The Right to be Forgotten from a Database: Meaningful Data Erasure in the Presence of Data Dependencies

ABSTRACT

Data regulations like GDPR require systems to support data erasure but leave the definition of 'erasure' open to interpretation. This ambiguity makes it challenging to ensure compliance, especially in databases where data dependencies can lead to erased data being inferred from remaining data. In this paper, we formally define a strong notion of data erasure, where the system guarantees that the probability of inferring the value of erased data, given a database that complies with a given set of dependencies, remains the same as it was before the data was inserted. We design erasure mechanisms that provide such a strong semantic guarantee with minimal cost. Additionally, we explore strategies to balance cost and throughput, batch multiple erasures, and proactively compute data retention times when possible. We demonstrate the practicality and scalability of of our algorithms using both real and synthetic datasets.

1 INTRODUCTION

The right to have personal data erased, commonly known as the 'Right to be Forgotten' [32], is a cornerstone of the GDPR and nearly all other major data protection regulations worldwide [1, 14, 36, 49]. The objective of data erasure provisions is twofold: to empower individuals to request organizations to delete their personal data at will, and to mitigate the risk of personal data being misused by restricting how long it can be retained. An incessantly rising number of compliance violations [2, 41] as well as recent research [4, 8, 15, 43, 46] highlights both the critical need for and the technical challenges associated with implementing these requirements in modern data management systems. Existing approaches that simply expunge the data the user wishes to be forgotten/deleted do not suffice, especially when dealing with complex semantic data stored in databases. Consider the following scenario that often occurs in social media and messaging applications like Facebook, WhatsApp, etc., which we use as a running example throughout the paper.

Example 1.1. The set of tables in a typical social network application is shown in Fig. 1. Every person (in the `Person` table) has a profile (`Profile`) with attributes such as profile picture (`profPic`), last known location (`lastLoc`); statistics and other data used for maintenance/analytics is stored in the `Profile_Stat` table with attributes such as average online activity (`avgAct`), total number of likes (`totLikes`), most frequented location (`freqLoc`), etc. Users can post photos (`Photos`) and posts (`Posts`) which have their own associated locations (`phtLoc` and `pstLoc`, respectively). Suppose, Bob, a user, wants to erase their location information collected by the application. Simply erasing their location is not sufficient as the system can possibly reconstruct Bob's location from data spread across multiple tables —`pstLoc` in `Posts`; `phtLoc` in `Photos`; `CheckIn` and `Events` tables; `freqLoc` in `Profile_Stat`. □

In such scenarios erasure requires deleting not just data the user wishes to delete but also deleting additional data from which the deleted data can be inferred. Today, systems do not offer any user-facing guarantees on persistent erasure of data [44] where data can be semantically dependent. Our work addresses this shortcoming by defining a meaningful semantic guarantee for data erasure by comparing what can be inferred about data after it has been erased to that at a specified time τ in the past. We call such a guarantee *exclusion post erasure* with respect to time τ. When τ is the time at which the data being erased was inserted, we refer to it as *partial exclusion post erasure* (P2E2). We focus on P2E2 in our work, which, to the best of our knowledge, is the first proposal to formally define erasure when the underlying data can be semantically dependent.

To build systems that support P2E2, we introduce *relational dependency rules* (RDRs). With SQL-like syntax, an RDR provides a simple way to specify which cells and the condition(s) in which they are dependent *without* the need to specify how they are dependent.

Example 1.2. Continuing our example, the RDR below states that the total-likes (`totLikes`) of a profile R is dependent on the likes (`pstLikes`) of all the posts (in `Posts`) authored by that profile.

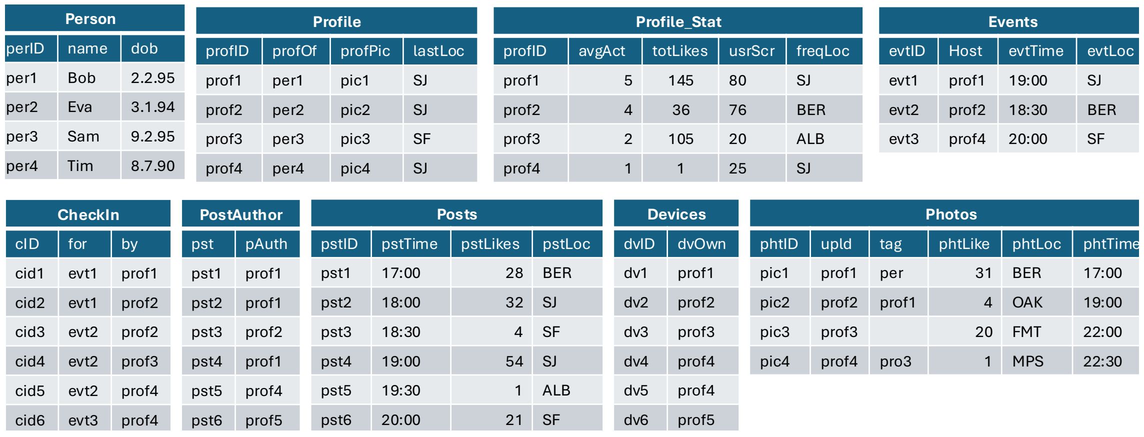
$$\begin{aligned} \texttt{totLikes}(R) \not\perp\!\!\!\perp \texttt{pstLikes}(M) \\ \textit{FROM}\ \texttt{Profile\_Stat}(R), \texttt{Post}(M), \\ \textit{WHERE}\ \texttt{profID}(R) = \texttt{author}(M) \end{aligned} \qquad (1)$$

On erasing `totLikes`, that total number of likes can still be inferred from the likes of the posts. To ensure *no* inference on[1] `totLikes`, the number of likes received by each post needs to be erased. □

RDRs offer a powerful tool to represent semantic data dependencies. They, in essence, provide a general way to express which data are not independent (i.e., when the value of a cell may lead to the inference on the value of another cell). Dependencies traditionally studied in database literature, such as aggregation constraints [40], (conditional) functional dependencies [12, 19], similarity inclusion dependencies [26], or more generally, denial constraints, are all expressible as RDRs. RDRs can also express semantic constraints and annotations studied in recent work to support data regulation compliance [5, 6, 16] in systems, and correlations and soft functional dependencies [23]. Moreover, RDRs can specify complex data erasure logic such as Meta's policy of erasing comments made by the requester, but not private messages sent to others, unless both parties request erasure [3]; selective data retention for legal/contractual obligations as stated in Article 17.3 in [32]), etc.

Using RDRs as a model to represent dependencies in data, we explore erasure mechanisms to ensure the P2E2 guarantee over erased data. Data erasure is usually classified into two types [8]: (a) *demand-driven erasure* that corresponds to ad-hoc requests by users/applications to delete data, and (b) *retention-driven erasure* where data, at the time of insertion, is associated with a retention period providing the system with prior knowledge of when the data is obligated to be erased. For example, WhatsApp [7] supports

[1] By inference, we mean being able to determine what the possible values `totLikes` can or cannot take.



Person		
perID	name	dob
per1	Bob	2.2.95
per2	Eva	3.1.94
per3	Sam	9.2.95
per4	Tim	8.7.90

Profile			
profID	profOf	profPic	lastLoc
prof1	per1	pic1	SJ
prof2	per2	pic2	SJ
prof3	per3	pic3	SF
prof4	per4	pic4	SJ

Profile_Stat				
profID	avgAct	totLikes	usrScr	freqLoc
prof1	5	145	80	SJ
prof2	4	36	76	BER
prof3	2	105	20	ALB
prof4	1	1	25	SJ

Events			
evtID	Host	evtTime	evtLoc
evt1	prof1	19:00	SJ
evt2	prof2	18:30	BER
evt3	prof4	20:00	SF

CheckIn		
cID	for	by
cid1	evt1	prof1
cid2	evt1	prof2
cid3	evt2	prof2
cid4	evt2	prof3
cid5	evt2	prof4
cid6	evt3	prof4

PostAuthor	
pst	pAuth
pst1	prof1
pst2	prof1
pst3	prof2
pst4	prof1
pst5	prof4
pst6	prof5

Posts			
pstID	pstTime	pstLikes	pstLoc
pst1	17:00	28	BER
pst2	18:00	32	SJ
pst3	18:30	4	SF
pst4	19:00	54	SJ
pst5	19:30	1	ALB
pst6	20:00	21	SF

Devices	
dvID	dvOwn
dv1	prof1
dv2	prof2
dv3	prof3
dv4	prof4
dv5	prof4
dv6	prof5

Photos					
phtID	upId	tag	phtLike	phtLoc	phtTime
pic1	prof1	per	31	BER	17:00
pic2	prof2	prof1	4	OAK	19:00
pic3	prof3		20	FMT	22:00
pic4	prof4	pro3	1	MPS	22:30

Figure 1: Tables in the database of a social networking application.

retention-driven erasure by letting users choose a retention period of one day, one week, or three months for disappearing messages.

Irrespective of the type of erasure, an erasure mechanism must determine a minimal set of dependent data that needs to be deleted to ensure the P2E2 guarantee. To this end, we first utilize Independence Axioms [21] in probability theory to (indirectly) reason about which data is semantically dependent based on RDRs. Using this reasoning framework, we formalize the problem of identifying a minimal set of data to delete in order to achieve P2E2 as the OPT-P2E2 problem. We generalize the problem into a cost-based variant to accommodate for potentially different costs for different data (as discussed in recent work on the economics of data markets). To identify a cost-effective strategy to achieve P2E2, we explore multiple approaches to solve OPT-P2E2 — using integer linear programming (ILP); a bottom-up tree-traversal approach; a top-down tree traversal approach (which attempts to trade off cost for higher throughput); batching multiple erasures.

While the mechanism(s) to achieve P2E2 can be applied to both types of erasures, demand-driven and retention-driven erasures differ on when that mechanism can be invoked. Since demand-driven erasure request could arise at anytime, the erasure algorithm needs to be initiated as and when such requests arise. In contrast, for retention-driven erasure, since the retention period is known upfront, erasures can be scheduled in advance. Such scheduled erasures can be beneficial, especially when base data is used to compute (potentially expensive) *derived data* (e.g., aggregates, materialized views, and models) which is used by downstream applications. Based on our example table in Figure 1, base data such as `Profile` and the `Posts` tables (Fig. 1) may be used to create a report (materialized view) that summarizes statistics of the top 1% active profiles in a city for each quarter. P2E2 guarantee on the base data (`Profile`, `Posts`) requires that any derived data (report) constructed using deleted base data also be erased and reconstructed. For retention-based erasure, we can determine the reconstruction schedule of derived data such that the number of additional erasure and reconstructions required for guaranteeing P2E2 for the erasure of base data is minimized.

We build erasure mechanisms to support P2E2 guarantees for both demand-driven and retention-driven erasures in databases. Given existing RDRs (specified by the analyst), the erasure mechanisms identify *what* data needs to be deleted and *when* should such deletions be performed. To reduce overheads, we explore batching demand-driven erasures and compute schedules to manage retention-driven erasures. We extensively evaluate our approaches on real and synthetic datasets and under various workloads. We analyze the overheads of guaranteeing P2E2—its associated cost and impacts on system performance in both demand-driven and retention-driven scenarios. We compare exact and approximate algorithms and analyze the trade off between the time taken and the cost of guaranteeing P2E2. We study the effects of batching using varying grace-periods and pre-computing retention periods of derived data on overheads.

In summary, we make the following contributions:

- **Relational Dependency Rules (RDRs):** a powerful model to represent semantic dependencies in data.
- **Partial Exclusion Post Erasure (P2E2):** a strong notion of data erasure suitable for databases when data may be dependent and could be inferred from other data.
- **Algorithms to support P2E2:** exact and approximate algorithms to guarantee P2E2 for both retention-based and demand-driven data erasure requests.
- **Extensive performance study:** to evaluate our approaches on real and synthetic datasets under demand-driven, retention-driven, and mixed data erasure workloads.

In the following, Section (Sec. henceforth) 2 formalizes data erasure and its semantic guarantee (P2E2). RDRs are introduced in Sec. 3, along with technical results on how to reason with RDRs when erasing a specific cell. Sec. 4 develops exact and approximate algorithms for demand-driven data erasure. In Sec. 5, we address retention-driven data erasures. We evaluate our algorithms and analyze the overheads of P2E2 in Sec. 6. We conclude with related work (Sec. 7) and a discussion on future work (Sec. 8). Due to space constraints, proofs have been deferred to a longer version[2].

[2]https://anonymous.4open.science/r/P2E2-Erasure-D806

2 FORMALIZING DATA ERASURE

In this section, we introduce the formal semantics of data erasure. We adopt standard notation and concepts from [21, 30].

2.1 Preliminaries

We borrow and extend the functional data model [47], and its recent adaptation [30]. Consider a database consisting of the set of relations $\mathcal{R} = \{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_m\}$. Each relation $\boldsymbol{R}_i$ has an associated set of attributes $\boldsymbol{R}_i^{attr} = \{\boldsymbol{R}_i.\mathsf{A}_1, \ldots, \boldsymbol{R}_i.\mathsf{A}_{n_j}\}$. We drop the relation and write only A_j when referring to an attribute when the corresponding relation is clear from the context. Some attributes in $\boldsymbol{R}_i^{attr}$ take as argument a key(s) (of the relation) and return a value in the domain of that attribute. We call these *attribute functions.* More specifically, for a relation $\boldsymbol{R}_i$ and an attribute function A_j, we have $\mathsf{A}_j : \boldsymbol{R}_i \longrightarrow Dom(\mathsf{A}_j)$, where $Dom(\mathsf{A}_j)$ is the set of all possible values the attribute function can take including *NULL*. An instantiation of an attribute function A_j is denoted as $\mathsf{A}_j(\boldsymbol{x})$ where $\boldsymbol{x}$ is a tuple of constants. The set of all attributes is denoted as $Attr(\mathcal{R})$ and the set of all attribute functions is denoted as $\mathscr{A}$. Note that $\mathscr{A} \subseteq Attr(\mathcal{R})$. A relational functional schema is a tuple $\mathcal{S} = (\mathcal{R}, Attr(\mathcal{R}), \mathscr{A})$.

