Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.26, NO.1, Feb. 2016 ISSN 2288-2715(Online)
http://dx.doi.org/10.13089/JKIISC.2016.26.1.125

MySQL MyISAM 데이터베이스의 삭제 레코드에 대한 복구 기법*

노 우 선,† 장 성 민, 강 철 훈, 이 경 민, 이 상 진‡
고려대학교 정보보호대학원

The Method of Deleted Record Recovery for MySQL MyISAM Database*

Woo-seon Noh,† Sung-min Jang, Chul-hoon Kang,
Kyung-min Lee, Sang-jin Lee‡
Center for Information Security Technologies, Korea University

요 약

MySQL 데이터베이스는 현재 데이터베이스 시장에서 높은 점유율을 보이며 많은 사용자들이 사용한다. MyISAM 스토리지 엔진의 경우, 이전에 디폴트 엔진으로 사용되었음에도 불구하고 실질적인 레코드 복구 방법이 존재하지 않았다. 삭제된 레코드에는 데이터베이스 수사 시 중요한 증거로 쓰일 수 있는 정보가 존재할 가능성이 높으며, 수사관이 방대한 양의 데이터베이스를 직접 조사하여 일일이 정보를 판별하는 것은 거의 불가능하다. 본 논문에서는 MySQL MyISAM 데이터베이스 구조를 분석하여 삭제된 레코드의 복구 기법을 제안하며 이를 도구로 구현하여 실험한 결과를 제시한다.

ABSTRACT

MySQL database is currently used by many users and It has gained a big market share in the database market. Even though MyISAM storage engine was used as a default storage engine before, but records recovery method does not existed. Deleted records have a high possibility for important evidence and it is almost impossible to determine that investigators manually examine large amounts of database directly. This paper suggests the universal recovery method for deleted records and presents the experimental results.

Keywords: MySQL database analysis, MyISAM storage engine, digital forensics, record recovery

I. 서 론

현대 사회에서는 방대한 양의 데이터를 효율적이고, 체계적으로 관리하기 위해 데이터베이스를 사용한다. 데이터베이스에는 운용목적과 관련된 대부분의 정보가 기록된다. 이 중에는 수사 시 도움이 될 수 있는 핵심 정보가 존재할 가능성이 매우 높으며, 결정적인 증거로써 수사에 활용될 수 있다.

데이터베이스를 수사관이 압수하여 수색하게 될 경우, 해당 데이터베이스 관리자가 사전에 결정적인 정보를 삭제하여 증거를 인멸하려 할 가능성이 있다. 데이터베이스 내 정보 삭제 여부가 수사에 막대한 영향을 끼치며, 전혀 다른 결과를 초래할 수 있기 때문에 삭제된 레코드를 복구하는 연구가 필요하다.

현재 세계 데이터베이스 시장에서 MySQL 데이터베이스의 점유율 순위는 2위로써 24%의 점유율을

Received(10. 07. 2015), Modified(1st: 11. 27. 2015, 2nd: 12. 24. 2015), Accepted(12. 28. 2015)
* 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단-공공복지안전사업의 지원을 받아 수행된 연구임(2012M3A2A1051106)
† 주저자, unity8891@korea.ac.kr
‡ 교신저자, sangjin@korea.ac.kr(Corresponding author)

차지하고 있다[1]. 이 수치로 보아 MySQL 데이터베이스를 많은 사람들이 널리 쓰고 있다는 것을 알 수 있다. 또한, MySQL의 주요 스토리지 엔진(storage engine)으로 MyISAM과 InnoDB 엔진이 있다. 이 중 MyISAM 엔진은 MySQL 5.4 버전까지 디폴트(default) 스토리지 엔진으로 사용되어 그 중요성이 높다.

현재 MySQL MyISAM 데이터베이스의 중요성이 높음에도 불구하고, 수사에 도움이 될 수 있는 실질적인 레코드 복구에 대한 연구가 전무하다.

본 연구는 MySQL MyISAM 데이터베이스의 삭제된 레코드를 복구하여 데이터베이스 수사 시 놓칠 수 있는 결정적 증거를 찾는데 의의가 있다. 또한, 본 연구는 MySQL 5.6.13 버전(version)을 기준으로 진행되었다. MySQL 5.0.37-5.6.13 버전에서 복구방법이 적용 가능하였고, 다른 버전에서도 추후 실험이 필요하다.

II. 관련 연구

2.1 MySQL 데이터베이스 연구 동향

현재까지 MySQL 데이터베이스는 InnoDB 엔진에 치중되어 연구되었다. P. Fruhwirt와 M. Huber(2010)의 연구는 MySQL 5.1.32 버전을 기준으로 InnoDB 엔진을 사용한 MySQL 데이터베이스의 구조를 포렌식 관점에서 분석하였다[2]. MyISAM이나 InnoDB엔진을 사용하여 테이블을 생성할 때, 폼(form) 파일(FRM)이 공통적으로 생성된다. 이 연구에서 MySQL 5.1.32 버전의 폼 파일 구조와 각 필드정보를 확인할 수 있다. 또한 P. Fruhwirt 등[3]은 MySQL 5.1 버전을 기준으로 InnoDB 엔진의 Redo logs에서 사용자가 과거에 행했던 데이터 삭제 및 업데이트 등의 쿼리를 복구하여 확인하는 방법에 대해 연구하였다. 이는 사용자의 행위를 추적하여 수사에 도움이 될 수 있다.

