The Impact of Privacy Regulations on DB Systems

Javam C. Machado, Paulo R. P. Amora

Computer Science Department
Federal Univesity of Ceará, Fortaleza, Brazil
{javam.machado, paulo.amora}@lsbd.ufc.br

Abstract.

Personal data usage and collection are activities that used to grow unrestricted. However, several laws in the physical world ensure rights to people regarding their privacy and information usage. In the last years, legislators passed many laws, regulations, and acts to replicate these rights to the digital world. By doing so, new constraints, rights, and duties appear on every component of the data usage and collection workflow. In this paper, we discuss legislations' implications, identifying impacts that these regulations introduce to current DBMS, and survey recent works that aim to solve the problems raised by these impacts, highlighting research opportunities and identifying how solutions can be achieved for the problems.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems; H.3.2 [**Information Storage and Retrieval**]: Information Storage

Keywords: databases, GDPR, purpose, data access

1. INTRODUCTION

Lately, data privacy regulations rule over sensitive personal information in many countries. European Union has the General Data Protection Regulation (GDPR). In the late 1990s, Canada approved the Personal Information Protection and Electronic Documents Act (PIPEDA). Several USA states have similar regulations, like the California Consumer Privacy Act (CCPA). In Brazil, the *Lei Geral de Proteção de Dados (LGPD)* comes into effect year 2020. In general, these regulations protect individuals' data stored in organizations, giving the individual control over how their data is shared and processed. They usually focus on data security, defining responsibilities organizations must have over personal data. Moreover, they restrict individuals' data usage to the purpose for which the owner has given authorization. Many applications collect personal data, such as mobile applications, e-commerce, social networks, and any transactional system where users are involved. For instance, the coronavirus pandemic led several countries, cities, and health organizations to develop mobile applications that collect personal contacts based on geolocation data. Although this initiative is of great importance for controlling the spread of the virus in a community, it is clear that this type of data might be very sensitive for an individual outside of this purpose.

Database systems (DBMS) are the primary tools organizations use to store and manage their data, including sensitive personal information. Mining data and sharing information between partners are both heavily based on data directly provided by DBMS. There is no doubt that personal data is stored, processed, and shared within organizations, among other transactions' data. Therefore, DBMSs have to provide capabilities that allow organizations to comply with the regulations. That involves at least five concepts:

This research was supported by LSBD/UFC.

—To identify personal data, which includes, automatically or in an assisted manner, conceptually separate sensitive data from other transactional data;
—To manage metadata about processing and sharing personal data as efficiently as possible since this type of data grows quickly in this new context. Moreover, metadata is more likely to be queried to access control and individual privacy;
—To give the user the correct tools for declaring personal data visibility and usage, considering both the owner and the data administrator;
—To provide efficient auditing interfaces for data usage that goes far beyond regular log files;
—To sanitize personal data before publishing or sharing among partners respecting data utility for analysis and to the partners.

In this paper, which is an extension of Machado et al. [Machado and Amora 2020], we investigate the impact of managing sensitive personal data on DBMS. We aim to address the main features that have to be reviewed at the core level of these systems to allow data controllers and processors to be compliant with privacy regulations. We enumerate law requirements that have to be met by DBMS when they store personal information. We survey and discuss several works that address the impacts of these requirements. We also identify several open issues for research opportunities.

1.1 Policy Requirements

From the concepts stated above, the regulations, acts, and laws have many points of intersection with regard to users' rights and data controllers' and processors' duties. To discuss these rights and obligations, we will take GDPR [General Data Protection Regulation 2016] as an example. GDPR is composed of 99 articles, ranging from data privacy, protection, cryptography as well as users' own right to access collected personal data, know what it will be used for as well as request removal of said data. As GDPR is an extensive regulation, we highlight five rights to focus our discussion:

Right of access - Article 15 states that the data subject can obtain confirmation from the controller as to whether their data is being processed and access to: data itself, purposes, categories of personal data, recipients of this data, the period of storage, usage in automated decision-making. This right enables customers to have property rights over their provided data. To make informed decisions and properly care for provided information, customers need to know how their data is stored, what it is being used on, for how long and where this data is stored, as well as if this data is used on automated decision-making. Through this right, users can request their data from data holders, check if other applications/companies are using their data accordingly and awareness if this data is being used on a black-box setting, e.g., machine learning. Aside from allowing data portability, in case the customer does not agree or trust the data holder, this right also impacts whether the data holder should use underlying patterns discovered through automated algorithms.

Right to be informed - Article 12 states that the data subject must be informed in a concise, transparent, and easily accessible form if their data is obtained or not, if data will be used in automated individual decision-making, if data about this subject is breached, as well as their capabilities of data portability and objection. This imposes restrictions on data holders and data controllers, as they must be able to track data owners and inform them of their uses and purposes. The customer must be clearly informed, and their decisions must be respected by the data holder and whoever uses the data. In case of a data breach, the data holder is responsible for contacting customers, make them aware of the breach and take appropriate measures to contain the breach, and ensure that the leaked data is useless.

