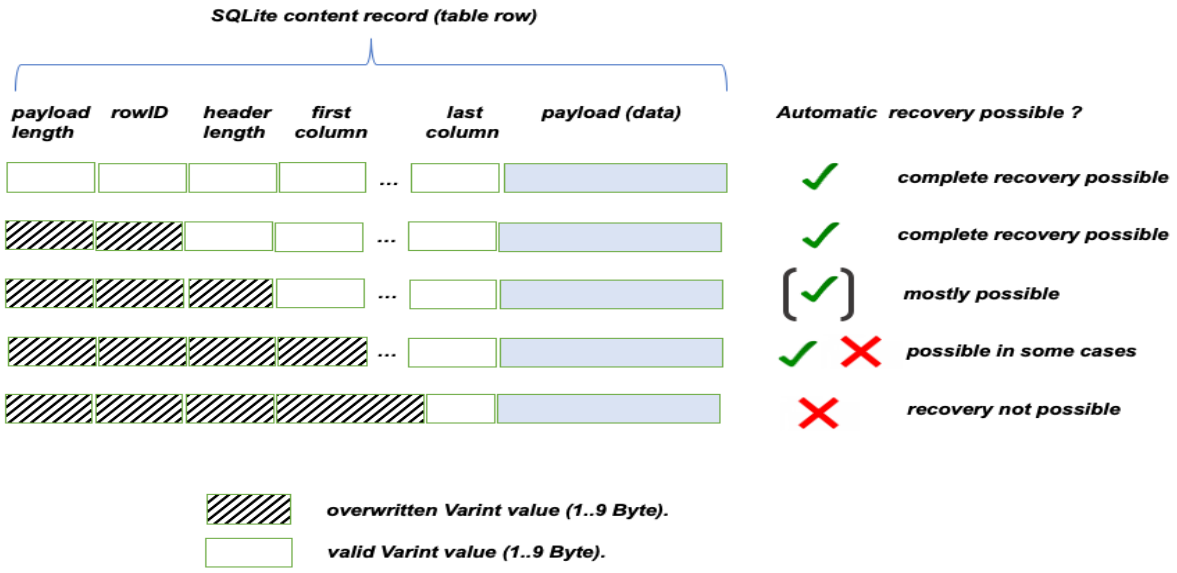


Figure 7. Deleted records and their structure.

Besides the free block areas, there is also the so-called unallocated space (Figure 8). It follows after the record header directly.
Usually, this space is empty and contains only zeros by default. However, sometimes, fragments of deleted records can be found here. If, for example, the last inserted record is deleted within a page, the cell pointer is overwritten. In this case, the offset of the active content area is moved towards the next regular record. SQLite also reorganizes the contents of pages in the background to avoid free blocks from time to time. This defragmentation process also causes content to be moved to the unallocated area.

Accordingly, it is much more complex to keep track of which parts of a page are allocated or free at any given time.
To summarize, all memory areas that do not contain active records and do not belong to the page header or the cell pointer region can contain potentially deleted artefacts. Nevertheless, where to begin? Fortunately, each record begins with a listing of serial types. It is situated in the record header (as described in the last section). If I first analyze the database schema, may be able to derive some pattern from it. I will use this information to search for artefacts. It works as a signature or fingerprint.

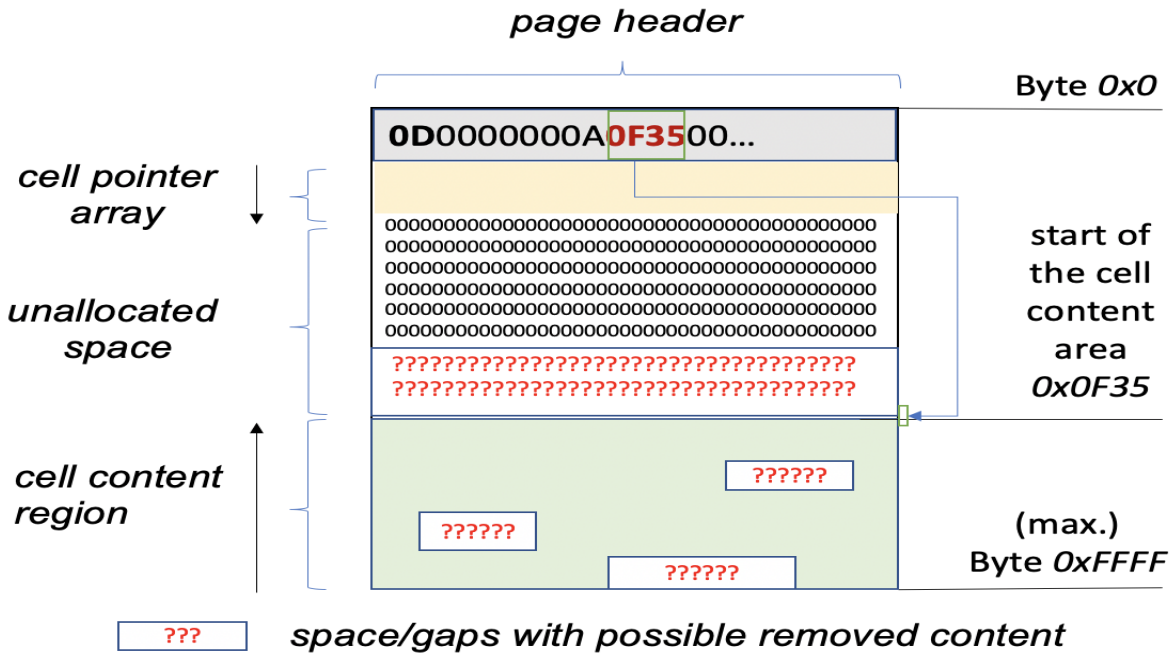


Figure 8. Possible slack for d**eleted artefacts.**

This procedure is based on the assumption that at least most of the header information is still intact. In this way, I can perform a search in the uncharted regions of the data page. In the following, I will discuss a practical method that is not guaranteed to be perfect but sufficient to deliver good detection results in most cases. Our heuristic makes an educated guess about the header of a record. Therefore, I first analyze the database schema to derive a search pattern for each table row.

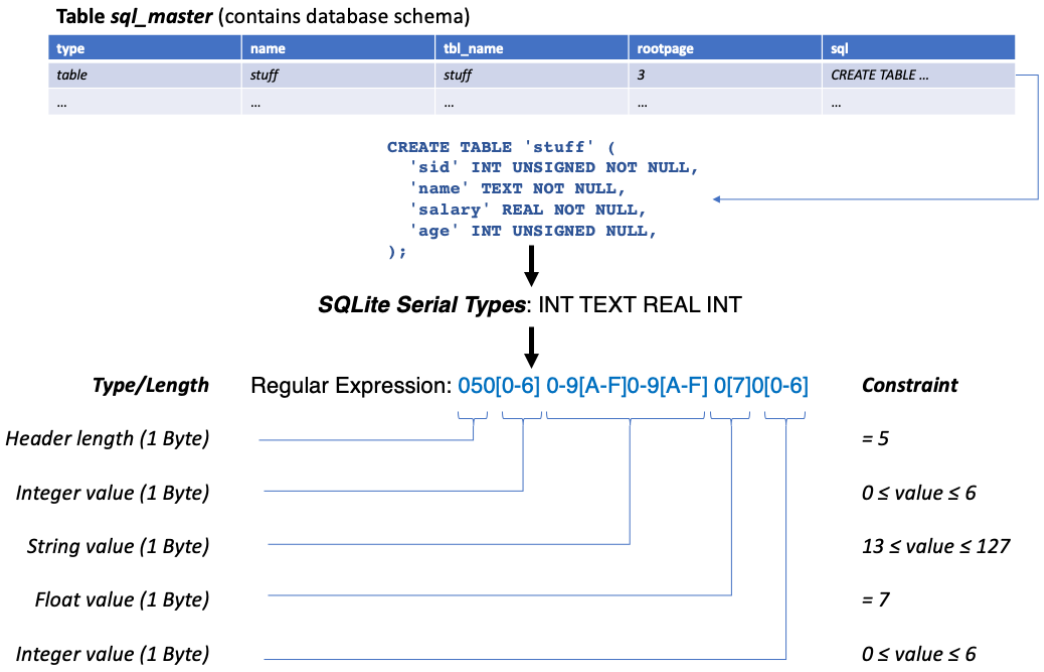


Figure 9. Deriving pattern from a table definition.

