SchengenDB: A Data Protection Database Proposal

Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner

MIT CSAIL

Abstract. GDPR in Europe and similar regulations, such as the California CCPA, require new levels of privacy support for consumers. Most challenging to IT departments is the "right to be forgotten". Hence, an enterprise must ensure that ALL information about a specific consumer be deleted from enterprise storage, when requested. Since enterprises are internally heavily "siloed", sharing of information is usually accomplished by copying data between systems. This makes finding and deleting all copies of data on a particular consumer difficult.

GDPR also requires the notion of purposes, which is an access control model orthogonal to the one customarily in SQL. Herein, we sketch an implementation of purposes and show how it fits within a conventional access control framework.

We then propose two solutions to supporting GDPR in a DBMS. When a "green field" environment is present, we propose a solution which directly supports the process of ensuring GDPR compliance at enterprise-scale. Specifically, it is designed to store every fact about a consumer exactly once. Therefore, the right to be forgotten is readily supported by deleting that fact. On the other hand, when dealing with legacy systems in the enterprise, we propose a second solution which tracks all copies of personal information, so they can be deleted on request. Of course, this solution entails additional overhead in the DBMS.

Once data leaves the DBMS, it is in some application. We propose "sandboxing" applications in a novel way that will prevent them from leaking data to the outside world when inappropriate. Lastly, we discuss the challenges associated with auditing and logging of data. This paper sketches the design of the above GDPR compliant facilities, which we collectively term SchengenDB.

1 Introduction

The General Data Protection Regulation (GDPR) took effect on May 25, 2018, affecting all enterprises operating within the European Union (EU) and the European Economic Area (EEA) [1]. The GDPR is the leading example of a new generation of privacy laws around the world that impose a strict and more comprehensive set of requirements on all systems that "process" personal data. In particular, the GDPR now mandates that no personal data may be "processed" at all without an adequate "legal basis." This legal attitude with respect to

personal data stands in sharp contrast to other legal systems, including that in force in the United States, in which companies can do whatever they choose with personal data unless there is a specific legal prohibition against a specific type of processing. Nevertheless, today the United States, and a number of other countries, are actively debating new privacy laws, many of which would entail similar requirements as imposed by the EU GDPR.

The requirement to keep personal data "under control" at all times imposes several important new conditions on enterprises processing personal data. We do not describe all of those here but rather concentrate on the new technology necessary to enable fundamental parts of GDPR compliance. We define two broad requirements for database implementation of GDPR rules. First, enterprises must now keep track of the legal basis under which data is allowed to be processed, and assure that applications, services, and analysis driven by enterprise data bases systems adhere to those legal restrictions. Second, enterprises also must keep data "under control" such that when an individual (aka. a "data subject" in GDPR parlance) requests that their data be "erased" or "forgotten", that such request is honored throughout the enterprise's own systems.

The GDPR mandates that a "legal basis" is required in order to permit any processing of personal data. That means that whenever a company collects, stores, analyzes, shares, publishes, or takes any other action on personal data it must point to a specific legal authority defined by the GDPR as the "legal basis" for such processing. Personal data must be kept under control by database systems so that enterprises can verify that when data is processed, that processing is permitted based on the relevant legal basis, effectively a permission (GDPR Art. 6). Contrary to popular misunderstanding of the GDPR, however, consent of the data subject is only one of several specific legal bases for processing. For clarity, we summarize the several legal bases for processing available under the GDPR. Data may be processed based on one of the following six legal conditions:

- **Consent**: An individual agrees to have their personal data processed for some specific purpose.
- **Contract**: The individual and the enterprise have entered into a contract providing the enterprise with the right to process personal data.
- **Legal obligation**: The enterprise can process data to comply with a legal obligation binding on that enterprise.
- **Vital interests**: The enterprise can process personal data to protect vital interests of the user or another person.
- **Public interest**: The enterprise can process, including disclose, personal data when it is in the public interest, generally as directed by a government authority.
- **Legitimate interest**: The enterprise can process data for purposes that are necessary to the legitimate interest of the enterprise, provided such interest is not overridden by the fundamental rights of the individual.

