Harmonizing Privacy Regarding Data Retention and Purging

Nick Scope
DePaul University
Chicago, IL, USA
nscope52884@gmail.com

Alexander Rasin
DePaul University
Chicago, IL, USA
arasin@cdm.depaul.edu

Ben Lenard
DePaul University
Chicago, IL, USA
blenard@anl.gov

Karen Heart
DePaul University
Chicago, IL, USA
kheart@cdm.depaul.edu

James Wagner
University of New Orleans
New Orleans, LA, USA
jwagner4@uno.edu

ABSTRACT

Data privacy requirements are a complex and quickly evolving part of the data management domain. Especially in Healthcare (e.g., United States Health Insurance Portability and Accountability Act and Veterans Affairs requirements), there has been a strong emphasis on data privacy and protection. Data storage is governed by multiple sources of policy requirements, including internal policies and legal requirements imposed by external governing organizations. Within a database, a single value can be subject to multiple requirements on how long it must be preserved and when it must be irrecoverably destroyed. This often results in a complex set of overlapping and potentially conflicting policies. Existing storage systems are lacking sufficient support functionality for these critical and evolving rules, making compliance an underdeveloped aspect of data management. As a result, many organizations must implement manual ad-hoc solutions to ensure compliance. As long as organizations depend on manual approaches, there is an increased risk of non-compliance and threat to customer data privacy.

In this paper, we detail and implement an automated comprehensive data management compliance framework facilitating retention and purging compliance within a database management system. This framework can be integrated into existing databases without requiring changes to existing business processes. Our proposed implementation uses SQL to set policies and automate compliance. We validate this framework on a Postgres database, and measure the factors that contribute to our reasonable performance overhead (13% in a simulated real-world workload).

CCS CONCEPTS

• **Security and privacy → Privacy protections**; **Usability in security and privacy**; *Management and querying of encrypted data*; Key management.

KEYWORDS

privacy, compliance, data governance, retention, purging

ACM Reference Format:
Nick Scope, Alexander Rasin, Ben Lenard, Karen Heart, and James Wagner. 2022. Harmonizing Privacy Regarding Data Retention and Purging. In *34th International Conference on Scientific and Statistical Database Management (SSDBM 2022), July 6–8, 2022, Copenhagen, Denmark.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3538712.3538718

1 INTRODUCTION

Data management by an organization is bound to data privacy regulations that define how the data must be stored (e.g., archived, preserved in backups, or destroyed). New legislation is often passed with the specific intent of giving consumers more control over their data and privacy. Additionally, organizations may choose to implement additional self-imposed internal policies or policies of another organization who is a business associate. Having a large amount of quickly evolving policy sources adds a large element of complexity to this field. Violating these requirements may potentially result in large fines and a permanent loss of customer data privacy. Unfortunately, relational database compliance solutions in the data governance domain has been extremely underdeveloped.

In the data privacy and governance domain, the evolution of new and evolving requirements has far outpaced the development of dedicated functionality in data management systems. Although some current industry tools can be adjusted to create some ad-hoc compliance procedures, dedicated data privacy compliance functionality has been overlooked. Therefore, organizations are forced to either repurpose other functionality or depend on manual compliance solutions. Both options carry a heavy risk of non-compliance due to the complexity of many data privacy requirements.

Compliance can be complex due to multiple overlapping requirements over the same data. For example, per the Office of the National Coordinator for Health Information Technology, each state in the United States has their own requirements for retaining and destroying healthcare data [31]. Adding to the complexity, the data of minors and adults are sometimes governed by different policies. Different rows or columns of a table belonging to a different healthcare record could be governed by different requirements. Furthermore, database administrators must consider the potential conflict between multiple requirements (e.g., retention versus destruction of the same data).

Another complication for implementing compliance is considering how each different database logical layout and capability (i.e., relational or NoSQL) has unique challenges. Furthermore, the read and write accessibility adds additional complexity. For example, any organization relying on manual solutions for their compliance must consider the high cost of enforcing compliance when records subject to different requirements are not guaranteed to always be physically accessible.

Understanding how different platforms affect an organization's ability to comply with policy requirements is critical to developing the necessary functionality to facilitate comprehensive compliance. To achieve this goal, standardized processes must be developed. Unfortunately, current database systems are missing key functionality to support compliance.

This paper outlines a system for automated data privacy regulation enforcement which can be integrated into any existing relational database. This research strives to achieve a more long term sustainable data management system. Therefore, we begin by outlining the policy requirements which must be satisfied to achieve comprehensive compliance. Without automated compliance enforcement, organizations must rely on manual or ad-hoc solution implementations.

This paper proposes a framework that formalizes data policy compliance rules regarding retention and purging. This framework allows database administrators to define policies to automate enforcement. Overall, this paper offers the following contributions:

(1) Defining the current state of privacy compliance functionality in databases.
(2) Implementing our retention and purging functionality in a relational database and demonstrating how these work together simultaneously to achieve comprehensive automated compliance.
(3) Evaluating our framework on a relational database that mirrors a typical database query workload.

2 BACKGROUND

2.1 Real-World Examples

Example 1: Healthcare Data Violations. As recently as February 2022, organizations are still failing to comply with data governance requirements and data privacy [30]. In New York, EyeMed Vision Care was fined $600,000 for failing to comply with New York's Stop Hacks and Improve Electronic Data Security Act (SHIELD Act). Among the violations, EyeMed failed to comply with the SHIELD Act's data retention requirements. A large breadth of healthcare and personal data (including medical diagnoses and conditions) was included in the findings that lead to the aforementioned fines. Records were improperly stored and risked exposing customer data.

Example 2: European Illegal Data Retention. In the European Union, the General Data Protection Regulation (GDPR) greatly expanded consumer power over personal data [8]. Any organization which offers goods or services or collects the data of EU residents must comply with GDPR requirements (regardless of whether the organization is based in the EU). One significant requirement from GDPR is the "Right to be Forgotten" [15, 37], which allows individuals to request that companies delete all of their personal data. Without an automated process, organizations must manually determine if they can execute requests within a deadline (which must be as quickly as one month [37]). Although it is one of many laws which govern data management, GDPR is considered to be one of the strongest data privacy regulations. We focus on GDPR because it typically meets or exceeds other legal requirements; an organization that meets GDPR requirements will typically be in compliance with other laws.

On January 15, 2020, a €27.8 million fine was issued to the Italian telecommunications company TIM [2] for violations of GDPR Article 5 (processing personal data) and Article 6 (right to erasure and excessive data retention). The right to erasure allows customers to request that all of their personal data is deleted; if no retention requirement exists, companies must delete all requested records. Data was retained past the amount of time allowed by law in some instances (10 years) as well as by what was set in internal company policies (5 years) in spite of customers requesting to be removed and not contacted. Furthermore, TIM admitted their internal coding lists did not accurately determine the individuals to contact for a promotional campaign. Moreover, they admitted they did not have the knowledge necessary to implement a compliant system.