Figure 9 shows an example of a search pattern derived in this way. I will start with the header byte. It is always at the beginning of the header. The value corresponds to the number of columns bytes in the table increased by one for the length counter itself. Thus, the header specification begins with the value 0x5 (decimal '05') since the table has four columns in the above example. Integer valued columns are always mapped with a value between 0x0 (no value) or 0x6 (INT64).

In this case, the pattern would have to be adjusted accordingly. If this information is combined, a search pattern can be derived from it.
I repeat the pattern building step for each table definition that could be found. The actual search process is straightforward. First, the data page to be carved is converted into a hex-string. Then a pattern matching is performed, using the regular expressions created previously. Again, I use the pattern to identify the probable (header) start of a previously deleted record. In our search, concentrate only on the slack areas. This implies that I have previously identified the regular records and the header. The FQLite-tool presented in the next section uses a Boyer-Moore algorithm for the actual search. Rather than first translating a regular expression into a deterministic finite automaton, the implementation matches the fly's regular expression. Naturally, a search based on a pattern carried out in this way can also lead to false-positive results.
Therefore, each match is subsequently checked for plausibility. The test is passed if: (a) the (probably) complete header information could be reconstructed, (b) all constraints are fulfilled and (c) the computed payload length derived from the header information

does fit into the actual cap (no overlapping with a regular record nor payload exceeds page size). Only if all three conditions are met, the record is restored. Since regular expressions do not support range checking for numbers, I need to define and check them with additional constraints. The constraints can be directly derived from the binary information (see Figure 7). In this way, the entire page is checked until either no further hits are found, or there are no slack areas. A flow chart of the recovery process discussed can be found in Figure 10.

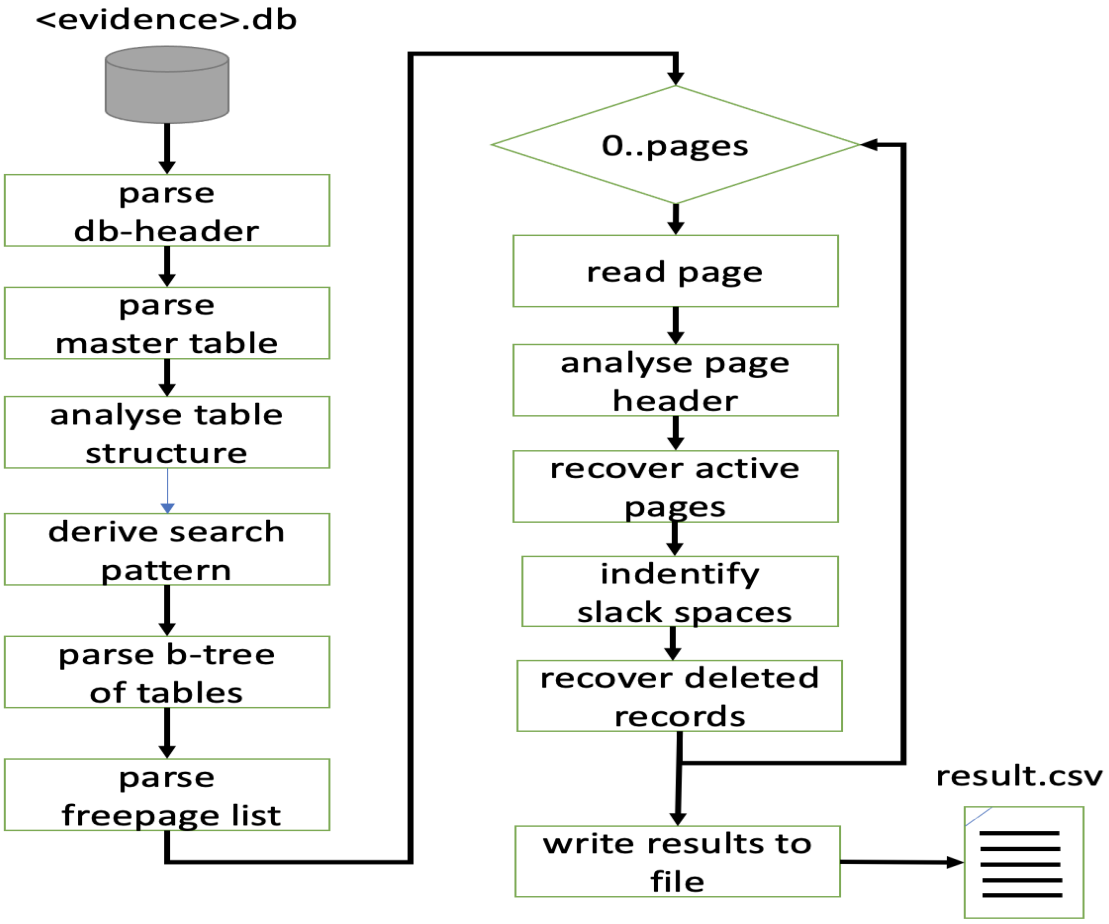


Figure 10. Flowchart of recovery process.

CHAPTER 4: RELATED WORK

Due to its popularity, various projects have started to gather information from SQLite databases over the last decade, in an overview of the current state of research in the field of database forensics. A good general introduction on the forensic investigation in SQLite is a book published by Paul Sanderson (Sanderson, 2018). However, it does not focus on the carving of deleted records. A somewhat older publication by Jeon et al. from 2012 already mentions important key issues when recovering data in SQLite databases. The authors use the schema table to collect information about the database structure and the data leaf pages. However, the authors overlook that by deleting tables, this information is usually lost as well. The recovery of data from the unallocated area is not discussed. Unfortunately, no concrete algorithm is presented. Only a tool called
SQLite Recover is mentioned, but it is not publicly available. In a publication from 2012, Bagley et al. discuss the recovery ofdata from the Firefox database (Bagley, 2012). The paper proposes an algorithm to recover deleted SQLite entries based on known internal record structures. The search for URL entries in the unallocated space is done using some simple kind of pattern matching. However, in contrast to FQLite (see next section), only strings starting with the protocol "http(s)://" are searched. Are
-construction of data records other than URLs is therefore impossible.
In Aouad et al. 2013, the authors propose a method to extract deleted message artefacts on Android devices. Unfortunately, the described algorithm is only tailored to specific message types. The authors do not provide a generic solution for arbitrary SQLite databases. Neither the program nor the source code is publicly available. The recovery of index records from SQLite databases is presented in Ramisch & Rieger, 2015. The idea behind their approach is to make use of entries from index-tablesto restore deleted records. Removed records sometimes can be unrecoverable from the regular SQLite table, but can still reside in the b-tree index leaf page. FQLite does not evaluate this information at the moment. The quite promising approach is unfortunately not complemented by a corresponding freely available program. A script used to recover free list pages from a database is presented in (DeGrazia, 2013). The program is written in Python and offers a simple graphical frontend. It is part of the evaluation section. It is limited to the recovery of free pages. Thus, it is comparable to the first algorithm presented in this document.