Right to be forgotten - Article 17 states that the data subject has the right to obtain the erasure of personal data without undue delay. This covers not only active requests but also covers collected data that is no longer necessary, unlawfully processed. The controller must also inform other controllers of the erasure request so that appropriate measures are taken. The implications of this right are twofold:

First, all data related to a customer must be accessible for deletion, therefore, this data and any copies made must be trackable. Secondly, this data must not be kept for a longer period of time than necessary. This requires special mechanisms that ensure that data is effectively removed from storage, be it by irrecoverable erasure, or complete inaccessibility, for example, erasing a cryptography key so that the information is completely lost.

Consent - Article 6 states that one of the conditions to make data processing lawful is that the data subject consent to the processing directed to one or more specific purposes. Consent is defined by article 4 as a freely given, informed, and unambiguous agreement, therefore, consented data collection may not be misused under a different purpose. Data holders and processors must be aware of the customers' choices and permissions to not violate them, as these violations may incur huge fines. This consent must be clearly stated in the terms and conditions. Consent mixes with purpose-based access in terms of restricting access to data, but it goes wider than that.

Singling out - Article 4 defines personal data as "any information relating to an identified or identifiable natural person; an identifiable natural person is one who can be identified, directly or indirectly". Following Recital 26, singling out is a way of identifying a natural person in a database. Therefore it is expected that the database system does not provide for sharing or publishing any data that would allow singling out a person. That is particularly true for any user who declares himself as "opted out" whenever he declines to allow the data holder to use his data differently from the original purpose. Opting out must ensure that the individual is not uniquely identified, even considering semi-identifiers, data that does not single out any customers, however, can be crossed and, after some processing, uniquely identify customers.

2. IMPACTS ON DB SYSTEMS

In this section we discuss six components necessary to achieve compliance, what are the responsibilities of the DBMS and storage engine, interweave related work to each component, and present research opportunities in each component.

2.1 Metadata explosion

This requirement relates to all rights described before given each right must have its own data to be guaranteed. It refers to the amount of metadata needed to comply with the regulation, that it will grow considerably, even exponentially, as new data is stored. For instance, additional information regarding user consent and purpose access, as well as auxiliary data structures to permit data tracking related to a user. The storage engine must be aware of these metadata, if it will be stored together with data or accessible in a different way. Furthermore, the DBMS must guarantee that the overhead of storing this extra data and updating it does not severely impact performance. The implications are twofold: Storage technology must be able to store this data without allowing an exponential growth of used space and access must be efficient, since this information must be available and up-to-date, for it to be used accordingly to the regulation.

GDPRBench [Shastri et al. 2020] is a benchmark suite built upon YCSB and proposes to evaluate the impacts on performance of making a DBMS GDPR compliant. Without any major changes, aiming only to evaluate the impact of fulfilling the requirements, modifications were made to Redis, a key-value storage and PostgreSQL, a relational DBMS. The work raises the following concerns with respect to metadata explosion, the ability of systems to protect data by design and support GDPR queries. The benchmark suite presents four types of profiles (Customer, Controller, Processor, and Regulator) and seven types of queries (create, delete_by, read_data_by, read_metadata_by, update_data_by, update_metadata_by, and get_system) concerning the participant entities. Each query is modified to comply with GDPR, for example, a *create* query not only inserts data into the table, but also updates all the related metadata. The work finds that even minor modifications to

comply with GDPR cause an increase of 3.5 times more data stored and this number rises to 5.95 times when secondary indexes are added to this metadata while decreasing performance considerably. Then, it concludes that even minor modifications greatly impact storage systems, either a full-fledged DBMS as well as a simple key-value storage engine. Although PostgreSQL has several mechanisms to speed up queries, these mechanisms backfire when met with the increased metadata volume. While Redis allows for simple querying, the performance also decreases when some rules are added, such as time-to-live on inserted data, which sets an expiration timestamp for a tuple, used when evaluating what data are still valid.

[Kraska et al. 2019] describe an end-to-end system, from application requirements to the DBMS, as well as a purpose-based data access model, in which purpose filters act between query execution and retrieval to avoid data leaking to unwanted purposes. These filters are added through bit vectors associated with each tuple. Each bit relates to a purpose, and is set if a query with this purpose is allowed to access this tuple. It also discusses the execution model and the amount of overhead space required to store this metadata, as well as suggest an encoding mechanism that allows filter compression. For the DBMS, it proposes a more concrete approach to the current relational model. Instead of using keys to represent relationships, auxiliary tables must be used to represent the relationships, which store surrogate keys associated with registers. These surrogate keys must not be visible outside the DBMS and be used only for traceability. The bit vectors used by purpose filters are estimated to have an overhead proportional to the number of tuples multiplied by purposes, although these can be compressed if there is a sense of hierarchy between purposes, through algorithms similar to Huffman encoding, for example.