Example 3: Adapting to changing regulations. It is a long-standing rule when litigation is anticipated, a party is obliged to preserve all evidence that may be relevant to the ligation [20]. This obligation is known commonly as a litigation hold. Between 2003 and 2004 in New York, the case of Zubulake v. UBS Warburg elaborated on the application of a litigation hold to electronically stored information [38]. Accordingly, once a party is informed or has the expectation of an impending lawsuit, they must begin retaining and protecting all relevant records, regardless of storage medium or accessibility.

2.2 Terminology

Business Record: Organizational rules and requirements for data management are defined in units of business records. United States federal law refers to a business record broadly as any "memorandum, writing, entry, print, representation or combination thereof, of any act, transaction, occurrence, or event [that is] kept or recorded [by any] business institution, member of a profession or calling, or any department or agency of government [...] in the regular course of business or activity" [32]. In other words, business records describe any interaction or transaction resulting in new data.

Business records can be represented using different logical layouts. A business record may consist of a single document for an organization (e.g., an email message). However, in a database, a business record may span many combinations of rows across multiple tables (e.g., a purchase order consisting of a buyer, a product, and the purchase transaction from three different tables). The process of mapping business records to underlying data can vary depending on an organization's requirements and their data storage platforms. In relational databases, business records are defined using Select-Project-Join SQL syntax.

Policy: A data policy is any formally established rule for organizations. Policies can originate from a variety of sources such as legislation or as a byproduct of a court ruling (e.g., Zubulake v. UBS Warburg in Example #3). Companies may also establish their own internal data retention policies to protect confidential data. In practice, database administrators work with domain experts and sometimes with legal counsel to define business records and retention requirements based on the written policy. Policies can use a combination of time and external events as the criteria for data retention and destruction.

Retention: Retention defines the conditions when a business record must be preserved. Some retention requirements, such as HIPAA, may require a complete historical log of any and all business record updates (e.g., current address and full history of address changes for a patient) [5]. Organizations subject to this level of retention must archive the complete business record before every update to ensure a complete audit trail history was preserved.

Purging: Purging is the permanent and irreversible destruction of data in a business record [16]. A business record purge can be accomplished by physically destroying the device which stored the data, encrypting and erasing the decryption key (although the ciphertext still exists, destroying the decryption key makes it irrecoverable), or by fully erasing data from all storage. If any part of a business record's data remains recoverable or accessible in some form, then the data purge is not considered successful. For example, if a file is deleted through a file system, but can still be recovered from the hard drive using forensic tools, this does not qualify as a purge [16]. Similarly, records deleted through a database are recoverable through forensic analysis [33]; such records can be retained indefinitely within database backups [21]. Some policies require an organization to purge business records purely when the data is no longer needed. GDPR requires organizations to purge business records at the request of a customer; organizations must be prepared to comply with purging policies as well as ad-hoc requests.

Verification: Database administrators must be able to query the policies and the status of all business records in storage. Data storage systems must support a standard mechanism for defining the policies, listing or modifying current policies, and checking for potential conflicts (e.g., policies requiring retention and destruction of the same data) or overlap between different policies. For example, if an organization is unable to destroy data when requested by a customer, their refusal must be justified, lest they face non-compliance penalties.

Enforcement: Enforcing policies includes archiving and deleting data as required as well as guaranteeing consent is verified when processing data. Enforcing a policy maintains an organization's compliance. Current database management systems do not incorporate automated robust data policy features; as a result, organizations are forced to develop manual solutions for policy compliance.

2.3 Current Industry Tools

Amazon S3 offers an object life-cycle management tool. One significant limitation is S3's object life-cycle management is limited to time criteria only. Moreover, S3 is file-based and therefore lacks sufficient granularity to map policies to individual business records. S3 is a prime example of how industry tools are currently unable to facilitate policy compliance.

Oracle's Golden Gate (GG) [9] and IBM's Change Data Capture (CDC) [13] allow changes to be replicated from one database to another. However, these software packages are not specifically designed to support data retention requirements (although they could be expanded to support it). GG operates by inspecting REDO logs, making it difficult to incorporate the concept of business records (which are critical in defining policies for verification and enforcement).

Mimeo [25] provides similar functionality for Postgres; and like CDC and GG, Mimeo would have to be revised to support retention in terms of business records since it operates on a per-table basis. IBM InfoSphere Optim Archive [14] has archiving functionality which can be used for retention. It archives data from an active database, removing data from active storage. A major limitation of IBM's solution is the archiving must be initiated manually or by a script. Overall, all of the industry tools exemplify that current tools do exist which can be used to create ad-hoc processes that will satisfy only some requirements of compliance. Unfortunately, these do not provide a process that can scale and adapt to the the evolving domain, let alone satisfy all current requirements in harmony.

3 RELATED WORK

Shrasti et al. [29] analyzed the impact that GDPR requirements have imposed on database performance. The amount of new policy requirements has resulted in databases needing a large amount of additional support functionality. These requirements also tend to scale poorly as the volume of data subject to compliance increases. For example, they noted in some instances of retention in their studied Postgres system incurred "a 30-40% overhead" while at times requiring over quadruple the storage space to log and store all necessary metadata. Therefore, implementing a compliant system which does not limit performance is a very difficult challenge.

Ataullah et al. [4] described some of the challenges associated with record retention implementation in relational databases; the authors propose an approach that uses a view-based structure to define business records for retention rules. This provided the guidance on how VIEWs can be used to define business records which can be referenced for retention compliance. We expand on these techniques to define query-based policies for purging compliance.

Scope et al. [27] expanded on Ataullah et al.'s work by developing an archiving system in relational databases via triggers. A retention policy defines the records which must be preserved with an associated trigger that archives a copy of a deleted record. The `DELETE` query proceeds whether or not the deleted record is covered by a retention policy; when the deleted record is protected by a policy, it is stored in an "archive" table. If the transaction aborts, archive tables can be asynchronously purged of data archived by this transaction. This paper also evaluated the impact various factors (e.g., policy size, the scope of a query) play in a retention framework overhead.

Kraska et al. [19] proposed a system designed to track customer personal information to facilitate purging compliance. Before a system can purge all necessary information, it first must know which data requires purging. Although this solution does provide many significant benefits to guarantee deletion of customer data in an accessible database, it does not address how to purge data stored in physically inaccessible backups.

Kamara and Lauter's [17] research has shown using cryptography can increase storage protections. Furthermore, their research

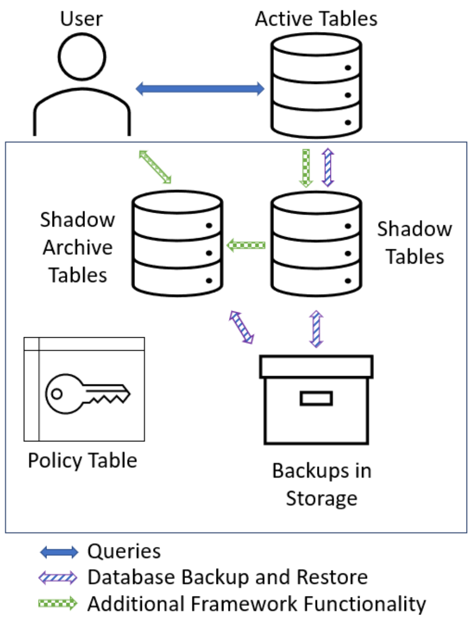

Figure 1: Process Overview

has shown that erasing an encryption key can fulfill purging requirements. This provided the original evidence of how properly implementing purging across business records would properly enforce purging policy requirements.

