Query Language Support for Timely Data Deletion
Subhadeep Sarkar
ssarkar1@bu.edu
Boston University, MA, USA
Manos Athanassoulis
mathan@bu.edu
Boston University, MA, USA

ABSTRACT
A key driver of modern data systems is the requirement for fast ingestion while ensuring low-latency query processing. This has led to the birth of write-optimized data stores that realize ingestion (inserts, updates, and deletes) in an out-of-place manner. Deletes in such out-of-place data stores are performed logically via invalidation while retaining the invalidated data for arbitrarily long. At the same time, with new policy changes, such as the introduction of the right to be forgotten (in EU’s GDPR), the right to delete (in California’s CCPA and CPRA), and the deletion right (in Virginia’s VCDPA), the importance of timely and persistent deletion of user data has become critical.

In this paper, we point out that state-of-the-art query languages lack the necessary support to express a user’s preferences for data retention and deletion. Toward this, we first identify two classes of deletes: (i) retention-based deletion and (ii) on-demand deletion, that need to be supported for regulation compliance. Next, we present the challenges in transforming these user deletion requirements into application-level specifications. For this, we propose query language extensions that can express both on-demand and timely persistent deletion of user data. Finally, we discuss how the application and system level modifications work hand-in-hand under the privacy regulations and act as stepping stones toward designing deletion-compliant data systems.

1 INTRODUCTION
Disruptive technological advancements across domains of computer science, such as the Internet-of-things, edge computing, 5G communications, and autonomous vehicles, generate a vast amount of personal data that is processed at large by the data companies [23, 31, 55]. The increasing demand for efficient collection, storage, and processing of user data has driven the development of data systems that can sustain high ingestion rates without compromising the ability to analyze the data quickly. Furthermore, modern data systems are designed around the assumption of perpetual data retention with no latency-bound on the time taken to physically delete a data object.

The Out-of-Place Paradigm. To support fast data ingestion and efficient data access, modern data system designs heavily rely on the out-of-place paradigm, which (i) is highly optimized for writes and (ii) offers low-latency query processing, (iii) without causing read/write interference. Thus, the out-of-place paradigm has been adopted in several relational and array-based data stores [13, 26, 30, 36, 38, 41, 44, 53, 64], and NoSQL data stores [8, 10, 12, 25, 28, 29, 34, 37, 59].

Relational and Array-based Systems. Relational systems, that buffer updates before applying them lazily to the base data, follow the out-of-place paradigm. Data stores in Vertica [44, 64], SciDB [51, 65], and TileDB [50, 67] use an in-memory storage to buffer incoming inserts, updates, and deletes out of place and apply the changes lazily to the disk. Similarly, the state-of-the-art column-store system MonetDB [38] uses an in-memory positional index for incoming data [36], SAP HANA uses a delta store per table to facilitate fast ingestion without affecting its read-optimized data layout [30]. Several research prototypes also propose using a separate delta store on faster storage (e.g., SSD/NVM) to offer efficient access to incoming data [13, 14, 26, 41, 53].

NoSQL Systems. Production-grade NoSQL key-value stores often employ the out-of-place paradigm. Key-value stores such as Facebook’s RocksDB [28, 29], LevelDB [35] and BigTable [21] at Google, X-Engine [37, 70] at Alibaba, Voldemort [45] at LinkedIn, Amazons’s DynamoDB [25], Cassandra [12], HBase [11], and Accumulo [10] at Apache, and hLSM [59] and cLSM [34] at Yahoo are based on the heavily write-optimized out-of-place data structure LSM-tree [46, 49, 71]. Other examples include B*-tree, B+-tree, and fractal tree-based storages with buffer-support, such as COLA [15], TokuDB [43], and BertFS [39].

