Pers Ubiquit Comput (2012) 16:707–715
DOI 10.1007/s00779-011-0428-7

ORIGINAL ARTICLE

A recovery method of deleted record for SQLite database

Sangjun Jeon · Jewan Bang · Keunduck Byun · Sangjin Lee

Received: 10 January 2011 / Accepted: 30 May 2011 / Published online: 24 July 2011

Abstract SQLite is a small-sized database engine largely used in embedded devices and local application software. The availability of portable devices, such as smartphones, has been extended over the recent years and has contributed to growing adaptation of SQLite. This implies a high likelihood of digital evidences acquired during forensic investigations to include SQLite database files. Where intentional deletion of sensitive data can be made by a suspect, forensic investigators need to recover deleted records in SQLite at the best possible. This study analyzes data management rules used by SQLite and the structure of deleted data in the system and in turn suggests a recovery tool of deleted data. Further, the study examines major SQLite suited software as it validates feasible possibility of deleted data recovery.

Keywords SQLite · Recovery · Record · Database

1 Introduction

The need for utilizing database systems for efficient management of vast amounts of data has been constantly increasing as most of data are converted to and saved in digital form. Contents stored in a database are often information necessary to system operation, providing the database to be a likely source of important information in a forensic investigation. Considering that deletion of data is frequently practiced in order to manage storage space or to update with the latest data, acquiring deleted data information is equally important as retrieving undamaged information from the database.

SQLite is an ANSI-C-based, open source database software. For the advantages of small size and fast running speed, the software is mainly used within application software where systematic storage is required to process and manage less complicated yet large amounts of data effectively or suited in onboard software made for embedded devices. With constant updates made into the software and for its widespread popularity among software developers, further and broader adoption of SQLite is anticipated. Accordingly suggested, the possibility is that forensic investigators would locate significant amount of data within a SQLite database file in a case provided with an embedded device as evidence. Compared to databases in different physical environments, deletion of data occurs more frequently in most embedded devices due to much limited storage spaces, asserting the importance of recovering deleted records in SQLite databases.

2 Related work

Research of database record recovery is already begun in 1983 by Haerder [1]. He suggested method of deleted record for database using transaction file of database. This method can be applied when the information of deleted record is included in transaction file.

S. Jeon · J. Bang · K. Byun · S. Lee (✉)
Digital Forensic Research Center, Korea University, Seoul, South Korea
e-mail: sangjin@korea.ac.kr

S. Jeon
e-mail: heros86@korea.ac.kr

J. Bang
e-mail: jwbang@korea.ac.kr

K. Byun
e-mail: gdfriend@korea.ac.kr

Record recovery by transaction was also applied to SQLite database. An application program suited with SQLite creates a journal file named as "<database file name>/<file extension>-journal" whenever a query is sent to the database, saving the history of input/output records made prior to the completion of the query process. A previous study conducted by Pereira [2] attempted recovering deleted records in Mozilla Firefox 3 using this journal file.

While the objective of deleted data recovery in SQLite is on the line of Pereira's work, the present study suggests a tool using recovery method that approaches actual data files instead of a journal file that would offer improved practical availability.

3 SQLite

3.1 SQLite

SQLite is an open source database software, light on system, and fast running; it is widely used in application software that needs to save simple data in a systematic manner or adopted into embedded device software. Unlike other types of database systems, SQLite is usually used as "local-only" database that saves all the results of database usage in a single file. When suited in an application program, SQLite engine performs direct read/write operations and manages them into a file consisted of table, index, trigger, and view entries. SQLite features small size of the library size, about 250 KB with full configuration and about 180 KB when running with necessary options only. It also runs on the minimum stack space of 16 KB and uses heap size of approximately 100 KB, which is very tiny. This enables the users to setup a database engine in an operation environment with limited active memory, such as mobile phones, PDAs, and MP3 players [3].

SQLite is considered to be highly reliable in general cases. The software supports most of standard SQL (ANSI/ISO SQL), while non-supported items are listed on the product homepage [4]. Additionally, it claims to guarantee ACID (Atomic, Consistent, Isolate, Durable) feature for transactions even in the incidents of unexpected system failure, such as power outage [3].

Written in ANSI-C, SQLite is furnished with bindings for various programming languages including TCL. Ninety-nine percent of the code has been tested, and it has no external dependencies. Further, the software supports Linux, UNIX, Mac OS X, OS/2, and Win32, enabling easy port between different operation systems, platforms, and architectures [3].

