ATLANTIS PRESS
Advances in Computer Science Research (ACRS), volume 54
International Conference on Computer Networks and Communication Technology (CNCT2016)

A SQLite Recovery Method for Various Primary Key and B+tree Reorganization

Jiang DU and Ming-jian LI*
College of Computer Science and Technology
Chongqing University of Posts and Telecommunications, China
*cqupt_limingjian@163.com

Keywords: SQLite, Recovery, Primary key, Free block, Free page.

Abstract. Free blocks and free pages have tremendous forensic potential. Based on analyzing layouts of cells which have different types of keys, the mechanism of deletion and free blocks coalescing, a new recovering method with byte level accuracy is proposed. First, aiming at multiple keys' types and rewriting cases, improved methods mentioned in this study were used to calculate the values of renewed bytes. Then, coalescent free blocks were split dynamically. In addition, deleted data also was extracted from trunk leaf pages because of tree structure and the free page generating principle. The results indicate that the method is suitable for different deleted data. The recovery rate relative to free blocks is over 90% in the case of the integer key. The rate can also archive 89% even if the key is not the integer.

Introduction

A The number of embedded device using SQLite has been increasing. Extracting data from embedded device has shown a significant increase. One of the most controversial issues about arguments involving digital forensics matters is how to recover deleted data and improve the recovery rate.

There have been a great number of studies in SQLite record recovery which have already begun in 1983.Haerder [1] and Pereira [2] both suggested that deleted record can be recovered by using several transaction files. R. Felix [3] thought index structures contained redundant data. Some researchers analyzed specific app's use of SQLite. C. Anglano [4] extracted some data from WhatsApp's chat database. Even though this method had a high recovery rate and a timeline of operation could be built, it was only applicable to specific applications.

Q. Li [5] and B. Wu [6] recovered historical deleted WAL and database files from file system by using the features of file system. The timeline of operations was built successfully.

The most basic study is analyzing structure of database file and the mechanism for space using. S. Jeon [7] suggested the size of payloadsize, rowId, and headersize could be traversed simply. In addition, to reduce the time complexity of the algorithm, Q. Fang [8] proposed a method that detects and estimates each Type field in one free block. However, there are still no observations at present about the case that one free block contains more than one cell and is not suitable for No-Intkey tables. A study conducted by J. Bai [9] who researched on the layout of No-Intkey table attempted to recover data from No-Intkey table, ignoring that adjacent free blocks were coalesced.

In this paper, the structure of the database file, mechanism of deletion and defragment has been studied. The layout of No-Intkey cell and the phenomenon of

coalescing free blocks and reducing the number of free block resulting from reorganizing of B+tree are also discussed. An effective recovery method with wide application range and high recovery rate is proposed. Many tests proved that the rate maintained a high level of recovery even if we increase deletion and insert operation.

SQLite Format

The file format of SQLite databases is described detailly in the official document [10]. For an understanding of the following chapters it is summarized below.

File Header: The first 100 bytes are header which contain global information of database (big endian) .16 bytes start at file is "SQLite format 3", a signature of SQLite. The head page number of FreeList is specified as 4-byte integer at an offset of 32. It is followed by a count of free pages which occupy 4 bytes too.

Page: Sizes of pages in same database are same. The first half part of page is header stored from up to bottom. The first byte is a page type with value of 0x05 (interior page) or 0x0D (leaf page). Offsets 1 and 2 are 2-byte integer indicating the first free block offsets and the value is 0 if the page has no free block. Offsets 5 and 6 sepcify the first cell in the page. Every two bytes can be indicated an offset of cell which in later half part named content area and allocated from bottom to up. central part is free area. Fig. 1 shows the structure of the page. Both content area and free area contain cells, free blocks and fragments. Cells in leaf page (lp) store records.

Cell in lp **:** The one-to-one mapping is proved between record and cell which is elementary unit of record if ignore overflow page. Cell size, types of columns and length of data exist in cell with big endian mode. Fig. 2 illustrates the cell's structure.

