Secure Deletion: Towards Tailor-Made Privacy in Database Systems

Alexander Grebhahn[1], Martin Schäler[2], Veit Köppen[2]

[1] Brandenburg University of Applied Sciences
P.O. Box 2132, 14737 Brandenburg, Germany
grebhahn@fh-brandenburg.de

[2] Department of Technical and Business Information Systems
Otto-von-Guericke University Magdeburg
P.O. Box 4120, 39016 Magdeburg, Germany
{schaeler, vkoeppen}@iti.cs.uni-magdeburg.de

Abstract: In order to ensure a secure data life cycle, it is necessary to delete sensitive data in a forensic secure way. Current state of the art in common database systems is not to provide secure deletion at all. There exist academic demonstrators that address some aspects of secure deletion. However, they are limited to their deletion approach. We argue, due to different data sensitivity levels (probably even on attribute level) and differences in policies (e.g., time when and how a data item has to be deleted), it is necessary to have a standardized, user defined opportunity to enforce secure data deletion in a forensic secure manner. Our literature analysis reveals that most approaches are based on overwriting the data. Thus, in this paper, we examine how it is possible to integrate user-defined overwriting procedures to allow a customizable deletion process based on existing default interfaces to minimize the integration overhead. In general, we propose an extension of SQL and a page propagation strategy allowing the integration of a user defined deletion procedure.

1 Introduction

The amount of data stored in computer-aided systems increases continuously. This inherits challenges with respect to privacy of sensitive data. Laws and guidelines exist for handling private information, like the HIPAA [Con96] or the Directive 2006/24/EC of the European Parliament [Eur06]. Forensic secure data deletion is an important privacy challenge. For instance, due to mentioned EU directive, it is only allowed to store data for a fixed period of time. After this, data have to be depersonalized or removed. Consequently, to ensure a forensic secure life cycle data have to be deleted in a forensic secure way [DW10].

Additionally, it is necessary to use a system that supports access control, additional security mechanisms (e.g., public key infrastructures), and recovery components, to store sensitive data. Because database systems support these requirements, they are a good choice for storing sensitive data. That is why we focus on database systems, in this paper.

Using a database system, new challenges for secure deletion arise. These challenges are due to copies of original data, for instance, in backups, roll back segments used in transaction management, or due to implementation details of database operations. To quantify the impact of these challenges or to enable forensic investigations in database system, there have been several examinations of existing database systems (e.g., [Fow08]), revealing that there is currently no secure deletion at all. Thus, there are several approaches proposed in literature that are mainly based on overwriting the data in a predefined way that is limited to this approach and is enforced for all data in the system (e.g., [SML07]).

However, there are use cases, where it is not useful to delete all data stored in a database in a forensically secure way, apply alternative overwriting patterns, or to delete specific data after different time slots. Typical reasons are different sensitivities of data, performance issues, new research results, or changing guidelines for secure deletion. As a result, it is necessary to have a choice to enforce user-defined, fine-grained data-privacy constraints that state the way of forensic secure deletion.

In this paper, we examine: How to integrate such user-defined, fine-grained data-privacy constraints based on existing standard interfaces? Moreover, we want to provide a solution that optimally introduces no or a limited performance overhead.

The remainder of this paper is structured as follows: In Section 2, we present background information of deleting data from a database system. Beside this, we give an overview of different data artifacts that remains in the database system. In Section 3, we give an overview of related work addressing forensic secure deletion from a hard disk and present results and limitations of existing prototypes. In Section 4, we present some SQL extensions to have tailor-made data privacy within a database system. Additionally, we present requirements on the propagation strategy used in a database system to forensic deleting data. Within this part, we also present some possibilities to improve performance of this strategy. Finally, we conclude our paper and present future work.