3.2 Applications

A number of application softwares currently use SQLite to save information generated during the process, represented by Safari, iPhone OS, FireFox3.x, Chrome, Android, Skype, and TweetDeck [5]. The saved file directory and stored contents for each application are described in Table 1.

iPhone and Safari are the products of Apple Inc. iPhone is a smartphone that uses SQLite to save data information resulted from smartphone communication, such as SMS, call logs and caller/receiver information, and multi-male attachments. In Safari, a web browser, SQLite, is used to save cache data produced during HTTP webpage search [6]. As for Mozilla Firefox, the file format Mork originally developed by Mozilla Foundation itself is being replaced by SQLite-based files. In Firefox 3.0, web history and web cookies are stored in SQLite file. Google's Android is a collection of software including operation system, middleware, and core applications for embedded devices. Its core applications mostly save data using SQLite. Google's web browser Chrome, since its first public stable release on December 11, 2008, has been using SQLite to save cache, history, and cookies. Skype is a messenger software whose ownership is partly shared by eBay and provides various options to communicate. The fact that it requires simple registration to be a user and all communications are fully internet based has built a great number of users worldwide. Skype features options including voice call over internet and text chat for users to communicate with other users on their list, and all the communication data (chat history, file transaction history, and contacted users' information) are stored in SQLite file. TweetDeck is an application software with which Twitter and Facebook, the widely used SNS (Social Networking Service), can be available without accessing respectable webpages for the service. In TweetDeck, SQLite is used to store the friend list and metadata on SNS usage.

3.3 Record deletion in application software

All the mentioned are SQLite suited, yet default deleted data management rule upon the deletion of record(s) differs to each application software.

The rules are largely classified into three as shown in Table 2. The first rule is where deletion of data prompts overwriting them with zero(s). This is used in web browsers Firefox 3, Safari, and Chrome; recovery of data is impossible in this case even if the deleted area is identified. The second rule is the case of iPhone when the deletion is made on small size of data. It deletes the area itself, and there is no way to even trace occurrence of deletion. The last rule is to set the data area as free, with the data itself is remained in the system. Deleted text chat messages in

Table 1 SQLite embedded software

Application	Stored data type	Saved file directory (Windows XP)
Safari	Web cache	%USERPROFILE%\Local Settings\Application Data\Apple Computer\Safari\Cache.db
iPhone	Smartphone communication data	At iPhone data backup, saved in a file as <Random Value>.mdata (Note: Signature verification is required to confirm the SQLite file since other data files are saved in the same format)
Firefox	Web history	%USERPROFILE%\Application Data\Mozilla\Firefox\Profiles\<Random Value>\places.SQLite
	Web cookie	%USERPROFILE%\Application Data\Mozilla\Firefox\Profiles\<Random Value>\cookies.SQLite
Chrome	Web cache	%USERPROFILE%\Local Settings\Application Data\Google\Chrome\User Data\Default\Cache\data_0
	Web history	%USERPROFILE%\Local Settings\Application Data\Google\Chrome\User Data\Default\History
	Web cookie	%USERPROFILE%\Local Settings\Application Data\Google\Chrome\User Data\Default\Cookies
Android	Smartphone communication data	/data/data/com.android. <Software Name>/databases
Skype	Messenger communication data	%USERPROFILE%\Application Data\Skype\<Skype User ID>\main.db
TweetDeck	Metadata/Friend list	%USERPROFILE%\Application Data\TweetDeckFast.<Random Value>\Local Store

Skype messenger and group deletion of SMS in iPhone follow this rule. Deleted data still remain in the file, and recovery is possible provided that they remain without being overwritten.

4 SQLite database file structure

4.1 SQLite database file

The overall structure of a SQLite file is provided in Fig. 1, and the file signature is "0x53 51 4c 69 74 65 20 66 6f 72 6d 61 74 20 33 00" (16 bytes) as Fig. 2 describes. SQLite files do not have a specific filename extension, and therefore, identifying a SQLite file is determined by the signature of target file.

In data storage in SQLite, a page is the unit used. As shown in Fig. 2, the size of page is specified as 2-byte integer at an offset of 16 (big endian), and default size is 0×400. The page appears at the beginning of file is called header page; in the upper part of a header page comes the database file's header including signature, while in the lower part is the schema table containing the table information of the database.

