Separation of Duties for Multiple Relations in Cloud Databases as an Optimization Problem

Ferdinand Bollwein
Clausthal University of Technology
Institute of Applied Stochastics and Operations Research
Clausthal-Zellerfeld, Germany
ferdinand.bollwein@tu-clausthal.de

Lena Wiese
University of Goettingen
Institute of Computer Science
Goettingen, Germany
wiese@cs.uni-goettingen.de

ABSTRACT

Confidentiality concerns are important in the context of cloud databases. In this paper, the technique of vertical fragmentation is explored to break sensitive associations between columns of several database tables according to confidentiality constraints. By storing insensitive portions of the database at different non-communicating servers it is possible to overcome confidentiality concerns. In addition, visibility constraints and data dependencies are supported. Moreover, to provide some control over the distribution of columns among different servers, novel closeness constraints are introduced. Finding confidentiality-preserving fragmentations is studied in the context of mathematical optimization and a corresponding integer linear program formulation is presented. Benchmarks were performed to evaluate the suitability of our approach.

CCS CONCEPTS

• **Security and privacy** → *Database and storage security*;

KEYWORDS

Separation of Duties, Vertical Fragmentation, Integer Linear Program

ACM Reference format:
Ferdinand Bollwein and Lena Wiese. 2017. Separation of Duties for Multiple Relations in Cloud Databases as an Optimization Problem. In *Proceedings of IDEAS '17, Bristol, United Kingdom, July 12-14, 2017,* 10 pages.
https://doi.org/10.1145/3105831.3105873

1 INTRODUCTION

Many cloud database providers not only offer flexible and scalable services, but also a wide range of possibilities to process the stored data. This spares the cost of maintaining an own computing center. However, outsourcing data also means that the user loses control over the data and the service providers have to be trusted in order to ensure confidentiality. Hence, outsourcing sensitive or business-critical data to a single untrusted cloud database provider is often not an option. Obviously, one possibility to ensure confidentiality would be to encrypt the data stored in the cloud database, however, this limits the provider's ability of processing the data to answer complex queries by the user. For this reason, this work proposes a *separation of duties* approach to face this problem. Generally, the term separation of duties means that a specific task is handled by multiple entities to prevent malicious behavior which could be carried out by a single entity in control of the whole task. In the context of preserving confidentiality in cloud databases, this is based on the observation that in many scenarios data only becomes sensitive in association with other data. By distributing the data among multiple database servers with a technique called *vertical fragmentation*, these sensitive associations can be broken up such that each server only maintains an insensitive portion of the data. As long as the servers are not collaborating to reestablish the association, this separation of duties approach ensures the privacy of the underlying data. As the data is outsourced in plaintext the servers' ability of performing complex computations is not limited and therefore this approach enables users to perform arbitrary queries on the data. However, this clearly involves that data has to be gathered from the different servers when querying for it and therefore additional query rewriting and optimization steps are necessary.

It is a challenging task to decide how the data is distributed among the servers. On the one hand, the security requirements have to be met but on the other hand the number of involved servers should be relatively small to limit the costs for the user and ensure efficient querying. Therefore, the proposed separation of duties approach can be viewed as a typical mathematical optimization problem. In this paper we analyze three variations of this problem in a multi-relational environment. Moreover, we present a prototype implementation which is capable of distributing databases among different servers and which appropriately analyzes and rewrites arbitrary SQL queries.

In detail, we make the following contributions:

- We analyze vertical fragmentation of multiple relations as a technique to protect data confidentiality in cloud databases and devise a formulation as a mathematical optimization problem. In contrast to prior work we explicitly allow multiple non-communicating servers in the problem and consider the minimization of the number of servers as an optimization goal.
- Moreover, visibility constraints and novel closeness constraints are introduced to improve the usability of the resulting fragmentations and to allow for efficient query answering. Those constraints are modeled as soft constraints – in

IDEAS '17, July 12-14, 2017, Bristol, United Kingdom

ACM ISBN 978-1-4503-5220-8/17/07…$15.00
https://doi.org/10.1145/3105831.3105873

contrast to the confidentiality constraints which are mandatory to satisfy. As a further extension, data dependencies among tables are considered.
- Lastly, a prototype implementation (using the feature of Foreign Data Wrappers in PostgreSQL) is presented which relies on a highly optimized off-the-shelf Integer Linear Program (ILP) solver to find the optimal fragmentation. It allows users to cast queries as if they would query the original non-fragmented database; this relieves the user from rewriting queries to act on fragments instead of the original tables. This implementation is further tested with a well-known data set (TPC-H) and under two scenarios (view-based querying versus query rewriting). To the best of our knowledge this is the first comprehensive benchmark of a confidentiality-preserving vertical fragmentation approach with a TPC data set.

We start this article with a survey of related work in Section 2. Section 3 sets the necessary terminology; Section 4 analyzes the theory of several Separation of Duties problems; Section 5 provides a translation into an integer linear program; Sections 6 and 7 describe the implementation and evaluation; Section 8 concludes the article.

2 RELATED WORK

Security and privacy have been challenging tasks ever since the introduction of the principle of data outsourcing and a lot of research has been carried out to address different aspects of this problem such as access control, data confidentiality and data integrity. Security can be achieved by encryption – carried out by the user before outsourcing the data to the database. Still, encryption operations are generally very costly and moreover, existing cryptographic techniques can still not be efficiently used to evaluate more complex queries like, for instance, computations on the data. In this paper we focus on data confidentiality by data fragmentation and distribution without encryption; however our proposed approach can indeed be combined with conventional encryption (as in [1]) or even novel encryption approaches (as for example order-preserving encryption in [17] or a combination of several encryption types in [16]).

Aggarwal et al. were the first ones to propose vertical fragmentation as a technique to distribute a database table by columns (attributes) among servers to ensure confidentiality in [1]. They model sensitive associations between columns of a single database table as subsets of columns and distribute the table vertically among two distinct *honest-but-curious*, *non-communicating* servers to break these sensitive associations. Additionally, they rely on encryption whenever it is impossible to meet the confidentiality requirements with those two servers only.

Ciriani et al. consider a similar scenario but waive encryption completely in [8] and [4]. To do so, a database table is distributed vertically among an untrusted remote and a trusted local database server. Their aim is to store a minimal subset of columns in the local database such that the remaining set of columns which is stored at the untrusted server is insensitive. Therefore, their approach is also called a *keep-a-few* approach.

Other proposals such as [5–7, 9, 11–13] use vertical fragmentation as a tool to split sensitive associations in a database table. It is required that those fragments do not have an attribute in common such that the fragments are *unlinkable*. Due to the unlinkability, the fragments can possibly be stored at a single untrusted server which is then in possession of all the data but cannot establish sensitive associations.

In [13] the concept of *data dependencies* is introduced. The authors state that certain combinations of attributes can be used by a sophisticated untrusted server to draw conclusion about other attributes which could potentially lead to the exposure of sensitive data.

Expressive constraints and dependencies in first-order logic have previously been analyzed in [2] for vertical as well as in [18] for horizontal confidentiality-preserving fragmentations.

