KIPS Tr. Comp. and Comm. Sys.
Vol.6, No.12 pp.487~496 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2017.6.12.487

A Study on the Improvement Method of Deleted Record Recovery in MySQL InnoDB

Jung Sung Kyun† · Jee Won Jang†† · Doo Won Jeoung††† · Sang Jin Lee††††

ABSTRACT

In MySQL InnoDB, there are two ways of storing data. One is to create a separate tablespace for each table and store it separately. Another is to store all table and index information in a single system tablespace. You can use this information to recover deleted data from the record. However, in most of the current database forensic studies, the former is actively researched and its structure is analyzed, whereas the latter is not enough to be used for forensics. Both approaches must be analyzed in terms of database forensics because their storage structures are different from each other. In this paper, we propose a method for recovering deleted records in a method of storing records in IBDATA file, which is a single system tablespace. First, we analyze the IBDATA file to reveal its structure. And introduce delete record recovery algorithm which extended to an unallocated page area which was not considered in the past. In addition, we show that the recovery rate is improved up to 68% compared with the existing method through verification using real data by implementing the algorithm as a tool.

Keywords : Database Forensic, MySQL InnoDB, File-Per-Table, IBDATA, Record Recovery

MySQL InnoDB의 삭제된 레코드 복구 기법 개선방안에 관한 연구

정 성 균† · 장 지 원†† · 정 두 원††† · 이 상 진††††

요 약

MySQL InnoDB에서는 데이터를 기록할 때, 테이블마다 테이블스페이스를 분할 생성하여 개별적으로 저장하는 방법과 모든 테이블 및 인덱스 정보를 단일 시스템 테이블스페이스에 저장하는 방법이 있다. 그리고 이렇게 기록된 정보들을 이용해 삭제된 데이터를 레코드 단위로 복구하는 것이 가능하다. 하지만 현재 진행된 대부분의 데이터베이스 포렌식 연구에서 전자의 경우는 연구가 활발하게 이루어져 그 구조에 대한 분석이 많이 이루어졌지만, 후자에 대해서는 아직 공개된 정보가 포렌식에 활용되기엔 충분치 않다. 위의 두 방식은 각각의 저장 구조가 서로 다르기 때문에 데이터베이스 포렌식 관점에서 본다면 둘 모두에 대한 분석이 이루어져야 한다. 본 논문에서는 단일 시스템 테이블스페이스인 IBDATA 파일에 레코드를 저장하는 방식에서의 삭제된 레코드 복구 방법을 제시한다. IBDATA 파일을 분석하여 그 구조를 밝히고 이를 활용하여 기존에 고려되지 않았던 미할당 페이지 영역까지 확장한 삭제 레코드 복구 알고리즘을 소개한다. 또한, 알고리즘을 도구로 구현하여 실제 데이터를 이용한 검증을 통해 기존 방법보다 복구율이 68%까지 향상되었음을 보인다.

키워드 : 데이터베이스 포렌식, MySQL InnoDB, File-Per-Table, IBDATA, 레코드 복구

1. 서 론

오늘날의 비즈니스 세계에서 사용되는 거의 모든 응용 프로그램들은 방대한 데이터의 효율적인 관리를 위해 데이터베이스를 기반으로 동작한다. 데이터베이스는 기업의 업무 데이터를 저장하는 수단이 되었기 때문에, 기업 부정과 관련된 사건에서 중요한 조사 대상으로 다뤄진다[1]. 이러한 포렌식 조사 과정에서는 피조사자가 데이터를 삭제하여 디지털 증거를 훼손할 가능성이 존재하며 이에 대한 대응책이 마련되어야 한다[2]. 이와 같은 사실은 데이터베이스의 삭제된 데이터 복원에 대한 중요성을 더하고 있다.

MySQL 데이터베이스는 세계에서 두 번째로 점유율이 높은 DBMS(Database Management Systems)로, 국내외 기업과

† 준 회 원 : 고려대학교 정보보호대학원 정보보호학과 석사과정
†† 준 회 원 : 고려대학교 정보보호대학원 정보보호학과 석사
††† 비 회 원 : 고려대학교 정보보호대학원 정보보호학과 석·박사통합과정
†††† 종신회원 : 고려대학교 정보보호대학원 교수
Manuscript Received : May 24, 2017
First Revision : June 8, 2017
Second Revision : July 25, 2017
Accepted : August 16, 2017
* Corresponding Author : Sang Jin Lee(sangjin@korea.ac.kr)

각종 애플리케이션 서비스에 널리 사용되고 있다. MySQL 엔진으로는 MyISAM과 이를 개선하여 더욱 우수한 성능을 제공하는 InnoDB가 있다. MySQL 5.5 버전부터 InnoDB가 디폴트(default) 엔진으로 배포되고 있어[4], InnoDB 엔진에 대한 디지털 포렌식 수요가 증가하고 있다. 이에 따라 학계에서는 InnoDB를 대상으로 디지털 포렌식 분석 방법론에 대한 다양한 연구가 진행되었다. 기존 연구들은 그 나름의 성과를 거두었으나 InnoDB의 구조 및 로그 파일에 대한 분석이 주로 이루었다[6, 7]. 삭제된 레코드 복구에 대한 연구도 존재하나[10] 특정 기능이 활성화된 경우에만 적용할 수 있어 제안한 기법을 MySQL InnoDB에 범용적으로 활용하기에는 한계가 있으며 복구 대상이 할당 페이지 영역에만 한정되는 문제점이 있다.

이에 본 논문에서는 기존 연구의 한계점을 극복할 수 있는 InnoDB 레코드 복구 기법을 제안한다. 먼저 2장에서 관련 연구 및 한계점을 밝히고 3장에서 InnoDB의 주요 파일들의 구성을 소개한다. 4장에서는 역공학 방법을 통해 새로이 밝혀낸 IBDATA 파일의 구조에 관해 설명하고 이를 기반으로 5장에서 개선된 레코드 복구 기법을 제시한다. 또한, 6장에서 실험을 통해 알고리즘을 평가 및 검증함으로써 조사 과정에서의 실효성을 입증한다. 마지막으로 7장에서 본 논문의 기여에 대해 논한다.

2. 관련 연구

MySQL 데이터베이스와 관련한 포렌식 연구는 꾸준하게 진행되어 왔다. Harmeet Kaur Khanuja는 MySQL 데이터베이스 포렌식 절차를 기술하고 유의미한 아티팩트들에 대하여 소개하였다[5]. 또한, Peter Frühwirt의 경우 MySQL InnoDB에서 테이블 스키마 정보가 정의되는 FRM 파일의 구조와 데이터 저장 공간에서의 자료형 일부를 조사하였다[6].

위 연구 결과를 바탕으로 데이터를 복원하는 연구가 진행되었다. Peter Frühwirt는 MySQL의 redo 로그를 기록하는 파일인 ib_logfile0, ib_logfile1에 대해서 분석하고 이를 바탕으로 트랜잭션 히스토리를 재구축하는 방법에 대해서 논했다[7]. 하지만 이 방법은 데이터베이스의 상태 변환을 파악할 수 있을 뿐, 변환된 데이터가 구체적으로 무엇인지 제시할 수 없는 한계가 있다.