Following the header page is consisted of b-trees, again largely separated as index b-tree and table b-tree where the latter stores data contents.

In SQLite, the elementary unit of store operation is a cell, while cell structure and stored data contents vary to

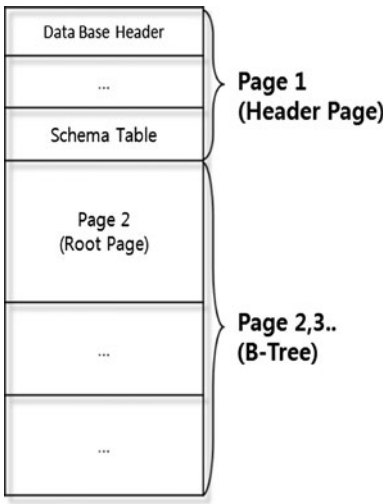

Fig. 1 SQLite file structure

internal and leaf pages. An internal page is found in the middle section of tree, and the cells store the pointers to locate lower page. A leaf page is located at the bottom of tree, and there is no lower page exists; the cells in this part contain database records.

4.2 Schema table

The schema table in a header page contains information on table, index, and trigger comprised in the SQLite file in an application software. The structure of schema table is

Table 2 Deleted data management rules in major applications

Deleted data management rule	Recoverability	Major application software
Overwrite with zero	No	Firefox 3, Safari, Chrome
Remove deleted area	No	iPhone OS (Selective SMS deletion)
Set as free space	Yes	Skype, TweetDeck, iPhone OS (Group SMS deletion)

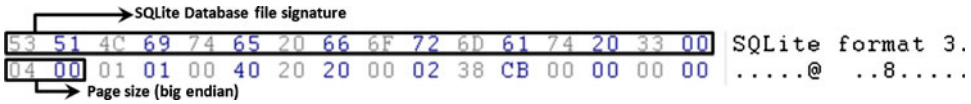

Fig. 2 SQLite header first 32 bytes

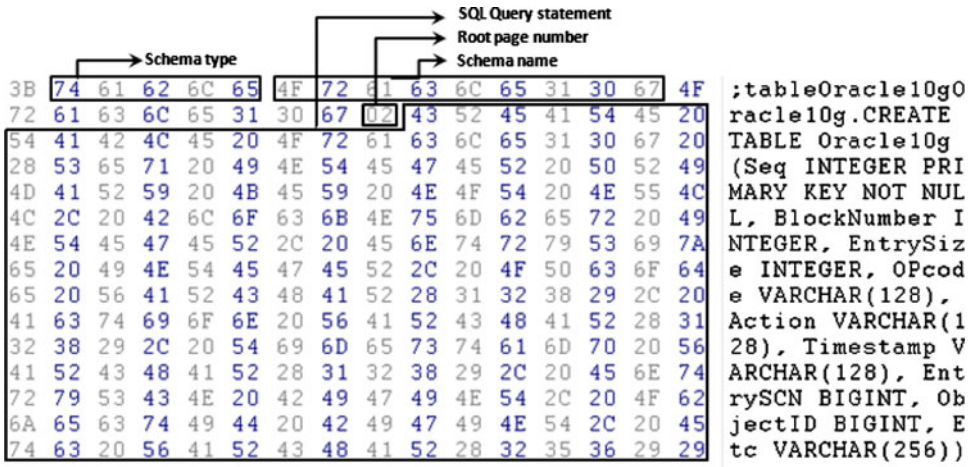

Fig. 3 SQLite schema table

presented in Fig. 3, where schema type indicates which of table, index, and trigger is the type of a specific schema. The schema name section contains generated schema names and then the copy of each name, followed by root page numbers in 1-byte integer. The string appearing after root page numbers is a SQL query statement that is sent to SQLite when the schema is created. In order to recover deleted records, the table information must be obtained by interpreting the SQL query statement specific to table schema type in the schema table. The information available in a schema table is table name, field type and name, and the number of fields.

There are four field types in SQLite, namely INTEGER, TEXT, BLOB, and NUMERIC. Datatypes including VARCHAR(125), BIG INT, and DATE that exist in the standard SQL query are converted to one of the four datatypes in SQLite, whichever the system determines as the most relevant. The conversion rules are described in the SQLite homepage [7].