최근의 연구는 레코드에 직접적으로 접근하여 레코드를 복구하려는 경향을 보인다. 남궁재웅 등의 연구(2013)[4]는 MyISAM과 InnoDB 엔진을 이용한 테이블의 레코드를 삭제했을 때의 레코드를 분석하고 레코드를 복구하는 방법을 제안하였다. 하지만 레코드를 복구하는데 필요한 정보의 위치가 가변적이므로 정확한 정보를 획득하기 어렵고, MyISAM 테이블의 삭제된 레코드의 내용을 컬럼 단위까지 파싱하는 것이 불가능한 문제점이 있었다. 장성민의 연구(2015)[5]는 InnoDB 엔진의 구조를 세밀하게 분석하여 실질적으로 레코드를 복구하는 방법을 설명하였다. 최근까지 MyISAM 엔진에 대한 레코드 복구 연구가 InnoDB에 비해 미진했던 이유는 MyISAM 엔진은 레코드 삭제 시 레코드의 앞부분을 삭제에 관련된 데이터로 덮어쓰기 때문에 레코드의 헤더가 훼손되어 복구가 어렵기 때문이다.

앞서 살펴본 내용과 같이 아직까지 MyISAM 엔진의 구조를 세밀하게 분석하여 삭제 레코드를 컬럼 단위로 복구하는 방법이 없으므로 MyISAM 엔진의 삭제된 레코드 복구 연구가 필요하다.

2.2 MySQL 데이터 타입(Data Type)

MySQL은 각 컬럼(column) 데이터 타입에 따라 다른 방식으로 레코드에 그 값을 저장한다. Oracle은 MySQL 데이터베이스의 내부구조에 대한 문서를 공개 하였고 2015년에 마지막으로 업데이트 하였다[6]. 이 문서에서 각 데이터 타입 별 저장 방식에 대해 알 수 있다.

각 데이터 타입 별 입력 가능한 최소, 최댓값의 범위와 데이터 입력 및 표시 방식은 "MySQL Developer's Library"를 참고하였다[7].

III. MySQL MyISAM 데이터베이스 구조 분석

남궁재웅 등의 연구(2013)[4]를 통해 기본적인 MySQL 데이터베이스의 구조, MyISAM 테이블의 종류, 테이블을 새로 만들 때 생성되는 파일(FRM, MYI, MYD)과 그 역할을 간단히 알 수 있다. 다음 소단원에서는 직접 실험을 통해 분석된 MyISAM 테이블의 각 구성파일(FRM, MYI)의 구조와 삭제 레코드에 대한 설명을 하도록 한다.

3.1 FRM 파일 분석

MySQL의 레코드 복구를 위해 스토리지 엔진의 종류, 테이블의 포맷 종류, MySQL 버전과 테이블의 스키마 영역(table schema area)의 위치 확인이 필요하다. 이 정보는 FRM 파일에서 확인할 수 있다. Table 1.은 FRM 파일에서 레코드 복구에 필요한 정보의 오프셋과 길이를 나타내며, 모든 정보

Table 1. Field information for recovery in FRM file

Offset	Length	Field Name	Description
0x03	1	storage engine	0x09 : MyISAM 0x0C : InnoDB
0x1E	2	table option	odd value : fixed table even value : dynamic table
0x33	2	MySQL version	convert to decimal number for interpret e.g.) 0xC5B5(50,613) = 5.6.13 ver
0x44	2	table schema area's offset	multiply by 0x100 for interpret

는 리틀 엔디안(little endian)으로 해석한다.

스토리지 엔진의 값이 0x09일 때 MyISAM 엔진을, 0x0C일 때 InnoDB 엔진을 사용하여 생성된 테이블임을 의미한다. 테이블 옵션의 값이 홀수일 때 정적 테이블(fixed table)을, 짝수일 때 동적 테이블(dynamic table)을 의미한다. MySQL 버전의 값을 10진수로 변환한 값에서 앞의 2자리는 메이저 버전(major version)으로, 뒤의 3자리는 마이너 버전(minor version)으로 해석한다. 테이블 스키마 영역의 오프셋 값에 0x100을 곱해야 실제 테이블 스키마 영역의 오프셋을 얻을 수 있다.

3.1.1 테이블 스키마 영역

테이블 스키마 영역은 컬럼의 개수, 컬럼의 이름과 각 컬럼의 메타데이터(metadata)가 저장되어 있다. 테이블 스키마 영역의 구조는 Fig.1.과 같다.