Every step taken to process personal data must conform to one of these legal conditions. So while there are circumstances in which personal data can

be processed without consent, it must always be processed under the control of some legal authority. Consider two motivating scenarios to understand how the GDPR operates and what requirements are placed on database systems:

Scenario 1: A company collects phone numbers for user authentication purposes (two factor authentication) but data subjects agree to provide their phone number only for that purpose. The company now has a database containing phone numbers but no explicit purpose associated with them. The company's marketing department decides to use the phone numbers for product promotion purposes without the knowledge that the phone numbers were collected only for the purpose of authentication, thus violating GDPR requirements.

Scenario 2: A company is running a study and would like to obtain a list of users that opted-in while excluding those that opted-out of participation in analytics. Users of the data must have tools to respect the preferences and purposes agreed to by the data subjects. To achieve this today, the company must redesign the database schema to incorporate all possible GDPR-related data usage purposes for every user which hinders the company's ability to gain insight from their data.

The above circumstances give rise to the following three requirements of systems that store personal data inside the EU (see also [9]):

- **Controlled storage and access**: The storage system must store the legal basis on which access is allowed, including specific purpose limitations.
- **Queries**: All queries of personal data must be associated with a purpose, which defines the data allowed to be accessed.
- **System wide erasure**: A data subject has the right to request that ALL of their personal data be erased, in which case the request will be honored throughout the enterprise. We adopt a modification of this requirement which states that personal data must be deleted to the extent technically practical or "placed beyond use" if full erasure is not possible.

Besides increasing interest in GDPR from database and cloud providers [7, 8, 4, 5], as of June 2019, we are unaware of any end-to-end system solution to manage personal data within the confines of GDPR. Hence, we present a data management system, SchengenDB, that can implement these restrictions efficiently. Our solution focuses on managing compliance within an enterprise, not between enterprises. Furthermore, we do not attempt to protect against malicious employees but rather protect and limit misuse through direct support for privacy requirements. Our solution has multiple parts. In Section 2, we describe a system that supports the definition of purposes and ensures that only personal data authorized for a given purpose is released to applications with that purpose.

Then Sections 3 deals with supporting GDPR within the DBMS. Primarily, we show how to support the right to be forgotten. First, we present a solution appropriate for new applications being constructed. In this case, we can force a logical data base design that stores each fact exactly once. Deleting this fact will thereby perform the appropriate "right to be forgotten". The second solution is appropriate for existing DBMS schemas, which often employ data redundancy. In this case, we need to track all personal data as the DBMS updates multiple

copies and constructs derived information through new tables and materialized views.

Once personal data leaves the DBMS, it resides in an application. In Section 4, we outline how "sandboxing", the use of virtual containers, can be used to disallow leaks. If that is too onerous, then we propose a second solution that trusts the owner of the sandbox to "do the right thing" when data leaks. Lastly, in Section 5 we discuss implementation issues dealing with audit and logging.

In summary, we make the following contributions:

- We propose SchengenDB, a database management system that helps enterprises comply with GDPR, through two different implementations of the right to be forgotten.
- SchengenDB's novel data purpose protection ensures that personal data can be used only for specific purposes for which the user gave explicit permission.
- SchengenDB's novel sandboxing helps ensure that applications do not leak personal data.
- SchengenDB provides efficient auditing procedures which facilitate the burden of proving enterprise-wide GDPR compliance and guaranteed data deletion.

2 Purpose-Based Access Control

One of the biggest changes brought by GDPR and related regulations is that personal data cannot be used within an enterprise for an arbitrary purpose. For example, as outlined in the introduction, a phone number might be usable only for authentication and not for direct marketing (e.g., by calling the phone number) or even indirect marketing (e.g., to infer the person's location). Thus, we propose a "purpose" based access control model. Unlike the existing database security model [2], purposes are associated with personal data in a fine-grained manner (e.g., different attributes of a record in a table can have different allowed purposes), and are "carried along" when the personal data is processed. In this section, we formalize the notion of purposes and describe our solution.