Further carving functionalities are not offered. An example of a viral script is Undark von Paul Daniels (Daniels, 2020).he program written in C++ can only be operated from the command prompt. The database is searched from back-to-backpage by page. More detailed information about the database schema will not be evaluated. Found data records cannot be uniquely assigned to a table. In (Skulkin & Mikhaylov, 2018) the authors describe a simple method to recover data from a damaged or corrupted database. Without special tool support, they are able to save most of the data into another SQLite database. However, a search for deleted artifacts does not take place. Another interesting contribution focuses on the analysis

WAL files (Yao, 2016). This special file type is used to insert new records into an SQLite database. The authors only addresshe problem of extracting content from a WAL-file. Therefore, the range of applications is very limited. FQLite currently offers no support for this special file format. Thus, the contribution is a useful add-on to our work. A comparable approach can beound in (Shun, 2014). In (Pawlaszczyk, 2017) a program called MOZRT SQLite recovery tool is introduced. It was special designed to recover data from the free-page list for browser caches of the MOZILLA browser.

CHAPTER 5: FQLITE RECOVERY TOOL

The algorithm and processing steps discussed in the last sections were implemented to check their quality and demonstrate practical use. To make the deleted information visible again, use an App which is called FQLite. This program allows someone to recover deleted entries from an SQLite3 database. It, therefore, examines the database for entries marked as deleted. Those entries can be recovered and displayed (see Figure 11). The tool does not only recover and browse the content of free list pages and deleted records. It also helps to create CSV-format export for the data found. It is 100% written with the
Java standard class library. It runs out of the box on any platform with a Java compiler version 1.8 or higher. To ensure maximum transparency and easy verifiability of our results, the application is distributed as open-source. Accordingly, the source code is published under Mozilla Public License Version 2 and the GNU General Public License Version 3 or later. So, anybody can modify or redistribute it under the conditions of these licenses. Have opted for this way to ensure maximum transparency and traceability.

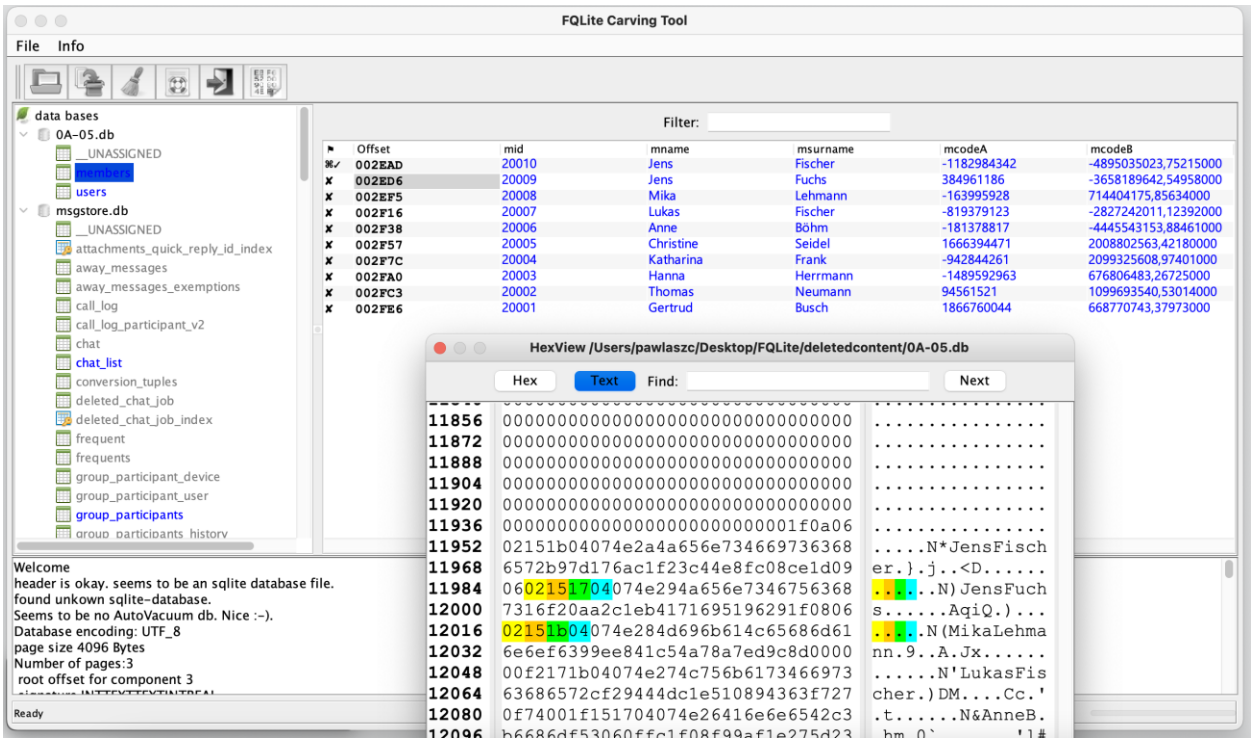


Figure 11. The FQLite Data Retrieval Tool.

The program can operate in two different modes. A simple and intuitively graphical frontend makes it possible to examine and export several SQLite databases simultaneously in GUI-mode. Found and restored records are managed via a table view. It is possible to jump to the location within the binary file at any time within a hex-view, to validate each match by hand. In the command-line interface (CLI) mode, can automate the extraction process. This operational modus offers the highest performance and should be used for time-critical investigations. On the console, only need to specify the input path of the database to be analyzed. The extraction process is entirely automatic.
Further user intervention is not required. In this way, the program does support batch-processing. Another essential feature is the support of multi-core processors for the data acquisition process.

CHAPTER 6: EVALUTION

In order to evaluate the quality of the proposed algorithms, various tests have been executed. First, the detection rate should be validated. This is undoubtedly the most critical indicator for the quality of a forensic tool. Since use a heuristic, false positive or false-negative results may occur when searching for deleted records. Mostly the latter should be avoided in a forensic investigation if possible. The standard corpus for forensic investigations was used for the experiments (Nemetz et al 2018).This framework was designed to evaluate and benchmark forensic analysis methods and tools. It comprises 77 SQLite database files. The dataset is divided into different categories. Some test the correct representation of regular records and database structures. Others can be used to validate the detection of deleted or overwritten records.
Accordingly, the first 50 database files of the test set to focus on a special feature of SQL statements and some internal characteristics of the SQLite file format, while 27 databases feature explicitly deleted artefacts that may be recovered. The results for the first four categories (regular data and structures) are depicted in Table 2, whereas the detailed results of category
V (recoverable content) are summarized in Table 3. FQLite was able to read, analyze all databases of the test set successfully.
This cannot be taken for granted. A case study with the same corpus in 2018 with six different tools showed a very mixed result (Nemetz et al., 2018). None of the tested programs managed to do this. The best program was able to read 66 of 77 databases. However, 26 out of 66 were incomplete or contained errors, when looking at all categories.

The differences between the individual tools become particularly clear when comparing the deleted records' detection rate with the results achieved by FQLite in the same test (see Table 4). In this test, only the models of category 5: Deleted & overwritten contents of the individual tools are compared. The 27 databases contain exactly 278 deleted records besides the regular ones. It is naturally of special interest to recover exactly these records for the quality of a forensic tool. The best-tested tool bring2lite manages to recon 52.9% of the deleted records. FQLite, however, was able to recover all(!) deleted records, at least in this test.
However, there were also some problems with the acquisition results. For example, in test databases 04-05 and 04-06, all data records were recognized. The test data encoded with UTF16LE and UTF16BE contain some characters which are represented by surrogate pairs (4 instead of 2 bytes). In our test, they were not displayed correctly. In test case 06-02, a virtual table with spatial coordinates is created. A virtual table is an object presenting an SQL table interface but which is not stored in the database file, at least not directly. Hence, not all data records could be completely restored. For this special case, a corresponding plugin is needed. Another corpus database contains a data record that is distributed over several overflow pages due to its size. In this

particular case, this long data record was deleted. The program recognizes the record and can partially restore it. However, one of the overflow pages has already been overwritten after deletion and reused as an internal data page (type 0x5). Therefore, the restored data record contains overwritten data areas. However, the program itself cannot recognize that some of the data have been changed while no longer belongs to the original record. In another test case, a table was delete (database 0B-01.db). A new table was then created with the same column types but different table and column names.