테이블 스키마 영역에는 헤더(header), 컬럼 이름 섹션 1(column name section 1), 컬럼 메타데이터 섹션(column metadata section), 컬럼 이름 영역(column name section 2), Set과 Enum 데이터 타입의 메타데이터 섹션(Set & Enum metadata section)이 순서대로 저장되어 있다. 테이블 스키마 영역의 크기가 굉장히 클 경우 컬럼 이름 섹션 1은 존재하지 않을 수 있다.

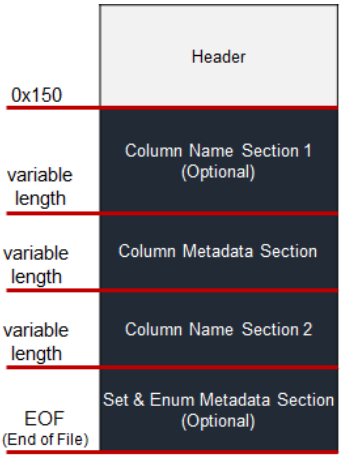

Fig. 1. Structure of table schema area

3.1.1.1 테이블 스키마 영역의 헤더

테이블 스키마 영역의 헤더에는 테이블 스키마 영역의 크기와 컬럼의 개수가 저장되어 있다. Table 4.는 이 정보의 테이블 스키마 영역 내 오프셋과 길이를 나타낸 표이며, 리틀 엔디안으로 해석한다.

Table 2. Field information in table schema area's header

Offset	Length	Field Name	Description
0x00	2	length	table schema area's length
0x02	2	column num	number of columns

3.1.1.2 컬럼 이름 섹션 1

컬럼 이름 섹션 1은 테이블 스키마 영역의 오프셋 0x150에 위치한다. 이 섹션은 컬럼의 개수만큼 컬럼 이름세트(column name set)가 연속적으로 나열되어 저장되어 있다. Fig.2.는 컬럼 이름 섹션 1에 저장된 컬럼 이름 세트의 구조를 보여준다.

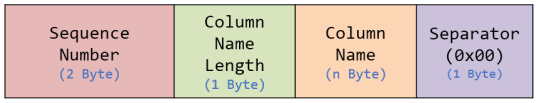

Fig. 2. Structure of column name set in Column Name Section 1

시퀀스 번호(sequence number)는 0x04부터 시작되어 0x16까지 증가하는 번호이다. 첫 번째 컬럼 이름에는 0x04가 부여되고, 두 번째 컬럼에는 0x05가 부여되는 방식으로 증가한다. 0x16 이후의 번호는 다시 0x04부터 할당하여 반복한다. 컬럼 이름의 길이(column name length)는 컬럼의 이름에 구분자의 길이까지 합한 길이이다. 이후 컬럼의 이름이 위치하며, 마지막으로 "0x00" 값의 구분자(separator)가 위치한다.

컬럼 이름 세트는 19개가 모여 하나의 집합을 이루며, 집합과 집합 사이에는 집합에 관한 0x30 바이트의 메타데이터가 저장된다. 컬럼 이름 섹션 1의 크기가 클 경우, 모든 집합의 메타데이터가 연속적으로 나열되고 그 이후 각 컬럼 이름 세트가 나열될 수 있다. 이는 컬럼 메타데이터 섹션의 위치를 파악하는데 영향을 미치므로 중요하다.

3.1.1.3 컬럼 메타데이터 섹션

컬럼 이름 섹션 1 이후에 컬럼 메타데이터 섹션이 위치한다. 이 섹션은 컬럼 개수만큼 컬럼 메타데이터(column metadata)가 연속적으로 나열되어 저장되어있다. 컬럼 메타데이터는 0x11 바이트의 길이로 데이터 타입, 컬럼 데이터 오프셋 등의 정보가 저장되어 있다. Fig.3.은 컬럼 메타데이터의 구조를 나타낸 그림이다. Table 3.은 컬럼 메타데이터 내 레코드 복구와 관련된 주요 필드를 설명한 표이다.

컬럼 할당 길이(column allocation length)와 팩 플래그(pack flag)는 decimal 데이터 타입의 해석을 위해 쓰인다. 데이터 오프셋+1(data offset +1) 필드는 정적 테이블 내 해당 컬럼의 데이터 오프셋을 가르키며 필드 값에 1을 빼주어야 정확한 오프셋을 알 수 있다. 인터벌 ID(interval ID)는 set과 enum 타입의 멤버 집합 ID(member set ID)를 뜻하며 Set & Enum meatadata section에 존재한다. 데이터 타입 필드는 콜레이션(collation)과 함께 데이터의 타입을 판별하는데 사용된다. 콜레이션은 컬럼 데이터의 문자 형식을 뜻하며 "show COLLATION" 쿼리를 이용하여 콜레이션의 종류 및 ID 값을 확인할 수 있다[8]. ID가 "0x21" 값일 때 문자형을, "0x3F" 값일 때 바이너리를 뜻한다.