Scope et al. [28] presented a generalized data purging workflow which supports "remote" destruction of expired data (e.g., inaccessible records stored in a backup) in a relational database via cryptographic erasure. Encryption keys are chosen based on the purging duration and policy; records not subject to purging are stored without encryption. When the purge date arrives, the encryption key is deleted, rendering all encrypted data permanently irrecoverable (i.e., purged). In this work, we also focus on destroying data in backups. Although deleted database records can temporarily survive in storage, purging of deleted records can be expedited through rebuilding of database tables and indexes [34].

All of the aforementioned solutions attempt to address requirements regarding data privacy compliance. None of these approaches work to address purging and retention simultaneously, nor do they discuss how these requirements must work together to achieve compliance. For example, retained data (which is typically archived in a separate location) is also subject to purging policies once all retention periods expire. In this paper, we take many of the previous contributions to develop a comprehensive framework which references both retention and purging policies to achieve full compliance across both requirements.

4 THE GENERALIZED FRAMEWORK FOR POLICY IMPLEMENTATION

This framework builds upon previous approaches, merging the capabilities of solutions in [28] and [27] within the same database. We evaluate our approach in PostgreSQL, but the same solution applies to any relational database (as long as it supports basic functionality such as triggers). Figure 1 provides an overview of our DBMS policy framework. We first define the building blocks of how our approach is integrated into a database.

Throughout this paper, we develop this framework using a trigger-based approach to move data within a DBMS. In databases which do not support triggers (e.g., columnar stores or NoSQL databases), the same functionality must be executed at the application level (see Section 6.2). Once an application level implementation of trigger replacement is developed, the overall framework will maintain its ability to achieve compliance.

The scope of this paper is limited to the non-forensically recoverable data in databases. Values from businesses records may still be found in logs and underlying database pages for a certain amount of time [36]. In order to fully remove the underlying data from all database pages and hardware, additional forensic tools and APIs (similar to [35]) are required.

Additionally, file content that resides outside of the database API access (i.e., flat files) requires an OS application-level approach. Examples include audit logs and transaction logs which are not typically accessed using SQL. Because these files cannot be accessed via a database API, we also consider these out of scope.

Active tables are the regular tables in the database schema, accessed by user queries. In order to guarantee that our changes do not impact database users, active tables are not altered by implementing the framework.

Shadow tables serve as the purging mechanism for data in other tables. These tables are backed up instead of the original active tables. Every data element that needs to be eventually purged in the original table is stored encrypted in the shadow table. All other values are stored unencrypted. Encryption is used to meet the requirement of purging data, even when this data is not physically accessible (e.g., off-site backup). Any approach to achieving purging compliance must be able to purge data that is both physically accessible and inaccessible.

The encryption key is chosen based on the purging policy that applies to the value and the required purging date. Purging is performed by destroying the corresponding encryption key. By destroying the encryption key, all associated encrypted data is rendered permanently and irreversibly irrecoverable.

Shadow Archive tables serve as the retention mechanism for data in active tables. Any time data that is covered by a current retention policy is deleted or updated, a copy of this data is stored in the shadow archive (also inheriting any encryption that was already applied). Thus, a user still deletes the data from the active tables but a copy is preserved to satisfy any retention requirement. Once data is deleted from both the shadow archive and active tables, it cannot be recovered from any backup or the active live database (although forensic tools may potentially access some purged data in an active database).

All of the the components within the rectangle in Figure 1 are transparent to the user interacting with active tables. Archived data request is a custom feature described in Section 4.4. Purging, retention, and archive access functionality is automated by our framework.

4.1 Shadow Purging

Purging policies require that business records are made irreversibly irrecoverable. Data that is not physically accessible (e.g., stored on tapes in a warehouse), must still be purged when it expires. Storage backups contain a mix of data that must be retained and data that must be purged. Thus, instead of physically destroying the entire storage medium, purging compliance requires a mechanism to purge only targeted data without compromising other records.

Our framework builds upon prior research by Scope et al. [28] by leveraging encryption as a means of remotely purging data. We creates shadow tables that mirror the original tables with additional columns to enable cryptographic erasure functionality. The shadow tables replace active tables in database backup; during restore, shadow tables are decrypted into the active table (minus the purged values). Each original column is augmented by an additional column `[original column]EncryptionID`. Because a table row may be subject to different encryption keys, this allows our system to uniquely identify the encryption key used for encrypting each value.

The encryption keys are stored in the `policyOverview` table; Table 1 describes `policyOverview` schema. We can delete encryption keys when their `expirationDate` passes. Although deleting the keys using a specialized form of secure delete is beyond the scope of this paper, regularly deleting expired keys can easily be automated using a cron-like DBMS job (e.g., [6] in PostgreSQL) which removes expired encryption keys from `policyOverview`.

The table containing the keys must be stored and backed up separately from the database to avoid the problem of having the encryption keys stored with the backup; otherwise the encryption keys could not be truly purged. In our proof-of-concept experiments, the `policyOverview` table is stored in the database. However, in a production system the key management tables will be stored in a separate database. Access to these tables could be established via a database link or in a federated fashion, allowing the keys to be kept completely separate from the actual data. Since the `policyOverview` table is much smaller than the database itself, it is practical to physically purge the encryption keys to maintain purging compliance.

Our framework uses a time-based policy criteria for purging, bucketed per-day by default. A bucket represents a collection of data, grouped by a time range and policy that is purged together as a single unit. All data in the same bucket for the same policy uses the same encryption key. Our default bucket size is set to one day because, for most purge policies, daily purging satisfies the requirements (e.g., GDPR: Article 25 [8]).

policyOverview	
encryptionID	Int
policy	Varchar(50)
expirationDate	Date
encryptionKey	Varchar(50)

Table 1: policyOverview Table Column Definitions

4.2 Archive Retention

Our proposed framework copies data to shadow archive tables when it is no longer needed (e.g., deleted) but still requires retention. By having a separate storage space for archived data and by moving data from shadow tables to the shadow archive tables, our framework is able to maintain compliance without imposing a high performance overhead.

Our framework combines elements of previous work, such as [4, 27, 28], by using triggers to transparently move data into the archive (*shadow archive*) which stores both encrypted and non-encrypted data to achieve retention compliance. This enables our framework to offer the benefits of retention and purging compliance simultaneously. Shadow archive tables match the schema of the shadow tables with the additional columns of `transactionID`, `sequenceNo`, `archivePolicy`, and `retentionDate` (all of which are added to the primary key of the table to guarantee database transactional integrity). Whenever a row is deleted from the active tables, a copy of the business record's data requiring retention (if any) is first archived in the shadow archive. Whenever a transaction aborts, we asynchronously delete all of the retained records with this transactionID.

The archived values are stored until the expiration of the corresponding `retentionDate`, when they can be deleted without the risk of non-compliance. Because the archive is used to preserve business records until retention is no longer necessary, the archive does not allow updates of any archived data. If any archived records require updating, that would be accomplished by retrieving the archive records (see Section 4.4), re-inserting them into the active tables, and re-archiving.

