Forensic analysis of SQL server transaction log in unallocated area of file system

By:
Hoyong Choi, Sangjin Lee

From the proceedings of
The Digital Forensic Research Conference
DFRWS APAC 2023
Oct 17-20, 2023

Forensic Science International: Digital Investigation 46 (2023) 301605

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2023 APAC - Proceedings of the Third Annual DFRWS APAC

Forensic analysis of SQL server transaction log in unallocated area of file system

Hoyong Choi *, Sangjin Lee

School of Cybersecurity, Korea University, 145 Anam-Ro, Seongbuk-Gu, Seoul, South Korea

ARTICLE INFO

Keywords:
Digital forensics
Database forensics
Database
SQL server
Transaction log

ABSTRACT

The importance of database forensics is increasing day by day as the use of databases to store sensitive corporate and personal data increases. Database forensics is a field of digital forensics that deals with database-related incidents such as data corruption, breaches, and leaks. One of the key functions of database forensics is information reconstruction, which is the tracing of actions from the time of an event to the present based on various information stored in the database. This feature allows investigators to identify unauthorized user actions and data deletion or manipulation when an incident occurs. Database log data is primarily used to reconstruct information. Database logs include transaction logs, error logs, event logs, and trace logs. Among them, we focus on the transaction log of Microsoft SQL Server (MSSQL), one of the most popular database management systems in the world. Raw-level studies have been conducted on the transaction logs of Oracle and MySQL, other databases used at the enterprise level. However, there is very little research on MSSQL transaction logs. For this reason, we analyze the internal structure of the MSSQL transaction log. Based on these finding, we present an empirical method to identify and extract transaction log records in unallocated area.

1. Introduction

Nowadays, everything that is carried out online is recorded and stored in databases. The importance of database security, which stores and manages personal information as well as sensitive corporate data, is higher than ever. Significant efforts are required to deal with security incidents through access control, encryption, and maintenance policies for databases. However, as has been revealed in various studies and practice, it is virtually impossible to completely prevent threats. Therefore, it is also important to prepare appropriate countermeasures.

Database forensics is a field of digital forensics that deals with database-related incidents such as data tampering, breach, and leakages (Choi et al., 2021). Database forensics aims to answer W5H questions (Who? What? When? Where? Why? and How?) about database incidents through a series of processes such as identification, collection, preservation, analysis, and reporting (Chopade and Pachghare, 2019). An important function of database forensics is to reconstruct the information stored in the database at the time of the incident and to reconstruct a series of actions until reaching the current state of the database (Sablatura and Zhou, 2017) (Fasan and Olivier, 2012a) (Fasan and Olivier, 2012b) (Adedayo and Olivier, 2015). In this regard, various studies have been conducted on various databases, and log files of databases have been used as input data in this process. In other words, log files contain very valuable information from the point of view of forensic analysts (Fowler and Gold).

Database logs include audit logs, error logs, transaction logs, database trace logs (Adedayo and Olivier, 2015). This study analyzes the transaction logs of Microsoft SQL Server (henceforth referred to as MSSQL), a relational database with a high global market share (DB-Engines Ranking, 2023). Previous studies have analyzed the internal structure of transaction log files of Oracle and MySQL, and data reconstruction have been conducted (Litchfield) (Frühwirt et al., 2013). However, few studies have been conducted on transaction logs of MSSQL databases. Also, in earlier versions of MSSQL, there was no capability with SQL Server's own tools to read the transaction logs. This meant that database experts were forced to either buy third-party tools, or take wild guesses at restoration points for database recovery (Foster et al., 2016). In conclusion, a raw-level study of the MSSQL transaction log is required.

The transaction log is a critical component of a MSSQL database for ACID (atomicity, consistency, isolation and durability) compliance. When the SQL server service is restarted, the database enters the

* Corresponding author.
E-mail addresses: hoyoi05@korea.ac.kr (H. Choi), sangjin@korea.ac.kr (S. Lee).

https://doi.org/10.1016/j.fsidi.2023.301605

Available online 13 October 2023

recovery state, in which the MSSQL Database Engine reads the SQL server transaction log file to make sure that the database is in a consistent state. It does this by writing the committed transactions to the data file in a roll-forward process, and undoing all uncommitted transactions in a roll-back process. In addition, the transaction log file is used to restore the database to a specific point of time, in case of a disaster or system failure. It can be also used to return the database to the previous state when a ROLLBACK statement is executed after a transaction (Yaseen, 2023).

In this study, we analyze the detailed internal structure of transaction log files and propose a method to identify and extract transaction log records remaining in the unallocated area of the file system by utilizing structural features. We also present a way to reconstruct the query that caused the log record to occur using the extracted log record. Finally, we implement the method proposed in this study as a tool to perform evaluations on real data as well as self-generated data. The main contribution of this paper is summarized as follows:

- We present the internal structure of the MSSQL transaction log file. There are few studies on MSSQL transaction log files. In this paper,
- We introduce identification algorithm of MSSQL transaction log record based on the record structure. In addition, we propose a method for reconstructing SQL queries.
- Data records can be recovered by extracting transaction log records from the unallocated area of the file system.
- User behavior can be tracked, and data records can be reconstructed through extracted transaction log records.
- We publish our implementation[1] to share database forensic knowledge.

The remainder of this paper is organized as follows: Related works are presented in Section 2. In Section 3, we introduce an internal structure of the MSSQL transaction log file. Next, we discuss the characteristics of transaction log records to be identified and present a method for extracting transaction log records in the unallocated area using these characteristics and reconstructing SQL queries in Section 4. In Section 5, we implement the method presented in this paper. Finally, in Section 6 and 7, we evaluate the performance of the proposed method based on actual data and self-generated data and discuss conclusions and future works.

2. Related works

Various forensic methods for examining databases, such as log analysis and investigation model development, have been studied. Among these methods, several studies have been conducted to reconstruct information stored in databases using log data.

2.1. Data reconstruction

Although there are various types of databases, each database has its own transaction log file. In addition to transaction log files, each database management system has several types of log data.

Adedayo et al. described what kind of log data the following databases contain: MySQL, MSSQL, PostreSQL, Oracle, DB2, and Sybase. They also researched the default logging configuration of each database (Adedayo and Olivier, 2015). The authors mention that the default log setting on a database may not be adequate for forensic information reconstruction. So, they proposed an ideal log setting method for reconstructing information stored in database.

Fasan et al. presented an algorithm, which is based on the formal model of relational databases, for reconstructing a database for forensic purposes (Fasan and Olivier, 2012b). The authors use a relational algebra log and value blocks of relations to perform database reconstruction along with inverse relational algebra operators.