Table 3. Field information for recovery in column metadata

Offset	Length	Field Name	Description
0x03	2	column allocation length	generally it is used for digit indication but this field is needed to interpret decimal data type
0x05	3	data offset+1	used only for fixed table
0x08	2	pack flag	generally it is used for column option but this field is needed to interpret decimal data type
0x0C	1	interval ID	set & enum's member set ID in Set & Enum metadata section
0x0D	1	data type	column's data type
0x0E	1	collation	it determine data type is binary or text

3.1.1.4 컬럼 이름 섹션 2

컬럼 이름 섹션 2는 컬럼 메타데이터 섹션 다음에 위치한다. 이 섹션은 컬럼의 개수만큼 "0xFF" 값의 구분자와 컬럼 이름이 연속적으로 나열되어 저장되어 있다.

3.1.1.5 Set & Enum 메타데이터 섹션

Set & Enum 메타데이터 섹션은 set과 enum 타입의 멤버 집합이 연속적으로 나열되어 있다. set과 enum 타입의 멤버 집합이 저장된 섹션으로 멤

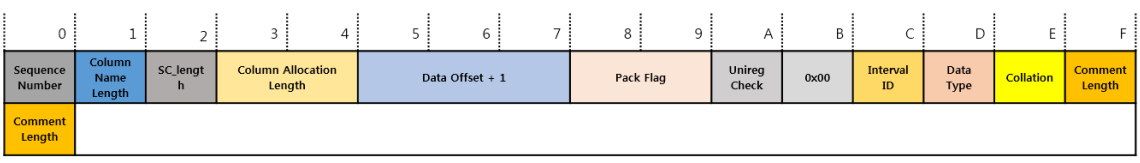

Fig. 3. Structure of column metadata

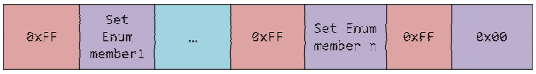

Fig. 4. Structure of Set & enum member set

버 집합의 구조는 Fig.4.와 같다.

각 멤버 앞에 구분자로 "0xFF"가 위치하며 멤버 집합 마지막엔 멤버 집합의 끝을 알리는 구분자와 "0x00" 값이 위치하는 것을 알 수 있다. Fig.5.는 Set과 Enum 타입을 이용한 테이블 생성문이며, Fig.6.은 테이블이 생성된 후 Set & Enum metadata section의 값을 나타낸 그림이다.

Fig.6.에서 (a, b, c), (a, b, d), (a, b, e) 멤버 집합이 나열되어 있다. 이 멤버 집합은 순서대로 1, 2, 3의 값이 부여되어 컬럼 메타데이터의 인터널 ID 필드에 사용된다.

```
CREATE TABLE `set_enum_test` (
  `enum_1` enum('a', 'b', 'c'),
  `set_1` set('a', 'b', 'd'),
  `set_2` set('a', 'b', 'e'),
  `set_3` set('a', 'b', 'c')
) ENGINE=MyISAM ROW_FORMAT= FIXED DEFAULT CHARSET=utf8;
```

