Meaningful Data Erasure in the Presence of Dependencies

Vishal Chakraborty
University of California (UC), Irvine
vchakrab@uci.edu

Youri Kaminsky
Hasso Plattner Institute
youri.kaminsky@hpi.de

Sharad Mehrotra
UC, Irvine
sharad@ics.uci.edu

Felix Naumann
Hasso Plattner Institute
felixnaumann@hpi.de

Faisal Nawab
UC, Irvine
nawabf@uci.edu

Primal Pappachan
Portland State University
primal@pdx.edu

Mohammad Sadoghi
UC, Davis
msadoghi@ucdavis.edu

Nalini Venkatasubramanian
UC, Irvine
nalini@uci.edu

ABSTRACT

Data regulations like GDPR require systems to support data erasure but leave the definition of "erasure" open to interpretation. This ambiguity makes compliance challenging, especially in databases where data dependencies can lead to erased data being inferred from remaining data. In this paper, we formally define a precise notion of data erasure that ensures any inference about deleted data, through dependencies, remains bounded to what could have been inferred before its insertion. We design erasure mechanisms that enforce this guarantee at 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 our algorithms using both real and synthetic datasets.

PVLDB Reference Format:
Vishal Chakraborty, Youri Kaminsky, Sharad Mehrotra, Felix Naumann, Faisal Nawab, Primal Pappachan, Mohammad Sadoghi, and Nalini Venkatasubramanian. Meaningful Data Erasure in the Presence of Dependencies. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/HPI-Information-Systems/P2E2-Erasure.

1 INTRODUCTION

Recently enacted data protection regulations worldwide [11, 19, 41, 44, 59] have codified the user's right to have their personal data erased and established principles such as data minimization, integrity, and transparency in data processing. The need to support data erasure, which is often referred to as the 'Right to be Forgotten' [41], has propelled recent work in both academic research [5, 50, 51, 54] and industry [15, 60] on the technical challenges and system overheads of implementing data erasure for compliance. In dealing with complex semantic data stored in databases, simply expunging the data a user wishes to be forgotten/deleted may not suffice, as deleted data could be inferred from the remaining data in the database. Such situations arise frequently in domains such as social media, business, smart spaces, patient databases, etc. — any domain in which applications store semantically rich data of their users.

Databases already support deletion of data beyond what the user requests, namely when a deletion may result in consistency violation [25, 35, 38]. For instance, deleting a record in a parent table can result in a cascade of deletions of all dependent records in another table connected to the deleted tuple through a foreign key constraint. Users can further specify application-specific deletions using triggers [38]. Cascading deletion (beyond those needed for maintaining consistency) has also been considered in works such as [3, 49, 54] with a focus on compliance to data regulations – a motivation similar to ours. In these works, "shallow" and "deep" deletions in graph databases through data structures [15] and annotation-based deletions in k9db [3] have been explored. These approaches, however, are proprietary, bespoke (applicable only to the problem setting for which they are developed), and ad-hoc (lacking any formal guarantees of deletion). They adopt an operational view of data deletion without clearly connecting the erasure mechanism to an underlying principle of the user's right to deletion ingrained in privacy regulations such as GDPR [52], specially when data may have semantic dependencies enabling inferences about deleted data.

Our goals in this paper are threefold: (a) to define a principled notion of deletion for databases that prevents deleted data from being inferred (by exploiting semantically dependent data) after deletion, (b) to develop effective and efficient ways to implement the developed deletion notion that minimizes additional data to be deleted, and (c) to understand (through a detailed experimental study across several domains and data sets) the implication and practical viability of such an approach.

Pre-insertion Post Erasure Equivalence. Towards the above goals, we develop a formal definition of deletion, entitled *Pre-insertion Post Erasure Equivalence* (P2E2), that limits inferences about the deleted data using dependencies in data. It is, to the best of our knowledge, a first formal proposal to define erasure when semantic dependencies among data may lead to inferences. We define P2E2 at the cell-level granularity – data regulations such as GDPR often require so[54]. Intuitively, P2E2 requires that no

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

inference can be made about deleted cells post-erasure other than what was possible to make at the time of their insertion.

Example 1.1. Consider a database with a relation and **R** and its attributes **R**(A, B, C). with a constraint: *if* B < 10, *then* 5 < A < 10. Suppose we have a tuple in the database (*NULL*, *NULL*, 30) and we insert a value of A to be 8 resulting in a tuple (8, *NULL*, 30) followed by inserting a value of 9 for B, resulting in the tuple (8,9,30). We subsequently delete the value of attribute A resulting in a tuple (*NULL*, 9, 30). The above constraint can be used to infer that for the tuple, the value of A is less than 10, knowledge we did *not* have prior to insertion of A. In such a case, P2E2 would require that we further delete *B* to ensure that we do not provide any additional inference post-deletion compared to what was possible pre-insertion. Notice in contrast, deleting the value of A in a state (8,9,30) may not result in a P2E2 violation if we reached the state through a different sequence of actions. Say we first insert the value of attribute B, and then A in the tuple resulting in intermediate states of (*NULL*, 9, 30) and (8,9,30). Now when we delete attribute A, we do not need to further delete B to guarantee P2E2 since knowledge that the value of A is less than 10 was already known prior to its insertion. □