2 Background

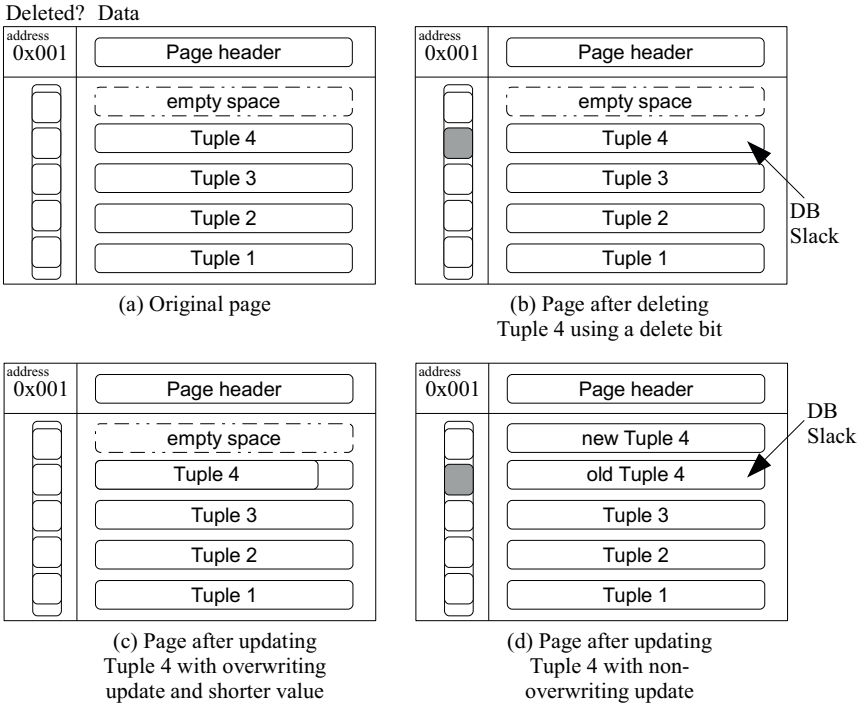
In the following, we list known challenges for forensic secure deletion from a database system to provide an overview of the complexity of this topic. Major challenges are hidden direct copies of data items or that some parts of the data are used (possibly in aggregated form) to create specific data structures out of the database system: For instance, such as indexes to improve performance of the overall system. Last, derived from the presented challenges, we present challenges, that are not in the scope of this paper.

2.1 Deletion of tuples from the database itself

With database, we mean the physical representation of data on persistent storage (e.g., hard disk) in the format of the database system. This is the obvious copy, we have to remove. However, in databases, supporting SQL, there are several operations that, in fact, require

deletion of a data item. Examples for these operations are: `DELETE`, `DROP TABLE`, `TRUNCATE TABLE`, `VACUUM`, and `UPDATE`. In the following, we unify our explanations for `DELETE`, `DROP TABLE`, `TRUNCATE TABLE`, since the last two operations enforce a deletion of all tuples equivalent to a `DELETE FROM <table>` command.

Pages in databases. Before we start to discuss details of the implementation of the single operations, we explain *pages*. In databases, a page is an abstraction of a physical segment of the persistent storage and the elementary unit that can be read or written. Consequently, whenever the system needs to access a single tuple the whole page is read. Hence, all our explanations are based on this elementary unit. In Figure 1.(a), we give an example for such a page. In our example, tuples are inserted from bottom to top.



(a) Original page

(b) Page after deleting Tuple 4 using a delete bit

(c) Page after updating Tuple 4 with overwriting update and shorter value

(d) Page after updating Tuple 4 with non-overwriting update

Figure 1: Creation of database slacks.

DB Slacks: Deleting a tuple. In databases, deleting a tuple usually means setting a delete bit such as in SQL Server [Fow08], Oracle 10g [Lit07a], MySQL 5.1.32 using the InnoDB storage engine [FHMW10, SML07], or PostgreSQL [Gre12]. In Figure 1.(b), we exemplary delete *Tuple 4* from Figure 1.(a) causing no change of the original tuple, but a modification of the delete bit. Hence, with simple analysis of databases and basic knowledge on database internals, we can reconstruct the deleted data item [SML07, Gre12,

FHMW10, Lit07a]. According to Stahlberg et al., we refer to these remains of a tuple as *database (DB) Slack* in the sequel.

DB Slacks: Updating a tuple. By updating a tuple, two different modifications of the page the item is located on can occur [SML07]. Firstly, the old version may be overwritten by its newer version (Figure 1.(c)). Secondly, the new version of the data item is stored at another physical storage segment (Figure 1.(d)). This new location is not necessarily at the same page. As result of the second alternative, the old version of the data item still exists after updating it. Even with the first behavior, information of the old version may remain on the disk, if the new version of the data item needs less space than the old one. In many major database systems, such as PostgreSQL[1], MySQL using the InnoDB storage engine, the second variant because they use the multi-version concurrency control protocol [SML07].