GDPR Compliance by construction [Schwarzkopf et al. 2019] suggests that all user data is stored in user shards. The shards are inaccessible for query, instead, materialized views based on the query and the purposes associated would be produced to provide the data while hiding the true data and allowing the user to request, remove, or revoke access. Aside from tracking data of the materialized views and what data is in them, the materialized views by themselves impose a massive overhead, because user data is copied to them, and multiple copies of the same data are allowed. This falls back to the widely known view maintenance problem. The work requires an efficient mechanism for view creation and destruction, and does not discuss the impact on storage space required by the existence of many materialized views, with reasonable overlapping of data, as well as the impact of tracking these copies, for when a user requires access/deletion. Figure 1 exemplifies this process.

Sypse [Deshpande 2021] proposes a stateless layer that coordinates access between different partitioned data, and manages data going in and out of these partitions. Sypse partitions data between two or more partitions, including pseudonymized identifiers and synthetic data to put the database in a private state. This inclusion of data also concern metadata explosion, because once more partitions are designed, more synthetic data will be added to this data. The work discusses three possible forms of creating these partitions, using synthetic keys, adding an encryption key to personal data, and using encryption with pseudorandom keys.

The aforementioned works rely on increasing metadata, be it embedded directly into data or using auxiliary data structures. Opportunities remain in data structures or mechanisms that provide compact representation of this metadata, allowing for fast retrieval and modification. There is a trade-off between storage space and access speed, however, it is possible to achieve a sweet spot on these points, although these problems can be tackled in a separate manner, be it by using a succinct structure to compact data and save storage space, or adding more data to perform a more efficient access.

2.2 Delete guarantees

This requirement relates to the right to be forgotten and brings the problem of ensuring that the requested deletes happen without undue delay. This means that deletes must be followed through, instead of them being a promise, as well as if such promise is made, that it is tracked, to be fulfilled in

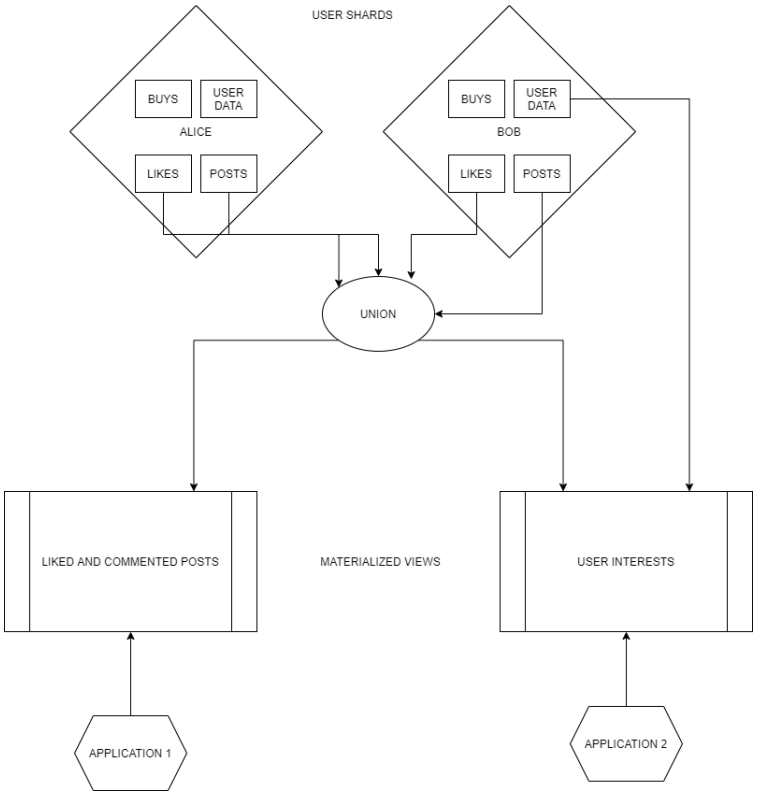

Fig. 1. Querying user shard data through materialized views (Adapted from [Schwarzkopf et. al. 2019]).

a given time. The storage engine must provide either a confirmation mechanism or have in its design the deletion guarantees, and the DBMS must guarantee that the deletes happen in able time, through management mechanisms handling the delete confirmation and/or enforcement. Another problem that arises is that whenever users request their data to be removed, this data might be distributed in several tables/databases within the same application. The delete operation must be able to access different representations of the same data. This problem is known as the entity resolution problem, and, while explored in academia, the main focus is to access scattered data or reconstruct an entity, not to remove this data and deal with the aftermath of separate data being selectively removed. On this topic, the recovery log is another place that may contain data related to a user, bringing up challenges related to the immutability of the log vs the erasure guarantees.