Forensic tools 1 to 5				
Undark	SQLite Deleted Record Parser	Stellar Phoenix Repair for SQLite	SysTools SQLite Database Recovery	Sanderson Forensic Browser for SQLite
95/278 34,2%	139/278 50%	0/278 0%	0/278 0%	33/278 11,9%
Forensic tools 6 to 10				
SQLabs SQLite Doctor	Sqlite Forensic Explorer	Autopsy SQLite Deleted Records Plugin	bring2lite	FQLite
0/278 0%	73/278 26,3%	0/278 0%	147/278 52,9%	**278/278** **100%**

A previously deleted page was partially overwritten with data records for the new table. As a result, there were suddenly records from two different tables in one data page. FQLite was able to recover all records - deleted and new. However, the deleted records, actual data of the old table, were erroneously assigned to the new table. Have not really found a solution to this problem. In this case, this could only be detected with the help of the SQL script that is included with every database in the test corpus. In practice, this information will unfortunately rarely be available to the investigator. On the other hand, such a constellation is not very likely to be encountered in practice.
Finally, there was a problem with the recognition of records. Instead of the expected 20 records, FQLite showed 21recovered records. Initially assumed a false-positive hit. In

fact, FQLite engine has worked perfectly correctly. One of the records was apparently moved to a new page after some data was deleted by the de-fragmentation process of SQLite. More precisely, a copy was created. Thus, the record was suddenly twice in the database in two different pages. Apart from these exceptions, all other databases of the corpus could be restored correctly. A very encouraging result.
Since forensic investigations are time-critical, the duration of the extraction process is also a crucial measure. In the second series of tests, the possibility of parallelizing the data acquisition process was tested. It is reasonable to assume that a significant speedup can be achieved when carving for artefacts concurrently. If analyze a database page, there are usually no dependencies to other pages within the database. An exception is the so-called overflow pages, which are used when there is not enough space in a page to store a data set.
The SQLite file format states that overflow pages are chained together using a singly linked list. To answer this question, have used a slightly adapted model of the *chinook.db* database
. Since there is no suitable benchmark for such questions at the moment, used an example that is frequently used for training situations. The database in our test contained about 2,048 million entries. For us, it was primarily interesting to see how the execution time is reduced when perform our analysis in parallel on several processor cores.
The test system used was a MacBook Pro 2017 with a 2.8GHz Quad-Core Intel Core i7, 16GB RAM, and a 256GB Apple
SSD. The system has a number of 4 physical cores. A total of more than 100 test runs were conducted. The results were used to calculate the average time required. The tests show a moderate relative speedup of 2,36 when scaling the system from 1 to 4 cores. The tests were performed with version 1.0 of the program. In order to better understand what the application requires the most computing time for, a profiler was used. Surprisingly, only about 30% of the execution time was used for the actual search process. The largest part of the computing time (around 63%) was needed to decode the recovered records and oncatenating them into UTF8 strings.

CHAPTER 7: CONCLUSION

Database forensics is an upcoming research discipline over the last decade (Chopade & Pachghare, 2019). The use of carving techniques in the field of databases, for many years, an established method in digital forensics, is a relatively new approach.
In this article, presented some techniques for carving and acquisition of deleted data in SQLite databases. Besides an
algorithm for the recovery of deleted data pages, a novel heuristic for detecting deleted records on a binary level within the database's slack areas was introduced. As proof of concept, an open-source application named FQLite was presented, which implements all proposed techniques. The benchmark scope has proved that the program can handle almost all pitfalls of the

SQLite data format. In particular, FQLite performs better in this field than many other forensic solutions currently on the market. It could achieve the highest recovery rate within our test. However, it must be emphasized that the application isexplicitly not meant to be a complete forensic tool. The range of functions is limited to searching for (deleted) data record sand their export into a CSV format. Functions like picture extraction or case handling are not implemented at the moment.
Nevertheless, the results achieved are very encouraging.
Planned future improvements are related to an error-free display of UTF16 encoded data. Full support of virtual tables is also planned for one of the next program versions. As a new feature, support for WAL files will be made available in addition to the standard database files. Beyond that, are working on a further improvement of the algorithm's recognition quality and further speed up the acquisition process. Finally, plan to harden our tool against anti-forensic measures like those described by Schmitt (Schmitt, 2018).

CHAPTER 8: REFERENCE

Aouad, L.M., Kechadi, T.M., Russ, R.D.: Ants road (2012). A new tool for SQLite data recovery on android devices. In: M.

Rogers, K. Seigfried-Spellar (Eds.), ICDF2C 2012, Vol. 114 of LNICST, Springer, pages 253-263. doi: 10.1007/978-3-
642-39891-9_16

Bagley, R., Ferguson, R. I., Leimich, P. (2012). On the digital forensic analysis of the Firefox browser via recovery of SQLite artifacts from unallocated space. 6th International Conference on Cybercrime Forensics Education & Training (CFET).

Chopade, R., Pachghare, V.K. (2019). Ten years of critical review on database forensics research. In: Digital Investigation
Volume 29, pages 180-197. doi: 10.1016/j.diin.2019.04.001
Comer, D.(1979). Ubiquitous b-tree. ACM Comput Surv,11 pp. 121-137, doi: 10.1145/356770.356776

Daniels,PL (2020). Undark - a SQLite deleted and corrupted data recovery tool. project homepage.
http://pldaniels.com/undark/ (last accessed 03/03/2020)

DeGrazia, M. (2013). Python Parser to Recover Deleted SQLite Database Data.
URL: http://az4n6.blogspot.be/2013/11/python-parser-to-recover-deleted-sqlite.html

Haldar, S. (2015). SQLite Database System Design and Implementation (Second Edition). pages 256 (2015).

Jeon, S., Bang, J., Byun, K., Sangjjn, L. (2012). A recovery method of deleted record for SQLite database. Pers Ubiquit Comput
16, 707–715. doi: 10.1007/s00779-011-0428-7

Liu, Y., Xu, M., Xu, J., Zheng, N., Lin, X. (2016). SQLite Forensic Analysis Based on WAL. In: Security and Privacy in Communication Networks 12th International Conference, SecureComm 2016, Guang-zhou, China, 2016, Proceedings
Meng, C., Baier, H. (2019). bring2lite: A Structural Concept and Tool for Forensic Data Analysis and Recovery of Deleted

SQLite Records. Digital Investigation: Volume 29, Supplement, July 2019, pages 31-41, (2019). doi:
10.1016/j.diin.2019.04.017

Nemetz, S., Schmitt, S., Freiling, F. (2018). A standardized corpus for SQLite database forensics. In: Digital Investigation,vol. 24, Supplement, 2018, pages 121-130. doi: 10.1016/j.diin.2018.01.015

Pawlaszczyk, D. (2017). Digitaler Tatort, Sicherung und Verfolgung digitaler Spuren. In: Labudde D., Spranger M. (eds)
Forensik in der digitalen Welt. Spring. doi: 10.1007/978-3-662-53801-2_5

Ramisch, F., Rieger, R. (2015). Recovery of SQLite Data Using Expired Indexes. IMF '15: Proceedings of the 2015 Ninth
International Conference on IT Security Incident Management & IT Forensics 2015 pages 19–25. doi: 10.1109/IMF.2015.11

Sanderson, P. (2018). SQLite Forensics. Independently published, ISBN 978-1980293071, 315 pages (2018).
Schmitt, S.:(2018). Introducing anti-forensics to SQLite corpora and tool testing. 11th International Conference on IT Security
Incident Management IT Forensics (IMF), pages 89-106, (2018). doi: 10.1109/IMF.2018.00014

ShuN., W., Zheng, M. Xu (2014). A history records recovering method based on WAL file of firefox, In: Journal of
Computational Information Systems 10(20):8973-8982, 2014. doi: 10.12733/jcis12145

Pune Vidyarthi Griha's College of Engineering and

Technology & G.K. Pate (Wani) Institute of Management,

Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By

Student Name	**Exam Seat No.**
Abhishek Dhamdhere	
Shyam Lokhande	
Neha Bhosale	
Vaishnav Bhujbal	

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha′s College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

A Seminar Report On

DATA RECOVERY USING NoSQL

Submitted in Partial Fulfilment for the Term-work of Third year in Computer Engineering of *Savitribai Phule Pune University*.