3 BACKGROUND

We focus on the relational database model as the most common model for databases. As usual, a relation schema $R(\{a_1, \ldots, a_n\})$, or simply $R(a_1, \ldots, a_n)$, consists of a *relation name* R and a finite set of *attributes* $\{a_1, \ldots, a_n\}$ with $n \geq 1$. Each attribute a_i is associated with a specific domain which is denoted by the expression $\mathrm{dom}(a_i)$. Next, a *relation* r, also denoted by $r(R)$, over the relation schema $R(a_1, \ldots, a_n)$ is defined as an ordered set of *n-tuples* $r = (t_1, \ldots, t_m)$ such that each *tuple* r_j is an ordered list $t_j = v_1, \ldots, v_n$ of values $v_i \in \mathrm{dom}(a_i)$ or $v_i = \mathsf{NULL}$. The *degree* of a relation is defined as the number if attributes in r.

A *database schema* $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ is defined by a name D and a set of relation schemes $R_i(A_i)$ where A_i denotes the corresponding set of attributes. Finally, a *database state* $d = \{r_1, \ldots, r_N\}$ over a database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ is a set of relations such that each r_i is a relation over the respective relation schema $R_i(A_i)$.

As a tuple identifier tid_s for relation schema R_s a subset of A_s is chosen that is a candidate key. Formally, if r_s is a relation over the schema $R_s(A_s)$ and $\mathrm{tid}_s \subseteq A_s$, for the two tuples $t_1, t_2 \in r_s$ the following holds

$$t_1[\mathrm{tid}_s] = t_2[\mathrm{tid}_s] \Rightarrow t_1 = t_2$$

The set of all tuple identifiers is denoted by $\mathrm{tid} := \bigcup_{s=1}^{N} \mathrm{tid}_s$.

To illustrate the individual steps, a small database consisting of two tables in a hospital scenario serves as a running example. The first table stores information about doctors and the second table stores information about patients:

$$D = \{\mathbf{D}(\mathsf{DocID}, \mathsf{Name}, \mathsf{DoB}, \mathsf{ZIP}, \mathsf{Specialty}),$$
$$\mathbf{P}(\mathsf{PatID}, \mathsf{Name}, \mathsf{DoB}, \mathsf{ZIP}, \mathsf{Diagnosis}, \mathsf{Treatment}, \mathsf{DocID})\}$$

where DocID and PatID serve as tuple identifiers.

4 SEPARATION OF DUTIES PROBLEMS

The presented separation of duties approach aims at protecting confidentiality. Hence, analogous to [1] and [8] we assume that Cloud service providers are "honest but curios". This means that servers handle requests and answer queries correctly; but, while they do not manipulate the stored data, still they analyze data and user behavior and try to gain sensitive information from it.

In order to express security policies, we have to specify which attribute combinations are secret. In a database containing multiple relations, sensitive associations can exist among relations. This is

expressed in the following definition of multi-relational confidentiality constraints:

Definition 4.1. **(Multi-Relational Conf. Constraints)** Let $A_1, \ldots, A_N$ denote pairwise disjoint sets of attributes. Moreover, let $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ be a database schema and $d = \{r_1, \ldots, r_N\}$ a database state over D. A *multi-relational confidentiality constraint* on D is defined by a subset of attributes $c \subseteq \bigcup_{s=1}^{N} A_s$. A multi-relational confidentiality constraint c with $|c| = 1$ is called a *singleton constraint*. If $|c| > 1$ it is called an *association constraint*.

The condition that the set of attributes are pairwise disjoint is introduced to assure that attributes can uniquely be associated with the according relation schema. In reality, this can easily be achieved by choosing a suitable naming convention. For example the names of the relations could be prepended to the attributes.
Vertical fragmentation is used to meet the security requirements expressed by the confidentiality constraints. Completeness ensures that the original data can be reconstructed. Disjointness avoids duplicate storage of attributes in different fragments; the only exception being the tuple identifiers which are needed to recombine the data as required by the reconstruction property.

Definition 4.2. **(Multi-Relational Vertical Fragmentation, cardinality, physical fragments)** Let $A_1, \ldots, A_N$ denote pairwise disjoint sets of attributes. Furthermore, let $d = \{r_1, \ldots, r_N\}$ be a database state over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$. A set $\mathbf{f} = (f_0, \ldots, f_k)$ where $f_j \subseteq \bigcup_{s=1}^{N} A_s$ for all f_j is called a *correct (multi-relational) vertical fragmentation* of d if the following conditions are met:

- **Completeness:** $\bigcup_{i=0}^{k} f_j = \bigcup_{s=1}^{N} A_s$
- **Disjointness:** $f_i \cap f_j \subseteq \text{tid}, \forall\, f_i \neq f_j$ with $f_i, f_j \neq \emptyset$
- **Reconstruction**: $\text{tid}_s \subset (f_j \cap A_s)$, if $f_j \cap A_s \neq \emptyset$

A fragmentation that satisfies all properties except the disjointness is called a *lossless* (multi-relational) vertical fragmentation of r. The *cardinality* of a vertical fragmentation $\text{Card}(\mathbf{f})$ is defined as the number of nonempty elements in $\mathbf{f}$: $\text{Card}(\mathbf{f}) = \sum_{\substack{j=0 \\ f_j \neq \emptyset}}^{k} 1$.
For fragment f_j, its multirelational *physical fragment* is the set of projections

$$d_j := \{\pi_{f_j \cap A_1}(r_1), \ldots, \pi_{f_j \cap A_N}(r_N)\}$$

which is a database over the database schema $D_j = \{R_1(f_j \cap A_1), \ldots, R_N(f_j \cap A_N)\}$. The individual relations $\pi_{f_j \cap A_s}(r_s)$ for $s \in \{1, \ldots, N\}$ are called *relation fragments*.

In this definition the completeness property ensures that every attribute is contained in at least one fragment. The reconstruction property makes sure that a fragment contains all necessary tuple identifiers to reconstruct the original relations by joining the corresponding relation fragments. The disjointness property prevents unnecessary copies of non-tuple identifier attributes. We now discuss three variants of separation of duties by vertical fragmentation.

4.1 Standard Separation of Duties

As a minimum, confidentiality-preserving fragmentations must obey security policies by ensuring that not all attributes contained in a confidentiality constraint at the same time. The exception is the owner fragment that is stored on the trusted client side and contains all singleton constraints as well as subsets of association constraints if they cannot be satisfied by distributing their attributes among several server fragments.

Definition 4.3. **(Confidentiality)** A vertical fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ is *confidentiality-preserving* with respect to a set of confidentiality constraints C if the following condition is met:

$$c \not\subseteq f_j \text{ for all } c \in C \text{ and } j \in \{1, \ldots, k\}$$

In this definition f_0 is the owner fragment (to be stored at the trusted database server) and $f_1, \ldots, f_k$ are the k server fragments (to be stored at the k untrusted database servers). A confidentiality-preserving vertical fragmentation therefore requires that the combination of attributes defined by a confidentiality constraint is *not* jointly visible in a server fragment. For singleton constraints, this implies that the corresponding attribute must be placed in the owner fragment.

To avoid redundancy and unwanted interactions with the tuple identifiers, we impose some restrictions on the set of confidentiality constraints. In particular, tuple identifiers are assumed to be harmless information and hence should not be contained in confidentiality constraints.