Fig. 5. Set & Enum table create query

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000021B0   08 82 11 00 01 F8 21 00 00 FF 65 6E 75 6D 5F 31  .,...ø!..ÿenum_1
000021C0   FF 73 65 74 5F 31 FF 73 65 74 5F 32 FF 73 65 74  ÿset_1ÿset_2ÿset
000021D0   5F 33 FF 00 FF 61 FF 62 FF 63 FF 00 FF 61 FF 62  _3ÿ.ÿaÿbÿcÿ.ÿaÿb
000021E0   FF 64 FF 00 FF 61 FF 62 FF 65 FF 00             ÿdÿ.ÿaÿbÿeÿ.
```

Fig. 6. Set & Enum metadata section

3.2 MYI 파일 분석

MYI 파일에서 삭제된 레코드의 개수와 마지막으로 삭제된 레코드의 오프셋을 획득할 수 있다. Table 4.는 이 정보의 MYI 파일 내 오프셋과 길이를 나타낸 표이며, 빅 엔디안으로 해석한다.

Table 4. Field information in table schema area's header

Offset	Length	Field Name	Description
0x24	8	deleted records	number of deleted records
0x34	8	deleted record link	last deleted record's offset

3.3 레코드 분석

3.3.1 정적 레코드(fixed record)

정적 레코드는 정적 테이블에 존재하는 레코드로써 헤더와 데이터 부분으로 구성되며 모든 레코드의 크기가 같다. 최소 레코드의 크기는 7바이트이다. 헤더의 최하위 비트에는 레코드의 삭제유무 비트가 위치한다. 레코드가 삭제되면 0으로, 삭제되지 않으면 1로 저장된다. 이후에는 각 컬럼의 값이 "null" 값인지 아닌지 그 여부를 표현하는 널 비트맵(null bitmap)이 위치한다. 해당 컬럼이 "null" 값인 경우 1로, "null" 값이 아닌 경우 0으로 저장된다. 최하위 비트부터 최상위 비트 순으로 첫 번째 컬럼부터 마지막 컬럼의 "null" 값 유무가 저장되며, "null" 값을 저장할 수 없는 컬럼의 경우 제외되어 비트맵에 저장되지 않는다. 삭제유무 비트와 NULL 비트맵의 크기가 바이트로 떨어지지 않는 경우 1비트로 패딩(padding)된다. 헤더를 제외한 데이터 부분에는 각 컬럼의 데이터가 순차적으로 저장된다.

3.3.2 동적 레코드(dynamic record)

동적 레코드는 동적 테이블에 존재하는 레코드로써 헤더, 스킵(skip) 비트맵, 널 비트맵, 데이터와 미사용 영역(unused space)으로 구성된다. 최소 레코드의 크기는 20바이트이며, 동적 테이블의 레코드는 13가지 종류가 존재한다[9]. 레코드의 종류에 따라 헤더의 크기와 헤더 내 구성 및 레코드의 최대 크기가 다르다. 헤더에는 헤더 타입, 레코드의 길이가 저장되며, 하나의 레코드가 여러 개의 블록으로 나누어 저장된 경우 인접 블록의 오프셋이 저장된다.

스킵 비트맵은 문자형 데이터 타입이 "null" 값이나 공백 값을 가질 경우, 숫자형 컬럼의 데이터 타입이 "null", "0", 공백 값을 가질 경우 MYD 파일에 데이터를 할당하지 않고 비트맵으로 이를 표현한다. 스킵 비트맵은 모든 데이터 타입마다 적용되지 않으며, 예외 데이터 타입은 Table 7.와 같다.

널 비트맵은 정적 테이블의 널 비트맵과 동일하다. 이후 각 컬럼의 데이터가 순차적으로 저장된다. 레코드의 타입에 따라 데이터 영역 이후에 미사용 영역이 존재할 수 있다. 미사용 영역은 "0x00" 값으로 채워져 있으며 레코드의 크기가 20바이트가 안되거나 레코드간 간격을 맞추기 위하여 존재한다.

Table 5. Exception data type of the skip bitmap

Data Type
varchar
varbinary
decimal
time
datetime
timestamp

3.4 삭제 레코드 분석

3.4.1 정적 테이블의 삭제 레코드

정적 테이블의 레코드 삭제 시 레코드 앞 7바이트가 삭제와 관련된 정보로 덮어쓰인다. Fig 7.은 7바이트의 삭제 시 덮어쓰이는 정보에 관한 그림이다.

첫 1바이트는 삭제된 레코드임을 나타내는 삭제 시그니처(signature)로 항상 "0x00" 값을 가진다. 이후 6바이트는 이전에 삭제된 레코드의 인덱스가 저장된다. 인덱스는 첫 번째 레코드를 뜻하는 "0x00"에서 시작하여 순차적으로 증가한다. 이전에 삭제된 레코드가 없을 경우 "0xFF" 값으로 채운다.

Fig. 7. Structure of fixed table's deleted record header

3.4.2 동적 테이블의 삭제 레코드