By : Neha Bhosale

Under The Guidance of

Prof. B.C.JULME

Department of Computer Engineering

Academic Year: - 2022-2023

Pune Vidyarthi Griha's College of Engineering and

Technology & G.K. Pate (Wani) Institute of Management,

Pune- 411009.

(Affiliated to Savitribai Phule Pune University)

CERTIFICATE

This is to certify that the Seminar report entitled, "**FORENSIC RECOVERY OF SQL SERVER DATABASE: PRACTICAL APPROACH"** Submitted by,

Student Name	Exam Seat No.
Abhishek Dhamdhere	
Shyam Lokhande	
Neha Bhosale	
Vaishnav Bhujbal	

is a record of bonafide work carried out by him/her, in the partial fulfillment of the Term-work of Third year in Computer Engineering of Savitribai Phule Pune University at Pune Vidyarthi Griha's College of Engineering and Technology & G.K. Pate (Wani) Institute of Management, Pune under Savitribai Phule Pune University, Pune. This work is done during the academic year 2022-23.

Date: - 19/10/2022

Place: - Pune

Prof. B.C.Julme	Prof. M.S.Pokale	Prof.D.D.Sapkal
Seminar Guide	External Examiner	H.O.D. (Computer Engg)

Acknowledgment

I would like to express my gratitude towards my guide, Prof.B.C.Julme, Assistant Professor in the Computer Engineering Department, who has been very concerned and has aided for all the help essential for the preparation of this work. He has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research oriented venture.

I am also thankful to Prof. D.D.Sapkal, Head of Department, Computer Engineering, for the motivation and inspiration that triggered me for the seminar Work.

Neha Bhosale

T2

(T.E.Computer Engineering)

Abstract

Database forensics is becoming more important for investigators with the increased use of the information system. Although various database forensic methods such as log analysis and investigation model development have been studied, among the database forensic methods, recovering deleted data is a key technique in database investigation for DB tampering and anti-forensics. There is research to recover small-sized databases such as SQLite and EDB, but there is no prior work describing the structure of data files and technology to recover deleted data of large databases used by enterprises or large organizations. In this context, I investigate NOSQL Server, which is one of the most used large databases. I look for evidence in those places and technologies designed by NoSQL for disaster recovery purposes – namely Undo segments, Flashback and the Recycle Bin - of a compromise and the actions an attacker may have taken To contribute to the digital forensic community, I also provide the source code of the implementation; it facilitates the knowledge sharing of database forensics.

INDEX

LIST OF FIGURES

CHAPTER 1 : INTRODUCTION

Database forensics is a branch of digital forensics that analyzes the structure of database and examines contents. It has motivated forensic researchers since database manages important data of personal or enterprise. With the increased use of Internet of Things (IoT) devices, database allows countless users to access a variety of applications and stores the users data and log; database forensics is becoming more important for a forensic investigator.

Various forensic methods have been studied to find critical evidence in database, such as log analysis and investigation model development. Among the database, the associate editor coordinating the review of this manuscript and approving it for publication was Shaohua Wan. Forensic methods, recovering deleted records plays an important role in finding the evidence, because it can recover data that existed in the past or that was intentionally deleted for anti-forensics. Especially, attackers try to the database tampering to take away sensitive information or try to delete the information; by recovering deleted records, an investigator can detect and deal with these behaviors. The recovery of deleted records is also used to solve the financial fraud cases in forensic accounting.

The techniques to recover deleted records are divided into two main categories: the log-based method and engine-based method. Many researchers proposed the log-based methods that analyze the transaction log or journal log. The approach showed a notable achievement, but there is a limit that logs containing critical evidence may not exist on the database.

CHAPTER 2 : MOTIVATION

Data is one of the most powerful weapons a business has. It provides key insights about employees, customers, products and competitors. It's collected from a whole host of sources and can make or break a firm's success.

Losing important and sensitive data can have a significant impact on the way our business operates. First, losing important data can put a halt in our operations. After all, how can I proceed with our business when important, and probably sensitive, files and data are missing.

CHAPTER 3 : INTRODUCTION TO NoSQL

When people use the term “NoSQL database,” they typically use it to refer to any non-relational database. Some say the term “NoSQL” stands for “non SQL” while others say it stands for “not only SQL.” Either way, most agree that NoSQL databases are databases that store data in a format other than relational tables.

Brief history of NoSQL databases :

NoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased. Gone were the days of needing to create a complex, difficult-to-manage data model in order to avoid data duplication. Developers (rather than storage) were becoming the primary cost of software development, so NoSQL databases optimized for

developer productivity.

Figure 1 : Logo of NoSQL

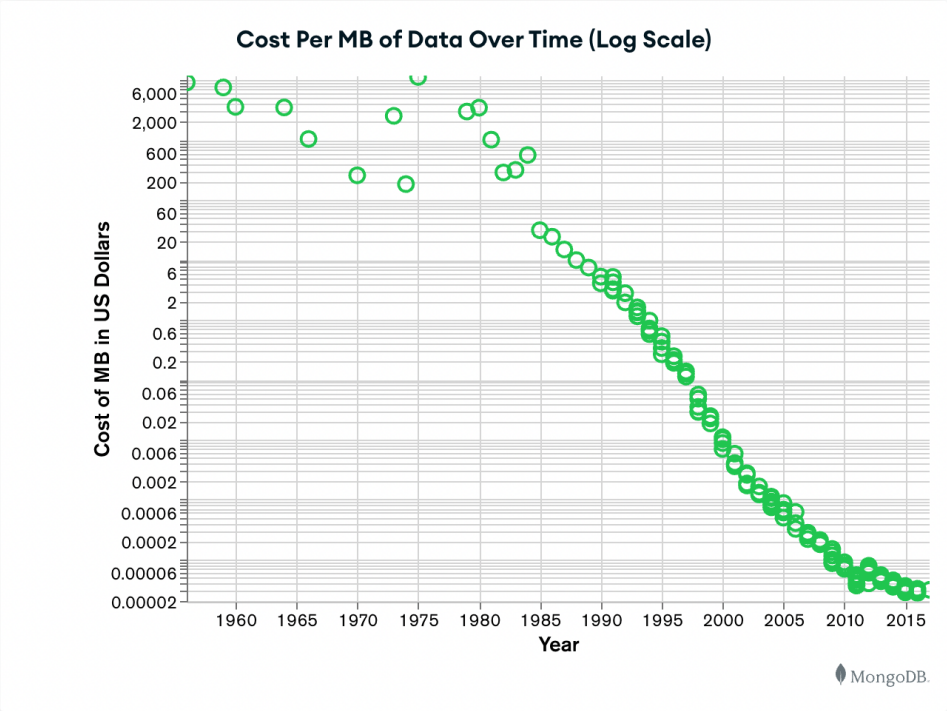


Figure 2 : Usage of Databases

3.1 NoSQL Database Features :

Each NoSQL database has its own unique features. At a high level, many NoSQL databases have the following features:

- Flexible schemas
- Horizontal scaling
- Fast queries due to the data model
- Ease of use for developers

1. Flexible Schema :

Unlike SQL databases, where you must determine and declare a table's schema before inserting data, MongoDB's collections, by default, do not require their documents to have the same schema. That is:

- The documents in a single collection do not need to have the same set of fields and the data type for a field can differ across documents within a collection.