4.3 Variable length integer

Integer values in SQLite are stored in variable length of 1–9 bytes based on the value, as shown in Table 3. In variable length integer format, also described in the table, it uses last 7 bits from each of 1–8 bytes to save variable length integers. As for 9-byte variable length integer, all 8 bits from the final byte are present, hence total 6\4-bit present.

The fact that SQLite saves integers in variable lengths based on the value may obscure locating of each field when analyzing deleted records. To recover deleted records, therefore, precise length of variable length areas in each cell must be identified.

Table 3 Variable length integer format

Byte	Value range	Bit pattern
1	7 bit	0XXXXXXX
2	14 bit	1XXXXXXX 0XXXXXXX
3	21 bit	1XXXXXXX 1XXXXXXX 0XXXXXXX
4	28 bit	1XXXXXXX 1XXXXXXX 1XXXXXXX 0XXXXXXX
5	35 bit	1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 0XXXXXXX
6	42 bit	1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 0XXXXXXX
7	49 bit	1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 0XXXXXXX
8	56 bit	1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 0XXXXXXX
9	64 bit	1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX 1XXXXXXX XXXXXXXX

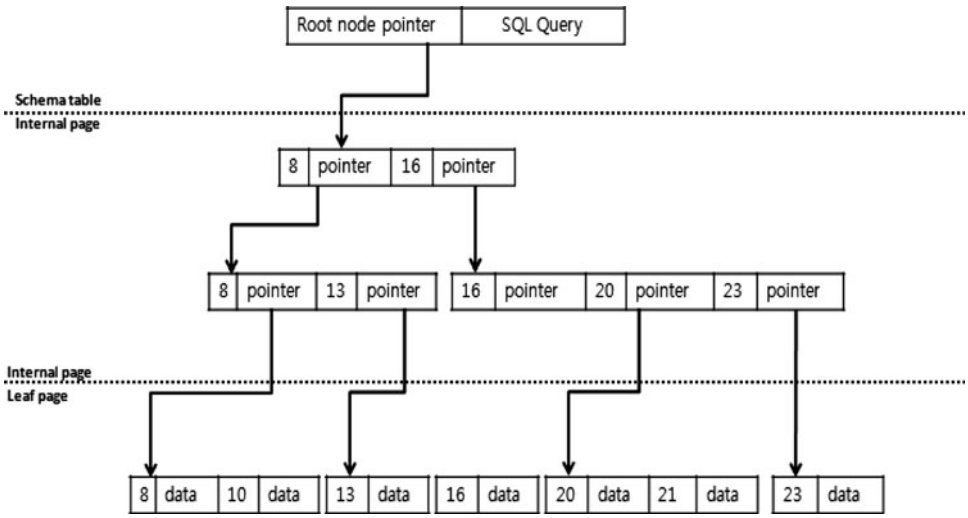

Fig. 4 SQLite b-tree structure

4.4 Table b-tree

Among b-trees existing in SQLite file, the purpose of a table b-tree is to store actual data. Structure of a table b-tree is provided in Fig. 4.

A root page number is found in a schema table, the 1-byte integer located just before the SQL query statement (see Fig. 3). Occasionally, a root page may as well be internal and leaf pages. An internal page contains pointers to the child page, while actual data are stored in a leaf page.

4.5 Page structure and unallocated area

All pages in a SQLite file, both internal and leaf pages, share the structure illustrated in Fig. 5. A page header is followed by list of big endian integers at 2-byte offset values, which specifies the location of a cell that contains actual data. Cells are used in upward sequence from the bottom of the page. The area between a cell offset and a cell is referred to as free space, filled with zeros when initially created. Beside free space, there can be unallocated space called free block within a leaf page, a place to store a cell until deleted and maintained to be used later for newly generated and assigned cell.

