Journal of Computational Information Systems 11: 15 (2015) 5415–5423
Available at http://www.Jofcis.com

# A Records Recovery Method for InnoDB Tables Based on Reconstructed Table Definition Files

Pianpian SUN*, Ming XU, Jian XU, Yizhi REN, Haiping ZHANG, Ning ZHENG

*College of Computer, Hangzhou Dianzi University, Hangzhou 310018, China*

**Abstract**

InnoDB is the most widely used storage engine for MySQL. It holds a large amount of information which is of great significance to records recovery. However, existent records recovery method for InnoDB tables gets less or inaccuracy records because of the less or inaccuracy table structures acquired from InnoDB data dictionary. To solve that problem, the proposed method in this paper uses the file carving technology to recover the continuous table definition files from the disk raw image and then the table structures are reconstructed by analyzing the carved table definition files. Finally records can be successfully recovered by applying the table structure on the matched data pages. The experimental results demonstrate that the efficiency of the proposed method with high precision and recall.

*Keywords*: InnoDB; Records Recovery; Table Structure; File Carving

## 1 Introduction

InnoDB is one of the best-known open-source storage engines for MySQL. InnoDB is ACID compliant and provides transactions, row-level locking, MVCC, automatic recovery and data corruption detection. It is widely used in MySQL-based Web, e-commerce, financial systems, health care, and retail applications so a significant amount of valuable information such as patients and customers is stored in InnoDB database. If the tables in InnoDB databases are dropped by a hacker, data recovery from the raw image or related files is very useful for database forensic analysis [1-3]. Unlike file recovery, database recovery is more challenging [4] because of databases' complex structures. Records are recovered with both of the table structure and leaf pages. The accurate table structure is very important to recover records because it can correctly interpret binary records from leaf pages into human-readable information and provide the records' size in order to extract compact records from leaf pages. When the tables are dropped, the table structure can't be gotten via SQL interface. From InnoDB data dictionary existent records recovery methods get less table structures or inaccurate table structure. A new method of recovering

*Corresponding author.
*Email address:* sunpian1@163.com (Pianpian SUN).

1553–9105 / Copyright © 2015 Binary Information Press
DOI: 10.12733/jcis14870
August 1, 2015

records by acquiring the table structure from the carved table definition files in raw image is proposed in this paper.

The rest of this paper is structured as below: Section 2 introduces related work. Section 3 describes structure of the InnoDB tables. Section 4 details the proposed method for recovering InnoDB table records. Section 5 gives a detailed demonstration on the capabilities of the recovery method. Section 6 concludes the paper and gives an outlook to future plans exploring another location storing the table structure to recover records of InnoDB tables.

## 2 Related Work

Clearly, the analysis of the structure of InnoDB database is a precondition for records recovery. While MySQL documentation provides a good starting point [5], it is important to understand in detail how InnoDB database is built. The detailed analysis of InnoDB table space file is described in [6] by Jemery Cole. Peter Frhwirt et al. [1] describe the format of table definition files and leaf pages in InnoDB table space file and show it is possible to recover records with the table structure stored in table definition files. In 2014 they [7] show again it is possible to restore database manipulation statement from InnoDB redo log files with the table structure stored in table definition files. In 2014 [8] they present a new approach for a forensic-aware database management system using transaction and replication sources and provide a prototype implementation in MySQL based on MySQL replication and InnoDB transaction log. Aleksandr Kuzminsky writes a set of open-source tools named percona-data-recovery-tool-for-innodb-0.5 [9] to recover InnoDB table records. It uses two ways to get the table structures. First, a perl script uses SHOW FIELDS FROM statement after connected to MySQL server. The second way is to recover table structure from InnoDB data dictionary. When the table is dropped, the first way can't recover the dropped tables' structures but the second way can recover the dropped tables' structures. Though the second way works, it gets few table structures of all the dropped tables. Aleksandr Kuzminsky releases a new tool named undrop-for-innodb-0.0-34 [10] in 2014. It is an improved version of percona-data-recovery-tool-for-innodb-0.5. It works better than percona-data-recovery-tool-for-innodb-0.5. It can recover the table structures of all the dropped tables from InnoDB data dictionary. InnoDB data dictonary doesn't store all information you can find in the table definition file. For example, InnoDB data dictionary doesn't store DECIMAL type as a binary string. It doesn't store precision of a DECIMAL field. So that information will be lost. The accuracy of the table structure is still the above two tools' problem.