4.3 Defining Policies

Our method of defining policies for both retention and purging uses SQL queries to define the underlying business records and the retention or purging criteria (pioneered by Ataullah et al. [4]). With retention, we require that the date column used to determine the criteria must not contain a NULL. With purging, a NULL in the date column is equivalent to a "purge immediately" requirement. Whenever the purge date is set to NULL, our framework will purge the data as soon as possible (i.e., at the expiration of the retention policy). To calculate the purge date, our framework computes the expiration of the longest applicable retention policy. If no retention policy applies to the value, we use today's date as the expiration date.

With retention policies, we propose new SQL syntax, `CREATE RETAIN`, to define the set of business records that must be protected from deletion. All policies must be defined using SELECT-PROJECT-JOIN (e.g., SQL queries cannot use aggregation).

`CREATE RETAIN` requires the `SELECT` clause to contain the primary key of every table appearing in the policy. Moreover, any columns referenced in the `WHERE` clause must also be included. These constraints are required to verify the retained copy in the archive against the relevant policy criteria. Each retention policy is handled independently, which may incur redundancy when policies overlap.

For example (using the schema shown in Figure 2), let us say a company has a requirement to retain all the data from both tables

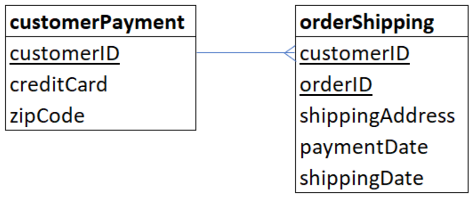

Figure 2: Example Schema

where the payment date is within the last 90 days. This would be accomplished by creating a policy using the syntax below.

This example customerOrderRetain policy definition uses Microsoft SQL Server syntax:

```
CREATE RETAIN customerOrderRetain AS
SELECT *
FROM customerPayment NATURAL JOIN orderShipping
WHERE DATEDIFF(day, orderShipping.paymentDate,
    date_part('day', CURRENT_DATE)) < 90;
```

Due to the standardization of SQL, minimal changes would be required to define a similar query in other relational databases. For example, PostgreSQL does not have an explicit `DATEDIFF` function.

For purging policies, we use the corresponding new syntax `CREATE PURGE`. When defining a purge policy, if any one primary key attribute is included in a purge policy, all other columns must be included. The purge definition must also include all child foreign keys of the table to maintain referential integrity.

Consider a policy for a company (Figure 2) that requires purging all shipping addresses where the shipping date is over 90 days old:

```
CREATE PURGE addressPurge AS
SELECT orderShipping.shippingAddress
FROM orderShpping
WHERE DATEDIFF(day, orderShipping.shippingDate,
    date_part('day', CURRENT_DATE)) > 90;
```

In this example schema, the `shippingDate` column must not contain NULL; therefore, at least one column can be used to determine the purge expiration date, satisfying our definition requirements.

4.4 Queries and Policies

SELECT. `SELECT` queries data in the active tables (which are not modified in our approach). Therefore, our framework does not introduce any additional overhead in `SELECT` queries. Retrieving data from the archive tables uses `SELECT ARCHIVE` (a new syntax that we introduce). Our framework automatically extends the `SELECT ARCHIVE` query with necessary join fields (to accommodate the additional archive-only fields such as `transactionID`). The retrieval of data from the shadow archive is preceded by a custom function (e.g., `PL/pgSQL` in PostgreSQL) which decrypts the shadow archive into a non-encrypted archive table. Depending on the size of the shadow archive, this could result in a significant number of decryptions, which could introduce a significant performance penalty.

In practice, organizations would only archive data when it is no longer actively needed. Therefore, it is unlikely that this functionality would be frequently used and, therefore, performance of archive retrieval is not a primary concern. Archive retrieval functionality would likely be invoked to retrieve data which has either been archived by accident or to comply with a legal request (e.g., a lawsuit).

INSERT. Whenever data is inserted into the database, our framework uses triggers to check if any of the values fall under a defined purging policy. If any purging policies apply to any data, the necessary values are encrypted and the values (regardless of encryption status) are inserted into the shadow tables (with the corresponding encryption key ID added in the `[column name]EncryptionID` columns). For values that do not require encryption, a value of `-1` is used for the `[column name]EncryptionID` columns.

UPDATE. `UPDATE` queries require updating the original values in the active tables as well as the shadow tables. In the shadow tables, this may require decrypting the encrypted values, updating them, and then re-encrypting them. If the `UDPATE` targets a date field that is used as the criteria for purging, this may require applying a different encryption key (due to the changing period length), to all updated values. If so, the framework decrypts all of the business record's values, determines the new key, and re-encrypts. Archived data cannot be targeted by `UPDATE`s.

With respect to retention compliance, whenever any value of a business record (subject to a retention policy) is updated, the entire business record is first archived. This action guarantees retention compliance by both providing a complete audit trail history of the business record and by assuring that `UPDATE`s will not be used to circumvent the retention policy. The `sequenceNo` column facilitates sorting the audit trail history of business records.

If a `DELETE` query targets any values belonging to a business record with retention requirements, we first check to see if any of those values also belong to a record requiring purging. If so, the encrypted values from the shadow copy are moved into the shadow archive tables. If a record does not have any purging requirement, it is still moved from the shadow table to the shadow archive (due to all data being contained in the shadow tables regardless of purging requirement status).

DELETE. Before a `DELETE` query is executed, our framework checks for any retention policies that apply to the target data. If no values require retention, the query proceeds as normal. We then run the same query targeting the values in the shadow tables. If any of the values are subject to a retention policy, the entire business record is inserted from the shadow tables into the archive tables (thereby inheriting the encryption and purging dates). Once the business record has been added to the shadow archive, the delete proceeds as normal.

4.5 Reconciling Policy Date Conflicts

Policy compliance dictates that retention takes priority over purging. If any value is subject to both a purging and retention policy, the value must first be retained until it is no longer subject to a retention policy; values may not be purged if there is still an active retention requirement. Therefore, a purging date must be greater than or equal to the retention date.

Consider the following example: Company X is required to delete customer financial data after five years. At some point in time, Company X is sued, and they retain financial data for their pending court case. They now have an active retention policy on all data relevant to the court case. If Company X were to delete the data due to it being subject to a purge policy, while the case is pending, it would be in violation of procedural law, as explained in [38]. Sanctions by the court may include the prohibition of introducing certain evidence of its defense, or the court may strike its defense altogether. Therefore, retention must take priority over any purging policy requirements.

When a business record is subject to both retention and purging, the first potential conflict that must be resolved is determining how long to save the data and when the business record must be purged. If the retention period is less than or equal to the purging date, there is no conflict, and therefore no changes are required. On the other hand, if the retention period's date is later than the purging date, we must delay the purging date until the retention policy's expiration.

This conflict resolution is achieved by determining which encryption key to use. If a retention policy dictates the purging date must be delayed, the new date will be used when determine which policy and purging date's encryption key must be used. Choosing which encryption key to apply is still determined on a per-value granularity. Because business records are defined by policies, individual values may contribute to multiple business records and would potentially be subject to multiple policies. Therefore, a purge policy's business record may only have a subset of its values purge periods postponed due to a retention policy's protection. In relational databases, in order to avoid violating a key constraint, delaying the purge of a non-key value in a tuple also requires delaying the purge of the tuple's primary key (although other non-key attributes can be purged earlier).