이러한 한계를 극복하기 위해 레코드 데이터 복구에 초점이 맞추어져 연구가 이루어졌다. James Wagner는 DB2, MySQL, Oracle 등 다양한 관계형 데이터베이스들이 공통적으로 가지는 구조적 특징을 기반으로 레코드 데이터를 복원하는 방안을 제시하였다[8]. 하지만 이를 단순히 관계형 데이터베이스라는 범용적인 특징을 적용하는 것만으로는 신뢰성 있는 복구결과를 획득하기 어렵다. Woo-seon Noh는 MySQL의 MyISAM 엔진[9], Jee-won Jang은 MySQL의 InnoDB 엔진[10]에서 운영된 데이터 파일들의 구조를 분석하고 복구 절차를 제시하였다. 하지만 MyISAM과 InnoDB의 구조가 상이하므로 [9]의 연구 결과를 InnoDB에 적용할 수 없다. [10]의 연구는 InnoDB에서 IBD 데이터 파일을 활용하는 경우만을 고려하고 있기 때문에 IBDATA 파일에 대한 분석이 전무하다. 또한, 할당 상태의 페이지만을 복원 대상으로 선정하여 미할당 페이지에 잔존하는 레코드를 복구할 수 없는 문제점을 가진다.

위의 연구들에서 살펴본 바와 같이 현재 제시된 InnoDB 엔진에서의 분석 및 복구 방안을 실무적으로 적용하기에는 제약사항이 따른다. 따라서 아직 분석되지 않고 레코드 단위 복구 절차가 정의되지 않은 InnoDB의 IBDATA 파일에 대한 연구가 필요하다. 본 논문에서는 File-Per-Table 옵션의 비활성화 상태에서 운영된 MySQL의 IBDATA 파일의 구조를 분석하고 이를 활용하여 삭제된 레코드를 복원하는 방법을 소개한다. 이 과정에서 성능 고도화를 위한 카빙 복구 기법을 새롭게 제시하며 이는 InnoDB의 어떠한 데이터 파일에도 적용 가능하므로 기존 연구의 한계를 극복할 수 있다.

3. 배경 지식

3.1 InnoDB 파일 구성

MySQL 설치 시, C:\ProgramData\MySQL 경로에 각 설치 버전에 해당하는 이름의 폴더가 생성되고, 해당 폴더에는 서버에서 관리하는 데이터 파일 및 설정 파일이 저장된다. Table 1은 MySQL 데이터베이스 포렌식 조사에서 분석 대상이 되는 주요 파일들의 경로를 나타낸다.

Table 1. Key File Paths in InnoDB

Name	Type	Path
my.ini	File	C:\ProgramData\MySQL\MySQL Server 5.x\
<database name>	Directory	C:\ProgramData\MySQL\MySQL Server 5.x\Data\
<table name>.frm	File	C:\ProgramData\MySQL\MySQL Server 5.x\Data\<database name>\
<table name>.ibd	File	C:\ProgramData\MySQL\MySQL Server 5.x\Data\<database name>\
ibdata	File	C:\ProgramData\MySQL\MySQL Server 5.x\Data\

my.ini는 설정 파일로 file-per-table 옵션의 활성화 여부를 확인할 수 있다. 데이터베이스를 생성하면 경로 C:\Program Data\MySQL\MySQL Server 5.x\Data\에 데이터베이스 이름의 폴더가 만들어진다. 생성된 폴더 내에 각 테이블의 스키마 정보를 저장하고 있는 FRM 파일이 생성된다.

레코드 데이터는 file-per-table 활성화 여부에 따라 IBDATA 혹은 <table name>.IBD에 저장된다[11]. 해당 기능을 활성화한 경우에는 테이블의 레코드는 각 IBD 파일에 저장되고, 비활성화 상태에서는 IBDATA에 일괄적으로 저장된다. 옵션에 따른 데이터의 저장방식이 상이하므로 각 상황에 따른 조사 기법 연구가 필요하다.

3.2 FRM 파일 구조

테이블 별로 개별적으로 생성되는 FRM 파일에는 레코드 데이터 해석에 필요한 모든 스키마 정보가 기록된다. Fig. 1은 FRM 파일 구조를 도식화한 예이다. FRM 파일은 자신이 나타내고 있는 테이블의 이름으로 생성되며 내부는 크게 FRM Header, Key Information, Column Information 3개의 영역으로 구분할 수 있다.

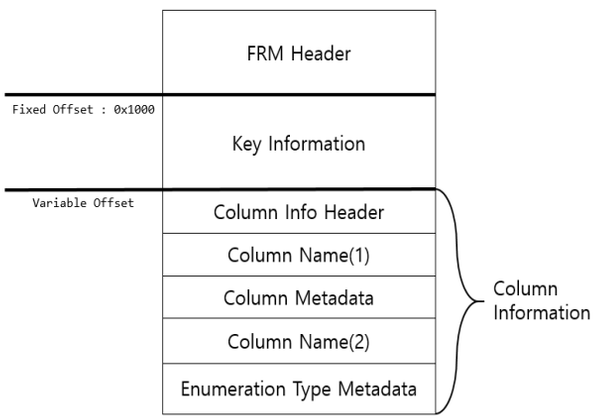

Fig. 1. FRM File Structure

FRM Header에서는 Column Information 영역의 시작 오프셋 값과 MySQL의 버전, 스토리지 엔진 타입 등의 정보가 있다. Key Information 영역은 오프셋 0x1000에서 시작하며 기본 키, 인덱스 키 등에 대한 정보가 있다. Column Information 영역에는 컬럼 이름과 데이터 타입이 저장돼 있다[6].

4. IBDATA 파일 구조

본 연구진은 MySQL 공식 사이트 및 현행 연구 내용을 토대로 InnoDB 엔진을 역공학 하여 IBDATA 파일의 데이터 저장 방식을 분석하였다.

4.1 기본 구조 및 특징

IBDATA 파일의 구조는 Fig. 2와 같다. IBDATA 파일은 16,384바이트의 페이지들로 구성되어 있으며 저장되는 정보의 성격에 따라 시스템 영역과 데이터 영역으로 나뉜다. 시스템 영역에는 사용자가 생성한 데이터베이스와 테이블에 대한 메타 정보 등이 있고, 데이터 영역에는 실제 데이터베이스 사용자가 저장한 데이터가 있다. 시스템 영역 중 레코드 복구에 필요한 정보는 IBDATA 파일의 오프셋 0x20000부터 존재한다. 데이터 영역의 페이지들의 위치는 유동적이며 시스템 영역의 Index 페이지들로부터 인덱싱된다.

IBDATA에 기록되는 각 페이지는 0x26(38)바이트 크기의 Fil Header 구조체가 있으며 주요 항목은 Table 2와 같다. FIL_PAGE_TYPE 항목은 페이지의 타입을 나타내는데 해당 값이 FIL_PAGE_INDEX (0x45BF)로 정의될 경우 그 페이지는 B+ 트리 구조의 Index 페이지 중 하나임을 의미한다.