Relational Dependency Rules. To define P2E2 more precisely, we introduce *relational dependency rules* (RDRs) that provide a simple, yet general, framework to express a variety of dependencies in data. RDRs use a SQL-like language and can express traditional dependencies including classes of both hard and soft constraints[1], such as functional & inclusion dependencies, denial constraints [7, 35] and correlation constraints, respectively. RDRs, in addition, allow constraints to be limited to only a subset of data that satisfies the SQL query enabling specifications of conditional constraints [47] such as (conditional) functional dependencies [9, 10, 21, 29], and similarity inclusion dependencies [30]. Frameworks, similar to RDRs, to express semantic constraints in database have previously been proposed in a variety of data processing contexts ranging from database design [16, 47], consistent query answering [4], to database repair [7]. We adopt RDRs as they are based on SQL and hence expressive, and intuitive. In addition, they can allow specification of aggregation constraints (often found in organizational databases) which existing framework for specifying constraints such as [25] do not consider. Furthermore, RDRs can express semantic constraints and annotations discussed in recent work motivated by compliance to data regulations [2, 3, 15], which span beyond the traditional data dependencies. RDRs can also specify complex data erasure logic, such as Meta's policy of erasing comments made by the requester, but not private messages sent to others [37] and selective data retention obligations in Art. 17.3, GDPR [41].

Minimizing overhead. Given a set of RDRs[2], and a data item to be deleted, additional data may need to be deleted to ensure P2E2. Such additional deletions, as illustrated earlier, depend on the database state, both at the time of deletion, as well as the time of insertion of the data item being deleted. In Example 1.1, the database state at the deletion time of attribute *A* was (8, 9, 30). However, depending upon the state at the time of insertion (viz. (*NULL*, 9,30) versus (*NULL*, *NULL*, 30)) the set of additional deletions required to implement P2E2 differed. Realizing P2E2 requires tracking what changes were made to the database between the time the data was inserted to the time it is to be deleted, and to develop a logic to reason with RDRs to determine if such changes can cause leakage beyond what was already possible prior to insertion. In addition, to minimize overhead, we would further like the set of additional deletions required (to implement P2E2) to be minimal. We note that such a logic to realize P2E2 cannot be easily encoded as simple deletion rules in the form of triggers[3]. We thus develop algorithms to realize P2E2 with minimal deletions and realize them in a middleware layer on top of the database.

Our problem of identifying a minimal such subset of deletions (formalized as the Opt-P2E2 problem[4]) is related to the problem of minimal repair studied in prior work such as [4, 14]. The minimal repair problem takes as input a set of constraints, such as denial constraints) [14, 35, 56] *C* and a database *D* s.t. *D* is inconsistent w.r.t. *C*. It generates a minimal set of modifications that if performed would make *D* consistent. In contrast, algorithms to achieve P2E2 take as input a database consistent w.r.t. a set of constraints and a data item *c* to be deleted, to identify a minimal subset of dependent data whose deletion would prevent any additional inferences about *c* than could have been made about *c* at the time of its insertion. While both problems minimize deletions, their objective (and hence solutions) differ. As such, as we will see, mechanisms used for database repair cannot directly be used to implement P2E2. So, using RDRs as a framework to represent data dependencies, we design erasure mechanisms to ensure P2E2.

Erasure mechanisms. We develop mechanisms to support P2E2 for two types of data deletion requests: (a) *demand-driven erasure*, wherein users can request data to be deleted at any instance, and (b) *retention-driven erasure*, where data is deleted after a predefined retention period [5]. For example, WhatsApp lets users set varying retention periods for "disappearing messages" [61]. In both cases, erasure mechanisms must identify a minimal set of dependent data to delete, ensuring P2E2. We propose multiple approaches to identify such a minimal set using integer linear programming, bottom-up tree traversal, top-down traversal (trading cost for throughput), and batching multiple erasures. These approaches offer trade-offs based on overheads (time and space) and the amount of data to be deleted. We further explore ways to reduce the overheads of P2E2 based on the type of erasure. For demand-driven erasure, we explore mechanisms to batch deletions using a grace period. For retention-driven erasure, where the system has prior knowledge of when a data item will need to be erased, we develop ways to schedule deletions to minimize costs. Scheduled erasures are particularly useful when derived data (e.g., aggregates, materialized views) must be reconstructed after the deletion of base data.

Evaluation. We evaluate our approaches on real and synthetic datasets under various workloads, analyzing the cost and performance impact of P2E2. We compare exact and approximate algorithms, studying trade-offs between computational overhead and

[1]Hard constraints refer to constraints that always hold while, soft constraints are constraints that are likely to hold [14] on the database instance, though are not guaranteed to hold.

[2]We assume a list of RDRs capturing semantic constraints in the data as input. These RDRs may be manually specified or automatically discovered [9, 21, 30].

[3]Trigger conditions only check the database state before and after the triggering event and, as such, are independent of the sequence of events that led to the state.

[4]We show that the problem is NP-hard and fixed parameter tractable.

cost. Additionally, we examine the effects of batching, varying grace periods, and pre-computing retention periods for derived data. Our evaluation on five data sets shows that, on average, the number of extra deletions to guarantee P2E2 for a given cell is low and depends on factors such as the number of cells in a tuple, dependencies, and the number of insertions and deletions. Note that these results are when we minimize (and approximate) deletions.