4.6 Backup and Restore Process

We define partial backups as either *incremental* or *delta*. For example, if we took a full backup on Sunday and daily partial backups and needed to recover on Thursday, database utilities would restore the full backup from Sunday and then either 1) apply delta backups from Monday, Tuesday, and Wednesday or 2) apply Wednesday's incremental backup. Because the trigger-based approach of this framework, delta and incremental backups are still possible, and therefore, organizations will not have to overhaul their backup processes if they depend on different backup types. Only the shadow and shadow archive tables are backed up and restored during a backup process. During a restore, our framework restores the backup with shadow and shadow archive tables. If any row in the shadow archive table has a retention date that has passed, this row is ignored during the restore.

Recall that the shadow tables include additional columns with encryption ID for each value. A -1 entry in an encryption ID column indicates that the column is not encrypted and, therefore, does not require decryption and would be restored as-is. Our framework decrypts all values with non-expired encryption keys into the corresponding active table. For any encrypted value associated with a purged encryption key our system restores the value as a NULL in the active table. Thus, non-primary-key columns subject to a purge policy must not prohibit NULLs, including any foreign key columns. We automatically expand the purging policy to include all child foreign keys of the table in order to maintain referential integrity. When all columns are purged from a row, the entire tuple will not be restored (i.e., ignored on restore).

For example, consider a database at a university where a student has a single advisor referenced by a foreign key. At this university, there is a policy that requires all faculty information to be purged when they leave the university. In that scenario, policy definition will be automatically extended by our framework to include the advisor column in the student table; the advisor column in the student table must be NULL-able.

4.7 Schema Change Limitations

Our framework prohibits any changes to the schema of any table with an active policy. This limits the ability to circumvent policy requirements by making changes to a table's logical layout. For example, if we were to allow a user to make changes to a table, a user could accidentally violate retention requirements by dropping a table column with protected values. In order to make schema changes, table policies must first be removed and reinstated after the changes have been made. This is similar to how other database constraints operate (e.g., it is not possible to drop a column belonging to a primary or a foreign key).

4.8 Key Management Overview

For our experiments we store encryption keys in a table within the same database. However, in a production environment, we require separating the `policyOverview` table from the main database to ensure that purged data cannot be decrypted. The keys must be kept in a separate database and exposed to the main database via a database link or a REST Service call. A database link allows tables from external database to be exposed to the current database as if the tables were co-located in the same database. This functionality (transparent data encryption) already exists by default or supplemental functionality in Postgres, SQL Server, Oracle, Db2, and newer versions of MySQL when using a Federated storage engine.

By keeping the encryption key data in a separate database which uses a separate backup methodology, targeted destruction of the encryption keys (associated with the purged records) is simplified. Keeping the keys separate ensures that the amount of corresponding meta-data that needs to be destroyed is significantly smaller than the primary database. The backups of the encryption key database can be relegated to regular drives rather than tape storage, making it easy to overwrite data as key ranges expire. Depending on organizational purging requirements, the key database can also be backed up to smaller external drives (e.g., USB drives) which could be physically destroyed rather than overwritten. Furthermore, one should partition the encryption keys based on a time range so the media to be erased or destroyed would be isolated to that expired range. Once the media is physically destroyed, the encrypted data would be unrecoverable.

Because business records are intermixed within files, there are values belonging to multiple business records subject to multiple policies (of both retention and purging). A full erasure of a file may at times satisfy purging requirements, but this would come at the cost of violating retention policy requirements. Therefore, a targeted erasure within a single file is required. On the other hand, with encryption keys, we must fully guarantee they are permanently irrecoverable. Because we store the encryption keys independently, a media-level or entire file erasure is possible.

For the purposes of choosing to overwrite (rather than destroy) the media, we need to consider what file erasure actually means. When a file is deleted in a most file-systems, the entry to the file's data is simply de-referenced, leaving the file content intact. In order to securely delete the file by overwriting the data several times, we recommend using a file-system which does not perform an update of a block by doing an insert and delete of the block. On Linux platform there are several tools to facilitate file erasure functionality, such as `shred`, `secure-delete`, `wipe`. The United States Department of Defense (5220.22-M) recommends 2 passes of:

(1) Writing all zeros
(2) Writing all ones
(3) Writing all random characters

After steps three and six (which is after each pass of the original three steps), a verification is conducted before proceeding [1]. In practice, this may not be necessary, but instead would depend on the discretion of the DBA and legal department.

To thoroughly cover the topic of securely deleting files, we need to discuss the differences between magnetic and solid state disks (SSD). Magnetic disks allow explicitly overwriting the same areas of disk, whereas SSDs write the changed block to a new location (due to hardware limitations and wear leveling of each cell) and then reset the block for reuse by the mechanisms of trim and active garbage collection[18]. We point out this difference out because, while unlikely, it is possible to disable garbage collection by manipulating the SSD's firmware and harvest the deleted/overwritten data. This would be more likely on an individual drive and not an SAN system due to the the LUN being spread across multiple drives and checks of the SAN system such as firmware validation.

5 EXPERIMENTS

We implemented a prototype of our framework in a PostgreSQL 14.1 database. The test database used the schema from Figure 2 and the example policies noted in Section 4.3 to demonstrate how our method supports retention and purging policies. This schema reflects only the tables needed for the policies that we evaluate and not all tables stored in the database. In practice, we expect most policies to cover data in one or two tables. Additional tables which do not directly apply to a policy will not impact query performance. The database VM server consists of 8GB of RAM, 8 vCPUs, 1 x vNIC and a 25GB VMDK file on an SSD. The VMDK file was partitioned into: 350MB/boot, 2GB swap, and the remaining storage was used for the / partition; this was done with standard partitioning and ext4 filesystem running CentOS 8 Stream on VMware Workstation 16 Pro. We demonstrate the viability of our approach by showing that it can be implemented without changing the original (user-facing) database schema and by extending backup procedures using only existing DBMS backup functionality.

We generated 30,000 test business records that would fall under 1) neither policy, 2) only the retention policy, 3) only the purging policy, and 4) both policies for the time period simulated in our experiments. Each of the four groups had approximately an equal amount of records. Our experiment confirmed our proposed framework facilitates retention and purging compliance. In Section 5.6, we verify that our framework successfully purged and retained the required business records.