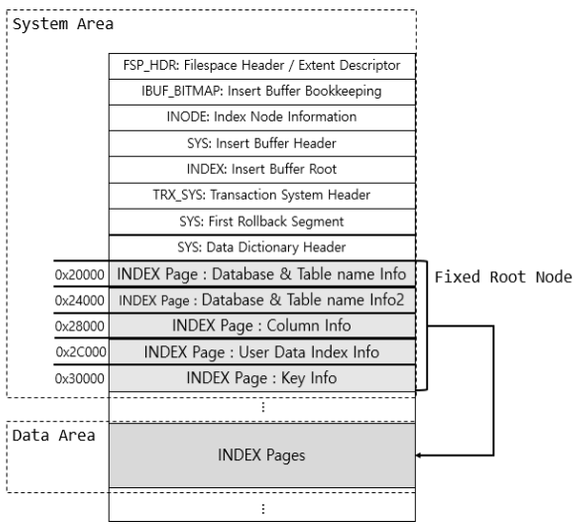

Fig. 2. IBDATA File Structure

Table 2. Key Values in Fil Header Structure

Name	Offset (From start of the page)	Size (Bytes)	Description
FIL_PAGE_OFFSET	0x04	4	Ordinal page number from start of space
FIL_PAGE_PREV	0x08	4	Offset of previous page in key order
FIL_PAGE_NEXT	0x0C	4	Offset of next page in key order
FIL_PAGE_TYPE	0x18	2	Type of page
FIL_PAGE_SPACE	0x22	4	ID of the space the page is in

InnoDB 엔진은 Index 페이지를 통해 모든 사용자 데이터 정보를 저장한다. 각 Index 페이지는 Fil Header 다음으로 Page Header와 레코드 데이터가 있다. Table 3은 Page Header의 주요 정보를 나타낸 것이다. Index 페이지가 가지는 B+ 트리 구조에서는 리프 노드 페이지에 실질적인 데이

Table 3. Key Values in Page Header

Name	Offset (From start of the page)	Size (Bytes)	Description
PAGE_FREE	0x2C	2	Record pointer to first free record
PAGE_LAST_INSERT	0x30	2	Record pointer to the last inserted record
PAGE_N_RECS	0x36	2	Number of user records
PAGE_LEVEL	0x40	2	Level within the index (0 for a leaf page)
PAGE_INDEX_ID	0x42	8	Identifier of the index the page belongs to

터가 기록되고 이외의 부모 노드들은 각각의 자식 노드들을 인덱싱하는 용도로 사용된다. 리프 노드는 Page Header 구조체가 가지는 2바이트 크기의 PAGE_LEVEL 값이 0인 경우를 의미한다.

페이지 내에 기록되는 레코드들은 링크드 리스트 형태로 연결되어 있으며 각각 할당된 2바이트 값으로 다음 레코드를 인덱싱한다. 위와 같은 헤더가 위치한 다음에는 infimum 그리고 supremum이라 불리는 두 레코드가 기록되는데 전자는 페이지 내의 첫 번째 레코드를 인덱싱하고, 후자는 마지막 레코드로부터 인덱싱된다.

4.2 시스템 Index 페이지 별 레코드 데이터 분석

앞서 언급한 것처럼, IBDATA 파일은 시스템 테이블 영역에서 사용자가 생성한 데이터베이스와 테이블에 대한 메타데이터를 저장하고 있다. 시스템 테이블 영역의 고정된 다섯 개 Index 페이지에 데이터베이스의 주요 정보들이 저장되어 있다(Fig. 2 참조). 해당 페이지들에는 각각 다른 유형의 정보가 저장되어 있어 개별적인 분석이 필요하다. 이번 절에서는 다섯 개의 페이지 중 레코드 복원에 직접 활용되는 두 페이지의 레코드 구조를 설명한다.

1) 데이터베이스 및 테이블 이름 페이지

FIL_PAGE_OFFSET이 0x08 값을 가지는 페이지에서는 시스템이 가지고 있는 모든 데이터베이스와 테이블에 대한 이름 정보를 표시한다. 레코드 헤더의 크기는 16바이트이며 레코드의 주요 항목은 Fig. 3 및 Table 4와 같다.

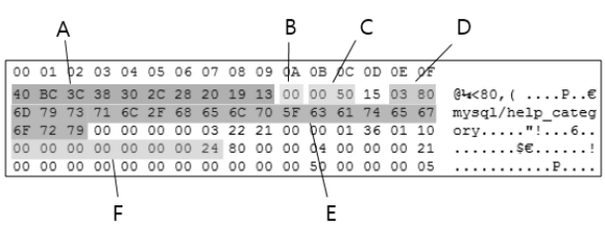

Fig. 3. User Record Structure of 0x08 Page

Table 4. Descriptions for the Upper Figure

Name		Size (Bytes)	Description
Record Header	A	10	Length info of contents in record
	B	1	State of record
	C	2	Checksum value
	D	2	Next record offset
Contents	E	variable	Database name/Table name
	F	8	Table ID

레코드 헤더의 첫 10바이트에서 해당 레코드가 가지는 각 데이터의 길이 정보를 1바이트 크기씩 역순 방향으로 점진적으로 합산하여 표시한다. 따라서 헤더의 첫 번째 바이트가 헤더를 제외한 레코드 데이터의 총 길이를 나타낸다. 그 다음 1바이트 크기로 레코드의 상태를 표시하는데, 삭제된 레코드의 경우 0x20 값을 가진다. 그리고 레코드 생성 때마다 8씩 증가하는 2바이트 크기의 체크 섬 값을 가진다. 헤더의 마지막 2바이트는 링크드 리스트로 연결된 자신의 다음 레코드 오프셋 값이다.

0x08 페이지의 레코드는 데이터베이스와 테이블의 이름 정보를 '/' 구분자를 사이에 두고 함께 표현하고 있다. 또한, 다른 테이블 관련 정보들과의 매핑(mapping)을 위해 사용되는 8바이트 크기의 Table ID가 있다. Table ID는 데이터베이스 시스템 내에서 테이블마다 값이 고유하므로 이를 통해 다른 페이지 영역에 접근했을 때 어떠한 이름의 테이블에 대한 정보인지 파악할 수 있다.

2) 사용자 데이터 인덱스 정보 페이지

FIL_PAGE_OFFSET이 0x0B 값을 가지는 페이지는 테이블의 데이터 즉, 사용자가 입력한 레코드 값들을 가지는 페이지들에 대한 인덱스 정보를 저장하고 있다. 레코드 헤더 크기는 15바이트이며 레코드의 주요 항목은 Fig. 4 및 Table 5와 같다.

