IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 24, 2020, accepted January 4, 2021, date of publication January 18, 2021, date of current version January 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3052505

Forensic Recovery of SQL Server Database: Practical Approach

HOYONG CHOI[1], **SANGJIN LEE**[1], **AND DOOWON JEONG**[2]
[1]Institute of Cyber Security and Privacy (ICSP), Korea University, Seoul 02841, South Korea
[2]College of Police and Criminal Justice, Dongguk University, Seoul 04620, South Korea

Corresponding author: Doowon Jeong (doowon@dgu.ac.kr)

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government (MSIT) (Development of Digital Forensic Integration Platform) under Grant 2018-0-01000.

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, we 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, we 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, we also provide the source code of the implementation; it facilitates the knowledge sharing of database forensics.

INDEX TERMS Database, forensics, SQL server, MSSQL, servers.

I. INTRODUCTION

Database forensics is a branch of digital forensics that analyzes the structure of database and examines contents [1]. 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 [2]; 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 [3]–[6] and investigation model development [7]–[9]. 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 [10]. Especially, attackers try to the database tampering to take away sensitive information or try to delete the information [11]; by recovering deleted records, an investigator can detect and deal with these behaviors [12]. The recovery of deleted records is also used to solve the financial fraud cases in forensic accounting [13].

The techniques to recover deleted records are divided into two main categories: the log-based method and engine-based method [14]. 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

The associate editor coordinating the review of this manuscript and approving it for publication was Shaohua Wan.

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 [15]–[17] and EDB [18], [19], 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.

In this paper, we propose an engine-based recovery method to recover deleted data in MSSQL that is one of the most used DBMS globally [20]. We 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, we reconstruct all types of records including LOB. Lastly, we 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 paper is summarized as follows.

- We present the detailed internal structure of the MSSQL data file.
- We introduce MSSQL data recovery algorithm based on the storage engine. To the best of our knowledge, this paper is the first systematic study about MSSQL recovery.
- The proposed method recovers all deleted records on the unallocated area and it is compatible with various MSSQL versions.
- This paper provides a comparative performance evaluation of our method and well-known forensic tools developed to recover deleted data of MSSQL.
- We publish our implementation[1] that facilitates the knowledge sharing of database forensics.

The remainder of this paper is organized as follows: Related works are presented in Section II. In Section III, we introduce an internal structure of the MSSQL storage. Next, we describe several system tables used to recover deleted records in Section IV. In Section V and VI, we describe and evaluate our proposed method by experiment. Finally, we give the conclusion and the future work in Section VII.

II. RELATED WORK

A. 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.

Khanuja and Adane [21] proposed a framework for analyzing and reconstructing the activity of the suspect's behaviour. They described their framework in general digital forensic procedures: identification, collection, analysis, validation, interpretation, report, and preservation. Beyers *et al.* [22] focused on the data model of the DBMS and proposed a method to transform the data model into a clean state. Flores and Jhumka [23] 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.* [19] 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.* [7], [24] 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 [25]' and 'DBCarver [26]'. 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.* [27] 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 we enhance the knowledge of metadata and structure of LOB in this paper.

B. 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.

1) MSSQL

Fowler [28] proposed a method for investigating MSSQL. The method consists of 4 investigation phases: preparation,

[1] URL: https://github.com/hoyoi05/database

incident verification, artifact collection, and analysis. Khanuja and Adane [29] 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 [30] 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. We doesn't research transaction log but data file, so we 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.

2) ORACLE

Litchfield [3] demonstrated that an Oracle log file can be used to reveal attacker events. He also presented a forensic model to recover the dropped objects like tables and views [31]. In his continuous research of Oracle, he introduced a method to find and capture evidence [32]–[34] and inspired forensic researchers by providing investigation technique using Oracle artifacts [35], [36]. Tripathi and Meshram [4] conducted a study of tamper detection for Oracle. They examined Oracle artifacts such as redo logs, data blocks, etc. Finnigan *et al.* [37] introduced the practical method to Oracle using SQL query.

Similar to MSSQL forensics, Oracle forensics also has focused on the method with log files or DBMS functions. MSSQL and Oracle are generally supplied for large-scale business systems. Thus, their internal structures are complicated and mechanisms to store data are sophisticated; these characteristics make a forensic analysis without a log or DBMS service more difficult for forensic researchers.

3) MySQL AND SQLite

As MySQL and SQLite are based on open source projects, identifying the internal structure is relatively easier than identifying that of MSSQL; there are several studies to reconstruct database based on analyzing the file format of the databases. Frühwirt *et al.* [5], [38] presented InnoDB, the storage engine of MySQL. Frühwirt *et al.* also analyzed the structure of FRM file containing metadata of MySQL and proposed the method to reconstruct transaction history through analyzing redo log files. Noh *et al.* [39] studied the internal structure of MYI and MYD, which are data files of MyISAM storage engine of MySQL. They proposed a method to recover deleted records by analyzing MYI and MYD. Jung *et al.* [14] suggested a method to recover MySQL based on the ibdata file, which is a data file of the InnoDB storage engine of MySQL.

In the SQLite forensics field, Jeon *et al.* [40] studied data management rules used by SQLite and the structure of deleted data in the system. Li *et al.* [41] proposed a method to recover SQLite data based on WAL (Write-Ahead Logging) log. They tested the algorithm on SQLite files of mobile phones. Nemetz *et al.* [15] suggested the method to recover deleted as well as partially overwritten data based on the analysis of the internal structure of SQLite.