동적 테이블의 레코드 삭제 시 레코드 앞 20바이트가 삭제와 관련된 정보로 덮어쓰인다. Fig. 8.은 삭제 시 덮어쓰이는 20바이트에 관한 그림이다.

첫 1바이트는 삭제된 레코드임을 나타내는 삭제 시그니처로 항상 "0x00" 값을 가진다. 이후 3바이트는 삭제된 레코드의 길이를 나타낸다. 이후 8바이트는 현제 레코드 이전에 삭제된 레코드의 오프셋을 가르키며 마지막 8바이트는 이후에 삭제된 레코드의 오프셋을 가르킨다. 이전이나 이후에 삭제된 레코드가 없을 경우 "0xFF" 값으로 채운다.

Fig. 8. Structure of dynamic table's deleted record header

3.5 길이 지시자(length indicator)

길이 지시자는 데이터의 길이가 동적으로 변하는 동적 데이터 타입의 데이터 앞에서 뒤에 따르는 실제 데이터의 길이를 알려준다. 길이 지시자가 붙게 되는 데이터 타입과 그 세부사항은 Table 8.과 같다.

Table 6. Dynamic data types attached length indicator

Offset	Length(byte)	Endian
varchar(n)	column allocation length * collation size 255 and under : 1 over 255 : 2	big
varbinary(n)		
tinyblob	1	little
tinytext		
blob	2	little
text		
mediumblob	3	little
mediumtext		
longblob	4	little
longtext		

IV. 삭제된 레코드 복구

레코드 복구를 하기 위해 MYI 파일에서 삭제된 레코드 개수를 확인하고, 삭제된 레코드가 존재하면 FRM 파일에서 앞서 설명한 복구에 필요한 정보들을 획득하여 테이블의 포맷에 따라 복구한다.

4.1 정적 테이블의 레코드 복구

정적 레코드는 삭제 시 레코드의 첫 7바이트가 삭제와 관련된 정보로 덮어쓰이므로 7바이트 이후부터 복구가 가능하다. Fig.3.의 컬럼 메타데이터 내 data offset+1 필드를 참조하여 각 컬럼 데이터 타입의 크기만큼 파싱하여 복구할 수 있다. 이전에 삭제된 레코드의 인덱스가 "0xFF" 값으로 채워져 있을 때 까지 반복하여 삭제된 레코드를 복구한다.

4.2 동적 테이블의 레코드 복구

동적 레코드는 삭제 시 레코드의 첫 20바이트가 삭제와 관련된 정보로 덮어쓰인다. 그러므로 삭제 전

의 레코드 타입을 알 수 없어 레코드의 헤더 크기 또한 파악할 수 없다. 헤더의 크기를 파악하지 못하면 스킵 비트맵, 널 비트맵 및 데이터의 시작 위치를 알 수 없다. 레코드의 헤더 크기가 4~16 바이트라는 것을 생각하면 삭제 시 덮어쓰이는 20바이트에 스킵 비트맵은 물론이고, 널 비트맵과 데이터까지 훼손될 가능성이 매우 높다[9].

동적 레코드를 복구하기 위해 삭제된 레코드의 크기와 각 컬럼 데이터 크기의 합을 대조하는 방법으로 복구를 진행한다. 동적 데이터 타입의 경우 길이 지시자를 이용하여 데이터의 크기를 파악할 수 있고, 동적 데이터 타입이 아닌 경우 Table 6.을 참고하여 데이터의 크기를 파악할 수 있다. 또한, 동적 레코드의 미사용 영역을 고려해야 한다. 데이터 영역 이후부터 삭제 레코드의 끝까지 "0x00" 값으로 채워져 있다면 이를 미사용 영역으로 볼 수 있다. 그러므로 삭제된 레코드의 크기는 다음 식(1)과 같다.

삭제된 레코드의 크기 =
20 + 데이터의 크기 + 미사용 영역의 크기 (1)

동적 레코드를 복구하는 과정은 다음과 같다.

[1] 완전히 복구 가능한 컬럼 데이터의 개수가 N개라고 가정한다.
[2] 오프셋 20을 기준(pivot)으로 첫 번째 컬럼 데이터가 시작한다고 가정한다.
[3] 기준으로부터 복구 가능한 컬럼의 각 데이터 크기의 합을 구한다.
[4] 미사용 영역을 확인한다.
[5] 식(1)을 만족하면 레코드를 복구하고, 만족하지 않으면 기준을 1증가시킨다.
[6] 기준이 레코드의 크기와 같으면 복구 가능한 컬럼의 개수를 1감소시키고, 기준을 다시 오프셋 20으로 설정한다. 과정[3]으로 돌아간다.
[7] 기준이 레코드의 크기보다 작으면 기준을 1 증가시키고 과정[3]으로 돌아간다.

동적 레코드 복구 시 과정[3]에서 각 데이터의 크기를 구할 때 스킵 된 데이터가 있을 수 있으므로 각 경우의 수를 고려하여 데이터의 크기를 구해야 한다. Fig.9.는 MySQL MyISAM 데이터베이스의 레코드 복구 순서도이다.

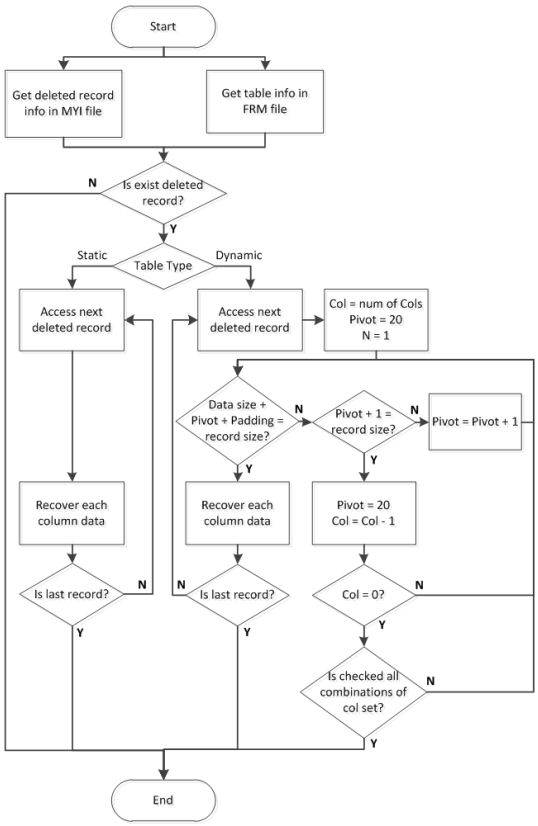

Fig. 9. Recovery procedure for deleted records in the MySQL MyISAM database

V. 구현 및 성능 평가

앞에서 제시한 복구 방법을 검증하기 위해 데이터베이스 정적 테이블 및 동적 테이블의 삭제 레코드를 복구하는 도구를 구현했다.