Deletes in Out-of-Place Systems. Contrary to common perception, deletion is a frequent operation in modern relational and NoSQL data stores. For example, ZippyDB, which is a distributed key-value engine that stores metadata for images and videos, processes 25.2M delete requests over a 24-hour window – this is 6% of the entire workload [19]. Out-of-place systems treat deletes (and updates) similarly to inserts, i.e., instead of deleting entries in place, they insert a new version of the entry to be deleted, which logically invalidates the older target entries. These special entries that are responsible for logical deletes are termed delete markers [64] or tombstones [28, 56].

Logical deletion of data is an out-of-place operation by definition, and it does not necessarily persistently remove the data to be deleted. Instead, the logically invalidated entries are retained in the data store for arbitrarily long with the optimistic hope of eventual persistent deletion [56]. In fact, most out-of-place data stores are built with the underlying assumption of perpetual data retention, in order to gain more insights from the user and organizational data [69]. Logical updates are applied lazily too, however, the implications of out-of-place deletes are critical in terms of the privacy regulations, and thus, are our main focus.

The Legal Frontier. Over the past decade, there have been several government-driven endeavors across the globe aiming to protect the privacy of user data and to put back the users in the control of their personal data. On the legal side, regulations, such as the EU’s GDPR [2], California’s privacy protection acts – CCPA [3] and CPRA [6], and Virginia’s VCDPA [7] mandate data companies to ensure privacy through deletion. GDPR’s right to be forgotten, CCPA and CPRA’s right to delete, and the deletion right in VCDPA particularly focus on persistent on-demand deletion of user data in a timely manner [9, 27, 33, 40, 54, 58, 62, 68].

The Technological Roadblock. Treating deletes as first-class citizens is a new requirement for the data systems community, and it requires a significant amount of work to transform classical data systems to be delete-efficient. In fact, despite using state-of-the-art data engines, data companies face critical challenges as they attempt to demonstrate compliance with the deletion-regulations and realize efficient on-demand user data deletion [60, 61, 63].
To translate this into numbers, between January 2020 and January 2022, the penalties under GDPR paid by data companies amounted to $1.2B, which includes large contributions from companies such as Amazon ($877M), WhatsApp ($255M), Google Ireland ($102M), and Facebook ($68M) [47, 66]. To demonstrate compliance and to ensure timely persistent deletion of user data, many companies end up performing expensive database-wide consolidations periodically (e.g., every few weeks) [56, 57]. Such operations are remarkably expensive in terms of time and money, cause undesirable latency spikes, and hence, should be avoided.

**Challenge: No Application Layer Support.** The legal regulations outline the users’ rights for timely deletion of their personal data, and thereby, lay the platform for privacy through deletion (Fig. 1: Policy layer). At the other end, recent endeavors at the systems level have been opportunistic in addressing specific delete use-cases to facilitate timely persistent data deletion (Fig. 1: System layer) [24, 48, 56]. To bridge the two, we need to (i) extract the user requirements from the policy layer and (ii) address the fact that the application layer lacks the query language support to convey user deletion preferences to the underlying system.

**Identifying Deletion Requirements.** We extract from the policy layer two classes of deletion requirements (Fig. 1: Requirements layer). The first class entails deletion of user data older than a specified retention duration from across all domains of a service provider. The second class pertains to on-demand deletion of user data from the providers’ domains. Realizing retention-based deletes entails periodically invoking a deletion task that persistently deletes data older than the retention duration. On the other hand, on-demand deletion requests must be applied within a set time period based on the legal regulations (e.g., 45-60 days), persistently removing all data of a user.

**Query Language Support for Timely Deletes.** Next, we propose an extension of SQL in order to capture the new deletion requirements (Fig. 1: Application layer). We augment the SQL data definition language (DDL) to express the (i) retention duration of data (for retention-based deletes) and (ii) the threshold for persistent data deletion (for on-demand deletes) during creation of a table. Further, we extend the INSERT data manipulation language (DML) to express the specific retention durations associated with the entries ingested. We also extend the DELETE DML to express the threshold time limit within which any logical deletes must be persisted, physically removing the target data entries from the database. Based on this information from the application layer, the underlying storage engine is made aware of the user deletion requirements, which are then realized at the system level.

