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-bounds 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 bLSM [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^ϵ-tree, and fractal tree-based storages with buffer-support, such as COLA [15], TokuDB [43], and BεrtFS [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].

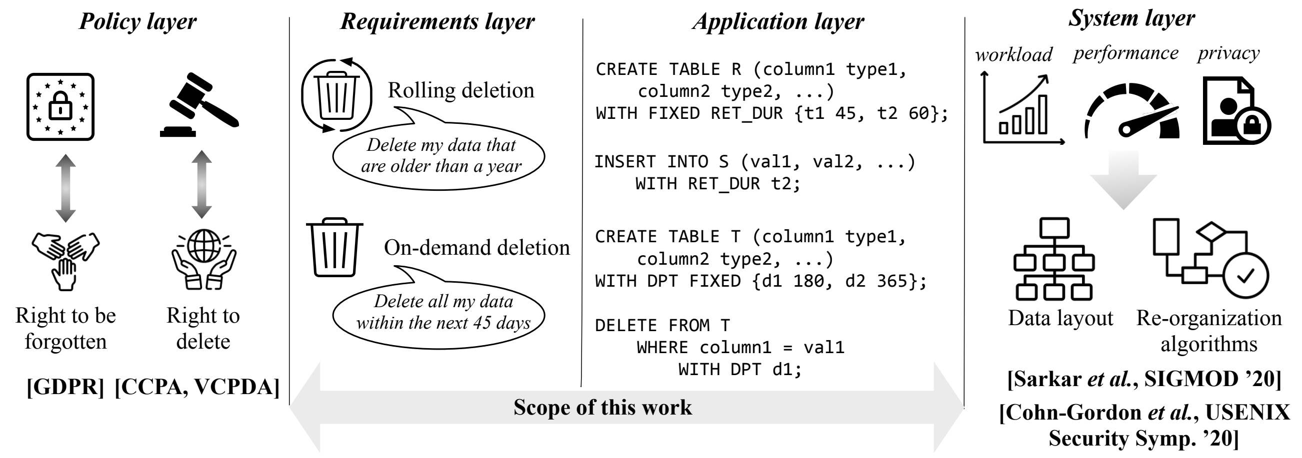

Fig. 1: Enforcing privacy regulations requires a bridge between their requirements and the necessary system support via an application layer augmented with richer deletion specification.

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.

The problem? While *StreamEra* wants to serve its users by ensuring timely persistent deletion of their personal data, SQL does not provide support for such an operation. The backend developers at *StreamEra* are expected to implement the required functionality at the application level as it is not native to SQL.

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.

```
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.

```
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 regulatory requirements. Below, we present the full syntax for the proposed SQL extensions.

```
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.