In summary, we make the following contributions:

- **Relational Dependency Rules:** an expressive framework to represent semantic and aggregate dependencies in data.
- **Pre-insertion Post Erasure Equivalence (P2E2):** a precise notion of data erasure, suitable for databases when data may be dependent and could be inferred from other data.
- **P2E2 Algorithms:** exact and approximate algorithms for both retention- & demand-driven erasure requests.
- **Extensive performance study:** evaluation of our approaches on real and synthetic datasets under demand-driven, retention-driven, and mixed data erasure workloads.

Roadmap. 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). *A longer version of this paper is available on GitHub with the artifacts) contains formal results and proofs based on this paper.*

2 PRELIMINARIES

In this section, we introduce the formal semantics of data erasure and adopt standard notation and concepts from [24, 36]. But first we introduce a running example, which we use throughout the paper to illustrate notations and explain the intuition behind P2E2.

Running Example. Consider a social media platform where users can make posts, upload photos, and interact with other users. It also offers various location-based services. We present a typical database instance of such a platform. The tables maintained by the platform are as follows (*Database Instance* depicted in Fig. 1a).
- `Person`: Stores user details, including their name, picture, travel status (`Trvl`), and last known location (`lstLoc`).
- `Posts`: Records posts authored by users, including the time, location, and the number of likes (`pLikes`) each post has received.
- `PostedBy`: Tracks which user authored each post.
- `Device`: Maintains the last location (`dLoc`) of the user's device.

The `lstLoc` attribute in `Person` is updated whenever: (1) The user's device location (`dLoc`) changes. (2) A new post by the user is added to the `Posts` table, using the post's location (`pLoc`).

The platform also maintains a `Statistics` table for analytics, which tracks average online activity (`activity`), most frequented location (`freqLoc`), and total likes (`totLikes`) received by all posts authored by a user. The system enforces: (1) When `pLikes` of a post changes, `totLikes` of the corresponding user is updated. (2) `Trvl` in `Person` is updated whenever either `freqLoc` or `lstLoc` changes. `Trvl` is set to `Y` if `freqLoc` and `lstLoc` are different. (3) Other attributes in `Statistics` are updated periodically (e.g., once a week).

In the following, we present some notation and discuss our data and retention model required to formalize the problem setting.

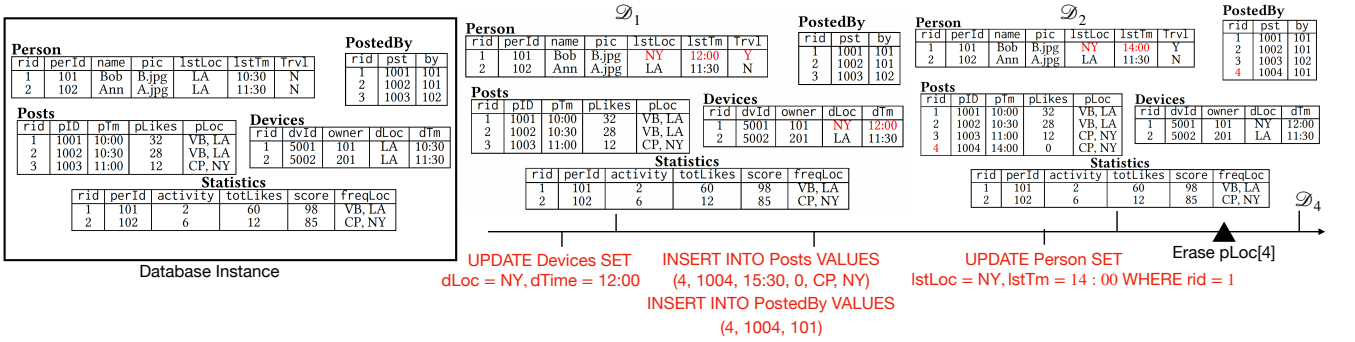
Functional Data Model. We extend the functional data model [55] and its recent adaptation [36]. A database $\mathcal{D}$ consists of relations $\mathcal{R} = \{\boldsymbol{R}_1, \ldots, \boldsymbol{R}_m\}$, each with attributes $\boldsymbol{R}_i^{attr} = \{\mathtt{A}_1, \ldots, \mathtt{A}_{n_j}\}$. Attributes are referred to as $\mathtt{A}_j$ when the relation is clear from context. In Fig. 1a, `Person` is a relation, attributes `perId`, `name`, `profPic`, and `lastLoc`. Each relation $\boldsymbol{R}_i$ contains records r_k^i, uniquely identified by $\mathtt{rid}_k$, its record-Id (`rid`). Records consist of cells, each corresponding to an attribute. We use the operator $Cells(\cdot)$ which returns all the cells in the argument. To an attribute $\mathtt{A}_j \in \boldsymbol{R}_i^{attr}$, we associate the attribute function $\mathtt{A}_j : \boldsymbol{R}_i(\mathtt{rid}) \rightarrow Cells(\mathcal{D})$ that takes in as input a `rid` of a relation and returns the cell corresponding to attribute A_j. A cell is denoted as $\mathtt{A}_j(\boldsymbol{R}_i(\mathtt{rid}_k))$, or $\mathtt{A}_j(\mathtt{rid}_k)$ when the relation is clear. In the database instance in Fig. 1a, `name(1)` refers to the cell containing "Bob". Each cell has an associated deletion cost given by the function $Cost : \mathcal{D} \rightarrow \mathbb{R}$. A relational functional schema is $\mathcal{S} = (\mathcal{R}, \mathcal{A})$ where $\mathcal{A} = \bigcup_{i=1}^{m} \boldsymbol{R}_i^{attr}$.

