Database Framework for Supporting Retention Policies

Nick Scope[1], Alexander Rasin[1], James Wagner[2], Ben Lenard[1], and Karen Heart[1]

[1] DePaul University, Chicago, IL 60604, USA
[2] University of New Orleans, New Orleans, LA. 70148, USA

Abstract. Compliance with data retention laws and legislation is an important aspect of data management. As new laws governing personal data management are introduced (e.g., California Consumer Privacy Act enacted in 2020) and a greater emphasis is placed on enforcing data privacy law compliance, data retention support must be an inherent part of data management systems. However, relational databases do not currently offer functionality to enforce retention compliance.
In this paper, we propose a framework that integrates data retention support into any relational database. Using SQL-based mechanisms, our system supports an intuitive definition of data retention policies. We demonstrate that our approach meets the legal requirements of retention and can be implemented to transparently guarantee compliance. Our framework streamlines compliance support without requiring database schema changes, while incurring an average 6.7% overhead compared to the current state-of-the-art solution.

Keywords: Retention Compliance · Databases · Privacy.

1 Introduction

Laws intended to protect privacy, prevent fraud, or support financial audits require companies to implement data retention policies. Companies may also establish internal data retention policies for confidential data (e.g., for routine business operation or audits) and to minimize risks (e.g., data destruction to prevent theft). Thus, companies can be subject to multiple data retention policies requiring preservation of some data and deletion of other data. For example, the US Health Insurance Portability and Accountability Act [13] requires medical data to be retained for at least 6 years, the Children's Online Privacy Protection Act states that personal information for children is retained "for only as long as is reasonably necessary to fulfill the purpose for which the information was collected" [15]. Moreover, recent laws such as European General Data Protection Regulation and California Consumer Privacy Act [7] established the "right to be forgotten", which entitles individuals to request deletion of their personal data.

Relational database management systems (DBMS) do not support mechanisms to enforce data retention requirements. As a result, organizations build

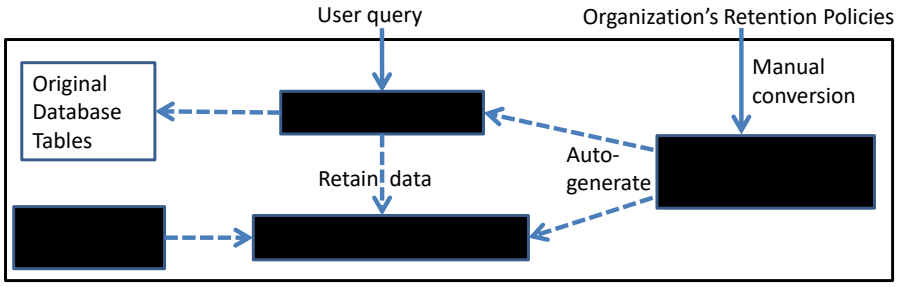

Fig. 1. Retention workflow overview. Gray boxes represent new components; dashed line represents automated framework steps; solid line represents manual steps.

ad-hoc solutions manually. As retention laws are created, databases will need to support automated retention compliance. Additionally, the solution must be intuitive for database curators to set up, and transparent from user's perspective.

In this paper, we describe a framework implementation that can guarantee data retention compliance in a relational database. Our approach builds on the work of Ataullah et al. [8] by expanding DBMS functionality to facilitate compliance with legal requirements of data retention. For example, we transparently move deleted (but retained) data to an archive (reflecting its new status) rather than block the delete operation.

Figure 1 provides an overview of our approach for enforcing retention policies. User delete (or update) transactions are allowed to proceed normally, but data that must be retained are automatically and transparently copied into the archive. As long as the retention policies are correctly defined (see Section 4), our database triggers can guarantee compliance by reacting to changes in data. `SELECT` queries are not affected, because deleted data is always removed from the original database tables. Archive tables store all deleted-but-retained data, mirroring the active table with two additional columns: *archivePolicy* and *transactionID* (see Section 4). Our contributions in this paper are:

- We define the requirements a database must support and enforce to comply with data retention policies (Section 2).
- We outline an add-on framework for complying with retention requirements within any relational database (Section 4).
- We detail how our framework meets the various requirements to facilitate data retention compliance (Section 5).

Section 6 evaluates our framework performance. We demonstrate that the retention policy implementation overhead is proportional to the number of tables and the number rows archived per-transaction. We further demonstrate that our extended functionality incurs only a 6.7% overhead over Ataullah et al. [8].

2 Retention Definitions and Requirements

Business Records: Data retention policies operate in terms of a business record. 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" [9].

Defining Policy: Business records may span multiple tables; therefore a comprehensive data retention framework must allow the mapping of policies across tables. Although some requirements may only preserve the records against deletion, other domains (such as the medical field) require that a complete history of record updates is retained as well. Our approach relies on SQL view syntax to define policies, making it intuitive for a database administrator (DBA) to formulate and verify policy settings. As long as the view correctly defines the protected business records, our framework will correctly identify them.