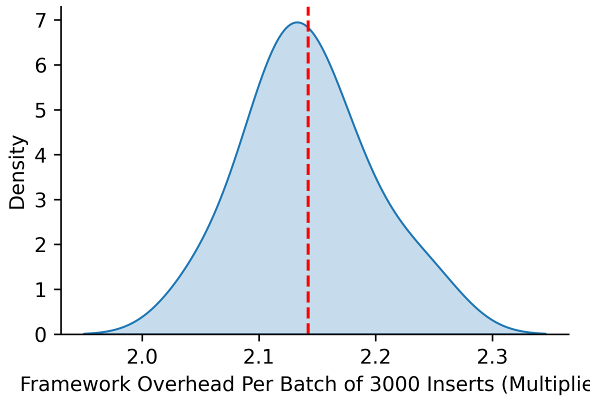

Figure 3: Framework Overhead Factor During INSERTs

5.1 Evaluating INSERT Performance

Because our framework adds data to the shadow tables during `INSERT`s, in this experiment we analyze the overhead of our trigger implementation. To determine the overhead costs, we inserted 30,002 business records (corresponding to two tables in our schema) in batches of approximately 3,000. For each batch, we measured the time the insertion took with and without our framework activated. While the framework was active, it automatically encrypted and added the business records to the shadow tables and encrypted the business records that required purging.

The distribution of our framework's overhead is shown in Figure 3. Overall, the `INSERT`s with our framework activated took an average of 2.15x times (see red line in Figure 3) compared to runtime without policy compliance (measured per 3,000 inserts). Specifically, our framework averaged 2.1 seconds per batch while a non-compliant database averaged 0.98 seconds per batch. Our experiment was designed so that every inserted row contained at least one value which required purging. In practice, not all rows in a table will contain values requiring purging and therefore will have a lower overhead cost. Overall, the overhead does increase as the number of rows requiring purging and encryption increases.

5.2 Evaluating Encryption Cost

The purpose of this experiment is to evaluate the impact of applying encryption (we used the PostgreSQL `pgcrypto` module function `PGP_SYM_ENCRYPT()`) and policy size has on our framework's overhead. Our testing policy requires purging all addresses, and therefore, every row requires a value to be encrypted. Note that within the simulated duration of time only 50% of the addresses are actually purged. However, all addresses are encrypted, with the remaining 50% to be purged in the future. To evaluate the encryption overhead cost, we eliminated the encryption step before inserting

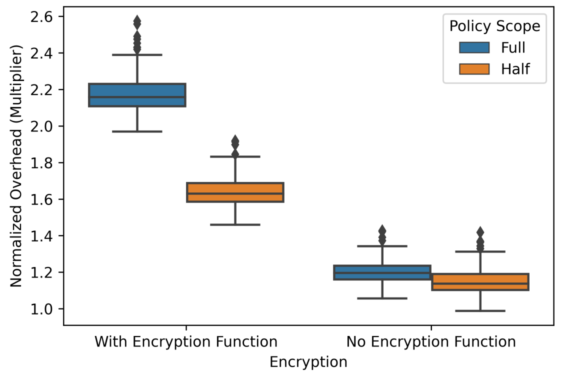

Figure 4: Encryption Overhead Analysis

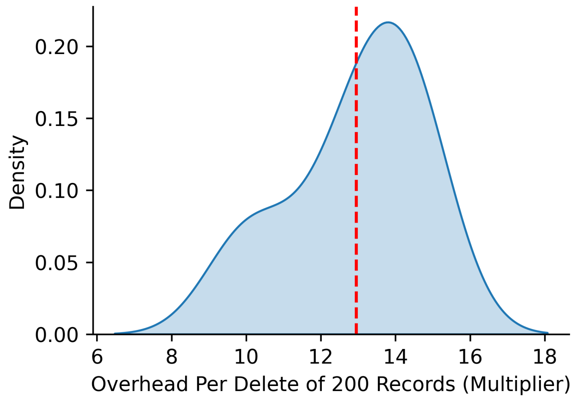

Figure 5: Framework Overhead Factor During DELETEs

the data into the shadow tables. We also considered a different purging policy that would only subject 50% of the business records to purging (i.e., 50% of the records would be exempt from purging altogether). Thus, for our evaluation, we considered five different combinations of policy and framework settings:

(1) With encryption function applicable to every record
(2) With encryption function applicable to 50% of the records
(3) Without encryption function applicable to every record
(4) Without encryption function applicable to 50% of the records
(5) Framework functionality disabled

These combinations allow us to evaluate the cost of both the encryption function and the size of the purging policy (comparing policy where 50% of the rows must be purged to the policy where 100% of the rows must be eventually purged). Figure 4 provides an overview of the runtime overheads, normalized with respect to the baseline (framework disabled).

From our analysis, we conclude that the overhead of the encryption function's application is the largest contributor to the overall runtime compared to the runtime of the triggers to check for encryption and insert the data into the shadow tables. When the policy applied to every row and encryption was applied, the average overhead is 2.15x the runtime compared to a database without our framework. In instances where encryption was applied to only half the rows, the framework overhead dropped to 1.64x on average. We then ran our no encryption variant on the data for all and half the data which returned average overheads of 1.20x and 1.15x respectively.

Therefore, we are able to conclude that the majority of the framework's overhead cost is associated with the encryption function. Although there is always going to be an associated performance cost with checking for applicable policies and inserting the data into the shadow tables, in practice, a lower cost function would greatly reduce the overhead associated with the framework.

5.3 Evaluating DELETE Performance

Our framework archives retained data whenever a DELETE is executed against it. If a table does not have applicable retention policies, there is no need to check whether archiving is required. On the other hand, if a policy is linked to a table, our framework must (at a minimum) check if archiving is necessary. Therefore, regardless of whether deleted data is archived, there is some performance overhead introduced by our framework.

For this experiment, we ran 10 DELETEs with and without our retention functionality activated. Each delete targeted approximately 200 rows in the `orderShipping` table. With our framework enabled, rows without retention would be deleted from both the active and shadow tables. For when our framework was not activated, rows were only deleted from the active tables without running any additional checks or shadow table INSERTs.

Our framework on average took 13x (red line in Figure 5) times longer to execute a DELETE targeting roughly 200 records. Note that a typical database warehouse workload contains fewer than 1% of DELETE queries [11]. Therefore, the delete compliance overhead would have a limited overall impact on the day-to-day performance of a typical database in production (this is further evaluated in Section 5.5). Furthermore, in order to identify the to-be-deleted row, our prototype decrypts the key columns (rather than match the encrypted value by equality). This is a PostgreSQL-specific implementation constraint, which may not be needed in other databases.

5.4 Database Storage Overhead

With all of our experiments, the size of the databases can be seen in Table 2 (the size does not include the transaction log). Compared to the size of the Postgres database the research conducted by Shrasti et al. [29], our framework imposes less storage overhead. The total database size with our framework after running our experiments (containing data in both the shadow and shadow archive tables) was 1.3x larger than a database that did not contain any compliance functionality (after the experiments have been completed). On the other hand, Shrasti et al.'s database with compliance functionality was noted at times to be up to 4x larger than the non-compliant databases.

Ultimately, storage overhead can greatly fluctuate depending on number and coverage of policies. For example, if the database contains more business records subject to purging or retention policies, this can greatly increase the size multiplier. Therefore, it is difficult to perform a complete equivalent comparison of the two,

Database	Size (MB)
After INSERTs With Framework	12.866
After INSERTs Without Framework	10.457
After DELETEs With Framework	13.309
After DELETEs Without Framework	10.474

Table 2: Experiment Database Size

but overall, our framework size requirement compares well to a state-of-the-art compliance system.