Tripathi et al. proposed a block diagram that may guide a database forensic examiner to obtain the evidence to detect database tampering (Tripathi and Meshram, 2012). The target database is Oracle, and locations where evidence for tamper detection can be obtained were identified based on Litchfield's Oracle database forensic papers. Key evidence includes redo logs that store all data change histories, data blocks that may include dropped objects, TNS (Transparent Network Substrate) Listener's log files, and audit trails that can identify attacker intrusion information.

Sablatura et al. discussed the concepts behind reconstruction in digital forensics, and the dimensions of database research (Sablatura and Zhou, 2017). They mentioned that each of the dimensions of database forensics are united under the use of log files to reconstruct the activity that has occurred on the database. They described ideal log setting and forensically aware database logging for database reconstruction.

2.2. Log file analysis on database

Wright et al. conducted forensic performance tests of LogMiner, a basic built-in tool available from Oracle 8i (Wright, 2005). It is possible to check the DML/DDL (Data Manipulation Language/Data Definition Language) history by analyzing the redo log. The authors tested whether a forensic timeline could be constructed through the tool and whether the derived results showed accurate time values. They concluded that LogMiner could be the main source of information in a forensic investigation and allow subsequent recovery of lost data.

Litchfield presented the mechanism that redo logs work and analyzed the undocumented binary format of redo logs (Litchfield). Oracle database has a function to dump log data and LogMiner to analyze log data, but the author noted that using these functions not only corrupts the data that could potentially become forensic evidence, but also extracts only part of the log data so that the forensic investigators may not be viewing all the evidence. He presented binary analysis and query reconstruction methods for DML and DDL queries, and mentioned that being able to interpret the binary format can turn up evidence that tools such as LogMiner or the ASCII dump files don't.

Frühwirt et al. proposed methods for forensic analysis of MySQL InnoDB by analyzing the redo logs, which are primarily used for crash recovery within the storage engine (Frühwirt et al., 2013). The authors analyzed the internal structure of the redo log and implemented a prototype to identify and recover transactions. Based on the prototype implementation, they showed methods for recovering INSERT, DELETE, and UPDATE statements issued against a database.

Li described the mechanism of locating transaction log records by LSN (Log Sequence Number) in the process of MSSQL instance recovery. He explained the detailed process of forward and rollback operations (Li, 2021), and how to interpret the meaning of LSN. He also calculated the physical location of redo log records in MSSQL transaction log file.

3. Basic structure of MSSQL transaction log file

MSSQL is one of the most popular relational databases worldwide. A DBMS (Database Management System) installation contains a set of system databases as well as user databases that are created as needed. Like the system databases, the user database consists of three types of files: the primary data file, secondary data files, and transaction log files. The primary and secondary data files are used to store database objects such as tables and rows. The transaction log files keep records of the events that occur on the database, making them useful for recovery from failed transactions and retrieval of a consistent version of the database if the system crashes.

In term of a forensic examiner, transaction log files can be valuable evidence that contain important information. In this paper, we analyze

[1] URL: https://github.com/dbforensic/sql-transaction-log-parser.

the internal structure of undocumented transaction log file to identify and extract log records. However, since the structure is not documented, the meaning of all fields cannot be interpreted, and continuous analysis is required.

3.1. Virtual Log Files (VLFs)

MSSQL internally divides each transaction log file into several Virtual Log Files (VLFs), as shown in Fig. 1. A VLF is made up of several blocks of 512 bytes in size. VLFs do not have a fixed size, and there is no fixed number of VLFs for transaction log files. MSSQL dynamically determines the size of the VLF while creating or extending the log file. Administrators cannot configure or set the size or number of VLFs.

The information of each VLF in the current database can be obtained by executing the *dbcc loginfo* command (Delaney and Freeman, 2013). The output of this command, as shown in Fig. 2. **FSeqNo** column in Fig. 2 indicates the sequential number of the VLF, and Status indicates whether the corresponding VLF is allocated. If this column value is 2, it means the corresponding VLF is currently allocated (Li, 2021). **FileSize** column is the size of the VLF, and **StartOffset** column is the starting offset of the VLF. Parity is a value for maintaining the integrity of the VLF. Each block in a VLF has parity bits. When the log is first created, it is zero-initialized. As the log is written, the first byte of each block in the VLF has parity bits stamped on it. The initial value of the parity bits is 64, and it changes to 128 when the VLF is reused. Then, for each successive reuse of a VLF, the parity bits flip back-and-forth between 64 and 128. The original data of the first byte of each block changed to the parity bits is stored in the log blocks to which the block belongs.

Information on each VLF can be found in the VLF header area, as shown in Fig. 3. The following information is included in the VLF header found in this study: **FSeqNo**, **FileSize**, **StartOffset**.

3.2. Log blocks

Log blocks consists of several blocks within the VLF, and their size is variable. Fig. 4 shows the overall structure of the log blocks, which includes the log block header, log records, the array of record offsets, and the original bytes of parity. The log record is represented in Section 3.3.

The log block header is 72 bytes in size. There are currently three field values analyzed: **The number of slots**, **the size of log blocks**, and **first LSN in log blocks**. Each field value in the log file can be interpreted as little-endian.

The number of slots field indicates the number of live log records within the log blocks. **The size of log blocks** field indicates the size of in-use area within the log blocks. A log blocks consists of several blocks, but there may be unused blocks. Therefore, the total size of the log blocks is a multiple of the block size, but the size of in-use area can be verified through **The size of log blocks** field. This field is the size from the beginning of the log blocks to the array of record offset area, and indicates the end of the array of record offset area. **First LSN in log blocks** field is literally the first LSN in the log blocks.

The array of record offsets indicates the location where log records are stored on the log blocks. It consists of 2-byte values that represent the start position of each record. As seen in Fig. 4, the value at the end of the

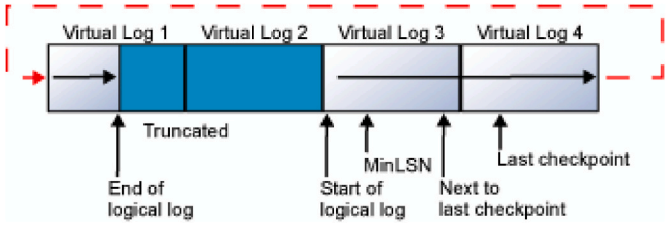


Fig. 1. MSSQL Log file structure (SQL Server transaction log architecture and management guide).