Database State. At any time t, we write $\mathcal{D}_t$ to denote the *database state*, i.e., the set of all records in the database at time t and their cell values. The value of a cell c in $\mathcal{D}_t$ is given by $Val(c, \mathcal{D}_t) \in Dom(\mathtt{A}_j)$ where $Dom(\mathtt{A}_j)$, the domain of $\mathtt{A}_j$, is the set of all possible values $\mathtt{A}_j$ can take including *NULL*. As an example, in Fig. 1a, $Val(\mathtt{name(1)}, \mathcal{D}_1) = \mathtt{Bob}$. With $\mathcal{D}t^-$ and $\mathcal{D}_{t^+}$ we denote the states immediately before and after $\mathcal{D}_t$, respectively. A cell's value becomes *NULL* (empty) when it is erased, and erasure of a record entails erasure of all cells within it which are not empty.

Types of Data. Relations are classified as *base* or *derived*. Base relations store personal data on which the user has direct control (request deletion, rectification, etc.). Derived relations contain data which result from processing base and/or other derived relations. In the example, `Posts` is a base relation, while `Statistics` is derived.

Base Relations. A cell c in base relations has: (1) creation timestamp $\kappa(c)$ — the time at which it is inserted, (2) erasure/expiration timestamp $\eta(c)$ — the time at which it is deleted. The retention period of a cell is the interval between $\kappa(c)$ and $\eta(c)$. When a record is inserted, for each cell c in the record, $\kappa(c)$ is initialized with the time at which the record was inserted. The expiration time of each cell is set to a fixed or user-specified time at which it needs to be deleted from the database. For a base cell c, users can adjust $\eta(c)$ to enable demand-driven erasure, which overrides the default retention period. Updating a cell is treated as an erasure followed by insertion, updating $\kappa(c)$ to the time of the update but retaining the original $\eta(c)$. Base data erasure satisfies P2E2.

Derived Relations. Derived data are computed from base or other derived relations, with periodic recomputation. For a derived cell c, we denote with $freq(c)$ the time period in which c has to reconstructed at least once. For example, $freq(c)$ for `score` is 30 days, and for `totLikes`, 7 days. When a derived cell c is recomputed, its creation timestamp $\kappa(c)$ is updated to the recomputation time, while $freq(c)$ remains unchanged. Derived data lack explicit erasure timestamps, as they may be reconstructed after base data erasure to prevent inferences. Derived data are deleted only when no longer required, ceasing further reconstruction. Particularly, users cannot directly ask derived data to be deleted.



(a) Database Instance in a social media platform and timeline of operations for running example.

Dependence: totLikes(R) ⫫ pstLikes(M)
Condition: SELECT S.rid, P.rid AS R, M
FROM Statistics S, Posts P, Person I, PostedBy B
WHERE S.perID = I.perID AND B.pst = P.pID
AND B.by = I.perID

(b) RDR R1

Dependence: lastLoc(U) ⫫ pLoc(M)
Condition: SELECT I.rid, P.rid AS U, M
FROM Person I, Devices D, PostedBy B, Posts P (
SELECT B.by AS by, MAX(P.pTm) AS max_pTm FROM
Postedby B, Posts P WHERE B.pst = P.pID
GROUP BY B.by) AS last_posts WHERE
I.perID = D.owner AND I.perID = B.by AND
B.by = last_posts.by AND
P.pTm = last_posts.max_pTM AND P.pTm > D.dTm

(c) RDR R2

Figure 1: Running Example

3 P2E2

In a database, the value of a cell often depends on that of other cells. In this section, we will formally define RDRs as a means of expressing data dependencies. We discuss how RDRs can be instantiated, and how we can reason about inferences using RDRs.

3.1 Specifying Background Knowledge by RDRs

RDRs express dependency (background knowledge) among cells without necessarily specifying how the cells are dependent. An RDR consists of two parts — a dependence statement (which itself has two parts: a head and tail) and a condition. The dependence statement specifies the dependency among cells, while the condition, is a query that identifies the cells on which a dependency holds and returns the rids corresponding to those cells.