Example 2.1. Let $S = (\mathcal{R}, Attr(\mathcal{R}), \mathscr{A})$ be the relational functional schema for the instance in Fig. 1. Examples of relations include `Profile`, `Person`, `Post`, `Photos` $\in \mathcal{R}$. For $P \in$ `Person`, $R \in$ `Profile`, $C \in$ `Posts`; attribute functions `perName`(P), `lastLoc`(R), `profOf`(R), `totLikes`$(C) \in \mathscr{A}$ return the respective values. □

Data Model At any instance t, each relation $\boldsymbol{R}_i$ in the database $\mathcal{D}$ contains a set of records $r_j^i \in \boldsymbol{R}_i$. We refer to the set of records in all the relations in $\mathcal{D}$ as the *database state* of $\mathcal{D}$ at time t, and denote it as $\mathcal{D}_t$. For a database state $\mathcal{D}_t$, we write $\mathcal{D}_{t^-}$ to denote the state immediately preceding $\mathcal{D}_t$ and $\mathcal{D}_{t^+}$ to denote the state immediately following $\mathcal{D}_t$.

Each attribute A_k of record r_j^i in a relation $\boldsymbol{R}_i$ in $\mathcal{D}_t$, i.e., $r_j^i.A_k$ is a *cell* and its value is denoted by $Val(c, \mathcal{D}_t)$, where $c = r_j^i.A_k$. To avoid excessive notation, we refer to the instantiation of an attribute function $\mathsf{A}(\boldsymbol{x})$ as a cell. Since cells can be erased without the tuple in which it resides being erased, its value is considered to be *NULL*, i.e., $Val(c, \mathcal{D}_t) = NULL$, for anytime t after its erasure. Erasure of a record is modeled as erasure of all the cells in the record.

Relations are classified as either *base* or *derived.* A base relation $\boldsymbol{R}_i$ consists of data collected directly (or indirectly) from the user or other sources. Derived relations, on the other hand, may be produced by processing data in base relations (or other derived relations). Such processing may including computing aggregations, view computation, enrichment, or running AI-models, etc. Derived relations is often created to support a variety of organization's analytical and maintenance tasks, and/or to drive application-level features. In Fig. 1, the relation `Profile` is a base relation whereas the relation `Prof_Stat` is a derived relation.

Retention Model. In the following, we discuss the retention model for data in base relations and data in derived relations.

Base Relations. Data in base relations is associated with two time stamps – a creation and an erasure timestamp. We associate such timestamps at the cell level. Thus, for a given cell c, we associate a: (i) creation -timestamp: $\kappa(c)$ that denotes when c is inserted, and (ii) erasure-timestamp: $\eta(c)$ that denotes when c is (to be) deleted. The difference between creation and the erasure timestamps of a cell is its *retention period.* Retention periods for cells are usually fixed at the time of insertion of the record to which it belongs. When a record r_j^i is inserted into a base relation $\boldsymbol{R}_i$, all cells c in r_j^i (for each attribute A_k) are initiated with $\kappa(c)$ which is the time of insertion of r_j^i and $\eta(c)$ - possibly based on the user's choice at the time of insertion. A user can, however, change $\eta(c)$ of their data as desired. Consider a cell c with default erasure-timestamp of 5pm, i.e, $\eta(c) = 5pm$. At $2pm$, the user wants their data in c to be erased then the erasure-time is changed to $\eta(c) = 2pm$. Such an erasure request essentially converts a retention-driven erasure into a demand-driven erasure. In addition to deleting a cell, a user can also update its value. Such an update is treated as an erasure of the cell, followed by an insertion with the modified cell value. In such a case, the value of its insertion timestamp $\kappa(c)$ is modified as well to the time of the update, while its erasure time $\eta(c)$ remains the same as prior to the update. When data (cells) in base relations are erased, the system ensures strong semantic erasure guarantee, P2E2, formalized later.

Derived Relations. Data in derived relations, as mentioned earlier, are computed from either base or other derived relations. Such data may periodically be recomputed at some frequency based on the need or organizations and/or business rules. For instance, in our running example, business rules may require `usrScr` (score assigned to a profile for analytics) fo to be computed at least once a month, while `totLikes` be recomputed at least once every week. Thus, for derived relations, in addition to associating a creation time stamp with each cell c, i.e., $\kappa(c)$ (as in the base relation), we further associate a parameter *freq*(c) which denotes the time in which it must be reconstructed at least once. In our example, *freq*(c) is 30 days and 7 days for c corresponding to `usrScr` and `totLikes` respectively. Whenever a cell in the derived data is recomputed/reconstructed, its creation-time, $\kappa(c)$, is updated to the time of reconstruction. Its *freq*(c) remains the same as before. We do not associate erasure timestamps with derived data since derived data is not explicitly deleted/erased by users. Derived data may, nevertheless, be erased as a side effect of a user's request to erase base data in case derived data can lead to inferences about base data. In such a case, it will reconstructed after the base data has been erased. Derived data can, however, be deleted when it is no longer needed in which case it will no longer be reconstructed after deletion.

Data Dependencies. We briefly introduce data dependencies here and present them more formally in Sec. 3. Derived data often depends on base data as well as other derived data. We call such dependencies among base and/or derived data *data dependencies.* For now, we assume that data dependencies are available as probabilistic dependence statements. More specifically, if instantiated attribute functions $\mathsf{A}_1(\boldsymbol{x}_1)$ and $\mathsf{A}_2(\boldsymbol{x}_2)$ are *not* independent, then the dependency is expressed as the RDR $\mathsf{A}_1(\boldsymbol{x}_1) \not\perp\!\!\!\perp \mathsf{A}_2(\boldsymbol{x}_2)$. A set of such instantiated RDRs corresponding to the database state $\mathcal{D}_t$ is denoted with $\Delta^-(\mathcal{D}_t)$. When data dependencies are present, they may reveal information about the value an attribute function may take, i.e., if the value of $\mathsf{A}_2(\boldsymbol{x}_2)$ is known (non-*NULL* value), it may reveal the value of $\mathsf{A}_1(\boldsymbol{x}_1)$, and vice versa.

Example 2.2. In our running example (Fig. 1), if the value of `pstLikes`(pst4) is known, i.e., 54, it leads to the inference that totLikes(prof1) is more than or equal to 54. Conversely, if the value of totLikes(prof1) is known to be 145, then it leads to the inference that pstLikes(pst4) has to be less than 145. Therefore, there is a dependency between pstLikes(pst4) and totLikes(prof1) which is expressed as pstLikes(pst4) $\not\perp\!\!\!\perp$ totLikes(prof1). □

2.2 Data Erasure Semantics

We formalize the semantic guarantees of data erasure in this section. We assume that at a given time, the database owner has access to the entire database at that time, and the dependencies. We do not consider adversarial scenarios wherein a malicious database owner maintains a copy of data secretly. Incorporating such a possibility will go well beyond the scope of erasure we are considering here.

First, we introduce some notation. Let $\mathcal{D}_t$ be a database state comprising the instantiations of attribute functions $\{\mathsf{A}_1(\boldsymbol{x}_1), \ldots, \mathsf{A}_1(\boldsymbol{x}_n), \ldots, \mathsf{A}_2(\boldsymbol{x}_1), \ldots, \mathsf{A}_2(\boldsymbol{x}_n), \ldots, \mathsf{A}_n(\boldsymbol{x}_n)\}$. For $\mathsf{A}_i(\boldsymbol{x}_j) \in \mathcal{D}_t$, we write $Pr(\mathsf{A}_i(\boldsymbol{x}_j) \mid \mathcal{D}_t)$ to denote the conditional probability distribution for each instantiated attribute function $\mathsf{A}_i(\boldsymbol{x}_j)$. If the value of $\mathsf{A}_i(\boldsymbol{x}_j)$ is not *NULL*, i.e., its value is known, then $Pr(\mathsf{A}_i(\boldsymbol{x}_j) = val \mid \mathcal{D}_t) = 1$, where *val* is the value of $\mathsf{A}_i(\boldsymbol{x}_j) \in \mathcal{D}_t$. We can now formally define P2E2, our semantic notion of erasure.

Definition 2.3 (Partial Exclusion Post Erasure (P2E2)). Given an attribute function $\mathsf{A}(\boldsymbol{x})$ with insertion time $\kappa(\mathsf{A}(\boldsymbol{x})) = t_b$, expiration time $\eta(\mathsf{A}(\boldsymbol{x})) = t_e$ and value $Val(\mathsf{A}(\boldsymbol{x}), \mathcal{D}_{t_e}) = val$ that is not null, we say that *partial exclusion post erasure (P2E2)* holds for $\mathsf{A}(\boldsymbol{x})$ if: $Pr(\mathsf{A}(\boldsymbol{x}) = val \mid \mathcal{D}_{t_b^-}) = Pr(\mathsf{A}(\boldsymbol{x}) = val \mid \mathcal{D}_{t_e^+})$.

Intuitively, P2E2 states that the probability of an attribute function taking a value post its erasure must be the same as that at the time prior its insertion. We could consider a more generalized definition of semantic erasure by instantiating t_b in the P2E2 definition above to any time τ (i.e., $t_b = \tau$) instead of instantiating it to $t_b = \kappa(\mathsf{A}(\boldsymbol{x}))$ as in the current definition. For example, instantiating t_b to 0 would require the adversary's ability to determine the value of an erased data to be the same as it was prior to the creation of the database. Such an instantiation is significantly more stringent (requiring many additional deletions) compared to P2E2: now we must delete additional data, even what was in the database before insertion of the data being erased, if such data (coupled with the domain knowledge in the form of RDRs) could lead to the inference on the possible value of the erased cell. Such a definition would require erasing *all* dependent data inserted into the database since $\tau = 0$. In contrast, P2E2 states that the probability of an attribute function taking a value post its erasure must be the same as that at the time prior its insertion.

We believe that P2E2 captures what one would expect and desire of an erasure in databases. Yet, the algorithms we study to implement P2E2 can be generalized to even more stringent definitions of erasure without too many changes.

3 DATA DEPENDENCIES

In a database, the value of a cell often depends on that of other cells. To account for such dependencies, we introduce *relational dependency rules* (RDRs) [3].

[3]Although the syntax of RDRs is inspired by relational causal rules [30] due to their ease in understanding and familiar SQL-like structure, RDRs express semantic data dependencies whereas relational causal rules express causal dependencies.

3.1 Specifying Background Knowledge by RDRs

Relational dependency rules (RDRs) express dependence (background knowledge) among cells without specifying how the cells are dependent or associating any preference to dependencies. Thus, they can be used to express traditional dependencies, such as denial constraints [10], as well as correlations among attributes and semantic dependencies at the application level.

Definition 3.1 (Relational Dependence Rules). Let $\boldsymbol{S} = (\mathcal{R}, Attr(\mathcal{R}), \mathcal{A})$ be a relational functional schema where $\mathsf{A}, \mathsf{A}_1, \ldots, \mathsf{A}_q \in \mathcal{A}$ are attribute functions, Q is a (standard) SQL query over the schema $\boldsymbol{S}$, and $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_p, \boldsymbol{V},$ and $\boldsymbol{Z}$ are sets of variables and/or constants. All variables in $\boldsymbol{X}, \boldsymbol{X}_i$, and $\boldsymbol{Z}$ must appear at least once in $\boldsymbol{V}$. A *relational dependence rule (RDR)* is given by

$$\mathsf{A}(\boldsymbol{X}) \not\perp\!\!\!\perp \mathsf{A}_1(\boldsymbol{X}_1), \ldots, \mathsf{A}_p(\boldsymbol{X}_p) \;\; \textit{FROM}\, \boldsymbol{V} \;\; \textit{WHERE}\, Q(\boldsymbol{Z}) \qquad (2)$$

We call $Q(\boldsymbol{Z})$ the *condition* of the rule. The expression to the left of the $\not\perp\!\!\!\perp$-symbol is called the *head* of the rule. The expression in between the $\not\perp\!\!\!\perp$ and *FROM* is called the *tail* of the rule. The *FROM* clause specifies the concepts pertaining to the head, tail, and the condition of the rule. We denote a set of RDRs with Δ^-.

Example 3.2. The RDR in Equation 3 expresses a dependency between the last location (lastLoc) of two profiles. The condition states that the profiles checked in at the same event (the event in the for attribute) have the same last location. Observe that it expresses a denial constraint across multiple relations. □

$$\begin{aligned}
&\mathtt{lastLoc}(P_1) \not\perp\!\!\!\perp \mathtt{profID}(R_1), \mathtt{profID}(R_2), \mathtt{by}(C_1),\\
&\qquad \mathtt{for}(C_1), \mathtt{by}(C_2), \mathtt{for}(C_2), \mathtt{lastLoc}(P_2)\\
&\qquad \textit{FROM}\ \mathtt{Profile}(R_1), \mathtt{Profile}(R_2), \mathtt{CheckIn}(C_1), \mathtt{CheckIn}(C_2),\\
&\qquad \textit{WHERE}\ \mathtt{profID}(R_1) = \mathtt{by}(C_1), \mathtt{profID}(R_2) = \mathtt{by}(C_2),\\
&\qquad \mathtt{for}(C_1) = \mathtt{for}(C_2) \qquad (3)
\end{aligned}$$

Instantiated RDRs An instantiated RDR is a rule in the normal form (as in Definition 3.1) that contains only constants from a database state $\mathcal{D}_i$ and no variables or condition (i.e., $Q \equiv \mathtt{true}$).