Lethe [Sarkar et al. 2020] is an LSM-based storage engine that upgrades delete operations to first-class citizens, without loss to other operations, such as reads and inserts. Lethe is based on two other mechanisms: Fast Deletion (FADE) and Key Weaving Storage Layout (KiWi). FADE works by triggering tree level compactions over time, instead of on demand, whenever nodes need to be split. Then, with each compaction, delete tombstones are pushed down until data is completely erased from the tree. FADE works based on an estimated time-to-live, which varies by tree level, meaning that lower levels will take longer to trigger the compaction, and ensuring an upper-bound for delete persistence. KiWi is a storage layout that organizes LSM data not only based on a sort key, using also a delete key. Files contain delete tiles, which are sorted by sort key. Inside delete tiles, pages are sorted by delete key. Inside each page, records are organized by sort key. The main idea behind this storage layout is to ensure fast access to data based on the delete key, namely the timestamp used as a time-to-live by FADE. KiWi also uses Bloom Filters and Fence Pointers to filter out non-existing entries. Lethe unifies FADE and KiWi, tuning the storage layout to find the optimal value for delete

tiles' granularity.

GDPR Compliance by construction [Schwarzkopf et al. 2019] provides the removal of a user shard as a solution for the delete guarantee, however, while it highlights that the underlying views must be aware of this removal, it does not discuss a mechanism ensuring this awareness. GDPRBench [Shastri et al. 2020] evaluates this impact by adding a time-to-live attribute to data, and using a daemon or adding this functionality to the DBMS to periodically probe the data in search of TTL violations, as well as deleting them when needed. [Kraska et al. 2019] point out that when an entry is deleted in their system, the surrogate keys become dangling, rendering data access impossible. Sypse [Deshpande 2021] proposes the deletion of synthetic data as a form of removal, since it would be impossible to decrypt keys.

Research opportunities remain in approaches that also treat deletes as a first-class citizen, as well as approaches similar to garbage collection, in an active/passive hybrid approach, to guarantee deletes on time as well as be opportunistic with relation to the resources. Another opportunity presents in ensuring recovery consistency when the log is also considered a sensitive location with respect to data.

2.3 Efficient auditing

This requirement relates to the ability of auditing and ensuring the rights are being respected. It brings the problem of making sure that all data accesses are properly recorded and it is possible to retrieve all accesses to a given user/register. The storage engine must be able to record each access without impairing overall performance, storing not only the access, but what was accessed, who accessed it and how it was accessed, and the DBMS must guarantee that this log is accessible in an efficient manner, through specialized and indexed logs.

Efficient access to logs has been widely explored in academia, and one example is Instant Recovery [Graefe et al. 2016], which proposes a structure to facilitate random access to the log, without hindering log write performance. The work describes a log structure geared towards single-page recovery, aided by two data structures, an index that finds a given page in a reduced time, and a transaction log chain, which links records related to a single transaction across pages. Through these structures, it is possible to restore the database to a partial state that is sufficient to answer queries, avoiding the high downtime required for complete recovery.

Solutions for auditing databases exist for quite some time, examples are Oracle Audit Vault [Oracle Inc. 2021] and pgAudit [Riggs et al. 2021]. However, these solutions provide a more general auditing mechanism and are placed outside of the database, meaning that they do not consider accesses from within the database, e.g., query processing access.

Audits can become more efficient by using either a recovery log or a more specialized log entity for audit if these records are easily searchable. For instance, a logging entity that pertains only to personal data instead of all data, with a higher level of detail.

A research opportunity presents in using such ideas in a more modern logging scheme, such as [Haubenschild et al. 2020], which low-latency logging and bounded recovery features can be exploited in this context, allowing for fast recovery of important data. Another area to be explored is anomaly detection through log analysis, in which possible violations can be detected.

2.4 Purpose-based access

This requirement relates to right of access, right to be informed, and consent, bringing the problem of only allowing data associated with a given purpose to be queried within this purpose. The storage engine must associate each data item with its purpose metadata, as well as return the data accordingly. The DBMS must guarantee that queries have well-defined purposes and that queries do not access unwanted data through access filters.

This problem has been relevant for some time, having works dedicated to it for at least 15 years. While the driving factor of previous works was mainly information security, there was no guideline on how this type of access should be made. Nowadays, this problem is tackled with the data regulations taken into consideration.

Rizvi et al. [Rizvi et al. 2004] describe an authorization model, in which queries posed against the data are rewritten to comply with the querier's allowed privileges. This is done through a set of authorization views, which are queried instead of the raw data. These queries are rewritten to be compatible with the views, but only if it is valid to be executed. Validity is checked through the existence of compatible authorization views for the query to be executed, which can be conditional, based on the current state of data.