Definition 3.1 (Relational dependency Rules). Let $\mathcal{S}=(\mathcal{R}, \mathcal{A})$ be a relational functional schema where $\mathtt{A}, \mathtt{A}_1, \ldots, \mathtt{A}_q \in \mathcal{A}$ are attribute functions, and Q is a SQL query over the schema $\mathcal{S}$ that returns the rids $\boldsymbol{X}, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_p$ of records that satisfy the condition of the query. A *relational dependency rule (RDR)* is given by

$$\begin{aligned} \text{Dependence: } & \mathtt{A}(\boldsymbol{X}) \not\perp \mathtt{A}_1(\boldsymbol{X}_1), \ldots, \mathtt{A}_p(\boldsymbol{X}_p) \\ \text{Condition: } & Q \end{aligned} \tag{1}$$

In Eqn. 1, the *condition* Q identifies rids of a subset of records (from the set of relations in $\mathcal{R}$) such that the records contain the attributes $\mathtt{A}, \mathtt{A}_1, \ldots, \mathtt{A}_p$ which are dependent. The dependence part of the RDR $\mathtt{A}(\boldsymbol{X}) \not\perp \mathtt{A}_1(\boldsymbol{X}_1), \ldots, \mathtt{A}_p(\boldsymbol{X}_p)$ uses attribute function notation to express the dependency between attribute values amongst the selected records. In particular, the RDR expresses that value the attribute $\mathtt{A}(\boldsymbol{X})$ (*head* of the RDR) can take is **not independent** of the value of the attributes $\mathtt{A}_1(\boldsymbol{X}_1), \ldots, \mathtt{A}_p(\boldsymbol{X}_p)$ (*tail* of the RDR).

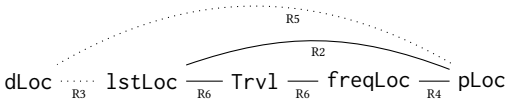


Figure 2: Dependence of attributes in R2 — R5.

Thus, instantiated values of the attributes in the tail of the RDR can enable inference about the value of the head of the attribute.

We illustrate the RDR notation using a couple of semantic dependencies among the data in our example. In Fig. 1b, RDR R1 states the dependency between the number of likes (plikes in Posts table) for a post authored by a person u and the total likes (totlikes in Statistics table) of u. The condition of R1 chooses rid pairs corresponding to the rids of the records in the Statistics and Post tables s.t. the two records are for the same person (due to join conditions in the WHERE clause).

As another illustration, we consider R2 (Fig. 1c) that states that the last location lstLoc of a person u and the location of the latest post (pLoc) made by u are dependent if the post was made before the time the location of the device of u was collected.

We present a few other dependencies for our running example which we will use later in the paper. Let u be a person with a device d and makes a post p:

- **R3**: lstLoc ⫫ dLoc, i.e., the last known location (lstLoc of u depends on the location of d (dLoc collected at dTm) if the last post made by u, pTm, is earlier than dTm.
- **R4**: The location of p (pLoc) and u's most frequented location (freqLoc) are dependent, i.e., freqLoc(R) ⫫ pLoc(M).
- **R5**: The location of d dLoc ⫫ pLoc of p if d's location (dLoc) was collected within an hour of the time of p.
- **R6**: For u, Trvl ⫫ freqLoc, lstLoc.

In Fig. 2 the dependent attributes are the nodes. The labeled (R2 — R6) dotted/solid edges express the dependencies.

RDRs can be discovered using existing work (such as [2, 3] which, motivated by data regulations, discovers data dependencies and annotations, and others like [30, 36]) or through data analysis. In our evaluations, we derived dependencies using existing work and manually. Moreover, rules used for data repairs and data cleaning can be easily expressed as RDRs.

Instantiated RDRs. To use RDRs for specific cells, we need to define *instantiated RDRs*. Instantiations of a RDR, on a database state $\mathcal{D}_t$, are all possible dependencies (in $\mathcal{D}_t$) between the relevant attribute functions (in $\mathcal{D}_t$) which are specified in the dependence statement of the RDR and satisfy the condition of the RDR. An instantiated RDR comprises just the dependence statement of the corresponding RDR, i.e., the head and the tail of the RDR, with all the variables $\boldsymbol{X}, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_p$ substituted with constants from the database state $\mathcal{D}_t$ which are returned by the condition of the RDR. For example, the instantiations of RDR $R1$ in Fig. (1b) for `Person(1)` are: δ_2^- : `totLikes(1)` $\not\perp$ `pstLikes(1)`, δ_3^- : `totLikes(1)` $\not\perp$ `pstLikes(1)`, and δ_4^- : `totLikes(1)` $\not\perp$ `pstLikes(2)`.

Definition 3.2 (Instantiations of RDRs). Let $\mathcal{D}_i$ be an instance over $\boldsymbol{S} = (\mathcal{R}, \mathcal{A})$. Given an RDR as in Eqn. 1, for an instantiated attribute function $\mathtt{A}_k(\boldsymbol{x}_k)$, we associate the *instantiated* RDR, denoted $\delta^-(\mathcal{D}_i, \mathtt{A}_k(\boldsymbol{x}_k))$, obtained by assigning to the variables $\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X}_1 = \boldsymbol{x}_1, \ldots, \boldsymbol{X}_p = \boldsymbol{x}_p$ such that $Val(\mathtt{A}_1(\boldsymbol{x}_1), \mathcal{D}_i), \ldots, Val(\mathtt{A}_p(\boldsymbol{x}_p), \mathcal{D}_i)$ are returned by the query Q evaluated on $\mathcal{D}_i$. When clear from the context, we drop the database state $\mathcal{D}_i$ and $\mathtt{A}_k(\boldsymbol{x}_k)$ from $\delta^-(\mathcal{D}_i, \mathtt{A}_k(\boldsymbol{x}_k))$ and write δ^-. An instantiated RDR is of the following form where $\mathtt{A}_i(\boldsymbol{x}_i)$ is the instantiated attribute function for $\mathtt{A}_i(\boldsymbol{X}_i)$.