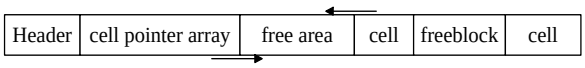


Figure 1. Layout of Page

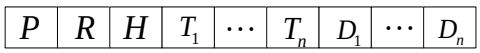


Figure 2. Layout of Cell

P, R, H, T_i everyone is variant integer described by 1-9 bytes. P is the byte numbers from H to D_n . R is number of keys. As indicated by sqlite3.c (line 57445, version 3150100), value of R is primary key value if key's type is integer. H is the length of header which includes H and $T_1 \cdots T_n$. Record data is stored in $D_1 \cdots D_n$. $T_1 \cdots T_n$ represent n columns' type and length which meaning of value showed in official document [10]. Since length of D_i is depended on T_i , theory length of $D_1 \cdots D_n$ can be obtained by calculating with $T_1 \cdots T_n$. T_1=0x00 if primary key's type is integer and the value of the key exists in R . T_1 still depends the length of first column in the situation that key's type is No-Integer.

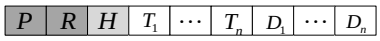
SQLite Deletion

If one cell is not overwritten after being deleted, database will transfer the cell to free block instead of clearing its data and add it into list [11]. We give two definitions of recovery rate. One is Relative Recovery Rate which is the ratio of cell recovered to the number of free blocs: $r = rc / fbc$. Another one is Absolute Recovery Rate which is the ratio of cell recovered to the number of deleted records: $R = rc / dc$.

Stage of Single Free Block: First two bytes of free block which exists in a list indicate the offset of next block and its value will be 0 if the fb is the end of the list. Offsets 3

and 4 present length of this block including the first 4 bytes. Formalization description of free block as follows: $fb = (len, b_1, b_2, \cdots, b_n)$, len is length of free block and $b_1, b_2, \cdots, b_n$ are bytes in free block. Set of fb : $Setfbs = \{ fb_i \mid i = 1, 2, \cdots, n\}$ can be obtained by traversing every lp .

Because $len(P + R + H) >= 3$ and only first four bytes will be rewritten when one cell is deleted, $T_2 \cdots T_n, D_1 \cdots D_n$ will remain, but T_1 may be changed. Let pre represent the number of bytes between cell start and T_1 . Two different cases are shown. (1) $pre >= 4$. T_1 does not located in the first four bytes and $T_1 \cdots T_n, D_1 \cdots D_n$ will not be changed, so $pre + 1$ could be the beginning of analysis and extraction. This case presented in Fig 3. (2) $pre = 3$. As indicated in Fig 4, T_1 will be renewed.



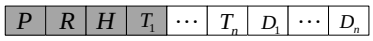


Figure 3. Situation pre >=4 Figure 4. Situation pre = 3

For this case, as discussed previously, T_1 always be 0x00 if primary key's type is integer while it will depend the length of D_1 if the key is No-Intkey. The value of T_1 must be identified accurately. Because one byte has 8 bits, let the value of byte whose position is $pre + 1$ increase from 0x00 to 0xFF. For each value, analysis and extraction can be applied. Exceptionally, per the principle of variant long integer, 0x80 is meaningless. In this paper's method, this value will be ignored.

Stage of Free Block Reorganizing: In sqlite3.c, function freeSpace (line 59811) and function defragmentPage (line 59576) will reorganized B+tree pages with the growth of times of deletions. All cells are moved to the end of the page and all free space is collected into one big FreeBlk that occurs in between the header and cell pointer array and the content area as well as coalescing adjacent fb . $pre > len(P + R + H)$ will occur if reorganization happened. What's more, one fb may contains more than one cells or half-cell after reorganizing and in this case, the fb must be split.

Stage of Free Page: Source code sqlite3.c(line 58776) indicates that the page will becomes a free-list page fp if all its data is deleted. Fig. 5 presents Free List's format.

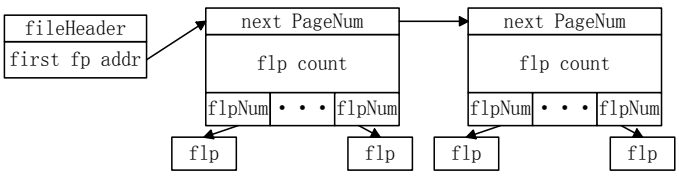


Figure 5. Structure of FreeList

TrunkPage: $tp = (nextNum, flp_1, \cdots, flp_n)$,if $nextNum$=0 , it`s the end of the List.