4) NoSQL

NoSQL forensics has been also researched. Yoon and Lee [10] and Yoon *et al.* [42] studied MongoDB, ranked 5th among overall databases, and ranked 1st among NoSQL category. They analyzed two storage engines MMAPv1 and WiredTiger of MongoDB and proposed a forensic framework for MongoDB. Xu *et al.* [43] developed the tool to extract the contents of deleted elements from Redis, a key-value database. They examined AOF (Append Only File) and RDB (Redis Database File); the AOF contains the execution of write operations and the RDB contains the actual data including deleted data. Kumbhare *et al.* [44] focused on tamper detection in MongoDB and CouchDB. They presented how to investigate the tampering in NoSQL data files. Golhar *et al.* [11] also proposed the method to detect NoSQL tampering. They explored log files of Redis and Cassandra to identify problems of databases such as data consistency, data integrity, and availability.

III. INTERNAL STRUCTURE OF THE MSSQL STORAGE

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 [45]. 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, we 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, we reverse-engineered MSSQL. Based on our findings, this section presents the internal structure of the data file and how MSSQL stores the data.

A. PAGE

The page is the smallest unit of MSSQL data file and its default size is 8,192 (0 × 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 0 × 00. 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).

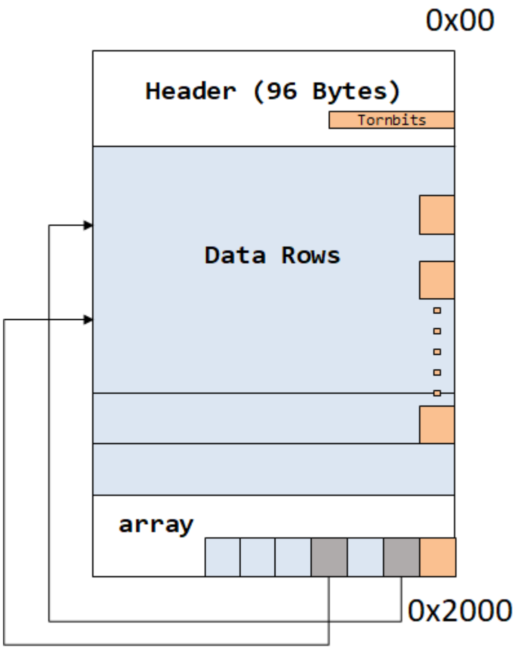


FIGURE 1. **Page structure (applied with Tornbits).**

The Type field indicates the type of the page. It is known that there are 14 types of the page [46]; this paper 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.

The Flag Bits field indicates how the Checksum field is used. If the Flag Bits value is 0 × 200, the Checksum field is used for validating the page. However, if the value is 0 × 100, the Tornbit is stored in the Checksum field. The Tornbit is described in Section III-D.

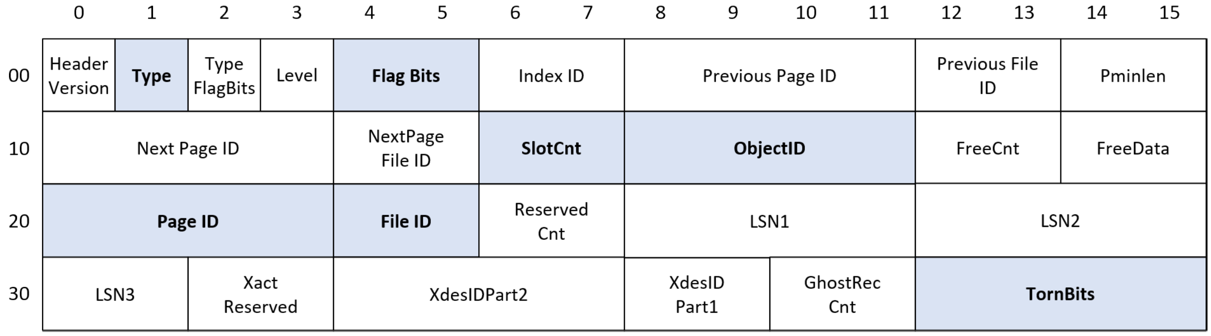
The SlotCnt field represents the number of normal records on the page. The Page Object ID field is used to store an identifier of the object such as a table, index, and so on. Therefore, by analyzing the Page Object ID, it is determined to which page the data of the table is allocated. The Page ID indicates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

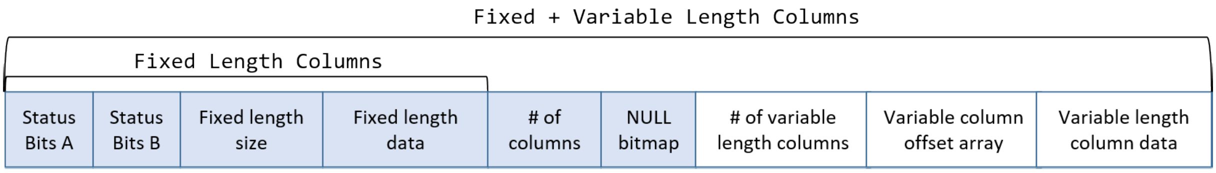
B. RECORD

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,



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	Header Version	**Type**	Type FlagBits	Level	**Flag Bits**		Index ID		Previous Page ID				Previous File ID		Pminlen	
10	Next Page ID				NextPage File ID		**SlotCnt**		**ObjectID**				FreeCnt		FreeData	
20	**Page ID**				**File ID**		Reserved Cnt		LSN1				LSN2			
30	LSN3		Xact Reserved		XdesIDPart2				XdesID Part1		GhostRec Cnt		**TornBits**			