This paper shows it is possible to recover the deleted table definition files of the dropped tables from the disk raw image. To the best of our knowledge, using file carving technology to recover the table definition files has not been done. Based on the reconstructed table definition files, More accurate table structure can be reconstructed.

## 3 InnoDB Tables

For an InnoDB table, the MySQL server keeps the information of table structure in .frm file which is in the database data directory and also keeps all records in a single ibdata1 when innodb_file_per_table is off or in .ibd file when innodb_file_per_table is on. When an InnoDB table is dropped, its .frm file and .ibd file are deleted but not physically. Below is the analysis of the structure of the InnoDB database.

## 3.1 Table definition files

Table definition file stores the information of the table structure. The basic format of the table definition file is shown as Table 1. File header contains general information. Padding bytes are filled with zeros. The key definition block is about the information of the key. Header block is about general information of the column and the remaining three blocks is about specific information of columns. The column definition block is about the column name and ID. The column structure block is about the column's data type. The column name block contains user-defined names of the columns.

## 3.2 Tablespace

Tablespace stores the records of the table. Tablespace file is ibdata1 or .ibd file. A single ibdata1 is also called system tablespace. InnoDB data dictionary in system tablespace stores metadata about user tables. InnoDB data dictionary is composed of some sytem tables. SYS_TABLES provides the information of the table. SYS_INDEXES provides the information of the index. SYS_COLUMNS provide the information of table columns. SYS_FIELDS provide the information about columns belonging to the index. Tablespace is composed of 16KB database pages. Leaf pages' format is shown as Table 2. A page has two header/trailer pairs. The inner pair, "page header" and "page directory", are mostly the concern of the inner status information of pages, while the outer pair, "fil header" and "fil trailer", are mostly the concern of the the outer status information of pages when involved with other database pages. Sandwiched between the headers and trailers, are the records and the free (unused) space. A page always begins with two unchanging records called the infimum and the supremum. Then come the user records. Between the user records (which grow downwards) and the page directory (which grows upwards) there is space for new records. In leaf pages, record format is shown as Table 3. F stands for number of fields in Table 3. The field start offsets is a list of numbers containing the information "where a field starts". The extra bytes is a fixed-size header. The field contents contains the actual data.

Table 1: Table definition file format

| Offset | Blocks in the .frm files |
|---|---|
| 0x00 | file header |
| | padding |
| 0x1000 | key definition |
| | Padding |
| 0x2000 | header block |
| | column definitions |
| | column structures |
| | column names |

Table 2: Leaf page format

| Blocks |
|---|
| fil header |
| page header |
| infimum and supremum records |
| user records |
| free space |
| page directory |
| fil trailer |

Table 3: Record format

| Name | Size |
|---|---|
| field start offsets | (F*1)or(F*2)bytes |
| extra bytes | 6 bytes(5 bytes if compact format) |
| field contents | depends content |

## 4 The Proposed Method

This section demonstrates a complete process of reconstructing records on the basis of the carved table definition files. First, it carves the table structures. Second, it recovers the table records.

### 4.1 Carving table structure

In this step, it first acquires the disk image. DD command is used to get the image under Linux operating system and under Windows operating system a tool Winhex is used to get the disk image. Then it carves the table definition files from the disk image. File carving is a forensics technique that recovers files based merely on file structure and content without any matching file system meta-data. Header-footer carving is a basic and classical carving technique [11]. The table definition files have distinctive header to find the start of the table definition file, but they don't have distinctive footer as end of the file. So the size of the definition file must be calculated. Supposing the number of columns is $k$, $t_i$ is the length of the column name, and $n$ is the file offset of the start of the columns. First the length of the column names $a$ can be calculated through Eq. (1).

$$a = \sum_{i=1}^{k} t_i \tag{1}$$

Then, the general size of the whole continuous file $b$ can be calculated through Eq. (2).

$$b = n + 2 * a + 3 * k + 17 * k + 2 \tag{2}$$

According to Eqs. (1)$\sim$(2), a simplified algorithm of recovering .frm files is shown below as Algorithm 1.

The purpose of line 1$\sim$4 is to search the file header of all the .frm files; and the line 5$\sim$8 is to determine the file offset of the start of the columns; the line 9$\sim$18 is to caculate the size of the .frm files; If the .frm file has comment part, the enum column or the set column, then the line 19$\sim$21 is to add the length of the comments, the enum values and the set values to the final size of the .frm files; and the line 23 is to create the .frm file to be analyzed in the process of generating table structures.