Definition 3.3 (Instantiations of rules). Let $\mathcal{D}_i$ be an instance over $\boldsymbol{S} = (\mathcal{N}, Attr(\mathcal{R}), \mathcal{A})$. Given a RDR in the normal form (Eqn. 2), for an instantiated attribute function $\mathsf{A}_k(\boldsymbol{x}_k)$, we associate the *instantiated* RDR, denoted $\delta^-(\mathcal{D}_i, \mathsf{A}_k(\boldsymbol{x}_k))$, obtained by assigning to the variables $\boldsymbol{Y} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_p\} - \boldsymbol{X}_k$ all constants $\boldsymbol{x}$ such that $\mathcal{D}_i \models Q(\boldsymbol{X}_k/\boldsymbol{x}_k, \boldsymbol{Y}/\boldsymbol{y})$. That is, the database satisfies the query by substituting the variables in $\boldsymbol{Y}$ with constants and treating the remaining variables in $\boldsymbol{Z}$ as existentially quantified. When clear from the context, we drop the database state $\mathcal{D}_i$ and the instantiated attribute function $\mathsf{A}_k(\boldsymbol{x}_k)$ from $\delta^-(\mathcal{D}_i, \mathsf{A}_k(\boldsymbol{x}))$ and write δ^- to denote an instantiated RDR. An instantiated RDR is of the form

$$\delta^- : \mathsf{A}(\boldsymbol{x}) \not\perp\!\!\!\perp \mathsf{A}_1(\boldsymbol{x}_1), \ldots, \mathsf{A}_p(\boldsymbol{x}_p) \qquad (4)$$

where $\mathsf{A}_i(\boldsymbol{x}_i)$ is the instantiated attribute function for the attribute function $\mathsf{A}_i(\boldsymbol{X}_i)$ with constants that satisfies $Q(\boldsymbol{Z})$.

The head of the instantiated RDR, denoted $Head(\delta^-)$, is $\mathsf{A}(\boldsymbol{x})$ and its tail, denoted $Tail(\delta^-)$ is given by the set $\{\mathsf{A}_1(\boldsymbol{x}_1), \ldots, \mathsf{A}_p(\boldsymbol{x}_p)\}$. The condition is omitted. We denote with $Cells(\delta^-)$ the set of all the attribute functions in δ^-, i.e., $\{Head(\delta^-)\} \cup Tail(\delta^-)$.

Example 3.4. In the following we present the instantiations of the RDRs in Example 3.2. The condition of the rule in Eqn. (3) is true for the vector of constants $\boldsymbol{v} = (\texttt{prof1}, \texttt{prof2}, \texttt{cid1}, \texttt{cid2})$. The corresponding instantiated RDR is

$$\delta_1^- : \texttt{lastLoc(prof1)} \not\perp\!\!\!\perp \texttt{profID(prof1)}, \texttt{profID(prof2)}, \texttt{by(cid1)}, \texttt{for(cid1)}, \texttt{by(cid2)}, \texttt{for(cid2)}, \texttt{lastLoc(prof2)} \tag{5}$$

The instantiations of the RDR in Example. (1.2) for `prof1` which has authored three posts are δ_2^- : $\texttt{totLikes(prof1)} \not\perp\!\!\!\perp \texttt{pstLikes(pst1)}$, δ_3^- : $\texttt{totLikes(prof1)} \not\perp\!\!\!\perp \texttt{pstLikes(pst2)}$, and δ_4^- : $\texttt{totLikes(prof1)} \not\perp\!\!\!\perp \texttt{pstLikes(pst4)}$. □

Given a database state $\mathcal{D}_t$, we denote with $\Delta^-(\mathcal{D}_t)$ the set of instantiated RDRs on that state.

3.2 Independence Through Dependence

In probability theory, two events A and B are independent, denoted $A \perp\!\!\!\perp B$, if $Pr(AB) = Pr(A) \cdot Pr(B)$, i.e., $Pr(A|B) = Pr(A)$. When A and B are not independent they are said to be dependent and denoted $A \not\perp\!\!\!\perp B$. Observe that an instantiated relational dependency rules (RDR) of the form in Equation (4) is an assertion that values taken by the attribute functions in that state are not independent, i.e., it is a dependence statement. Contrast this to the condition for P2E2 which asserts a probabilistic independence. Therefore, we need to ensure that an independence asserted by P2E2 is "consistent" with respect to the RDRs. Intuitively, given a set dependence statements (expressed by instantiated RDRs), we want to derive a set of independence statements such that these two sets are consistent, i.e., given a set of dependence statements in the form of instantiated RDRS, we have to derive a set of independence statements from those RDRs such that P2E2 is guaranteed. To that end, we will use the following characterization of Independence [21] to prove some of the results in this paper. We adopt standard notation and denote a set of dependence statements with Σ^- and a set of independence statements with Σ^+. We refer to statements of the form $A \perp\!\!\!\perp B$ as independence statements and $A \not\perp\!\!\!\perp B$ as dependence statements.

Theorem 3.5 (Characterization of Independence [21]). *For every set Σ^+ of independence statements closed under the following axioms, there exists a distribution P such that for each independence statement σ, we have σ holds for P if and only if $\sigma \in \Sigma^+$:*

- *Axiom 1 [Ax. 1].* $\boldsymbol{X} \perp\!\!\!\perp \emptyset$
- *Axiom 2 [Ax. 2].* $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y} \implies \boldsymbol{Y} \perp\!\!\!\perp \boldsymbol{X}$
- *Axiom 3 [Ax. 3].* $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{YW} \implies X \perp\!\!\!\perp Y$
- *Axiom 4 [Ax. 4].* $\boldsymbol{X} \perp\!\!\!\perp \boldsymbol{Y}$ & $\boldsymbol{XY} \perp\!\!\!\perp \boldsymbol{W} \implies \boldsymbol{X} \perp\!\!\!\perp \boldsymbol{YW}$

For a finite set of variables $\boldsymbol{U}$, and a set of independence statements Σ^+ we denote with $cl(\Sigma^+)$ the closure on Σ^+ by successively applying the Axioms(1-4). For a set Σ^- of dependence statements, whether $\Sigma^+ \cup \Sigma^-$ is consistent[4] can be determined in time polynomial in $|\Sigma^+|$ [21].

Given a set of RDRs, a database state $\mathcal{D}_t$, and an attribute function $\texttt{A}(\boldsymbol{x})$, to determine the other cells $\texttt{A}(\boldsymbol{x})$ is dependent on, the relevant RDRs need to be instantiated.

[4]For a finite set of variables $\boldsymbol{U}$, let Σ^+ be a set of independence statements on variables in $\boldsymbol{U}$ and let Σ^- be a set of dependence statements on variables in $\boldsymbol{U}$. We say $\Sigma^+ \cup \Sigma^-$ is *consistent* if there exists a probability distribution that satisfies it.

Direct Inference. An instantiated RDR δ^- leads to *direct* inference of the attribute function $\texttt{A}(\boldsymbol{x})$ if $\texttt{A}(\boldsymbol{x})$ is in $Cells(\delta^-)$.

Example 3.6. Consider the set $\{\delta_1^- : \texttt{A}(\boldsymbol{x}) \not\perp\!\!\!\perp \texttt{A}_1(\boldsymbol{x}); \delta_2^- : \texttt{A}_1(\boldsymbol{x}) \not\perp\!\!\!\perp \texttt{A}_2(\boldsymbol{x})\}$ of instantiated RDRs. RDR δ_1^- leads to direct inference of $\texttt{A}(\boldsymbol{x})$ and RDR δ_2^- leads to the direct inference of $\texttt{A}_1(\boldsymbol{x})$. □

The case when an instantiated RDR contains $\texttt{A}(\boldsymbol{x})$ in the head, is straightforward. Deleting any one attribute function from the tail of the instantiated RDR ensures that no inference on $\texttt{A}(\boldsymbol{x})$ is possible using this RDR as the remaining attribute functions in the tail and $\texttt{A}(\boldsymbol{x})$ are independent. Consider δ_1^- (in Equation 5). To prevent the inference of `lastLoc(prof1)` using δ_1^-, at least one of the attribute functions in the tail needs to be erased.

The second case is where the instantiated RDR δ^- contains $\texttt{A}(\boldsymbol{x})$ in its tail along with other attribute functions, i.e., let $\delta^- : \texttt{A}_1(\boldsymbol{x}_1) \not\perp\!\!\!\perp \texttt{A}(\boldsymbol{x}), \texttt{A}_2(\boldsymbol{x}_2), \ldots, \texttt{A}_n(\boldsymbol{x}_n)$. We will show that it suffices to consider the dependency $\texttt{A}(\boldsymbol{x}) \not\perp\!\!\!\perp \texttt{A}_1(\boldsymbol{x}_1)$ instead of δ^-, i.e., all the attribute functions besides $\texttt{A}(\boldsymbol{x})$ from the tail can be dropped. Consequently, erasing the attribute function in the head of δ^- suffices to prevent inference on $\texttt{A}(\boldsymbol{x})$ using δ^-.

For ease of presentation, we introduce the notion of *shortening* of a dependence statement and introduce some syntax. Let $\delta^+ : \texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}(\boldsymbol{x}), \texttt{A}_2(\boldsymbol{x}_2), \ldots, \texttt{A}_n(\boldsymbol{x}_n)$ be a independence statement. We refer to $\delta^- := \texttt{A}_1(\boldsymbol{x}_1) \not\perp\!\!\!\perp \texttt{A}(\boldsymbol{x}), \texttt{A}_2(\boldsymbol{x}_2), \ldots, \texttt{A}_n(\boldsymbol{x}_n)$ as the corresponding dependence statement of δ^+. We refer to the dependence statement $\texttt{A}(\boldsymbol{x}) \not\perp\!\!\!\perp \texttt{A}_1(\boldsymbol{x}_1)$ as the shortening of δ^- w.r.t. $\texttt{A}(\boldsymbol{x})$ and denote it as $shrt(\delta^-, \texttt{A}(\boldsymbol{x}))$. We want to show that including the dependence statement $\texttt{A}_1(\boldsymbol{x}) \not\perp\!\!\!\perp \texttt{A}(\boldsymbol{x})$ ensures that $\delta^+ \notin cl(\Sigma^+)$, and thus δ^+ need not be instantiated.

Example 3.7. Let $\delta_4^+ : \texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}(\boldsymbol{x}), \texttt{A}_2(\boldsymbol{x}_2)$ and the corresponding dependence statement be $\delta_4^- : \texttt{A}_1(\boldsymbol{x}_1) \not\perp\!\!\!\perp \texttt{A}(\boldsymbol{x}), \texttt{A}_2(\boldsymbol{x}_2)$. The shortening of δ_4^- w.r.t.$\texttt{A}(\boldsymbol{x})$ denoted $shrt(\delta_4^-, \texttt{A}(\boldsymbol{x}))$ is $\texttt{A}_1(\boldsymbol{x}_1) \not\perp\!\!\!\perp \texttt{A}(\boldsymbol{x})$. Observe that for the set $\Sigma^+ = \{\texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}_2(\boldsymbol{x}), \texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}_2(\boldsymbol{x})\}$ of independence statements, both $\Sigma^+ \cup \{\delta_4^-\}$ and $\Sigma^+ \cup \{shrt(\delta_4^-, \texttt{A}(\boldsymbol{x}))\}$ are consistent. In particular, neither $\Sigma^+ \cup \{\delta_4^+\} \cup \{shrt(\delta_4^-, \texttt{A}(\boldsymbol{x}))\}$ nor $\Sigma^+ \cup \{\delta_4^+\} \cup \{\delta_4^-\}$ is consistent. □

The following result shows that if the shortening of a dependence statement and a set of independence statements are consistent, then the corresponding dependence statement and the set of independence statements are also consistent. In other words, including the dependence statement $\texttt{A}_1(\boldsymbol{x}) \not\perp\!\!\!\perp \texttt{A}(\boldsymbol{x})$ ensures that $\delta^+ \notin cl(\Sigma^+)$.

Theorem 3.8 (Shortening). *Let $\boldsymbol{U}$ be a finite set of variables, Σ^+ be a set of independences on variables in $\boldsymbol{U}$, $\delta^+ := \texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}(\boldsymbol{x}), \texttt{A}_2(\boldsymbol{x}_2), \ldots, \texttt{A}_n(\boldsymbol{x}_n)$ and δ^- be the corresponding dependence statement. For every set Σ^+, s.t. $cl(\Sigma^+) \cup \{shrt(\delta^-, \texttt{A}(\boldsymbol{x}))\}$ is consistent, we have $cl(\Sigma^+) \cup \delta^-$ is consistent.*

Proof. (Outline)Since Σ^+, s.t. $cl(\Sigma^+) \cup \{\texttt{A}_1(\boldsymbol{x}_1) \not\perp\!\!\!\perp \texttt{A}(\boldsymbol{x})\}$ is consistent, the following must hold:

- $\texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}(\boldsymbol{x})$ itself is not in Σ^+.
- $\delta^+ \notin \Sigma^+$ otherwise on applying Axiom 2 would produce $\texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}(\boldsymbol{x})$
- Moreover, any independence statement of the following form $\texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}(\boldsymbol{x})... \notin \Sigma^+$ as on applying Axiom 3 would produce the statement $\texttt{A}_1(\boldsymbol{x}_1) \perp\!\!\!\perp \texttt{A}(\boldsymbol{x})$.

Due to the above, there is no independence statement such that Axiom 4 produces δ^+ or any other statements from which δ^+ can be produced. □

In the following result we show that the set of instantiated RDRs obtained by shortening is consistent.