FIGURE 2. **Page header structure.**



Fixed + Variable Length Columns								
Fixed Length Columns								
Status Bits A	Status Bits B	Fixed length size	Fixed length data	# of columns	NULL bitmap	# of variable length columns	Variable column offset array	Variable length column data

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

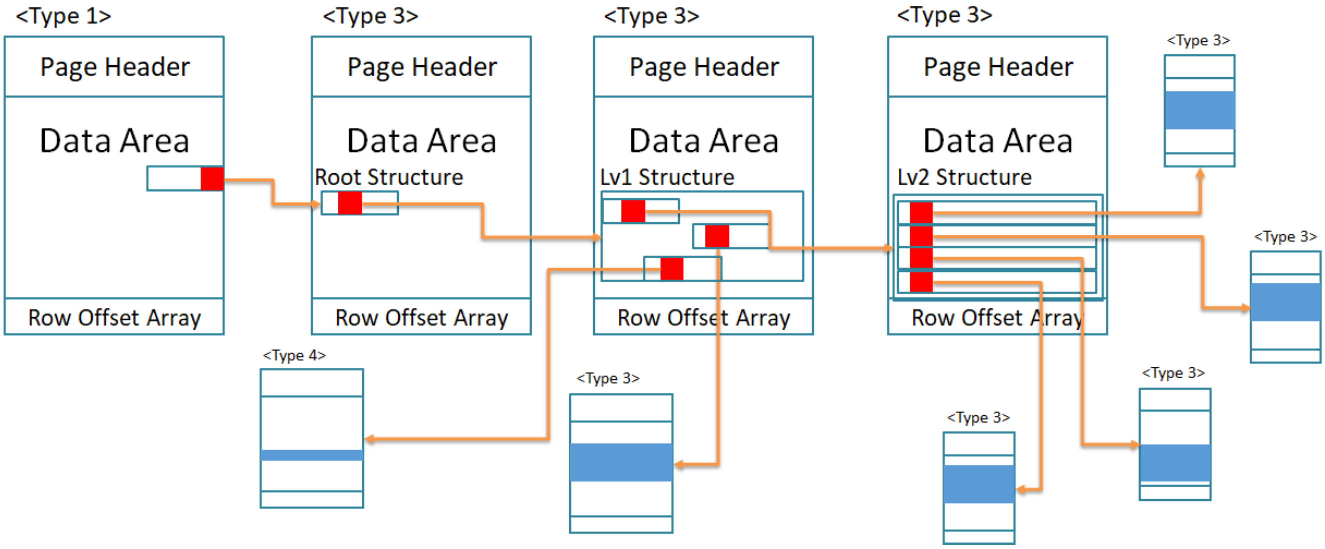


FIGURE 4. Large data structure (B-tree).

nvarchar, and varbinary are followed. The description of each field is as follows [47].

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.

C. 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*.

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.

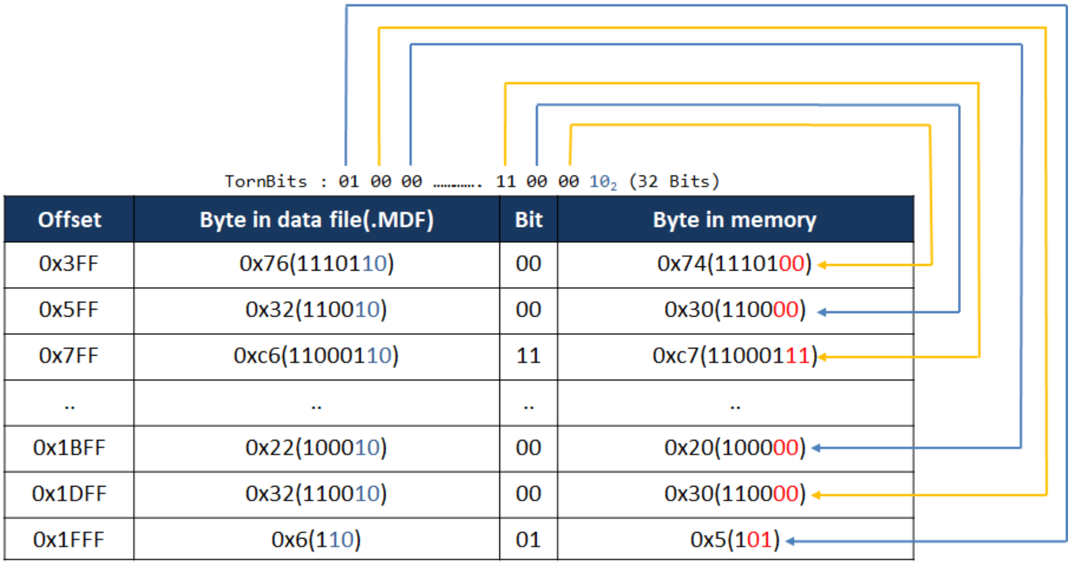
LOB Data called unrestricted-length large object data is used when the type is text, ntext, image, varchar, nvarchar, or varbinary. Since MSSQL 2005, the text, ntext, and image data type are deprecated and varchar, nvarchar, and varbinary data type with MAX specifier are substituted for them. We name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

D. TORNBITS

There are two ways to verify the page's integrity in MSSQL; One is *Checksum* and the other is *TornPageDetection*. 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 [48].