2.1 Data & Access Model

In our model, database users can define arbitrary purposes describing how they intend to use personal data. For example, a member of the marketing team can associate his team with the purpose "marketing" which will restrict their access to personal data that has the associated "marketing" purpose. Who is authorized to define purposes and how they get the consent of data subjects are administrative tasks beyond the scope of this paper. However, we assume that each data subject can opt-out or opt-in for each purpose.

It is important to note that purposes restrict system users to a logical subset of personal data in the DBMS. SQL access control has a similar function,

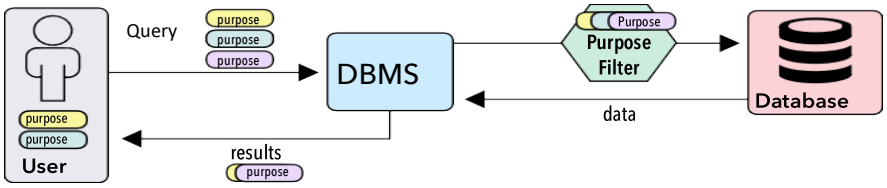

Fig. 1. Example of purpose usage of personal data in queries. The client has two associated purposes and issues a query with a third purpose. SchengenDB queries the database and returns the filtered results to the client.

but there are important differences between purposes and SQL access control. First, SQL deals with system users not with applications. Second, SQL protects relational views, so all rows of data are treated uniformly. Purposes may define so-called "ragged" tables. Lastly, each data subject must be able to opt-in or opt-out of each purpose. Hence, access is defined individual-by-individual. On the other hand, SQL protects logical subsets of the data (so called views). Hence, the definition facility is totally different. It is certainly possible to modify SQL access control to deal with these differences. However, in the following we present a direct implementation.

Tables and columns that contain personal data must be declared and are designated with a schema-level notation. A purpose is defined by a documented use and by the queries and applications that are used to implement that purpose. From this information, tables and columns that contain personal data can be deduced. This activates personal data checking for all accesses to that table and column.

Each cell in a personal data column can indicate if the user agreed to, or opted-out of, any of the purposes. GDPR requires that the default value be "opt-out". In principle, the query processor simply skips "opt-out" cells.

Purposes are designed, developed, documented, and maintained by an individual who is responsible for that purpose. Each query and application that accesses personal data must be associated with at least one purpose. To add, delete, or modify a purpose, the responsible individual works with trusted database administrators (DBAs) who authenticate, verify, and implement the purpose. Legal verification may be required.

For management purposes, and to reassure data subjects of authorized control, the use of purposes can be restricted to specific users or roles. Hence, processing a database access to personal data involves mutually applying the purposes of the database access, the user, and the user's role, as illustrated in Figure 1.

This system does not replace the current SQL access control system. Instead, it is implemented in addition to the current system. Specifically, a system user must have SQL access to a datum in addition to purpose access. In the following, we focus on purpose access, as SQL access is well understood.

2.2 Execution Model

Given a query with a purpose, SchengenDB checks if the system user is allowed to access the specified tables and columns given the indicated purpose. Assuming the system user is allowed to proceed with the query, the execution engine returns all personal data that matches the query except for the records which do not permit the purpose specified in the query. As a result, a query may give different answers based on the purpose associated with the query. In addition to the result, the database indicates how many items were omitted due to a purpose violation. This latter feature is useful for debugging and for endowing the system with operational transparency.

2.3 Implementation and Optimization

Macro-level purposes (database, table, column purposes) involve minimal storage overhead and can be safely ignored in the present discussion. Hence, we focus on row and cell level purposes.

Our first (naive) solution stores row and cell purposes as a bit vector. Given N rows and C columns, we require N bit vectors for the rows and N * C bit vectors for the cells. We focus herein only on the cell-level overhead. Given a set of purposes T that require cell level specification, and given the assumption that a fraction α of cells have a non-default purpose then the overhead is:

$$O(\alpha * N * C * T) \tag{1}$$

To illustrate the overhead, consider a table of 1 Billion rows with 100 columns and 50 purpose bits where 5% of the cells have cell-specific bits. The overhead is $(0.05 * 10^9 * 100 * 50)$ bits which is more than 30GB of storage. If 10% of the cells have cell-specific bit vectors then the overhead jumps to almost 70GB. This could represent as much as 20% of the size of the table. There are both benefits and drawbacks to the naive solution.