Theorem 3.9. *Let $\boldsymbol{U}$ be a finite set of variables, Σ^+ be a set of independence statement on variables in $\boldsymbol{U}$, and $\Sigma^- = \{\delta_1^-, \ldots, \delta_p^-\}$ be a set of instantiated RDRs on the variables in $\boldsymbol{U}$ such that, for $1 \le i \le p$, each dependency is of the form $\delta_i^- = \mathsf{A}_k(\boldsymbol{x}_k) \not\perp\!\!\!\perp \mathsf{A}(\boldsymbol{x}), \ldots$. For each Σ^+ s.t. $cl(\Sigma^+) \cup \Sigma^-$ is consistent, we have $cl(\Sigma^+) \cup \{shrt(\delta_i^-, \mathsf{A}(\boldsymbol{x})) \mid \delta_i^- \in \Sigma^-\}$ is consistent.*

Proof. (Outline) Base Case: By Theorem 3.8, we have that $S(1)$ is true, i.e., if $cl(\Sigma^+) \cup \{\delta_1^-\}$ is consistent then $cl(\Sigma^+) \cup \{shrt(\delta_1^-, \mathsf{A}(\boldsymbol{x}))\}$ is consistent.
Inductive Hypothesis: Let $S(k)$ be true. That is, if $cl(\Sigma^+) \cup \{\delta_1^-, \ldots, \delta_k^-\}$ is consistent then $cl(\Sigma^+) \cup \{shrt(\delta_i^-, \mathsf{A}(\boldsymbol{x})) \mid 1 \le i \le k\}$ is consistent.
We will show $S(k) \implies S(k+1)$.
Let $cl(\Sigma^+) \cup \{\delta_1^-, \ldots, \delta_{k+1}^-\}$ be consistent. Therefore, $cl(\Sigma^+) \cup \{\delta_1^-, \ldots, \delta_k^-\} \cup \{\delta_{k+1}^-\}$ is consistent which implies $cl(\Sigma^+) \cup \{shrt(\delta_i^-, \mathsf{A}(\boldsymbol{x})\} \cup \{\delta_{k+1}^-\}$ is consistent (by inductive hypothesis).
Observe that $cl(\Sigma^+) \cup \{\delta_{k+1}^-\}$ must be consistent. By Theorem 3.8 we have that, $cl(\Sigma^+) \cup \{shrt(\delta_{k+1}^-, \mathsf{A}(\boldsymbol{x}))\}$ is consistent. Thus, $cl(\Sigma^+) \cup \{shrt(\delta_i^-, \mathsf{A}(\boldsymbol{x})\} \cup \{shrt(\delta_{k+1}^-, \mathsf{A}(\boldsymbol{x}))\}$ is consistent. □

Indirect Inference. Observe that inference of an attribute $\mathsf{A}(\boldsymbol{x})$ may also occur *indirectly* through an attribute function not in the same instantiated RDR as $\mathsf{A}(\boldsymbol{x})$. Consider the following.

Example 3.10. Continuing Example 3.6, even if the attribute function $\mathsf{A}_1(\boldsymbol{x})$ is erased, δ_2^- leads to direct inference of $\mathsf{A}_1(\boldsymbol{x})$ and then δ_1^- leads to direct inference of $\mathsf{A}(\boldsymbol{x})$. We say that the dependency δ_2^- leads to indirect inference of $\mathsf{A}(\boldsymbol{x})$. □

Note that an indirect inference of an attribute function $\mathsf{A}(\boldsymbol{x})$ comprises a series of direct inferences of other attribute functions (such as direct inference of $\mathsf{A}_1(\boldsymbol{x})$ in Example 3.10 leads to indirect inference of $\mathsf{A}(\boldsymbol{x})$). Therefore, the above results can be easily extended to indirect inferences.

3.3 P2E2 as a Set of Independence Statements

By definition, for P2E2 to hold for a cell, we need to find, given a set of instantiated RDRs (which are dependence statements), a set of independence statements that are consistent with the set of the instantiated RDRs. In this section, we discuss how to determine that set of independence statements.

For a finite set $\boldsymbol{U}$ of random variables, let δ^- be a dependence $\mathsf{A}(\boldsymbol{x}) \not\perp\!\!\!\perp \mathsf{A}_1(\boldsymbol{x}_1), \ldots, \mathsf{A}(\boldsymbol{x}_n)$ and Σ^+ be a set of independence statements. Observe that the independence statement $\delta^+ \notin cl(\Sigma^+)$. Moreover, the following independence statements cannot be in Σ^+:

(1) $\mathsf{A}_1(\boldsymbol{x}_1), \mathsf{A}_2(\boldsymbol{x}_2), \ldots, \mathsf{A}(\boldsymbol{x}_n) \not\perp\!\!\!\perp \mathsf{A}(\boldsymbol{x})$ (due to Ax. 1).
(2) Independence statements of the form
$\mathsf{A}(\boldsymbol{x}) \perp\!\!\!\perp \ldots, \mathsf{A}_1(\boldsymbol{x}_1), \mathsf{A}_2(\boldsymbol{x}_2), \ldots, \mathsf{A}(\boldsymbol{x}_n)$ or
$\mathsf{A}_1(\boldsymbol{x}_1), \mathsf{A}_2(\boldsymbol{x}_2), \ldots, \mathsf{A}(\boldsymbol{x}_n), \ldots \perp\!\!\!\perp \mathsf{A}(\boldsymbol{x})$ (due to Ax. 2).
(3) For $1 \le m \le n$ and for $j_1 \ne \ldots \ne j_n$ take values in $\{1, \ldots, n\}$, the independence statements $A(x) \perp\!\!\!\perp \mathsf{A}_{j_1}, \mathsf{A}_{j_2}, \ldots, \mathsf{A}_{j_m}$ and $A(x), \mathsf{A}_{j_1}, \mathsf{A}_{j_2}, \ldots, \mathsf{A}_{j_m} \perp\!\!\!\perp \mathsf{A}_{j_{m+1}}, \mathsf{A}_{j_{m+2}}, \ldots, \mathsf{A}_{j_n}$ (due to Ax. 3).

Thus, if at least one of the variables, for $1 \le i \le n$, $\mathsf{A}_i(\boldsymbol{x}_i)$ is removed, it is impossible to derive the independence statement δ^+. The same reasoning can be extended to a set of independence statements. If at least one variable from each dependence statement is removed from the set $\boldsymbol{U}$, no independence statements can be derived (using the axioms) that lead to an inconsistency. Next, we present algorithms which guarantee P2E2 based on the technical results in this section.

4 SUPPORTING DEMAND-DRIVEN ERASURE

In this section, we focus on demand-driven erasure. Given a set of RDRs and a cell to be erased, our goal is to identify a subset of data that needs to be deleted such that the P2E2 guarantee holds. This subset to be deleted should incur minimal costs. We begin by defining our problem more precisely.

Definition 4.1 (Optimal P2E2 problem: Opt-P2E2). Given a database state $\mathcal{D}_t$, an instantiated attribute function $\mathsf{A}(\boldsymbol{x})$ in $\mathcal{D}_t$, the set of instantiated RDRs $\Delta^-(\mathcal{D}_t)$, and a cost function $Cost : \mathcal{D} \longrightarrow \mathbb{R}$ that maps each instantiated attribute function in $\mathcal{D}_t$ to its associated cost, find a set $\mathcal{T} = \{\mathsf{A}_i(\boldsymbol{x}_i) \mid 1 \le i \le |\mathcal{T}|\}$ such that when the value of each $\mathsf{A}_i(\boldsymbol{x}_i)$ is set to *NULL*, P2E2 holds in $\mathcal{D}_{t^+}$ for $\mathsf{A}(\boldsymbol{x})$ such that $\sum_{\mathsf{A}_i(\boldsymbol{x}_i) \in \mathcal{T}} Cost(\mathsf{A}_i(\boldsymbol{x}_i))$ is minimized. □

Below we develop several approaches to address the above Opt-P2E2 problem. For all the approaches we develop, the first step is to instantiate RDRs (Sec. 4.1). We present an ILP reduction (Sec. 4.2) for the Opt-P2E2 problem which produces the optimal result for any given set of RDRs. In Sec. 4.3, we develop a hypergraph-based approach and present an approximation variant of the same in Sec. 4.4. Finally, Sec. 4.5 explores batching erasures.

4.1 Instantiating RDRs

Given a database state $\mathcal{D}_t$, and an instantiated attribute function $\mathsf{A}(\boldsymbol{x})$, we denote with $\Delta^-(\mathcal{D}_t, \mathsf{A}(\boldsymbol{x}))$ the set of all instantiated relational dependency rules (RDRs) with $\mathsf{A}(\boldsymbol{x})$ in the head or tail. Observe that to prevent *direct* and *indirect* inferences on $\mathsf{A}(\boldsymbol{x})$, we need to consider instantiated RDRs in $\Delta^-(\mathcal{D}_t, \mathsf{A}(\boldsymbol{x}))$ as well as all the instantiated attribute functions in $\bigcup_{\delta^- \in \Delta^-(\mathcal{D}_t, \mathsf{A}(\boldsymbol{x}))} Cells(\delta^-)$ besides $\mathsf{A}(\boldsymbol{x})$, and recursively so on.

For an instantiated RDR δ^- (as in Eqn. 4), to *not* lead to the inference on $\mathsf{A}(\boldsymbol{x})$, at least one of the attribute functions in the tail needs to be set to *NULL*. To suffice the definition of P2E2, we need to focus only on those attribute functions $\mathsf{A}_i(\boldsymbol{x}_i)$ that were inserted *after*, and expire after, $\mathsf{A}(\boldsymbol{x})$, more specifically, those for which $\kappa(\mathsf{A}(\boldsymbol{x})) < \kappa(\mathsf{A}_i(\boldsymbol{x})_i))$ and $\eta(\mathsf{A}(\boldsymbol{x})) \le \eta(\mathsf{A}_i(\boldsymbol{x}_i))$.

Theorem 4.2. *Given a database state $\mathcal{D}_t$, an attribute function $\mathsf{A}(\boldsymbol{x})$ for which P2E2 must hold, and set Δ^- of RDRs. Let $\Delta^-(\mathcal{D}_t)$ be the set of all instantiated RDRs. For P2E2 to hold, it suffices to consider those instantiations $\delta^- \in \Delta^-(\mathcal{D}_t)$ s.t. there exists at least one cell c in $Cells(\delta^-)$ with $\kappa(\mathsf{A}(\boldsymbol{x})) < \kappa(c)$.*

Proof. (Outline) Recall that by definition of P2E2 (Equation ??), the LHS takes into account all dependencies in the database state $\mathcal{D}_{\kappa(\mathsf{A}(\boldsymbol{x}))^-}$. Any insertion (or update) of a cell c such that

$(\kappa(\mathsf{A}(\boldsymbol{x})) < \kappa(c) < \eta(\mathsf{A}(\boldsymbol{x}))] < \eta(c)$ can lead to a RDR be instantiated at time $\eta(\mathsf{A}(\boldsymbol{x}))$. If $\eta(c) < \eta(\mathsf{A}(\boldsymbol{x}))$ then the cell is not present in any instantiated RDR at time $\eta(\mathsf{A}(\boldsymbol{x}))$. Therefore, it sufficec to consider only those RDRs which have at least once cell c such that $\kappa(\mathsf{A}(\boldsymbol{x})) < \kappa(c)$. □

Algorithm 1 Instantiating RDRs for P2E2

```
1: procedure DepInst($\mathcal{D}_t, \Delta^-, \mathsf{A}(\boldsymbol{x})$)
2:     $Q \leftarrow \{\mathsf{A}(\boldsymbol{x})\}$                    ▷ Queue of cells
3:     $\mathcal{V} \leftarrow \emptyset$                              ▷ List of instantiated attributes
4:     $\mathcal{I} \leftarrow \emptyset$                              ▷ List of instantiated RDRs
5:     while $Q \neq \emptyset$ do
6:         $attf \leftarrow Q.pop$
7:         if $attf \notin \mathcal{V}$ then
8:             $\mathcal{V} \leftarrow \mathcal{V} \cup attf$
9:             $Rules \leftarrow \Delta^-(attf)$
10:            for $rule \in Rules$ do
11:                if $attf \in rule.head$ then
12:                    $\delta^- \leftarrow eval(rule)$
13:                else
14:                    $\delta^- \leftarrow eval(short(rule, attf))$
15:                $\mathcal{I} \leftarrow \mathcal{I} \cup \{\delta^-\}$
16:                for $tail \in Tail(\delta^-)$ do
17:                    $Q.push(tail)$
18:    return $\mathcal{V}, \mathcal{I}$
```

The instantiated RDRs from the set $\Delta^-(\mathcal{D}_t)$, which need to be considered for P2E2 to hold w.r.t. to an instantiated attribute function $\mathsf{A}(\boldsymbol{x})$, is denoted as $\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))$. Given $\mathsf{A}(\boldsymbol{x})$ and a database $\mathcal{D}_t$, Algorithm 1 generates the set $\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))$ of instantiated RDRs. The algorithm iteratively instantiates RDRs corresponding to direct or indirect inference of $\mathsf{A}(\boldsymbol{x})$. The Instantiate function (lines 1-7) evaluates the condition of the rule and returns the instantiations that satisfy the condition in Theorem 4.2. If the attribute function to be erased (*attrf*) is in the head of the rule (*rule.head*) then instantiation is straightforward; if the attribute function is in the tail of the rule, then it returns the shortened instantiation of the rule (Theorem 3.8). Note that we adopt the convention that when instantiating RDRs for an attribute function $\mathsf{A}(\boldsymbol{x})$, we always take $\mathsf{A}(\boldsymbol{x})$ to be the head of the instantiated RDR.

Theorem 4.3. *For an attribute function $\mathsf{A}(\boldsymbol{x})$, and a database state $\mathcal{D}_t$, let $\Delta^-_H(\mathsf{A}(\boldsymbol{x}))$ (and, $\Delta^-_T(\mathsf{A}(\boldsymbol{x}))$) be a set of dependence statements such that for each $\delta^- \in \Delta^-(\mathsf{A}(\boldsymbol{x}))$, the attribute function $\mathsf{A}(\boldsymbol{x})$ appears either in the head (and, tail, respectively). For all Σ^+, s.t. $cl(\Sigma^+) \cup \Delta^-_H(\mathsf{A}(\boldsymbol{x})) \cup \Delta^-_T(\mathsf{A}(\boldsymbol{x}))$ is consistent, we have $cl(\Sigma^+) \cup \Delta^-_H(\mathsf{A}(\boldsymbol{x})) \cup \{shrt(\delta^-, \mathsf{A}(\boldsymbol{x})) \mid \delta^- \in \Delta^-_T(\mathsf{A}(\boldsymbol{x}))\}$ is consistent.*

Proof. (Outline) Follows directly from Theorems 3.9 and 4.2. □

4.2 ILP-Approach

We present a reduction from the Opt-P2E2 problem to integer linear programming (ILP). To provide an intuition of the reduction, we introduce some concepts and notation.