Lastly, generates the table structures from the reconstructed table definition files. First find the key block in table definition file. Each key of an InnoDB table has a key entry. Find and analyze the primary key's corresponding key entry in order to get the columns which compose of the primary key. In table definition file, a column corresponds to a column definition block and a block column structure. Analysis of these two blocks is to get the column 's name and data type [12]. Like that way, The information of all the columns is gotten. To restore the records, then the resulting table structure is enough.

### 4.2 Recovering records

In this step, it first extracts all the leaf pages from the disk image or ibdata1 and groups them by the same index. Leaf pages are identified by infimum and supremm records. The default size of database pages is 16KB. Find leaf pages' index id from page header, write leaf pages as files on disk and classify them by the same index. Second, matches the table structure to the

**Algorithm 1** Recovering .frm files

```
Input: the image file of a disk partition $D$ with size of $S$ bytes
Output: the set of .frm files $F$
 1: Set the number of the image's clusters $ClustersNum = S/4096$
 2: for $i = 1$; $i < ClustersNum$; $i = i + 8$ do
 3:     Locate at the file offset 4096*(i-1) bytes
 4:     Read 32768 bytes into the buffer array $Buffer[32768]$
 5:     for each ($consecutive\ eight\ bytes \in Buffer[42768]$ do
 6:         if $consecutive\ eight\ bytes = 0xFE$ 01 09 $0C$ 03 00 00 10 $or$ $0xFE$ 01 $0A$ $0C$ 03 00 00 10 then
 7:             Read one byte into $m$ at the file offset as sum of the current file offset and 8448 bytes
 8:             if $m = 0x00$ then
 9:                 $n \leftarrow 0x3100$
10:             else
11:                 $n \leftarrow 0x2100$
12:             end if
13:             for each $column \in this.frmfile$ do
14:                 Read column definition block into $ColD[]$
15:             end for
16:             Calculate the length of the column names $a$ with Eq. (1)
17:             for each $column \in this.frmfile$ do
18:                 Read column structure block $ColS[]$
19:                 Get the length of the comment $p$
20:                 if the data type of the culumn is enum then
21:                     Get the length of the enum values $l$
22:                 end if
23:                 if the data type of the culumn is set then
24:                     Get the length of the set values $q$
25:                 end if
26:             end for
27:             Calculate this .frm files size with Eq. (2)
28:             if $m = 0x02$ then
29:                 $b \leftarrow b + 48 + p + l + q$
30:             else
31:                 $b \leftarrow b + p + l + q$
32:             end if
33:             Create this .frm file
34:         end if
35:     end for
36: end for
```

corresponding primary index. InnoDB data dictionary stores correspondence between table and index. Find the same column in SYS_COLUMNS with the carved table structure and get the table id. Find the same table id from SYS_INDEXES and get the primary index id from the index name "primary". This can be explained well by the following SQL statements.

SELECT TABLE_ID FROM SYS_COLUMNS WHERE 'NAME' = 'column_name'

SELECT ID FROM SYS_INDEXES WHERE TABLE_ID='table_id' and NAME='primary'

Finally, all of records are recovered by applying the table structure on the matched leaf pages. For redundant records, its size can be gotten from the field start offsets. For compact records, its fixed-length field's size can be derived from the carved table structure and its varied-length fields size can be derived from the field start offsets. After the records are extracted from the leaf pages of the primary index, the records are interpreted into human-readable information according the carved table structure.

## 5 Evaluation Experiments

This section evaluates the effectiveness of the proposed method outlined in Section 4. The recovery depends on whether InnoDB kept all records in a single ibdata1 or each table had its own tablespace. Therefore, the experiment is consisted of two parts.

The two criterions to evaluate the proposed method are the precision rate and recall rate (defined as Eqs. (3) and (4)), and the F-value is used to evaluate the quality of recovery method (defined as (5)). The recovered table means recovering all of the records of the table. In these equations, the A means the number of recovered tables which belong to the dataset; the B means the number of recovered tables which do not belong to the dataset; and the C means the number of tables which belong to the dataset but do not be recovered from the disk image. The proposed method is compared with Percona InnoDB Recovery Tool Release 0.5 using InnoDB data dictionary and undrop-for-innodb-0.0-34.

