DBCompliant: Extending Database Management Systems to Support Compliance Functionality*

Alexander Rasin✉[1], Nick Scope[1], Ben Lenard[1], Moaz Reyad[1], and James Wagner[2]

[1] DePaul University, Chicago, IL 60604, USA
arasin@cdm.depaul.edu
[2] The University of New Orleans, New Orleans, LA. 70148, USA

Abstract. Data privacy policy requirements are a quickly evolving part of the data management domain. Healthcare (e.g., HIPAA), financial (e.g., GLBA), and general laws such as GDPR or CCPA impose controls on how personal data should be managed. Relational databases do not offer built-in features to support data management features to comply with such laws. As a result, many organizations implement ad-hoc solutions or use third party tools to ensure compliance with privacy policies. However, external compliance framework can conflict with the internal activity in a database (e.g., trigger side-effects or aborted transactions). In our prior work, we introduced a framework that integrates data retention and data purging compliance into the database itself, requiring only the support for triggers and encryption, which are already available in any mainstream database engine. In this demonstration paper, we introduce DBCompliant – a tool that demonstrates how our approach can seamlessly integrate comprehensive policy compliance (defined via SQL queries). Although we use PostgreSQL as our back-end, DBCompliant could be adapted to any other relational database. Finally, our approach imposes low (less than 5%) user query overhead.

Keywords: database compliance, GDPR, retention, purging, privacy

1 Introduction

Data management in an organization is bound by data privacy regulations that specify how the data must be stored (e.g., archived, backed up, or destroyed). Legislation is passed with the intent of giving customers more control over their data and privacy. An organization may also impose additional internal policies or need to comply with policies of a business associate. Violating these requirements may result in large fines or a loss of reputation, such as a data breach that compromises "expired" (i.e., should have already been deleted) data.

Reliably tracking and accurately enforcing a single policy can be a challenge when it spans select columns from multiple tables or when retention and purging

* Partially funded by the US National Science Foundation Grant IIP-2016548

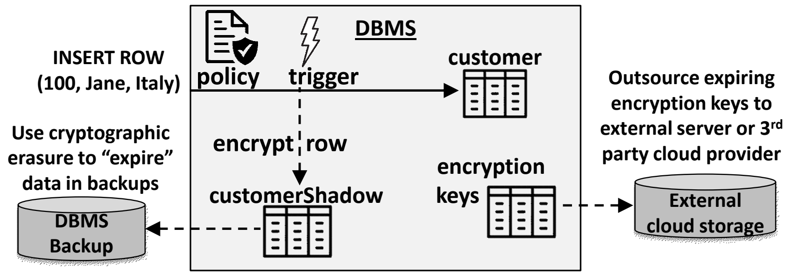

INSERT ROW
(100, Jane, Italy)
policy
trigger
DBMS
customer
encrypt row
customerShadow
encryption keys
Use cryptographic erasure to "expire" data in backups
DBMS Backup
Outsource expiring encryption keys to external server or 3rd party cloud provider
External cloud storage

Fig. 1. Data lifecycle workflow changes in a DBMS to support data purging policies. For example, if name (but not country) is subject to purging, "Jane" will be encrypted when placed into the `customerShadow` table. Retention policies (not pictured) will similarly use triggers and defined policies to retain data in an additional `customerArchiveShadow` table, which archives the data and applies encryption based on applicable policies.

rules overlap over the same data. Policies can be complex and vary by jurisdiction. For example, in the United States [8]: Oregon hospitals to retain all records for 10 years after the last discharge; Hawaii requires medical record history to be retained for 7 years after the last data entry. North Carolina hospitals are required to retain patient data for 11 years following discharge, while the data of minors must be retained until the patient's 30th birthday instead.

2 Related Work

Ataulla et al. [1] first proposed the idea of defining data retention policies through SQL queries and enforcing them through triggers as a first step towards native DBMS policy support. Scope et al. [7] first leveraged DBMS triggers (natively supported by all major database vendors) and made revisions to the backup workflow to support policy-based data purging. Scope et al. [6] also proposed using triggers to enforce retention by archiving (instead of blocking as in Ataulla et al. [1]) the deleted records which are still covered by a retention policy.