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



TornBits : 01 00 00 ……….. 11 00 00 10₂ (32 Bits)

Offset	Byte in data file(.MDF)	Bit	Byte in memory
0x3FF	0x76(1110110)	00	0x74(1110100)
0x5FF	0x32(110010)	00	0x30(110000)
0x7FF	0xc6(11000110)	11	0xc7(11000111)
..	..	..	..
0x1BFF	0x22(100010)	00	0x20(100000)
0x1DFF	0x32(110010)	00	0x30(110000)
0x1FFF	0x6(110)	01	0x5(101)

FIGURE 5. Tornbits example.

Fig. 5 shows an example of the Tornbits. As seen in the figure, the last two bits of bytes from 0 × 3FF to 0 × 1FFF at 0 × 200 interval are replaced 10_2, 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.

IV. MSSQL 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*.

We 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 V is independent of versions because system tables are analyzed before data in user tables are accessed. The schema structures of system tables are described in Appendix A.

A. 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.

B. 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.

C. 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 V-C.

TABLE 1. The order change according to primary key setting.

Key	Column Name	Data Type	Order
	num	int	1→3
	name	varchar(50)	2→4
*	student_num	char(10)	3→1
	course	varchar(50)	4→5
	project	varchar(50)	5→6
	email	nvarchar(50)	6→7
*	address	nvarchar(256)	7→2
	note	text	8→8

D. 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*.

E. SYSALLOCUNITS

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.

V. RECOVERY METHOD

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. We implement the proposed method in this paper as open-source tool. It can be downloaded from GitHub as mentioned in Section I.

A. SCANNING PAGES

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.

B. 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.

C. COLLECTING USER TABLE INFORMATION

In this phase, the information of the user table to be recovered is obtained by parsing the system table analyzed in Section V-B. This phase is further divided into four phases.

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.

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 datetime) are obtained in this phase. Based on the information, a temporary table is created to store the recovered records, by using *CREATE TABLE* query.

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).

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.

$$\begin{aligned} indexid &= allocUnitId \gg 48 \\ pageObjectId &= (allocUnitId - (indexid \ll 48)) \gg 16 \end{aligned} \quad (1)$$

To obtain the Page ID to which user table is allocated, the Page Object ID of each page obtained in Section V-A 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×page unit size(0 × 2000).

D. RECOVERING DELETED RECORDS

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

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

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

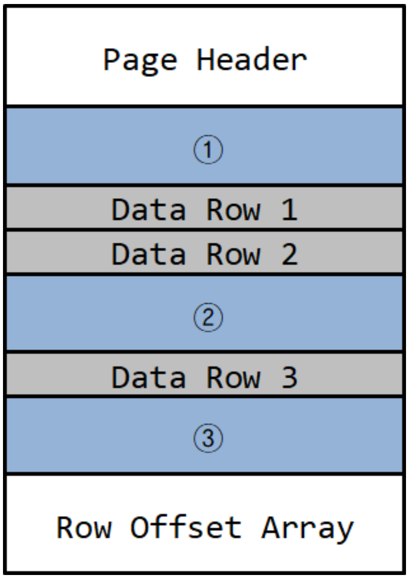


FIGURE 6. Cases of the unallocated area in a data page.

Algorithm 1 Reconstruct Deleted Records

```
Input: Unallocated area data
Output: Array of INSERT query
$L_{Unalloc} \leftarrow$ Unallocated area data size;
$S_{Schema} \leftarrow$ Table schema;                    /* from syscolpars and sysiscols */
$L_{Total} \leftarrow 0$ ;                                         /* Recovered record length */
Create empty array $Q$ ;                       /* Create empty array for storing INSERT query */
while $L_{Total} < L_{Unalloc}$ do
    $L_{Record} \leftarrow$ Deleted record length;
    if $L_{Record} + L_{Total} \leq L_{Unalloc}$ then
        Create empty array $D_{columns}$;             /* Create empty array for storing column data */
        $O_{Static} \leftarrow$ Start offset of fixed-length data area;
        $O_{Variable} \leftarrow$ Start offset of variable-length data area;
        foreach column information $C \in S_{Schema}$ do
            if $C.type$ is STATIC_COLUMN then
                Access static column data area using $O_{Static}$;
                $D \leftarrow$ get_ColumnData($O_{Static}, C.size$);
                $O_{Static} \leftarrow O_{Static} + C.size$;
            else
                /* if VARIABLE_COLUMN                                                          */
                Access variable column data area using $O_{Variable}$;
                Calculate $L_{Variable}$ from variable column offset array;
                $D \leftarrow$ get_ColumnData($O_{Variable}, L_{Variable}$);
                $O_{Variable} \leftarrow O_{Variable} + L_{Variable}$;
            end
            Decode $D$ according to $C.datatype$;
            Add $D$ to $D_{columns}$;
        end
        $L_{Total} \leftarrow L_{Total} + L_{Record}$;
        $q \leftarrow$ make_Query($D_{columns}$);
        Add $q$ to $Q$;
    else
        break;
    end
end
```

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

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. First, it calculates the size of the unallocated data area ($L_{Unalloc}$). Next, the length of the record to be recovered in the unallocated area (L_{Record}) is calculated. Then, to get the data of each column, the starting position of the divided fixed-length (O_{Ststic}) and variable-length column ($O_{Variable}$) 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 (S_{Schema}) 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 (L_{Record}) is stored in L_{Total} and compared to the size of the unallocated area ($L_{Unalloc}$). If the value L_{Total} is greater than or equal to the value $L_{Unalloc}$, recovery of the record in the input unallocated area is finished.