FS Slacks: The Vacuum command. The existence of a large amount of DB Slacks consumes unnecessary disk space and reduces overall performance of database systems, because there are deleted data items on pages which have to be read/written. To overcome this challenge there are operations like `VACUUM` (e.g., in PostgreSQL) that minimizes the space used by database systems. When performing this operation, firstly, active tuples are reorganized to minimize the overall number of needed pages. Secondly, disk space of no more needed pages is returned to the file system. We call these pages and respective tuples according to Stahlberg et al. [SML07] *files system (FS) Slacks*. To sum up, to delete a tuple in a forensic secure way, the database system either has to prevent or to be aware of all copies of a tuple that exist in DB Slacks or FS Slacks. If these copies exist, they have to be deleted too.

2.2 Hidden copies in database systems

Besides database, metadata of a database system and the log file have to be considered. In general, metadata information can be divided in two groups [Gre12]. The first group are data *independent* information not relevant for secure deletion. Examples for this group are table schemata or integrity constraints. The second group, however, data *dependent* information relevant for secure deletion. Examples for the second group are histograms and indexes.

Histograms. In contrast to the database itself, metadata do not necessarily contain a whole copy of a data item. For example, in a histogram only information of one column of data items is stored. In general, considering histograms, it is only possible to retrieve information in which range, the value of a deleted tuple was located. Nevertheless, in some database system (e.g. SQL Server [Fow08]), the values of existing tuples are used

[1] http://www.postgresql.org/docs/9.1/static/storage-vm.html

to set the border of two histogram buckets. Thus, by deleting the tuple, it is necessary to modify the border of the buckets.

Indexes. Moreover, index structures have to be considered, too. One of the most famous index structure is the B^+-Tree. In a B^+-Tree, values of tuples are used as decision criterion for searching in the tree. That means, deleting these values from the database, it is necessary to modify the structure of the tree, too. In worst case, this reconstruction can cause a complete new construction of the tree.

Materialization of intermediate results. Some database systems store additional data dependent information. Examples for this are intermediate results, stored for speed up query performance. A database system storing this information is MonetDB [IKNG09]. From this information, conclusions about values of deleted data items can be drawn. As a result, it is necessary to consider this information by forensic deleting data items.

Database log. Finally, database logs have to be considered. Here, it is necessary to differentiate whether a logical or a physical log protocol is used. By using a physical log protocol, for every modification a BEFORE image of the unmodified data item is stored. In contrast, in a logical protocol, functions are stored which describe modifications to be executed. At this point, one has to be aware, such functions cannot only modify one item [BHG87]. As a result, different solutions have to be used for the different protocols.

2.3 Challenges beyond the scope of this paper.

Besides the challenges we introduced so far, there are several effects that raise the difficulty for forensic secure deletion. However, these challenges are, for instance, due to operations system behavior and thus, it is hardly possible to address them by means of a database system. Although, these challenges are important, they are beyond the scope of this paper:

- Swap main-memory to persistent storage.
- Copying a page to a different location on the persistent storage device like on SSDs [WGSS11].
- Treatment of copies in backups, for instance, stored magnetic tapes.
- Reconstruction of the data through the use of microscopes like in [WKSR08]

3 Analysis of related work

In this paper, we do not want to provide a new approach how to delete data in a specific database, but abstraction from current solutions to provide a more general approach.

Furthermore, we want to introduce a general solution that can be integrated into existing database systems due to the modification of default interfaces. Therefore, we analyze existing approaches to identify similarities to provide such a generalized solution as well as to point out shortcomings of the state of the art to motivate the necessity of our new approach. As a consequence, we first present related work on forensic secure deletion from a hard disk in general.

Forensic secure deletion from hard disk Deleting data in a database system means removing the data traces from persistent memory (e.g., HDD). Thus, we have to consider related work on secure deletion from disk as well. There are approaches to physically destroy the storage medium and modifications that affect neighboring storage cells that shall not be removed. However, these approaches are infeasible for our purpose. Moreover, cryptographic approaches rely on keeping the encryption key secret and there may be a successful attack to break the encryption in the near future. Hence, we do not consider these approaches. Finally, there has been a lot of research how to overwrite data in a way it cannot be restored (e.g., [Gut96, WKSR08, DW10]). Since this approach allows secure deletion without destroying the storage medium, and is possible to implement with means of the database system as shown by Stahlberg [SML07] and demonstrated in our own work [Gre12], we consider it as the *appropriate technique for forensic secure deletion*. However, there are different results on how to overwrite data including the number of overwrites (PASSES) and bit sequences (PATTERN) that have to be applied [DW10]. For instance, Gutman suggests performing 35 PASSES using predefined and random PATTERN. In contrast to this, Wright et al. proposes, that one overwriting pass with random data is adequate for secure deletion [WKSR08].