$$presion = \frac{A}{A+B} \tag{3}$$

$$recall = \frac{A}{A+C} \tag{4}$$

$$F - value = \frac{2 * presion * recall}{presion + recall} \tag{5}$$

### 5.1 Scenario one

Mysql 5.1.32 is installed from source in under ubuntu operating system. The innodb_file_per_table option is off. The default character set is latin1. Sample InnoDB databases tpcc are installed. The tpcc database is used to simulate TPC-C test. Then seven of nine tables are randomly dropped, creating sample data loss. The 18GB raw image and ibdata1 are recovery prerequisites. Following the process of the proposed method, the concrete experimental result is shown as Table 4. In database/table column of the Table 4, means the table has been dropped. In table structure column of the Table 4, 1 means the table's table structure is gotten and 0 means the table's table structure is not gotten. In table reords column of the Table 4, 0 means the table's table records is not gotten or incorrect and other numbers mean the table's table records are gotten correctly. The precision rate, recall rate and the F-value of comparing result is shown as Fig. 1.

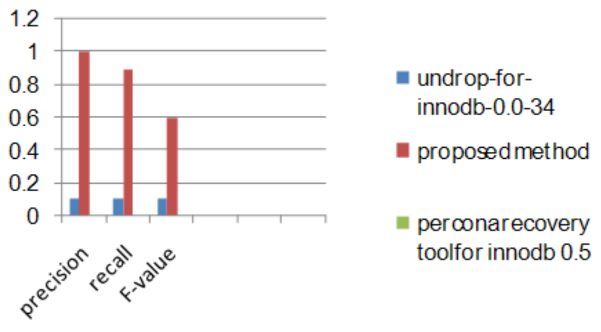

Fig. 1: Comparison result in scenario one

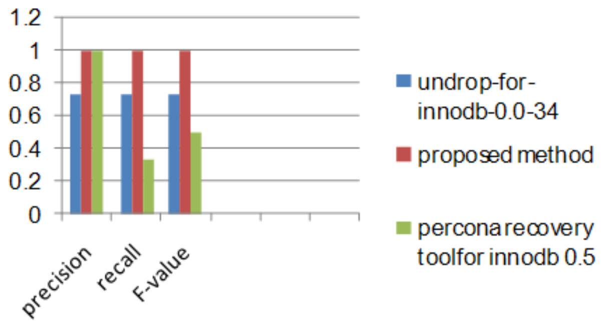

Fig. 2: Comparison result in scenario two

Table 4: Experimental result of scenario one

| Database/table | Percona Innodb Recovery Tool release 0.5 | | Undrop-for-innodb -0.0-34 | | Proposed method | |
|---|---|---|---|---|---|---|
| | table structure | table records | table structure | table records | table structure | table records |
| tpcc/customer* | 0 | 0 | 1 | 0 | 1 | 30000 |
| tpcc/district* | 1 | 0 | 1 | 0 | 1 | 10 |
| tpcc/history* | 0 | 0 | 1 | 0 | 1 | 30000 |
| tpcc/item | 1 | 0 | 1 | 0 | 1 | 100000 |
| tpcc/new_orders* | 0 | 0 | 1 | 0 | 1 | 9000 |
| tpcc/order_line* | 0 | 0 | 1 | 0 | 1 | 299984 |
| tpcc/orders* | 0 | 0 | 1 | 30000 | 1 | 30000 |
| tpcc/stock* | 0 | 0 | 1 | 0 | 0 | 0 |
| tpcc/warehouse | 1 | 0 | 1 | 0 | 1 | 1 |

## 5.2 Scenario two

MySQL 5.5.34 is installed under ubuntu operating system. Sample InnoDB databases sakila are installed. The sakila sample database is designed to represent a DVD rental store. It was developed by Mike Hillyer, a former member of the MySQL AB documentation team. The innodb_file_per_table option is on. The default character set is utf-8. Then ten of fifteen tables are randomly dropped. The process of the proposed method is performed. The 20GB raw image and ibdata1 are recovery prerequisites. Following the process of the proposed method, the concrete experimental result is shown as Table 5. The explanation way of Table 5 is the same with Table 4. The precision rate, recall rate and the F-value of comparing result is shown as Fig. 2.