	RecoveryUnitId	FileId	FileSize	StartOffset	FSeqNo	Status	Parity	CreateLSN
1	0	2	253952	8192	37	2	64	0
2	0	2	253952	262144	0	0	0	0
3	0	2	253952	516096	0	0	0	0
4	0	2	278528	770048	0	0	0	0
5	0	2	16777216	1048576	0	0	0	37000000020800001
6	0	2	16777216	17825792	0	0	0	37000000020800001
7	0	2	16777216	34603008	0	0	0	37000000020800001
8	0	2	16777216	51380224	0	0	0	37000000020800001

Fig. 2. dbcc loginfo

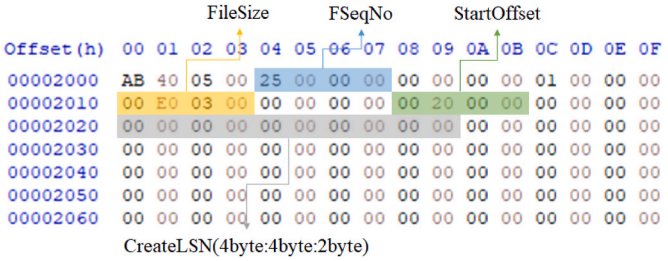


Fig. 3. VLF header structure.

array points to the first record on the log blocks. The array of record offsets is stored in reverse order.

As mentioned above, the parity bits can have a value of 0, 64, or 128. However, the first byte of each block, which is now used for parity bits, also includes other information. The fifth bit of the first byte indicates whether the block is the first block in the log blocks, and the fourth bit indicates whether it is the last block. For example, if the first byte of the block is 0x88, the parity bits is 128, which indicates that the block is the last block of the log blocks. The parity bits is similar to Fixup Arrays of NTFS (Fixup, 2023) and Tornbits of MSSQL data files (Choi et al., 2021).

The original byte of parity is an area that stores the original data of the first byte of each block that has been changed to the parity bits. It stores the original bytes in reverse order as a 1-byte array. If there is an unallocated block in the log blocks, the original byte of the block is also overwritten with the parity bits.

3.3. Log record

A transaction log consists of groups of log records, which are stored in log blocks. The fn_dblog() system function, which allows users to read through the active portion of the transaction log file and retrieve useful information about modification activities in the database, is used to analyze the log record structure. All log records have a fixed size format of 24 bytes, and the rest of the structure has a different format depending on the operation. The operations and identifiers, which are the main research subjects of this paper, are as follows:

- LOP_INSERT_ROWS (0x02)
- LOP_DELETE_ROWS (0x03)
- LOP_MODIFY_ROW (0x04)
- LOP_BEGIN_XACT (0x80)
- LOP_COMMIT_XACT (0x81)

Fig. 5 shows the structure of the 24-byte common area of all log records. **Log Record Fixed Length** field indicates the size of the fixed-length area of the record and has a different value depending on the operation. **Previous LSN** exists for the log record chain of log records. Log records belonging to the same transaction can be grouped based on **Transaction ID**. **Operation** field indicates what type of data is stored in the log record. For example, when using an INSERT query, the **Operation** field has a value of LOP_INSERT_ROWS, a value of LOP_DELETE_ROWS for a DELETE query, and a value of LOP_MODIFY_ROW for an

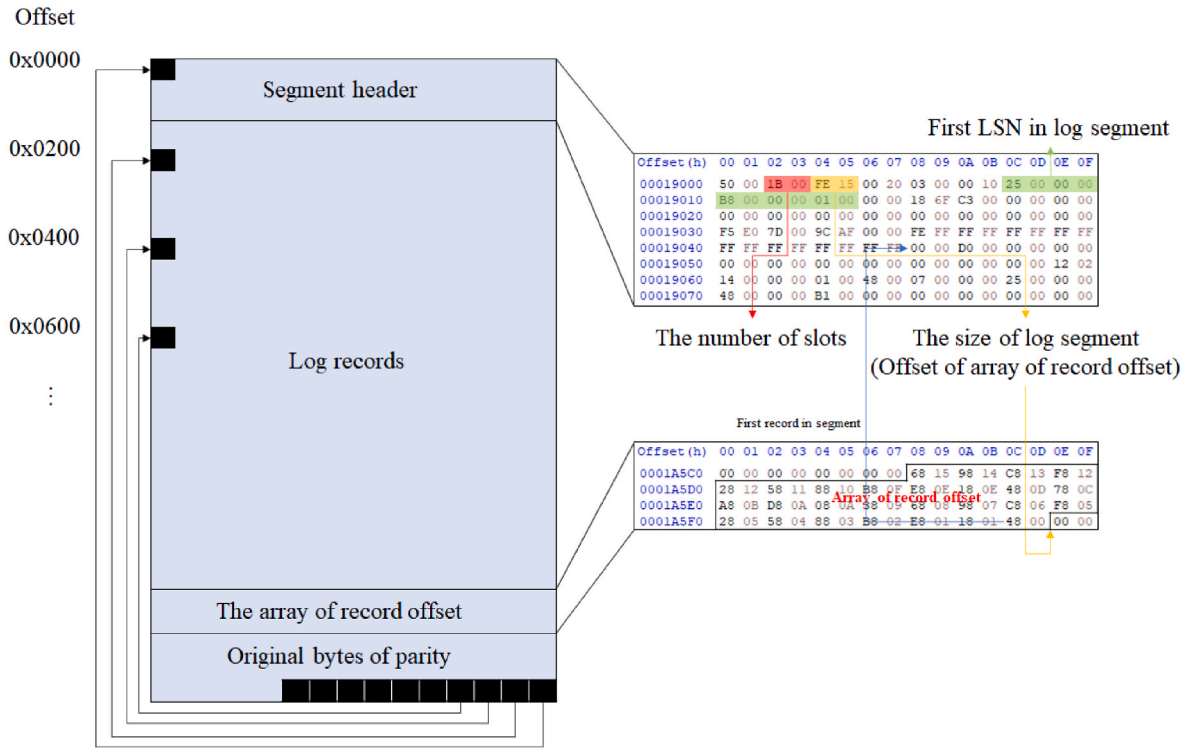


Fig. 4. Log blocks structure (applied with parity).

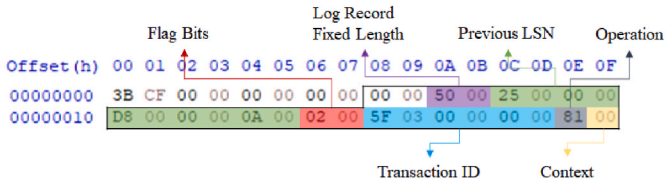


Fig. 5. Record structure: Common area.