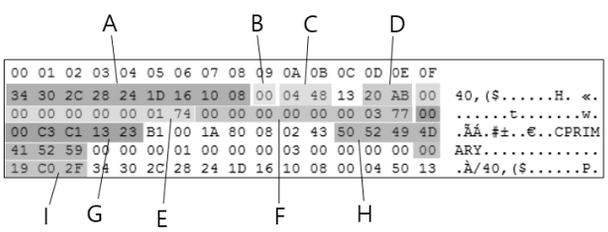

Fig. 4. User Record Structure of 0x0B Page

Table 5. Descriptions for the Upper Figure

Name		Size (Bytes)	Description
Record Header	A	9	Length info of contents in record
	B	1	State of record
	C	2	Checksum value
	D	2	Next record offset
Contents	E	8	Table ID
	F	8	Page Index ID
	G	6	Transaction ID
	H	variable	Key name
	I	4	Page offset of User data page

헤더 다음으로 0x0B 페이지의 레코드가 갖는 데이터는 Table ID, Page Index ID, Transaction ID, Key name, 데이터 페이지 오프셋 정보 등이 있다. Table ID 또한 위에서 언급된 것과 같이 인덱싱하는 데이터 페이지가 어떠한 테이블에 속한 것인지 표현하는 값이고, Page Index ID는 각 페이지의 Page Header에 있는 8바이트 크기의 PAGE_INDEX_ID 값을 의미한다. 그리고 가변 길이의 키 이름을 가지는데 이는 해당 레코드가 인덱싱하고 있는 테이블에 정의된

키의 형태에 따라 값을 달리한다. 만약 하나 이상의 기본 키가 정의되어 있다면 “PRIMARY” 문자열이 기록되고, 기본 키 없이 하나 이상의 인덱스 키를 가진다면 가장 첫 번째로 정의된 인덱스 키의 이름이 저장된다. 끝으로 레코드의 마지막 4바이트 값을 통해 해당 테이블의 레코드 데이터를 저장하는 사용자 데이터 페이지 오프셋 정보를 표현한다. 그리고 해당 오프셋으로부터 인덱싱된 페이지는 각 테이블이 독립적으로 가지는 B+ 트리 구조에서의 루트 노드를 의미한다. 따라서 해당 페이지로부터 리프 노드를 찾고, Fil Header가 가지는 FIL_PAGE_NEXT를 참조하여 테이블 내의 모든 리프 페이지를 순회할 수 있다.

0x0B 페이지가 저장하고 있는 레코드의 수는 정의된 테이블의 개수와 동일하며 서로 일대일로 대응하기 때문에 데이터베이스 시스템 내에 할당된 모든 사용자 데이터 페이지를 식별 및 접근할 수 있다. 하지만 레코드 삭제 행위로 인해 B+ 트리 구조에서 탈락한 미할당 페이지는 더 이상 0x0B 페이지를 통해 접근할 수 없다.

4.3 삭제 레코드의 저장

InnoDB 엔진은 선행연구 [10]에서 분석한 IBD 파일과 동일한 방법으로 IBDATA 파일에서 삭제된 레코드를 저장 및 관리한다. 할당 및 미할당 페이지 각각에서의 삭제 레코드 기록 방식은 다음과 같다.

1) 할당 페이지 내의 삭제 레코드

사용자 데이터 페이지의 레코드는 자신이 속한 테이블이 어떠한 스키마로 정의되었는지에 따라서 기록되는 방식이 달라지기 때문에 정확한 스키마 정보를 기반으로 해석이 이루어져야 한다. 이러한 사전 정보를 가진 상태에서 페이지에 접근한다면 Page Header에 있는 PAGE_FREE 항목을 참조하여 삭제 레코드에 접근하고 이를 복원해낼 수 있다. Fig. 5는 사용자 데이터 페이지에서 PAGE_FREE를 통한 삭제 레코드 파싱에 대한 예를 보인다. PAGE_FREE는 linked list 형태로 연결된 삭제 레코드 집합에서의 가장 첫 번째

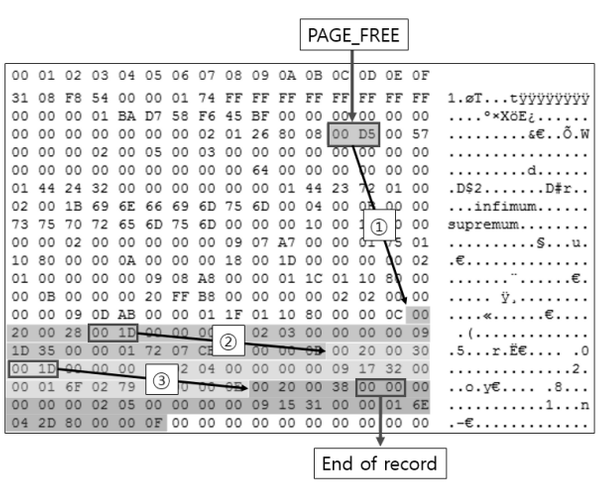

Fig. 5. Parse the Deleted Records in Allocated Page

레코드를 가리키고 있기 때문에 해당 페이지 내 삭제된 모든 레코드에 대한 접근이 가능하다.

2) 미할당 페이지 내의 삭제 레코드

여러 레코드가 삭제될 경우 MySQL 엔진은 단순히 특정 데이터 페이지의 할당 정보를 해제하는 것으로 작업을 마친다. 따라서 이러한 미할당 페이지의 경우 삭제 레코드가 정상 레코드의 메타데이터를 가질 수 있다. 이로 인해 PAGE_FREE 및 레코드 헤더 영역의 상태 값은 레코드를 저장한 페이지가 할당되어 있을 때만 유효하며, 미할당 페이지의 경우 이를 신뢰할 수 없다.

4.4 삭제 레코드의 유실

페이지에 기록되는 삭제 상태의 레코드는 정상 레코드와는 달리 새로운 레코드를 삽입하거나 기존 레코드를 삭제하는 행위로 인해 해당 영역이 다른 데이터로 덮어져 완전히 지워질 수 있다. 레코드의 삽입 및 삭제 행위 각각에서의 삭제 레코드 유실 과정은 다음과 같다.

1) 새로운 레코드 삽입

InnoDB는 기본적으로 페이지 내에서 기존 레코드가 끝난 직후에 새로운 레코드를 삽입한다. 하지만 만약 삭제 상태의 레코드의 크기가 새로운 레코드가 기록될 크기보다 작거나 같을 경우 해당 삭제 레코드는 새로운 레코드로 덮어써 진다.

Index 페이지의 B+ 트리 구조에서 새로운 레코드 삽입으로 인해 리프 페이지가 추가로 할당되어 B+ 트리의 높이가 증가한다면 기존 저장된 레코드에서 삭제 레코드를 제외한 정상 레코드만으로 새로운 B+ 트리를 생성한다. 이 과정에서 기존 삭제 레코드가 저장되어 있던 페이지들은 미할당 상태가 되어 새로 할당되는 페이지에 의해 덮어 써질 수 있다.

2) 기존 레코드 삭제

페이지에 저장된 레코드는 사용자에 의해 삭제될 경우 페이지 내의 infimum 및 supremum으로 구성된 linked list에서 탈락하며 Page Header에서의 PAGE_FREE와 연결된 새로운 linked list에 속하게 된다. 만약 단일 루트 페이지만으로 이루어진 B+ 트리 구조에서 페이지 내의 모든 레코드가 삭제된다면 해당 페이지는 초기화되어 모든 레코드의 흔적이 사라진다.

그리고 2 이상의 높이를 가진 B+ 트리의 경우 레코드 삭제 행위로 인해 정상 레코드를 할당하는 데에 필요한 페이지의 개수가 줄어든다면 삭제 레코드가 포함된 페이지는 B+ 트리 구조에서 탈락하고 여분의 페이지들에 레코드를 다시 기록한다. 이 과정에서 삭제 상태의 레코드는 다시 기록되지 않기 때문에 흔적이 사라질 수 있다.

