

Article

Forensic Recoverability of Deleted Records Under Database Shrink in Microsoft SQL Server 2025: A Version-Comparative Experimental Study

Jiho Shin ¹  and Byoung Hun Moon ^{2,*}¹ Division of Computer Science Convergence, Mokwon University, Daejeon 35349, Republic of Korea; suchme@mokwon.ac.kr² Department of Forensics, Sungkyunkwan University, Seoul 03063, Republic of Korea

* Correspondence: mbh010@police.go.kr

Abstract

Databases serve as critical repositories of digital evidence in criminal investigations, and the recoverability of deleted data is a key determinant of forensic success. Microsoft SQL Server, one of the most widely deployed relational database management systems, has been the subject of multiple forensic studies examining how deleted records persist in physical database files across different acquisition methods. A previous study established a reference baseline using SQL Server 2008 and 2017, demonstrating that the Database Shrink operation causes version-specific and method-specific behavior: under logical collection with Shrink applied in SQL Server 2017, unallocated deleted data becomes fully initialized, rendering recovery impossible—a pattern not observed in SQL Server 2008 or under physical collection in either version. With the release of SQL Server 2025, the most significant architectural update to the platform in a decade, it remained unknown whether these forensic behaviors persist in the latest version. This study replicates the experimental design of in a controlled SQL Server 2025 environment, applying the same deletion scenario (DELETE command without conditions), the same two acquisition methods (logical and physical collection), and the same Shrink condition. The results demonstrate that SQL Server 2025 does not reproduce the version-specific initialization behavior observed in SQL Server 2017: across all four experimental conditions, deleted data residue in unallocated page space remains recoverable, indicating a fundamental change in the interaction between the Shrink operation and the logical collection mechanism. This recoverability is a double-edged property: while it benefits forensic investigators by preserving deleted evidence, it simultaneously represents a data-sanitization risk from a security and privacy standpoint, as deleted records are not reliably erased. These findings provide updated forensic guidance for digital investigators operating in contemporary SQL Server environments. Specifically, the results inform acquisition-method selection in real-world investigations where a suspect may have deleted records and where only a logical backup (.bak) is available to investigators.

Academic Editor:
Douglas O'Shaughnessy

Received: 5 May 2026

Revised: 9 June 2026

Accepted: 22 June 2026

Published: 26 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.**Keywords:** Microsoft SQL Server; database forensics; database shrink; deleted record recovery; digital investigation; logical collection; physical collection; page structure

1. Introduction

In digital forensic investigations, databases frequently serve as primary repositories of evidentiary information. The ability to recover deleted records from database systems

is of substantial importance, as suspects may deliberately delete data to conceal criminal activity. Among the various relational database management systems (RDBMSs) deployed in enterprise and public sector environments, Microsoft SQL Server is one of the most widely used platforms globally, making it a frequent target of forensic examination.

Previous research on SQL Server forensics has predominantly focused on transaction log-based recovery, whereby deleted records are reconstructed from the DML/DDL operation logs stored in the .ldf file [1]. While effective when logs are available, this approach fails when logs are absent, truncated, or when the suspect has employed log-minimizing deletion commands such as TRUNCATE. To address this limitation, direct inspection of physical database files (.mdf) for deleted data residue in unallocated page space has been proposed and experimentally validated [2–5].

Building on this foundation, Shin [3] experimentally confirmed in 2017 that all three principal deletion events in SQL Server—DELETE, TRUNCATE, and DROP—leave recoverable data residue in the unallocated space of data pages, regardless of the deletion method. Subsequently, Shin [2] extended this analysis in 2018 to compare the effect of the evidence acquisition method (logical vs. physical collection) on forensic recoverability across SQL Server 2008 and 2017, introducing the Database Shrink operation as an additional variable. The 2018 study identified a critical finding: in SQL Server 2017, performing a Shrink operation prior to logical collection causes the unallocated deleted data to be fully initialized, making recovery impossible—a behavior not observed in SQL Server 2008 or under physical collection in either version.

Among these three events, the present study focuses specifically on the unconditional DELETE command. DELETE is the most forensically subtle case: unlike TRUNCATE and DROP, which deallocate pages wholesale, DELETE removes rows individually while leaving page allocation intact, producing the finest-grained residue pattern in unallocated page space. It is also the deletion operation most commonly encountered in real-world evidence-tampering scenarios, where a perpetrator selectively removes incriminating records rather than discarding an entire table. Holding the deletion event constant as DELETE further preserves direct comparability with the 2018 baseline [2].

This version-specific behavioral difference raises a fundamental question for contemporary forensic practice: as SQL Server continues to evolve, does the forensic behavior of the Shrink operation remain consistent across major versions? SQL Server 2025, released in late 2025, represents the most architecturally significant update to the platform in approximately a decade, introducing native AI integration, new storage engine optimizations, and revised internal page management [6]. Whether these architectural changes affect the forensic behavior established in 2017 has not been examined.

This study addresses this gap by replicating the exact experimental design of [2] in a SQL Server 2025 environment and comparing the results against the previously established 2008 and 2017 baselines. Two competing research hypotheses are examined. Hypothesis 1 (H1) posits that the version-specific Shrink behavior identified in SQL Server 2017—wherein logical collection with Shrink causes unallocated deleted data to be initialized—persists in SQL Server 2025 due to continuity of the underlying storage engine architecture. Hypothesis 2 (H2) posits that architectural changes introduced in SQL Server 2025 alter this behavior, either eliminating or modifying the initialization effect. The contributions of this study are as follows:

- Empirically verify whether H1 or H2 is supported by experimental evidence across four controlled conditions;
- Provide a direct, controlled comparison between SQL Server 2008, 2017 [2], and 2025 under identical experimental conditions;

- Derive updated forensic guidelines for digital investigators operating in current SQL Server environments;
- Establish that a previously documented anti-forensic risk (Shrink-induced data destruction) no longer applies in the current SQL Server release, thereby revising—not merely replicating—the forensic guidance derived from prior work.