Pros: such a purpose storage solution would allow very efficient query processing (simply check the correct bit in the bit vector).

Cons: storage overhead is linear in the number of purposes. This may be acceptable when the number of purposes is small or when a very small percentage of cells require cell-specific purposes. In general, we need a more efficient solution.

Consider the worst case for which a set of purposes is defined for every cell in a table. This requires $O(N * C * T)$ storage. Suppose purposes are not randomly assigned to cells but follow some distribution (e.g., exponential). In this case, we can efficiently compress purposes using a variant of Huffman encoding [3]. Such an encoding can be used in conjunction with a bit vector representations to minimize storage for frequent combination of purposes found in the database while leaving infrequent combinations as-is.

A second mechanism for cell purpose encoding assumes that the number of purpose combinations follows some distribution. Hence, most cells in the table have some combination A_i of tags, for example:

$$A_1 = (marketing, analytics, support)$$

and a less frequent combination,

$$A_2 = (marketing, support)$$

and so on, for some combination A_n, which is the nth most frequent combination of purposes in the database.

Arrange these combinations in sorted order $A_1, A_2, A_3, ..., A_n$. Purpose bit vectors can then be stored with $O(logn)$ overhead by intelligently encoding them using a compression scheme. However, such encoding can introduce bottlenecks at query time because combinations must be decoded and matched against query specified purposes. Additionally, such encodings do not easily support updates to the database.

It is conceivable that the compression is efficient enough (i.e., a large enough fraction of cells in the database have the same combinations of purposes) that querying with purposes can be achieved by first scanning the purpose combinations to determine which compressed representations must be included. The remaining elements can be matched using bit vectors as in the naive solution. It is likewise reasonable to assume that changes to the database will follow a similar purpose distribution. However, in the case that a certain percentage of the database has purposes that are no longer "optimally" encoded, a re-encoding procedure may be necessary. Moreover, adding a new purpose can be done using the compressed representation and does not require re-encoding.

3 DBMS Support for GDPR

In the previous section, we explained how to ensure that personal data will be returned only for personal data with an opt-in value for the purpose associated with a query. In this section, we turn to the "right to be forgotten". To address this issue in SchengenDB, we need a reliable way to identify all personal data for a single user and to delete it efficiently. In this section, we propose two solutions, one for a "Green Field" application and another which deals with an existing schema. First we explore support within a single database and then we show how to support this requirement across systems.

3.1 Green Field Within a Single System

The key idea is to disallow duplicate or derived data to be stored. To do so, we propose to enforce an entity-relationship (E-R) model on the data. This data model requires data to be stored as:

Entities: these are features that have independent existence. Hence, they have a unique identifier and can only be inserted and deleted. Entities can have attributes that describe the entity. For example, an entity might be an employee with e-id as its identifier and attributes birthday, home address, etc.

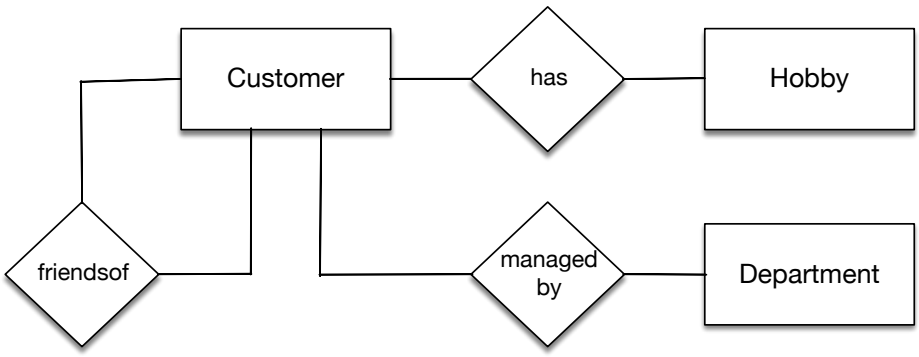

Fig. 2. Customer contains data about each customer, for example, address, birth_date and date_of_first_service. Department is an entity showing departments in the enterprise. The Hobby entity has hobby-specific data. The enterprise collects information that links customers to other entities. The three relationships indicate what department manages the customer, what hobbies they has, and who their friends are.