Database specific forensic analysis Within the last years, a lot of work was done to analyze, which information from a database system can be used to allow forensic examinations.

Different database systems store different specific information with different granularities, most of the work thus focuses on specific database systems. For instance, Lichfield analyzes different parts of oracle database[2]. In some parts of his examination, he focus for example on dropped objects [Lit07a] or on undo segments [Lit07b]. Fowler analyzes the information stored by SQL Server [Fow08]. Frühwirt et al. describe the structure of MySQL Database 5.1.32. with InnoDB Storage Engine [FHMW10]. In their work, they give a detailed overview on the structure of the information and show how to use this information to reconstruct parts of a database table after data items are deleted. However, since the main goal of all these work is to enable forensic examinations of databases, which is in direct opposition to our goal of forensic secure deletion, they do not provide solutions how to perform such a deletion. Nevertheless, their basic research is a fundamental prerequisite to enable forensic secure deletion.

[2] http://www.databasesecurity.com/oracle-forensics.htm

Solutions for forensic secure deletion in database systems. Miklau et al. consider forensic deletion of items from a database in a more theoretical way [MLS07]. They examine, which challenges a database designer have to handle with, by creating a database system that has a securely management of its history.

To the best of our knowledge, Stahlberg et al. [SML07] have conducted the most comprehensive study and provide the most general solution for forensic secure deletion. They consider secure deletion of data items for MySQL using the InnoDB storage engine. Their solution for forensic deletion is *overwriting* values of the item at the moment the storage space of the data is set to be free for storing new items. Furthermore, they considered one index to highlight generality of their solution. When evaluating their method, they could not prove a performance penalty, because always the whole page is written to persistent storage independent whether a delete bit is changed or the data itself shall be overwritten. As a result, overwriting values of a deleted data item in MySQL using InnoDB can be done without performance penalties. However, even their solution is restricted to one (common) database system, and does not address different granularities for forensic secure deletion or temporal constraints when a tuple has to be deleted.

In prior work, we show that Stahlberg's solution also works for different database systems [Gre12]. Firstly, we modify PostgreSQL version 9.1. In our implementation, we overwrite the values of the data items at the time, when no transaction is performed that can access the data to delete. At this moment, the whole page of the item is written to disk. For that reason, we do not measure a performance penalty. So, the solution presented by Stahlberg et al. can applied to PostgreSQL. Secondly, we extend HyperSQL[3] with this method. Evaluating our extension, we recognize a time penalty, because of the page propagation strategy of HyperSQL. Results of a committed transaction are not materialized at commit time due to performance issues. The results are materialized, when the item was not used recently and the buffer, the results are stored in exceeds the assigned main memory. In worst case, this is never done, because of high amount of main memory. Thus, we modify the propagation strategy in a way that after each committed transaction, deleted and old versions of updated items are overwritten. Because of this modification, we introduce an additional I/O operation. We notice a nearly constant performance penalty, when measuring the performance of our modified HyperSQL version.

Additionally, to support a secure data management where no unnecessary information about history of the database is stored in metadata, history-independent index structures have to be used. Within this type of structures no information is stored about operation sequence that lead to current state of the index [NT01].

To sum up, implementing an overwriting strategy for forensic secure deletion is possible using a modified page propagation strategy. Moreover, based on the way the propagation is implemented, single overwrites do not affect the performance at all or introduce a constant overhead. However, since performance is a critical issue, optimizing page propagation especially with multiple overwrites is an important task.

[3] http://hsqldb.org/

3.1 Limitations and resulting generalization of related work

According to our literature research, current solutions for secure deletion are limited w.r.t. three criteria:

1. Database system specific: All solutions are currently limited to a small number of database systems since they demonstrate the feasibility of the proposed approach.
2. Fixed granularity: All current solutions do not allow a fine-grain definition of the deletion mechanisms to enforce at table or attribute level, nor they are able to define different deletion dates based on data sensitivity.
3. Fixed overwrite procedure: Current approaches specify a fixed overwriting procedure (containing number of PASSES and PATTERN), however due to new research results, evaluation purposes, and nation-specific guideline, such as [Gov06], we need a more flexible approach.

4 User defined privacy

In this section, we describe extensions a database system needs to support different policies defined by guidelines to delete data items forensic secure. Due to limited space, we only focus on two abstract interfaces provided by a database system. Firstly, we focus on the SQL interface. Through this interface, a user communicates with the database system. For offering a user a choice to tailor her privacy constraints, it is necessary to extend the SQL syntax supported by the system. This is because, to the best of our knowledge, tailor-made privacy in not covered by the SQL standard. Secondly, we focus on the layer propagating cached pages on a persistent storage.