구현한 도구의 정확성 여부를 평가하기 위해 직접 테이블을 생성한 후, 데이터를 삭제하여 복구된 레코드와 삭제한 레코드가 일치하는지 비교하였다.

5.1 실험 환경

테이블의 각 컬럼은 int, int, int, varchar, time, longtext, float, float, timestamp, blob, blob, blob 타입을 사용하여 테이블을 생성했다. 정적 테이블의 경우 longtext대신 varchar를, blob 대신 varbinary를 사용했다. 레코드의 크기는 4,096 바이트로 설정했으며, 데이터의 크기가 동적일 수 있는 타입은 크기가 모두 같도록 분배했

다. 각 컬럼의 데이터는 20% 확률로 null 값을, 나머지의 경우 랜덤 값을 입력하여 삭제한 뒤 복구하는 작업을 100회 반복했다.

5.2 실험 결과

Table 7.은 정적 테이블과 동적 테이블의 복구된 레코드 개수와 정확하게 복구된 레코드 개수를 나타낸 표이다.

정적 테이블의 경우 100%의 복구율을 보였다. 동적 테이블은 모든 레코드를 복구했지만, 스킵 데이터로 인한 오탐으로 97개의 레코드만 정확히 복구했다. 결과적으로 정적 테이블은 각 레코드의 오프셋을 알 수 있기 때문에 100% 복구할 수 있지만, 동적 테이블은 각 레코드의 오프셋을 알 수 없으므로 복구 시 오탐 확률이 있다.

Table 7. Accuracy of deleted record recovery

Table format	Recovered	Exactly recovered
fixed	100	100
dynamic	100	97

5.3 실데이터 복구

가상의 환경이 아닌 실제 게시판 서비스를 운용중인 데이터베이스로 레코드 복구 실험을 했다. 테이블은 4개로써, 모두 동적 테이블이며 컬럼 개수는 12, 22, 31, 47개이다. 각 테이블을 최적화하여 단편화를 제거한 뒤, 각 테이블의 레코드를 100개씩 선별 삭제하여 삭제 데이터를 기록하였다. 이를 복구 도구를 이용하여 복구한 뒤 기존에 기록해 둔 삭제 데이터와 비교하는 방식으로 복구율을 측정하였다. Table 8.은 각 테이블별 복구 개수와 오탐, 미탐 개수를 보여준다. Complete는 삭제된 20바이트를 제외한 컬럼의 데이터를 완전히 복구한 것을 의미한다. Partial은 일부 컬럼을 제외하고 뒤쪽 컬럼의 데이터를 일부 복구한 것을 의미한다. False Positive는 오탐을, False Negative는 미탐을 의미한다.

실데이터 복구 결과 컬럼이 12개인 경우 완전 복구한 레코드의 개수가 54개이지만 오탐이 34개로 오탐율이 높은 것을 알 수 있다. 컬럼이 12개인 테이블을 제외하면 완전 복구에 성공한 테이블은 없었고 부분 복구는 할 수 있었다. 컬럼이 31개인 테이블과 컬럼이 47개인 테이블을 비교하면 컬럼이 47개인 테이블의 오탐이 31개인 테이블에 비해 매우 낮고 12개의 컬럼을 가진 테이블과 같은 것을 확인할 수 있다. 이는 31개의 컬럼을 가진 테이블의 평균 삭제 레코드 크기가 약 1MB로 47개의 컬럼을 가진 테이블의 삭제 레코드 크기보다 약 10배 가량 컸기 때문이다. 여기서 복구율에 미치는 영향이 컬럼의 개수보다 삭제 레코드의 크기가 더 큰 영향을 미치는 것을 알 수 있었다. 복구 결과 오탐율이 상대적으로 높은 이유는 레코드 끝 부분의 제로 패딩(zero padding)과 전수 조사를 하여 복구 가능한 모든 스키마를 수사관에게 제시하여 선택하게 하는 방법이 이상적이지만 실제로는 컴퓨터 파워의 부족으로 많은 시간이 걸려 모든 스키마를 제시할 수 없으므로 오탐율이 상대적으로 높았다.

Table 8. Result of deleted record recovery

Col num	12	22	31	47
Complete	54	0	0	0
Partial	12	99	33	66
False Positive	34	1	66	34
False Negative	0	0	0	0

VI. 결론 및 향후 연구

MySQL MyISAM 데이터베이스는 많은 사용자의 주요 기록들을 저장, 관리하는 목적으로 사용되기 때문에 이에 대한 연구는 포렌식 관점에서 중요하다. MySQL MyISAM 데이터베이스의 레코드 복구 연구는 현재까지 거의 진행되지 않아 수작업으로 레코드를 복구해야 했다. 레코드 복구는 포렌식 관점에서 복구에 소요되는 시간보다 결정적인 증거가 복구 될 수 있는지에 대한 여부가 중요하다. 본 연구는 대용량의 데이터베이스에서 실질적으로 레코드 복구가 가능하다는 것을 보여준다. 또한, 처음으로 MySQL MyISAM 데이터베이스에서 실질적인 레코드 복구를 시도했다는 것에 큰 의미가 있다. 본 연구에서 제시한 알고리즘은 기초적인 알고리즘이며, 향후 각 데이터 타입의 특성을 고려하여 더 빠르게 복구할 수 있도록 연구가 진행 중이다. 또한, 동적 테이블의 헤더와 비트맵의 크기를 계산하여 더 효율적으로 기준