```
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 is as follows.

```
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 the attribute based on which the data is organized (sorted, hashed, or range partitioned) are generally realized logically by invalidating prior entries. To persist such logical deletes in a timely manner, prior work proposes a data layout reorganization policy that enables LSM-based key-value stores to persist logical deletes within a given threshold through a process called compaction [56]. The work introduces a family of deletion-aware compaction strategies that prioritizes compaction of files based on the delete persistence threshold, and thus, honors the requirement for timely purging of deleted data. Piggybacking deletion with the process of data layout reorganization reduces the cost of realizing deletes while ensuring timely delete persistence without hurting performance.

Deletes in online social networks. Cohn-Gordon *et al.* proposed a deletion framework *DELF* that ensures reliable data deletion from an online social network (OSN) [24]. *DELF* enables detection of inconsistent data deletion in OSNs and also facilitates data recovery in cases where user data was incorrectly deleted. Minaei *et al.* [48] proposed a framework for persistently deleting all instances of a user's data in presence of observers, ensuring privacy through timely content concealment and removal.

Deletes on Secondary Attribute. 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 light-weight 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.

REFERENCES

[1] Right to erasure. *https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/individual-rights/right-to-erasure/*.

[2] Regulation (EU) 2016/679 of the European Parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC. *Official Journal of the European Union (Legislative Acts)*, pages L119/1 – L119/88, 2016.

[3] California Consumer Privacy Act. *Assembly Bill No. 375, Chapter 55*, 2018.

[4] Data Protection Act 2018. *https://www.legislation.gov.uk/ukpga/2018/12/pdfs/ukpga_20180012_en.pdf*, 2018.

[5] PIPEDA in brief. *https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/pipeda_brief/*, 2019.

[6] The California Privacy Rights Act of 2020. *https://thecpra.org/*, 2020.

[7] Virginia Consumer Data Protection Act. *https://www.sullcrom.com/files/upload/SC-Publication-Virginia-Second-State-Enact-Privacy-Legislation.pdf*, 2021.

[8] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. R. Borkar, Y. Bu, M. J. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, E. Gabrielova, R. Grover, Z. Heilbron, Y.-S. Kim, C. Li, G. Li, J. M. Ok, N. Onose, P. Pirzadeh, V. J. Tsotras, R. Vernica, J. Wen, and T. Westmann. AsterixDB: A Scalable, Open Source BDMS. *Proceedings of the VLDB Endowment*, 7(14):1905–1916, 2014.

[9] M. L. Ambrose and J. Ausloos. The Right to Be Forgotten Across the Pond. *Journal of Information Policy*, 3:1–23, 2013.

[10] Apache. Accumulo. *https://accumulo.apache.org/*.

[11] Apache. HBase. *http://hbase.apache.org/*.

[12] Apache. Cassandra. *http://cassandra.apache.org*, 2021.

[13] M. Athanassoulis, S. Chen, A. Ailamaki, P. B. Gibbons, and R. Stoica. MaSM: Efficient Online Updates in Data Warehouses. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 865–876, 2011.

[14] M. Athanassoulis, S. Chen, A. Ailamaki, P. B. Gibbons, and R. Stoica. Online Updates on Data Warehouses via Judicious Use of Solid-State Storage. *ACM Transactions on Database Systems (TODS)*, 40(1), 2015.

[15] M. A. Bender, E. D. Demaine, and M. Farach-Colton. Cache-Oblivious B-Trees. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 399–409, 2000.

[16] B. Bhattacharjee, T. Malkemus, S. Lau, S. Mckeough, J.-A. Kirton, R. V. Boeschoten, and J. Kennedy. Efficient Bulk Deletes for Multi Dimensionally Clustered Tables in DB2. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1197–1206, 2007.

[17] C. T. Brown and T. D. Manoranjan. South Korea Releases Guidance on Right to Be Forgotten. *https://www.lexology.com/library/detail.aspx?g=21be3837-0c43-4047-b8b5-9e863960b0b9*, 2016.

[18] G. A. Brown. Consumers' "Right to Delete" under US State Privacy Laws. *https://www.securityprivacybytes.com/2021/03/consumers-right-to-delete-under-us-state-privacy-laws/*, 2021.