- To change the structure of the documents in a collection, such as add new fields, remove existing fields, or change the field values to a new type, update the documents to the new structure.

This flexibility facilitates the mapping of documents to an entity or an object. Each document can match the data fields of the represented entity, even if the document has substantial variation from other documents in the collection.

In practice, however, the documents in a collection share a similar structure, and you can enforce document validation rules for a collection during update and insert operations.
See Schema Validation for details.

2. Document Structure :

The key decision in designing data models for MongoDB applications revolves around the structure of documents and how the application represents relationships between data. MongoDB allows related data to be embedded within a single document.

Embedded Data

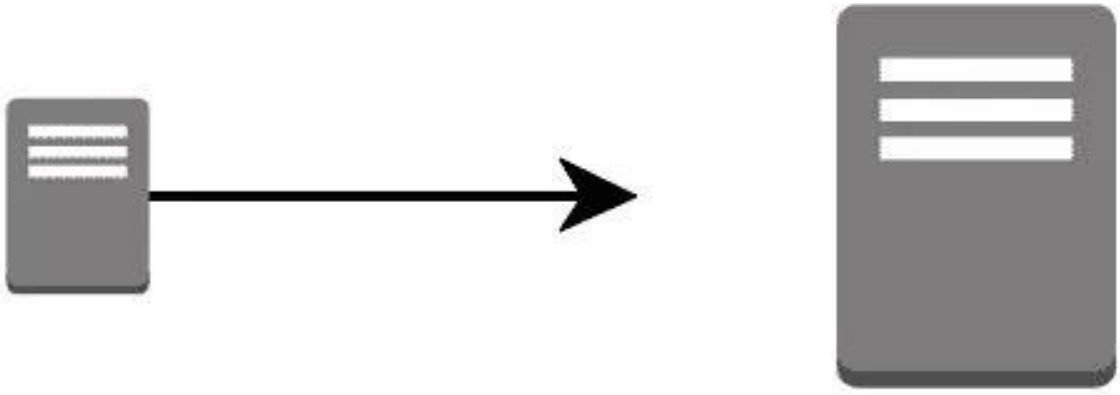
Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures in a field or array within a document. These *denormalized* data models allow applications to retrieve and manipulate related data in a single database operation.

For many use cases in MongoDB, the denormalized data model is optimal.

See Embedded Data Models for the strengths and weaknesses of embedding documents.

3. Vertical Versus Horizontal :

Figure 3 : Vertical Scaling



Vertical scaling refers to increasing the processing power of a single server or cluster. Both relational and non-relational databases can scale up, but eventually, there will be a limit in terms of maximum processing power and throughput. Additionally, there are increased costs with scaling up to high-performing hardware, as costs do not scale linearly.

Figure 4 : Horizontal Scaling

Horizontal scaling, also known as scale-out, refers to bringing on additional nodes to share the load. This is difficult with relational databases due to the difficulty in spreading out related data across nodes. With non-relational databases, this is made simpler since collections are self-contained and not coupled relationally. This allows them to be distributed across nodes more simply, as queries do not have to ″join" them together across nodes.

Scaling MongoDB horizontally is achieved through sharding (preferred) and replica sets.

4. Ease of use :

Document databases are highly flexible, allowing variations in the structure of documents and storing documents that are partially complete. One document can have others embedded in it. Fields in a document play the role of columns in a SQL database, and like columns, they can be indexed to increase search performance.

3.2 WHY NoSQL?

NoSQL databases are used in nearly every industry. Use cases range from the highly critical (e.g., storing financial data and healthcare records) to the more fun and frivolous (e.g., storing IoT readings from a smart kitty litter box).

In the following sections, I'll explore when you should choose to use a NoSQL database and common misconceptions about NoSQL databases.

3.3 WHERE TO USE NoSQL?

When deciding which database to use, decision-makers typically find one or more of the following factors lead them to selecting a NoSQL database:

- Fast-paced Agile development
- Storage of structured and semi-structured data
- Huge volumes of data
- Requirements for scale-out architecture
- Modern application paradigms like microservices and real-time streaming
- NoSQL Data storage systems makes sense for applications that process very large semi-structured data-like Log Analysis, Social Networking Feeds, Time-based data.
- To improve programmer productivity by using a database that better matches on application's needs.
- To improve data access performance via some combination of handling larger data volumes, reducing latency, and improving throughput.

CHAPTER 4 : TYPES OF NoSQL

Over time, four major types of NoSQL databases emerged: document databases, key-value databases, wide-column stores, and graph databases.

- Document databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, booleans, arrays, or objects.
- Key-value databases are a simpler type of database where each item contains keys and values.
- Wide-column stores store data in tables, rows, and dynamic columns.
- Graph databases store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes.

4.1 KEY-VALUE DATABASES :

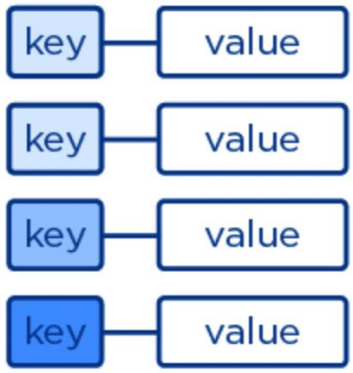


Figure 6 : Key value databases

The simplest type of NoSQL database is a key-value store. Every data element in the database is stored as a key value pair consisting of an attribute name (or "key") and a value. In a sense, a key-value store is like a relational database with only two columns: the key or attribute name (such as "state") and the value (such as "Alaska").

Use cases include shopping carts, user preferences, and user profiles.

4.2 DOCUMENT DATABASES :

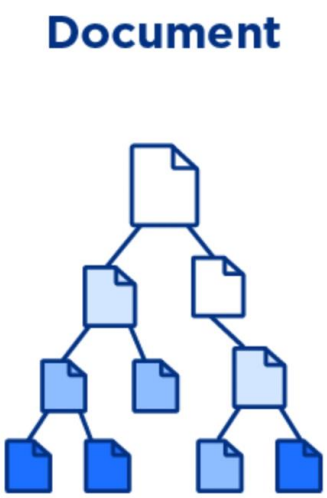


Figure 7 : Document Databases

A document database stores data in JSON, BSON, or XML documents (not Word documents or Google Docs, of course). In a document database, documents can be nested. Particular elements can be indexed for faster querying.

Documents can be stored and retrieved in a form that is much closer to the data objects used in applications, which means less translation is required to use the data in an application. SQL data must often be assembled and disassembled when moving back and forth between applications and storage.

Document databases are popular with developers because they have the flexibility to rework their document structures as needed to suit their application, shaping their data structures as their application requirements change over time. This flexibility speeds development because, in effect, data becomes like code and is under the control of developers. In SQL databases, intervention by database administrators may be required to change the structure of a database.

The most widely adopted document databases are usually implemented with a scale-out architecture, providing a clear path to scalability of both data volumes and traffic.

Use cases include ecommerce platforms, trading platforms, and mobile app development across industries.

Comparing MongoDB vs. PostgreSQL offers a detailed analysis of MongoDB, the leading NoSQL database, and PostgreSQL, one of the most popular SQL databases.