5. 레코드 데이터 복구 기법

5.1 FRM 파일을 이용한 스키마 정의

FRM 파일에는 레코드 데이터 해석에 필요한 모든 스키

마 정보가 기록된다. 따라서 모든 자료형 데이터에 대한 신뢰성 있는 복원을 위해 3장 2절에 설명된 FRM 파일 구조를 참조하여 스키마 정보를 추출한다.

5.2 사용자 데이터 페이지 매핑

IBDATA 파일의 경우 스키마와 사용자 데이터 페이지를 인덱싱하는 레코드들에 식별 가능한 고유 ID를 부여하기 때문에 서로를 연결할 수 있다. 하지만 FRM 파일에는 IBDATA 파일 내에서 인덱싱 정보로 활용되는 ID 값들이 저장되지 않는다. 따라서 각 FRM 파일과 IBDATA 파일 내의 사용자 데이터 페이지의 매핑 과정이 필요하다. 여기에 활용될 수 있는 시그니처는 다음과 같은 조건들이 요구된다.

- FRM과 IBDATA 두 파일에서 공통으로 수집할 수 있는 값이어야 한다.
- IBDATA 파일 내에서의 고정된 5개 페이지 0x08 ~ 0x0C를 통해 접근 가능해야 한다.
- 데이터베이스 시스템 내에서 고유한 값이어야 한다.

본 논문에서는 위 조건을 만족하는 값으로 데이터베이스와 테이블의 이름 조합을 활용하였다. FRM 파일은 데이터베이스 이름의 폴더 내에 테이블 이름으로 생성된다. 그리고 IBDATA 파일은 0x08 페이지와 0x09 페이지에서 데이터베이스와 테이블 이름을 기록한다. 또한, 동일한 데이터베이스 시스템 내에서는 데이터베이스와 테이블 이름 모두가 동시에 중복될 수 없기 때문에 이 두 이름의 조합 또한 고유한 값으로 사용될 수 있다. 따라서 0x08 페이지 혹은 0x09 페이지에 접근하여 수집한 데이터베이스와 테이블 이름 정보가 FRM 파일이 속한 폴더 및 파일명과 일치한다면, 해당 레코드에 기록된 Table ID 값을 활용하여 사용자 데이터 페이지에 접근할 수 있다.

5.3 삭제 레코드 복구 알고리즘

이번 절에서는 앞서 분석된 IBDATA 및 FRM 파일의 구조를 기반으로 수집된 데이터베이스 파일을 재해석하여 삭제된 레코드를 복원하는 알고리즘을 소개한다.

IBDATA 파일에 저장되는 각 페이지는 자신이 저장한 모든 레코드가 삭제될 경우 더 이상 메타데이터를 통해 접근할 수 없는 미할당 상태가 된다. 따라서 IBDATA 파일에 저장된 사용자 데이터 페이지로부터 삭제된 레코드를 복구할 때, 해당 페이지의 할당 정보를 고려해야만 한다. 본 논문에서는 페이지 할당 여부에 따라 서로 다른 복구 알고리즘을 적용하는 방법을 제안한다.

1) 스키마 구조 정의

관계형 데이터베이스 구축 시, 레코드를 삽입할 스키마 정의가 필수적으로 선행되어야 한다. 따라서 레코드 데이터를 참조할 IBDATA 파일에 접근하기 전에 데이터베이스 내에 정의된 스키마 즉, FRM 파일들을 먼저 해석해야 한다. Fig. 6

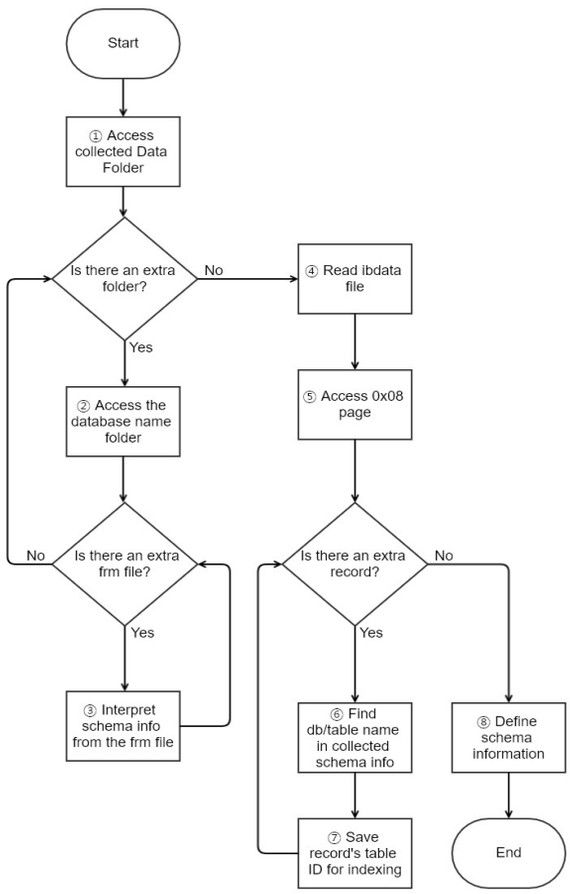

Fig. 6. Define Schema Information

은 이와 같은 스키마 정의 과정에 대한 순서도를 나타낸다.

MySQL 데이터베이스 저장 경로의 파일들을 수집함으로써 삭제 레코드 복원에 필요한 데이터베이스 이름 폴더 및 내부의 FRM 파일들과 IBDATA 파일에 접근할 수 있다(1). 우선 스키마 정보를 가져오기 위해 각 데이터베이스 이름 폴더에 접근하여(2) 내부에 저장된 FRM 파일들을 해석한다(3). 또한, 이 과정에서 폴더와 파일명으로부터 데이터베이스 및 테이블 이름을 파악할 수 있다. 각 데이터베이스 이름 폴더 내부의 모든 FRM 파일들을 스캔하고 난 이후에는 다음과 같이 IBDATA 파일에서의 시스템 테이블 영역을 파싱하는 과정을 거친다(4). 먼저 데이터베이스 및 테이블의 이름 정보를 기록하고 있는 0x08 페이지에 접근한다(5). 그리고 해당 페이지의 레코드와 데이터베이스 폴더 및 FRM 파일로부터 수집한 이름 정보를 비교하여 동일한 경우를 찾고(6), 각각을 Table ID로 연결한다(7). 이러한 매핑 과정 이후에는 0x0B 페이지의 레코드 각각이 어떠한 테이블의 사용자 데이터 페이지를 인덱싱하는지 구분할 수 있으며, 이로써 삭제 레코드 복구를 위한 스키마 정의가 가능하다(8).

2) 할당 페이지에서의 레코드 복구

복구 대상 데이터베이스의 스키마 구조가 정의된 이후에는 레코드 데이터를 저장하고 있는 IBDATA 파일을 읽어

들여 이를 기반으로 다음과 같이 삭제 레코드 복원 절차를 수행할 수 있다. Fig. 7은 본 논문에서 제안하는 할당 페이지 내에서 삭제된 레코드 복원을 위한 절차이다.