In particular our solution offers:

1. Generality: Due to an extension of the SQL language and the page propagation strategy.
2. Granularity: Definition of the deletion procedure at table and attribute level, separately.
3. Tailoring: An SQL extension to specify user-defined overwriting procedures (including PASSES and respective PATTERN).

In Figure 2, we give a schematic overview of the structure. Within the persistent data storage, three tables are located. In table 2 and 3 sensitive data is stored that have to be deleted in a forensically way. Due to the fact, that different policies have to be supported through the deletion performed from table 2 and 3, different delete algorithms have to be supported. We discuss details of our solution in the following sections.

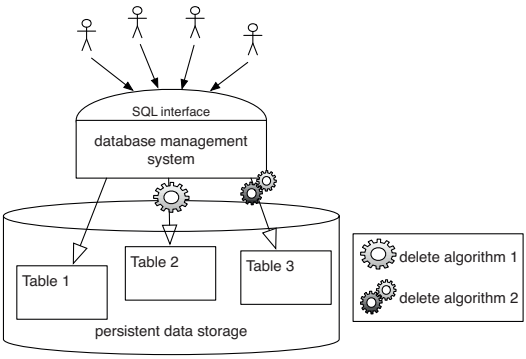


Figure 2: Schematically structure of our solution.

4.1 Tailor-made privacy through SQL commands

In order to have tailor-made privacy and to support different guidelines, it is necessary to offer a user an interface where she can define privacy constraints that have to be supported by the system. Thus, database systems have an interface for communicating with the user, namely the SQL interface, we extend this interface. It is necessary to define different overwriting pattern and to use these pattern in a specific sequence. After defining these pattern and passes, it is necessary to use them to delete a data item in a forensic secure way. As a result, it is not only necessary to insert new SQL commands, but also to extend existing one.

In general, there are two different moments when to propagate to the system, that data items have to be deleted forensic secure. Firstly, in creating the database and secondly, when performing a modification on the data item. Well, before creating the database, one has to be aware of the information stored and the guidelines that have to be supported. Thus, it is possible to extend the syntax of the `CREATE TABLE` statement. Additionally, when performing operations that have an impact on the physical representation of data items from different domains, one has to be aware of all regulations, if they are not propagated at creation time of the tables. In other words, if for example a `VACUUM` operation has to be performed on all tables, a user has to extend the SQL command with all needed constraints defined through policies. In databases having many tables, this can become very confusing. Moreover, if privacy constraints are known before inserting data, the amount of persistent stored metadata can be minimized. Thus, it is possible, not to store indexes or histograms persistent. As a result, we focus on SQL solutions propagating privacy constraints as soon as possible.

To summarize, for supporting a fine-grain, tailor-made forensic deletion of data items, the following definitions have to be supported via SQL interface:

1. Definition of pattern and overwriting passes,
2. Definition of secure tables, using predefined passes, and

3. Selective deletion of data items and columns of data after a fixed time

Definition of pattern and overwriting passes. In Listing 1, we present an extension of the SQL syntax for defining specific pattern to overwrite data. In detail, we extend SQL with two additional keywords, first `PATTERN` and second `WITH`. With the `PATTERN` keyword, it is possible to define a new overwrite pattern with a name (e.g., `p1` in Line 1). To define the design of this pattern, we use the keyword `WITH`. In this statement, it is possible to define sequences of `0`s and `1`s (see, for example, Line 1 and Line 2 in Listing 1). Additionally, it has to be possible, to reuse existing predefined pattern (Line 3). Due to the fact that pattern have to be used for data items of different length, pattern have to be repeated until the whole item is overwritten.

Listing 1: Definition of pattern.

```
1 CREATE PATTERN p1 WITH 0;
2 CREATE PATTERN p2 WITH 100;
3 CREATE PATTERN p3 WITH p1,p2;
```

After defining pattern, it is necessary to use these patterns in a defined sequence. For defining these sequences, we propose the syntax of the SQL statements given in Listing 2. We propose the `PASS` keyword. By the use of this keyword, new sequences of pattern can be defined for forensically deleting data items. Again, we use the `WITH` keyword, as proposed earlier, to define the design. In the `WITH` clause, it should be possible, (a) to use predefined pattern, (b) to define new pattern, (c) to define a pass containing of random data (`RANDOM()` see Line 1 in Listing 2), and (d) to reuse already defined passes (see for example Line 2).