In this section, we 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.

VI. EXPERIMENT

While there is no article introducing the internal structure of the MSSQL storage, we assess the performance of the proposed method through comparative analysis with commercial tools, used in the practical forensic field, such as ApexSQL Recover [49], Stellar Repair for MSSQL [50], and SysTools SQL Recovery [51]. To identify the algorithms used in these commercial tools, reverse engineering of software protection should be conducted; it does not comply with the ethical research standards. We create data sets with various data types, security settings, and scenarios and then indirectly verify the tools whether they can deal with each element.

As there is no sample data set for evaluating the algorithm for MSSQL recovery, we created four data sets based on SQL Server 2017 that is the most widely used version. The data set A and A′, which consist of only *In Row Data* records, and the data set B and B′, which consist of both *In Row Data* and the large data records. As mentioned in Section III-D, there are two ways to verify the page's integrity in MSSQL; the data set A and B use the *Checksum* and the data set A′ and B′ use the *TornPageDetection*.

Based on the data sets, we evaluate recovery rates according to the following six cases.

- Case 1: Delete 1,000 records.
- Case 2: Delete 3,000 records.
- Case 3: Delete 5,000 records.
- Case 4: Insert 1,000 records after deleting 5,000 records.
- Case 5: Insert 3,000 records after deleting 5,000 records.
- Case 6: Insert 5,000 records after deleting 5,000 records.

In case 1∼3 the records were randomly deleted, and these cases are designed to check the difference in recovery rate as the number of deleted records increases. In case 4∼6, 1,000, 3,000, and 5,000 records were inserted into the data set created in case 3, and these cases were designed to check the difference in recovery rate according to the number of records inserted after deletion. In particular, the scenario, deleting records and then inserting new records, is constructed by referring to previous studies [10], [25] to verify the recovery rate of large data introduced in Section III-C. The large data is stored in dispersed pages, so metadata linking the pages may not be restored correctly when an inserted record overwrites part of a deleted records in the unallocated area. Therefore, by measuring the recovery rates in case 4∼6, we verify how well the tools handle issues related to the large data.

TABLE 2. Table schema of data sets (Note, the * mark means the primary key).

Data set	Column Name	Data Type
A / A′	no (*)	int
	name	nchar (255)
	path	nvarchar (1000)
	size	int
	mdate	datetime
B / B′	no (*)	int
	name	nchar (255)
	path	nvarchar (1000)
	size	int
	mdate	datetime
	contents	varbinary (MAX)

A. DATA SET

The schema of the table to be recovered is shown in Table 2. The number of records in the original table of data set A and A′ is 255,861 and the size of the data files is about 200 MB. The data set A and A′ are generated to evaluate whether the tools can recover deleted records with the `In Row Data`. On the other hand, the number of records in the data set B and B′ is 50,000 and the size of the data files is 13.8 GB. The schema of the data set B and B′ has an additional column, varbinary (MAX), to evaluate the performance of the large data recovery. To evaluate whether the tools can consider the *TornPageDetection*, we applied the Tornbits option to the data set A′ and B′.

In the digital forensics field, it is very important that evidence must be collected while maintaining the integrity of the evidence, so we applied a rigorous standard for recovery. We judged whether the recovery is successful by comparing the hash values of the recovered data and original data. If the data was partially restored, we determined that it was not restored correctly.

B. RESULT

As shown in Table 3, in case 1∼3 which are no insert operation after deletion, the recovery rate is 100% except for Stellar Repair for MSSQL. By verifying the non-recovered records when using Stellar Repair, it is identified that the Stellar Repair does not recover deleted records stored after page header, which is the first case of Fig. 6. We speculate that the reason lies in an implementation error when accessing the unallocated area on a data page.

TABLE 3. Recovery performance for data set A / A′ (%). For example, '70/0' means that the forensic tool recovers 70% of deleted records in data set A but it recovers 0% in data set A′.

Case #	Proposed Method	ApexSQL	Stellar Repair	SysTools
Case 1	100 / 100	100 / 100	89.3 / 0	100 / 100
Case 2	100 / 100	100 / 100	85.9 / 0	100 / 100
Case 3	100 / 100	100 / 100	85.4 / 0	100 / 100
Case 4	93.6 / 93.6	90.2 / 90.2	82.2 / 0	93.6 / 93.6
Case 5	81.5 / 81.5	75.0 / 75.0	75.8 / 0	81.5 / 81.5
Case 6	70.2 / 70.2	55.6 / 55.6	70.0 / 0	70.2 / 70.2

In case 4∼6, deleted records are recovered partially. It shows that the recovery rate decreases as the number of records inserted after deletion increases. This tendency is natural because it is more likely that the deleted data is overwritten as new data is inserted. We also observed that the difference in accuracy between the proposed method and ApexSQL increases as new records are inserted after the delete operation. It is estimated that ApexSQL skips pages that are partially overwritten when scanning data pages of the data file under certain conditions.