UPDATE query.

Fig. 6 shows the log record structure for the following operations: LOP_INSERT_ROWS, LOP_DELETE_ROWS, and LOP_MODIFY_ROW. **Slot ID** and **Page ID** fields indicate the location of the data row in the data file that the log record affects. **Page ID** field contains the file ID and page number of data file. For example, in the case of a log record with an operation value of LOP_MODIFY_ROW, the position of changed data in the data file can be verified through **Slot ID** and **Page ID** fields. **PartitionId** field indicates the table the log record affects. **Offset in Row** field indicates the starting position of the modified data within the data row,

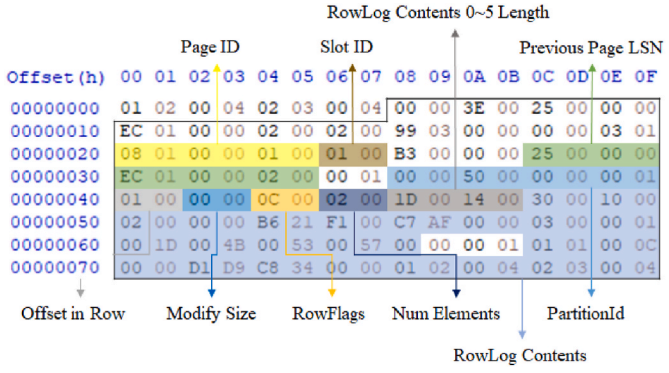


Fig. 6. Record structure: INSERT/DELETE/MODIFY.

and **Modify Size** field indicates the size of the modified data. The log records have a fixed length of 0x35, and after the fixed-length area, the total log record length is determined through **Num Elements** and **RowLog Contents Length** fields.

The data to be stored in **RowLog Contents** for each operation is determined. In the LOP_INSERT_ROWS record, the entire inserted data is stored in the first element of **RowLog Contents**. In the LOP_DELETE_ROWS record, data to be deleted is stored in the first element of **RowLog Contents**. In the above two cases, data records can be reconstructed only with the data stored in **RowLog Contents**. However, in the case of LOP_MODIFY_ROW, data before change is stored in **RowLog Contents 0** and data after change is stored in **RowLog Contents 1**, and each data is part of a data record. Therefore, the entire data record stored in the data file is required to reconstruct the data record before the change.

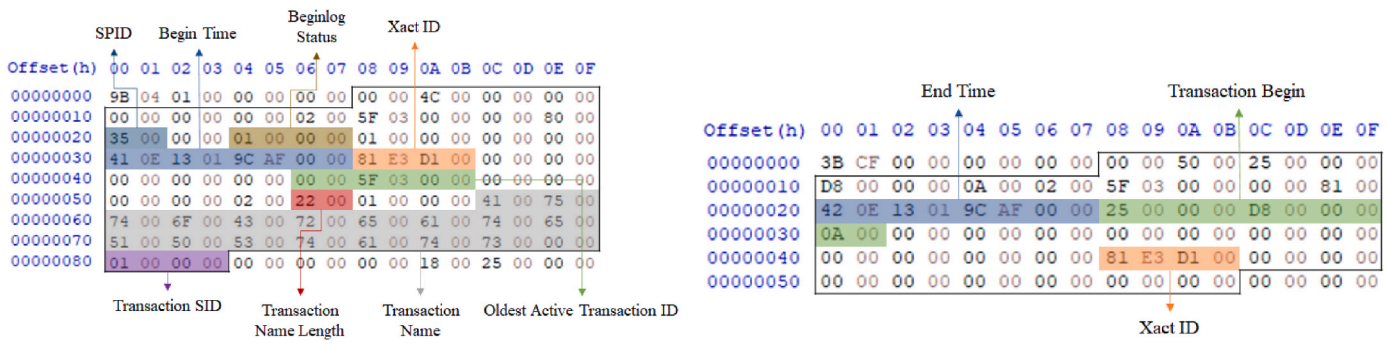
Fig. 7 shows the log record structures of LOP_BEGIN_XACT and LOP_COMMIT_XACT. Each log record contains the execution and end times of a transaction. The time value consists of 8 bytes, the first 4 bytes (0x0000AF9C as in Fig. 7a) as the number of days since 1900-01-01, and subsequent 4 bytes (0x01130E41 as in Fig. 7a) as the number of ticks (1/300 of second) since the midnight (CAST Hex Value for Datetime, 2023).

4. Proposed method

This section proposes a method for identifying and extracting log records in the unallocated area. In this paper, we assume that there are two cases in which transaction log remains in the unallocated area. The first is when the size of the transaction log file shrinks. Transaction log files shrink in the following cases (Delaney and Freeman, 2013):

- Execute BACKUP LOG statement
- Set recovery mode to SIMPLE
- Never do a full backup of the database

Contrary to the assumption that data remains in the unallocated area, when the size of a transaction log file shrinks, the data is truncated



(a) BEGIN (b) COMMIT

Fig. 7. Record structure: BEGIN/COMMIT.

and permanently deleted. That means all log records prior to the oldest active transaction are invalidated and all VLFs not containing any part of the active log are marked as reusable (Delaney and Freeman, 2013). Therefore, it does not remain in the unallocated area. The second case is when the backup file of the transaction log file is deleted. In general, database servers use a backup function periodically to control the size of transaction log files and roll back data files in preparation for accidents. The backup files are kept for a certain period according to the maintenance policy and then deleted to adjust the capacity of the storage volume. Part of the deleted transaction log backup file remains in the unallocated area of the file system.

There are two methods for identifying data fragments in the unallocated area: content-based and format-based (Park and Lee, 2014). This paper only focuses on log records, so a format-based method is adopted. Fig. 8 shows the procedure of the method presented in this paper, which consists of two steps: log record identification and query reconstruction.

4.1. *Log record identification*

The first step is to identify log records using signatures. Although there is no signature or magic number for identifying log records, the log records' fixed length area is used as a signature for signature detection. Table 1 shows signatures and operation identifiers for each log record. Log record identification verifies the existence of an operation value that matches the signature at a 16-byte location based on the detected location, after signature detection.