$$\delta^- : \mathtt{A}(\boldsymbol{x}) \not\perp \mathtt{A}_1(\boldsymbol{x}_1), \ldots, \mathtt{A}_p(\boldsymbol{x}_p) \tag{2}$$

We denote a set of RDRs with Δ^-. Given a database state $\mathcal{D}_t$, we denote with $\Delta^-(\mathcal{D}_t)$ the set of all instantiated RDRs on that state. With $Head(\delta^-)$, we refer to $\{\mathtt{A}(\boldsymbol{x})\}$, the head of the instantiated RDR. The tail, denoted $Tail(\delta^-)$ is given by the set $\{\mathtt{A}_1(\boldsymbol{x}_1), \ldots, \mathtt{A}_p(\boldsymbol{x}_p)\}$. The condition is dropped. We denote with $Cells(\delta^-)$ the set of all the attribute functions in δ^-, i.e., $Head(\delta^-) \cup Tail(\delta^-)$.

To avoid excessive use of notation, in the following we denote with $\mathtt{A}(x)$ the random variable that takes values in the domain of the cell $\mathtt{A}(\boldsymbol{x})$. Note that $\Pr(\mathtt{A}(\boldsymbol{x}) \mid Tail(\delta^-)) \neq \Pr(\mathtt{A}(\boldsymbol{x}))$ and $\forall 1 \leq i \leq p$, we have $\Pr(\mathtt{A}_i(\boldsymbol{x}_i) \mid \mathtt{A}(\boldsymbol{x}) \cup Tail(\delta^-) \setminus \{\mathtt{A}_i(\boldsymbol{x}_i)\}) \neq \Pr(\mathtt{A}_i(\boldsymbol{x}_i))$.

3.2 Dependency Sets

Given a set of instantiated RDRs, we define the notion of dependency sets. Dependency sets capture how an attribute function $\mathtt{A}(\boldsymbol{x})$ can be probabilistically influenced by, or can influence other data in the database. Instantiated RDRs can lead to the inference of an attribute function $\mathtt{A}(\boldsymbol{x})$ in two ways: (1) direct and (2) indirect.

Direct inference. Direct inference takes place through explicitly stated dependencies that involve $\mathtt{A}(\boldsymbol{x})$.

Example 3.3. The instantiation δ_5^- : `freqLoc(1)` $\not\perp$ `pLoc(4)` of R4 and the instantiation δ_6^- : `lastLoc(1)` $\not\perp$ `pLoc(4)` of R2 lead to the direct inference of `pLoc(4)`.

Indirect Inference. An indirect inference on $\mathtt{A}(\boldsymbol{x})$ exists when a sequence of instantiated RDR can be used to infer $\mathtt{A}(\boldsymbol{x})$ through shared elements between each pair of instantiated RDRs. Continuing Eg. 3.3, let δ_7^- : `lastLoc(1)` $\not\perp$ `devLoc(1)` be another instantiated RDR. Observe that `pLoc(1)` does not directly depend on `devLoc(1)`. However, `pLoc(1)` depends on `lasLoc(1)` through δ_6^-, which in turn depends on `devLoc(1)`.

Definition 3.4. Given a database state $\mathcal{D}_t$ and an attribute function $\mathtt{A}(\boldsymbol{x}) \in \mathcal{D}_t$, and a set of $\Delta^-(\mathcal{D}_t)$ of instantiated RDRs, we define the set of dependencies on $\mathtt{A}(\boldsymbol{x})$ in $\mathcal{D}_t$, denoted $dep(\mathtt{A}\boldsymbol{x})$, to contain $\delta_i^- \in \Delta^-(\mathcal{D}_t)$ such that for all cells $c \in Cells(\delta_i^-)$, we have $Val(c \mid \mathcal{D}_t) \neq NULL$ and one of the following holds:

- $\mathtt{A}(\boldsymbol{x}) \in Cells(\delta_i^-)$
- there exists $\delta_1^-, \ldots, \delta_i^-, \ldots, \delta_K^-$ such that, for $1 < i \leq K$, we have $Tail(\delta_\ell^-) \cap Head(\delta_{\ell+1}^-) \neq \emptyset$ and $\mathtt{A}(\boldsymbol{x}) \in Cells(\delta_1^-)$.

3.3 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 goes well beyond the scope of erasure we are considering here.

Note that when a cell $\mathtt{A}(\boldsymbol{x})$ expires at t_e, its value is set to *NULL*. We write $Val(\mathtt{A}(\boldsymbol{x})) \leftarrow val$ to denote the assignment of the value val to the cell $\mathtt{A}(\boldsymbol{x})$. In particular, with $\mathcal{D}_{t_e^+} \cup \{Val(\mathtt{A}(\boldsymbol{x})) \leftarrow val\}$ we denote the state that is identical to $\mathcal{D}_{t_e^+}$ except that the cell $\mathtt{A}(\boldsymbol{x})$ which has the value `val`. We now formally define P2E2.