Enforcing Policy Compliance: Data in a database can be deleted or updated either by user directly (using SQL) or indirectly (e.g., by trigger effects). A comprehensive data retention framework must ensure that data is retained and purged according to the policy definition. It should not be possible for any action to bypass or interfere with retention rules; at the same time, retention should not interfere with normal database operation. We rely on triggers to ensure that all data changes (direct or indirect) are checked against the policies. Archiving the data *before* the change takes place guarantees non-interference. Retention requirements typically preserve data for a period of time (although some may be permanently such as "for life of the company"). We use a DBMS-specific built-in scheduler to perform a regular purge as requested by a policy (see Section 5). Finally, our approach relies completely on DBMS functionality to minimize external dependencies; consequently, transactional ACID guarantees are maintained as supplied by the DBMS.

3 Related Work

Public Research: Ataullah et al. described some of the challenges associated with record retention implementation in relational databases [8]; the authors propose an approach that uses a view-based structure to define business records for retention rules, similar to our solution. However, instead of interrupting queries, we allow them to proceed as-is after we archive retained business records (conceptually similar to a write ahead log). As discussed in Sections 4 and 5, our approach may generate redundancy but avoids risk of retention policy conflict.

Private Sector Tools: Amazon S3 offers an object life-cycle management tool. The DoD's "Electronic Records Management Software Applications Design Criteria Standard" (DoD 5015-02-STD) defines requirements for record-keeping systems storing DoD data; a DoD compliant retention system must support retention thresholds such as time or event (C2.2.2.7). S3's object life-cycle management is limited to time criteria only. Moreover, S3 is file-based and therefore lacks sufficient granularity.

Oracle's Golden Gate (GG) [14] or IBM's Change Data Capture (CDC) [1] allow changes to be replicated from one database to another. However, these software packages are not specifically designed to support data retention re-

quirements (although they could be expanded to support it, similarly to our approach). GG operates by inspecting REDO logs, making it difficult to incorporate the concept of business records.

Mimeo [5] provides similar functionality for Postgres; similar to 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 [11] has archiving functionality which can be used for retention. It archives data from an active database, removing data from active storage. Users define business records for archiving using Select-Project-Join (SPJ) queries, same as our and Ataullah et al. [8] approaches. A major limitation of IBM's solution is that archiving must be initiated manually or by a script.

4 Policy Setting

Policy Mapping: In practice, DBAs work with domain experts and legal counsel to implement retention policies. We assume that DBAs can expresses a business record as a view and that relevant event data is available in the database (e.g., the date of receiving a subpoena to preserve certain data). Initially mapping the business records and retention policies to database tuples is a manual process; thereafter, our SQL-based system automates enforcement of the policies.

Creating Retention Policies: A business record is mapped as a Select-Project-Join (SPJ) view. SPJ queries are sufficient to define regulations and contractual terms business records (used by both IBM [11] and Ataullah et al. [8]). We propose new SQL syntax, `CREATE RETAIN`, to implement views that express the business records that must be protected from deletion.

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

Suppose that a company imposes a retention rule that requires retaining all applications and their interview history data (excluding interview notes) for any hired applicant. When a `DELETE` is issued, the target data is checked against existing policies to see if it belongs to a business record(s) that must be archived. For each active retention policy, our system automatically creates a mirror policy for the archive. Archive protection policies guarantee that business records in the archive will not be purged until the retention criteria has expired. The names of the archive policy and archive tables are prepended with "archive_". The archive tables and policies also include additional columns: *transactionID* to purge retained data in instances of aborted transactions and *archivePolicy* to purge policy-specific records in the archive. Because the same table row may be archived by two different overlapping policies, *archivePolicy* also serves to uniquely identify archive rows.

Because some retention requirements mandate a complete history of updates, we propose an optional `EXACT` keyword to the `RETAIN` syntax in order to implement views that identify business records that must retain a complete history. `RETAIN EXACT` policy ensures that the business record is archived before an update. Preserving a comprehensive history is a common requirement in the medical field where every update to a patient's record must be preserved.

Any column subject to an active retention policy (`RETAIN` or `RETAIN EXACT`) cannot be removed or altered in the schema. In order to drop or change a column, the DBA would first remove all retention policies that apply to that column. Our approach is designed to behave like any other database constraint (e.g., a foreign key) and therefore it must be addressed before schema changes can be applied.