Definition 4.4. **(Well-defined Conf. Constraints)** Let $A_1, \ldots, A_N$ denote pairwise disjoint sets of attributes and let $\text{tid}_s \subset A_s$ denote the designated tuple identifier for $A_1, \ldots, A_N$ respectively. Moreover, let $d = \{r_1, \ldots, r_N\}$ denote a database state over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$. A set of confidentiality constraints C is *well-defined* if it satisfies the following conditions:

- $c \not\subseteq c'$ for all $c, c' \in C$ with $c \neq c'$
- $c \cap \text{tid}_s = \emptyset$ for all $c \in C$ and $s \in \{1, \ldots, N\}$ with $c \subseteq A_s$

The first condition requires that no confidentiality constraint c is a subset of another c' – due to the requirements for a confidentiality-preserving multi-relational vertical fragmentation, the restriction that c is not jointly visible in any server fragment already implies that c' is not jointly visible in any server fragment.

The second condition of the previous definition further implies that confidentiality constraints c that contain only attributes from a single relation are not allowed to contain tuple identifier attributes if they contain at least one non-tuple identifier attribute. Such constraints can simply be replaced by the semantically equivalent constraints $c \setminus \text{tid}_s$. This will avoid unnecessary case differentiations in the remainder of this work.

As a last component, we specify a weight function that assigns a weight to each set of attributes: $w_d : \mathcal{P}(A) \longrightarrow \mathbb{R}^+$. A simple weight function could for example count the number of attributes in the set. We then consider a maximum capacity W_j for each server S_j and require that the summed weights of the fragment does not exceed the capacity of the server that hosts this fragment.

With these preliminaries the definition of the (Standard) Multi-relational Separation of Duties Problem is as follows:

Definition 4.5. **(Multi-relational Separation of Duties)** Given a database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$, a database state $d = \{r_1, \ldots, r_N\}$, a well-defined set of multi-relational confidentiality constraints C, tuple identifiers $\text{tid}_s \subset A_s$ for all $s \in \{1, \ldots, N\}$,

a weight function w_d, servers $S_0, \ldots, S_k$ and corresponding maximum capacities $W_0, \ldots, W_k \in \mathbb{R}_0^+$, find a correct confidentiality-preserving fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ of minimal cardinality $\mathrm{Card}(\mathbf{f})$ such that the capacities of the storage are not exceeded, i.e. $w_d(f_j) \leq W_j$ for all $0 \leq j \leq k$.

The maximum capacity W_0 of the owner fragment can be set such that the owner fragment only stores the attributes in the singleton constraints – which cannot be outsourced due to their sensitivity. This can be achieved by choosing a suitable capacity of the owner fragment. Yet, it must be considered that the correct tuple-identifier attributes must also be part of the owner fragment to satisfy the reconstruction property. Therefore, if $A^* := \{c \in C \mid |c| = 1\}$ denotes the set of all sensitive attributes, the right choice for W_0 is given by:

$$W_0 = \sum_{s:A_s \cap A^* \neq \emptyset} w_d(\mathrm{tid}_s) + \sum_{c \in C:|c|=1} w_d(c).$$

We just mention that the Standard Separation of Duties Problem can be viewed as a combination of two famous NP-hard problems, the bin packing problem due to the capacity constraints of the storage locations and the vertex coloring problem due to the confidentiality constraints. Both problems would be suitable for a NP-hardness proof. Due to space restrictions we do not state the proof in this paper.

Continuing our example, for confidentiality constraints

$$C = \{\{\mathbf{P}.\mathsf{Name}\}, \{\mathbf{D}.\mathsf{Name}\}, \{\mathbf{P}.\mathsf{DoB}, \mathbf{P}.\mathsf{ZIP}, \mathbf{P}.\mathsf{Diagnosis}\}, \{\mathbf{D}.\mathsf{DoB}, \mathbf{D}.\mathsf{ZIP}\}\}$$

a confidentiality-preserving fragmentation consists of one owner fragment

$$\{\mathbf{P}_0(\mathsf{PatID}, \mathsf{Name}), \mathbf{D}_0(\mathsf{DocID}, \mathsf{Name})\}$$

and two server fragments

$$\{\mathbf{P}_1(\mathsf{PatID}, \mathsf{DoB}, \mathsf{DocID}), \mathbf{D}_1(\mathsf{DocID}, \mathsf{ZIP})\} \text{ and } \{\mathbf{P}_2(\mathsf{PatID}, \mathsf{ZIP}, \mathsf{Diagnosis}, \mathsf{Treatment}), \mathbf{D}_2(\mathsf{DocID}, \mathsf{DoB}, \mathsf{Specialty})\}.$$

4.2 Visibility and Closeness

In the multi-relational scenario it is very important to control the resulting fragmentations to increase the usability of the fragmented database. Increased usability means that certain combinations of attributes are stored on a single server because they are often queried together. The notion of visibility constraints will be adapted to this scenario: visibility constraints are defined as subsets of attributes that should be placed in a single fragment if possible. Formally, the definition of visibility constraints is as follows:

Definition 4.6. **(Visibility constraint, satisfaction)** Let $d = \{r_1, \ldots, r_N\}$ be a database state over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$. A (multi-relational) *visibility constraint* over D is a subset of attributes $v \subseteq \bigcup_{s=1}^{N} A_s$. A multi-relational fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ satisfies v if there is an $0 \leq j \leq k$ such that $v \subseteq f_j$. If such an f_j exists, define $\mathrm{Sat}_v(\mathbf{f}) := 1$. Otherwise, define $\mathrm{Sat}_v(\mathbf{f}) := 0$. For any set V of visibility constraints, the number of satisfied visibility constraints is

$$\mathrm{Sat}_V(\mathbf{f}) := \sum_{v \in V} \mathrm{Sat}_v(\mathbf{f})$$

Note that as opposed to other approaches we treat visibility constraints as soft constraints; that is, conflicts in the specification are allowed and visibility constraints will be satisfied only if the confidentiality can still be ensured. Hence, it may happen that not all visibility constraints can be satisfied. However, even in the case of an unsatisfied visibility constraint, we want to reduce the amount of servers on which a set of attributes is distributed.

For example, focusing only on our patient example table, a visibility constraint $v = \{\mathsf{DoB}, \mathsf{ZIP}, \mathsf{Diagnosis}\}$ is introduced while to preserve the privacy of the patients, the confidentiality constraint $c = \{\mathsf{DoB}, \mathsf{ZIP}\}$ is enforced. However, because $c \subset v$, the visibility constraint cannot be satisfied. Hence, one possible solution to the problem is given by $\mathbf{f} = \{f_0, f_1, f_2, f_3\}$ with: $f_0 = \emptyset$, $f_1 = \{\mathsf{PatID}, \mathsf{DoB}\}$, $f_2 = \{\mathsf{PatID}, \mathsf{ZIP}, \mathsf{Treatment}\}$, $f_3 = \{\mathsf{PatID}, \mathsf{Diagnosis}\}$.

Another possible solution is given by the fragmentation $\mathbf{f}' = \{f_0', f_1', f_2', f_3'\}$ with: $f_0' = \emptyset$, $f_1' = \{\mathsf{PatID}, \mathsf{DoB}\}$, $f_2' = \{\mathsf{PatID}, \mathsf{ZIP}, \mathsf{Diagnosis}\}$, $f_3' = \{\mathsf{PatID}, \mathsf{Treatment}\}$.