Relationships: entities can participate in relationships. For example, the entities employee and department may have a relationship, works-in, that indicates that an employee works in a specified department. Entities are often represented graphically as boxes with relationships indicated as arcs between the boxes. Relationships are usually further specified as 1-N or M-N to indicate allowed participation, but that feature is not needed in our discussion. Hence, we require a DBA to construct an E-R model for the data using standard E-R modelling techniques, which are discussed in any undergraduate text on database concepts.

Standard E-R practice requires that each entity have a primary key, which uniquely identifies the entity. In addition, we require every entity to have an additional "surrogate key". For example, although Customer name may be a unique identifier, we require that the Customer entity also have a surrogate key, which we require to be a random set of bits. Hence, the relational schema for the data of Figure 2 is shown in Listing 1.

```
1 Customer (cname, c-surrogate-key, other-fields)
2 Department (dname, d-surrogate key, other-fields)
3 Hobby (hname, h-surrogate-key, other-fields)
4
5 friendsof (c1-surrogate-key, c2-surrogate-key, other-fields)
6 managedby (c-surrogate-key, d-surrogate-key, other-fields)
7 has (h-surrogate-key, c-surrogate-key, other-fields)
```

Listing 1. The Relational Schema for Figure 2

There is a table for each entity type and one for each relationship that contains the surrogate keys for the pairs of records in that relationship. Although

1-N relationships can be optimized as additional fields in one of the entity tables, we do not pursue this improvement herein.

Hence, the information base for an enterprise is an E-R diagram, which is a graph of entities interconnected by relationship edges, together with a relational implementation of this structure, with surrogate keys defining the relationships.

There are a few constraints we impose on accessing and updating this structure. Since all the relationship data uses surrogate keys, SchengenDB can lazily delete "dead" relationship data as circumstances permit, through a background process the finds "dead" surrogates. To support lazy deletion, the following restrictions must be put in place:

Materialized views must be prohibited. Otherwise, there are data copies elsewhere in the database, which would have to be discovered and an appropriate additional delete performed. To avoid this error-prone and costly operation, we disallow materialized views.

Second, surrogate keys must be hidden from users. Hence, every query to the database must be expressed in E-R form and must begin by referencing an entity. Therefore, queries which directly access a relationship such as:

```
SELECT . . . FROM . . . WHERE surrogate_key = value
```

must be disallowed. This restriction is required to ensure that surrogate keys are not seen by a user. Were that true, then users could query (and store in user code) surrogate keys. In this case, lazy deletion of surrogate keys would leak information.

With these restrictions, the implementation of deletes is straightforward. To delete an entity, the appropriate record in the appropriate entity table is found and removed. This makes all surrogate keys "dangling" and unusable for generating query results. Over time, a background process can find and delete "dead" relationship data. Of course, one could also implement an "eager delete" system which would not need surrogate keys, but would require all references to an entity to be found and removed, which would increase the response time for deletes.

3.2 Existing Schema Within a Single System

While the "Green Field" solution has compelling advantages, in most cases it requires a complete redesign of the schema and the application, which can be a huge cost factor. As such, it is appropriate for new applications, but we need another solution for existing schemas.

To handle this case, we propose fine-grained tracking of changes. Every insert into SchengenDB has to be done on behalf of a specific data subject, i.e., owner of the personal data (as before). Thus, every inserted record belongs to one (or conceivably more) person. Furthermore, every derived record (think materialized view) automatically inherits the owners of the records from which it was derived. If the enterprise is concerned about the aggregation of information, then many owners will have to be recorded. This information can be stored using standard

lineage techniques [6, 3, 11]. Although this may result in onerous overhead, there is no other way to track all the personal data as it is spread around the database.

This allows SchengenDB to track all records related to a specific data subject and delete them when asked. We now turn to the copying of personal data information between systems.