Definition 3.5 (Pre-insertion Post Erasure Equivalence (P2E2)). Given a cell $\mathtt{A}(\boldsymbol{x})$ with insertion time $\kappa(\mathtt{A}(\boldsymbol{x})) = t_b$, expiration time $\eta(\mathtt{A}(\boldsymbol{x})) = t_e$ and value $Val(\mathtt{A}(\boldsymbol{x}), \mathcal{D}_{t_e}) = val$ that is not *NULL*, we say that *pre-insertion post erasure equivalence (P2E2)* holds for $\mathtt{A}(\boldsymbol{x})$ if: $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e^+} \cup \{Val(\mathtt{A}(\boldsymbol{x})) \leftarrow val\})) \subseteq dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$.

Informally, P2E2 states that the set of dependencies on an attribute function when it is inserted is the same as the set of dependencies on the attribute function after it has expired. Note that $\mathtt{A}(\boldsymbol{x})$ must be set to *NULL* after it has expired. In Fig. 1a, suppose Bob wants the location of their latest post (i.e., attribute `pLoc` in table `Posts`, `rid=4` with `pID=1004`) to be forgotten. We say P2E2 guarantee holds for `pLoc(4)` if the dependencies on `pLoc(4)` in state $\mathcal{D}_1$ is the same as that in the state after the ▲, i.e., state $\mathcal{D}_4$.

We note that if the set of RDRs completely captures the probability distribution of the attribute function $\mathtt{A}(\boldsymbol{x})$, then P2E2 implies that the probability distributions prior to the insertion of $\mathtt{A}(\boldsymbol{x})$ is identical to that post erasure such that P2E2 holds.

We can now formally define the problem of minimal erasure which is NP-Hard.

Definition 3.6 (Opt-P2E2). Given a database state $\mathcal{D}_t$, the set of instantiated RDRs $\Delta^-(\mathcal{D}_t)$, and a cell $\mathtt{A}(\boldsymbol{x})$ in $\mathcal{D}_t$. Find a set $\mathcal{T} = \{\mathtt{A}(\boldsymbol{x})\} \cup \{\mathtt{A}_i(\boldsymbol{x}_i) \mid 1 \leq i \leq |\mathcal{T}|\}$ such that when the value of each $\mathtt{A}_i(\boldsymbol{x}_i)$ is set to *NULL*, P2E2 holds in $\mathcal{D}_{t^+}$ for $\mathtt{A}(\boldsymbol{x})$ and $\sum_{\mathtt{A}_i(\boldsymbol{x}_i) \in \mathcal{T}} Cost(\mathtt{A}_i(\boldsymbol{x}_i))$ is minimized.

3.4 Identifying Relevant Dependencies

Given a set of RDRs, and an attribute function $\mathtt{A}(\boldsymbol{x})$ for which P2E2 has to be guaranteed, where $\kappa(\mathtt{A}(\boldsymbol{x})) = t_b$ and $\eta(\kappa)(\boldsymbol{x}) = t_e$, we

need to instantiate RDRs to determine the sets $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$ and $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e})$ to compute $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x})) = dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e}) \setminus dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$. Then we address each dependency in $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ to ensure that P2E2 holds.

Constructing $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_t)$. Recall that inference using RDRs can be direct or indirect. Direct inference occurs when an instantiated RDR δ^- contains $\mathtt{A}(\boldsymbol{x}) \in Cells(\delta^-)$. Therefore we need to instantiate all such RDRs where $\mathtt{A}(\boldsymbol{X})$ is in the head or the tail of the dependence. To prevent indirect inference on $\mathtt{A}(\boldsymbol{x})$, we need to consider all RDRs that lead to the direct inference of some attribute function $\mathtt{A}'(\boldsymbol{x})$ such that $\mathtt{A}'(\boldsymbol{x})$ leads to to direct inference of $\mathtt{A}(\boldsymbol{x})$.

With both sets of dependencies constructed as above, the difference $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x})) = dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e}) \setminus dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$ can be readily identified. However, constructing sets of dependencies is computationally intensive as it entails recursively instantiating RDRs to account for both direct and indirect inference.

Constructing $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$. We note that instead of constructing the two sets of dependencies, namely, $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e})$ and $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$, and then computing their difference, we can directly construct $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ To that end, let E be the set of cells erased and N be the set of cells inserted after state $\mathcal{D}_{t_b}$. Therefore, $(\mathcal{D}_{t_b} - E) \cup N = \mathcal{D}_{t_e}$. We first observe that the dependencies on $\mathtt{A}(\boldsymbol{x})$ in $\mathcal{D}_{t_b} \setminus E$ are necessarily contained in the set of dependencies on $\mathtt{A}(\boldsymbol{x})$ in $\mathcal{D}_{t_b}$. This is because since some cells are erased, no new dependencies are introduced. Therefore, we only need to focus on the set of dependencies on $\mathtt{A}(\boldsymbol{x})$ introduced by the cells in N.