2. Related Works

2.1. Database Forensics and Deleted Record Recovery

The foundational work in transaction-based database recovery was established by Haerder and Reuter [1], who formalized the principles of transaction-oriented recovery and demonstrated that deleted records could be reconstructed from transaction logs. This log-centric paradigm has dominated subsequent research; however, it is predicated on the existence and availability of complete transaction logs—a condition that cannot be assumed in forensic contexts where suspects may have deliberately truncated or cleared logs.

To address this limitation, Page and Schwartz [7] proposed direct page-level inspection of database files as an alternative recovery strategy. Park [4] and Ryu [5] subsequently validated this approach specifically for Microsoft SQL Server, demonstrating that deleted records persist in unallocated page regions of the .mdf file and can be carved through direct binary inspection. Fowler [8] further systematized SQL Server forensic artifacts into categories including volatile data (Data Cache, Active VLFs), non-volatile data (Data Files, Authorization Catalogs), and non-resident data (System Event Logs), providing a structured framework for forensic examination.

Shin [3] extended this body of work by comparing the forensic implications of three distinct deletion events—DELETE, TRUNCATE, and DROP—across page allocation information, row offset arrays, and unallocated data residue, confirming that all three leave recoverable data in unallocated page space. Subsequent work by Shin [2] introduced the evidence acquisition method and Database Shrink as additional variables, revealing differential recovery outcomes across SQL Server 2008 and 2017. Choi et al. [9] proposed a practical framework for SQL Server forensic recovery at the raw data file level, demonstrating the recovery of deleted records from enterprise-scale databases without relying on transaction logs. Choi and Lee [10] extended this line of work by analyzing the internal structure of SQL Server transaction logs residing in unallocated areas of the file system, providing a method for reconstructing log-based evidence even when active logs have been purged. Shin [11] further advanced SQL Server forensic analysis by proposing a memory-based recovery procedure that combines transaction log analysis with real-time Buffer Pool inspection, enabling recovery of deleted records without taking the database offline. More recently, Shin and Moon [12] extended the forensic examination of SQL Server to cloud environments, demonstrating that the structural constraints of Azure SQL Database render conventional page-based and transaction log recovery techniques structurally inapplicable. At the framework level, Al-dhaqm et al. [13] developed a Database Forensic Metamodel (DBFM) to address interoperability and heterogeneity challenges across different database management systems and forensic investigation approaches, providing a structured conceptual foundation for the field.

2.2. Database Shrink: Operational Mechanics and Forensic Implications

Database Shrink is a maintenance operation built into Microsoft SQL Server that reduces the physical size of database files by relocating data pages from the end of the file to unoccupied space toward the beginning, and then releasing the freed tail space to the operating system [14]. The operation is triggered either manually via the DBCC

SHRINKDATABASE or DBCC SHRINKFILE command, or automatically through the AUTO_SHRINK database property.

From a forensic perspective, Shrink presents a dual concern. First, the page relocation process may overwrite unallocated space—where deleted records reside—with active data pages, thereby destroying forensic evidence. Second, the operation may interact with the acquisition method: the 2018 study [2] found that under logical collection in SQL Server 2017, Shrink caused unallocated deleted data to be fully initialized, whereas the same Shrink operation under physical collection left deleted data intact. This version- and method-dependent interaction represents the central empirical question of the present study. Ko et al. [15] noted that storage reclamation operations in RDBMS environments can interfere with traditional digital evidence recovery but did not empirically quantify the effect. Beyer and Brumbaugh [16] discussed anti-forensic database techniques in a broader context but did not isolate Shrink as a variable. The present study provides the first controlled, version-comparative experimental analysis of Shrink's forensic impact across a seventeen-year version span (2008 to 2025).

2.3. Digital Forensics in Evolving Database Environments

The challenge of maintaining forensic validity across evolving software versions is not unique to SQL Server. Prior work in adjacent domains—cloud storage forensics [17,18] and the broader digital-forensics literature [19,20]—has repeatedly stressed that forensic procedures validated on one platform generation cannot be assumed to hold on the next. Nelson et al. [21] make this explicit for software forensics: procedures validated on one version cannot be assumed to generalize to subsequent versions without empirical verification. The same caution applies within database forensics specifically, where Anglano et al. [22] showed that SQLite recovery procedures differ across engine versions. These observations motivate the version-specific, empirically grounded validation pursued in the present study.

3. Microsoft SQL Server Architecture

3.1. Database File Structure

Microsoft SQL Server stores all database content in two primary physical files: the data file (.mdf) and the log file (.ldf). The data file contains all user data, indexes, and metadata organized as a sequence of fixed-size logical units called pages. From a forensic standpoint, the data file is the primary target for deleted record recovery, as transaction logs may be unavailable or incomplete [2,3]. Data files are organized hierarchically into extents—units of eight contiguous 8 KB pages, each extent occupying 64 KB. Pages are catalogued in allocation maps (IAM, PFS, GAM, SGAM) that track whether each page is allocated or unallocated. When records are deleted via DELETE, the corresponding page entries in the row offset array are updated (zeroed), but the page itself remains allocated and the data remains in place [3,23].