The important thing to notice here is that in $\mathbf{f}$ the attributes in v are spread among three and in $\mathbf{f}'$ among only two servers. As a result, a query for the three attributes DoB, ZIP and Diagnosis involves three servers for the first fragmentation and only two for the second. Hence, the query will be processed faster for the second fragmentation.

To achieve a lower distribution of attributes, we introduce the novel notion of closeness constraints. With these constraints it can be assured that certain sets of attributes are not spread arbitrarily among the fragments which reduces the setup time and the overhead for query answering. For example, we could express a closeness constraint as $\gamma = \{\mathsf{DoB}, \mathsf{ZIP}, \mathsf{Diagnosis}\}$ which enables us to reduce the number of fragments among which this subset of attributes is distributed and hence $\mathbf{f}'$ is a better solution than $\mathbf{f}$.

Definition 4.7. **(Closeness constraint, distribution)** Let $d = \{r_1, \ldots, r_N\}$ be a database state over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$. A *closeness constraint* over D is a subset of attributes $\gamma \subseteq \bigcup_{s=1}^{N} A_s$. Let $\mathbf{f} = (f_0, \ldots, f_k)$ be a vertical fragmentation of d, the distribution $\mathrm{Dist}_\gamma(\mathbf{f})$ of γ is defined as the number of fragments that contain one of the attributes in γ:

$$\mathrm{Dist}_\gamma(\mathbf{f}) := \sum_{\substack{j=0 \\ f_j \cap \gamma \neq \emptyset}}^{k} 1$$

For any set Γ of closeness constraints the distribution $\mathrm{Dist}_\Gamma(\mathbf{f})$ is defined as the sum of distributions of $\gamma \in \Gamma$:

$$\mathrm{Dist}_\Gamma(\mathbf{f}) := \sum_{\gamma \in \Gamma} \mathrm{Dist}_\gamma(\mathbf{f})$$

The minimization of the distribution of the closeness constraints will be the third goal in the following Extended Multi-relational Separation of Duties problem. However, minimizing the number of servers, maximizing the number of fulfilled visibility constraints and minimizing the distribution of the closeness constraints are three separate goals that are not complementary. Hence, a balance has to be found between them. Therefore, the objective stated in the problem definition is expressed as a weighted sum of these three goals using three weights $\alpha_1, \alpha_2, \alpha_3$. On the other hand of course, satisfying the confidentiality constraints will still be mandatory.

Notably, omitting the disjointness property of fragmentation helps increase the number of fulfilled visibility constraints. Therefore, in the following problem statement, only a lossless fragmentation is required.

Definition 4.8. **(Extended Multi-relational Separation of Duties)** Given a database state $d = \{r_1, \ldots, r_N\}$ over the database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$, a well-defined set of multi-relational confidentiality constraints C, visibility constraints V, a set of closeness constraints Γ, tuple identifiers $\text{tid}_s \subseteq A_s$ for all $s \in \{1, \ldots, N\}$, a weight function w_d, storage spaces $S_0, \ldots, S_k$ with maximum capacities $W_0, \ldots, W_k \in \mathbb{R}_0^+$ and positive weights $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{R}_0^+$, find a lossless confidentiality-preserving fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ of minimal cardinality which satisfies $w_d(f_j) \leq W_j$ for all $0 \leq j \leq k$ such that the weighted sum $\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_2 \operatorname{Sat}_V(\mathbf{f}) + \alpha_3 \operatorname{Dist}_\Gamma(\mathbf{f})$ is minimal.

Using the weighted sum serves three purposes:

(1) $\alpha_1 \operatorname{Card}(\mathbf{f})$ is responsible for minimizing the cardinality (amount of fragments) of the fragmentation.
(2) By subtracting $\alpha_2 \operatorname{Sat}_V(\mathbf{f})$ each satisfied visibility constraint will lower the overall objective.
(3) Distribution of the closeness constraints is minimized by $\alpha_3 \operatorname{Dist}_\Gamma(\mathbf{f})$.

The obvious question arises of how to appropriately choose the weights α_1, α_2 and α_3. In the following lemma we suggest a choice for the weights α_1, α_2 and α_3 assigning highest priority to the minimization of the cardinality of the fragmentation. Among those fragmentations the number of satisfied visibility constraints should be maximal and again among those fragmentations the distribution of the closeness constraints should be minimized.

Lemma 4.9. *Consider weights $\alpha_1 > 0$, $\alpha_2 > 0$ and $\alpha_3 > 0$ satisfying (1) $\alpha_2|V| + \alpha_3 k|\Gamma| < \alpha_1$ and (2) $\alpha_3 k|\Gamma| < \alpha_2$. If $\mathbf{f}$ is a solution to the Extended Multi-Relational Single-Relational Separation of Duties Problem and $\mathbf{f}'$ is a lossless confidentiality-preserving fragmentation that does not violate the capacity constraints $w_d(f_j) \leq W_j$ for all $0 \leq j \leq k$, the following three statements are true:*

(1) $\operatorname{Card}(\mathbf{f}') \geq \operatorname{Card}(\mathbf{f})$
(2) *If* $\operatorname{Card}(\mathbf{f}') = \operatorname{Card}(\mathbf{f})$, *then* $\operatorname{Sat}_V(\mathbf{f}) \geq \operatorname{Sat}_V(\mathbf{f}')$
(3) *If* $\operatorname{Card}(\mathbf{f}') = \operatorname{Card}(\mathbf{f})$ *and* $\operatorname{Sat}_V(\mathbf{f}) = \operatorname{Sat}_V(\mathbf{f}')$, *then* $\operatorname{Dist}_\Gamma(\mathbf{f}) \leq \operatorname{Dist}_\Gamma(\mathbf{f}')$

4.3 Dependencies

De Capitani di Vimercati et al. [13] have explored the technique of fragmentation to ensure data confidentiality in presence of dependencies among columns. Such dependencies can enable a server to deduce much more information even though it only stores fragments of a confidentiality-preserving fragmentation. Dependencies on the patient table could for example be DoB, ZIP $\rightsquigarrow$ Name (that discloses the name of a patient from the date of birth and zip code) or Treatment $\rightsquigarrow$ Diagnosis (that discloses a diagnosis from the treatment). De Capitani di Vimercati et al. explore this problem only in a single-relational environment. For our application, the definitions and theories will be translated into a multi-relational context.

Definition 4.10. **(Data Dependency)** For a given database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$ a *dependency* δ over D is an expression of the form $X \rightsquigarrow Y$, with $X, Y \subset \bigcup_{s=1}^{N} A_s$ and $X \cap Y = \emptyset$. The left hand side of a dependency δ is called the premise while the right hand side is called the consequence of δ. For simplicity, the notations $\delta.premise$ and $\delta.consequence$ (or $\delta.p$ and $\delta.c$, for short) are used to denote the respective part of the dependency.