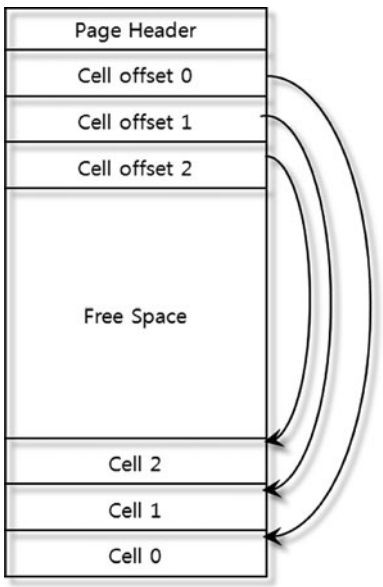

Fig. 5 SQLite page structure

4.5.1 Page header

4.5.1.1 Internal page header As seen in Table 4, an internal page header consists of 12-byte, where the first byte is a page flag with value of 0×05. Offsets 1 and 2 are 2-byte big endian integers indicating the first free block offsets. The following offsets 3 and 4 present the number of cells existing in the page, written in 2-byte big endian integers. Where the number of cells is described as zero, the specific page is a free page. Offsets 5 and 6 are the offset of first appeared cell. The offset 7 describes the number of free blocks in size of 3-byte and smaller, written in 1-byte integer. The final offsets from 8 to 11 are the rightmost page number found within the lower section of the current page, present as 4-byte big endian.

Table 4 Internal page header contents

Offset	Contents
0	Page flag: 0×05
1–2	First free block Offset
3–4	Number of cells within the page
5–6	First appeared cell offset
7	Number of free blocks less than 3 bytes
8–11	Rightmost child page number

Table 5 Leaf page header contents

Offset	Contents
0	Page flag: 0 × 0D
1–2	First free block offset
3–4	Number of cells within the page
5–6	First appeared cell offset
7	Number of free blocks less than 3 bytes

4.5.1.2 Leaf page header A leaf page header has a structure similar to an internal page header. As provided in Table 5, the leaf page header consists of 8-bytes, where the first byte is a flag with value of 0 × 0D. The rest contents are identical to that of an internal page; the only difference is that the last 4-byte information found in an internal page is now omitted as there is no child page.

4.5.2 Cell

As stated earlier, SQLite uses a cell as the elementary unit of stored information amount. Cells within a leaf page of a table b-tree include database records. Recovering deleted records require to identify the structure and the rules for leaf page cells.

4.5.2.1 Internal cell The cells in an internal page, as described in Fig. 6, are for maintaining a b-tree and have pointers to the child page (child page number). A key, the identification value of a cell, is in variable length integer; the first 4-byte in a cell is the child page number in big endian format.

4.5.2.2 Leaf cell In order to describe the records stored in a leaf cell, records are managed in three areas: cell header, record header, and record data area (see Fig. 7). In a cell header are the length of the cell, excluding the cell header, and rowID in variable length integers. A record header contains the value of record header length in variable length integer format, followed by the record data area in which byte length information of each field of a record is stored as Table 6 specifies. For TEXT and BLOB types, however, the length is described in values larger than 2-byte in the record header when the data length exceeds 56-byte.

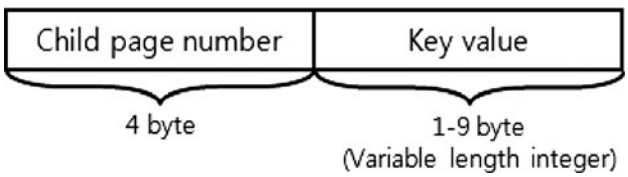

Fig. 6 Internal cell structure

Table 6 Describing format of type area

Value	Data type	Data size
0	NULL	0
N (N = 1–4)	Signed integer	N
5	Signed integer	6
6	Signed integer	8
7	IEEE float	8
8–11	Reserved	
N > 12 (N:even)	BLOB	(N−12)/2
N > 13 (N:odd)	TEXT	(N−13)/2

5 Recovery method of deleted records in SQLite

5.1 Recovery process

The first step for deleted record recovery is identifying the number of fields in a schema table and the type of each field. Then, a page is scanned in order, identifying all unallocated area (free space and free block) within the page and retrieving data thereof. Once the scanning of all unallocated area completes, the child page is scanned. The scanning repeats for the rest of pages to complete the recovery process. Regarding deleted area, although it is specified as unallocated area, all areas that have values other than zero can be technically considered as deleted area.