3.2. Page Structure

Each 8192-byte SQL Server page consists of three regions: a 96-byte page header, a variable-length data row region, and a row offset array at the end of the page, as illustrated in Figure 1 [12]. The page header contains metadata including the page type, page ID, object ID, and the count of rows stored on the page. The row offset array stores 2-byte pointers indicating the starting offset of each row within the page. When a row is deleted via DELETE, its corresponding entry in the row offset array is set to 0 × 0000 (or overwritten with a prior offset value), while the actual row data remains in the data region as unallocated

residue [3,23]. This behavior—clearing the slot pointer while retaining the physical row bytes—is the structural basis for the deleted data recovery techniques examined in this study.

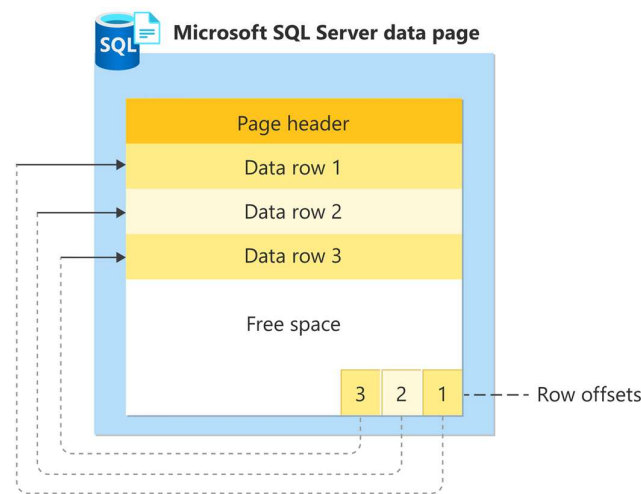


Figure 1. SQL Server data page structure showing Page Header, Data Rows, Free Space, and Row Offset Array (reproduced from Shin and Moon [12]).

3.3. Row Structure

Each data row stored within a page follows a structured binary layout consisting of a row header, a fixed-length data region, a column count field, a NULL bitmap, and a variable-length data region. Knowledge of the table schema enables an investigator to parse and reconstruct row content from raw binary page data even when the row has been logically deleted [3–5].

3.4. Database Shrink Mechanism

The Shrink operation operates by scanning the data file from the end and relocating any allocated pages found in the tail region to unoccupied space toward the beginning of the file. Once all tail pages have been relocated, the freed space at the end of the file is released to the operating system, reducing the physical file size [14]. This page relocation process has two forensically relevant consequences: (1) pages that contained deleted (unallocated) data residue in the tail region may be overwritten by relocated active pages, and (2) the tail region itself is truncated from the file, permanently removing whatever data remained there. The AUTO_SHRINK property is disabled by default in SQL Server installations. Manual Shrink is invoked via DBCC SHRINKDATABASE (catalog-level) or DBCC SHRINKFILE (file-level) [2,14].

4. Evidence Acquisition Methods in SQL Server Forensics

4.1. Logical Collection

Logical collection refers to the use of SQL Server’s built-in backup functionality (BACKUP DATABASE command or the Management Studio GUI) to create a .bak backup file. This method can be performed while the SQL Server service is running, without interrupting database operations. The backup file captures the logical state of the database as recognized by the SQL Server Engine, including all allocated pages and transaction logs. A key forensic characteristic of logical collection is that the resulting backup is processed by the SQL Server Engine itself, which may apply internal transformations—including page-level reorganization—that affect the residual content of unallocated regions [2].

4.2. Physical Collection

Physical collection involves directly copying the raw database files (.mdf and .ldf) from the underlying NTFS file system, bypassing the SQL Server Engine entirely [24]. Because the SQL Server service holds an exclusive lock on these files during operation, physical collection typically requires either stopping the SQL Server service or detaching the database catalog (DETACH operation) to release the file lock before copying. The resulting files are bit-identical copies of the original storage medium, preserving all page content including unallocated deleted data residue without Engine-level processing [2]. The critical forensic difference between the two methods lies in this Engine-level processing: logical collection subjects the backup content to SQL Server’s internal management logic, whereas physical collection captures the raw binary state of the file. This distinction becomes particularly relevant when Database Shrink has been performed, as Shrink may cause the Engine to initialize certain page regions during the logical backup process [2].

5. Experimental Design

This study replicates the experimental design established in [2] with the single modification of substituting SQL Server 2025 as the target version. The 2008 and 2017 results from [2] serve as the reference baseline. All other experimental parameters—deletion event type, acquisition methods, Shrink conditions, table schema, and record count—are held constant to enable direct version comparison. This deliberate single-variable design—isolating SQL Server version as the only changed factor—is what allows the observed behavioral difference to be attributed to the version change rather than to confounding parameters.

5.1. Experimental Environment

The experimental environment for the SQL Server 2025 experiments is configured as shown in Table 1.

Table 1. Experimental environment properties for SQL Server 2025.