5.5 Evaluating Performance Using Real-World Transaction Frequencies

Recall that none of SELECT queries incur any overhead in our framework. Because real-world data warehouse workloads are typically 90% SELECTs [11], the compliance overhead would apply only to a small fraction of database queries. As we have previously evaluated the overhead of our framework on specific query transaction types, in this experiment we consider the overhead on a database whose workloads mirrors real-world transactions. For the UPDATEs, we continued to use a policy that requires purging a value in every row using the function PGP_SYM_ENCRYPT() to encrypt the data. Because our framework runs the same process for UPDATEs and INSERTs, we use UPDATEs for performance evaluation. Table 3 shows the frequency of each transaction type for this experiment. We ran the same query workload in the same order on two identical databases, one with our framework enabled and one without any additions.

These queries were run with the different types mixed randomly. We measured the elapsed time every 1000 transactions to evaluate performance. After running the above queries, our proposed framework on average had a 13% overhead compared to the database without the compliance framework activated. The overall performance can be seen in Figure 6. In smaller samples, the performance will depend on the types of queries run. Therefore, we believe that our framework can be implemented without causing an unreasonable decrease in overall performance.

5.6 Validating the Framework

At the beginning of our validation experiment, we inserted 10,002 business records into a new database. During the insertions, all records were encrypted with their corresponding encryption key (aligned to a specific date) and added to the shadow tables. We then executed DELETEs to force our system to archive necessary data before executing the query. This resulted in 1161 business records moved from the shadow tables to the shadow archive tables. Next, we created a backup, deleted required encryption keys

Query Type	Transactions
SELECT	9000
UPDATE	700
DELETE	300

Table 3: Query Transactions by Type

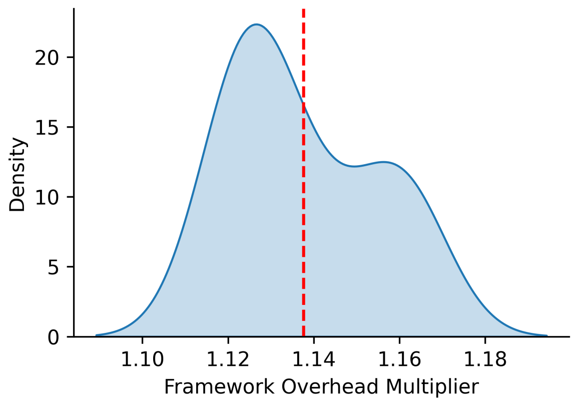

Figure 6: Framework Overhead in Practice

(corresponding to the simulation of time), and restored the database from the backup. During the restoration, 1859 records were ignored due to their encryption keys being removed and the values being purged (confirming compliance). Additionally, no other records were purged during this process. Finally, we validated that the shadow archive retained the necessary 563 business records (maintaining retention compliance) and the database kept exactly the 7,580 records as expected.

When comparing our archive to the types of data accessibility outline in Zubulake v. UBS Warburg, our archiving system would facilitate satisfactory recovery of data [38]. Additionally, when comparing our purging system to the accessibility options described by the judge in the case, our purging solution (via cryptographic erasure) at minimum renders the data inaccessible (per the judge's standards); in reality, it is considered irreversibly irrecoverable.

6 DISCUSSION

6.1 Implementation Challenges

Many organizations leverage multiple database platforms. Since data compliance is enforced at an organizational level, organizations need consistency in the tools that they use. Having a standardized process will help ensure consistent policy enforcement and avoid compliance violations such as the one discussed in Example 1.

Laws dictating policy are formulated from a privacy perspective, often before technology and implementation considerations are taken into account. Therefore, many organizations must have their legal and technology experts partner to discuss compliance processes. Having a well-defined process for implementation and enforcement of policies is necessary to bridge the gap between the legal and technology domains. Subject matter experts must be able to understand the compliance system to verify that policies are correctly interpreted and enforced. Ad-hoc solutions and systems risk misapplication of policy enforcement tools and inaccurate policy mappings, potentially leading to policy violations.

6.2 Considerations for Other Database Logical Layouts

The challenge of implementing standardized processes across systems is translating the policy requirements into platform-dependent implementations. Outside of the SQL standard adopted in RDBMS platforms, there is little consistency between the different database platforms. As we discussed, SQL standard offers the most consistency. Although SQL language extensions have more variety, they generally offer functionality necessary for data retention and purging support.

With NoSQL databases, each platform has its own assortment of processes and idiosyncrasies. From this analysis, we concluded that NoSQL systems have greater implementation variability; all compliance solutions must work within these limitations caused by the platform idiosyncrasies. Although the basic process remains the same in NoSQL as in RDBMS platforms, additional effort is necessary during the implementation of a system-specific solution (e.g., the retention trigger enforcement mechanism must be moved into the application layer).

In columnar stores, such as Db2 BLU [12] and SAP Hana [26], triggers are not universally supported like in the row-based databases. Db2 BLU, IBM's columnar store, does not support the use triggers on its tables [7], however SAP Hana does support the use of triggers. In Db2 BLU, a trigger-like functionality would be achieved by moving the same logic into the application level (for example, deploying Db2 SQL/PL in parallel to a Db2 BLU database). In other words, `INSERT`s, `UPDATE`s, and `DELETE`s must be funnelled through a SQL/PL function to ensure compliance rules are implemented.

Regarding in-memory databases, many of the major platforms provide ACID guaranties, SQL language and joins, and triggers. Oracle Times Ten is an in-memory database where the transaction and or the data are shipped to a traditional Oracle database for long term storage [23, 24]. Similarly, VoltDB is an in-memory ACID database with data and transaction logs shipped to traditional databases for long-term storage. Since these databases have niche uses, the deployment of the workflows would be dependent on where long-term data is held. Therefore, our outlined compliance workflows (see Section 4) would be implemented on the long-term storage database instead of the in-memory systems.

Because cloud storage solutions leverage the databases discussed above, from a data management perspective, no special considerations would need to be added to facilitate compliance. Enforcing compliance over the network traffic is beyond the scope of this paper.

6.3 Future of Data Governance

Additional regulations are continually being enacted by various legislative bodies. Many governments (such as New York) are still actively debating expanding data privacy law requirements [22]. Furthermore, Colorado and Virginia have fully enacted new laws to require additional customer data privacy [3]. As organizations are subject to an increasing number of policies from differing government bodies, the need for a comprehensive compliance system will only grow.

Although GDPR covers all of Europe, countries like the United States pass laws at the individual state level. Laws on privacy and business matters are generally considered matters of state legislation. With the exception of interstate commerce (which is federally regulated), requirements can greatly differ from state-to-state. Therefore, the Uniform Law Commission is attempting to create a uniform standard that simplifies the requirements for businesses to increase consistency and facilitate compliance [10]. Because the compliance domain is still maturing, systems must adapt to the changes of existing policies and evolve as the data privacy field advances.

6.4 Future Work

In this paper, we outlined a framework which can support retention and purging policy compliance in relational databases. For future research, we believe there is a large opportunity for improving runtime performance and usability. While our current framework generates PL/SQL functions and triggers to support this functionality, we would like to develop an integrated language in which the policies can be represented, modified, and validated. While we believe the performance overhead from our framework is practical, we would like to explore overhead mitigation techniques for large-scale databases and applications that are latency sensitive; for example, using a cloned copy to offload the additional work. Additionally, exploring the individual components most strongly contributing to the performance overhead would provide guidance for further optimization. Furthermore, we feel the need to explore having the archive copy in an independent database for environments subject to separation of duties, such as in a banking environment.