In Line 1, we give an example for the syntax of a pass supporting the ”Declassifying Electronic Data Storage Devices” of Government of Canada [Gov06], to delete protected information of class B (”Compromise could result in grave injury, such as loss of reputation or competitive advantage”)[4]. Within three passes, first, data is overwritten with 0s, second with 1s, and last with a pseudo random pass.

Listing 2: Definitions of overwriting passes.

```
1 CREATE PASS over1 WITH p1, 1, RANDOM();
2 CREATE PASS over2 WITH p2, over1, p3;
```

Definition of a forensic table. Next, it is necessary to use overwriting passes and related pattern for forensic secure deletion data. Thus, we prefer solutions defining sensitive information at creation time of a table, thus we extend the `CREATE TABLE` statement. In Listing 3, we present our syntax for using predefined overwriting passes. As you can see, we extend the SQL syntax with two additional words. Firstly, with the `FORENSIC` keyword, we state that sensitive information is stored in the table. Secondly, by the use of keyword `USE`, we define which overwriting pass has to be used, when forensically deleting data items. To support a fine-grain column wise specification, we give the possibility

[4] http://www.tbs-sct.gc.ca/pol/doc-eng.aspx?id=16557&section=text

to use different overwriting passes for different columns of one table. We give an example in Line 2 in Listing 3.

Listing 3: Usage of overwriting passes in a CREATE TABLE statement.

```
1 CREATE FORENSIC TABLE t1(c1 varchar(40), c2 int) USE
    over1;
2 CREATE FORENSIC TABLE t2(c1 varchar(40) USE over1, c2
    int USE over2);
```

If two different overwriting passes are used within one table, some implementation challenges have to be focused. If the database system uses a column oriented storage, finding the specific storage segment of value of a given column is unproblematic. In contrast to this, if data items are stored tuple wise, identifying the storage region of a data item is much more complex. As a result, is may be necessary, to extend the meta information stored about storage length of data items.

Definition of a forensic table with a fixed time. According to some policies defined through guidelines (e.g. Telekomminikationsgesetz [Bun04]), it is necessary to delete information after a fixed time. For automatic forensic data deletion after given time, it is necessary to know when and how to delete. In Listing 4, we provide an example. As stated earlier, we use again the `FORENSIC` and the `USE` keyword and extend the syntax with the `FOR` keyword. In the part of this keyword, it is possible to define a time slot a data item retrains in the database. For internal management of this constraint, it is possible to use existing techniques, like jobs from an oracle database system. Moreover, it may be necessary to forensic delete only a column of data items after a fixed time, while the rest of the item still remains in the database. Well, this is only allowed, if null is allowed as value of the column. However, this partial deletion brings additional challenges in propagation of the data item, we discuss later.

Listing 4: Storage for a specific time.

```
1 CREATE FORENSIC TABLE t3(c1 varchar(40), c2 int) USE
    over1 FOR 10*60*24;
2 CREATE FORENSIC TABLE t4(c1 varchar(40) USE over1 FOR
    10*60*24, c2 int);
```

4.2 Propagation of data items

After offering the user the possibility to define several overwriting passes for forensic data deletion, it is necessary to perform multiple propagation of a page to overwrite the storage segments multiple times. As a result, it is necessary to modify the propagation strategy used of the database system. In contrast to existing propagation strategies, it is necessary to differentiate between inserting a new data item on a page and deleting an existing one. This is simply because, when inserting a new item, it is only necessary to propagate the

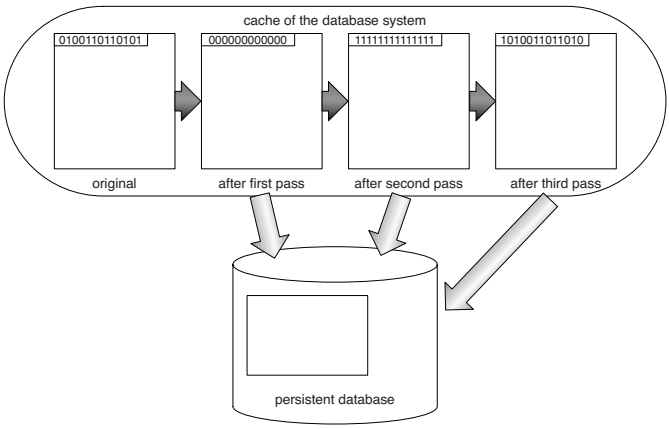


Figure 3: Schematic overview of propagation performed for a page to delete an item.