Deleted areas in SQLite are differentiated for the three cases shown in Fig. 8. The property of each area type is as follows:

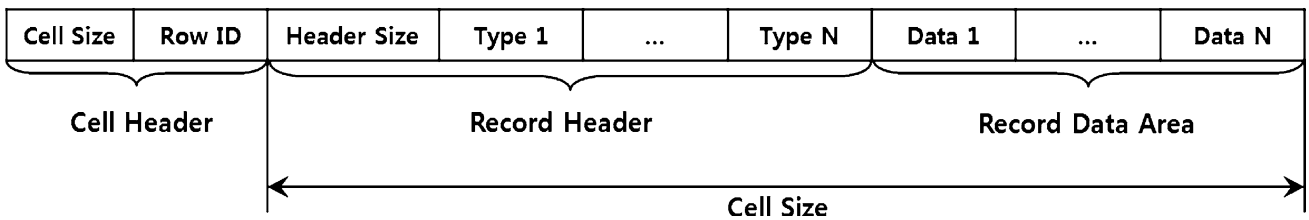

Fig. 7 Leaf cell structure

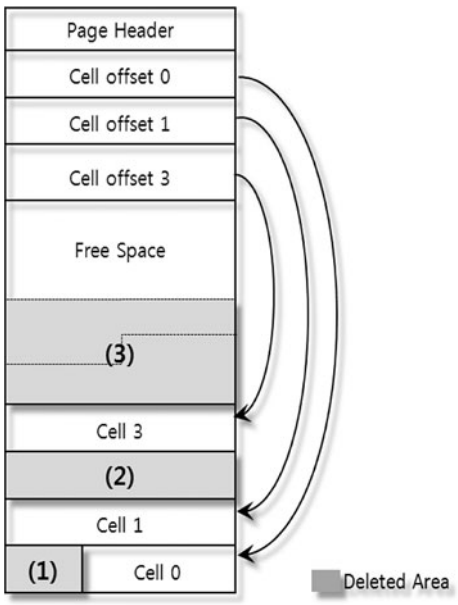

Fig. 8 Deleted area within page

1. A case where the deleted area contains a single independent record
2. A case where the deleted area contains more than two records
3. A case where remained data are partial due to overwriting with other data after the deletion

The recoverability and recovery process differ to each case due to the case property. The case (1) is relevant to Inspect Free Block Case 1 in the recovery process described in Fig. 9, where immediate recovery is possible without further process. In the case (2), case 2 in Fig. 9, the data can be recovered once they are split and formed as in the case (1). However, data recovery is impossible in the case (3) as the length of each field in the cell cannot be identified.

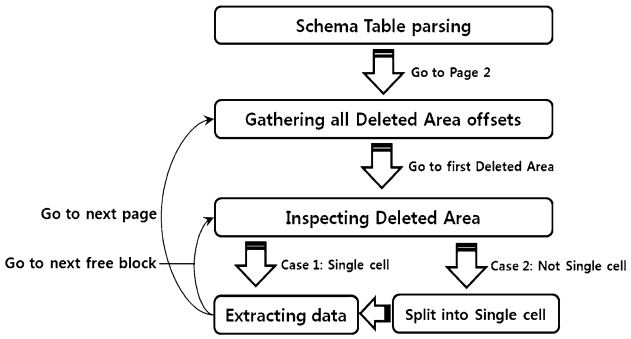

Fig. 9 Recovery process of deleted area

5.2 Gathering all deleted area offsets

Approaching deleted area requires leaf page scan. As the value of first byte in the leaf page is 0 × 0D, it can be classified from other pages by identifying the first byte while scanning the file at page size level.

According to the page header information described in the Sect. 4.5.1.2, a leaf page stores the offset value of the first free block at offsets 2 and 3 in 2-byte. The free block existing within a single page is always more than zero, and each block stores in its first 2-byte the offset value of the immediate next free block, resulting in a free block chain. The last free block has the value of 0 × 0000 in its first 2-byte that indicates the end of free block chain. This way, SQLite is designed to enable scanning all free blocks in a page using this free block chain.

A deleted cell is usually found in a free block, but occasionally, in a free space located before a normal cell, a free space is filled with zeros at the time of file creation, and thus, deleted area is identified by scanning for areas with values other than zero within the free space.

5.3 Inspecting deleted area and extracting data

When record deletion occurs, SQLite renews the initial 2-byte of the first 4-byte information of a cell with the offset value of next free block and the latter 2-byte with the length of current free block to assign the cell area where the deleted record is located as free block. This is the reason that the first 4-byte values of deleted cells differ from the normal case, and therefore, the approach to identify each field of the cell should also be different from one for normal data.