**Contributions.** The contributions of our work are as follows.

- We identify the two classes of user delete requirements for which there is lack of query language support.
- We augment SQL to support retention-based deletes based on either arbitrary or predefined retention durations.
- We further augment SQL to facilitate on-demand deletion based on either arbitrary or a predefined set of delete persistence thresholds.

2 BACKGROUND & MOTIVATION

Recent changes in the legal landscape of data privacy protection call for a transformation on data management and storage. More specifically, supporting the regulatory requirements for privacy through timely and persistent data deletion has become a fundamentally pressing issue for data management systems.

2.1 Regulations on Timely Data Deletion

We particularly focus on the legal policies concerning data retention and data deletion, as we aim to ensure privacy through deletion. Below, we present the different active deletion rights.

**Right to be Forgotten, EU/UK GDPR.** The General Data Protection Regulation (GDPR) has revolutionized the data privacy landscape for the EU countries and the UK [1, 2, 4, 54]. A fundamental component of the GDPR is the right to be forgotten, which empowers users with the right to request a service provider to delete their personal data persistently [1, 2]. Service providers must comply with the erasure requests within 30-60 days.

**Right to Delete, CCPA, CPRA.** The California Customer Protection Act (CCPA) and the California Privacy Rights Act (CPRA) allow the users/consumers in California to request from the service providers to permanently delete all their personal data [3, 6]. The service providers must acknowledge such a request within 10 days, and respond to it within 45 days [18]. Persistent deletion must remove the target data across all domains, barring archive and backup systems, and anonymize the data as required.
Right to Delete. VCDPA. Similarly to CCPA, the Virginia Consumer Data Protection Act (VCDPA) empowers users in Virginia to exercise their right to delete their personal data from the providers’ domain [7]. VCDPA requires the service providers to serve a delete-request from a user within 45 business days [18].

Other Efforts. Among other countries, Argentina [20, 52], Singapore [22], India [42], Canada [5], and South Korea [17] also have some implementation of the right to deletion as a part of their respective privacy protection acts.

2.2 Limitations of Query Languages
We now present two real-life scenarios to highlight the limitations of state-of-the-art query languages with respect to supporting timely and on-demand deletion of data.

Scenario 1. Alice uses a smart-home ecosystem, HomeComp, that provides real-time services including video surveillance and remote temperature and illumination control. Alice is concerned about her personal data privacy, and she wants HomeComp to delete all her data older than 30 days.

The problem? Like most service providers, HomeComp’s data model is built around the assumption of perpetual data retention; deletion of user data needs a human-in-the-loop that performs the necessary actions. Further, state-of-the-art SQL syntax does not provide to the application developers the tools to express the need for periodic data deletion. Overall, HomeComp cannot facilitate user requests for retention-based data deletion.

Scenario 2. StreamEra provides real-time insights for data streams, and allows its users to request on-demand deletion of their personal data, as it is bound by the right to be forgotten. StreamEra uses an SQL-based wrapper on top of its storage layer.

What’s missing? The above scenarios demonstrate that despite (i) legally enforcing user data retention limits and (ii) the system-level efforts to facilitate persistent and timely deletion of data, there is a missing link in the intermediate application layer. This work presents a new set of SQL extensions which expresses the policy requirements for both retention-driven deletes and on-demand data deletion, bridging the deletion policy with the system capabilities to support timely data deletion.

3 EXTENDING SQL SUPPORT
We now present in detail the application layer modifications necessary to bridge the gap between the legal framework and the system support for timely deletion. We first identify the two classes of deletion requests that the providers must facilitate in order to demonstrate regulation compliance (§3.1). Next, we introduce a new set of application layer tools by augmenting SQL to capture and transform the user-requirements for deletes (§3.2).