A simple approach to make the information disclosed by dependencies visible in a fragment is adding the implied attributes to the fragment (this approach is called fragment and dependency composition in [13]):

Definition 4.11. **(Dependency Composition)** For database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$, a subset $f_j \subseteq \bigcup_{s=1}^{n} A_s$ of attributes and a set Δ of dependencies, the *composition* of f_j with dependency $\delta \in \Delta$ is the set of attributes:

$$f_j \otimes \delta = \begin{cases} f_j \cup \delta.consequence, & \text{if } \delta.premise \subseteq f_j \\ f_j, & \text{else} \end{cases}$$

Next, in [13] the closure of a set of attributes is a superset that is immune to dependency composition:

Definition 4.12. **(Closure)** Let $A_1, \ldots, A_N$ denote pairwise disjoint sets of attributes and let $d = \{r_1, \ldots r_N\}$ be a database state over the schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$. Moreover, let Δ denote a set of dependencies. For any subset $f \subseteq \bigcup_{s=1}^{n} A_s$ of attributes the *closure* with respect to δ is defined as the minimal set $\overline{f}$ which satisfies $f \subseteq \overline{f} \subseteq \bigcup_{s=1}^{N} A_s$ and for all $\delta \in \Delta$ it holds that $\overline{f} \otimes d = \overline{f}$. If the subset f satisfies $f = \overline{f}$ it is called *closed*.
If $\mathbf{f} = (f_0, \ldots, f_k)$ denotes a lossless/correct fragmentation of d the closure of that fragmentation with respect to Δ is defined as $\overline{\mathbf{f}} \coloneqq \left(\overline{f}_0, \ldots, \overline{f}_k\right)$. A fragmentation for which every server fragment $f_j \in \{f_1, \ldots f_k\}$ is closed is called a closed fragmentation.

It is generally not possible to find a closed correct fragmentation satifying the disjointness property. Hence, the following problem statement focuses on finding a closed lossless multi-relational fragmentation.

Definition 4.13. **(Multi-relational Separation of Duties in Presence of Data Dependencies)** Given a database state $d = \{r_1, \ldots, r_N\}$ over the given database schema $D = \{R_1(A_1), \ldots, R_N(A_N)\}$, tuple identifiers $\text{tid}_s \subset A_s$ for all $s \in \{1, \ldots, N\}$, a well-defined set of multi-relational confidentiality constraints C, visibility constraints V, a set of closeness constraints Γ, a set of dependencies Δ, a weight function w_d, servers $S_0, \ldots, S_k$ with maximum capacities $W_0, \ldots, W_k \in \mathbb{R}_0^+$ and positive weights α_1, α_2 and $\alpha_3 \in \mathbb{R}_0^+$, find a **closed** lossless confidentiality-preserving fragmentation $\mathbf{f} = (f_0, \ldots, f_k)$ of d which satisfies $w_d(f_j) \leq W_j$ for all $0 \leq j \leq k$ such that the weighted sum $\alpha_1 \operatorname{Card}(\mathbf{f}) - \alpha_2 \operatorname{Sat}_V(\mathbf{f}) + \alpha_3 \operatorname{Dist}_\Gamma(\mathbf{f})$ is minimized.

One might wonder, whether the condition that the fragmentation is closed will prevent finding a solution when a non-closed confidentiality-preserving fragmentation exists that would solve the problem. As opposed to [13], in our problem statement this could be the case due to the capacity constraints. As every fragment f_j is a subset of its closure $\overline{f}_j$ it also holds that $w_d(f_j) \leq w_d(\overline{f}_j)$. Hence,

it is up to the user to make sure that the servers' capacities are large enough to find such a solution. On the other hand however, aside from the capacity constraints, the restriction that the fragmentation is closed is not an issue: if $\mathbf{f} = (f_0, \ldots, f_k)$ is a confidentiality-preserving fragmentation, then its closure $\overline{\mathbf{f}} = \left(\overline{f}_0, \ldots, \overline{f}_k\right)$ is also confidentiality-preserving. Therefore, it is always possible to find a closed confidentiality-preserving fragmentation when there exists a non-closed fragmentation that preserves confidentiality, however this closed fragmentation will potentially violate the capacity constraints of the fragments even if the non-closed fragmentation satisfied those.

For the Extended Multi-relational Separation of Duties Problem it was recommended to choose the capacity of the owner fragment such that it can only hold the attributes contained in singleton constraints and the respective tuple-identifier attributes (see Section 4.1). When dependencies are taken into account however, using this capacity could make the problem unsolvable because additional attributes should be stored in the owner fragment because they are sensitive on their own due to dependency. To illustrate this, a dependency $\delta \in \Delta$ with $|\delta.premise| = 1$ and $\delta.premise \neq c$ for all $c \in C$ is supposed. The premise contains a single attribute which is not contained in a singleton constraint. At first glance, it seems that this attribute is not sensitive on its own and therefore, the attribute in $\delta.premise$ will not be placed in the owner fragment when choosing the owner capacity W_0 as described in Section 4.1. Because the problem statement requires a closed fragmentation, the server fragment that contains the attribute in $\delta.premise$ needs to hold $\delta.consequence$, too. Obviously, a problem arises if there exists a confidentiality constraint $c \in C$ with $c \subseteq \delta.premise \cup \delta.consequence$ because the fragmentation obeying W_0 cannot be confidentiality-preserving and therefore, no solution exists. Such situations occur if the closure of an attribute is sensitive. Therefore, the actual set of sensitive attributes is given by the union of the two sets $A^* := \{c \in C \mid |c| = 1\}$ and $A^{**} := \bigcup_{s=1}^{N} \left\{a_i^s \in A_s \mid \exists c \in C \text{ with } c \subseteq \overline{\{a_i^s\}}\right\}$.

There are two possible solutions for this. The first one is to introduce new confidentiality constraints $c = \{a_i^s\}$ for all $a_i^s \in A^{**}$ which is justified because those attributes can be regarded as sensitive attributes. The other solution is to increase the capacity of the owner fragment such that it holds all the attributes in A^* and A^{**} and the necessary tuple-identifier attributes to ensure the reconstruction property of the fragmentation. This can be accomplished by defining the capacity of the owner fragment as:

$$W_0 = \sum_{s:A_s \cap (A^* \cup A^{**}) \neq \emptyset} w_d(\mathrm{tid}_s) + \sum_{a \in (A^* \cup A^{**})} w_r(a),$$

5 ILP FORMULATION

We now translate the Multi-relational Separation of Duties Problem in Presence of Data Dependencies into an ILP representation. All indicator variables $z_v, u_{vj}, q_{\gamma j}, p_{\delta j}, y_j$ and x_{ij}^s are binary. Moreover, we establish the convention that $s \in \{1, \ldots, N\}$, $a_i^s \in A_s$, $a_{i'}^s \in \mathrm{tid}_s$, $j \in \{0, \ldots, k\}$, $c \in C$, $v \in V$, $\gamma \in \Gamma$, $\delta \in \Delta$. The ILP then looks as follows:

$$\min \quad \alpha_1 \sum_{j=0}^{k} y_j - \alpha_2 \sum_{v \in V} z_v + \alpha_3 \sum_{\gamma \in \Gamma} \sum_{j=0}^{k} q_{\gamma j} \tag{1}$$

$$\text{s.t.} \quad \sum_{j=0}^{k} x_{ij}^s \geq 1, \tag{2}$$

$$\sum_{a_i^s \in A_s^*} x_{ij}^s \leq x_{i'j}^s |A_s^*|, \tag{3}$$

$$\sum_{a_i^s \in A_s^*} x_{ij}^s \geq x_{i'j}^s, \tag{4}$$

$$\sum_{s=1}^{N} \sum_{a_i^s \in A_s} w_d(a_i^s) x_{ij}^s \leq W_j y_j, \tag{5}$$