It can be seen that the proposed method gets highest scores of precision, recall and F-value from Fig. 1 and Fig. 2. In experiment result of scenario one shown as Table 4, Percona InnoDB Recovery Tool Release 0.5 gets one dropped table's structure and gets all undropped tables' structure. But it doesn't recover even one table. In experiment result of scenario two shown as Table 5, Percona InnoDB Recovery Tool Release 0.5 gets none of dropped table's structure and recovers all the undropped tables. In experiment results of scenario one and two shown as Table 4 and Table 5, although undrop-for-innodb-0.0-34 got all the tables' structure, it only recover one table's records correctly in scenario one and eleven tables in scenario two. For tables which has table structure but has not table records in two above methods, these tables's structures have decimal data type or datetime type. The precision of the decimal data type can't be derived from data dictionary. These tables's structures are not accurate. In experiment result of scenario one shown as Table 4, the proposed method doesn't get the store table's structure. In experiment result of scenario two shown as Table 5, the proposed method gets all the tables's structures and recovers all the tables correctly.

# 6 Conclusion

In this paper, a new recovery method for InnoDB table records based on the reconstructed table definition files is proposed. It uses file carving method to get table definition files from a raw

Table 5: Experimental result of scenario two

| Database/table | Percona Innodb Recovery Tool release 0.5 | | Undrop-for-innodb -0.0-34 | | Proposed method | |
|---|---|---|---|---|---|---|
| | table structure | table records | table structure | table records | table structure | table records |
| sakila/store | 1 | 2 | 1 | 2 | 1 | 2 |
| sakila/staff | 1 | 2 | 1 | 2 | 1 | 2 |
| sakila/rental* | 0 | 0 | 1 | 0 | 1 | 16044 |
| sakila/payment* | 0 | 0 | 1 | 0 | 1 | 16049 |
| sakila/language* | 0 | 0 | 1 | 6 | 1 | 6 |
| sakila/inventory* | 0 | 0 | 1 | 4581 | 1 | 4581 |
| sakila/film_category* | 0 | 0 | 1 | 1000 | 1 | 1000 |
| sakila/film_actor* | 0 | 0 | 1 | 5462 | 1 | 5462 |
| sakila/film * | 0 | 0 | 1 | 0 | 1 | 1000 |
| sakila/customer* | 0 | 0 | 1 | 0 | 1 | 599 |
| sakila/country | 1 | 109 | 1 | 109 | 1 | 109 |
| sakila/city | 1 | 600 | 1 | 600 | 1 | 600 |
| sakila/category * | 0 | 0 | 1 | 16 | 1 | 16 |
| sakila/address | 1 | 603 | 1 | 603 | 1 | 603 |
| sakila/actor * | 0 | 0 | 1 | 200 | 1 | 200 |

image. Then it analyzes those table definition files to get table structures. Finally it matches the table structures to the right primary index of the leaf pages to get valid records. The experimental results show the proposed method is effective. In the future, we plan on getting the table structure from redo log files because redo log encodes requests to change InnoDB database.

## Acknowledgement

This work is supported by the Natural Science Foundation of China under Grant No. 61070212 and 61003195, the State Key Program of Zhejiang Province Natural Science Foundation of China under Grant No. LZ15F020003.

## References

[1] Fruhwirt P, Huber M, Mulazzani M, et al. Innodb database forensics. Advanced Information-Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010: 1028-1036.

[2] Pavlou K E, Snodgrass R T. Forensic analysis of database tampering. ACM Transactions on Database Systems (TODS), 2008, 33(4): 30.

[3] Stahlberg P, Miklau G, Levine B N. Threats to privacy in the forensic analysis of database systems. Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 2007: 91-102.

[4] Olivier M S. On metadata context in Database Forensics. digital investigation, 2009, 5(3): 115-123.

[5] Peter K. MySQL Internals Manual. 2011.

[6] Frühwirt P, Kieseberg P, Schrittwieser S, et al. InnoDB database forensics: Enhanced reconstruction of data manipulation queries from redo logs. Information Security Technical Report, 2013.

[7] Deep Dive into InnoDB Internals. http://forums.mysql.com/read.php?22,577068,577068#msg-577-068.

[8] Frühwirt P, Kieseberg P, Krombholz K, et al. Towards a forensic-aware database solution: Using a secured database replication protocol and transaction management for digital investigations. Digital Investigation, 2014, 11(4): 336-348.

[9] Percona InnoDB Recovery Tool Release 0.5. https://launchpad.net/percona-data-recovery-tool-for-innodb.

[10] undrop-for-innodb-0.0-34. https://launchpad.net/undrop-for-innodb.

[11] Garfinkel S L. Carving contiguous and fragmented files with fast object validation. digital investigation, 2007, 4: 2-12.

[12] Axmark D, Widenius M. MySQL Reference Manual. O'Reilly, 2002.