Figure 1 summarizes the integration of our data purging steps in a DBMS (retention mechanism is not pictured). Policies are defined with database queries, such as "customer name, address, and phone columns in EU must be retained for a duration of 5 years". Each inserted (or updated) row is checked by a trigger against applicable policies. Any fields covered by a purging requirement are encrypted by a trigger and inserted into the `customerShadow` table. The `customerShadow` table replaces the active `customer` table in database backups. This allows for "remote" erasure (upon policy expiration) by deleting the corresponding encryption. In order to fully satisfy purging requirements, the database must also securely delete encryption keys from backups [3]. The keys can be managed through a third-party service or in-house (see Scope et al. [5]).

Our corresponding retention mechanism (not pictured in Figure 1) checks deleted rows for values that are currently protected by a retention policy. Such values are stored in an archive table (e.g., `customerArchive`). Archived data is the data that was deleted but could not be destroyed due to an active retention policy. Since archived data remains subject to purging rules, in practice we store archived data in a `customerArchiveShadow`, which follows the same policy-based encryption principles as `customerShadow` (see Scope et al. in [4]).

3 DBCompliant

In this demo, the user interacts with a DBCompliant front-end to add or remove retention and purging policies, and can verify compliance with defined policies in a database. We use a schema based on MIMIC [2] with PostgreSQL 14.1 at the back-end (see Figure 2). PostgreSQL is running on a remote cloud instance and is a default installation with added `pgcrypto` extension that enables encryption for the shadow tables. Thus, although we use PostgreSQL, the demo is applicable to any major relational database (with minor syntactic adjustment for triggers, as our demo assumes PL/pgSQL syntax for triggers).

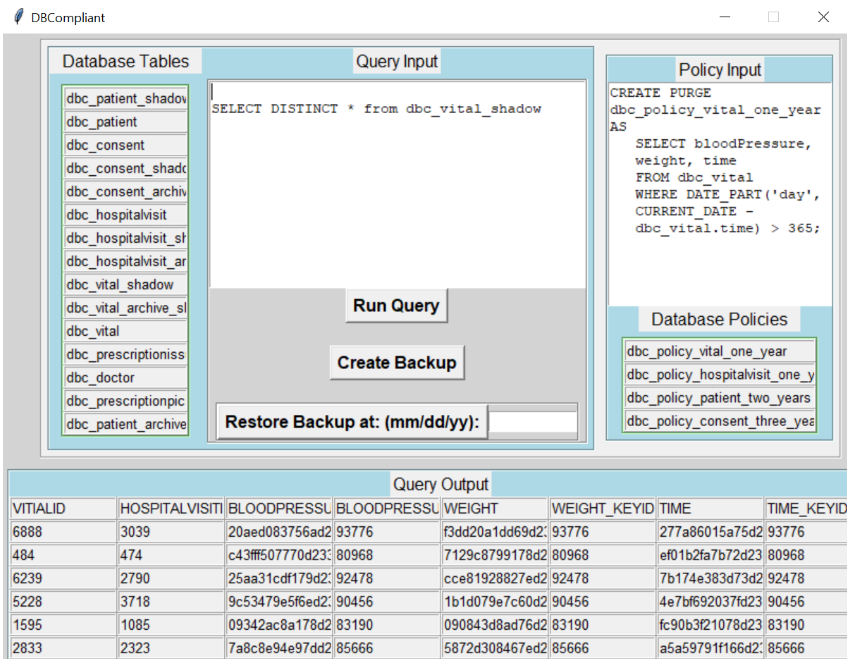
Initialization: Initialization of DBCompliant creates all tables necessary to support retention and purging capabilities: `tableName_shadow` for purging and `tableName_archive_shadow` for retention, respectively (e.g., `vital_shadow` in Figure 2). Additional tables will be created automatically if a new policy (on a new table) is created. The initialization also requires a key management table in order to support purging policies (retention policies do not require encryption keys). In our demo, the table containing encryption keys is local (see [5]) instead of a third party service. Finally, we create and initialize sequence objects for bookkeeping. Our approach also requires changing the backup and restore workflow, e.g., backing up `vital_shadow` instead of `vital` to enable data purging.