3.3 Across Systems in the Enterprise

In a large enterprise there may be hundreds to thousands of separate databases. When a system user needs information from multiple databases, a prevalent practice is to copy needed information from the primary copy to a secondary one. Otherwise, a federated query must be performed, which is much slower than the same query to a single database.

To achieve this functionally in a Green Field schema, an application will request some entities from one database and then copy them into a second database. To support this operation in a GDPR-compliant way, we require that entity identifiers be global to the enterprise. For example, there must be single global notions of Customer, Hobby, and Department. If there is not a single notion of Customer, then GDPR will be impossible to implement because it is impossible to tell if, for example, Mike Stonebraker, M.R. Stonebraker, and Michael Stonebraker are 1, 2, or 3 entities. Hence, the enterprise must engage in a data integration project for GDPR compliant entities to ensure global uniqueness of these entities.

All the purposes attached to an entity record must be carried over from the first system to the second system. This requires purposes to be unique across the enterprise. In addition, we need to record in a global entity catalog that an entity has been copied into system 2 from system 1. We call this catalog the Data Management Server (DMS), which will also be used in the next section. Notice that DMS records information that happens outside the DBMS. In this case, when a GDPR compliant entity is deleted from either system, a trigger must be run to delete all copies of the entity.

In an existing schema, there may be no E-R schema associated with the data. In that case, a data copy to a second system must preserve the owners of records from the first system. Obviously, owners (i.e., data subjects) must be global to the enterprise. Hence, the DMS must record and manage all data subjects. With this caveat, the same trigger processing will work for existing schemas.

4 Application Support for Purposes

In decision support environments it is common practice (and often essential) to copy data from the database in order to analyze it using separate tools such as Python, R, or Tableau. Moreover, such analysis often involves a pipeline of operations. In this case, personal data is outside the confines of the DBMS. Some might argue that application users are trustworthy, and therefore we do not need to worry about applications leaking. However, it seems clear that stringing

together application systems can yield inadvertent leakage. This section describes a mechanism of protecting such pipelines from inadvertently leaking.

We propose a "sandboxing" approach, which monitors copies of personal data at a higher level which will prevent misuse. A sandbox is a virtual machine which allows unrestricted access and movement of data inside the sandbox, but controls interaction with other VMs. We propose a sandbox for every purpose. That sandbox contains all the applications that use that purpose in a query. If there are N purposes, then there are N VMs. The DBMS allows access only from this collection of VMs, so DBMS requests can come only from one of these sandboxes. Hence, pipelines of programs with the same purpose can freely exchange data. Otherwise, sandboxes cannot be allowed to communicate with each other, since if data is moved from a sandbox with a less restrictive purpose to one that is more restrictive, then a leak has occurred.

These restrictions can be enforced easily at the networking level without any changes to the applications. For example, in modern virtualized environments, it is possible to configure the environment so that certain VMs get special IP-ranges and that only those ip-ranges are allowed to access the database system, or one can restrict the privilege of opening a connection to the outside to a certain set of VMs. Furthermore, thanks to dockers and similar light-weight virtualization mechanisms, even hosting a large number of virtual machines no longer pose a technical challenge.

However, if an application has queries with multiple purposes, then it is placed in multiple sandboxes, wherein each sandbox can read a portion of the overall data. It is then likely that these VMs will have to communicate to get the overall task accomplished. To support such applications, we now propose a "loosey goosey" version. In this world, we point out potential violations instead of completely forbidding interaction between sandboxes. Since every communication between sandboxes is a potential violation, when a communication occurs we alert the owner of the sandbox, who is the owner of the purpose associated with the sandbox. Their VM is assumed to be non-compliant. It is up to them to figure out how to bring the VM into compliance. This will likely mean deleting offending personal data. To ensure compliance, we use a timeout mechanism for the communication operation. At the end of the timeout the VM owner has either brought the VM into compliance or we terminate the VM. Of course, this requires us to trust the owner of the purpose to "do the right thing". The strategy of reply on *ex ante* compliance checking, as opposed to *a priori* compliance guarantees is recognzied as a necessary strategy involving privacy rules for complex information systems, as it is often simply impossible to detect all violations with certainty in advance of processing [10]. As noted in Section 5, we can rely on the auditing system to discover violations after the fact, and to hold employees accountable.