5 Policy Execution

Enforcing Retention Policies: Our system uses triggers to check which of the to-be-deleted rows fall under retention policies. To ensure transactional consistency, we archive retained rows before proceeding with deletion; should the `DELETE` or `UPDATE` transaction abort, we will (eventually) delete unneeded records from the archive. Archive clean up can be executed any time after the `DELETE` transaction was aborted because all archive entries are uniquely identified by a transaction id and retention policy. Anytime a `DELETE` is run on a table with retention protections, a `BEFORE DELETE` trigger would fire and insert all data protected by retention policies into the archive table(s) before `DELETE` executes. Columns not covered under a retention policy default to `NULL` in the archive. Using triggers ensures that we protect *all* data, including data that is indirectly targeted by cascading `DELETE`s and `UPDATE`s. Similar to how `RETAIN` protects data against deletions, policies defined using `RETAIN EXACT` additionally archive all protected business records when an `UPDATE` is made to the underlying data. If a user were to update a table that is protected by `RETAIN EXACT`, the policy (at a minimum) would also include referenced primary keys of other tables. If additional update queries target the same business record, the data would again be copied to the archive before the `UPDATE` query executes.

Interacting with the Archive: The retention archive tables contain deleted data and should not be accessible without special permissions. We propose the SQL syntax `SELECT ARCHIVE` to retrieve data from the archive (similar to how IBM's InfoSphere Optim Archive operates). `INSERT`, `DELETE`, and `UPDATE` operations against the archive are prohibited to protect data integrity. To comply with the data purging requirements, we propose a `PURGE` command that deletes all eligible (i.e., no longer protected by a retention policy) records from the archive. The `PURGE` command translates the name of a provided policy into a series of `DELETE` queries. Because the records in the archive incorporate the policy name, `PURGE` does not require checks for overlapping policies. Organizations may wish to automatically and regularly purge all data which is no longer required to be retained. For example, HIPAA requires medical information to be retained as long as it is used, after which it must immediately be removed [13]. A regular

and automatic purging would remove all unprotected records. The purging process would be executed by the DBMS scheduler ([6] in Oracle, [2] in Db2, [4] in PostgreSQL, [3] in MariaDB). Most DBMSes include a cron-like scheduler to execute tasks on a set interval.

6 Experiments

In order to evaluate the performance of our framework, we measure the per-transaction runtime overhead. We assess linear schemas (a linked series of tables), where each table is linked to a single child table with a foreign key. All tables include columns for a primary key (`char`), a foreign key (`char`), a retention criteria (`boolean`), a delete criteria (`boolean`), and lorem (`varchar`), except the top-most parent table of the schema which does not contain a foreign key column.

The child-most table in the schema always contains approximately 50M rows (roughly 3.38GB). Each primary key from a parent table is joined to an average of 2.5 rows in its child table. The dataset for each schema was built independently to fit these parameters. We chose this schema type as the most expensive (to measure the upper-bound overhead) for implementing retention; a star schema would offer additional choices for optimizing join queries. Our experiments were performed on a server with an Intel i7 7700k processor with 16GB of RAM on spinning disk drive using PostgreSQL 12.3 with default settings on Windows 10. Both the Python 3.7 script (used to collect the runtimes) and PostgreSQL ran on the same machine.

The goal of these experiments is to determine the driving factors for our framework's overhead and quantify the performance penalties. The runtimes are evaluated on a per-query single-transaction basis. We evaluate delete transactions that affect between 1 and 100 rows (we also verified that update overhead is equivalent to delete overhead). The size of the average transaction is based on the evaluation performed by Hsu et al. [10] who quantified the number of pages written by real-world database workloads. The median number of pages written by a transaction was shown as 1.1 on average, with variations between domains (e.g., Bank, Retail, Insurance) [10]. In our analysis, we therefore assume that the number of rows written by one transaction is frequently less than 10.

Framework Overhead Analysis: In this experiment, we tested combinations of the policy size (0-100 rows), delete size (1-100 rows), and overlapping percents (0-100%). Overlap percent refers to the intersection of the retention policy and the delete query (e.g., a `DELETE` of 50 rows and a policy covering 50 rows with overlap of 50% corresponds to an overlap of 25 rows). Overlap rows refers to the number of rows that are ultimately archived (e.g., 25 in this example).

To establish a performance baseline, we executed `DELETE`s without our retention framework. In subsequent experiments, we subtracted the runtime of baseline `DELETE`s from the runtime with retention enabled. We then normalized the results by computing the percentage overhead introduced by our system.

There was a 0.894 correlation between the number of rows requiring archiving and the performance overhead (illustrated in Figure 2). This was the strongest

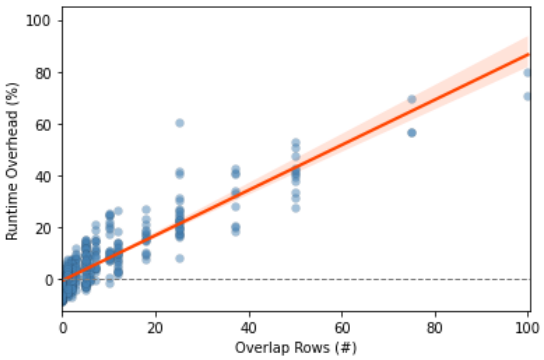

Fig. 2. Single Delete Txn Overhead (Two Tables)

relationship between variable combination. On the other hand, the size of the policy had a correlation of 0.192 with the runtime overhead percent, and the delete size had a correlation of 0.279. Checking if rows require archiving minimally impacts the runtime. Therefore, the overhead is driven by archiving the business records.