In particular, it shows that Stellar Repair does not recover any deleted record for data set A′. The same result is shown in data set B′ (See Table 4). This finding indicates that the Stellar Repair considers the *Checksum* only, not *TornPageDetection*.

TABLE 4. Recovery performance for Data set B / B′ (%).

Case #	Proposed Method	ApexSQL	Stellar Repair	SysTools
Case 1	100 / 100	100 / 100	45.0 / 0	99.7 / 99.7
Case 2	100 / 100	100 / 100	44.8 / 0	99.5 / 99.5
Case 3	100 / 100	100 / 100	44.8 / 0	99.2 / 99.2
Case 4	91.8 / 91.8	91.8 / 91.8	44.8 / 0	91.0 / 91.0
Case 5	86.2 / 86.2	86.2 / 86.2	44.6 / 0	85.4 / 85.4
Case 6	39.5 / 39.5	39.5 / 39.5	26.2 / 0	38.9 / 38.9

The experimental result for the data set B and B′ reports that the proposed method shows slightly better performance than SysTools, when recovering the large data records, as seen in Table 4. By analyzing the records that could not be recovered by SysTools, we identified that SysTools cannot recover the last part of some large data records. In digital forensics, the integrity of the evidence is a crucial aspect, so we regarded the partial recovery of the deleted record as a failure.

In conclusion, the proposed method shows a high recovery rate compared to other tools. The proposed method recovers deleted records in database applied with not only *Checksum* but also *TornPageDetection*. The implemented tool is designed to investigate MSSQL database in the practical forensic field, so it recovers the deleted large data records completely, including the last part of the records; the experimental results show better performance than other commercial tools.

C. DISCUSSION

Our experiment measured the recovery rate as a performance indicator. Although recovery speed is also important to forensic investigators, we did not measure the speed because our tool is CLI-based implementation while the commercial tools are GUI-based software. Ease of use may also be one of the performance indicators, but it was also not considered in our experiment because it comes down to a matter of preference.

When a record is deleted in MSSQL, the flag value of the metadata changes, but the record data remains intact. This is similar to the file system. Most file recovery algorithms use the method of analyzing the flag value of metadata to reconstruct the deleted data [52]. As the performance of file recovery tools differs in the process of finding and reconstructing deleted data, it is also observed that MSSQL record recovery tools also differ in performance.

TABLE 5. Syscolpars schema.

Column Name	Data Type	Length
id	int	4
number	smallint	2
colid	int	4
name	sysname	256
xtype	tinyint	1
utype	int	4
length	smallint	2
prec	tinyint	1
scale	tinyint	1
collationid	int	4
status	int	4
maxinrow	smallint	2
xmlns	int	4
dflt	int	4
chk	int	4
idtval	varbinary	64

TABLE 6. Sysschobjs schema.

Column Name	Data Type	Length
id	int	4
name	sysname	256
nsid	int	4
nsclass	tinyint	1
status	int	4
type	char	2
pid	int	4
pclass	tinyint	1
intprop	int	4
created	datetime	8
modified	datetime	8
status2	int	4

TABLE 7. Sysiscols schema.

Column Name	Data Type	Length
idmajor	int	4
idminor	int	4
subid	int	4
status	int	4
intprop	int	4
tinyprop1	tinyint	1
tinyprop2	tinyint	1
tinyprop3	tinyint	1
tinyprop4	tinyint	1

To recover deleted records from MSSQL, it is needed to understand how records are stored. Among various data types of MSSQL, a record of large data types store data on multiple pages. Furthermore, the large data is classified into three large data categories. Because each category has a different way of storing records, the details of navigating pages in which deleted data is stored in order are also different. The result of the experiment shows that some commercial tools are unable to fully navigate deleted records for some large data. On the other hand, our proposed method identified all deleted records, which was verified by comparing data sets and original data files that all records are live.

Tornbits used for the page's integrity must be considered to correctly interpret the page's information. The experimental result for one of the commercial tools shows that it may

TABLE 8. Sysrowsets schema.

Column Name	Data Type	Length
rowsetid	bigint	8
ownertype	tinyint	1
idmajor	int	4
idminor	int	4
numpart	int	4
status	int	4
fgidfs	smallint	2
rcrows	bigint	8
comprlevel	tinyint	1
fillfact	tinyint	1
maxnullbit	smallint	2
maxleaf	int	4
maxint	smallint	2
minleaf	smallint	2
minint	smallint	2
rsguid	varbinary	16
lockres	varbinary	8
scope_id	int	4

TABLE 9. Sysallocunits schema.

Column Name	Data Type	Length
auid	bigint	8
type	tinyint	1
ownerid	bigint	8
status	int	4
fgid	smallint	2
pgfirst	binary	6
pgroot	binary	6
pgfirstiam	binary	6
pcused	bigint	8
pcdata	bigint	8
pcreserved	bigint	8

not restore any deleted records without considering the *TornPageDetection* option.

VII. 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.

In this paper, we have researched MSSQL, which is the most used DBMS globally. We have described the internal structure of MSSQL including large data, tornbits. To recover deleted records regardless of the MSSQL version, several system tables have been analyzed. We 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 we have focused on one DBMS, MSSQL, 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 [13]. To overcome this circumstance, we will study practical analysis methods for various DBMSes based on the methods proposed in this paper.

APPENDIX A
SYSTEM TABLE SCHEMA

Each system tables' schema is shown in Table 5, 6, 7, 8, and 9 respectively.