Category	Value
Operating System	Microsoft Windows 11 (64-bit, x64-based)
Processor	11th Gen Intel Core i7-1165G7 @ 2.80 GHz
RAM	32.0 GB
Database Version	Microsoft SQL Server Express (64-bit) v17.0.1000.7
Auto Shrink	False (OFF)
Recovery Model	Full
File System	NTFS
Analysis Tool	SQL Server Management Studio (SSMS) v22.5.2+175.11723.231

5.2. Experimental Procedure

The experimental procedure follows the protocol established in [2]. A database named ForensicTestDB is created in SQL Server 2025, and a test table named CustomerInfo is created with the following schema, which is functionally identical to the schema used in [2]:

```
USE ForensicTestDB;
GO
CREATE TABLE CustomerInfo (
  CUSTOMER_ID INT IDENTITY(1,1) PRIMARY KEY,
  NAME NVARCHAR(100),
  PHONE VARCHAR(20),
```

```
EMAIL VARCHAR(100),
VIP_STATUS BIT
);
```

The schema comprises one INTEGER-type primary key column (CUSTOMER_ID), one NVARCHAR-type column (NAME), two VARCHAR-type columns (PHONE, EMAIL), and one BIT-type column (VIP_STATUS). Twenty arbitrary English-name records are inserted into the table, matching the record count used in [2]. The complete INSERT statement used in the experiment is as follows:

```
INSERT INTO CustomerInfo (NAME, PHONE, EMAIL, VIP_STATUS) VALUES
(N'James Carter', '010-1234-5678', 'james.carter@gmail.com', 1),
(N'Emily Johnson', '010-2345-6789', 'emily.johnson@naver.com', 0),
(N'Michael Brown', '010-3456-7890', 'michael.brown@daum.net', 1),
(N'Sophia Davis', '010-4567-8901', 'sophia.davis@gmail.com', 0),
(N'William Wilson', '010-5678-9012', 'william.wilson@naver.com', 1),
(N'Olivia Martinez', '010-6789-0123', 'olivia.martinez@outlook.com', 0),
(N'Benjamin Taylor', '010-7890-1234', 'benjamin.taylor@gmail.com', 0),
(N'Isabella Anderson', '010-8901-2345', 'isabella.anderson@daum.net', 1),
(N'Ethan Thomas', '010-9012-3456', 'ethan.thomas@naver.com', 0),
(N'Mia Jackson', '010-0123-4567', 'mia.jackson@gmail.com', 1),
(N'Alexander White', '010-1357-2468', 'alex.white@naver.com', 0),
(N'Charlotte Harris', '010-2468-1357', 'charlotte.harris@gmail.com', 1),
(N'Daniel Martin', '010-3579-4680', 'daniel.martin@daum.net', 0),
(N'Amelia Thompson', '010-4680-5791', 'amelia.thompson@outlook.com', 1),
(N'Henry Garcia', '010-5791-6802', 'henry.garcia@gmail.com', 0),
(N'Harper Robinson', '010-6802-7913', 'harper.robinson@naver.com', 0),
(N'Sebastian Clark', '010-7913-8024', 'sebastian.clark@gmail.com', 1),
(N'Evelyn Lewis', '010-8024-9135', 'evelyn.lewis@daum.net', 0),
(N'Jack Walker', '010-9135-0246', 'jack.walker@naver.com', 1),
(N'Luna Hall', '010-0246-1357', 'luna.hall@gmail.com', 0);
```

For each experimental condition, the database is restored to its pre-deletion state via Attach/Detach of a preserved copy, and the DELETE command is executed without conditions to remove all rows:

```
DELETE FROM CustomerInfo;
```

Where applicable, the Shrink condition is applied manually using the following command immediately after deletion and before acquisition:

```
DBCC SHRINKDATABASE('ForensicTestDB', 0); -- [25]
```

The resulting database file is then analyzed for (a) page allocation information, (b) row offset array content, and (c) the presence of unallocated deleted row data. Page allocation information is obtained via:

```
DBCC IND('ForensicTestDB', 'CustomerInfo', -1);
```

Page internal structure is analyzed using hexadecimal dump mode (output format 2), which presents the complete page binary as a continuous byte stream suitable for forensic inspection of deleted data residue:

```
DBCC TRACEON(3604);
DBCC PAGE('ForensicTestDB', 1, 544, 2);
```

Output format 2 was selected because it presents the entire page as a contiguous hexadecimal dump, enabling direct visual inspection of unallocated regions. Format 3, which parses and displays each row individually, does not expose unallocated content that the SQL Server Engine no longer recognizes as valid rows. The data page identifier (PagePID 544) was confirmed via DBCC IND prior to analysis.

5.3. Experimental Conditions

The four experimental conditions applied to SQL Server 2025 are defined by the combination of acquisition method and Shrink application (Table 2). Figure 2 presents the complete experimental procedure as a branching flowchart, illustrating the shared pre-experiment setup, the DELETE execution, the Shrink decision branch, the collection method branch, and the common analysis phase applied to all four conditions.

Table 2. Four experimental conditions for SQL Server 2025.

Condition	Collection Method	Shrink	Acquired File Type
Condition 1	Logical	Non-shrink	.bak (non-shrink.bak)
Condition 2	Physical	Non-shrink	.mdf + .ldf (ForensicTestDB.mdf/_log.ldf)
Condition 3	Logical	Shrink	.bak (shrink.bak)
Condition 4	Physical	Shrink	.mdf + .ldf (ForensicTestDB.mdf/_log.ldf)