Information of cell size, rowID, and header size in a leaf cell are in the form of variable length integers (see Fig. 10), but the renewed information overwrites part of this information upon cell deletion (see Fig. 11), making it

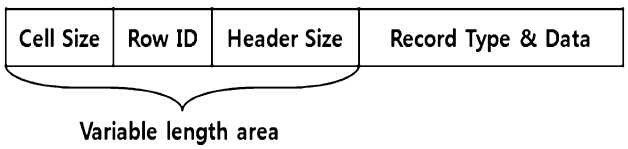

Fig. 10 Variable length area within cells

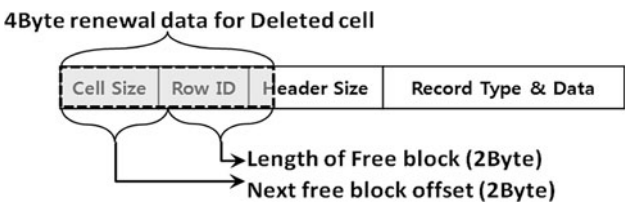

Fig. 11 Renewal information in First 4 byte of Deleted Cell

impossible to identify the exact length of each area in the deleted cell.

Hence, in order to recover a deleted cell, the process should involve estimating the variable length area described in Fig. 10 to identify the beginning of the record type and validating the estimated value. To validate, first, the information of the length of deleted cell needed to be inputed, and then, the validation initiates with given (estimated length of variable length area) value 3 (the minimum length of each field of variable length area is 1-byte), continuing with increasing the value in order.

Validation process includes back-calculating the cell length based on the estimated length of variable length area and comparing it with the initially inputed cell length. Once the length of variable length area is estimated, the length of record types immediately following the record header size of the record header area can be calculated as (the number of fields) +, where it is the value generated as the fields are presented in more than 2-bytes in TEXT and BLOB types with length value of 56-byte and higher. The record data area can be calculated using the contents of record type. Accordingly, the total length of estimated cell is calculated with the below-described formula.

where

f: formula to calculate the total cell length

x: estimated length of variable length area (Cell size + Row ID + Header size)

F: the number of fields

a: value created with TEXT and BLOB in more than 56-byte

l: the length of record data area

$$f(x) = x + F\,a + l$$

The value generated from the formula is compared with the initially obtained cell length. The estimated length of variable length area is considered correct when both values match and the recovery initiates. In the case of value mismatch, the above process is repeated with value increased by 1. When value exceeds 27-byte (the maximum length of each field of variable length area is 9-byte), the current cell is the case of (2) or (3) described in Sect. 5.1 that the deleted area contains partially overwritten (damaged) or multiple cells, and therefore, cell split is attempted.

5.4 Split to single cell

Recovery failure in the process described in Sect. 5.3 means the area is either damaged or included with multiple cells. The input of cell length to be compared with the value generated from the formula in Sect. 5.3 is unavailable in those cases; the recovery attempt repeated based on the Sect. 5.3 instruction, where the base cell length is then estimated by the initial n (n: the number of fields) and the value is increased by 1 at each reattempt. Given that the target area is not damaged, increasing estimated value reaches the point of recovery and the results are extracted accordingly. The repeats on the remaining area only, repeating until all areas are covered. The area is decided as damaged if recovery fails even after the estimated value of cell length is increased to the size of target area.

6 SQLiteRecover

A recovery tool for deleted records is developed based on the recovery method suggested in Chapter 5. SQLiteRecover analyzes a SQLite file and generates deleted records, as well as the normally stored records by scanning the table's b-tree structure.

Figure 12 presents the result of SMS data stored in iPhone 3GS using SQLiteManager. Of the backup data in iPhone 3GS, SQLite file contains SMS data in the table named message where received and sent SMS data are stored. Group (or bulk) deletion of SMS data in iPhone 3GS only sets the area of deleted SMS data without actually deleting the data. SQLiteRecover is able to read such data; Fig. 13 presents deleted SMS data recovered by using the tool. In addition to test case of iPhone's SMS data, SQLiteRecover is applicable for all SQLite files where