Definition 4.4 (Induced Bipartite Graph). For a cell $\mathsf{A}(\boldsymbol{x})$ in a database state $\mathcal{D}_t$, given the set $\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x}))) = \{\delta^-_1, \ldots, \delta^-_n\}$ of instantiated RDRs, we define the induced bipartite graph $\mathcal{B}(\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))) = (V = V_L \cup V_R, E = E_H \cup E_T)$ where:

- $V_L = \{\delta^-_i | 1 \leq i \leq n\}$ and $V_R = \{c_j \mid c_j \in Cells(\delta^-_i)\}$ are the bipartition of the vertex set V.
- The set $E = E_H \cup E_T$ of edges contain, for every $\delta^-_i = c_{i_1} \not\perp\!\!\!\perp c_{i_2}, \ldots, c_{i_{n_i}}$, an edge $(\delta_i, c_{i_1}) \in E_H$ and for, $2 \leq j \leq n_i$, edges $(\delta_i, c_{i_j}) \in E_T$.

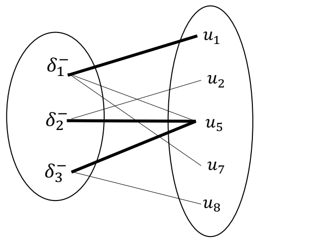


Figure 2: Example of a bipartite graph.

Fig. 2 shows the bipartite graph for the set $\{\delta^-_1 : c_1 \not\perp\!\!\!\perp c_5, c_7,\ \delta^-_2 : c_5 \not\perp\!\!\!\perp c_2,\ \delta^-_3 : c_5 \not\perp\!\!\!\perp c_8\}$ of instantiated RDRs.

Reduction. Consider, for a database state $\mathcal{D}_t$, a cell $\mathsf{A}(\boldsymbol{x})$, denoted with c_d, such that P2E2 must hold for c_d, and, for the set $\Delta^-(\mathcal{D}_t, \mathsf{A}(\boldsymbol{x}))$ of instantiated RDRs, the induced bipartite graph $\mathcal{B}(\Delta^-(\mathcal{D}_t, \mathsf{A}(\boldsymbol{x}))) = (V_L \cup V_R, E_H \cup E_T)$. We introduce the following variables: for $\delta_i \in V_L$ a binary variable b_i; for each $c_j \in V_R$, a binary variable a_j; for each edge $(\delta_i, c_j) \in E_H$, a binary variable h^j_i; for each edge $(\delta_i, c_j) \in E_T$ a binary variable t^j_i.

We specify the constraints in the following.

1. For P2E2 to hold for unit c_d, it must be erased. This is ensured by the constraint $a_d = 1$.
2. For each unit c_j, where $1 \leq j \leq m$, that needs to be erased, all instantiated RDRs where c_j is in the head can be used to infer it. To prevent this, we need to address the instantiated RDR. This is stated, for all i, using the constraints $a_j = h^j_i$.
3. For each instantiated RDR δ_i, if the unit in its head is hidden, then we set $b_i = h^i_j$.
4. For a instantiated RDR δ_i, where $1 \leq i \leq n$, to prevent the inference of the unit in its head, a unit from the tail has to be erased. So we introduce the constraint $\sum_{j \in \{i_1, \ldots, i_{n_i}\}} t^j_i \geq b_i$.
5. For each unit c_j, for $1 \leq j \leq m$, if the unit is erased, then all of the tail edges incident on it must indicate so. This is ensured by the constraint, for all $1 \leq i \leq n$, we have $a_j = t^j_i$.
6. To minimize the number of units erased, we introduce the constraint
$$W = \min \sum_{j=1}^{m} a_j. \tag{6}$$

Let O be the preceding ILP instance. Note that O has $O(nm)$ binary variables and $O(mn)$ constraints. Observe that for a cost model where, for $1 \leq j \leq m$, the cost of erasing cell c_j is $Cost(j) \in \mathbb{N}$, changing the function to $W = \min \sum_{j=1}^{m} c_j Cost(j)$ minimizes the total cost of P2E2-guarantee. The optimization problem can be solved with any off-the-shelf optimizer.

We want to show that a 0-1 solution to O with $W = w$, where $w \in \mathbb{N}$, is the optimal number of cells that need to be erased to ensure that P2E2 holds for c_d.

Theorem 4.5. *Let $w \in \mathbb{N}$. The following statements are equivalent:*

- *The minimum number of cells to be erased in $\mathcal{D}_t$ such that P2E2 holds for c_d is w.*
- *The system O has a 0-1 solution with $W = w$.*

Proof. (Outline) Let the minimum number of cells to be erased in $\mathcal{D}_t$ such that P2E2 holds for c_d be w. Since c_d is erased, the variable $a_d = 1$. For all instantiated RDR δ_i^-, where $c_d \in Head(\delta_i^-)$, one cell in $Tail(\delta_i^-)$ must be erased, otherwise, it will lead to direct inference of c_d. Therefore, it holds that for all i, we have $h_i^d = 1 = a_d$, the variables $b_i = 1$ and $t_i^d = 1 \geq b_i$. For each instantiated RDR δ_i, the variable corresponding to the cell erased must be 1. This reasoning follows for all the cells that are set to *NULL* for P2E2 of c_d to be guaranteed. Thus, the equations in system O is satisfied and $W = w$.
For the other direction, assume that the system of equations defined above has a 0-1 solution with $W = w$. By the first constraint, $a_d = 1$. Therefore c_d is erased. For each edge (δ_i, a_d), the variable $h_i^d = 1$ (constraint 2) and by constraint 3, for all δ_i, we have $b_i = 1$. By constraint 4, since W is minimized, for all edges from a_d, we have $t_i^d = 1$. Since constraint 5 is satisfied, it implies that the variable a_j corresponding to the cell c_j in the tail of each δ_i is set to 1, i.e., are deleted. This reasoning extends to all instantiated RDRs and cells that lead to indirect inference of c_d. Since once cell is deleted from each instantiated RDR that lead to direct or indirect inference of c_d, P2E2 of c_d is guaranteed as any subset of the cells in $\mathcal{D}_t$ are either independent of c_d, or, if not, they were in the state $\mathcal{D}_{\kappa(c_d)^-}$. By constraint 6, we have that w cells are deleted, therefore the minimum number of cells to be erased in $\mathcal{D}_t$ is w.

□

4.3 Dependence Hypergraph

Here, we present a new approach to solve Opt-P2E2. Given a set of instantiated RDRs $\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))$ we construct a *dependence hypergraph* where the vertices are cells and a hyperedge connects, for an instantiated RDR δ^-, its head $Head(\delta^-)$ to its tail $Tail(\delta^-)$.

Definition 4.6 (Dependence Hypergraph). For a cell $\mathsf{A}(\boldsymbol{x})$ in a database state $\mathcal{D}_t$, given the set $\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))$ of instantiated RDRs, we define the *dependence hypergraph* $\mathcal{H}(\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x}))))$ $= (V, E)$ where $V = \bigcup_{\delta_i^- \in \Delta^-(\mathcal{D}_t)} Cells(\delta_i^-)$ is the set of vertices and $E = \{(Head(\delta_i^-), Tail(\delta_i^-)) \mid \delta_i^- \in \Delta^-(\mathcal{D}_t)\}$ is the set of edges.

For a dependence hypergraph $\mathcal{H}(\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))) = (V, E)$, a vertex $v \in V$ is called a *root* if v is not in the tail of any instantiated RDR, i.e., for all $\delta_i \in \Delta^-(\mathcal{D}_t)$ we have $v \notin Tail(\delta_i)$. Similarly, a vertex $v \in V$ is called a *leaf* if v is not the head of any instantiated RDR, i.e., for all $\delta_i \in \Delta^-(\mathcal{D}_t)$ we have $v \notin Head(\delta_i)$.

Example 4.7. Fig. 3 shows the dependence hypergraph for the set $\{\delta_1^- : c_1 \not\perp\!\!\!\perp c_5, c_7;\ \delta_2^- : c_5 \not\perp\!\!\!\perp c_2;\ \delta_3^- : c_5 \not\perp\!\!\!\perp c_8;\ \delta_4^- : c_1 \not\perp\!\!\!\perp c_9, c_{10};\ \delta_5^- : c_{10} \perp\!\!\!\perp c_{11};\ \delta_6^- : c_{11} \perp\!\!\!\perp c_{12}\}$ of instantiated RDRs. □

Given a set Δ^- of RDRs, a database state $\mathcal{D}_t$, an instantiated attribute function $\mathsf{A}(\boldsymbol{x})$, Algorithm 1 can be easily adapted to construct a dependence hypergraph. We define paths and complete paths in a dependence hypergraph to characterize the P2E2-guarantee.

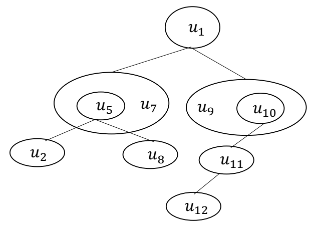


Figure 3: Example of a dependence hypergraph.

Definition 4.8. For an dependence hypergraph $\mathcal{H}(\mathcal{D}_t)$, and a sequence $P : v_1, v_2, \ldots, v_n$ of vertices we define the following.

- We say that the sequence P is a *path* if the following hold:
  (1) There exists δ_i s.t. $v_1 \in Head(\delta_i)$.
  (2) For $1 \leq i < n$, there exists δ_i s.t.
  $v_i \in Head(\delta_i)$ and $v_{i+1} \in Tail(\delta_i)$
- We say that a sequence $P' : v_b, \ldots, v_e$ where $1 \leq b \leq e \leq n$, is a *subpath* of P if P and P' are both paths. A path is, trivially, a subpath of itself.
- We say that the sequence P is a *complete (sub-)path* if P is a (sub-)path such that v_n is a leaf.

Example 4.9. In Fig. 3, the vertex c_1 is a root, vertices c_7 and c_8 are leaves; the sequence $P_1 : c_1, c_{10}, c_{11}$ is a path; the sequence $P_2 : c_1, c_5, c_2$ is a complete path; $P_3 : c_5, c_2$ is a subpath of P_2. □

The following result characterizes, for a cell c_d, the dependent cells which need to be set to *NULL* to ensure that P2E2 holds for c_d. Recall that the notion of a vertex cover of a hypergraph [27] is a set C of vertices such that every edge of the graph constrains a vertex in the set C. Observe that for P2E2 to hold, we do not need find a cover of the dependence hypergraph. While setting the vertices in a cover of the dependence hypergraph to *NULL* is sufficient to ensure that P2E2 holds for c_d, it is not a necessary condition (and suboptimal in that the cover contains vertices which do not need to be set to *NULL* for P2E2 to hold).

Theorem 4.10. *For a database state $\mathcal{D}_t$, cell $c_d \in \mathcal{D}_t$, and a set Δ^- of RDRs, let $\mathcal{H}(\Delta^-(\mathcal{D}_t, P2E2(c_d))) = (V, E)$ be a dependence hypergraph with c_d as its root, let $\mathcal{P}$ be a set of complete paths in $\mathcal{H}(\Delta^-(\mathcal{D}_t, P2E2(c_d)))$, and let $\mathcal{T} = Cells(\mathcal{P})$. P2E2 holds for c_d when all such units c in $\mathcal{T}$ are set to NULL if the following holds:*

- *There exists a complete path in $\mathcal{P}$ from c_d.*
- *For all $v \in V$ if v is in a complete path, then for each edge $(v, T) \in E$, there exists $P \in \mathcal{P}$ such that $P : v, v_1, \ldots, v_e$ is a complete subpath where $v_1 \in T$.*

Proof. (Outline) Let $P := c_d, v_2, \ldots, v_n$ be a complete path in $\mathcal{P}$ from c_d. Without loss of generality, assume that only vertex v_i has an edge (v_i, T_i). Let $P' : v_1', \ldots, v_e'$ be a compete subpath such that $v_1' \in T$ and P' in $\mathcal{P}$. Since $\{c_d, v_2, \ldots, v_n, v_1', \ldots, v_e'\} \subseteq \mathcal{T}$, they are all set to *NULL*.
Let the instantiated RDRs in the state $\mathcal{D}_{\kappa(c_d)^-}$ be Δ_1 and that inthe state $\mathcal{D}_{\eta(c_d)^+}$ be Δ_2. By construction (Algorithm 1), if any cell c has

$\kappa(c) > \kappa(c_d)$ and it is in any instantiated RDR δ_j^- that directly or indirectly leads to the inference of c_d then $\delta_j^- \in (\Delta^-(\mathcal{D}_t, P2E2(c_d)))$. Therefore, for each instantiated RDR $\delta_j^- \in (\Delta^-(\mathcal{D}_t, P2E2(c_d)))$, that leads to direct or indirect inference of c_d, there exists at least one cell is set to *NULL*. Therefore, there exists no instantiated RDR δ^- such that $\delta^- \in \Delta_2$ with each cell $c \in Val(c, \mathcal{D}_{\eta(c_d)^+}) \neq NULL$ and $\delta^- \notin \Delta_1$. In other words, all possible subsets of cells in the state $\mathcal{D}_{\eta(c_d)^+}$ are either independent of c_d or, if any subset is dependent on c_d, then it is also a subset in the state $\mathcal{D}_{\kappa(c_d)^-}$. So it must be the case that $Pr(Val(c_d) = val \mid \mathcal{D}_{\kappa(c_d)^-}) = Pr(Val(c_d) = val \mid \mathcal{D}_{\eta(c)^+})$ and thus P2E2 holds for c_d. □

Optimization. Given a dependence hypergraph, Algorithm 2 produces an optimal solution. We assume that the graph is cycle-free. This is necessary only for proving optimality [5]. For a hypergraph, with an attribute function $\mathsf{A}(\boldsymbol{x})$ as its root, a bottom-up traversal is required to determine the minimum cost for P2E2 to hold for $\mathsf{A}(\boldsymbol{x})$.