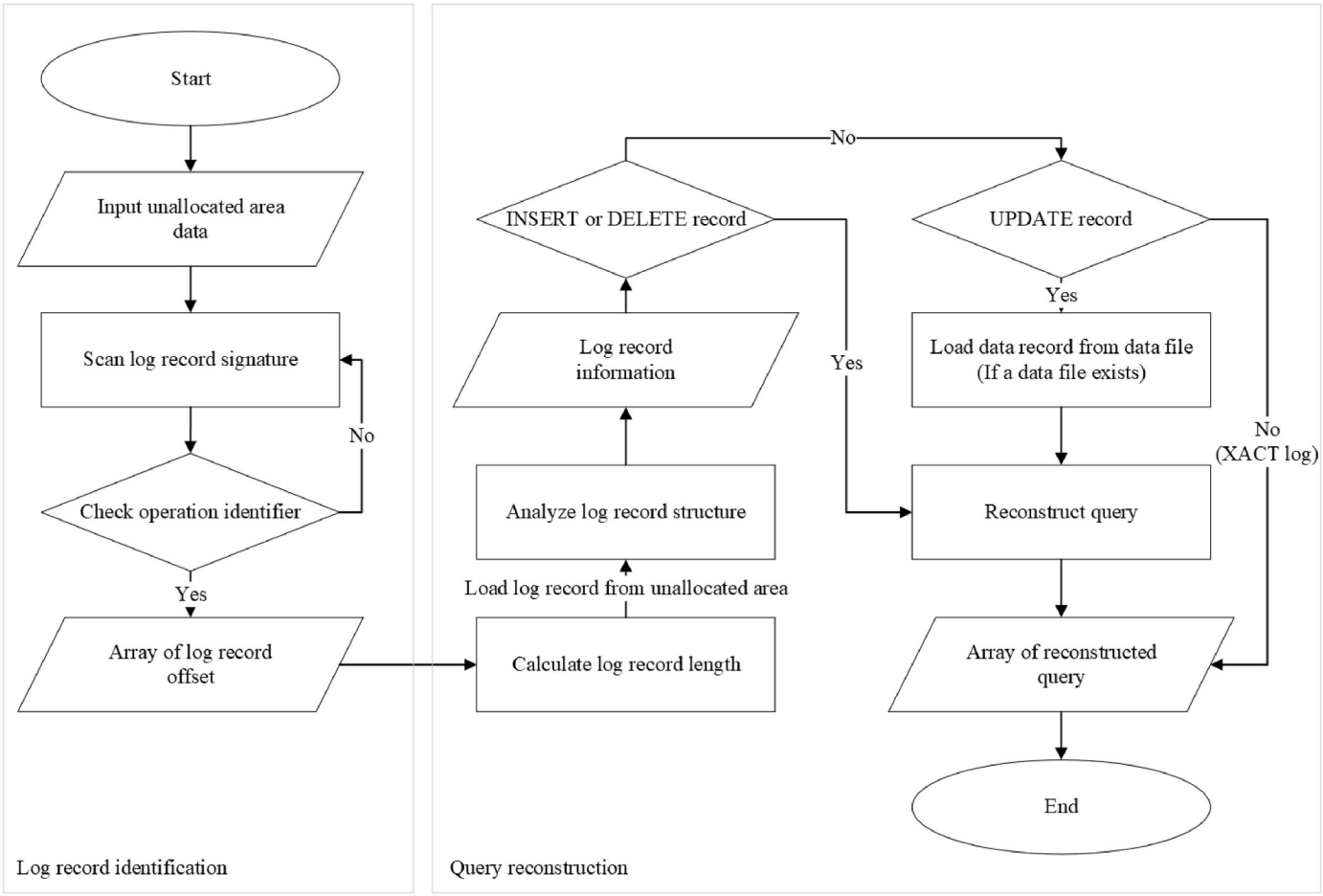


Fig. 8. Procedure of the proposed method.

Table 1
Log record signature.

Operation (identifier)	Signature
LOP_INSERT_ROWS (2)	0x00003E00
LOP_DELETE_ROWS (3)	0x40003E00
LOP_MODIFY_ROW (4)	0x48003E00
	0x80003E00
	0x88003E00
LOP_BEGIN_XACT (0x80)	0x00004C00
	0x40004C00
	0x48004C00
	0x80004C00
	0x88004C00
LOP_COMMIT_XACT (0x81)	0x00005000
	0x40005000
	0x48005000
	0x80005000
	0x88005000

Since the data to be verified may be the first 4 bytes of the block, signatures considering the parity bits are also added. For example, if signature 0x40003E00 is detected and the value located at the offset 0x16 based on the detected location is 0x02, then the data at the detected location is used for INSERT query reconstruction.

4.2. Query reconstruction

In this paper, we perform reconstruction for INSERT, DELETE, and UPDATE queries. INSERT, DELETE, and UPDATE queries can be reconstructed through a single log record analysis. However, queries such as DROP and CREATE TABLE cannot be reconstructed only with a single log record analysis. These queries can only be analyzed by reconstructing the entire transaction log by analyzing each log record that occurs. Therefore, in this paper, INSERT, DELETE, and UPDATE queries that can be analyzed based on a single log record were selected as analysis targets. Query reconstruction consists of two steps: log record structure analysis and data record structure analysis.

The procedure to analyze log record from unallocated area data is shown in Fig. 8. First, it calculates the log record length based on the identified log record offset. Next, log record data is loaded from the unallocated area through the calculated record length. Lastly, it analyzes the log record structure according to the operation.

Query reconstruction is performed according to the operation. To reconstruct a query, not only the log record, but also the data file of the database is required because information of the table schema is needed. So, Choi et al. is referred to analyze data file and data record (Choi et al., 2021).

As explained in Section 3.3, the INSERT and DELETE data record is stored in the first element of **RowLog Contents**. Query reconstruction is performed using the data records stored in **RowLog Contents 0**. Since the conditional statement at the time of DELETE query execution is not known, the conditional statement is constructed based on the column data of the data record. Fig. 9 shows the reconstruction results of INSERT or DELETE queries.

Unlike DELETE and INSERT records, only part of the data record is stored in the log record for UPDATE records. Therefore, the data records to which the UPDATE query is applied are acquired from the data file, and then query reconstruction is performed. As described in Section 3.3, in the case of the LOP_MODIFY_ROW log record created when an UPDATE query is executed, the data before change is stored in the first element of **RowLog Contents** and the data after change is stored in the second element. Using the data of **RowLog Contents** and **Offset in row** field, the data record obtained from the data file can be converted to the data before executing the UPDATE query. UPDATE query reconstruction is conducted through the acquired data records before and after UPDATE query execution. Fig. 10 shows the reconstruction results of UPDATE query.

After reconstructing the query, LOP_BEGIN_XACT and LOP_COMMIT_XACT log record, which have the same transaction ID as the reconstructed log record, indicate the start and end times of the query.