3.1 Types of Deletion Requests
Based on the retention regulations, we classify the user delete requests into two categories: (a) retention-driven and (b) on-demand (Fig. 1: Requirements layer).

Retention-driven deletes. Deletes of this class are rolling in nature and enable persistent deletion of user data that is older than a pre-set retention duration. The retention duration serves as the lifetime of data within a service provider’s domain. In practice, the retention duration is suggested by the service provider as part of the service level agreement (SLA), and users can specify the granularity of rolling deletes as preferred.

Deletion on-demand. The delete-regulations also allow users to submit on-demand deletion requests which entails persistent deletion of a user’s personal data. Such deletion requests can be submitted through an API provided by a service provider. Upon request, the target data is purged persistently within a threshold period, set by the regulations and as specified in the SLA.

3.2 SQL Support for Deletes
With the legal framework in place and the user preferences about deletion accounted for, we now translate the user requirements to new SQL constructs. The proposed extensions can be also integrated into other query languages, like GraphQL, DMX, LINQ, and N1QL, with simple template modifications.

Goal. The objective of the proposed SQL extension is three-fold.

1. Supporting retention-driven deletion: We first augment the CREATE TABLE and INSERT SQL statements to bind every ingestion to a specific retention duration, after which the inserted data is deleted.

2. Ensuring timely persistence of on-demand deletes: We further augment the CREATE TABLE and the DELETE SQL statements to facilitate on-demand deletion requests with timely persistence guarantees.

3. Supporting arbitrary delete thresholds: Lastly, we extend the SQL support described above for both fixed and arbitrary retention durations and delete persistence thresholds.

Augmenting SQL. Supporting timely and persistent data deletion requires augmenting both the data definition language (DDL) and the data manipulation language (DML) parts of SQL.

Enabling retention-driven deletes. To support retention-driven data deletion, we extend (i) the CREATE TABLE DDL and (ii) the INSERT DML in SQL. The CREATE TABLE statement now allows an application developer to specify the different retention durations supported as a table-property.

```sql
CREATE TABLE R (column1 type1, column2 type2, ...) WITH RET_DUR FIXED (t1 <ret1>, t2 <ret2>, ...);
```

The above SQL statement creates a table R that supports retention-based deletes with specific retention durations of ret1, ret2, etc. The WITH RET_DUR clause is optional, and is only necessary for tables that need to support deletes with predefined retention durations. When a table supports a predefined set of retention durations, each INSERT statement can use only one of those. For example, a table that is configured to support retention durations of 30 days and 60 days (CREATE TABLE R (....) WITH RET_DUR FIXED (t1 '30 days', t2 '60 days')), can only receive inserts with retention durations t1 or t2. An ingestion without a retention period explicitly mentioned is kept perpetually following the logic of a classical insert. Note that, in general, the predefined retention durations stem from the delete-SLAs that a specific application requires. Following is the syntax for inserts.

```sql
INSERT INTO R (val1, val2, ...) WITH RET_DUR t<i>;
```

Support for arbitrary retention durations. We further extend SQL to support arbitrary retention durations for deletes. To do so, we add the ARBITRARY keyword to both the CREATE TABLE and INSERT statements. Supporting arbitrary retention durations
is common in distributed frameworks that replicate data across physical data stores in different geographic locations, each bound by different legal requirements. Below, we present the full syntax for the proposed SQL extensions.

```sql
CREATE TABLE R (column1 type1, column2 type2, ...) WITH RET_DUR
   (ARBITRARY | FIXED (t1 <ret1>, t2 <ret2>, ...));
INSERT INTO R (val1, val2, ...)
   WITH RET_DUR ( <t> | t<i> ) ;
```

Note that having a pre-defined set of retention durations provides more information to the system compared to allowing arbitrary durations. As a result, it allows the system to better prepare to offer efficient retention-driven deletes.