Policy Creation: As discussed in [4], we propose a view-like policy definition which enables administrators to use SQL queries to describe and review their policy requirements. For example, purging the 3 following columns from `dbc_vital` table after 1 year can be described as:

```
CREATE PURGE dbc_policy_vital_one_year AS
  SELECT bloodPressure, weight, time FROM dbc_vital
  WHERE DATE_PART('day', CURRENT_DATE - dbc_vital.time) > 365;
```

User Interaction: In Figure 2 UI, users can run queries against the database to inspect tables (output of the queries is displayed at the bottom of the window). Policy input window with the corresponding list of policies below is on the right side of Figure 2. The query output displayed in Figure 2 contains rows from `dbc_vital_shadow` and is showing the encryption imposed by the `dbc_policy_vital_one_year` policy. The three columns subject to (future) purge are encrypted and the corresponding decryption keys are available in the additional columns (to decrypt during restore). First two columns are unaffected.

Retention functionality is self-evident in the archive table. In order to observe the purging functionality, the user will create a current database backup and restore the database at an arbitrary time for the future. Thus, they could observe data values expiring at the future time based on their custom-created policies.



```
SELECT DISTINCT * from dbc_vital_shadow
```

```
CREATE PURGE
dbc_policy_vital_one_year
AS
   SELECT bloodPressure,
   weight, time
   FROM dbc_vital
   WHERE DATE_PART('day',
   CURRENT_DATE -
   dbc_vital.time) > 365;
```

VITIALID	HOSPITALVISITI	BLOODPRESSU	BLOODPRESSU	WEIGHT	WEIGHT_KEYID	TIME	TIME_KEYID
6888	3039	20aed083756ad2	93776	f3dd20a1dd69d2	93776	277a86015a75d2	93776
484	474	c43fff507770d23	80968	7129c8799178d2	80968	ef01b2fa7b72d23	80968
6239	2790	25aa31cdf179d2	92478	cce81928827ed2	92478	7b174e383d73d2	92478
5228	3718	9c53479e5f6ed2	90456	1b1d079e7c60d2	90456	4e7bf692037fd23	90456
1595	1085	09342ac8a178d2	83190	090843d8ad76d2	83190	fc90b3f21078d23	83190
2833	2323	7a8c8e94e97dd2	85666	5872d308467ed2	85666	a5a59791f166d2	85666

Fig. 2. DBCompliant UI for creation and verification of purging and retention policies.

References

1. 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)
2. Johnson, A.E., Pollard, T.J., Shen, L., Lehman, L.w.H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., Mark, R.G.: Mimic-iii, a freely accessible critical care database. Scientific data **3**(1), 1–9 (2016)
3. Reardon, J., Basin, D., Capkun, S.: Sok: Secure data deletion. In: 2013 IEEE symposium on security and privacy. pp. 301–315. IEEE (2013)
4. Scope, N., Rasin, A., Lenard, B., Heart, K., Wagner, J.: Harmonizing privacy regarding data retention and purging. In: Proceedings of the 34th International Conference on Scientific and Statistical Database Management. pp. 1–12 (2022)
5. Scope, N., Rasin, A., Lenard, B., Wagner, J.: Compliance and data lifecycle management in databases and backups. In: International Conference on Database and Expert Systems Applications. pp. 281–297. Springer (2023)
6. Scope, N., Rasin, A., Wagner, J., Lenard, B., Heart, K.: Database framework for supporting retention policies. In: Database and Expert Systems Applications: DEXA 2021, September 27–30, 2021, Proceedings, Part I 32. pp. 228–236. Springer (2021)
7. Scope, N., Rasin, A., Wagner, J., Lenard, B., Heart, K.: Purging data from backups by encryption. In: Database and Expert Systems Applications: DEXA 2021, September 27–30, 2021, Proceedings, Part I 32. pp. 245–258. Springer (2021)
8. The Office of the National Coordinator for Health Information Technology: State medical record laws: Minimum medical record retention periods for records held by medical doctors and hospitals (2022)