Byun et al. [Byun and Li 2008] treat purposes as hierarchical entities, establishing operations to transition between purposes, providing a notion of intended purpose. Based on this model, the work discusses how to access data using purposes, and what additional information is required in place for the model to work.

Other existing works deal with that by associating purposes to tuples as bit-vectors [Kraska et al. 2019] and creating purpose filters to avoid unwanted data leaking. In an application sense, it also describes a "sandbox" mechanism in which the DBMS would only be accessible by applications through specific VMs, each with their own purpose. Meanwhile, there are several advances in filters.

Sieve [Pappachan et al. 2020] presents a middleware aimed to scale up purpose-based query processing. Sieve does not require internal code changes to the DBMS, relying only on a query rewriting layer and User Defined Functions to provide guidance in creating and selecting query guards. These guards guarantee that queriers only access data within their capabilities. By modeling the querier and policies, which are obtained from a table in the database, Sieve rewrites queries and whenever possible, force the DBMS to use indexed structures to calculate and generate guards. A cost function is applied to select the best guard for each query, and finally the query is rewritten using the selected parameters. Through this approach, Sieve attains a very low response time in comparison to baselines, and this performance gain remain even with larger datasets.

HOPE [Zhang et al. 2020] proposes a key compression mechanism for in-memory search trees, then, it can be used to query filters and indexes in an efficient and space-saving manner. HOPE relies on a String Axis model, in which all possible source strings are organized in lexicographical order. Then, each string is mapped on this string axis using different intervals. From this model, six different compression schemes are derived. HOPE uses these compression schemes to encode in-memory search structures, in order to reduce storage size and not hinder the order-preserving capabilities of these structures. Therefore, it is feasible to have auxiliary structures to filter access by purpose if using a technique that reduces their storing size.

Stacked Filters [Deeds et al. 2021] provide a constructed structure able to incorporate workload knowledge within itself, without being a learned filter. It works by layering filters one on top of the other, alternating the layers between positive and negative. Whenever an element is queried, it is tested against the topmost layer of the filter, which is positive. If it is accepted, it is retested by the underlying layer, which is negative. If it is accepted by the positive and rejected by the negative, it means that the total result is positive. If not, the element is tested against lower layers until the filter is finished. Each underlying layer is smaller than the upper ones, ensuring that the filter is constructed within an expected size boundary. The strategy also allows for flexibility based on the underlying filters. Stacked Filters can also be built incrementally, starting from the first layer and constructing bottom ones as false positives start to occur in upper layers.

Current DBMSs have mechanisms to enforce such rules through a series of constraints and triggers. Denial Constraints are an evolution of these concepts, being able to model functional dependencies and more complex relationships. [Pena et al. 2020] describe VioFinder, a mechanism aimed towards

speeding up the processing of these constraints to find violations, through a process of clustering and refinement. These constraints and their violations may be modeled to represent unauthorized purpose-based access.

Research opportunities remain in improvements of storing purpose data efficiently and evaluating queries against purposes in a fast way. Designs based on specialized hardware can aid this evaluation, such as on-disk processing [Mishra and Somani 2017], allowing only the data transfer of appropriate data. Further integration of the mechanisms discussed in query execution engines can also be explored.

2.5 Opted data

This requirement relates to right of access, singling out, and consent, bringing the problem of storing data, but filtering its access to only some types of queries. The storage engine must filter out all results that are opted out, however, in aggregation queries, these results must be computed, and the DBMS must guarantee that query results do not allow users to be singled out, as well as disallow single target purposed queries in opted-out data.

In GDPR, the notion of singling out is detailed in an additional document of the European Data Protection Board, further describing anonymization as a mechanism to ensure security against individual re-identification in a data release [A29WPT 2018]. The document analyses whether specific classes of anonymization techniques, like k-anonymity, l-diversity, differential privacy, and tokenization, provide security against singling out [Cohen and Nissim 2020]. There exist approaches to implement some of those techniques into database systems, either on the storage manager or at the query language processing level. The general idea is to anonymize personal identification data or to sanitize query answers before delivering results.

K-anonymity, l-diversity, and tokenization are commonly referred to as syntactic models [Sweeney 2002; Machanavajjhala et al. 2007; Domingo-Ferrer et al. 2016]. They apply data transformation based on generalization, suppression, or perturbation to make obscure property values that otherwise would be easier to single out an individual on a dataset [Fung et al. 2010]. Although applying syntactic models is insufficient to guarantee strong individual privacy, recently several database systems have provided tools or extensions to anonymize data according to these models.

Azure SQL Database provides a feature, called Dynamic Data Masking, that hides the sensitive data in a query result for designated table attributes, while the data in the database is not changed [Microsoft 2021]. Azure SQL implements masking functions that the database administrator can use to control personal data exposure. Instead of returning raw data, masking functions generalize attribute values that will compose the query result set. This technique can be used to hide individual identifiers and to anonymize sensitive data stored in the database. MariaDB also offers similar feature with its masking filter [MariaDB 2019].