Furthermore, the overhead is modeled using a linear regression (illustrated as the line in Figure 2). The model shows the overhead of our framework is a function of the number of archived rows. Our experiments found that for fewer than 10 overlap rows, the runtime overhead with archiving was statistically insignificant compared to the runtime without archiving (Figure 2). In practice, most `DELETE`s and `UPDATE`s do not target a large quantity of rows [10]. Therefore we conclude that our framework overhead is acceptable in practice.

Comparison of Archiving to State-of-the-Art: The major difference between our proposed approach and Ataullah et al.'s work [8] is that we automatically and transparently archive retained data instead of blocking the transaction with an error. In this experiment, each query is executed as a separate transaction; therefore, whenever an exception is returned, that single query is undone (no additional rollback of previous transactions). As with Ataullah et al. [8], we use triggers to check queries against defined policies. In this experiment, we compare the runtime of these transactions when stopping the transaction using an exception versus archiving the data and letting the transaction proceed. We used the same process and data as the previous experiment.

Overall, our process averaged a runtime overhead of 6.7% compared to Ataullah et al. [8]. Although our framework introduces overhead, it eliminates potential conflicts between the retention system and existing user queries and triggers. Therefore, it ensures that organization processes are continued without the concern of violating retention policy requirements.

7 Conclusion and Future Work

In this paper, we presented and evaluated a database framework for retention policy compliance. We use views to define business records and policy conditions,

thereby ensuring accurate retention (as long as the view correctly reflects the business records). When records are targeted by a delete or an update query, they are automatically and transparently retained in the archive before the data is modified. Our framework has the significant benefit of using SQL-based commands to define policies and automates archiving business records through triggers. Our experiments demonstrate that our framework can guarantee retention compliance requirements with an acceptable performance overhead.

Although our framework ensures that requested data is deleted, some data will remain in the underlying database storage as well as in previously created backups [12]. Further research must address these sources of remaining data to fully facilitate retention compliance in purging databases. Additionally, we plan to investigate migrating the archive tables to an external DBMS instance. Finally, we plan to extend a similar framework to NoSQL databases.

References

1. https://www.ibm.com/support/knowledgecenter/SSTRGZ_10.2.1/com.ibm.cdcdoc.cdcforzos.doc/concepts/infospherechangedatacaptureoverview.html
2. Admin_task_add procedure - schedule a new task, https://www.ibm.com/support/knowledgecenter/SSEPGG_11.1.0/com.ibm.db2.luw.sql.rtn.doc/doc/r0054371.html
3. Event scheduler, https://mariadb.com/kb/en/event-scheduler/
4. pg_cron, https://github.com/citusdata/pg_cron
5. Pgxn, https://pgxn.org/dist/mimeo/1.2.3/doc/mimeo.html
6. Scheduling jobs with oracle scheduler, https://docs.oracle.com/cd/E11882_01/server.112/e25494/scheduse.htm#ADMIN034
7. California consumer privacy act (Jul 2020), https://oag.ca.gov/privacy/ccpa
8. Ataullah, A.A., Aboulnaga, A., Tompa, F.W.: Records retention in relational database systems. In: Proceedings of the 17th ACM conference on Information and knowledge management. pp. 873–882 (2008)
9. Congress, U.S.: 28 u.s. code §1732 (1948)
10. Hsu, W.W., Smith, A.J., Young, H.C.: Characteristics of production database workloads and the tpc benchmarks. IBM Systems Journal **40**(3), 781–802 (2001)
11. IBM: Infosphere optim archive, https://www.ibm.com/products/infosphere-optim-archive
12. Lenard, B., Rasin, A., Scope, N., Wagner, J.: What is lurking in your backups? In: IFIP International Conference on ICT Systems Security and Privacy Protection. pp. 401–415. Springer (2021)
13. for Medicare & Medicaid Services, C., et al.: The health insurance portability and accountability act of 1996. http://www.cms.hhs.gov/hipaa p. 158 (1996)
14. Oracle: (Sep 2018), https://docs.oracle.com/goldengate/c1230/gg-winux/GGCON/introduction-oracle-goldengate.htm#GGCON-GUID-EF513E68-4237-4CB3-98B3-2E203A68CBD4
15. University, W.S.: Retention guidelines for protected data (2020), https://www.libraries.wright.edu/special/recordsmanagement/retention