We turn now to the last matter dealing with applications. When a delete of personal data is requested by a data subject, the DBMS will perform the actions specified in the previous section. However, personal data may be present at the application level. In this case, we assume that the DMS logs, at the

application level, every query to the DBMS for every sandbox. Every permitted communication between sandboxes is similarly logged. When a person requests to be forgotten, we can determine which sandboxes may have the relevant personal data (or data derived from that personal data) by reading the log. We alert the owner of the sandbox to this potential violation who can then take action, as described above.

So far, the tracking and the deactivation are pessimistic and might cause false positives, i.e., unjustified warnings to sandbox owners and terminations of VMs that are, in fact, compliant. For example, a sandbox might read data from SchengenDB but then does not store the data within the sandbox or a sandbox does an aggregate query such as `SELECT COUNT(*) FROM Customer` which requires a scan of all data but does not extract any user-specific data. Both cases will cause warnings for non-existing violations after a request from a person to delete their personal data.

Fortunately, a wide variety of optimizations are possible to reduce the number of false positives. For example, privacy-preserving analysis could be used to determine that data derived from `SELECT COUNT(*) FROM Customer` do not contain GDPR violations. Furthermore, we could provide annotations to indicate that a sandbox is stateless, transient, or has a specific time-to-live. Similarly, a data warehouse dashboard might be in violation, but if the data warehouse is refreshed every day, the violation will resolve itself after a time-to-live.

We could provide additional annotations to provide sandbox owners more fine-grained control. For example, if a sandbox is used to build a machine-learning model and the model is then deployed in a service, according to the previous section the entire model might be in violation. However, if the developer considers the model to be GDPR compliant, they could annotate that the model does not contain GDPR violations and mark the data transfer between the sandboxes as safe.

5 The Audit Process

The audit process consists of two components, one within SchengenDB and one at the application level.

5.1 Audit within SchengenDB

The fundamental auditing technology in SchengenDB is the DBMS log. Log processing is well understood by the DBMS community and is implemented in all commercial DBMSs. Specifically, all operations which alter the database are logged, typically on a record-by-record basis, with the before image of the record (so the change can be backed out if the application running the transaction fails to commit) and the after image (to restore the change if there is a crash or other unforeseen event). To support GDPR compliance, we must also log all reads, together with the query invoked and its purposes. A similar statement applies to updates. Obviously, this will slow down log processing; however, in current

systems the log is highly optimized and does not consume excessive resources. Hence, an audit merely entails inspecting the log to ensure that the purposes allowed by the enterprise are enforced. If SchengenDB is operating correctly, there should be no violations. In the unlikely event of a violation, the offending user and request can be quickly discovered and dealt with accordingly.

As stated in the introduction, we assume that enterprise employees are not malicious. Hence, users with a legitimate access to data are assumed not to share it outside the SchengenDB system, for example by copying a result into an e-mail message and sending it to an unauthorized user. Dealing with such inadvertent or purposeful leaks is outside the scope of this paper. A similar comment applies to the DBA of a SchengenDB database, which has unfettered access to everything.

However, the presence of the log raises the following question. If person X asserts their right to erasure, then a sequence of updates will occur in SchengenDB. Such updates are logged and contain the before images of deleted records.

Hence, the information about X has been deleted from the database, but not from the log. In the case of the GDPR, we understand that it is still an open question whether respect for the right to erasure requires deleting personal data from DBMS logs along with the accessible instance of the database. For reasons explained below, from a technical perspective, there are reasons to exclude the log from the scope of the right to erasure. In theory, log files can be purged after a sufficient delay, thereby deleting records for X. However, we would caution against this strategy. To deal with application errors, for example a buggy app inadvertently gives a raise to Y, the database is typically "rewound" to a time before the errant app, and then the log is replayed forward. Hence, the log must be retained for a period of time. In addition, legal requirements often require the log to be retained much longer. Removing log files is therefore not recommended. Also, logs are write-once and are never updated. Hence, updating the log to remove X's log records is not recommended. This would allow an errant log updater to wreck real havoc.