[19] Z. Cao, S. Dong, S. Vemuri, and D. H. C. Du. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 209–223, 2020.

[20] E. L. Carter. Argentina's Right to be Forgotten. *Emory International Law Review*, 27(1), 2013.

[21] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 205–218, 2006.

[22] W. B. Chik. The Singapore Personal Data Protection Act and an assessment of future trends in data privacy reform. *Comput. Law Secur. Rev.*, 29(5):554–575, 2013.

[23] Cisco. Cisco Global Cloud Index: Forecast and Methodology, 2016–2021. *White Paper*, 2018.

[24] K. Cohn-Gordon, G. Damaskinos, D. Neto, J. Cordova, B. Reitz, B. Strahs, D. Obenshain, P. Pearce, I. Papagiannis, and A. Media. DELF: Safeguarding deletion correctness in Online Social Networks. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, 2020.

[25] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's Highly Available Key-value Store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[26] F. Deng, Q. Cao, S. Wang, S. Liu, J. Yao, Y. Dong, and P. Yang. SeRW: Adaptively Separating Read and Write upon SSDs of Hybrid Storage Server in Clouds. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 76:1—-76:11, 2020.

[27] A. Deshpande and A. Machanavajjhala. ACM SIGMOD Blog: Privacy Challenges in the Post-GDPR World: A Data Management Perspective. *http://wp.sigmod.org/?p=2554*, 2018.

[28] S. Dong, M. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum. Optimizing Space Amplification in RocksDB. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017.

[29] Facebook. RocksDB. *https://github.com/facebook/rocksdb*, 2021.

[30] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA Database – An Architecture Overview. *IEEE Data Engineering Bulletin*, 35(1):28–33, 2012.

[31] Gartner. Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. https://tinyurl.com/Gartner2020, 2017.

[32] A. Gärtner, A. Kemper, D. Kossmann, and B. Zeller. Efficient Bulk Deletes in Relational Databases. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 183–192, 2001.

[33] M. Goddard. The EU General Data Protection Regulation (GDPR): European Regulation that has a Global Impact. *International Journal of Market Research*, 59(6):703–705, 2017.

[34] G. Golan-Gueta, E. Bortnikov, E. Hillel, and I. Keidar. Scaling Concurrent Log-Structured Data Stores. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, pages 32:1–32:14, 2015.

[35] Google. LevelDB. *https://github.com/google/leveldb/*, 2021.

[36] S. Héman, M. Zukowski, and N. J. Nes. Positional Update Handling in Column Stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 543–554, 2010.

[37] G. Huang, X. Cheng, J. Wang, Y. Wang, D. He, T. Zhang, F. Li, S. Wang, W. Cao, and Q. Li. X-Engine: An Optimized Storage Engine for Large-scale E-commerce Transaction Processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 651–665, 2019.

[38] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. MonetDB: Two Decades of Research in Column-oriented Database Architectures. *IEEE Data Engineering Bulletin*, 35(1):40–45, 2012.

[39] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, M. A. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter. BetrFS: A Right-optimized Write-optimized File System. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pages 301–315, 2015.

[40] M. L. Jones. It's About Time: Privacy, Information Lifecycles, and the Right to Be Forgotten. *Stanford Technology Law Review*, 16(2):54, 2012.

[41] W.-H. Kang, S.-W. Lee, and B. Moon. Flash as cache extension for online transactional workloads. *The VLDB Journal*, 25(5):673–694, 2016.

[42] P. Kittane, I. S. Charles, A. Kamath, and G. Gokhale. Privacy and Data Protection – India Wrap 2020. *The National Law Review*, XI(162), 2021.

[43] B. C. Kuszmaul. A Comparison of Fractal Trees to Log-Structured Merge (LSM) Trees. *Tokutek White Paper*, 2014.

[44] A. Lamb, M. Fuller, and R. Varadarajan. The Vertica Analytic Database: C-Store 7 Years Later. *Proceedings of the VLDB Endowment*, 5(12):1790–1801, 2012.

[45] LinkedIn. Voldemort. *http://www.project-voldemort.com*.

[46] C. Luo and M. J. Carey. LSM-based Storage Techniques: A Survey. *The VLDB Journal*, 29(1):393–418, 2020.