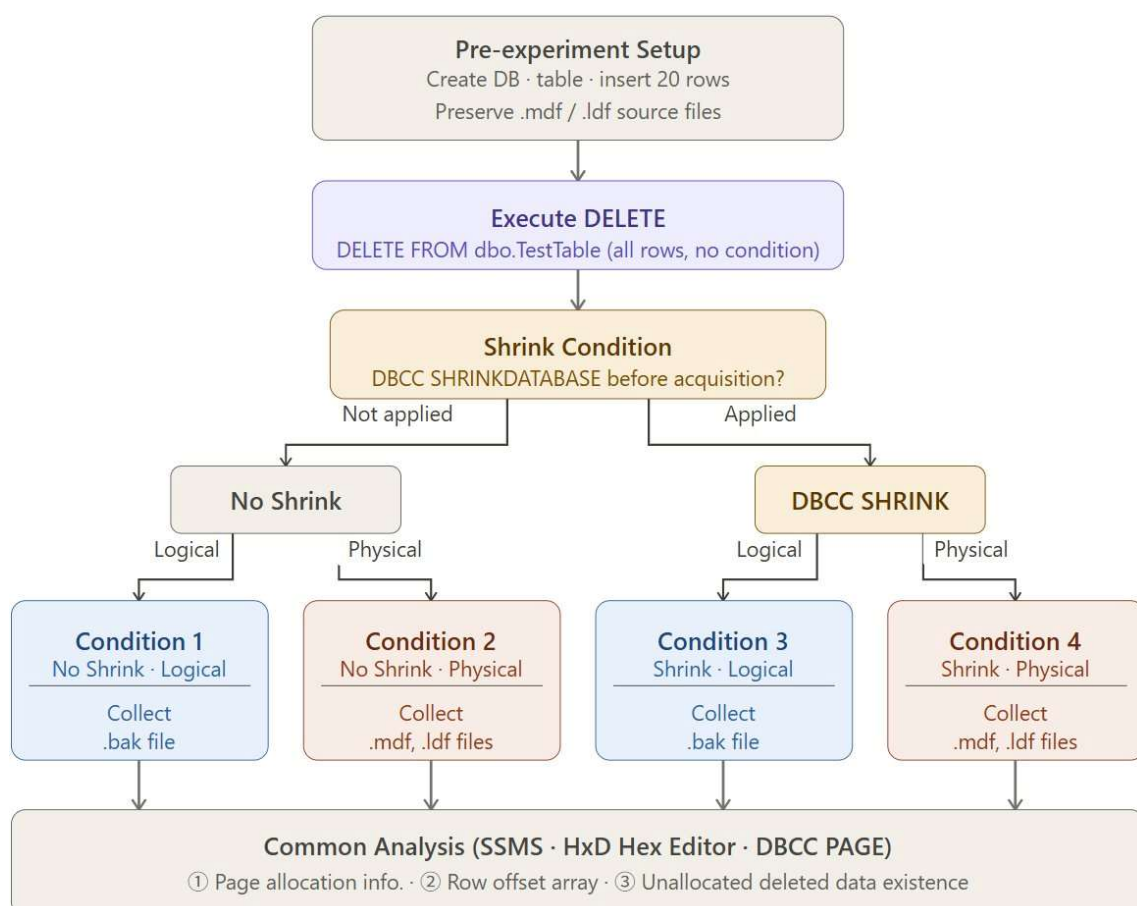


Figure 2. Experimental design flowchart for SQL Server 2025 experiments.

The full per-condition protocol is as follows: (1) attach a preserved pre-deletion copy of ForensicTestDB; (2) verify the 20-record baseline and confirm the target page (PagePID 544) via DBCC IND; (3) execute DELETE FROM CustomerInfo with no WHERE clause; (4) for Shrink conditions only, execute DBCC SHRINKDATABASE('ForensicTestDB', 0) immediately

after deletion; (5) acquire evidence by the assigned method—BACKUP DATABASE to a .bak file (logical) or a raw file-system copy of the detached .mdf/.ldf (physical); (6) record file name, size, and hash; (7) analyze via DBCC IND for allocation state and DBCC PAGE(...,2) in hexadecimal-dump mode for unallocated residue. All commands were issued under the Full recovery model on the environment specified in Table 1.

5.4. Acquired Evidence Files

To ensure the integrity and reproducibility of the experimental evidence, all acquired database files were recorded with their file names, sizes, and SHA-512 hash values immediately following acquisition. Table 3 presents the complete specification of the six evidence files collected across the four experimental conditions. File integrity was verified by recomputing hash values prior to analysis.

Table 3. Acquired evidence file specifications for SQL Server 2025 experiments.

Acquisition Method	Shrink	File Name	File Size (Bytes)	Hash (SHA-512)
Logical	Non-shrink	non-shrink.bak	4,902,912	520E82C8796C462D17F3542715D3C02917F0DF1150B60EB737C624950E8DA40BEC61F64FB7103C3C6B234B4DF801833D11045E1F7260530B1F01670ED0729E48
Logical	Shrink	shrink.bak	4,902,912	79CC3D34488B488D850BD026553FEF37D20003CC4204B1F2D75ED2AF98D04D8C54FAC6F9CDD90664C3BF1C387F1A8C7A0EB330F81D170FD8AFE1CD619885AC03
Physical	Non-shrink	ForensicTestDB.mdf	8,388,608	8052FA6F9F3D8B7EA4DE80F1229D788B3887674400C9A3756930C7314AD7E357AE450D4E03BBBEBF2225D24FA0420DC3A7DC9BEBFF7004D71AFE4C9C59D90E9F
Physical	Non-shrink	ForensicTestDB_log.ldf	8,388,608	9D2DD708D7BBA0474354464C2EADF17D5E800107E5D799C18C6F9B4D52289E71893BB558337130B92349B49E47821CC6D8D8434395AFDFE186B944A67783F3B9
Physical	Shrink	ForensicTestDB.mdf	8,388,608	C26760A28D0036E193E54EA706FEE590697587E65045504CF229FD87242332A5DA3836425788499030D351D73649D2AC70D2E472829104402E87D1EFC3A39DCB3
Physical	Shrink	ForensicTestDB_log.ldf	8,388,608	D74BBD19F9A78F5E722385DD4DC6E65FD533FE91F8C026EC8A223F4982E39AF097C234659510BF5EB683F5C669C241EC34C47B1A8708BBAF301F43AF235A8A3C

5.5. Experimental Results Summary

The full set of experimental conditions and results across all three SQL Server versions is presented in Table 4. Conditions for SQL Server 2008 and 2017 (Rows 1–8) are drawn from [2]; the four SQL Server 2025 conditions (Rows 9–12) represent the results of the present study.