0x0B 페이지의 레코드에서 가지는 오프셋 정보와 고유 Table ID를 이용해(1) 테이블 별로 독립된 B+ 트리 구조의 데이터 페이지에 접근한다(2). 그리고 해당 루트 노드로부터 실질적인 레코드 데이터가 저장된 리프 노드를 찾은 뒤, PAGE_FREE 항목을 참조하여(3) 삭제 레코드 복원을 수행한다(4). 만약 PAGE_FREE 참조 오프셋 값이 0x00 일 경우 해당 리프 페이지에는 삭제 상태의 레코드가 없음을 의미한다(5). 또한, 삭제 레코드에서의 다음 레코드 오프셋이 해당 페이지의 시작점 즉, 0x00을 가리킬 경우 해당 레코드가 링크드 리스트에서의 마지막 노드임을 알 수 있다. 단일 페이지 내에서 더 이상 복원할 레코드가 없다면 FIL_PAGE_NEXT 항목을 참조하여(6) 다음 리프 페이지로 접근 및 동일한 삭제 레코드 복구 작업을 반복한다. 특정 페이지에서의 4바이트 크기 FIL_PAGE_NEXT 참조 오프셋 값이 전부 0xFF로 채워질 경우 링크드 리스트에서의 마지막 노드 또는 페이지를 의미한다. 마지막 페이지까지 레코드 복원이 완료되었다면 다시 0x0B 페이지로 돌아가 다음 레코드를 읽고, 첫 과정을 반복하여 모든 테이블에 대해 레코드 복구 절차를 수행할 수 있다.

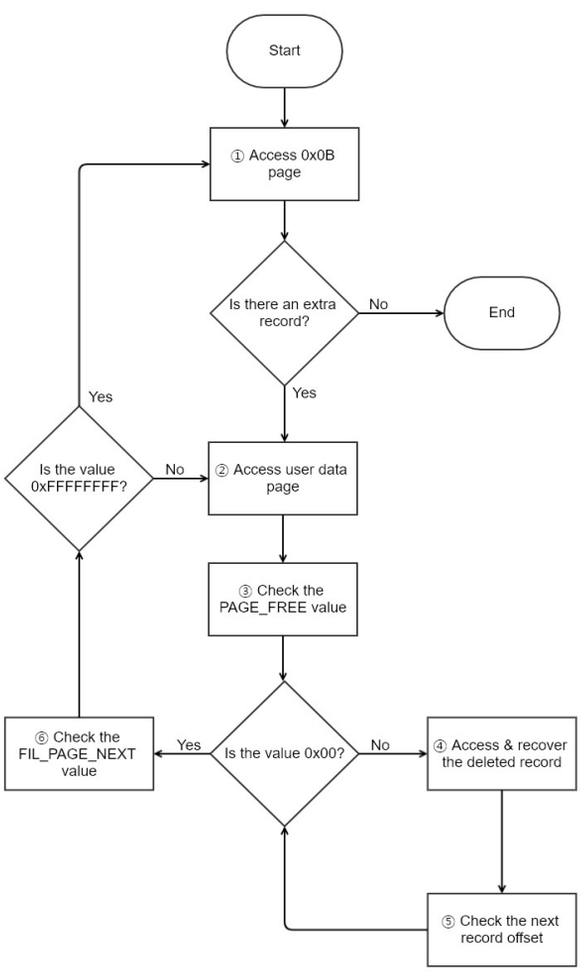

Fig. 7. Recover Deleted Records from the Allocated Pages

3) 미할당 페이지에서의 레코드 복구

미할당 페이지는 할당 상태에서와 동일한 스키마 구조를 기반으로 레코드를 저장하고 있지만, 메타데이터가 유실되어 더 이상 0x0B 페이지를 통해 참조할 수 없기 때문에 기존 방식으로 복원할 수 없다.

본 논문에서는 미할당 사용자 데이터 페이지 영역에 대해 카빙 기법을 적용하여 데이터를 복원하는 새로운 방법을 소개하며 그 절차는 Fig. 8과 같다. 해당 기법은 일반적인 파일 카빙 방식과 마찬가지로 임의의 시그니처에 의존하여 IBD 또는 IBDATA 파일 전체를 페이지 크기 단위로 스캔하는 방법이다. 따라서 다음과 같이 카빙에 사용될 시그니처를 먼저 특정하는 사전 작업이 필요하다.

a) FIL_PAGE_TYPE

페이지 상단에 기록되는 Fil Header 구조체에는 4바이트 크기 FIL_PAGE_TYPE 값으로 해당 페이지의 타입을 표현한다. 이때, 모든 사용자 데이터 페이지는 Index 페이지를 의미하는 FIL_PAGE_INDEX 타입을 가진다.

페이지 타입에 따라 표현되는 고윳값이 다르기 때문에

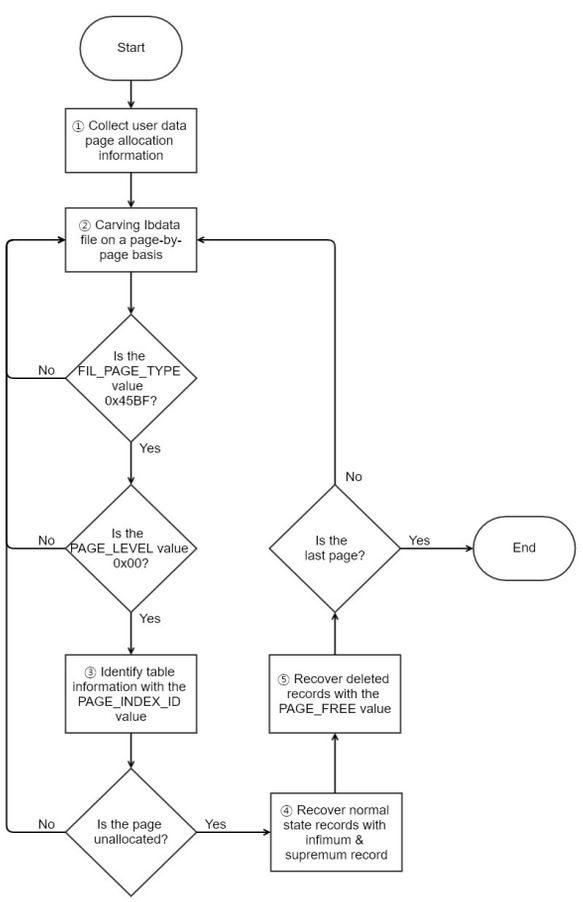

Fig. 8. Recover Deleted Records from the Unallocated Pages

FIL_PAGE_INDEX의 값인 0x45BF를 참조하여 Index 페이지 식별 값으로 활용할 수 있다.

b) PAGE_LEVEL

B+ 트리 구조에서 사용자 데이터를 가진 리프 페이지 외에도 해당 페이지를 인덱싱하는 부모 페이지 또한 FIL_PAGE_INDEX 타입을 가진다. 따라서 FIL_PAGE_TYPE만으로는 정확한 분류 작업을 수행할 수 없다.