**Enabling timely on-demand deletion.** To support on-demand data deletion in a timely manner, we introduce the notion of delete persistence threshold (DPT), which denotes the maximum delay between a logical delete and its persistence. Each table can provide support for several such user-defined thresholds. Similarly to retention-based deletes, we also extend SQL to support arbitrary DPTs when the DPTs are not specified a priori. Below, we outline the modifications to the DDL and DML necessary to support on-demand timely deletion requests.

```sql
CREATE TABLE S (column1 type1, column2 type2, ...) WITH DPT
   (ARBITRARY | FIXED (d1 <dpt1>, d2 <dpt2>, ...));
DELETE FROM S WHERE ( ... )
   WITH DPT ( <d> | d<i> );
```

Table S can support several DPTs (dpt1, dpt2, etc.), if the DPTs are specified before-hand, and applications can trigger on-demand deletion with any DPT through the DELETE command. Similarly to retention-driven deletes, timely persistent on-demand deletion is easier to handle from a storage engine if the DPTs supported are known a priori during the table creation.

**Putting everything together.** Putting the proposed DDL extensions together, a table can support multiple (pre-defined or arbitrary) thresholds for both retention-based and on-demand deletes. The complete syntax for CREATE TABLE S is as follows.

```sql
CREATE TABLE T (column1 type1, column2 type2, ...) WITH RET_DUR
   (ARBITRARY | FIXED (t1 <ret1>, t2 <ret2>, ...));
WITH DPT
   (ARBITRARY | FIXED (d1 <dpt1>, d2 <dpt2>, ...));
```

Note that retention-based deletes come from the application requirements, and on-demand deletion requests are issued by the user. Further, note that while these SQL extensions allow us to express deletion preferences, they rely on the system layer to correctly realize them.

### 4 DISCUSSION

We now briefly discuss efforts on supporting timely deletes on the systems layer and the key open challenge of demonstrating regulation compliance.

**Storage Layer Endeavors.** Realizing timely deletes without hurting performance of the underlying storage engines is critical. The efficiency of deletion depends on (i) the schema and the physical data layout, (ii) the data re-organization strategy, (iii) the workload, and (iv) the storage engine tuning.

*On-demand deletes in NoSQL engines.* Deletes issued on a different attribute (or on attributes that have no particular organization) are hard to facilitate, as they require inspection of all data objects in a data store, which is very costly. Efficient realization of such delete requests requires arranging the data on disk with some order based on the deletion attribute. In relational data stores, the records can be re-arranged on disk as re-sorted on the delete attribute, or they can be indexed based on the delete attribute to facilitate such operations [16, 32]. In NoSQL key-value stores, an inter-weaved data layout helps clustering the qualifying entries, which allows invalidating entire blocks of data at a time, facilitating efficient garbage collection [56]. The intuition here is to create a logical data collection of consecutive disk pages, within which the entries are sorted based on the delete attribute.

**Compliance.** Proving compliance is a known challenge when regulations meet technology, as this entails tracking data access patterns and execution paths within a data system. Current solutions demonstrate regulation-compliance through inspection of code, data, and legal substantiation. Another way is to be able to quickly inspect the data (essentially via querying and accessing data files) while ensuring timely garbage collection for the deleted data. However, providing system-level guarantees on timely data deletion is challenging as it entails tracking the data-flow within a system and secure data deletion at the device level. In the longer term, the community should work toward building system-tools with lightweight checks that can prove deletion compliance. This remains an open research challenge in the systems and database community.

### 5 CONCLUSION

In this paper, we point out that state-of-the-art query languages lack the necessary tools to express new legally mandated requirements for user data deletion. Toward this, we identify the two classes of deletion requirements that need to be supported. We then identify the missing links at the application layer and present the modifications made to the SQL DDL and DML to facilitate both retention-based and on-demand user data deletion. Finally, we discuss how the proposed query language extensions work hand-in-hand with legal regulations and system solutions for persistent data deletion to ensure privacy through deletion.

**Acknowledgments.** This work was partially funded by NSF Grant No. IIS-1850202 and a Facebook Faculty Research Award.