Another significant element of many new data governance legislation is to require customer consent before processing personal data. This requires filtering out data which has not been permissioned for certain uses (e.g., marketing). This functionality must also be implemented into databases to ensure complete compliance.

Overall, this paper focuses on data residing in a DBMS. Files created by databases which reside outside of the DBMS (e.g., logs) are also subject to retention and purging requirements. Therefore, additional operating system level compliance tools must be developed to ensure compliance outside of the DBMS.

7 CONCLUSION

Data management research must address the current shortcomings of compliance functionality in relational databases. Organizations can no longer depend on manual or ad-hoc solutions to address their compliance obligations. As organizations are increasingly being subjected to more policies, this will only increase the complexity of maintaining compliance. This proposed framework provides the necessary critical functionality for an automated compliance solution. As compliance only continues to grow in importance, developing and implementing a non-intrusive solution will be key to protecting the privacy of individuals and organizations alike.

ACKNOWLEDGMENTS

This work was partially funded by US National Science Foundation Grant IIP-2016548, CME Group, and Argonne National Laboratory.

REFERENCES

[1] 2021. DOD 5220.22-M – the secure wiping standard to get rid of data. https://www.bitraser.com/blog/dod-wiping-the-secure-wiping-standard-to-get-rid-of-data/
[2] 2021. €27,8 million GDPR fine for Italian Telecom -TIM. https://dataprivacymanager.net/
[3] Jeff Alexio. 2022. 2022 Data Privacy Laws: Emptech blog. https://emptech.com/2022-data-privacy-laws/

[4] Ahmed A Ataullah, Ashraf Aboulnaga, and Frank Wm Tompa. 2008. Records retention in relational database systems. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 873–882.
[5] Centers for Medicare & Medicaid Services. 1996. The Health Insurance Portability and Accountability Act of 1996 (HIPAA).
[6] Citus Data. 2022. pg_cron. https://github.com/citusdata/pg_cron
[7] IBM DB2. 2020. https://www.ibm.com/
[8] European Union 2020. Regulation (EU) 2016/679 of the European Parliament and of the Council. https://gdpr.eu/tag/gdpr/
[9] Oracle Goldengate. 2018. https://docs.oracle.com/goldengate/
[10] Jake Holland. 2022. 2022 privacy legislation success viable as three states lead way. https://news.bloomberglaw.com/
[11] Windsor W Hsu, Alan Jay Smith, and Honesty C. Young. 2001. Characteristics of production database workloads and the TPC benchmarks. *IBM Systems Journal* 40, 3 (2001), 781–802.
[12] IBM. 2020. https://www.ibm.com/docs/en/db2/11.5?topic=tables-shadow
[13] IBM. 2022. About InfoSphere CDC. https://www.ibm.com/support/knowledgecenter
[14] IBM. 2022. InfoSphere Optim Archive. https://www.ibm.com/products/infosphere-optim-archive
[15] International Commissioner's Office. 2020. Right to Erasure. https://ico.org.uk/
[16] International Data Sanitization Consortium. 2017. Data Sanitization Terminology and Definitions. https://www.datasanitization.org/data-sanitization-terminology/
[17] Seny Kamara and Kristin Lauter. 2010. Cryptographic cloud storage. In *International Conference on Financial Cryptography and Data Security*. Springer, 136–149.
[18] Richard Kissel, Andrew Regenscheid, Matthew Scholl, and Kevin Stine. 2014. Guidelines for Media Sanitization. https://csrc.nist.gov/publications/detail/sp/800-88/rev-1/final
[19] Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner. 2019. SchengenDB: A data protection database proposal. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 24–38.
[20] Kronisch 1998. Kronisch v. US (2nd Circuit).
[21] Ben Lenard, Alexander Rasin, Nick Scope, and James Wagner. 2021. What Is Lurking in Your Backups?. In *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 401–415.
[22] Pravin Mehta. 2022. New York Privacy Act 2021: An Insight. https://www.bitraser.com/article/new-york-data-privacy-law.php. *BitRaser* (2022).
[23] Oracle. 2009. https://www.oracle.com/a/tech/docs/timesten-usps-customer-case-1446282.pdf
[24] Oracle. 2019. https://www.oracle.com/a/tech/docs/con4758-tmobile-timesten-combined.pdf
[25] PostgreSQL 2022. PGXN. https://pgxn.org/dist/mimeo/1.2.3/doc/mimeo.html
[26] SAP. 2022. https://help.sap.com/viewer/6b94445c94ae495c83a19646e7c3fd56/2.0.01/en-US/bd2e9b88bb571014b5b7a628fca2a132.html
[27] Nick Scope, Alexander Rasin, James Wagner, Ben Lenard, and Karen Heart. 2021. Database Framework for Supporting Retention Policies. In *International Conference on Database and Expert Systems Applications*. Springer.
[28] Nick Scope, Alexander Rasin, James Wagner, Ben Lenard, and Karen Heart. 2021. Purging Data from Backups by Encryption. In *International Conference on Database and Expert Systems Applications*. Springer.
[29] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. 2019. Understanding and benchmarking the impact of GDPR on database systems. *arXiv preprint arXiv:1910.00728* (2019).
[30] Damon W Silver and Gregory C Brown. 2022. $600,000 reasons to review your shield act compliance program: NY attorney general announces significant settlement stemming from email data breach. https://www.natlawreview.com/article/600000-reasons-to-review-your-shield-act-compliance-program-ny-attorney-general
[31] The Office of the National Coordinator for Health Information Technology. 2022. State Medical Record Laws: Minimum Medical Record Retention Periods for Records Held by Medical Doctors and Hospitals.
[32] United States Congress. 1948. 28 U.S. Code §1732. https://www.law.cornell.edu/uscode/text/28/1732
[33] James Wagner, Alexander Rasin, and Jonathan Grier. 2015. Database forensic analysis through internal structure carving. *Digital Investigation* 14 (2015), S106–S115.
[34] James Wagner, Alexander Rasin, and Jonathan Grier. 2016. Database image content explorer: Carving data that does not officially exist. *Digital Investigation* 18 (2016), S97–S107.
[35] James Wagner, Alexander Rasin, Karen Heart, Tanu Malik, and Jonathan Grier. 2020. DF-toolkit: interacting with low-level database storage. *Proceedings of the VLDB Endowment* 13, 12 (2020).
[36] James Wagner, Alexander Rasin, Tanu Malik, Karen Heart, Hugo Jehle, and Jonathan Grier. 2017. Database Forensic Analysis with DBCarver. CIDR.
[37] Ben Wolford. 2020. Everything you need to know about the "Right to be forgotten". https://gdpr.eu/right-to-be-forgotten/
[38] Zubulake 2005. Zubulake v. UBS WARBURG LLC, No. 02 Civ. 1243 SAS, Dist. Court, SD New York. https://sosmt.gov/wp-content/uploads/attachments/E-ZubulakeV.pdf?dt=1519325634100