REFERENCES

[1] M. S. Olivier, "On metadata context in database forensics," *Digit. Invest.*, vol. 5, nos. 3–4, pp. 115–123, Mar. 2009.

[2] M. Díaz, C. Martín, and B. Rubio, "State-of-the-art, challenges, and open issues in the integration of Internet of Things and cloud computing," *J. Netw. Comput. Appl.*, vol. 67, pp. 99–117, May 2016.

[3] 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.

[4] S. Tripathi and B. B. Meshram, "Digital evidence for database tamper detection," *J. Inf. Secur.*, vol. 3, no. 2, pp. 113–121, 2012.

[5] P. Frühwirt, P. Kieseberg, S. Schrittwieser, M. Huber, and E. Weippl, "InnoDB database forensics: Enhanced reconstruction of data manipulation queries from redo logs," *Inf. Secur. Tech. Rep.*, vol. 17, no. 4, pp. 227–238, May 2013.

[6] J. Sablatura and B. Zhou, "Forensic database reconstruction," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 3700–3704.

[7] A. Al-Dhaqm, S. A. Razak, S. H. Othman, A. Nagdi, and A. Ali, "A generic database forensic investigation process model," *Jurnal Teknologi*, vol. 78, nos. 6–11, pp. 45–57, Jun. 2016.

[8] 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.

[9] A. Al-Dhaqm, S. Abd Razak, D. A. Dampier, K.-K. R. Choo, K. Siddique, R. A. Ikuesan, A. Alqarni, and V. R. Kebande, "Categorization and organization of database forensic investigation processes," *IEEE Access*, vol. 8, pp. 112846–112858, 2020.

[10] J. Yoon and S. Lee, "A method and tool to recover data deleted from a MongoDB," *Digit. Invest.*, vol. 24, pp. 106–120, Mar. 2018.

[11] A. Golhar, S. Janvir, R. Chopade, and V. Pachghare, "Tamper detection in Cassandra and Redis database—A comparative study," in *Proc. Int. Conf. Comput. Sci. Appl.* Singapore: Springer, 2020, pp. 99–107.

[12] J. Wagner, A. Rasin, K. Heart, T. Malik, J. Furst, and J. Grier, "Detecting database file tampering through page carving," in *Proc. 21st Int. Conf. Extending Database Technol.*, 2018, pp. 1–12.

[13] R. Chopade and V. K. Pachghare, "Ten years of critical review on database forensics research," *Digit. Invest.*, vol. 29, pp. 180–197, Jun. 2019.

[14] S. K. Jung, J. W. Jang, D. W. Jeong, and S. J. Lee, "A study on the improvement method of deleted record recovery in MySQL InnoDB," *KIPS Trans. Comput. Commun. Syst.*, vol. 6, no. 12, pp. 487–496, 2017.

[15] S. Nemetz, S. Schmitt, and F. Freiling, "A standardized corpus for SQLite database forensics," *Digit. Invest.*, vol. 24, pp. S121–S130, Mar. 2018.

[16] C. Meng and H. Baier, "Bring2Lite: A structural concept and tool for forensic data analysis and recovery of deleted SQLite records," *Digit. Invest.*, vol. 29, pp. S31–S41, Jul. 2019.

[17] L. Zhang, S. Hao, and Q. Zhang, "Recovering SQLite data from fragmented flash pages," *Ann. Telecommun.*, vol. 74, nos. 7–8, pp. 451–460, Aug. 2019.

[18] J. Kim, A. Park, and S. Lee, "Recovery method of deleted records and tables from ESE database," *Digit. Invest.*, vol. 18, pp. S118–S124, Aug. 2016.

[19] R. Bria, A. Retnowardhani, and D. N. Utama, "Five stages of database forensic analysis: A systematic literature review," in *Proc. Int. Conf. Inf. Manage. Technol. (ICIMTech)*, Sep. 2018, pp. 246–250.

[20] *DB-Engines Ranking—Popularity Ranking of Database Management Systems*. Accessed: Dec. 24, 2020. [Online]. Available: https://db-engines.com/en/ranking

[21] H. K. Khanuja and D. Adane, "A framework for database forensic analysis," *Comput. Sci. Eng., Int. J.*, vol. 2, no. 3, pp. 27–41, 2012.

[22] H. Beyers, M. S. Olivier, and G. P. Hancke, "Arguments and methods for database data model forensics," in *Proc. WDFIA*, 2012, pp. 139–149.

[23] D. A. Flores and A. Jhumka, "Implementing chain of custody requirements in database audit records for forensic purposes," in *Proc. IEEE Trustcom/BigDataSE/ICESS*, Aug. 2017, pp. 675–682.

[24] A. Al-Dhaqm, S. Razak, S. H. Othman, K.-K. R. Choo, W. B. Glisson, A. Ali, and M. Abrar, "CDBFIP: Common database forensic investigation processes for Internet of Things," *IEEE Access*, vol. 5, pp. 24401–24416, 2017.

[25] J. Wagner, A. Rasin, B. Glavic, K. Heart, J. Furst, L. Bressan, and J. Grier, "Carving database storage to detect and trace security breaches," *Digit. Invest.*, vol. 22, pp. S127–S136, Aug. 2017.