$$\sum_{a_i^s \in c} x_{ij}^s \leq |c| - 1, \tag{6}$$

$$\sum_{a_i^s \in v} x_{ij}^s \geq u_{vj} |v|, \tag{7}$$

$$\sum_{j=0}^{k} u_{vj} \geq z_v, \tag{8}$$

$$\sum_{a_i^s \in \gamma} x_{ij}^s \leq |\gamma| q_{\gamma j}, \tag{9}$$

$$\sum_{a_i^s \in \delta.p} x_{ij}^s \leq |\delta.p| - 1 + p_{\delta j}, \tag{10}$$

$$\sum_{a_i^s \in \delta.p} x_{ij}^s \geq p_{\delta j} |\delta.p|, \tag{11}$$

$$\sum_{a_i^s \in \delta.c} x_{ij}^s \geq p_{\delta j} |\delta.c|, \tag{12}$$

5.1 Translating confidentiality

Indicator variables $x_{ij}^s \in \{0, 1\}$ are used to express that attribute $a_i^s \in A_s$ from relation r_s is placed on server j. For every $s \in \{1, \ldots, k\}$ let $A_s^* := A_s \setminus \mathrm{tid}$ denote the set of non-tuple identifiers of A_s. Binary variables $y_1, \ldots, y_k \in \{0, 1\}$ are introduced which take a value of one if fragment f_j should be non-empty and a value of zero otherwise. In the objective function (1) the expression $\alpha_1 \sum_{j=0}^{k} y_j$ minimizes the cardinality of the fragmentation. Condition (2) ensures that every attribute is placed in at least one fragment satisfying the completeness property. Constraint (3) conditions that if there is a non-tuple identifier attribute contained in fragment f_j, i.e. the left hand side of the inequality is greater than one, then the right hand side must equal $|A_s^*|$ which is fulfilled if the variable $x_{i'j}^s$ for the tuple identifier attribute $a_{i'}^s \in \mathrm{tid}_s$ also equals one and is hence also part of the fragment. The definition of the reconstruction property of multi-relational fragmentation requires that the tuple identifiers tid_s are proper subsets of the fragments. Side constraint (4) takes care of this by allowing the variables $x_{i'j}^s$ for attribute $a_{i'}^s \in \mathrm{tid}_s$ to equal one if at least one variable x_{ij}^s belonging to a non-tuple identifier attribute $a_i^s \in A_s^*$ equals one. Constraint (5) makes sure that the storage capacities are not exceeded and that y_j must take a value of one, whenever any

attribute a_i^s is included in fragment f_j. Lastly Condition 6 is used to guarantee a confidentiality-preserving fragmentation, because at most $c-1$ variables x_{ij}^s for $a_i^s \in c$ can be equal to one.

5.2 Translating visibility and closeness

Additional binary variables u_{vj} are introduced for every *visibility* constraint $v \in V$ and fragment $j \in \{1,\dots,k\}$. These variables should only take a value of one if all attributes contained in v are placed in fragment f_j. Moreover, indicator variables z_v for all visibility constraints $v \in V$ are used to indicate whether there is at least one fragment that contains all attributes of visibility constraint $v \in V$.

Furthermore, for every *closeness* constraint $\gamma \in \Gamma$ and every fragment $f_j \in \{0,\dots,k\}$ binary variables $q_{\gamma j}$ are introduced to indicate whether an attribute contained in γ is placed in γ. These variables are used to determine the distribution of the closeness constraints.

In the objective function (1) the summand $-\alpha_2 \sum_{v\in V} z_v$ maximizes the number of satisfied visibility constraints and the summand $\alpha_3 \sum_{\gamma\in\Gamma} \sum_{j=0}^{k} q_{\gamma j}$ minimizes the distribution of the closeness constraints.

The constraints that ensure the proper treatment of the visibility constraints are given by Conditions (7) and (8). The former ensures that for every fragment f_j and every visibility constraint $v \in V$ variable u_{vj} can only be equal to one if the visibility constraint is satisfied in fragment f_j. The latter then makes sure that z_v, the indicator variable for visibility constraint $v \in V$, can only be equal to one if there is at least one u_{vj} for $j \in \{1,\dots,k\}$ that equals one, i.e. visibility constraint v is satisfied on at least on server. Constraint (9) ensures the proper behavior of the indicator variables $q_{\gamma j}$ which indicate whether an attribute from closeness constraint $\gamma \in \Gamma$ is contained in fragment f_j. $q_{\gamma j}$ can only be equal to zero if there is no attribute $a_i^s \in \Gamma$ and no fragment f_j for which the variable x_{ij}^s equals one.

5.3 Translating dependencies

It can easily be seen that a server fragment f_j equals its closure $\overline{f}_j$ if and only if for every data dependency $\delta \in \Delta$ one of the conditions $\delta.p \not\subseteq f_j$ or $\delta.p \cup \delta.c \subseteq f_j$ is true [13]. Hence, to check whether a server fragment f_j is closed, one first has to check for every data dependency $\delta \in \Delta$ if $\delta.p \in f_i$. In the ILP formulation indicator variables $p_{\delta j} \in \{0,1\}$ we introduce for each data dependency $\delta \in \Delta$ and server $j \in \{1,\dots,k\}$ that should take a value of one if and only if all attributes in $\delta.p$ are stored in server fragment f_j. After that, we have to make sure that if a dependency premise is contained in a fragment the consequence must also be contained. Together, Constraints (10) and (11) ensure that $p_{\delta j}$ equals one if and only if all attributes in $\delta.p$ should be placed into the same server fragment f_j.

Finally, Constraint (12) requires that all attributes in $\delta.c$ are part of server fragment f_j if all attributes in $\delta.p$ are part of f_j: if $p_{\delta j}$ is equal to one the right hand side of the inequality takes the value $|\delta.c|$. In this case the constraint can only be fulfilled if the sum on the right side is also $|\delta.c|$ which means that x_{ij}^s equals one for all $a_i^s \in \delta$. On the other hand if $p_{\delta j}$ equals zero the condition is always fulfilled.

A solution to the Separation of Duties Problem in Presence of Data Dependencies can be derived from a solution to this ILP by constructing the fragments as follows:

$$f_j \coloneqq \begin{cases} \bigcup_{s=1}^{N} \{a_i^s \in A_s \mid x_{ij}^s = 1\}, & \text{if } y_j = 1 \\ \emptyset, & \text{else} \end{cases}$$

6 IMPLEMENTATION

There are the following entities involved in the system:

- **Untrusted Database Servers:** These servers store the server fragments and can process queries involving their respective fragment only. The individual physical fragments are organized in database tables.
- **Trusted Database Server:** This server stores the owner fragment and manages connections to the untrusted servers. Most common DBMSs provide means to include database tables stored at remote servers. In PostgreSQL for example, this can be realized with so-called *Foreign Data Wrappers* and MySQL provides the *FEDERATED Storage Engine*. This enables the later presented distributed database client to issue adequate high-level (possibly SQL) queries directly to the trusted database server instead of issuing subqueries to each individual database server and then calculating the desired result. Instead, the built-in query processor of the trusted database server will decide how the query is actually optimized and executed.
- **Distributed Database Client:** The client acts as an additional layer between the database users and the database servers. It computes the fragmentation using an ILP solver, stores the metadata of the fragmentation (i.e. at which server the columns are stored) and processes and rewrites user queries such that they are based on fragments instead of relations of the original database. The distributed database client can either access the database servers directly if a query exclusively involves columns of a single fragment or it can issue queries to the trusted database server which makes all other fragments stored at the untrusted databases servers visible. Those queries are then analyzed by the database system's query processor which decides how the query is internally executed by applying adequate optimization techniques. Finally, the results of the queries are transferred to the user.