PostgreSQL Anonymizer [Dalibo 2020] is a declarative approach for masking sensitive personal data in PostgreSQL databases. Masking rules are declared for relational tables using PostgreSQL data definition language. They associate data with masking functions which can be used for randomization, faking, scrambling, and noise addition. Besides, the anonymizer extension also allows for the generalization of attribute values.

Distinctively from the syntactic models, differential privacy adds noise to query answers according to a probability distribution associated with the domain of possible query results [Dwork 2006]. With that random procedure, differential privacy aims to introduce incertitude within the answer, although much of the original data distribution is kept, allowing for data analysis. Differential privacy, which is commonly implemented as a randomized mechanism, has strong privacy guarantees. With much success, it has been used for aggregated queries and statistical data analysis.

PINQ [McSherry 2010] is a data analysis platform designed to provide differential privacy guaran-

tees. PINQ uses its query system, similar to SQL, called LINQ, and acts as an intermediate layer between a database and the data holder. Therefore it allows the data holder to carry out consultations without compromising the privacy of individuals that contributed with their data to the database.

PrivSQL [Kotsogiannis et al. 2019], a differentially private SQL query engine created upon two tenets: Answer workloads through constructed synopses and construct these synopses over a set of views, that reflect the data organization of the underlying database, without violating privacy rules. Once a query is posed against the system, it is analyzed within the context of previously generated views and rewritten accordingly. This causes the query sensitivity to be bounded within the answering view sensitivity. The work also describes how to generate the synopses for data defined by the views, respecting the a given differential privacy budget and definitions.

SAP HANA [Kessler et al. 2019], a widely known commercial DBMS, also has its take on privacy. From implementing more broad methods such as k-anonimity to applying the concept of local differential privacy, the DBMS integrates these concepts in a mechanism called Privacy Views.

These works offer the power of differential privacy to answer numerical queries closer enough to the real results to be useful but slightly different, so that person re-identification is unlikely. Although several approaches exist for publishing and sharing information in a differentially private way, they cover most of the aggregate queries, such as COUNT(*), and noise that has to be added in publishing settings is usually too high. Moreover, mechanism integration to existing DBMS is still a vast subject of research.

2.6 User related data retrieval

This requirement relates to two of the policies described: right of access and consent. It brings the problem of timely retrieving all the data related to a given user, even if distributed across relations and derived information. The storage engine must account for this data, and facilitate retrieval of all user-related data. Moreover, the DBMS must guarantee that every user has their own data tagged and available through metadata tags and index structures. This is also a vastly explored theme in academia, however, points to be taken into consideration are space allocation, as well as timely retrieval and deletion.

Compliance by Construction [Schwarzkopf et al. 2019] suggests that all user data is stored in user shards. The shards are inaccessible for query, instead, materialized views based on the query and the purposes associated would be produced to provide the data while hiding the true data and allowing the user to request, remove or revoke access.

Other cited works also have solutions for this component, be them explicit, such as Sypse [Deshpande 2021] personal information partition, or implicit, like SchengenDB [Kraska et al. 2019] and GDPRBench [Shastri et al. 2020], which embed the information within data. Although there is not a clear separation of user related data, it is retrievable through its unique identifiers.

Research challenges in this area overlap with others discussed before, highlighting the entity resolution problem, fast access of all user related data and succinct data structures that can represent the whole user, facilitating data retrieval.

2.7 Discussion

As Shah et al. [Shah et al. 2019] mentions, GDPR compliance can be seen as a 2-dimensional spectrum, ranging from real-time to eventual compliance. This means that either the system may be GDPR-compliant at all times, or guarantee said compliance given some time. The system can also be evaluated from a full vs partial compliance stance, in terms that either it complies with all GDPR policies or some of them. We agree with this spectrum, arguing that this flexible compliance should be dynamic and configurable. DBMSs must have a well-defined set of rules regarding compliance,

which component is accountable for each rule, and how it is configured. They must also make it clear to applications what should be considered to achieve full compliance, as well as to avoid interference from applications into the DBMS guarantees.

Data replication is an issue since when data is copied, it produces more metadata as well as needs more resources to track the copies. There are position papers on both sides, that data should never be copied and that data must always be provided as a copy. We believe that copies should be discouraged, as tracking data may slow down the system as well as leak information, however, derived results that do not break single individual privacy can be copied because, even if a user requests removal of the data, their data is not possible to single out.

Another issue that appears is that many of the modifications may conflict so much with fundamental DB design that they can be either impractical or place trust in the DB administrator (DBA), a human component subject to malice and failure. SchengenDB provides examples of log maintenance since erased data will still be present in log records, as well as in the suggested "sandbox" approach, instead of stopping intercommunication altogether, the system can be permissive and warn that access across different purposes happens, then, it would notify the DBA, placing the trust that appropriate action is taken.