Fil Header 구조체는 2바이트 크기의 PAGE_LEVEL 값으로 Index 페이지에서 노드의 레벨을 나타낸다. 해당 값은 루트 노드에 가까울수록 값이 증가하며 리프 노드는 0 값을 가진다. 이때, 복구 대상 데이터는 리프 노드에 저장되기 때문에 FIL_PAGE_INDEX 타입의 페이지 중 PAGE_LEVEL이 0 값을 가지지 않는 경우 인덱싱 페이지로 분류하여 복구 대상 페이지에서 제외할 수 있다.

c) PAGE_INDEX_ID

레코드의 복원을 위해서는 데이터 해석을 위한 스키마 정보가 필수적으로 요구된다. 따라서 FIL_PAGE_TYPE과 PAGE_LEVEL 두 개의 시그니처만으로 특정 사용자 데이터 페이지를 식별 및 접근했을 때, 어떠한 테이블에 종속되어 있는지에 대한 정보가 추가로 필요하다.

Page Header 구조체에서의 PAGE_INDEX_ID는 8바이트 크기로 해당 페이지의 종속 정보를 나타내며 미할당 페이지에서도 할당 상태와 동일한 값을 가진다. 따라서 이 값을 0x0B 페이지의 레코드에서 가지는 Page Index ID와 비교하여 해당 페이지의 스키마 정보를 특정할 수 있다.

d) 페이지 할당 정보

페이지의 할당이 해제되었다는 것은 해당 페이지 내의 모든 데이터가 삭제되었음을 의미한다. 따라서 미할당 사용자 데이터 페이지에 접근한다면 Page Header의 PAGE_FREE 항목을 통해 접근 가능한 삭제 레코드뿐만이 아닌 infimum 및 supremum 레코드와 연결된 정상 레코드도 복구 대상이 된다.

앞서 언급한 FIL_PAGE_TYPE, PAGE_LEVEL 그리고 PAGE_INDEX_ID라는 세 개의 시그니처를 활용한 카빙 방식으로도 사용자 데이터 페이지에 접근하여 레코드 복원이 가능하다. 하지만 미할당 페이지와 마찬가지로 할당 페이지에서도 위의 시그니처에 대해 동일한 값을 가지기 때문에 페이지의 할당 상태를 구분 지을 수 없으며 이로 인해 할당 여부에 따라 달라지는 삭제 레코드 복구 절차 또한 결정할 수 없는 문제가 발생한다. 페이지 할당 정보는 기존 시그니처와 달리 Fil Header 및 Page Header 구조체에 관련 항목이 존재하지 않는다. 이처럼 페이지 내부 메타데이터로부터 페이지 할당 여부를 식별할 수 없기 때문에 임의의 새로운 시그니처를 생성하여 활용해야만 한다.

본 논문에서 제안하는 복구 기법은 미할당 페이지에 대한 카빙을 적용하기에 앞서 할당 페이지의 스캔 과정을 먼저 진행하는 방법을 사용한다(1). 이와 같은 전처리 과정에서 페이지의 고유 순서 값인 FIL_PAGE_OFFSET을 토대로 할당 상태의 사용자 데이터 페이지만을 목록화할 수 있다. 이후에는 카빙 기법으로(2) 특정 사용자 데이터 페이지에 접근했을 때(3) 해당 페이지의 FIL_PAGE_OFFSET이 할당 페이지 목록에 존재하지 않을 경우 미할당 페이지로 분류할 수 있게 된다. 위의 조건들이 전부 충족될 경우 해당 페이지에 존재하는 모든 레코드를 복원한다(4, 5).

6. 구현 및 성능 평가

본 논문에서는 제시된 알고리즘의 실효성 검증을 위해 이를 도구로 구현하여 테스트를 진행하였다. 웹 게시판 서비스에 사용되었던 데이터베이스를 대상으로 실험하였으며 테이블들은 int, double, varchar, char, tinyint, mediumtext, text 등 다양한 데이터 타입으로 구성되어있다.

복구 실험은 원본 테이블에서 레코드 일부를 삭제하는 Case A와 모든 레코드를 삭제하는 Case B 그리고 Case A의 테이블에서 새로운 레코드를 삽입하는 Case C에 대해 이루어졌으며 각 테이블의 컬럼과 레코드 개수는 Table 6에서 확인할 수 있다. 모든 Case에서의 레코드 삭제 및 삽입 행위는 1번 테이블부터 7번 테이블까지 순차적으로 적용되었다.

미할당 영역에서의 카빙 기법 적용 전과 후로 실험 결과를 도출한 후 이를 비교 분석하여 제안 기법의 효용성을 검증하고자 하였다. 그 결과, 4장 4절에서 언급한 데이터 유실에 해당하지 않는 경우, 삭제 레코드를 전부 복구할 수 있음을 확인하였다.

Table 6. Specification of Tables

Table	Column number	Number of normal record			
		Original	Case A	Case B	Case C
1	7	13,690	8,000	0	10,500
2	41	6	2	0	52
3	4	2	1	0	51
4	4	6	2	0	52
5	14	1	1	0	1
6	4	1	1	0	1
7	47	27,114	21,846	0	24,346

6.1 부분 삭제(Case A)

부분 삭제의 경우 복구 대상 테이블에서 5, 6번 테이블은 1개의 레코드 개수만을 가지기 때문에 부분 삭제가 불가하므로 이를 제외한 나머지 테이블에서 임의 개수만큼 부분 삭제가 이루어졌다.

부분 삭제에 대한 레코드 복원 결과는 Table 7과 같다. 상대적으로 레코드 개수가 적은 2, 3, 4번 테이블에서의 경우 카빙 기법의 적용 여부와는 상관없이 전부 복원되었다. 하지만 1번 및 7번 테이블에서는 카빙 기법의 적용 여부에 따라 레코드가 추가로 복구되었다. 이때, 가장 먼저 삭제 행위가 이루어진 1번 테이블의 경우 다른 테이블에서의 삭제

Table 7. Recovery Result of Partial Deletion

Table	Delete count	Number of recovered record	
		Previous Method[10]	Proposed Method
1	5,690	1,262 (22%)	4,085 (72%)
2	4	4 (100%)	4 (100%)
3	1	1 (100%)	1 (100%)
4	4	4 (100%)	4 (100%)
7	5,268	5,117 (97%)	5,213 (99%)

행위로 인해 28% 정도의 삭제 레코드가 유실되었다. 반면에 마지막으로 삭제 행위가 이루어진 7번 테이블은 1%를 제외한 대부분의 삭제 레코드가 잔존하여 복구가 가능하였다.

6.2 전체 삭제(Case B)

전체 삭제의 경우 복구 대상 테이블이 가진 모든 레코드에 대해 삭제 행위가 이루어졌다.

전체 삭제에 대한 레코드 복원 결과는 Table 8과 같다. 7개 테이블 전부 할당 페이지 영역에서는 어떠한 삭제 레코드도 복원되지 않았다. 또한, 상대적으로 레코드 개수가 적은 2~6번 테이블은 모든 데이터가 유실되어 미할당 페이지 영역에서도 복원이 불가하였다. 반면 1번과 7번 테이블의 경우 카빙 기법을 통해 미할당 페이지 영역에서 일부를 복구할 수 있었다. 가장 먼저 삭제 행위가 이루어진 1번 테이블은 삭제 레코드가 대부분 유실되어 28%만이 복구되었고, 마지막으로 삭제된 7번 테이블은 상대적으로 많은 68%가 복구되었다.