The net-net is to trust DBAs (who are the only people with access to the log) to do their job and not be malicious. After all, they can easily leak tax returns and/or financial records of important individuals, which will be far more damaging than the issues we are discussing in this section.

5.2 Application Audit

In contrast to the audit of SchengenDB itself, the audit process between sandboxes is more involved and less automatic. The data rights sandboxing approach relies on the trustworthiness of its sandbox owners. For example, the system user needs to be trusted if they declares that a GDPR violation was manually resolved. Similar, they needs to be trusted to provide the correct annotations or correct implementations of delete functions. Obviously, this can lead to violations if users make (un-)intentional mistakes.

While we do not believe it is possible to entirely avoid such violations, the SchengenDB framework can provide tools to make it easier to detect potential violations and allow an internal audit to detect potential problems, before an external audit might discover any problems. For example, the DMS could simulate a worst-case scenario for which it ignores all user-provided annotations and mistrusts all manually-resolved GDPR violations. This simulation could now be used to create a list of sandboxes and tables within SchengenDB that are potentially in violation of GDPR or in which owners made mistakes. It is also reasonable to assume, that the same simulation could be used to rank the risk of violation or mistakes. An internal auditor could then manually check some of the reported sandboxes.

Furthermore, it might be possible (with limitations of course) to scan the sandboxes for potential GDPR violations based on finger prints. For example, let's assume that we associate a random 256bit key to every GDPR-related record. If the bit sequence is found for a deleted GDPR record within a sandbox it is a strong indication that the sandbox is in violation.

Finally, any communication with the outside (e.g., between an application running in a sandbox and the web) is impossible to audit. While it might be possible to log all such communication, it will be very hard to provide a full audit as these logs are not trivial to analyze, as they are usually much less structured than DBMS logs.

6 Conclusion

In this paper, we have presented SchengenDB, which provides the infrastructure to support GDPR and other possible future privacy regulations. It does so with modest overhead for purpose processing and expanded log processing. In addition, it suggests doing "clean" database design, which will benefit an organization in multiple downstream ways (easier application maintenance, easier security control, etc.). When this is not possible, then additional lineage information must be preserved. At the application level, we suggest "sandboxing" such modules that access personal data to ensure the security of this data.

References

1. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). `https://eur-lex.europa.eu/eli/reg/2016/679/oj`. Accessed: 2010-05-25.
2. R. Chandramouli and R. Sandhu. Role-based access control features in commercial database management systems. In *Proceedings of the 21st National Information Systems Security Conference (NISSC '98)*, 1998.
3. B. Glavic and G. Alonso. Perm: Processing provenance and data on the same data model through query rewriting. In *2009 IEEE 25th International Conference on Data Engineering*, pages 174–185, March 2009.

4. Google. Google cloud and the gdpr. technical report. https://cloud.google.com/security/gdpr/.
5. Oracle. 5 perspectives on gdpr. https://www.oracle.com/applications/gdpr/.
6. F. Psallidas and E. Wu. Smoke: Fine-grained lineage at interactive speed. *PVLDB*, 11(6):719–732, 2018.
7. A. Rayani. Safeguard individual privacy rights under gdpr with the microsoft intelligent cloud. https://www.microsoft.com/en-us/microsoft-365/blog/2018/05/25/safeguard-individual-privacy-rights-under-gdpr-with-the-microsoft-intelligent-cloud/.
8. A. Shah, V. Banakar, S. Shastri, M. Wasserman, and V. Chidambaram. Analyzing the impact of GDPR on storage systems. *CoRR*, abs/1903.04880, 2019.
9. S. Shastri, M. Wasserman, and V. Chidambaram. The seven sins of personal-data processing systems under GDPR. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, Renton, WA, 2019. USENIX Association.
10. D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman. Information accountability. *Communications of the ACM*, 51(6):82, 2008.
11. J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. pages 262–276, 01 2005.