5. Implementation

In this paper, two types of modules were implemented. First, to verify the log file structure in Section 3, a module was developed to analyzes the transaction log file(.ldf). This module basically analyzes the live log record in the log file, and when data files (.mdf) are input, INSERT, UPDATE, and DELETE query reconstruction of the log record on the user table is performed.

Second, to verify the method proposed in this paper, a module was developed to identify log records in the unallocated area and analyze the log record structure. This module also reconstructs the query for the user table when the data file of the database that is the target of transaction log backup is input.

Fig. 11 shows the developed modules and how to use them. It was developed based on Python 3.11 and consists of 3 files. The datafile.py file is a data file analysis module of MSSQL (Choi et al., 2021), and the logfile.py file is a log file analysis module in which the method proposed in this paper is implemented. According to the type of input data, there are four modes.

- 0: Only transaction log file (.ldf)
- 1: Transaction log file with data file (.mdf)
- 2: Only unallocated area data
- 3: Unallocated area data with data file (.mdf)

For modes 0 and 2, only log record identification and structural analysis are performed, as there are no data files required to reconstruct the query. In modes 1 and 3, the input data file is analyzed, and query reconstruction is also performed.

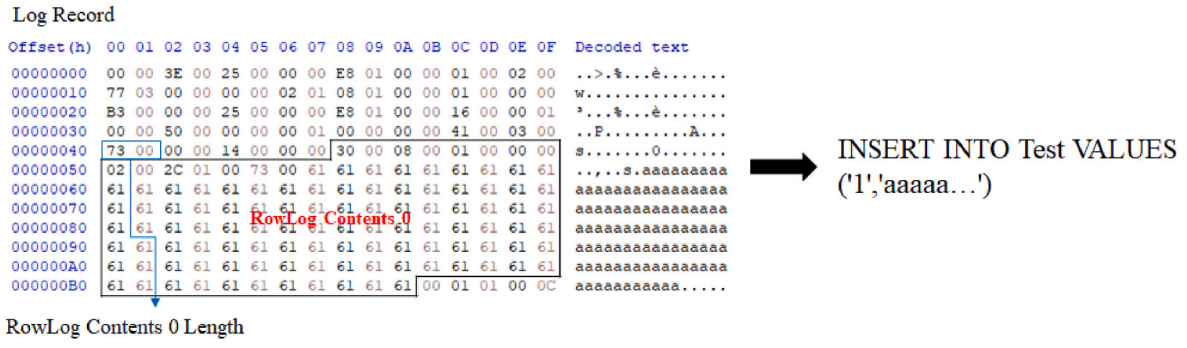


Fig. 9. Result of reconstruction query: INSERT or DELETE.

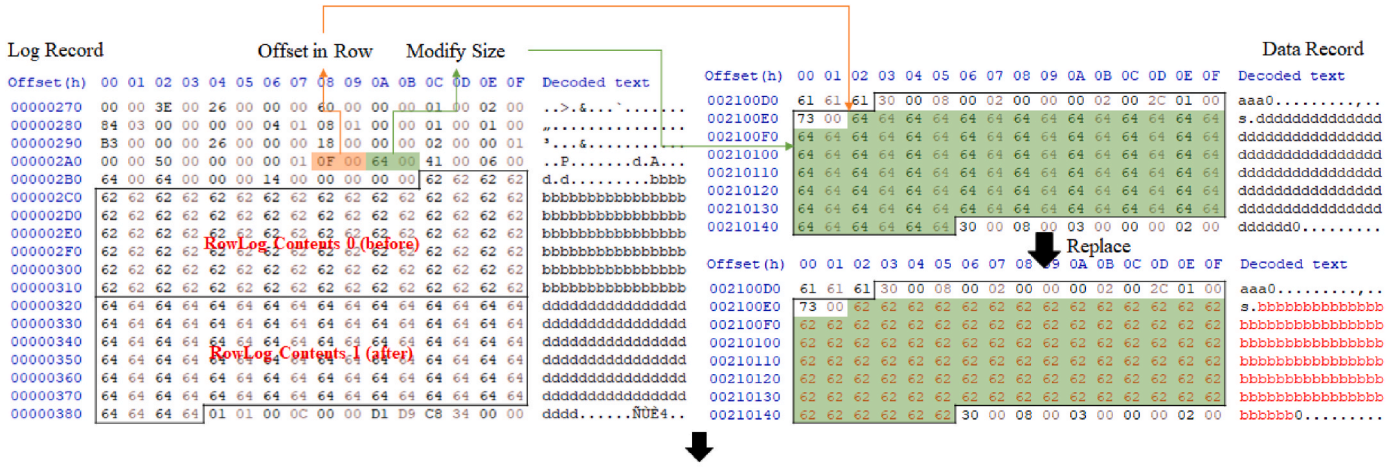


Fig. 10. Result of reconstruction query: UPDATE.

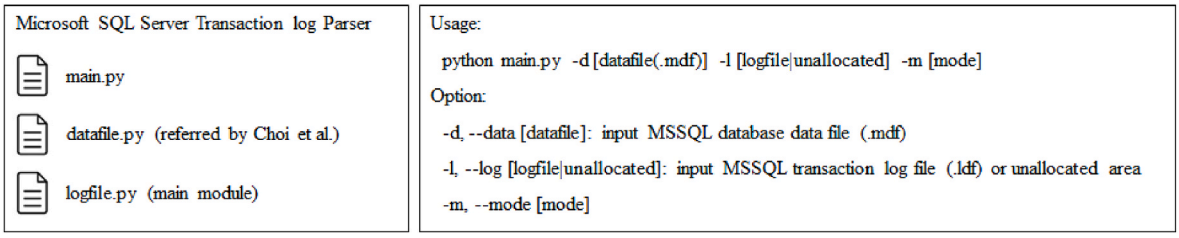


Fig. 11. Components of implemented modules and an usage.

The result data includes query strings for data that has been successfully reconstructed, and log records that cannot be reconstructed are output as raw data along with transaction start and end times.

6. Experiment and result

In this paper, we present a method for identifying and extracting transaction log records in the unallocated area and reconstructing the query. To evaluate this method, experiments were conducted based on two types of data sets.

The first data set was self-generated, as there is no sample data set to evaluate the method proposed in this paper. The second data set utilized live SQL server data.

6.1. Self-generated data set

In this section, two types of experiments were conducted. First, to verify the log file structure analyzed in Section 3, the self-generated data set log file was analyzed using the module implemented in Section 5. Additionally, to verify the analysis results, a comparative analysis is conducted with commercial tools used in the forensic field, such as ApexSQL Recover (ApexSQL) and Stellar Log Analyzer for MSSQL Database (Stellar Data Recovery). All data sets generated in this paper are based on SQL Server 2019.