Table 8. Recovery Result of Full Deletion

Table	Delete count	Number of recovered record	
		Previous Method[10]	Proposed Method
1	13,690	0 (0%)	3,809 (28%)
2	6	0 (0%)	0 (0%)
3	2	0 (0%)	0 (0%)
4	6	0 (0%)	0 (0%)
5	1	0 (0%)	0 (0%)
6	1	0 (0%)	0 (0%)
7	27,114	0 (0%)	18,393 (68%)

6.3 삭제 후 레코드 삽입(Case C)

Table 7과 동일하게 부분 삭제가 이루어진 테이블을 대상으로 임의 개수만큼 새로운 레코드를 삽입하고 이후 할당 및 미할당 페이지 각각에서의 복구 레코드 개수를 비교하였다.

삭제 후 레코드가 삽입된 경우에서의 레코드 복원 결과는 Table 9와 같다. 새로운 레코드의 삽입으로 인해 잔존했던 삭제 레코드가 일부 유실되어 5개 테이블 모두에서 할당 및 미할당 페이지에 대해 복원된 레코드의 개수가 감소하였다. 이는 할당 페이지에서 삭제 레코드가 새로운 정상 레코드로 덮어써지고, 삭제 레코드가 저장된 미할당 페이지가 새로이 할당되는 페이지로 대체되어 나타난 결과이다.

Table 9. Recovery Result of Record Insertion

Table	Number of inserted records	Number of recovered record	
		Previous Method[10]	Proposed Method
1	2,500	1,104 (19%)	3,683 (65%)
2	50	2 (50%)	2 (50%)
3	50	0 (0%)	0 (0%)
4	50	1 (25%)	1 (25%)
7	2,500	4,688 (89%)	4,715 (90%)

7. 결 론

최근 다양한 데이터베이스의 출현에 따라 데이터베이스 복구 방법에 대한 연구가 활발하게 진행되고 있다. 하지만 현재 제시된 대부분의 데이터베이스 포렌식 연구들은 실용적인 측면이 고려되지 않거나 제한적인 복원 결과만을 도출하여 실무에 곧바로 적용될 수 없는 제약이 있다.

본 논문에서는 기존에 밝혀지지 않았던 IBDATA 파일의 구조에 대해 분석하고 이를 기반으로 file-per-table 옵션의 비활성 상태로 운영된 MySQL InnoDB의 데이터베이스를 레코드 단위로 복원하는 알고리즘을 소개했다. 이로써, 기존 MySQL InnoDB의 데이터 저장 형식에 따라 복구를 시도하지 못하는 제한점을 해소할 수 있다.

또한, 본 논문에서 제시하는 복구 방법은 기존 연구된 기법에서 더 나아가 할당 영역뿐만이 아닌 미할당 영역에서도 복원 절차를 수행함으로써 데이터 파일 내의 모든 삭제 레코드의 복원이 가능하다. 해당 알고리즘의 효율성을 검증하기 위해 이를 도구로 구현하여 실제 서버에서 사용되었던 데이터베이스를 대상으로 삭제 및 복원 실험을 진행하였다. 그 결과, 또 다른 데이터로 덮이지 않은 상태로 남아있는 모든 삭제 레코드를 복구할 수 있음을 확인했다. 이러한 복구 기법은 특정 데이터 파일들의 세밀한 구조 파악에 기초하여 수행되므로 신뢰성 있는 결과를 도출할 수 있다. 따라서 본 연구가 MySQL InnoDB를 대상으로 진행되는 데이터베이스 포렌식 조사과정에 크게 기여할 수 있을 것으로 기대된다.

References

[1] D. C. Lee and S. J. Lee, “Research of organized data extraction method for digital investigation in relational database system,” *Journal of the Korea Institute of Information Security and Cryptology*, Vol.22, No.3, pp.565-573, 2012.

[2] R. Harris, “Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem,” *Digital Investigation*, Vol.3, pp.44-49, 2006.

[3] Solid IT, DB-Engines Ranking [Internet], http://db-engines.com/en/ranking

[4] Oracle, Benefits of Using InnoDB Tables [Internet], https://dev.mysql.com/doc/refman/5.7/en/innodb-benefits.html

[5] H. K. Khanuja and D. S. Adane. “A framework for database forensic analysis,” *Computer Science & Engineering*, Vol.2, No.3, p.27, 2012.
[6] P. Frühwirt, M. Huber, M. Mulazzani, and E. R. Weippl, “Innodb database forensics,” *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on.*, IEEE, pp.1028-1036, 2010.
[7] P. Frühwirt, P. Kieseberg, S. Schrittwieser, M. Huber, and E. Weippl, “InnoDB database forensics: Enhanced reconstruction of data manipulation queries from redo logs,” *Information Security Technical Report*, Vol.17, No.4, pp.227-238, 2013.
[8] J. Wagner, A. Rasin, and J. Grier, “Database image content explorer: Carving data that does not officially exist,” *Digital Investigation*, Vol.18, pp.S97-S107, 2016.
[9] 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,” *Journal of the Korea Institute of Information Security & Cryptology*, Vol.26, No.1, pp.125-134, 2016.
[10] J. Jang, D. Jeoung, and S. J. Lee, “The Recovery Method for MySQL InnoDB Using Feature of IBD Structure,” *KIPS Transactions on Computer and Communication Systems*, Vol.6, No.2, pp.59-66, 2017, DOI: 10.3745/KTCCS.2017.6.2.059.
[11] Oracle, InnoDB Tablespaces [Internet], https://dev.mysql.com/doc/refman/5.7/en/innodb-tablespace.html

정 성 균
http://orcid.org/0000-0002-5134-153X
e-mail : skjung@korea.ac.kr
2015년 강원대학교 컴퓨터정보통신공학과 (학사)
2015년～현 재 고려대학교 정보보호대학원 정보보호학과 석사과정
관심분야 : Digital Forensic, Information Security, Machine Learning

장 지 원
http://orcid.org/0000-0002-7142-7744
e-mail : jeewon0838@gmail.com
2015년 국민대학교 컴퓨터공학(학사)
2015년～2017년 고려대학교 정보보호대학원 정보보호학과(석사)
관심분야 : Digital Forensic & Information Security

정 두 원
http://orcid.org/0000-0002-2997-2335
e-mail : dwjung77@gmail.com
2011년 고려대학교 산업경영공학과(학사)
2011년～현 재 고려대학교 정보보호대학원 정보보호학과 석·박사통합과정
관심분야 : Digital Forensic, Information Security, Big Data Analysis

이 상 진
http://orcid.org/0000-0002-6809-5179
e-mail : sangjin@korea.ac.kr
1987년 고려대학교 수학과(학사)
1989년 고려대학교 수학과(석사)
1994년 고려대학교 수학과(박사)
1989년～1999년 ETRI 선임연구원
1999년～현 재 고려대학교 정보보호대학원 교수
2008년～현 재 고려대학교 디지털포렌식연구센터 센터장
2017년～현 재 고려대학교 정보보호대학원 원장
관심분야 : Digital Forensic, Steganography, Hash Function