Algorithm 2 Optimizing the Dependence Hypergraph

```
1:  procedure OptPath($\mathcal{V}, \mathcal{I}$)                ▷ Output of Algorithm 1
2:      $Q \leftarrow \{leafs(\mathcal{V})\}$                    ▷ Queue of cells
3:      $\mathcal{S} \leftarrow \emptyset$                       ▷ Set of seen cells
4:      while $Q \neq \emptyset$ do
5:          $attf \leftarrow Q.pop$
6:          if $attf \notin \mathcal{S}$ then
7:              $\mathcal{S} \leftarrow \mathcal{S} \cup \{attf\}$
8:              $Cost(attf) \leftarrow 1$                        ▷ Initialize node cost
9:              $Rules \leftarrow \mathcal{I}(attf)$             ▷ All RDRs that contain attf
10:             for $\delta^- \in Rules$ do
11:                 if $attf \in Head(\delta^-)$ then
12:                     $Sum(Cost(attf), min\ Cost(Tail(\delta^-)))$ ▷
                        Add cheapest node of tail to node's cost
13:                 else
14:                     $Q.push(Head(\delta^-))$          ▷ Add RDR head to queue
15:     $\mathcal{T} \leftarrow \{\mathsf{A}(\boldsymbol{x})\}$   ▷ Set of cells to delete
16:     $Q \leftarrow \{\mathsf{A}(\boldsymbol{x})\}$
17:     $\mathcal{S} \leftarrow \emptyset$
18:     while $Q \neq \emptyset$ do
19:         $attf \leftarrow Q.pop$
20:         if $attf \notin \mathcal{S}$ then
21:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{attf\}$
22:             $Rules \leftarrow \mathcal{I}(attf)$
23:             for $\delta^- \in Rules$ do
24:                 if $attf \in Head(\delta^-)$ then
25:                     $child \leftarrow arg\,min\ Cost(Tail(\delta^-))$
26:                     $\mathcal{T} \leftarrow \mathcal{T} \cup child$
27:                     $Q.push(child)$ ▷ Delete cheapest node in tail
                        and continue traversal
28:     return $\mathcal{T}$
```

In practice, we perform the following procedure. The cost of every node is set to 1, as a base cost to erase a single attribute function (Line 8). The *leafs* cannot incur a higher cost, as they do not cause additional erasures. Next, we traverse the tree upwards and compute the cost of every *inner node* as the sum of the nodes' cost and the minimal cost of every attached hyper edge (Line 12). When reaching the root node, we construct the complete path by traversing the tree to the bottom and always choosing the node with minimal cost (Line 25). Thus, the algorithm guarantees P2E2 at minimal cost but has to traverse the tree twice.

[5]If cycles are present, our approach produces a correct solution but not necessarily an optimal one. To ensure that cycles are not present in the hypergraph, for all pairwise instantiated dependencies δ_1^- and δ_2^-, if $Tail(\delta_1^-) \cap Tail(\delta_2^-) \neq \emptyset$, discard the instantiated RDR with the larger number of attribute functions in its tail. The solution produced by Algorithm 2 is optimal with respect to the thus obtained set of RDRs.

Theorem 4.11. *When Algorithm 2 terminates, the set $\mathcal{T}$ contains cells which, when set to NULL, guarantees P2E2 for the input $\mathsf{A}(\boldsymbol{x})$. Moreover, $\sum_{\mathsf{A}_i(\boldsymbol{x}_i) \in \mathcal{T}} Cost(\mathsf{A}_i(\boldsymbol{x}_i))$ is minimized.*

Proof. (Outline) The proof consists of two parts. First, we will show that when Algorithm 2 terminates, the set $\mathcal{T}$ contains cells which when set to *NULL* guarantees P2E2 for the input $\mathsf{A}(\boldsymbol{x})$. Then we will show optimality.

For a database state $\mathcal{D}_t$, cell $c_d = \mathsf{A}(\boldsymbol{x}) \in \mathcal{D}_t$, and the set $\mathscr{I}$ of instantiated RDRs (produced using Algorithm 1), let $\mathcal{H}(\mathcal{I}) = (V, E)$ be a dependence hypergraph with c_d as its root. Lines 22-35 traverses the tree from the leaf nodes, bottom up, to the root. Observe that for every cell (*attrf*), all instantiated RDRs in $\mathcal{I}$ have one cell in its tail in the set $\mathcal{T}$. Therefore, these cells form a complete path $P : v_1, \ldots, v_n$ such that for all $v \in \mathcal{P}$, for each edge (v, T), cells comprising a complete subpath $P : v, v_1, \ldots, v_e$ where $v_1 \in T$, are also present in $\mathcal{T}$. By Theorem 4.10, P2E2 guarantee for $c_d = \mathsf{A}(\boldsymbol{x})$ holds.

Optimality: In this proof, for a node v in the hypergraph, $Cost(v)$ refers to that computed in Line 12 of Algorithm 2. For contradiction, assume that there exists a set $\mathcal{T}'$ of cells with $\mathcal{T} \neq \mathcal{T}'$, such that $\sum_{c \in \mathcal{T}'} Cost(c) < \sum_{c \in \mathcal{T}} Cost(c)$ and when cells in $\mathcal{T}'$ are set to *NULL*, they guarantee P2E2 for c_d. Let the complete path corresponding to $\mathcal{T}$ be $P : c_d, v_2, \ldots, v_n$ and that corresponding to $\mathcal{T}'$ be $P' : c_d, v_2', \ldots, v_m'$. There exists an index k such that for all $2 \leq i < k$, we have $v_i = v_i'$, node $v_k \neq v_k'$. This implies, that $Cost(v_k') < Cost(v_k)$. Since $v_k \in \mathcal{T}$, it must be the case due to Line 29, that $Cost(v_k)$ is the minimum. Therefore $Cost(v_k') > Cost(v_k)$ which is a contradiction. □

4.4 Approximate Algorithm

In this section, we present an approximation variant of Algorithm 2, which, given a cell, determines a (not necessarily the smallest) set of dependent cells to delete for P2E2 to hold.

We adapt the algorithm to determine the minimum cost of guaranteeing P2E2 (Algorithm 2) such that instead of constructing the entire dependence hypergraph and then traversing it bottom-up, a partial top-down construction of the dependence hypergraph is sufficient: whenever there is an instantiated RDR that has more than two attribute functions, we only instantiate the next *level* in the tree and choose the one that has lowest cost to erase. The other attribute functions are not instantiated fully, thereby saving time. Moreover, the hypergraph is traversed only once (top-down as it is being constructed). Observe that greedily choosing the attribute function with the lowest cost to erase may, later on, force the erasure of another attribute function that incurs a high cost.

4.5 Batching Erasures

The approaches discussed until now focus on the erasure of one cell. Since regulatory data erasure requirements allow for a reasonable delay between the time at which the data is requested to be erased and the actual erasure of the data (referred to as grace period and denoted with Γ), it is possible to batch data erasures. The grace period can be used to batch multiple data erasure requests and instead of constructing and solving an individual optimization model for each cell, we attempt to construct models that allow for multiple cells to be erased such that the P2E2 holds for each of them.

Intuitively, we instantiate RDRs for the cells to be erased, which maximizes the possibility that the corresponding dependence hypergraphs have shared vertices. In practice, over a Γ period of time, we collect all cells that have to be erased such that P2E2 holds for them. Let this set of cells be S. We instantiate RDRs for each cell $s \in S$ at a time. Whenever an instantiated RDR corresponding to s contains a dependent cell s' also in S, we mark it to be set to *NULL* and only instantiate the RDRs for s'. This not only reduces the number of RDRs instantiated and, thus, the number of leafs in the tree but also the time taken to traverse the tree. Moreover, fewer models (ILP or hypergraphs) need to constructed and optimized.

5 RETENTION-DRIVEN ERASURE

So far we have considered demand-driven erasures (an user wants to erase a cell c before its expiration time $\eta(c)$) and batching such erasures. Now we turn to retention-driven erasure (where cell c is erased at its preset $\eta(c)$). For such erasures, we investigate how to minimize the overheads of P2E2 on derived data. For base data, we adopt the batching approach discussed in Sec. 4.5.

Guaranteeing P2E2 for cells, often requires additional and potentially undesirable update and reconstruction of derived data.

Example 5.1. Suppose, for a derived cell c, with the parameter $freq(c) = 1hr$, depends on cells $c_1, c_2,$ and c_3. It is reconstructed at 1pm, 2pm, 3pm, and so on. The cells $c_1, c_2,$ and c_3 expire at 1:30pm, 3:00pm, and 4:30pm, respectively. To guarantee P2E2 for the dependent cells, cell c needs to be reconstructed at 1pm, 1:30pm, 2:30pm, 3pm, 4pm, and at 4:30pm thus incurring additional overheads.

Retention-driven erasures offer an opportunity to reduce additional reconstructions due to P2E2 by exploiting the already known expiration times. We present an algorithm that, given a derived cell c and its dependencies $c_1, \ldots, c_n$ with corresponding erasure time intervals[6] $(\eta_1^b, \eta_1^e), (\eta_2^b, \eta_2^e), \ldots, (\eta_n^b, \eta_n^e)$, determines an erasure schedule $Sch(c)$ that takes into account when derived data has to be erased while maintaining the invariant that c is reconstructed at least once every $freq(c)$. Intuitively, we progressively build the reconstruction schedule $Sch(c)$ by determine the maximum overlap between the retention periods of the dependent data to minimize the number of extra reconstructions due to P2E2.

Our algorithm (Algorithm 3) is in two parts: Step 1 (Lines 1-11) creates the reconstruction schedule $Sch(c)$ of the cell c, and Step 2 (Lines 12-18) updates the schedule when required to ensure that newly inserted dependencies are accounted for. At any given time, *depSet* for a derived cell c denotes the set $\{(c_i, \eta_i^b, \eta_i^e) \mid 1 \leq i \leq n\}$ of all its dependencies, their insertion time, and their erasure time, respectively. The MAXOVERLAP function is a standard algorithm to find the maximum overlap given a set of time intervals.

Step 1: The first step finds a reconstruction time ρ that maximizes for $1 \leq i \leq n$ the overlap between the erasure time intervals (η_i^b, η_i^e) of the dependent cells, and the time interval $(\kappa(c), \kappa(c) + freq(c))$ in which c must be reconstructed at least once. Therefore, P2E2 holds for any cell c_i where $\rho \in (\eta_i^e, \eta_i^e)$. The algorithm iteratively finds the maximum overlap and creates a list $Sch(c) : \rho_1, \ldots, \rho_m$ of reconstruction times for c such that P2E2 holds for its dependencies in the database at the time of the construction of the schedule.

Step 2: The update procedure is called when a derived cell c is reconstructed. It checks whether there exists a dependent cell c_i that has to be erased after the current reconstruction but before the next scheduled reconstruction. If it does, then a new reconstruction schedule is created using the first step described above. Observe that if the erasure time of any dependent cell c_i is updated, this step (Step 2) ensures that P2E2 holds.

Algorithm 3 Reconstruction Scheduler

```
1:  procedure CREATESCHEDULE($c$, $freq(c)$, $depSet$)
2:      $\kappa(c) \leftarrow time.now()$
3:      $Sch[0] \leftarrow$ MAXOVERLAP($depSet, \kappa(c) + freq(c)$)
4:      $depSet \leftarrow depSet \setminus \{c_i \mid Sch[0] \in (\eta_i^b, \eta_i^e)\}$
5:      $i \leftarrow 1$
6:      while $depSet \neq \emptyset$ do
7:          $Sch[i] \leftarrow$ MAXOVERLAP($depSet, Sch[i-1] + freq(c)$)
8:          $depSet \leftarrow depSet \setminus \{c_i \mid Sch[0] \in (\eta_i^b, \eta_i^e)\}$
9:          $i \leftarrow i + 1$
10: procedure UPDATESCHEDULE($c$, $freq(c)$, $Sch$, $depSet$)
11:     for $(c_i, \eta_i^b, \eta_i^e) \in depSet$ do
12:         if $\eta_i^b > Sch[0] \wedge \eta_i^e < Sch[1]$ then
13:             CREATESCHEDULE($c$, $freq(c)$, $depSet$)
```

The following theorem shows the correctness of Algorithm 3, that it guarantees P2E2 for derived cells and their dependencies.

THEOREM 5.2. *Let c be a derived cell, which needs to be reconstructed at least once in $freq(c)$, and has dependencies $c_1, \ldots, c_n$ with erasure time intervals (η_i^b, η_i^e), for $1 \leq i \leq n$. The reconstruction schedule $Sch(c) = [\rho_1, \rho_2, \ldots, \rho_m]$ such that c is erased and reconstructed at time ρ_j, generated in Algorithm 3 ensures the following:*

- *For all $1 \leq j < m$, we have $\rho_{j+1} - \rho_j \leq freq(c)$.*
- *For all $1 \leq i \leq n$, there exists $\rho_j \in Sch(c)$ s.t. $\eta_i^b \leq \rho_j \leq \eta_i^e$.*

PROOF. (Outline) The first claim follows directly from Lines 3 and 7. The *freq(c)* parameter in the MAXOVERLAP function call ensures that the difference between two consecutive reschedule time is is no more than $freq(c)$.
Initially, when the schedule is created for the first time, Lines 3 and Lines 7-8 iteratively determines the schedule such that for each cell dependent cell c_i, there exists a $\eta_i^b \rho_j \leq \eta_i^e$. The UPDATESCHEDULE procedure (Lines 12-18) is run every time there is a reconstruction where Line 14 explicitly forces a schedule to e recreated if there is a cell c_k such that there is no $\eta_k^b \leq \rho_j \leq \eta_k^e$.