Table 4. Experimental conditions and results across SQL Server 2008, 2017, and 2025. Rows 1–8: results from [2]. Rows 9–12 (shaded): results of the present study.

Ver.	Collection Method	Shrink	Page Allocation Info.	Row Offset Info. in Page	Unallocated Deleted Data	Recovery Possibility
2008 [2]	Logical	Non-shrink	Maintained	Initialized	Exist	Possible
	Logical	Shrink	Initialized	Initialized	Exist	Possible
	Physical	Non-shrink	Maintained	Initialized	Exist	Possible
	Physical	Shrink	Initialized	Initialized	Exist	Possible
2017 [2]	Logical	Non-shrink	Maintained	Initialized	Exist	Possible
	Logical	Shrink	Initialized	Initialized	Initialized	Impossible
	Physical	Non-shrink	Maintained	Initialized	Exist	Possible
	Physical	Shrink	Initialized	Initialized	Exist	Possible
2025	Logical	Non-shrink	Maintained	Initialized	Exist	Possible
	Logical	Shrink	Maintained	Initialized	Exist	Possible
	Physical	Non-shrink	Maintained	Initialized	Exist	Possible
	Physical	Shrink	Maintained	Initialized	Exist	Possible

6. Experimental Results

6.1. Baseline Confirmation: SQL Server 2008 and 2017 Results

The results for SQL Server 2008 and 2017 are drawn directly from [2] and serve as the experimental baseline. In SQL Server 2008, deleted data residue was preserved across all four experimental conditions—regardless of collection method or Shrink application. While Shrink caused page allocation information to be initialized in both versions, the unallocated deleted data itself remained intact in SQL Server 2008 under all conditions, yielding a recovery possibility of Possible in every case. In SQL Server 2017, the behavior diverged under one specific condition: Logical Collection with Shrink applied. Under this condition, the unallocated deleted data was fully initialized, rendering recovery impossible. Under all other 2017 conditions, deleted data residue was preserved and recovery remained possible. This version-specific behavioral difference constitutes the primary finding of [2] and the baseline against which the present study is compared.

6.2. SQL Server 2025 Results

6.2.1. Logical Collection Without Shrink (Condition 1)

The acquired file for this condition is non-shrink.bak (4,902,912 bytes; SHA-512 prefix: 520E82C8...). Following DELETE and logical collection without Shrink in SQL Server 2025, page allocation information was maintained, the row offset array was initialized, and unallocated deleted data was found to exist in the acquired file. Recovery was confirmed as possible. This result is consistent with the corresponding 2008 and 2017 baseline conditions.

6.2.2. Logical Collection with Shrink (Condition 3)

The acquired file for this condition is shrink.bak (4,902,912 bytes; SHA-512 prefix: 79CC3D34...). This condition represents the most forensically critical scenario, as it was the condition under which SQL Server 2017 exhibited the initialization of unallocated deleted data. Following DELETE, manual execution of DBCC SHRINKDATABASE, and logical collection in SQL Server 2025, the results were as follows: page allocation information was maintained (not initialized), the row offset array was initialized, and unallocated deleted data was found to exist in the acquired file. Recovery was confirmed as possible.

This result constitutes the primary finding of the present study. Unlike SQL Server 2017, SQL Server 2025 does not initialize unallocated deleted data under logical collection with Shrink. Furthermore, page allocation information was maintained rather than initialized—diverging from the 2017 baseline and aligning more closely with SQL Server 2008 behavior. Notably, DBCC SHRINKDATABASE reported that the file could not be shrunk due to insufficient free space, indicating that no physical page relocation occurred. Despite this, the forensic behavior under logical collection differed from 2017, suggesting that the initialization effect observed in 2017 reflects a version-specific Engine-level behavior during backup processing rather than a consequence of physical page relocation.

6.2.3. Physical Collection Without Shrink (Condition 2)

The acquired files for this condition are ForensicTestDB.mdf (8,388,608 bytes; SHA-512 prefix: 8052FA6F...) and ForensicTestDB_log.ldf (8,388,608 bytes; SHA-512 prefix: 9D2DD708...). Page allocation information was maintained, the row offset array was initialized, and unallocated deleted data was found to exist. Recovery was confirmed as possible. This result is consistent with the corresponding 2008 and 2017 baseline conditions.

6.2.4. Physical Collection with Shrink (Condition 4)

The acquired files for this condition are ForensicTestDB.mdf (8,388,608 bytes; SHA-512 prefix: C26760A2...) and ForensicTestDB_log.ldf (8,388,608 bytes; SHA-512 prefix:

D74BBD19...). Page allocation information was maintained, the row offset array was initialized, and unallocated deleted data was found to exist. Recovery was confirmed as possible. This result confirms that physical collection continues to preserve deleted data residue regardless of Shrink application across all three tested versions.