TLeafPage: $flp = (type, < c_1, c_2 \cdots c_n >), type \in \{0x0D, 0x0A, 0x05, 0x02\}$

By traversing the list, a list of free leaf pages whose types are 0x0D can be obtained:

$SetFlp = \bigcup_{i=1}^{n} (getflp(flp_i) \;\; flp.type = 0\text{x}0D)$, $getFlp(tp_i)$ represents getting all flp .

In addition, numbers of fb will decrease because of reorganizing which leads to the increasing of number of fp . In this reason, if we only recover data from fb , the

decreasing of R is not avoidable though r remains at a high level. This paper suggests it is necessary that recovering data from fp after recovering from fb.

Based on mechanism discussed in previous paper, this paper put forward an effective method named ESR based on analyzing fb, estimating T_1 and splitting fb ,which is widely suitable for database file and has high R.

Recovery Method

Some symbols defined here: *Judging-State of* fb : $Fb = (pre, n, fb)$ pre 's value is estimated position of T_1 in fb and n represents the number of one table's columns.

Set of Fb : $SetFbs = \{Fb_j \mid Fb_j.pre \in (3, len)\}$. One fb_i corresponds to $len-3$ judging-states because of T_1 's varied beginning positions. Each fb_i will generate a set of judging-states: $(fb_i \mid i = 1, 2, \cdots, n) \rightarrow SetFbs_i$.

Set of unsuccessful fb : $SetFailfbs = \{fb_i \mid i = 1, 2, \cdots, n\}$ means fb_i is recovered unsuccessfully by Algorithm 1.

Set of recovered cells: $SetCells = \{cell_i \mid i = 1, 2, \cdots, n\}$

Functions of Recovering

$$RcCell(Fb) = \begin{cases} \bigcup_{i=0}^{255} RC(Fb, i),\ Fb.pre = 3 \\ rc(Fb),\ Fb.pre \in (4, Fb.fb.len) \end{cases} \tag{1}$$

$\bigcup_{i=0}^{255} RC(Fb, i)$ contains the following operations: Traversing possible value of T_1, from 0x00 to 0xFF. Analyzing will be executed for every statement that one value corresponds just as shown in the following formula: $Fb.fb.b_4 = i,\ rc(Fb)$. The return value is depended on $rc(Fb)$. If true is returned, stop traversing.

$rc(Fb)$ whose target is to return a Boolean value indicates whether effective cell could be got from the Fb . Detail operations as follows: Start with a byte in position $Fb.pre+1$, $T_1 \cdots T_n$ are identify as variant long integers. Three numbers will be got: $len(T) = \sum_{i=1}^{n} len(T_i)$:bytes count of $T_1 \cdots T_n$; $realDLen = len - pre - \sum_{i=1}^{n} len(T_i)$:real length of $Data$; $theoryDLen = \sum_{i=1}^{n} val(T_i)$: theory length of $Data$ calculated by $T_1 \cdots T_n$.

$realDLen = theoryDLen$ means one effective deleted cell which hided in the Fb could be extract and True will be returned. Otherwise, False will be returned.

Algorithms

Algorithm 1. First, beginning with $rootPageNum$, lps existing in $B+tree$ will be traversed. As described in previous paper, we will get n $SetFbs_i$ by collecting all fb.

Second, Eq. 1 will be applied in every Fb_j belongs to $SetFbs_i$ .if false returned, jumping to next Fb. if true returned, a cell will be extracted from the Fb using function $getCell(Fb)$ which means getting deleted cell from Fb.

Then, adding the recovered cell into $SetCells$ if it is effective and jumping to step 5. If the cell is invalid, continue to judging next Fb and jump to step 1.

Forth, if there is no effective $cell$ after traversing $SetFbs_i$, add the corresponding fb_i into $SetFailfbs$.

Fifth, $i = i + 1$, judging next $SetFbs_i$.

Algorithm 2. A preliminary result including $SetCells$ and $SetFailfbs$ will be got after the execution of Algorithm1. Two-level mapping will be applied in the $SetFailfbs$: 1) let the judging value of $fb.len$ increase from 4 to $len - 3$ with step size 1. A set is got: $SetTempfbs_i = \{ fb_{ij} \mid fb_{ij}.len = j, j \in (4, len) \}$; 2) Each $tempfb_{ij}$ corresponds one $SetFbs_i$, so we could get a set of $SetFbs_i$ generated by $tempfb_{ij}$ in $SetTempfbs_i$. Merging those $SetFbs_i$, we will get a temp set of judging-states: $SetTempFbs = \bigcup_{i=1}^{n} SetTempFbs_i$. Applying Algorithm 1 in $SetTempFbs$. Specially, using the following step to instead of the step 5 in Algorithm 1: $i = i + 1$, deleting brother $SetTempFbs_i$ (generated from same fb_i), judging next $SetTempFbs_i$.