[26] J. Wagner, A. Rasin, T. Malik, K. Heart, H. Jehle, and J. Grier, "Database forensic analysis with DBCarver," in *Proc. 8th Biennial Conf. Innov. Data Syst. Res.*, 2017, pp. 1–10.

[27] J. Wagner, A. Rasin, and J. Grier, "Database image content explorer: Carving data that does not officially exist," *Digit. Invest.*, vol. 18, pp. S97–S107, Aug. 2016.

[28] K. Fowler, *SQL Server Forenisc Analysis*. London, U.K.: Pearson, 2008.

[29] H. K. Khanuja and D. D. S. Adane, "Forensic analysis of databases by combining multiple evidences," *Int. J. Comput. Technol.*, vol. 7, no. 3, pp. 654–663, Jun. 2013.

[30] E. Toombs, "Microsoft SQL server forensic analysis," Ph.D. dissertation, Utica College, Utica, NY, USA, 2015.

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] P. Finnigan, P. Finnigan, and Gennick, *Oracle Incident Response and Forensics*. New York, NY, USA: Apress, 2018.

[38] P. Frühwirt, M. Huber, M. Mulazzani, and E. R. Weippl, "InnoDB database forensics," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Apr. 2010, pp. 1028–1036.

[39] W.-S. Noh, S.-M. Jang, C.-H. Kang, K.-M. Lee, and S.-J. Lee, "The method of deleted record recovery for MySQL MyISAM database," *J. Korea Inst. Inf. Secur. Cryptol.*, vol. 26, no. 1, pp. 125–134, Feb. 2016.

[40] S. Jeon, J. Bang, K. Byun, and S. Lee, "A recovery method of deleted record for SQLite database," *Pers. Ubiquitous Comput.*, vol. 16, no. 6, pp. 707–715, Aug. 2012.

[41] Q. Li, X. Hu, and H. Wu, "Database management strategy and recovery methods of android," in *Proc. IEEE 5th Int. Conf. Softw. Eng. Service Sci.*, Jun. 2014, pp. 727–730.

[42] J. Yoon, D. Jeong, C.-H. Kang, and S. Lee, "Forensic investigation framework for the document store NoSQL DBMS: MongoDB as a case study," *Digit. Invest.*, vol. 17, pp. 53–65, Jun. 2016.

[43] M. Xu, X. Xu, J. Xu, Y. Ren, H. Zhang, and N. Zheng, "A forensic analysis method for Redis database based on RDB and AOF file," *J. Comput.*, vol. 9, no. 11, pp. 2538–2544, Nov. 2014.

[44] R. Kumbhare, S. Nimbalkar, R. Chopade, and V. Pachghare, "Tamper detection in MongoDB and CouchDB database," in *Proc. Int. Conf. Comput. Sci. Appl.* Singapore: Springer, 2020, pp. 109–117.

[45] *Files and Filegroups Architecture*. Accessed: Dec. 24, 2020. [Online]. Available: https://technet.microsoft.com/en-us/library/ms179316(v=sql.105).aspx

[46] *John Huang's Blog*. Accessed: Dec. 24, 2020. [Online]. Available: http://www.sqlnotes.info/2011/10/31/page-type/

[47] *SQL Server Storage Engine: Data Pages and Data Rows*. Accessed: Dec. 24, 2020. [Online]. Available: http://aboutsqlserver.com/2013/10/15/sql-server-storage-engine-data-pages-and-data-rows/

[48] *Fixup—Concept—NTFs Documentation*. Accessed: Dec. 24, 2020. [Online]. Available: https://flatcap.org/linux-ntfs/ntfs/concepts/fixup.html

[49] ApexSQL. *SQL Server Recovery Tool*. Accessed: Dec. 24, 2020. [Online]. Available: https://www.apexsql.com/sql-tools-recover.aspx

[50] Stellar Data Recovery. *SQL Recovery Tool. Repair MDF, NDF Database Files*. Accessed: Dec. 24, 2020. [Online]. Available: https://www.stellarinfo.com/sql-recovery.php

[51] SysTools. *SQL Recovery Software to Repair MDF, NDF Database File*. Accessed: Dec. 24, 2020. [Online]. Available: https://www.systoolsgroup.com/sql-recovery.html

[52] G. Horsman, "Tool testing and reliability issues in the field of digital forensics," *Digit. Invest.*, vol. 28, pp. 163–175, Mar. 2019.

HOYONG CHOI received the B.S. degree from the Division of Computer and Communication Engineering, Korea University, in 2014. He is currently pursuing the Ph.D. degree with the Graduate School of Information Security, Korea University. His research interests include database forensics, artificial intelligence, and digital forensics.

SANGJIN LEE received the Ph.D. degree from the Department of Mathematics, Korea University, in 1994. From 1989 to 1999, he was with the Electronics and Telecommunications Research Institute, Korea, as a Senior Researcher. He has been with the Digital Forensic Research Center, Korea University, since 2008. He is currently the President of the Division of Information Security, Korea University. He has authored or coauthored over 130 papers in various archival journals and conference proceedings and over 200 articles in domestic journals. His research interests include digital forensics, data processing, forensic framework, and incident response.

DOOWON JEONG received the B.S. degree from the Division of Industrial Management Engineering, Korea University, in 2011, and the Ph.D. degree from the Graduate School of Information Security, Korea University, in 2019. He is currently an Assistant Professor with the College of Police and Criminal Justice, Dongguk University. His research interests include digital forensics, information security, artificial intelligence, and digital profiling.