The data set to validate for the log file structure is created in the following scenario:

1. Insert 1000 data rows 3 times
2. Update 1000 data rows
3. Delete 1000 data rows

If log files are correctly analyzed, a total of 5000 log records should be identified. The analysis module implemented in this paper and commercial tools successfully detected 5000 log records. Table 2 shows the validation of the analysis in Section 3.

Second, to verify the method proposed in this paper, a virtual volume was created to back up the transaction log. Then, some of the backup files were deleted. The detailed scenario is as follows:

1. Database full backup and transaction log backup
2. Insert 1,000 data rows
3. Transaction log backup
4. Insert 1,000 data rows
5. Transaction log backup
6. Insert 1,000 data rows
7. Transaction log backup
8. Update 1,000 data rows & Delete 1,000 data rows
9. Database full backup and transaction log backup
10. Delete the 1st to 3rd transaction log files

The scenario was created such that database full backups and transaction log backups were performed periodically on a real system. It was assumed that deleted transaction log files were deleted after the retention period in the real system. Then, the unallocated area in the volume was extracted using EnCase, a widely used forensic tool (OpenText Encase Forensic, 2023), and analyzed through the module implemented in this paper.

Table 2
Log record identification performance for each query (the number of identified log record/total log record).

Tool	Insert	Delete	Update
Proposed Method	3000/3000	1000/1000	1000/1000
ApexSQL	3000/3000	1000/1000	1000/1000
Stellar Log	3000/3000	1000/1000	1000/1000

6.2. Real case: SQL server 2014

Experimental data were extracted from the SQL Server 2014 server that had been used for about 8 years. The server contains several databases, and the maintenance plan function was used to perform database and transaction log backup. The backup files have an expiration date and are deleted after the expiration date.

In the case of the database to be analyzed, daily database backup and hourly transaction log backup were performed, and a plan to keep the backup files for two weeks was set. Fig. 12 shows database backup files viewed through EnCase. As backup files were stored or deleted in a 2 TB volume, the data extraction function of EnCase was used to extract unallocated area data from the volume. Then, analysis was conducted by inserting the unallocated area data and data files of the database to be analyzed into the module implemented in this paper. The size of the extracted unallocated area was 1.18 TB.

6.3. Experimental results

As a result of the experiment on the self-generated data set, it was confirmed that the log file structure was correctly analyzed, as shown in the Table 2. Additionally, to verify the method proposed in this paper, the results of inserting the unallocated area of the sample data and the database data file are shown in the Table 3.

As mentioned in the scenario of self-generated data, since three transaction log backup files were deleted, the number of LOP_INSERT_ROWS, LOP_DELETE_ROWS, and LOP_MODIFY_ROW existing in each file was compared with the number of records reconstructed by the module implemented in this paper. The first file is the back up file before the transaction occurred, so there are no log records for the user table. The second file contains log records with 1,000 INSERT records. Of these, 974 log records were detected and reconstructed in the unallocated area. The last file has 1,000 INSERT records, of which 967 were reconstructed from the unallocated area.

As a result of experiments on the data of a real case, the identification method presented in this paper identified 37 million expected offsets as log records. 50,000 offsets were sampled to analyze the log record structure and reconstruct queries. Of the 50,000 sampled offsets, log record analysis was performed on a total of 2,587 offsets, of which 27 records were reconstructed. Fig. 13 shows part of the reconstructed queries.

The results of the experiment show that more than 95% of the deleted log records for the self-generated data set were detected and reconstructed in the unallocated area. In data set of the real case, a large number of data expected to be log records were detected, and some of them were sampled and reconstructed. However, the number of log records reconstructed was small compared to the number of signatures detected. This is because the data set of real data has a very large capacity and a long period of use, so various types of data are scattered.

Name	File Ext	Logical Size
backup_2022_06_29_110002_0461767.trn	trn	83,456
backup_2022_06_29_100001_1014541.trn	trn	83,456
backup_2022_06_29_090001_2864015.trn	trn	83,456
backup_2022_06_29_080001_5238412.trn	trn	83,456
backup_2022_06_29_070001_5317590.trn	trn	83,456
backup_2022_06_29_060001_7658204.trn	trn	83,456
backup_2022_06_29_050001_4375604.trn	trn	148,992
backup_2022_06_29_041002_1461032.bak	bak	8,336,264,704
backup_2022_06_29_040001_4313123.trn	trn	83,456
backup_2022_06_29_030001_6514979.trn	trn	83,456
backup_2022_06_29_020001_5125844.trn	trn	1,023,444,480

Fig. 12. Backup files.

Table 3
Log record detection rate for each deleted transaction backup file.

Deleted Backup file	Detection rate(%)
1st file	0
2nd file	97.4 (974/1000)
3rd file	96.7 (967/1000)

The fact that transaction log records executed in 2020 were identified and reconstructed in the unallocated area of the server used until 2022 is a meaningful result. Additionally, the fact that the structure of the log record did not change from SQL Server 2014 to 2019 suggests that the method presented in this paper can be applied to transaction log records of databases from SQL Server 2014 to 2019.

6.4. Limitations

As a result of analyzing the extracted data, several limitations were identified.

- The study presented in this paper focuses on the identification and analysis of single log records, not the entire transaction log file. This means that the original data area of the parity bits, which should be accessed in units of log blocks, cannot be accessed. Therefore, reconstruction must be conducted in a state where the parity bits are overwritten, not the original data of the log record.
- If log records are fragmented according to the minimum unit stored in the file system, query reconstruction is difficult. For example, in the case of NTFS, files are stored in units of clusters. If a transaction log file is fragmented and a single log record is divided and stored in separate clusters, reconstruction is difficult.

These limitations are due to the fact that the method is designed to reconstruct log records that have been deleted from the file system. When a log record is deleted, it is not actually removed from the file system. Instead, the space occupied by the log record is marked as free. This means that the original data of the log record is still present in the file system, but it is not accessible. The method presented in this paper reconstructs log records by searching for the signatures of log records in the unallocated area of the file system. However, if the log records are fragmented, data fragments of the log records may be spread out across multiple clusters. This make it difficult to reconstruct the log records.

Despite these limitations, the method presented in this paper is a valuable tool for forensic investigators. It can be used to reconstruct log records that have been deleted from the file system, which can provide valuable insights into the activities that took place on the system.