7. Discussion

The experimental results support H2 over H1: the version-specific Shrink behavior identified in SQL Server 2017 does not persist in SQL Server 2025. As defined in Section 1, H1 predicted continuity of the initialization effect under Logical Collection with Shrink, while H2 predicted that architectural changes in SQL Server 2025 would alter this behavior. The four experimental conditions collectively demonstrate that SQL Server 2025 does not initialize unallocated deleted data under any tested combination of collection method and Shrink condition. Its behavior is instead consistent with the SQL Server 2008 pattern, in which all conditions yielded recoverable deleted data. Methodologically, this work is a controlled, reproducible behavioral-validation study: because the forensic behavior in question is deterministic rather than stochastic, the appropriate evidentiary standard is exact, hash-verified reproducibility across controlled conditions rather than statistical hypothesis testing over a sampled population.

This finding has two significant forensic implications. First, digital investigators operating in SQL Server 2025 environments can expect deleted data residue to be preserved in unallocated page space even after Shrink has been applied, under both physical and logical collection methods. Second, the forensic procedure validated in [2]—specifically, the recommendation to prioritize physical collection when Shrink activity is detected—remains conservative and valid advice, but is no longer strictly necessary in SQL Server 2025 environments based on the present experimental evidence.

To illustrate the practical relevance of these findings, consider a representative investigative scenario. A suspect, seeking to conceal incriminating transactions, issues a DELETE against the relevant table and subsequently runs DBCC SHRINKDATABASE in an attempt to destroy the residual data, then surrenders only a logical backup (.bak) to investigators while withholding access to the live server files. Under SQL Server 2017, this exact sequence would have rendered the deleted records unrecoverable from the .bak. The present results show that in SQL Server 2025 the same sequence leaves the deleted records recoverable from the logical backup, materially changing the evidentiary value of a suspect-supplied .bak file. This is precisely the decision point at which an investigator must choose between logical and physical acquisition, and the version-dependent behavior documented here directly informs that choice.

Two findings can be stated with confidence from the page-level evidence. First, across all four conditions in SQL Server 2025, unallocated deleted data persisted and was recoverable. Second, DBCC SHRINKDATABASE reported insufficient free space and performed no physical page relocation, yet the logical-collection outcome still differed from that of SQL Server 2017. These are directly observed experimental facts.

The mechanism responsible for the divergence from SQL Server 2017 cannot be established from page-level analysis alone and is therefore offered as a hypothesis rather than a confirmed cause. Because the differing outcome arose without any physical page relocation, the 2017 initialization effect appears to have been coupled to Engine-level backup-processing logic rather than to physical page movement; the architectural revisions to the backup engine introduced in SQL Server 2025 are a plausible—but not verified—source of the change. Confirming this would require instrumentation of the backup engine's internal I/O path, which is beyond the scope of page-level forensic analysis. From an anti-forensic standpoint, the practical consequence is unambiguous regardless of mechanism:

Shrink is no longer an effective evidence-destruction tool against logical collection in SQL Server 2025.

8. Limitations

This study replicates the scope and constraints of [2] to enable direct comparison, and therefore shares certain limitations. The experiments are conducted on a single hardware and operating system configuration and do not assess variations across file systems (e.g., ReFS) or operating system versions. The deletion scenario is limited to the DELETE command applied to all rows without conditions; partial deletion and UPDATE-based data modification scenarios are outside scope. Moreover, acquisition in this study occurs immediately after deletion (and after Shrink, where applicable); the experiments therefore characterize a best-case boundary condition for recoverability. In practice, additional transactions, checkpoints, or page reuse occurring between deletion and acquisition may progressively overwrite the residual data, and the persistence of recoverability under such intervening activity is left to future work.

The version comparison spans SQL Server 2008 and 2017 (via [2]) and 2025, without intermediate versions (2019, 2022). While this scope was chosen for practical reasons—reinstallation of discontinued versions poses significant operational challenges—it means that the precise version at which the behavioral change occurred cannot be determined. Future work should include SQL Server 2019 and 2022 in the comparison set. Finally, the experimental table schema is relatively simple; the behavior of Shrink and acquisition methods in the context of more complex configurations—including BLOB fields, XML columns, row-compressed and page-compressed tables, partial (conditional) deletions, and the TRUNCATE and DROP events—may differ and warrants separate investigation. Extending the present single-variable design to these scenarios is the principal direction for future work.

9. Conclusions

This study replicates the experimental design established in [2] within a SQL Server 2025 environment to determine whether the version-specific forensic behavior of the Database Shrink operation, identified in SQL Server 2017, persists in the latest major SQL Server release. The study provides a controlled cross-version comparison spanning SQL Server 2008, 2017, and 2025.

The 2018 baseline [2] established that logical collection with Shrink in SQL Server 2017 results in the initialization of unallocated deleted data, rendering recovery impossible, while SQL Server 2008 preserved deleted data under all conditions. Testing H1 and H2 as defined in Section 1, the present study finds that H2 is supported: SQL Server 2025 does not reproduce the 2017 behavior. Across all four experimental conditions, unallocated deleted data residue remains present and recoverable. This indicates that the forensic behavior of SQL Server 2025 has reverted to a pattern consistent with SQL Server 2008, and that the initialization effect observed in 2017 was a version-specific implementation characteristic that has since been removed or modified. The practical implication for forensic investigators is that deleted data recovery from SQL Server 2025 environments is feasible across both logical and physical collection methods, and that Shrink no longer functions as an effective anti-forensic tool against logical collection in this version.