Algorithm 3. To build $SetFlp$, all tp will be searched beginning from offsets 32,33 of file header; Trying to extract every complete $cell$ in flp_i which belongs to $SetFlp$. If one cell is effective, add it into $SetCells$ as shown in following formula: $SetCells = \bigcup GetCell(flp_i), flp_i \in SetFlp, flp_i.type = 0\text{x}0D$, $GetCell(flp)$ means getting complete $cell$ from flp. Pseudocode as shown in Table 1.

Table 1. Algorithms of data recovering

Algorithm 1:Preliminary Recovering	Algorithm 2:Splitting Analyzing
input: $SetFbs$, n output: $SetCells$, $SetFailfbs$	input: $SetFailfbs$, n output: $SetCells$

for $SetFbs_i$ in $SetFbs$
for Fb_j in $SetFbs_i$
if $Fb_j.pre == 3$
for $i = 0$ to 255
if
$RC(Fb_j, i) != null$ and Effective
add $getCell(Fb_j, i)$ to $SetCells$
isFind = true;
break;
else // $Fb_j.pre == 4 \cdots len$
if $rc(Fb_j) != null$ and Effective
add $getCell(Fb_i)$ to $SetCells$
isFind = true;
if isFind == true
break; // end current SetFbs
if isFind == false
add fb_i to $SetFailfbs$
end for
	for fb_i in $SetFailfbs$
	for $len = 4$ to $fb.len$
	$fb_i.len = len$
	execute two-level mapping
	change step 5 and call Algorithm 1

	Algorithm 3:Free pages recovering
	input: tp_0
	output: $SetCells$
	while($tp_i.nextNum \neq 0$) if $getFlp(tp_i).type = 0\text{x}0D$ Add $getFlp(tp_i)$ to $SetFlp$; $i = i + 1$; $tp_i = getNextTp(nextTpAddr)$; **for** flp_j in $SetFlp$ add $GetCell(flp_i)$ in $SetCells$

Experiment

To confirm those algorithms proposed previously, some tests were conducted. Because tables' key types were different, two groups of tests were executed. One was for the integer key while another one for the text key(No-Intkey). In each group, the following operations were executed: 1) Generating 4 tables that had same structures and each table occupied one database file. 2) The original records count of those were 10000, 7500, 5000 and 2500 respectively. 3) Imitating a user, deleting and inserting records randomly until records count was 0. 4) Appling Algorithm 1, 2 and 3 in those tables, r and R were obtained, and R 's value would be renewed after executing Algorithm 3.Statistical results of r are shown in Fig. 6 and Fig 7.

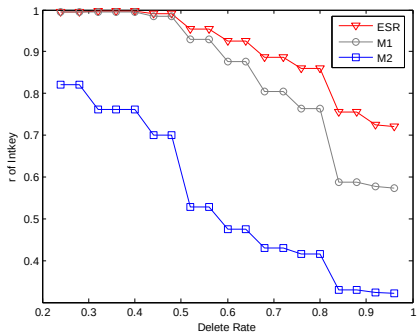


Figure 6. trend of r of Intkey

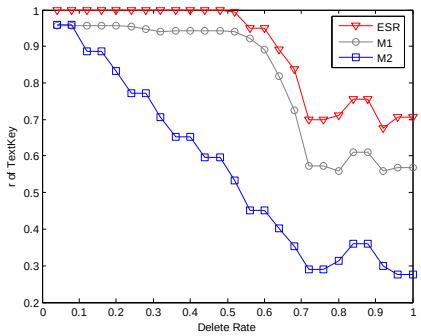


Figure 7. trend of r of No-Intkey

It was evident from the result that relative recovery rate r kept at a high level in most cases. r of M1(proposed in references [8]) were 84.826% and 81.273%, in the meantime, M2's (proposed in references [9]) were 54.323% and 55.738%. The r in ESR are 90.403% and 89.321%, were much higher than previous in many cases.