4.3 WIDE-COLUMN DATABASES :

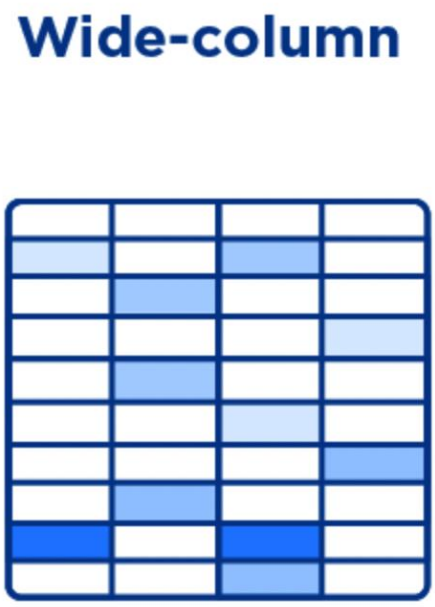


Figure 8 : Wide-Column Database

While a relational database stores data in rows and reads data row by row, a column store is organized as a set of columns. This means that when you want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data. Columns are often of the same type and benefit from more efficient compression, making reads even faster. Columnar databases can quickly aggregate the value of a given column (adding up the total sales for the year, for example). Use cases include analytics.

Unfortunately, there is no free lunch, which means that while columnar databases are great for analytics, the way in which they write data makes it very difficult for them to be strongly consistent as writes of all the columns require multiple write events on disk. Relational databases don't suffer from this problem as row data is written contiguously to disk.

4.4 GRAPH DATABASES :

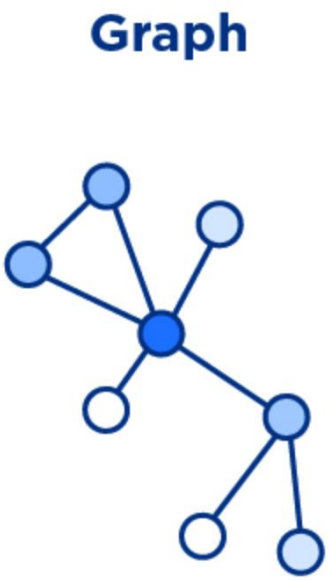


Figure 9 : Graph Database

A graph database focuses on the relationship between data elements. Each element is stored as a node (such as a person in a social media graph). The connections between elements are called links or relationships. In a graph database, connections are first-class elements of the database, stored directly. In relational databases, links are implied, using data to express the relationships.

A graph database is optimised to capture and search the connections between data elements, overcoming the overhead associated with JOINing multiple tables in SQL.

Very few real-world business systems can survive solely on graph queries. As a result graph databases are usually run alongside other more traditional databases.

Use cases include fraud detection, social networks, and knowledge graphs.

As you can see, despite a common umbrella, NoSQL databases are diverse in their data structures and their applications.

CHAPTER 5 :ARCHITECTURE OF NoSQL

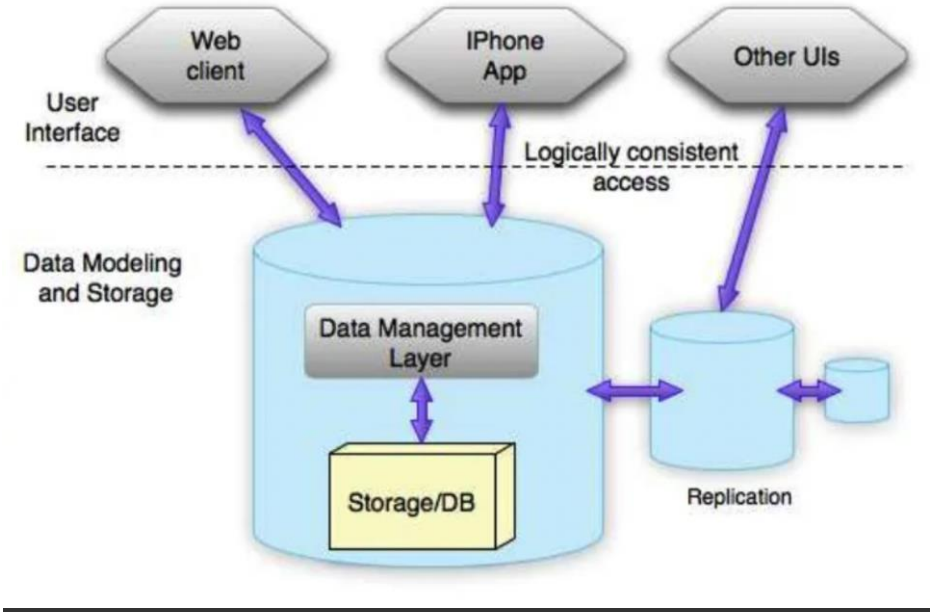


Figure 9 : Architecture of NoSQL Database

Architecture Pattern is a logical way of categorising data that will be stored on the Database. NoSQL is a type of database which helps to perform operations on big data and store it in a valid format. It is widely used because of its flexibility and a wide variety of services.

5.1 ADVANTAGES OF NoSQL :

(i) Flexible Data Model:

NoSQL databases are highly flexible as they can store and combine any type of data, both structured and unstructured, unlike relational databases that can store data in a structured way only.

(ii) Evolving Data Model :

NoSQL databases allow you to dynamically update the schema to evolve with changing requirements while ensuring that it would cause no interruption or downtime to your application.

(iii) Elastic Scalability:

NoSQL databases can scale to accommodate any type of data growth while maintaining low cost.

(iv) High Performance:

NoSQL databases are built for great performance, measured in terms of both throughput (it is a measure of overall performance) and latency (it is the delay between request and actual response).

(v) Open-source:

NoSQL databases don′t require expensive licensing fees and can run on inexpensive hardware, rendering their deployment cost-effective.

5.2 DISADVANTAGES OF NoSQL :

(i) Lack of Standardization:

There is no standard that defines rules and roles of NoSQL databases. The design and query languages of NoSQL databases vary widely between different NoSQL products – much more widely than they do among traditional SQL databases.

(ii) Backup of Database:

Backups are a drawback in NoSQL databases. Though some NoSQL databases like MongoDB provide some tools for backup, these tools are not mature enough to ensure proper complete data backup solution.

(iii) Consistency:

NoSQL puts a scalability and performance first but when it comes to a consistency of the data NoSQL doesn′t take much consideration so it makes it little insecure as compared to the relational database e.g., in NoSQL databases if you enter same set of data again, it will take it without issuing any error whereas relational databases ensure that no duplicate rows get entry in databases.

(iv) ACID:

One of the most frequently cited drawbacks of NoSQL databases is that they don't support ACID (atomicity, consistency, isolation, durability) transactions across multiple documents.

5.3 DIFFERENCE BETWEEN RDBMS AND NoSQL :

Relational Database	NoSQL
It is used to handle data coming in low velocity.	It is used to handle data coming in high velocity.
It gives only read scalability.	It gives both read and write scalability.
It manages structured data.	It manages all type of data.
Data arrives from one or few locations.	Data arrives from many locations.

It supports complex transactions.	It supports simple transactions.
It has single point of failure.	No single point of failure.
It handles data in less volume.	It handles data in high volume.
Transactions written in one location.	Transactions written in many locations.
Deployed in vertical fashion.	Deployed in Horizontal fashion.

CHAPTER 6 : RECOVERING OF DATA

NoSQL databases offer high throughput, support for huge data structures, and capacity to scale horizontally at the expense of not supporting relational data, ACID consistency and a standard SQL syntax. Due to their simplicity and flexibility, NoSQL databases are becoming very popular among web application developers. However, most NoSQL databases only provide basic backup

and restore mechanisms, which allow recovering databases from a crash, but not to remove undesired operations caused by accidental or malicious actions.