The major advantage of the presented framework is that the only component that we had to implement is the distributed database client (available at [3]) that relies on the advanced query optimization techniques already provided by today's DBMS. The chosen programming language is JAVA and the implementation relies on the popular open source DBMS PostgreSQL. Therefore, the trusted database server and the untrusted database servers need to run a standard installation of PostgreSQL. To solve the ILPs the IBM ILOG CPLEX solver is used. Lastly, to analyze and rewrite the users' queries, the distributed database client uses the open source project JSQLParser which is a SQL parser for JAVA. After solving the ILP using the CPLEX solver, the distributed database client continues with creating databases on the necessary database servers to store

the respective physical fragments. In particular, a new database is set up at the trusted database server to store the owner fragment. Subsequently, the table fragments are set up and populated with the data from the original database. Using the foreign data-wrapper module postgres_fdw, the tables stored at the untrusted servers are made visible in the newly created database at the trusted server. Therefore, the database at the trusted database server contains each of the table fragments either as a local table if the fragment is part of the owner's physical fragment or as foreign table, otherwise. As a final step, the distributed database client sets up views in the database of the trusted database server using the local and foreign tables which correspond to the tables of the original database.

7 EVALUATION

The prototype implementation is tested with the popular TPC-H benchmark for databases [15]. The database defined by the TPC-H benchmark consists of 8 tables and an overall number of 61 columns. The benchmark also includes 22 complex SQL queries which are interesting to test the implementation's capability in terms of query processing.

All of the tests were executed on a single PC equipped with an Intel Xeon E3-1231v3 @ 3.40GHz (4 Cores), 32GB DDR3 RAM and a Seagate ST2000DM001 2TB HDD with 7200 rpm. The PC is running Ubuntu 16.04 LTS. The database servers – including the trusted database server hosting the owner fragment – are running in separate, identical virtual machines which are assigned 4 cores and 8GB of RAM. The virtual machines are running Ubuntu Server 16.04 LTS with an instance of PostgreSQL 9.6.1 installed. By running the servers in identical virtual machines, it is guaranteed that the results are not influenced by hardware or software differences. Lastly, the CPLEX version used by the implementation is CPLEX 12.7.

To set up the test database, the HammerDB [14] tool was used with a scale factor of 1. Moreover, the query generator provided by DBT-3 [10] was used to obtain 22 TPC-H-like queries conforming with PostgreSQL's standard.

7.1 Settings

The number of tables in the TPC-H database is reasonably small, so that the following artificial scenario is used as the foundation for the tests:

- **Confidentiality Constraints**: The following rules are established for defining the constraints:
  - The name and the account balance of the customers and suppliers are sensitive:
    $c_1 = \{customer.c\_acctbal\}$,
    $c_2 = \{supplier.s\_acctbal\}$
  - The discount given on any order is sensitive:
    $c_3 = \{lineitem.l\_discount\}$
  - A customer's name and its address cannot be placed in the same server fragment:
    $c_4 = \{customer.c\_name, customer.c\_address\}$
  - A customer's name can not be associated with a specific order:
    $c_5 = \{customer.c\_name, orders.o\_custkey\}$
  - A supplier's name can not be associated with a line item:
    $c_6 = \{supplier.s\_name, lineitem.l\_suppkey\}$
  - The date of an order can not be associated with the total price:
    $c_7 = \{orders.o\_odate, orders.o\_totalprice\}$
  - A supplier's name can not be associated with the supplier's cost for a specific part:
    $c_8 = \{supplier.s\_name, partsupp.ps\_suppkey, partsupp.ps\_supplycost\}$
- **Dependencies**: Moreover, the following dependencies are introduced, concerning personal information about the customers and suppliers:

$$\delta_1 = \{customer.c\_address\} \rightsquigarrow \{customer.c\_name\}$$
$$\delta_2 = \{customer.c\_phone\} \rightsquigarrow \{customer.c\_name\}$$
$$\delta_3 = \{supplier.s\_address\} \rightsquigarrow \{supplier.s\_name\}$$
$$\delta_4 = \{supplier.s\_phone\} \rightsquigarrow \{supplier.s\_name\}$$

- **Visibility Constraints**: As the main purpose of visibility constraints is to speed up the execution of specific queries, a visibility constraint is introduced for each of the 22 queries consisting of all attributes in the query. Therefore, if a visibility constraint is satisfied, the execution of the corresponding query potentially involves a single database server only.
- **Closeness Constraints**: A closeness constraint is introduced for every table – containing all the of the corresponding attributes – to limit the number of overall table fragments that have to be created.

The weights α_1, α_2 and α_3 that are also needed for the problem statement are chosen to satisfy the system of inequalities presented in Lemma 4.9. Finding an optimal solution to this specific instance of the Multi-relational Separation of Duties Problem and setting up the vertically fragmented database takes around two and a half minutes and the resulting fragmentation satisfies 5 of the 22 visibility constraints. Overall, the tables are distributed among 12 table fragments on a total of 3 database servers. One of those is the trusted database server and the remaining two are untrusted.

7.2 Test runs

After the database is set up, the execution time of the 22 queries can be analyzed. For that, each query is executed with the following methods:

(1) The original non-fragmented database is queried. To ensure the comparability, the original database is stored separately at the trusted database server.
(2) The queries are rewritten by our trusted database client to act on table fragments instead of the original tables.
(3) Instead of rewriting, the queries are cast to specific views set up in the trusted database server to recreate the original tables.
(4) If a query can be evaluated by a single database server (because it physically stores all the involved attributes), the query is directly cast to this server.

Figure 1 summarizes the results of the test run. This evaluation shows the major advantage of the separation of duties approach. Because the columns of the tables are outsourced in plaintext, every