[47] R. McKean, E. Kurowska-Tober, and H. Waem. DLA Piper GDPR fines and data breach survey: January 2022. *https://www.dlapiper.com/en/us/insights/publications/2022/1/dla-piper-gdpr-fines-and-data-breach-survey-2022/*, 2022.

[48] M. Minaei, M. Mondal, P. Loiseau, K. P. Gummadi, and A. Kate. Lethe: Conceal Content Deletion from Persistent Observers. *Proceedings on Privacy Enhancing Technologies (PoPET)*, 2019(1):206–226, 2019.

[49] P. E. O'Neil, E. Cheng, D. Gawlick, and E. J. O'Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.

[50] S. Papadopoulos, K. Datta, S. Madden, and T. Mattson. The TileDB Array Data Storage Manager. *Proceedings of the VLDB Endowment*, 10(4):349–360, 2016.

[51] Paradigm4. Online reference. *https://www.paradigm4.com/*.

[52] D. Pardo. First Decision on the "Right to be Forgotten" in Argentina. *https://scholarlycommons.law.emory.edu/cgi/viewcontent.cgi?article=1097&context=eilr*, 2020.

[53] M. Sadoghi, K. A. Ross, M. Canim, and B. Bhattacharjee. Exploiting SSDs in operational multiversion databases. *The VLDB Journal*, 25(5):651–672, 2016.

[54] S. Sarkar, J.-P. Banâtre, L. Rilling, and C. Morin. Towards Enforcement of the EU GDPR: Enabling Data Erasure. In *Proceedings of the IEEE International Conference of Internet of Things (iThings)*, pages 1–8, 2018.

[55] S. Sarkar, S. Chatterjee, and S. Misra. Assessment of the Suitability of Fog Computing in the Context of Internet of Things. *IEEE Transactions on Cloud Computing (TCC)*, 6(1):46–59, 2018.

[56] S. Sarkar, T. I. Papon, D. Staratzis, and M. Athanassoulis. Lethe: A Tunable Delete-Aware LSM Engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 893–908, 2020.

[57] S. Sarkar, D. Staratzis, Z. Zhu, and M. Athanassoulis. Constructing and Analyzing the LSM Compaction Design Space. *Proceedings of the VLDB Endowment*, 14(11):2216–2229, 2021.

[58] M. Schwarzkopf, E. Kohler, M. F. Kaashoek, and R. T. Morris. Position: GDPR Compliance by Construction. In *Selected Papers from VLDB Workshop on Polystore Systems for Heterogeneous Data in Multiple Databases with Privacy and Security Assurances (POLY)*, volume 11721 of *Lecture Notes in Computer Science*, pages 39–53, 2019.

[59] R. Sears and R. Ramakrishnan. bLSM: A General Purpose Log Structured Merge Tree. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 217–228, 2012.

[60] A. Shah, V. Banakar, S. Shastri, M. Wasserman, and V. Chidambaram. Analyzing the Impact of GDPR on Storage Systems. In *Proceedings of the USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*, 2019.

[61] S. Shastri, V. Banakar, M. Wasserman, A. Kumar, and V. Chidambaram. Understanding and Benchmarking the Impact of GDPR on Database Systems.

Proceedings of the VLDB Endowment, 13(7):1064–1077, 2020.

[62] S. Shastri, M. Wasserman, and V. Chidambaram. The Seven Sins of Personal-Data Processing Systems under GDPR. In *Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[63] S. Shastri, M. Wasserman, and V. Chidambaram. GDPR anti-patterns. *Communications of the ACM*, 64(2):59–65, 2021.

[64] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. R. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. Zdonik. C-Store: A Column-oriented DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 553–564, 2005.

[65] M. Stonebraker, J. Duggan, L. Battle, and O. Papaemmanouil. SciDB DBMS Research at M.I.T. *IEEE Data Eng. Bull.*, 36(4):21–30, 2013.

[66] Tessian. 25 Biggest GDPR Fines So Far (2019, 2020, 2021, 2022). *https://www.tessian.com/blog/biggest-gdpr-fines-2020/*, 2022.

[67] TileDB. Online reference. *https://tiledb.io*.

[68] A. Tsesis. The Right to be Forgotten and Erasure: Privacy, Data Brokers, and the Indefinite Retention of Data. *Wake Forest Law Review*, 48:51, 2014.

[69] Z. Whittaker and N. Lomas. Even years later, Twitter doesn't delete your direct messages. *https://techcrunch.com/2019/02/15/twitter-direct-messages/*, 2019.

[70] L. Yang, H. Wu, T. Zhang, X. Cheng, F. Li, L. Zou, Y. Wang, R. Chen, J. Wang, and G. Huang. Leaper: A Learned Prefetcher for Cache Invalidation in LSM-tree based Storage Engines. *Proceedings of the VLDB Endowment*, 13(11):1976–1989, 2020.

[71] Q. Zheng, C. D. Cranor, D. Guo, G. R. Ganger, G. Amvrosiadis, G. A. Gibson, B. W. Settlemyer, G. Grider, and F. Guo. Scaling Embedded In-Situ Indexing with DeltaFS. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 3:1—-3:15, 2018.