Table I. Comparison of works x impacts.

	Metadata Explosion	Delete Guarantees	Efficient Auditing	Purpose-based Access	Opted Data	User Data Retrieval
[Shastri et al. 2020]	Partial	Complete	Partial	Partial	N/A	Partial
[Kraska et al. 2019]	N/A	N/A	Partial	Complete	Partial	Complete
[Schwarzkopf et al. 2019]	N/A	Complete	Partial	Complete	Partial	Complete
[Deshpande 2021]	N/A	Complete	N/A	Partial	Complete	Complete
[Sarkar et al. 2020]	N/A	Optimized	N/A	N/A	N/A	N/A
[Kotsogiannis et al. 2019]	N/A	N/A	N/A	N/A	Optimized	Partial
[Haubenschild et al. 2020]	N/A	N/A	Optimized	N/A	N/A	N/A
[Pappachan et al. 2020]	Partial	N/A	N/A	Optimized	Partial	N/A
[Zhang et al. 2020]	Partial	N/A	N/A	Partial	N/A	N/A
[Deeds et al. 2021]	Partial	N/A	N/A	Complete	Partial	N/A

Table I provides a summarized comparison of selected works described in this paper against the impacts mentioned, in an overview relating their characteristics. While there are works that address many of the challenges, they either only discuss possible solutions, or provide acknowledgement of the issue. More specialized works provide optimized solutions, although these works tend to focus only on one of the issues.

From this comparison, one takeaway is that current systems need to be redesigned considering these new restrictions. Another possible solution is to add a new layer to the hierarchy, to ensure that data coming out of the DBMS is properly handled. However, this approach can generate many vulnerabilities and add an overhead to general query processing, which can add up and ultimately hinder performance in the long run.

3. CONCLUSION AND FUTURE WORK

In this work, we extend the discussion of [Machado and Amora 2020] adding more references and an increased depth to the discussion of each work. Moreover, with the detailing of each right and impact, we achieve a higher level of understanding the implications of these rights, and better relate existing solutions to each impact, providing more detail on their strengths and weaknesses. Considering all the arguments and opportunities presented, we can build the following propositions:

The DBMS workload profile will change drastically, once read-only transactions become write-intensive, as a result of increasing metadata and recording accesses and purposes. In a non-optimized compliant relational DBMS, there is a decrease in performance in 2 orders of magnitude. While some of this overhead is unavoidable, optimizations are possible once these new characteristics are considered in design.

The DBMS, as well as the storage manager, must become aware of the queries, the data, and the metadata, to provide required data as well as to make sure that no unauthorized data leaks through a query while addressing the performance overhead caused by these constraints. Moving the constraints to the DBMS brings two benefits: Access to personal data can be protected even from outside the application, since the DBMS will have awareness regarding these accesses, and more accurate query processing estimates can be done, with respect to query optimization and plan selection, allowing for a reuse of existing components without the need for retrofitting.

Data must be treated as a first-class citizen, instead of being neglected in favor of overall performance increase and low response times. Operations must be confirmed, not only in state, but also in physical storage and data structure, with a more attentive look to deletes, which many times happen logically to favor a high throughput in these operations. These operations may add new overheads or increase the relevance of existing ones.

A new class of activity recording must also emerge, either to facilitate audit activities, but also to identify unusual patterns within the tasks performed by the database, which may indicate a data breach and allow the DBA to act accordingly.

These new challenges oppose the usual beliefs of DBMS design and, to achieve performance and compliance, it is necessary to rethink data structures as well as database architectures. For future works, there are possibilities in either new algorithms or data structures for reducing each impact, be it metadata explosion, data or log access and retrieval; or in adapting different solutions to work together, to address different impacts simultaneously. By integrating different solutions, an optimized, GDPR-compliant DBMS may be achieved.

REFERENCES

A29WPT, A. . W. P. T. Article 29 Working Party. `https://ec.europa.eu/newsroom/article29/news-overview.cfm`, 2018. [Online; accessed 03-Mars-2021].

Byun, J. and Li, N. Purpose based access control for privacy protection in relational database systems. *VLDB J.* 17 (4): 603–619, 2008.

Cohen, A. and Nissim, K. Towards formalizing the GDPR's notion of singling out. *Proceedings of the National Academy of Sciences* 117 (15): 8344–8352, 2020.

Dalibo. PostgreSQL Anonymizer. `https://postgresql-anonymizer.readthedocs.io/en/stable/`, 2020. [Online; accessed 14-Mars-2021].

Deeds, K., Hentschel, B., and Idreos, S. Stacked filters: Learning to filter by structure. *Proceedings of the VLDB Endowment* 14 (4): 600 – 612, 2021.