To solve this problem I propose NOSQL UNDO, a recovery approach and tool that allows database administrators to remove the effect of undesirable actions by undoing operations, leading the system to a consistent state. NOSQL UNDO leverages the logging and snapshot mechanisms built-in NoSQL databases, and is able to undo operations as long as they are present in the logs. This is, as far as I know, the first recovery service that offers these capabilities for NoSQL databases. The experimental results with MongoDB show that it is possible to undo a single operation in a log with 1,000,000 entries in around one second and to undo 10,000 incorrect operations in less than 200 seconds.

6.1 FULL-RECOVERY AND ALGORITHM :

The full recovery algorithm is the simplest recovery method among the two. It works by loading the most recent snapshot of the database, then it updates the state by executing the remaining operations, which were previously recorded in a log. The algorithm takes as input a list of incorrect operations that it is suppose to ignore when it is executing the log operations. Algorithm shows the full recovery procedure. The algorithm takes as input the most recent snapshot before the first incorrect operation and a list with the incorrect operations to undo. In line 1 a new database instance is created using that snapshot. The log entries are fetched from the global log using the getLogEntries method. This method returns an ordered list with every log entry after snapshot. Correct operations are executed in line 5. When the algorithm finishes , recoveredDatabase is a clean copy of the database without the effects of incorrect operations. This algorithm is simple and effective, but is not efficient when there are a small number of operations to undo, since it requires every correct operation in the log after the snapshot to be re-executed.

Algorithm 1 Full recovery algorithm.

```
1: recoveredDatabase ← snapshot
2: logEntries ← getLogEntries(snapshot)
3: for logEntry ∈ logEntries do
4:     if logEntry ∉ incorrectLogEntries then
5:         recoveredDatabase.execute(logEntry)
6:     end if
7: end for
8: return recoveredDatabase
```

6.2 FOCUSED-RECOVERY AND ALGORITHM :

The idea behind Focused Recovery is that instead of recovering the entire database just to erase the effects of a small set of incorrect operations, only compensation operations are executed. A compensation operation is an operation that corrects the effects of a faulty operation. The algorithm works basically the following way. For each faulty operation the affected record is reconstructed in memory by NOSQL UNDO. When the record is updated, NOSQL UNDO removes the incorrect record and inserts the correct one in the database. On the contrary of Algorithm, this algorithm only requires two write operations in the database for each faulty operation. Algorithm describes the process of erasing the effect of incorrect operations. The algorithm iterates through every incorrect operation . For each incorrect document it fetches every log entry that affected this record. For simplicity the pseudo-code assumes that there is no older snapshot and that every operation executed is in the log, therefore the recovered document is initialised empty . If there was a snapshot, then the recovered document would be initialised as a copy of the incorrect document in the snapshot. Then the reconstruction begins and every correct operation is executed in memory in the recovered record, not in the database . Once every correct operation is executed, the record is ready to be inserted in the database. First the incorrect record is deleted and finally the correct one is inserted.

Algorithm 2 Focused recovery algorithm.

```
1: for incorrectOperation ∈ incorrectOperations do
2:     corruptedRecord ← incorrectOperation.getRecord()
3:     documentLogEntries ←
       getRecordLogEntries(corruptedRecord)
4:     recoveredRecord ← {}
5:     for recordLogEntry ∈ recordLogEntries do
6:         if recordLogEntry ≠ incorrectOperation then
7:             recoveredRecord ←
               updateRecord(recoveredRecord, recordLogEntry)
8:         end if
9:     end for
10:    database.remove(corruptedRecord)
11:    database.insert(recoveredRecord)
12: end for
```

6.2 DIFFERENCE BETWEEN FULL AND FOCUSED RECOVERY :

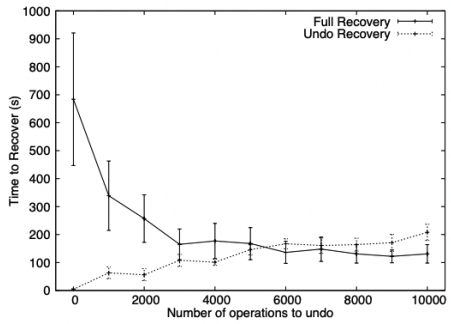


Fig. 5: Focused Recovery performance vs Full Recovery time with a confidence level of 95%.

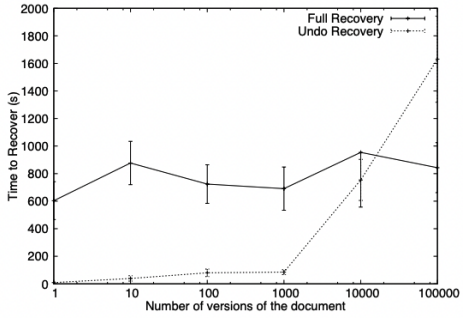


Fig. 6: Focused and Full recovery a document with different versions with 95% confidence level.

CHAPTER 7 : CONCLUSION

Database is used by many users to store and manage sensitive data of personal or enterprise. Furthermore, with the increased use of Internet of Things devices, database allows countless users to access a variety of applications and stores the users data and log; database forensics is becoming more important for a forensic investigator. Although some previous researches proposed a universal investigation method to DBMSes, there is insufficient information on how to investigate DBMS and recover deleted records practically.

The redo logs can be a rich source of evidence for a forensic examiner when they are investigating a compromised NoSQL database server. I can potentially find evidence of authentication attacks in both the TNS Listener′s log file and the audit trail.

NoSQL database provides us many benefits, consistency, availability and partition tolerance. It also provides facilities to easily store the graph data, which is not available with SQL databases.

Instead of that, there are some problems with NoSQL databases. One of the drawbacks of NoSQL databases follows CAP, according to this database which follows CAP can obtain only two out of three and have to skip the third one. Because with the CAP only two properties can be achieved. But it provides us with the facility to store the data in the form of denormalised form.

REFERENCES

[1]R. Cattell, ″Scalable SQL and NoSQL data stores,” ACM SIGMOD Record, vol. 39, no. 4, pp. 12–27, 2011.

[2]Y. Li and S. Manoharan, ″A performance comparison of SQL and NoSQL databases,” in Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, 2013, pp. 15– 19.

[3] A. Lakshman and P. Malik, ″Cassandra: a decentralized structured storage system,” ACM SIGOPS Operating Systems Review, vol. 44, no. 2, pp. 35–40, 2010.

[4] ″MongoDB.” [Online]. Available: http://www.mongodb.org

[5] T. White, Hadoop: The Definitive Guide. O′Reilly, 2009.

[6] M. Brown, Getting Started with Couchbase Server. O′Reilly, 2012.

[7] S. Sivasubramanian, ″Amazon DynamoDB: a seamlessly scalable nonrelational database service,” in Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012, pp. 729–730.

[8] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, ″Bigtable: A distributed storage system for structured data,” ACM Transactions on Computer Systems, vol. 26, no. 2, p. 4, 2008.

[9] M. Vora, ″Hadoop-HBase for large-scale data,” in Proceedings of the 2011 IEEE International Conference on Computer Science and Network Technology, 2011, pp. 601–605.

Annexure A: Glossary

1. Forensics : scientific tests or techniques used in connection with the detection of crime.
2. Database : a large amount of data that is stored in a computer and can be easily used, added to, etc.
3. RDBMS: Relational Database Management System

4. Servers : a computer that stores information that a number of computers can share
5. NoSQL: Not only SQL

Annexure B: Reference Paper

1. Title: Forensic Recovery of SQL Server Database: Practical Approach

2. Published In: Journal IEEE Access Volume 9
ISSN: 14564-14575

Annexure C: Plagiarism Report

Page Count: 32

Word Count: 4304

Character Count: 28907

Unique: 78%

Plagiarism: 22%