item once. In contrast to this, it is necessary to overwrite the physical storage segment of a data item multiple times if the item has to be deleted. In Figure 3 we give a schematic overview of the multiple propagation of one page to delete a data item. In detail, we show exemplarily the propagation needed to support the "Declassifying Electronic Data Storage Devices" of Government of Canada [Gov06]. First, after the data item was deleted from the original page, the space used by the data item is overwritten with 0s. Next, it is necessary to propagate the page with the overwritten data. After propagating the page, the storage segment of the page where the item is located have to be overwritten with 1s, then again propagated. Last, the segment is overwritten with random data and propagated again. Beside `INSERT` and `DELETE`, there are other operations having an impact on the physical representation of a data item. On the pages level, each operation can be split into insert and delete operations. For example, an update of a data item is nothing more than an insert of the new version and a delete of the old one. As a result, the old version has to be overwritten multiple times and the new one has to be inserted once. Thus, in this section, we focus only on delete and insert of data items on a page.

Propagation of data modifications. As stated earlier, for supporting a policy, it is necessary to overwrite storage segments multiple times with different predefined pattern in a defined sequence. Thus, each write operation performs an additional I/O-operation, performance of the database system decreases dramatically, if synchronous page propagation is performed. To overcome this, we prefer an asynchronous propagation of modified data items. Through this, it is possible to modify data stored at pages while performing deletion of another item stored at the same page. Yet, this asynchronous propagation brings some drawbacks for user experience. This is because, there is a time delay between deleting a data item through a SQL command with receiving feedback from the system and the real physical overwriting of the data, in write intensive systems. Consequently, it is necessary to define a maximum time delay that is allowed by the system.

However, due to the fact that multiple operations with different pattern have to be per-

formed to forensically delete one item, it is necessary to store additional information on the page. For example, it is necessary to store number of already performed overwriting operation for a data item, because after a system error occurred, it is necessary to continue with not finished overwriting passes.

Partial deletion from data items. After performing a forensic deletion of a whole data item, it is uninteresting, which values are stored at the storage location of the data item. In contrast to this, when performing a partial secure deletion, some values might bring problem. This is simply because, the rest of the data item has to be readable after deleting the values of the column and overwriting parts of the data random values might produce unpredictable side effects. To overcome this, different solutions can be performed. Firstly, after all write operations are performed to delete the data secure, an additional pass containing of binary nulls can be performed. Secondly, the new version of the data item can be stored at a new storage segment without the values of the column of interest and the old version of the item can overwritten with the defined pattern.

One disadvantage of the first version is the additional I/O-operation performed in marking the no more used storage segment as null. An additional disadvantage is the unused storage segment that might occur within data items. For example, after the secure deletion of a value of Column c1 from a data item stored in Table t4 (see Listing 4), the data item only needs space to mark that c1 is null and to store value of c2. In contrast to the required space, the data item uses the space it needs before deleting c1. In contrast to this, the second solution overcomes this problem. A drawback of the second solution is the modification of the location the data was stored at. As a result, it is necessary to update all references pointing to the old location of the data item. To sum it up, there are different solutions how to perform a secure deletion of parts of a data item.

Optimizations of the propagation. To minimize amount of write operations, it is possible to merge propagation of different data items stored at the same page. Especially in write intensive systems, this brings a huge performance benefit. In general, merging two different operations is only possible, if different storage segments are used for the data items. For example, it is only possible to store a new data item at a specific storage segment after all overwriting passes of an old data item former located at the same storage segment where performed. However, when merging propagation operations of two or more data items, one have to differentiate between (1) insert and delete operation performed on the same page and (2) multiple delete operations. For simplicity, we only describe how to merge two operations, but the proposed method can be extended to more than two data items without loss of generality.

1. If a new data item is added to the page, while performing the delete passes of another one, it is possible, to integrate the propagation of the new item into the I/O-operation the next deletion pass of the old one performs.

2. If more than one data item has to be deleted from the page, it is possible to merge passes of the delete operations. In detail, let i be the number of deletion passes

already performed for the first data item, j be the number of passes that have to be performed for every data item and P_i the pattern for the ith delete operation. Then perform delete operation $i + k$ $(1 \leq k \leq j)$ with pattern P_{i+k} for the first data item and operation P_k for the second data item with the same I/O-operation.