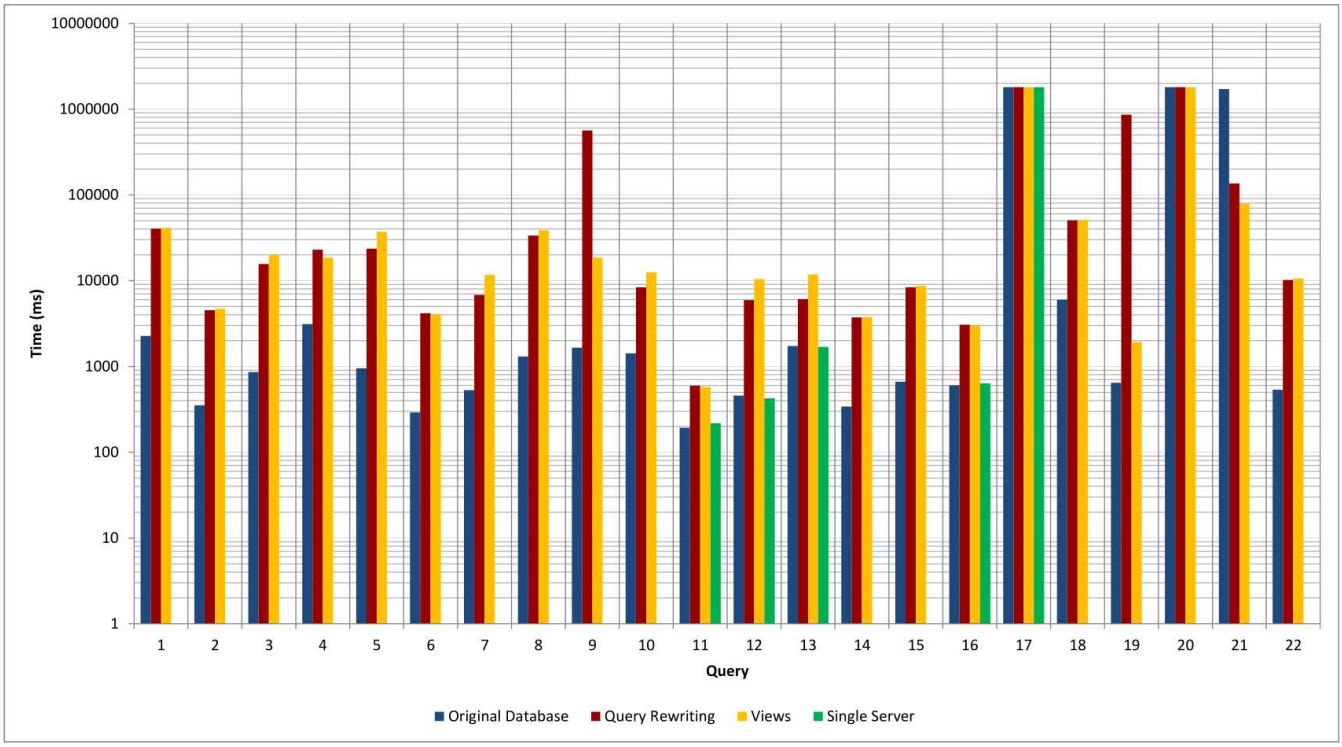

Figure 1: Runtime results for TPC-H queries

query can potentially be executed. However, two of the queries, namely Q_{17} and Q_{20}, were canceled because the timeout limit (30 minutes) was exceeded. Yet, the reason why these queries take so much time is not related to the vertically fragmented database as the timeout was also reached for the original non-fragmented database. Therefore, it can be concluded that there is some issue related to PostgreSQL which cannot find an adequate execution plan for those queries. As it was suspected, queries Q_{11}, Q_{12}, Q_{13} and Q_{16} that can be evaluated in a reasonable amount of time by a *single* server of the fragmented database can be executed in about the same time as in the non-fragmented database. For these 4 queries, a visibility constraint could be satisfied which perfectly illustrates the benefits of introducing those constraints. Interestingly, query rewriting and using views performed considerably worse for three of those 4 queries (Q_{11}, Q_{12}, Q_{13}). This is especially noticeable because rewriting the query also leads to a situation where the query involves only one database server but this is obviously not detected by PostgreSQL in conjunction with the foreign data wrapper extension postgres_fdw. This observation justifies a prior analysis of the queries as implemented in our distributed database client.

There is one query, Q_{21}, for which the execution time on the fragmented database is lower than the execution time for the non-fragmented database. For this query, the fragmented database probably profited from a better execution strategy that could be established by PostgreSQL due to the query rewriting or the use of the views. However, we assume that such situations occur very rarely in practice and are caused by PostgreSQL's execution strategy.

An interesting thing to notice is that query rewriting outperformed querying the views 13 times; querying the views was better for only 7 queries. Even more interesting, rewriting the queries performed better in 9 out of 12 times (ignoring the canceled queries) when the number of involved table fragments was lower than for the views. This illustrates the advantage of query rewriting over using views because unnecessary table fragments can be omitted with the former method. The overhead introduced by rewriting the queries is very small for all of the queries compared to the execution time and can therefore be neglected. Consequently, one can conclude that query rewriting is generally the better strategy than using views. However, if for some reason a rewritten takes very long to process, querying the views can potentially lower the execution time. An example for such a situation is query Q_{19}.

8 CONCLUSION

In this article, a separation of duties approach, based on vertical fragmentation, was presented to address the problem of preserving confidentiality in cloud databases. The confidentiality constraints and data dependencies are capable of expressing a wide range of confidentiality concerns that appear in the context of cloud databases. Moreover, visibility constraints and closeness constraints were introduced to increase the usability of the resulting vertically fragmented database. Mathematical optimization is a natural way of modeling the underlying Separation of Duties Problem and one can rely on existing solution strategies to obtain usable, confidentiality-preserving vertical fragmentations. The provided implementation shows that vertically fragmented databases can be set up with a

relatively small effort such that vertical fragmentation is in fact a viable technique to preserve confidentiality in cloud databases in practical scenarios which allows the execution of queries of arbitrary complexity.

REFERENCES

[1] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. In *The Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, 2005.

[2] J. Biskup, M. Preuß, and L. Wiese. On the inference-proofness of database fragmentation satisfying confidentiality constraints. In *ISC*, volume 7001 of *Lecture Notes in Computer Science*, pages 246–261. Springer, 2011.

[3] F. Bollwein. CloudDBSOD Client. http://www.uni-goettingen.de/de/558180.html.

[4] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Selective data outsourcing for enforcing privacy. *Journal of Computer Security*, 19(3):531–566, 2011.

[5] V. Ciriani, S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. An OBDD approach to enforce confidentiality and visibility constraints in data publishing. *Journal of Computer Security*, 20(5):463–508, 2012.

[6] V. Ciriani, S. D. C. Di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *European Symposium on Research in Computer Security*, pages 171–186. Springer, 2007.

[7] V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation design for efficient query execution over sensitive distributed databases. In *ICDCS*, pages 32–39. IEEE Computer Society, 2009.

[8] V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 440–455. Springer, 2009.

[9] V. Ciriani, S. D. C. D. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Combining fragmentation and encryption to protect privacy in data storage. *ACM Transactions on Information and System Security (TISSEC)*, 13(3):22, 2010.

[10] DBT-3. http://osdldbt.sourceforge.net/.

[11] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragments and loose associations: Respecting privacy in data publishing. *Proceedings of the VLDB Endowment*, 3(1-2):1370–1381, 2010.

[12] S. D. C. di Vimercati, R. F. Erbacher, S. Foresti, S. Jajodia, G. Livraga, and P. Samarati. Encryption and fragmentation for data confidentiality in the cloud. In *Foundations of Security Analysis and Design VII*, pages 212–243. Springer, 2014.

[13] S. D. C. di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati. Fragmentation in presence of data dependencies. *IEEE Transactions on Dependable and Secure Computing*, 11(6):510–523, 2014.

[14] HammerDB. http://www.hammerdb.com/.

[15] Transaction Processing Performance Council. TPC-H Benchmark Version 2.17.1. http://www.tpc.org/tpch/.

[16] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the VLDB Endowment*, volume 6, pages 289–300. VLDB Endowment, 2013.

[17] T. Waage, D. Homann, and L. Wiese. Practical application of order-preserving encryption in wide column stores. In *SECRYPT*, pages 352–359. SciTePress, 2016.

[18] L. Wiese. Horizontal fragmentation for data outsourcing with formula-based confidentiality constraints. In *IWSEC*, volume 6434 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2010.