Deshpande, A. Sypse: Privacy-first Data Management through Pseudonymization and Partitioning . In *CIDR*. www.cidrdb.org, 2021.

Domingo-Ferrer, J., Sánchez, D., and Soria-Comas, J. Database anonymization: Privacy models, data utility, and microaggregation-based inter-model connections. *Synthesis Lectures on Information Security, Privacy, & Trust* 8 (1): 1–136, 2016.

Dwork, C. Differential privacy. In *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*. Lecture Notes in Computer Science, vol. 4052. Springer, pp. 1–12, 2006.

Fung, B. C. M., Wang, K., Chen, R., and Yu, P. S. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys* 42 (4): 1–53, June, 2010.

General Data Protection Regulation. 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. *Official Journal of the European Union* vol. 59, pp. 1–88, 2016.

Graefe, G., Guy, W., and Sauer, C. *Instant Recovery with Write-Ahead Logging: Page Repair, System Restart, Media Restore, and System Failover, Second Edition.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2016.

Haubenschild, M., Sauer, C., Neumann, T., and Leis, V. Rethinking logging, checkpoints, and recovery for high-performance storage engines. In *SIGMOD Conference*. ACM, pp. 877–892, 2020.

Kessler, S., Hoff, J., and Freytag, J. SAP HANA goes private - from privacy research to privacy aware enterprise analytics. *Proc. VLDB Endow.* 12 (12): 1998–2009, 2019.

Kotsogiannis, I., Tao, Y., Machanavajjhala, A., Miklau, G., and Hay, M. Architecting a differentially private SQL engine. In *CIDR*. www.cidrdb.org, 2019.

Kraska, T., Stonebraker, M., Brodie, M. L., Servan-Schreiber, S., and Weitzner, D. J. SchengenDB: A data protection database proposal. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB 2019 Workshops, Poly and DMAH, Los Angeles, CA, USA, August 30, 2019*. Lecture Notes in Computer Science, vol. 11721. Springer, pp. 24–38, 2019.

Machado, J. C. and Amora, P. R. P. How can db systems be ready for privacy regulations. In *SBBD*. SBC, 2020.

Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. *L*-diversity: Privacy beyond *k*-anonymity. *TKDD*, 2007.

MariaDB. MaxScale Filters Masking. `https://mariadb.com/kb/en/mariadb-maxscale-21-masking/`, 2019. [Online; accessed 06-Mars-2021].

McSherry, F. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *Communications of the ACM* 53 (9): 89–97, 2010.

Microsoft. Dynamic data masking. `https://docs.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview`, 2021. [Online; accessed 25-Feb-2021].

Mishra, M. and Somani, A. K. On-disk data processing: Issues and future directions. *CoRR* vol. abs/1709.02718, 2017.

Oracle Inc. Oracle Audit Vault, 2021. Accessed: 2021-03-16.

Pappachan, P., Yus, R., Mehrotra, S., and Freytag, J. Sieve: A middleware approach to scalable access control for database management systems. *Proc. VLDB Endow.* 13 (11): 2424–2437, 2020.

Pena, E. H. M., Filho, E. R. L., de Almeida, E. C., and Naumann, F. Efficient detection of data dependency violations. In *CIKM*. ACM, pp. 1235–1244, 2020.

Riggs, S., Menon-Sen, A., Barwick, I., and Steele, D. PGAudit, 2021. Accessed: 2021-03-16.

Rizvi, S., Mendelzon, A. O., Sudarshan, S., and Roy, P. Extending query rewriting techniques for fine-grained access control. In *SIGMOD Conference*. ACM, pp. 551–562, 2004.

Sarkar, S., Papon, T. I., Staratzis, D., and Athanassoulis, M. Lethe: A tunable delete-aware LSM engine. In *SIGMOD Conference*. ACM, pp. 893–908, 2020.

Schwarzkopf, M., Kohler, E., Kaashoek, M. F., and Morris, R. T. Position: GDPR compliance by construction. In *Poly/DMAH@VLDB*. Lecture Notes in Computer Science, vol. 11721. Springer, pp. 39–53, 2019.

Shah, A., Banakar, V., Shastri, S., Wasserman, M., and Chidambaram, V. Analyzing the impact of GDPR on storage systems. In *HotStorage*. USENIX Association, 2019.

Shastri, S., Banakar, V., Wasserman, M., Kumar, A., and Chidambaram, V. Understanding and benchmarking the impact of GDPR on database systems. *Proc. VLDB Endow.* 13 (7): 1064–1077, 2020.

Sweeney, L. k-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 10 (5): 557–570, 2002.

Zhang, H., Liu, X., Andersen, D. G., Kaminsky, M., Keeton, K., and Pavlo, A. Order-preserving key compression for in-memory search trees. In *SIGMOD Conference*. ACM, pp. 1601–1615, 2020.