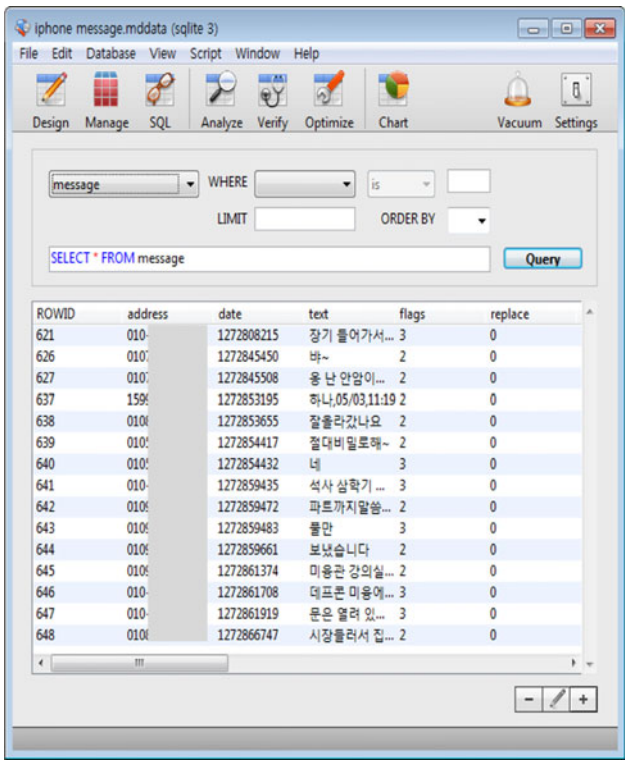

Fig. 12 Data view on SQLiteManager

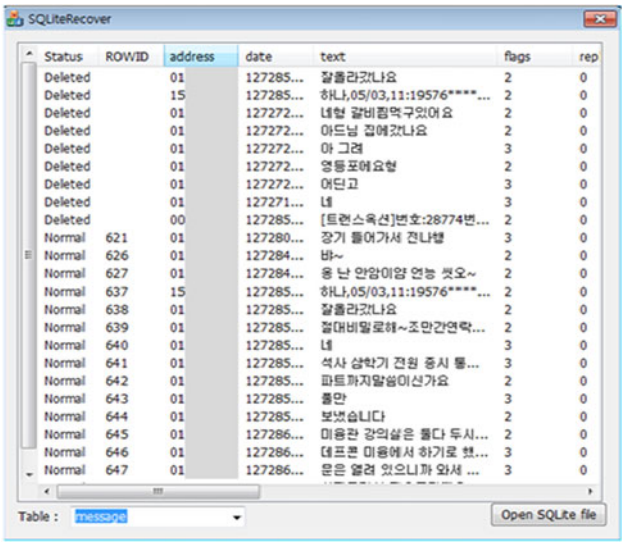

Fig. 13 Contents recovered by SQLiteRecover

deleted data still remain. Data recovery is also possible for the pages not included in the table's b-tree (free space), as the tool is designed to scan and save deleted data from all pages.

According to the result, SQLiteRecover extracts nine deleted SMS message from the backup data in iPhone 3GS. Apple has launched iPhone 4 and iPhone user in the world has increased greatly in year 2010. SQLiteRecover also works on iOS version 4.

7 Conclusion

Increasing adoption of SQLite reveals likewise growing forensic significance of SQLite, yet little efforts have been made on recovering deleted area to this day.

This study suggests effective recovery method of data remaining in the free space configured by analyzing the format of SQLite files. The suggested method can be used for any SQLite suited application software, and the test confirms that deleted but system-remained records can be recovered by the tool. Considering the widespread of portable devices, hence the use of SQLite, this method is expected to be a subfractional aid for digital forensic investigations to obtain significant information from SQLite files.

Acknowledgments "This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the "ITRC" support program supervised by the NIPA (National IT Industry Promotion Agency)" (NIPA-2010-C1090-1001-0004).

References

1. Haerder T (1983) Principles of transaction-oriented database recovery, ACM Comput Surv 15:287–317
2. Pereira MT (2004) Forensic analysis of the Firefox 3 Internet history and recovery of deleted SQLite records, Digit Investig 5:93–103
3. Newman C (2004) "SQLite" MacmillanComputerPub
4. SQLite (2004) SQL as understood by SQLite, http://www.sqlite.org/lang.html
5. SQLite (2004) Well-known users of SQLite, http://www.sqlite.org/famous.html
6. SQLite (2004) iPhone OS data management, http://developer.apple.com/technologies/iphone/data-management.html
7. SQLite (2004) Datatypes in SQLite version 3, http://www.sqlite.org/datatype3.html