[6]Erasure time interval refers to the time interval in which a cell has to be erased. Usually $\eta_i^b + \Gamma = \eta_i^e$. However, here we allow for cells to have different time intervals in which they must be erased.

Table 1: Summary of the dataset characteristics

Dataset	# Cells	# RDRs	# Unique attributes
𝕏 (Twitter)	21 926 096	11	18
Tax	16 000 000	8	13
SmartBench	94 424	5	8

□

6 EVALUATION

In this section, we evaluate our approaches for guaranteeing P2E2. We compare the ILP approach (Section 4.2) with the graph-based algorithms presented in Section 4.2 (referred to as Alg 2) and its approximate version from Section 4.4 (called Approx). We analyze the efficiency and effectiveness of all three algorithms when applied to individual demand-driven erasures as well with a set of erasures using our batching method presented in Section 4.5 (referred to as Batch). Additionally, we investigate our approach for retention-driven erasures (Algorithm 3, referred to as Scheduler).

6.1 Experimental Setup

All experiments were run on an Ubuntu-based (20.04 LTS) server (Intel Xeon E5-2650; RAM: 256 GB). All algorithms are single-threaded, running on Java 11 and the datasets are stored in a PostgreSQL (v12.20) database. The ILP approach uses the Gurobi (v11.03) solver.

Datasets. Table 1 shows the characteristics of the three datasets we used to evaluate our approaches. The last column depicts the number of unique attributes in the RDRs.

(1) *Twitter.* This dataset is a subset of tweets posted on 𝕏 (formerly Twitter)[7] that represents a real-world instance of our running example. We enriched the collection of posts by maintaining aggregates of statistics, e.g., likes or replies. The RDRs in this dataset link the individual posts of users to their profiles. Thereby, erasing a derived statistic may require the processing of multiple posts.

(2) *Tax* [13]. This synthetic dataset uses the real-world distribution of values of American tax records. The RDRs include the conditional functional dependencies used in the original publication.

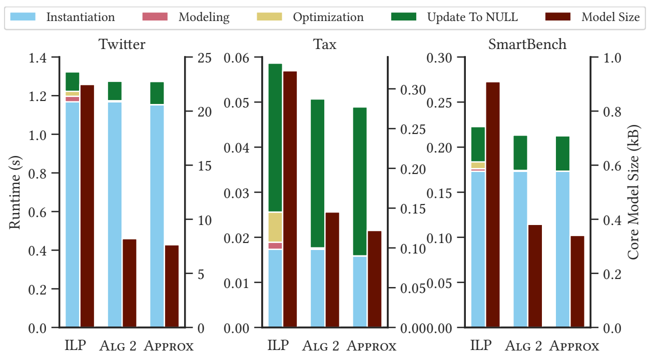
(3) *SmartBench* [22]. This is a synthetic reconstruction of the collection of multiple sensors deployed in a smart campus. RDRs capture dependencies between multiple physical sensors which are used to compute derived metrics, e.g. occupancy from Wi-Fi AP locations. The set of RDRs for each dataset is cycle free. Some RDRs in the Twitter and SmartBench dataset join on non-key columns. To speed up the instantiations, we index those columns separately.

Metrics. We measure the (i) total number of deletions, (ii) time taken, and (iii) space overheads to guarantee P2E2. For demand-driven erasure, we also measure the number of reconstructions.

Workload. Given the lack of suitable deletion benchmarks [43], we evaluate demand- and retention-driven erasures separately, as well as varying combinations of each.

6.2 Experiments

Experiment 1. To test the cost of demand-driven erasures, we erase 100 random cells for each unique attribute in the RDRs. We depict the average runtime and model size for a single erasure in Fig. 4a. The total runtime is divided into the following four steps: (i) RDR instantiation (Algorithm 1); (ii) model construction (graph construction in Alg 2 and Approx and defining the ILP instance); (iii) model optimization (traversing the graph and keeping track of the minimal cost erasure for Alg 2 and solving the ILP; there is no optimization phase in Approx, as it greedily chooses the next edge to process); (iv) update to *NULL* (which modifies the database to guarantee P2E2.) We measure the average number of instantiated and deleted cells beyond the initial erased cell. We present them in Table 4b.



(a) Average runtime and space overhead

Dataset	Instantiated cells			Deleted cells		
	ILP	Alg 2	Approx	ILP	Alg 2	Approx
X (Twitter)	135.79	135.79	135.29	48.83	48.83	67.74
Tax	3.54	3.54	3.38	2.85	2.85	2.85
SmartBench	9.23	9.23	9.23	8.99	8.99	8.99

(b) Average number of instantiated and deleted cells

Figure 4: Evaluation of demand-driven erasures

In comparison, the complexity of the Twitter dataset becomes apparent. While it is not significantly larger than the Tax dataset, erasing a cell is more than an order of magnitude more expensive. Interestingly, erasure takes longer on the smaller SmartBench dataset than on the Tax dataset. P2E2 erasure is less dependent on the dataset size, but sensitive to the amount of related data, i.e., the number of instantiated cells. In the Twitter and SmartBench datasets, the instantiation consequently takes the most amount of time, whereas the final update step is proportionally costlier in the Tax dataset.

We observe that the ILP approach has the highest overheads (runtime and memory) for all three datasets. However, it is optimal in that it guarantees P2E2 using a minimum set of additionally deleted cells. Likewise, Alg 2 always produces an optimal result, but consumes significantly less memory. Both approaches need to instantiate all available RDRs for all applicable cells, so their instantiation time is identical. However, the model construction and optimization overhead for Alg 2 is negligible compared to the ILP approach. Approx instantiates fewer cells, so it outperforms the other approaches in all datasets. Since it does not keep track of an erasure cost, it also consumes less memory. However, it cannot

[7] https://huggingface.co/datasets/enryu43/twitter100m_tweets

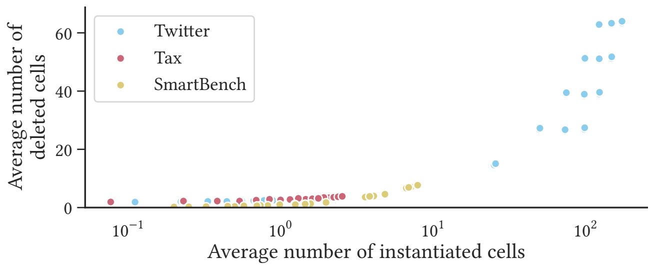


Figure 5: Impact of dependencies (log-axis) on P2E2 overhead

guarantee optimality for its result set, as exemplified in the Twitter dataset. Due to the additional update effort of Approx, Approx and Alg 2 take almost the same amount of time (only 1.39ms difference).

Experiment 2. To investigate the influence of the degree of dependence in the data on the number of deleted cells, we conducted the following experiment. To measure the degree of dependence in the data, we count the number of cells that are part of instantiated RDRs. For each dataset, we randomly sample one hundred erasures. Each sampled erasure is processed given every subset of RDRs. In Fig. 5, we plot the average number of instantiated cells compared to the average number of deleted cells. Intuitively, the more dependent cells a cell has, the more needs to be deleted. However, we observe differences in the scaling behavior. For less instantiated cells, the number of necessary deletes scales significantly slower than for more dependent cells. In more connected RDRs, separate RDRs are linked together by shared cells and cause more deletion overhead. We observe that the Twitter dataset is the most connected. On the right-hand side of the plot, some points require the same amount of deletes, but have a different number of instantiated nodes. This behavior occurs because the underlying RDRs have the same head but different tail lengths. As we need to delete only one element per tail, the number of instantiated cells increases if an RDR has many attribute functions in its tail, but not the number of necessary deletes.

Experiment 3. We evaluate the impact of Batch, which exploits the grace period Γ to batch as many erasures as possible. First, we batch erasures based on a time interval. The actual number of erasures in such a batching strategy are workload dependent. We further create batches based on the fixed number of erasures to study the impact of batching across datasets in a workload independent fashion.

Batching based on time : To unify the process for the Twitter and SmartBench datasets, we randomly sample 1000 erasures from a twelve-hour window in our data. For the Tax dataset, we assume 1000 record updates per hour. After sampling the erasures, we use three different batch sizes: one, three, and six hours. The cumulated runtime and number of cells deleted for each batch size is depicted in Fig. 6. In general, larger batch sizes require fewer cells to hide and are faster to process. While in the SmartBench dataset, the number of additionally deleted cells scales linearly, the runtime exhibits a steeper slope between the batch sizes of one and three hours. For the Tax dataset, the number of deleted cells only reduces for Alg 2 and ILP. Initially, batching provides a larger benefit, as

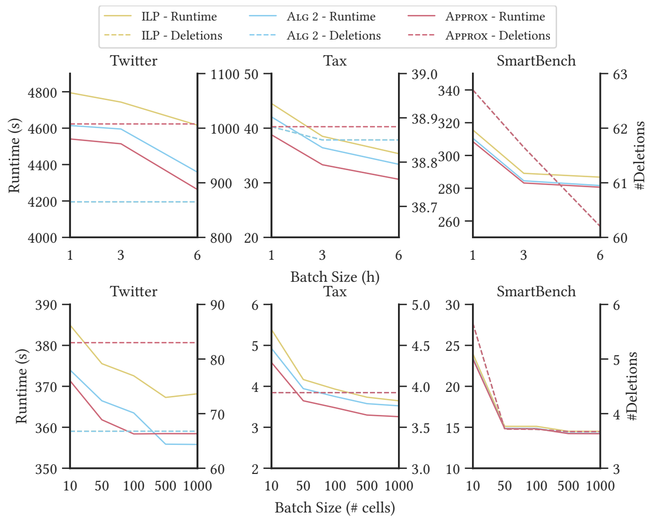


Figure 6: Evaluation of the batching

the impact of finding the first cells that are already instantiated is larger.

Batching based on cardinality: We randomly sampled 1000 erasures from the entire dataset, and we grouped them in batches of size 10, 50, 100, 500, and 1000. We present the runtime and number of deleted cells for the entire set of erasures in Fig. 6. We observe a similar pattern as in the time-based method, larger batches perform better. However, the scaling trend is even more obvious. Initially, enlarging the batches greatly improves performance, while the change between a batch size of 500 and 1000 erasures is negligible. In this experiment, the number of deleted cells does not change when increasing the batch size. The reason is that these erasures are sampled from the entire dataset. While they share some cells, which leads to the performance improvement, the overall paths are so distinct that it is not optimal to adapt the already established optimal set of erasures.

Experiment 4. Here, we evaluate the effectiveness of Scheduler in reducing extra reconstructions of derived data for retention-driven erasures. To solely focus on derived data, we use just the Twitter dataset. The aggregated statistics represent real-world examples of typical derived data. On the one hand, they need to be updated regularly to reflect the underlying profiles. On the other hand, it is cost-prohibitive to compute them on the fly, so minimizing the

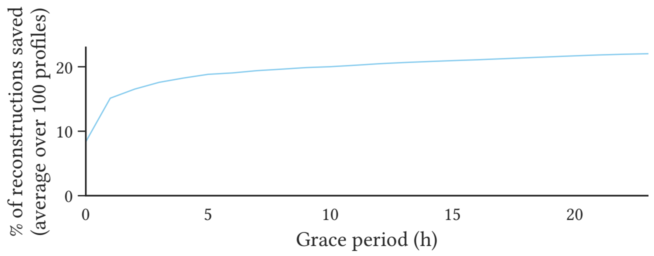


Figure 7: Number of saved reconstructions vs. grace period

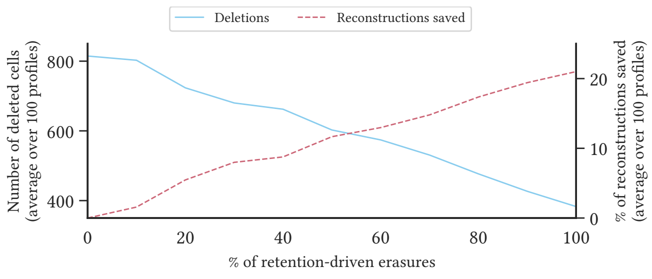


Figure 8: P2E2 overheads per profile for varying workloads

number of necessary reconstructions is desirable. Therefore, we randomly sampled one hundred profiles and sequentially deleted their entire post history. We set the *freq* for each profile to one day and varied the grace period (Γ) in an hourly interval between zero and 23. In Fig. 7, we depict the average number of reconstructions that we save compared to the baseline of just updating the aggregates every day, as well as after every erasure with P2E2 guarantee.

We observe the effectiveness of Scheduler in all cases. Even without a grace period, it saves about 10% reconstructions of the derived data cell. Even allowing for a minimal grace period drastically increases the save potential because only this enables scheduling multiple deletes together. In Fig. 7, allowing a grace period of just one hour increases the average saving of reconstructions to 18%. Increasing the grace period further also decreases the amount of necessary reconstructions. However, the trend levels off, as most deletes that are temporally close to each other are already merged.

Experiment 5. In this experiment, we combine both demand-driven and retention-driven erasures. To investigate the effect of different shares of retention-driven erasures, we employ a similar setting to Experiment 4. Again, we randomly sample 100 profiles from the Twitter dataset. We vary the fraction of profiles which are erased using Scheduler (based on retention-time) vs. demand-driven on the fly. The demand-driven erasures are simulated by generating a random deletion time between the experiment start and the expiration time of the profile. For the entire experiment, we allow a grace period of one hour (Γ = 60 minutes). In both erasure methods, we adopt a time-based batching.

We present the average number of deleted cells and the necessary reconstructions per profile in Fig. 8. The figure shows that the number of deleted cells and the number of reconstructions reduce proportionally as the share of the retention-driven erasures increases. The variation in the amount of change between data points can be explained by the difference in the number of posts authored by each profile. Some larger profiles have more impact depending on the method used to delete them. When comparing 0% to 100% retention-driven erasures, we can save about 53% of deletions (814.6 to 382.8) and about 21% (2274.6 to 1796.9) of reconstructions.