Author Contributions: Conceptualization, J.S.; methodology, J.S.; software, J.S.; validation, J.S.; formal analysis, B.H.M.; investigation, J.S.; resources, J.S.; data curation, B.H.M.; writing—original draft preparation, J.S.; writing—review and editing, B.H.M.; visualization, J.S.; supervision, J.S.; project administration, J.S.; funding acquisition, B.H.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Development of Integrated Analysis and Inference System for Cybercrime Investigation (R&D) Program through the Korea Institutes of Police Technology (KIPoT) funded by the Korean National Police Agency (No.RS-2025-02218280).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed at the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Haerder, T.; Reuter, A. Principles of Transaction-Oriented Database Recovery. *ACM Comput. Surv.* **1983**, *15*, 287–317. [CrossRef]
- Shin, J. Comparison of Recoverability of Deleted Data According to Collection Methods of Microsoft SQL Server. *J. Korea Inst. Inf. Secur. Cryptol.* **2018**, *28*, 859–868. [CrossRef]
- Shin, J. Comparison of Remaining Data According to Deletion Events on Microsoft SQL Server. *J. Korea Inst. Inf. Secur. Cryptol.* **2017**, *27*, 223–232. [CrossRef]
- Park, S.Y. A Research for Record Recovery Method in Database. Master's Thesis, Korea University, Seoul, Republic of Korea, December 2013.
- Ryu, G.H. A Study for Recovering Records of Microsoft SQL Server's Database. Master's Thesis, Korea University, Seoul, Republic of Korea, December 2014.
- Microsoft. What's New in SQL Server 2025. Available online: <https://learn.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-2025> (accessed on 1 January 2026).
- Page, D.; Schwartz, B. *High Performance MySQL: Optimization, Backups, Replication, and More*, 2nd ed.; O'Reilly Media: Sebastopol, CA, USA, 2008.
- Fowler, K. *SQL Server Forensic Analysis*; Addison-Wesley Professional: Boston, MA, USA, 2008.
- Choi, H.; Lee, S.; Jeong, D. Forensic Recovery of SQL Server Database: Practical Approach. *IEEE Access* **2021**, *9*, 14564–14575. [CrossRef]
- Choi, H.; Lee, S. Forensic Analysis of SQL Server Transaction Log in Unallocated Area of File System. *Forensic Sci. Int. Digit. Investig.* **2023**, *46*, 301605. [CrossRef]
- Shin, J. Memory-Driven Forensic Analysis of SQL Server: A Buffer Pool and Page Inspection Approach. *Sensors* **2025**, *25*, 3512. [CrossRef]
- Shin, J.; Moon, B.H. Cloud Database Forensics in Practice: Structural Challenges and Investigative Lessons from Azure SQL Database. *IEEE Access* **2025**, *13*, 204168–204183. [CrossRef]
- Al-dhaqm, A.; Razak, S.; Othman, S.H.; Ngadi, A.; Ahmed, M.N.; Mohammed, A.A. Development and Validation of a Database Forensic Metamodel (DBFM). *PLoS ONE* **2017**, *12*, e0170793. [CrossRef]
- Microsoft. Shrink a Database. Available online: <https://docs.microsoft.com/en-us/sql/relational-databases/databases/shrink-a-database> (accessed on 1 March 2025).
- Ko, M.; Lee, J.; Kim, K. A Study on Anti-Forensics Techniques and Countermeasures in Database Environments. *J. Digit. Forensics* **2014**, *8*, 19–31.
- Beyer, S.; Brumbaugh, M. Database Anti-Forensics. In *Proceedings of the 2011 Information Security Curriculum Development Conference, Kennesaw, GA, USA, 30 September–1 October 2011*; ACM: New York, NY, USA, 2011; pp. 12–19.
- Quick, D.; Choo, K.K.R. Impacts of Increasing Volume of Digital Forensic Data: A Survey and Future Research Challenges. *Digit. Investig.* **2014**, *11*, 273–294. [CrossRef]
- Martini, B.; Choo, K.K.R. An Integrated Conceptual Digital Forensic Framework for Cloud Computing. *Digit. Investig.* **2012**, *9*, 71–80. [CrossRef]
- Lillis, D.; Becker, B.; O'Sullivan, T.; Scanlon, M. Current Challenges and Future Research Areas for Digital Forensic Investigation. *arXiv* **2016**, arXiv:1604.03850.
- Garfinkel, S.L. Digital Forensics Research: The Next 10 Years. *Digit. Investig.* **2010**, *7*, S64–S73. [CrossRef]
- Nelson, B.; Phillips, A.; Steuart, C. *Guide to Computer Forensics and Investigations*, 5th ed.; Cengage Learning: Boston, MA, USA, 2014.
- Anglano, C.; Canonico, M.; Guazzone, M. Forensic Analysis of the ChatOn Instant Messaging Application on Android Smartphones. *Digit. Investig.* **2016**, *19*, 44–59.
- Microsoft. Pages and Extents Architecture Guide. Available online: <https://docs.microsoft.com/en-us/sql/relational-databases/pages-and-extents-architecture-guide> (accessed on 1 March 2025).

24. Carrier, B. *File System Forensic Analysis*; Addison-Wesley Professional: Upper Saddle River, NJ, USA, 2005.
25. Microsoft. DBCC SHRINKDATABASE (Transact-SQL). Available online: <https://learn.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-shrinkdatabase-transact-sql> (accessed on 1 March 2025).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.