Until now, merging of propagation is only performed if two operations having an impact on the same page have to be performed at the same time. To minimize number of I/O-operations in systems having less write operations, it might be possible to wait with the write operations of one data item for a specific time until another data item has to be deleted from the same page.

5 Conclusion & Future work

In this paper, we explained what has to be considered, when forensically deleting data from a database system. In particular, it is necessary to delete information of all data inventories created by the system. To support deletion strategies proposed in different guidelines in one database system, it is necessary to have the possibility to perform multiple propagation of a modified data item. As a result, we propose a propagation strategy. For giving a user the possibility to tailor her privacy constraints, we extend the SQL syntax with (a) the possibility to define overwriting pattern, and passes, (b) define forensic tables and (c) to extend definition of forensic tables for storing data only for a fixed time.

In future work, we want to extend HSQLDB with the presented propagation strategy, extend the SQL syntax, and evaluate the performance. Due to the fact that different database operations have different number of I/O-operations performed by the propagation strategy, we want to perform benchmarks and evaluate performance.

6 Acknowledgments

The work in this paper has been funded in part by the German Federal Ministry of Education and Science (BMBF) through the Research Programme under Contract No. FKZ:13N10816 and FKZ:13N10817.

References

[BHG87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[Bun04] Bundesministeriums der Justiz. Telekommunikationsgesetz, 2004.

[Con96] United States Congress. Health Insurance Portability and Accountability Act (HIPAA), 1996.

[DW10] Sarah M. Diesburg and An-I Andy Wang. A survey of confidential data storage and deletion methods. *ACM Compututing Surveys*, 43(1):2:1–2:37, December 2010.

[Eur06] European Parliament and the Council of the European Union. Directive 2006/24/EC of the European Parliament and of the Council of 15 March 2006 on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending Directive 2002/58/EC. *Official Journal of the European Union*, L 105:0054–0063, 2006.

[FHMW10] Peter Frühwirt, Marcus Huber, Martin Mulazzani, and Edgar R. Weippl. InnoDB Database Forensics. In *AINA*, pages 1028–1036, Washington, DC, USA, 2010. IEEE Computer Society.

[Fow08] Kevvie Fowler. *SQL Server Forensic Analysis*. Addison-Wesley Professional, 2008.

[Gov06] Government of Canada. Clearing And Declassifying Electronic Data Storage Devices (ITSG-06). UNCLASSIFIED publication, issued under the authority of the Chief, Communications Security Establishment (CSE), July 2006.

[Gre12] Alexander Grebhahn. Forensisch sicheres Löschen in relationalen Datenbankmanagementsystemen. Master thesis, University of Magdeburg, Germany, 2012. In German.

[Gut96] Peter Gutmann. Secure Deletion of Data from Magnetic and Solid-State Memory. In *USENIX Security*, pages 77–89, 1996.

[IKNG09] Milena G. Ivanova, Martin L. Kersten, Niels J. Nes, and Romulo A.P. Gonçalves. An Architecture for Recycling Intermediates in a Column-store. In *SIGMOD*, pages 309–320, New York, NY, USA, 2009. ACM.

[Lit07a] David Litchfield. Oracle Forensics Part 2: Locating Dropped Objects. *NGSSoftware Insight Security Research (NISR) Publication, Next Generation Security Software*, 2007.

[Lit07b] David Litchfield. Oracle Forensics Part 6: Examining Undo Segments, Flashback and the Oracle Recycle Bin. *NGSSoftware Insight Security Research (NISR) Publication, Next Generation Security Software*, 2007.

[MLS07] Gerome Miklau, Brian Neil Levine, and Patrick Stahlberg. Securing history: Privacy and accountability in database systems. In *CIDR*, pages 387–396, 2007.

[NT01] Moni Naor and Vanessa Teague. Anti-persistence: History Independent Data Structures. *IACR Cryptology ePrint Archive*, 2001:36, 2001.

[SML07] Patrick Stahlberg, Gerome Miklau, and Brian Neil Levine. Threats to Privacy in the Forensic Analysis of Database Systems. In *SIGMOD*, pages 91–102, New York, NY, USA, 2007. ACM.

[WGSS11] Michael Yung Chung Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. Reliably Erasing Data from Flash-Based Solid State Drives. In *FAST*, pages 105–117. USENIX, 2011.

[WKSR08] Craig Wright, Dave Kleiman, and Shyaam Sundhar R.S. Overwriting Hard Drive Data: The Great Wiping Controversy. In *ICISS*, pages 243–257, Berlin, Heidelberg, 2008. Springer-Verlag.