7 RELATED WORK

Sensitive information being recovered from resold hard drives [20, 39], led to work on disk sanitization practices [31] and the field of secure deletion [9], which focused on how to support the persistent erasure of marked files at the memory level [9, 24, 25, 38, 48]. A categorization of erasure mechanisms, often employed in various operating systems, has been presented in [24] and has been alluded to in [15]. However, these do not define or address the semantics of data erasure or consider data dependencies. While [17] contrasts logical and physical erasure in an abstract probabilistic knowledge based framework, their focus is on erasure policy specification. Our work addresses data dependencies — how to express such dependencies (using RDRs) and what erasure means given a set of RDRs. A rich line of work exists in data dependency discovery [11, 35] which may be helpful in specifying RDRs. RDRs can express data annotations explored in [6, 16] to build compliant systems and discovered data dependencies using tools such as [5].

Though the work in [8] discusses some challenges of data erasure compliance in systems, prior work focusing on defining and interpreting data erasure in and beyond the context of compliance is scarce. A few variants of data erasure have been proposed in [15]. More specific works related to data erasure are Lethe [43], which focuses on making LSMs “delete-aware”, and [29], which explores erasure in block-chains. DelF [16] is a bespoke framework at Facebook that lets engineers annotate data erasures on data type definitions which are used to validate erasures, flag errors, and suggest fixes. Approaches to adapt query languages to enable specification of retention periods have been explored in [42]. While these approaches are helpful in data erasure, the data dependencies captured are extremely limited compared to RDRs. SchengenDB [28] is built specifically for GDPR-compliance and focuses on creating logical copies of data to manage data dependence tracing and erasure at the cost of extensive metadata and coarse-grained policies.

Data leakage, i.e., inference control has been explored using query control and access control policies in [18, 33, 37], and recently in [34]. There, a view is selectively constructed by deleting sensitive data based on fine-grained access control policies and inference using denial constraints and functional constraints to answer a query. While we focus on data erasure, our security model is a generalization of the full deniability security model considered in [34] and RDRs considered in our work capture a more general set of dependencies. Existing work, such as [6, 16, 28, 45], do not state or define any formal semantics of data erasure or offer any guarantees. The P2E2-guarantee, in our work, coupled with RDRs, facilitates meaningful data erasure with well-defined guarantees.

8 CONCLUSIONS AND FUTURE WORK

We model the semantics of safe *data erasure* in databases as an inference problem and present multiple definitions to formalize the concept. We focus on one such definition – partial exclusion post erasure (P2E2) and develop exact and approximate algorithms to support P2E2 in databases. While P2E2 supports safe erasure, it comes at the cost of increased overhead to detect data dependencies in data and additional erasures of dependent data. We address the challenge of minimizing such overhead.

In complex data processing systems, safe erasure entails addressing data-dependencies and beyond, giving rise to interesting future work. For example, data erased in a table should also be erased from the entire system, such as logs, back-ups, etc., within a specified amount of time. In such scenarios, relational dependence rules can extended such that they capture dependencies across components of the data-processing pipeline. Such system-level dependencies and memory-level dependencies impact the fundamentals of how systems work — serializability of transactions, logging, and recovery — remain to be explored in future work.

Verification of safe data erasure in databases needs further exploration. Consider P2E2: given a data unit and a database state, the first challenge is to show that the P2E2 guarantee holds. This can be done by extending the result in Theorem 4.10: given a data unit, a database state, and RDRs, ensure that all direct and indirect dependencies have at least one unit erased. Another challenge is attesting that data being claimed to have been P2E2-erased, has indeed been erased in cases where the database owner is not trusted.

REFERENCES

[1] AI Act. https://www.consilium.europa.eu/en/press/press-releases/2023/12/09/artificial-intelligence-act-council-and-parliament-strike-a-deal-on-the-first-worldwide-rules-for-ai/#:~:text=The%20AI%20act%20is%20a,both%20private%20and%20public%20actors., last accessed on 2024-09-01.

[2] GDPR Enforcement Tracker - List of GDPR fines. https://www.enforcementtracker.com/, last accessed on 2024-09-01.

[3] Permanently Delete Your Facebook Account. https://www.facebook.com/help/224562897555674. Accessed: 2024-09-3.

[4] Daniel Abadi, Anastasia Ailamaki, David Andersen, Peter Bailis, Magdalena Balazinska, Philip A. Bernstein, Peter Boncz, Surajit Chaudhuri, Alvin Cheung, Anhai Doan, Luna Dong, Michael J. Franklin, Juliana Freire, Alon Halevy, Joseph M. Hellerstein, Stratos Idreos, Donald Kossmann, Tim Kraska, Sailesh Krishnamurthy, Volker Markl, Sergey Melnik, Tova Milo, C. Mohan, Thomas Neumann, Beng Chin Ooi, Fatma Ozcan, Jignesh Patel, Andrew Pavlo, Raluca Popa, Raghu Ramakrishnan, Christopher Re, Michael Stonebraker, and Dan Suciu. The Seattle report on database research. *Commun. ACM*, 65(8):72–79, jul 2022. doi:10.1145/3524284.

[5] Archita Agarwal, Marilyn George, Aaron Jeyaraj, and Malte Schwarzkopf. Retrofitting gdpr compliance onto legacy databases. *Proc. VLDB Endow.*, 15(4):958–970, apr 2022. doi:10.14778/3503585.3503603.

[6] Kinan Dak Albab, Ishan Sharma, Justus Adam, Benjamin Kilimnik, Aaron Jeyaraj, Raj Paul, Artem Agvanian, Leonhard Spiegelberg, and Malte Schwarzkopf. K9db:{Privacy-Compliant} storage for web applications by construction. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 99–116, 2023.

[7] Whats APP. About disappearing messages. 2020. https://faq.whatsapp.com/673193694148537/?helpref=uf_share, last accessed on 2024-09-01.

[8] Manos Athanassoulis, Subhadeep Sarkar, Zichen Zhu, and Dimitris Staratzis. Building deletion-compliant data systems. *A Quarterly bulletin of the Computer Society of the IEEE Technical Committee on Data Engineering*, 45(1), 2022.

[9] Steven Bauer and Nissanka B Priyantha. Secure data deletion for linux file systems. In *10th USENIX Security Symposium (USENIX Security 01)*, 2001.

[10] Leopoldo Bertossi. *Database repairing and consistent query answering*, volume 20. Morgan & Claypool Publishers, 2011.

[11] Tobias Bleifuß, Sebastian Kruse, and Felix Naumann. Efficient denial constraint discovery with Hydra. *PVLDB*, 11(3):311–323, 2017.

[12] Tobias Bleifuß, Thorsten Papenbrock, Thomas Bläsius, Martin Schirneck, and Felix Naumann. Discovering functional dependencies through hitting set enumeration. *Proceedings of the ACM on Management of Data (SIGMOD)*, 2(1):1–24, 2024.

[13] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 746–755, 2007. doi:10.1109/ICDE.2007.367920.

[14] CCPA. Title 1.81.5. california consumer privacy act of 2018 [1798.100 - 1798.199.100]. https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5, last accessed on 2024-09-01.

[15] Vishal Chakraborty, Stacy Ann-Elvy, Sharad Mehrotra, Faisal Nawab, Mohammad Sadoghi, Shantanu Sharma, Nalini Venkatsubhramanian, and Farhan Saeed. Data-case: Grounding data regulations for compliant data processing systems. *Proceedings 27th International Conference on Extending Database Technology, EDBT, Italy*, 2024.

[16] Katriel Cohn-Gordon, Georgios Damaskinos, Divino Neto, Joshi Cordova, Benoît Reitz, Benjamin Strahs, Daniel Obenshain, Paul Pearce, and Ioannis Papagiannis. Delf: Safeguarding deletion correctness in online social networks. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC, USA, 2020. USENIX Association.

[17] Filippo Del Tedesco, Sebastian Hunt, and David Sands. A semantic hierarchy for erasure policies. In *Information Systems Security: 7th International Conference, ICISS 2011, Kolkata, India, December 15-19, 2011, Procedings 7*, pages 352–369. Springer, 2011.

[18] Harry S. Delugach and Thomas H. Hinke. Wizard: A database inference analysis and detection system. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):56–66, 1996.

[19] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems (TODS)*, 33(2):1–48, 2008.

[20] Simson L Garfinkel and Abhi Shelat. Remembrance of data passed: A study of disk sanitization practices. *IEEE Security & Privacy*, 1(1):17–27, 2003.

[21] Dan Geiger, Azaria Paz, and Judea Pearl. Axioms and algorithms for inferences involving probabilistic independence. *Information and Computation*, 91(1):128–141, 1991.

[22] Peeyush Gupta, Michael J Carey, Sharad Mehrotra, and oberto Yus. Smartbench: a benchmark for data management in smart spaces. *Proceedings of the VLDB Endowment*, 13(12):1807–1820, 2020.

[23] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. Cords: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD, page 647–658, New York, NY, USA, 2004. Association for Computing Machinery. doi:10.1145/1007568.1007641.

[24] Nikolai Joukov, Harry Papaxenopoulos, and Erez Zadok. Secure deletion myths, issues, and solutions. In *Proceedings of the Second ACM Workshop on Storage Security and Survivability*, StorageSS, page 61–66, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1179559.1179571.

[25] Nikolai Joukov and Erez Zadok. Adding secure deletion to your favorite file system. In *Third IEEE International Security in Storage Workshop (SISW'05)*, pages 8–pp. IEEE, 2005.

[26] Youri Kaminsky, Eduardo HM Pena, and Felix Naumann. Discovering similarity inclusion dependencies. *Proceedings of the ACM on Management of Data*, 1(1):1–24, 2023.

[27] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[28] Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner. Schengendb: A data protection database proposal. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 24–38. Springer, 2019.

[29] Michael Kuperberg. Towards enabling deletion in append-only blockchains to support data growth management and gdpr compliance. In *2020 IEEE International Conference on Blockchain (Blockchain)*, pages 393–400. IEEE, 2020.

[30] David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, Hung Q Ngo, Babak Salimi, Harsh Parikh, Moe Kayali, Lise Getoor, Sudeepa Roy, and Dan Suciu. Causal Relational Learning. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 241–256, 2020. doi:10.1145/3318464.3389759.

[31] National Institute of Standards and Technology (NIST). Guide to selecting information security products: Media sanitizing (archived). *???*, 2003. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-36.pdf, last accessed on 2024-09-01.

[32] Publications Office. Regulation (eu) 2016/679, 2019. https://eur-lex.europa.eu/eli/reg/2016/679/oj, last accessed on 2024-09-01.

[33] Kerim Yasin Oktay, Sharad Mehrotra, Vaibhav Khadilkar, and Murat Kantarcioglu. Semrod: secure and efficient mapreduce over hybrid clouds. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 153–166, 2015.

[34] Primal Pappachan, Shufan Zhang, Xi He, and Sharad Mehrotra. Don't be a tattle-tale: Preventing leakages through data dependencies on access control protected data. *Proc. VLDB Endow.*, 15(11):2437–2449, 2022. URL: https://www.vldb.org/pvldb/vol15/p2437-pappachan.pdf.

[35] Eduardo HM Pena, Fabio Porto, and Felix Naumann. Fast algorithms for denial constraint discovery. *Proceedings of the VLDB Endowment*, 16(4):684–696, 2022.

[36] PIPEDA. Personal information protection and electronic documents act (s.c. 2000, c. 5). https://laws-lois.justice.gc.ca/ENG/ACTS/P-8.6/index.html, last accessed on 2024-09-01.

[37] Xiaolei Qian, Mark E Stickel, Peter D Karp, Teresa F Lunt, and Thomas D Garvey. Detection and elimination of inference channels in multilevel relational database systems. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 196–205. IEEE, 1993.

[38] Joel Reardon, Claudio Marforio, Srdjan Capkun, and David Basin. User-level secure deletion on log-structured file systems. In *Proceedings of the 7th ACM symposium on information, computer and communications security*, pages 63–64, 2012.

[39] J Rosenbaum. In defense of the delete key. http://greenbag.org/v3n4/v3n4_articles_rosenbaum.pdf, last accessed on 2024-09-01.

[40] Kenneth A Ross, Divesh Srivastava, Peter J Stuckey, and S Sudarshan. Foundations of aggregation constraints. *Theoretical Computer Science*, 193(1-2):149–179, 1998.

[41] Eduard Rupp, Emmanuel Syrmoudis, and Jens Grossklags. Leave no data behind–empirical insights into data erasure from online services. *Proceedings on Privacy Enhancing Technologies*, 3:437–455, 2022.

[42] Subhadeep Sarkar and Manos Athanassoulis. Query language support for timely data deletion. In *EDBT*, pages 2–429, 2022.

[43] Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, and Manos Athanassoulis. Lethe: A tunable delete-aware LSM engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 893–908, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3318464.3389757.

[44] Subhadeep Sarkar, Dimitris Staratzis, Ziehen Zhu, and Manos Athanassoulis. Constructing and analyzing the lsm compaction design space. *Proc. VLDB Endow.*, 14(11):2216–2229, jul 2021. doi:10.14778/3476249.3476274.

[45] Malte Schwarzkopf, Eddie Kohler, M Frans Kaashoek, and Robert Morris. Position: Gdpr compliance by construction. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 39–53. Springer, 2019.

[46] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. Understanding and benchmarking the impact of gdpr on database systems. *Proc. VLDB Endow.*, 13(7):1064–1077, mar 2020. doi:10.14778/3384345.3384354.

[47] David W Shipman. The functional data model and the data languages DAPLEX. *ACM Transactions on Database Systems (TODS)*, 6(1):140–173, 1981.

[48] Muthian Sivathanu, Lakshmi N Bairavasundaram, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Life or death at block-level. In *OSDI*, volume 4, pages 26–26, 2004.

[49] VDPA. Sb 1392 consumer data protection act (Virginia). https://lis.virginia.gov/cgi-bin/legp604.exe?211+sum+SB1392, last accessed on 2024-09-01.