In this case, which fbs were coalesced or renewed as the results of deleting rate increasing, estimating T_1, splitting fb were mentioned in ESR method to improve r .

As shown in Fig. 8, it is consistent with being discussed previously that R reduced to 0 gradually even the corresponding r remains steady. With increasing of deleting rate, defragment operations were executed to reduce the number of fb . Extracting data from fp is necessary. Fig. 9 shows the Algorithm 3 result considering fp .

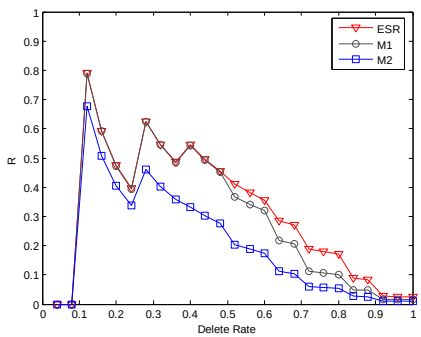


Figure 8. trend of R

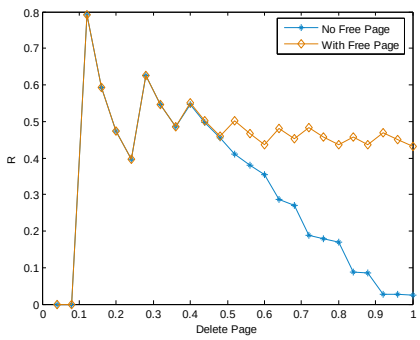


Figure 9. R with algorithm 3

As anticipated, even though R in Algorithm 2 reduced to 0, R seems to maintaining a high level by adopting Algorithm 3 in most cases. Average of it is 45.53% which higher than 31.596% generated by Algorithm 2.

Conclusions

We have discussed the structure of SQLite and the deletion mechanism, and explained the differences between Intkey and No-Intkey. This paper provides an effective method named ESR to recover data, which based on, estimating T_1, splitting fb and reading fp. As evident from the last four figures, this method improves the recovery rate for tables whose keys are the integer. Especially, it is much more valid than other methods in the case of No-Intkey. ESR also shows a superior result in the case of high deletion rate. Future work will focus on extracting data from free space and auxiliary files such as WAL, journal file, etc.

Acknowledgement

The research is supported by Innovation Fund for Technology based Firms (Project No. 13C26215115075).

References

[1] Haerder T, Reuter A. Principles of transaction-oriented database recovery[J]. ACM Computing Surveys (CSUR), 1983, 15(4): 287-317.

[2] Pereira M T. Forensic analysis of the Firefox 3 Internet history and recovery of deleted SQLite records[J]. Digital Investigation, 2009, 5(3): 93-103.

[3] Ramisch F, Rieger M. Recovery of SQLite Data Using Expired Indexes[C]//IT Security Incident Management & IT Forensics (IMF), 2015 Ninth International Conference on. IEEE, 2015: 19-25.

[4] Anglano C. Forensic analysis of WhatsApp Messenger on Android smartphones[J]. Digital Investigation, 2014, 11(3): 201-213.

[5] Li Q, Hu X, Wu H. Database management strategy and recovery methods of Android[C]//Software Engineering and Service Science (ICSESS), 2014 5th IEEE International Conference on. IEEE, 2014: 727-730.

[6] Wu B, Xu M, Zhang H, et al. A recovery approach for SQLite history recorders from YAFFS2[C]//Information and Communication Technology-EurAsia Conference. Springer Berlin Heidelberg, 2013: 295-299.

[7] Jeon S, Bang J, Byun K, et al. A recovery method of deleted record for SQLite database[J]. Personal and Ubiquitous Computing, 2012, 16(6): 707-715.

[8] Fang Q, Zhang Q, Dong R. Research on Recovery Method of Deleted Data for Android System [J]. Computer Engineering, 2014, 40(10): 275-280.

[9] Bai J, Sun H, Hu Z. A recovery Method of Deleted Data Based on SQLite3 File Format [J]. Journal of Chinese Computer System, 2016, 37(3): 505-509.

[10] Information on https://sqlite.org/fileformat2.html

[11] Owens M. The Definitive Guide to SQLite[M]. New York, USA: Springer-Verlag, 2006