7. Conclusion

Due to the increasing use of databases for storing corporate and private sensitive data, the importance of database forensic is increasing day by day. Information reconstruction, one of the key features of database forensics, is a technique that tracks the database from the time when an incident occurred to the current state. Information reconstruction uses various type of log data generated by the databases, such as transaction logs, error logs, system logs, and trace logs. In this paper, we analyze the transaction log of MSSQL databases. Among the many types of log records, we focus on INSERT, DELETE, and UPDATE queries. For these queries, there is a 1:1 correspondence between each query and a log record operation. However, in the case of table CREATE and DROP queries, there is no 1:1 correspondence between the query and a log record operation. Instead, these queries are performed by combining multiple log records within a single transaction. In other words, reconstructing CREATE and DROP table queries requires analyzing the entire

Begin Time	End Time	Operation	Query
11/13/2020 01:46:25.566667	11/13/2020 01:46:25.566667	Operation.LOP_DELETE_ROWS	delete from BindLog where IP='192.168.0.74' and SWCode=8102 and PrintPCSet=''
12/14/2020 13:43:02.410000	12/14/2020 13:43:02.410000	Operation.LOP_INSERT_ROWS	insert into BindLog values ('192.168.0.27',8102,'')
12/02/2020 16:51:43.230000	12/02/2020 16:51:43.230000	Operation.LOP_DELETE_ROWS	delete from BindLog where IP='192.168.0.27' and SWCode=8102 and PrintPCSet=''
12/02/2020 17:11:43.006667	12/02/2020 17:11:43.006667	Operation.LOP_INSERT_ROWS	insert into BindLog values ('192.168.0.12',8002,'B')
12/02/2020 17:33:27.373333	12/02/2020 17:33:27.373333	Operation.LOP_DELETE_ROWS	delete from BindLog where IP='192.168.0.12' and SWCode=8002 and PrintPCSet='B'
03/15/2022 12:53:20.946667	03/15/2022 12:53:20.946667	Operation.LOP_INSERT_ROWS	insert into BindLog values ('192.168.0.152',8102,'')

Fig. 13. Analysis result: real case.

transaction log, not a single log record. So, we will analyze transaction logs for CREATE and DROP table queries and conduct query reconstruction studies based on log records.

We have focused on MSSQL among various DBMSs, but there are many other DBMSs and new DBMSs are continuously being developed. All databases record transaction logs by default. Therefore, based on the method presented in this paper, we will study practical forensic methods on log data for various databases of the latest version.

Acknowledgements

This work was supported by Police-Lab 2.0 Program (www.kipot.or.kr) funded by the Ministry of Science and ICT (MSIT, Korea) & Korean National Police Agency (KNPA, Korea) [Project Name: Research on Data Acquisition and Analysis for Counter Anti-Forensics/Project Number: 210121M07].

References

Adedayo, O.M., Olivier, M.S., 2015. Ideal log setting for database forensics reconstruction. Digit. Invest. 12, 27–40. https://doi.org/10.1016/j.diin.2014.12.002.

CAST Hex value for Datetime - SQL server to MySQL migration - SQLines tools — sqlines.com. https://www.sqlines.com/sql-server-to-mysql/cast_datetime_hex. (Accessed 10 May 2023).

Choi, H., Lee, S., Jeong, D., 2021. Forensic recovery of sql server database: Practical approach. IEEE Access 9, 14564–14575. https://doi.org/10.1109/ACCESS.2021.3052505.

Chopade, R., Pachghare, V.K., 2019. Ten years of critical review on database forensics research. Digit. Invest. 29, 180–197.

DB-Engines Ranking — db-engines.com. https://db-engines.com/en/ranking. (Accessed 9 May 2023).

Delaney, K., Freeman, C., 2013. Microsoft SQL Server 2012 Internals: Micro SQL Serve 2012 Int_p1. Microsoft Press.

Fasan, O.M., Olivier, M., 2012a. Reconstruction in database forensics. In: Advances in Digital Forensics VIII: 8th IFIP WG 11.9 International Conference on Digital Forensics, vol. 8. Springer, Pretoria, South, pp. 273–287. Africa, January 3-5, 2012, Revised Selected Papers.

Fasan, O.M., Olivier, M.S., 2012b. On Dimensions of Reconstruction in Database Forensics.

Fixup - concept - NTFS Documentation — inform.pucp.edu.pe. http://inform.pucp.edu.pe/ inf232/Ntfs/ntfs_doc_v0.5/concepts/fixup.html. (Accessed 11 May 2023).

Foster, E.C., Godbole, S., Foster, E.C., Godbole, S., 2016. Overview of microsoft sql server, Database Systems. A Pragmatic Approach, pp. 461–467.

K. Fowler, G. Gold, Sql Server Database Forensics, Memory.

Frühwirt, P., Kieseberg, P., Schrittwieser, S., Huber, M., Weippl, E., 2013. Innodb database forensics: Enhanced reconstruction of data manipulation queries from redo logs. Inf. Secur. Tech. Rep. 17 (4), 227–238.

Li, A., 2021. Research on redo log record locating mechanism in sql server instance recovery. In: Journal of Physics: Conference Series, vol. 1952. IOP Publishing, 032056.

D. Litchfield, Oracle Forensics Part 1: Dissecting the Redo Logs, NGSSoftware Insight Security Research (NISR), Next Generation Security Software Ltd, Sutton.

OpenText Encase Forensic — opentext.Com. https://www.opentext.com/products/encase-forensic. (Accessed 11 May 2023).

Park, J., Lee, S., 2014. Data fragment forensics for embedded dvr systems. Digit. Invest. 11 (3), 187–200.

SQL Server transaction log architecture and management guide - SQL Server — learn.microsoft.com. https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide?view=sql-server-ver16. (Accessed 13 May 2023).

Sablatura, J., Zhou, B., 2017. Forensic database reconstruction. In: 2017 IEEE International Conference on Big Data (Big Data). IEEE, pp. 3700–3704.

Tripathi, S., Meshram, B.B., 2012. Digital evidence for database tamper detection. J. Inf. Secur. 3, 113–121. https://doi.org/10.4236/jis.2012.32014.

Wright, P.M., 2005. Oracle database forensics using logminer. In: June 2004 Conference. SANS Institute, pp. 1–39.

Yaseen, A.. SQL server transaction log architecture — sqlshack.com. https://www.sqlshack.com/sql-server-transaction-log-architecture/. (Accessed 9 May 2023).

Stellar Data Recovery. SQL log analyzer tool - open amp; read SQL server transaction log — stellarinfo.com. https://www.stellarinfo.com/mssql-log-analyzer.php. (Accessed 10 May 2023).

Sql Server Recovery Tool, ApexSQL. https://www.apexsql.com/sql-tools-recover.aspx (last accessed 24 December 2020).