점을 제어하기 위한 연구를 하고 있다. 마지막으로 인덱스 파일인 MYI 파일에서 인덱스가 설정된 컬럼의 데이터를 복구하는 연구를 진행할 예정이다.

Appendix. A. 콜레이션 크기(collation size)

콜레이션의 문자 저장 크기는 문자열 타입의 데이터가 MYD 파일에 저장될 때 관여한다. 예를 들어 10바이트의 문자열을 저장할 때 콜레이션 크기가 3인 utf8 콜레이션을 사용할 경우 30바이트를, 콜레이션 크기가 2인 euckr 콜레이션을 사용할 경우 20바이트를 MYD 파일에 저장한다.

References

〔1〕 Solid IT, DB-Engines Ranking, "http://db-engines.com/en.ranking"

〔2〕 P. Fruhwirt and M. Huber, "InnoDB database forensics," 24th IEEE International Conference on Advanced Information Networking and Applications (AINA). pp. 1028-1036, Apr. 2010

〔3〕 P. Fruhwirt, P. Kieseberg, S. Schrittwieser, M. Huber and E. Weippl, "InnoDB database forensics:Reconstructing data manipulation queries from redo logs," Seventh International Conference on Availability, Reliability and Security (ARES), pp. 625-633, Aug. 2012

〔4〕 Jae-ung Namgung, Nam-heun Son, Sang-jun Jeon, Cheul-hoon Kang and Sang-jin Lee, "The scheme for recovering deleted records in MySQL database," Journal of Korean Digital Forensics Society, 7(1), pp. 1-14, Dec. 2013

〔5〕 Sung-min Jang, "Deleted records recovery research for MySQL InnoDB," Master's Thesis, Korea University, Feb. 2015

〔6〕 Oracle, MySQL Internals Manual, "https://dev.mysql.com/doc/internals/en"

〔7〕 Paul DuBois, MySQL Developer's Library, 4th Ed., Addison-Wesley Professional, Aug. 2008

〔8〕 Oracle, SHOW COLLATION Syntax, "https://dev.mysql.com/doc/refman/5.0/en/show-collation.html"

〔9〕 Oracle, Layout of the Record Storage Frame, "https://dev.mysql.com/doc/internals/en/layout-record-storage-frame.html"

〈저 자 소 개〉

노 우 선 (Woo-seon Noh) 학생회원
2014년 2월: 경희대학교 컴퓨터공학과 졸업
2014년 3월~현재: 고려대학교 정보보호대학원 정보보호학과 석사과정
〈관심분야〉 디지털 포렌식, 데이터베이스 포렌식, 모바일 포렌식

장 성 민 (Sung-min Jang) 정회원
2013년 2월: 인제대학교 전자정보대학 컴퓨터공학과 공학사
2015년 2월: 고려대학교 정보보호대학원 정보보호학과 석사
〈관심분야〉 디지털 포렌식, 정보보호, 역공학

강 철 훈 (Chul-Hoon Kang) 정회원
2006년 12월~현재: 대검찰청 디지털포렌식센터 데이터베이스포렌식 팀장
〈관심분야〉 Database Forensic, Accounting Forensic

이 경 민 (Kyung-min Lee) 정회원
2002년 2월: 상명대학교 행정학과 학사
2007년 2월: 동국대학교 국제정보대학원 정보보호학과 석사
2008년 7월~2012년 3월: 서울 방배경찰서 사이버범죄수사팀 근무
2013년 1월~현재: 대검찰청 과학수사부 디지털수사과 근무
〈관심분야〉 데이터베이스 포렌식, 디지털 포렌식, 정보보호

이 상 진 (Sang-jin Lee) 종신회원
1987년 2월: 고려대학교 수학과 학사
1989년 2월: 고려대학교 수학과 석사
1994년 8월: 고려대학교 수학과 박사
1989년 10월~1999년 2월: ETRI 선임 연구원
1999년 3월~2001년 8월: 고려대학교 자연과학대학 조교수
2001년 9월~현재: 고려대학교 정보보호대학원 교수
2008년 3월~현재: 고려대학교 디지털포렌식연구센터 센터장
〈관심분야〉 디지털 포렌식, 심층 암호, 해쉬 함수