Observe that not all dependencies in the set $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e})$ are in the desired set $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ of dependencies that violate P2E2. More specifically, $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e})$, also contains dependencies that existed in the state $\mathcal{D}_{t_b} - E$. Observe that if all the cells of an instantiated RDR $\delta^- \in dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e})$ were inserted before $\mathtt{A}(\boldsymbol{x})$, it must have been in $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b} - E)$, i.e., it does not violate P2E2. Consider the instantiated RDR in our running example δ_5^- (E.g. 3.3), although this is in $dep(\mathtt{pLoc(4)} \mid \mathcal{D}_4)$, it does not violate P2E2 as this dependency on $\mathtt{pLoc(4)}$ existed in the state $\mathcal{D}_1$. We show that for an instantiated RDR to be in $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ it must contain at least one attribute function that was inserted after $\mathtt{A}(\boldsymbol{x})$.

Theorem 3.7. *Let $\mathtt{A}(\boldsymbol{x})$ be a cell with $\kappa(\mathtt{A}(\boldsymbol{x})) = t_b$ and $\eta(\mathtt{A}(\boldsymbol{x})) = t_e$. Let E be the set of erased cells between the states $\mathcal{D}_{t_b}$ and $\mathcal{D}_{t_e}$. The following holds:*

- $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b} - E\} \subseteq dep(\mathtt{A}(\boldsymbol{x} \mid \mathcal{D}_{t_b}\}$.
- $dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e}) - dep(\mathtt{A}(\boldsymbol{x} \mid \mathcal{D}_{t_b} - E) = \{\delta_i^- \mid \delta_i^- \in dep(\mathtt{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e}) \wedge \exists c \in Cells(\delta_i^-)$ *such that* $\kappa(c) > t_b\}$.

Consider the timeline in Fig. 1a, which shows the insertion of Bob's new post and other changes to the database. We want to guarantee P2E2 for the location of Bob's latest post ($\mathtt{pLoc(4)}$). Not all the given dependencies hold for $\mathtt{pLoc(4)}$. RDR R3 is not applicable in this case. Since the difference between the time of the post ($\mathtt{pTm(4)}$) and the time ($\mathtt{dTm(1)}$) at which the location of Bob's device ($\mathtt{dLoc(1)}$) was updated (dotted edge in Fig. 2), $\mathtt{pLoc(4)}$ and $\mathtt{dLoc(1)}$ are not dependent. RDRs R2, R4, and R6 are applicable.

Resolving Violating Dependencies. What remains is to resolve the dependencies in $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ to ensure that the P2E2 condition for $\mathtt{A}(\boldsymbol{x})$ is satisfied. We want to identify a set $\mathcal{T}$ of cells in $\mathcal{D}_{t_e}$ such that when they are deleted(set to *NULL*), P2E2 guarantee holds for $\mathtt{A}(\boldsymbol{x})$. We show that for each instantiated RDR that violates P2E2, we have to delete a cell from its head or tail.

4 SUPPORTING DEMAND-DRIVEN ERASURE

In this section, we focus on demand-driven erasure. Given a set of RDRs and a cell c_d for which P2E2 must hold, the first step is to instantiate the relevant RDRs (Sec. 4.1). Recall that the Opt-P2E2 problem takes as input a set of instantiated RDRs, c_d, and a database state $\mathcal{D}_{\eta(c_d)}$ and outputs a set $\mathcal{T} \subseteq \mathcal{D}_{\eta(c_d)}$ such that when all cells in $\mathcal{T}$ are set to *NULL*, P2E2 holds for c_d and $\mathcal{T}$ incurs the least cost. Our first approach reduces Opt-P2E2 to ILP (Sec. 4.2) which can be solved using readily available ILP solvers. Optimal answers obtained from solvers often have high overheads. We develop a hypergraph-based approach (Sec. 4.3) that provides the optimal answer when RDRs are acyclic. Finally, we extend this to a heuristic approach (Sec. 4.4) that guarantees P2E2, but not at the least cost.

4.1 Instantiating RDRs

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

Algorithm 1 Instantiating RDRs for P2E2

```
1:  procedure DepInst(D_t, Δ⁻, A(x))
2:      Q ← {A(x)}                          ▷ Queue of cells
3:      V ← ∅                               ▷ List of instantiated attributes
4:      I ← ∅                               ▷ List of instantiated RDRs
5:      while Q ≠ ∅ do
6:          attf ← Q.pop
7:          if attf ∉ V then
8:              V ← V ∪ attf
9:              Rules ← Δ⁻(attf)
10:             for rule ∈ Rules do
11:                 δ⁻ ← eval(rule)
12:                 I ← I ∪ {δ⁻}
13:                 for tail ∈ Tail(δ⁻) do
14:                     Q.push(tail)
15:     return V, I
```

Given $\mathtt{A}(\boldsymbol{x})$ and a database $\mathcal{D}_t$, Algorithm 1 generates the set $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ of instantiated RDRs. The algorithm iteratively instantiates RDRs corresponding to direct or indirect inference of $\mathtt{A}(\boldsymbol{x})$. The *eval()* function (in line 11) evaluates the condition of the rule, determines if the rule could have been used for inference in the state $\mathcal{D}_{\kappa(\mathtt{A}(\boldsymbol{x}))^-}$ (Thm. 3.7) and returns the instantiated RDRs.

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 4.1 (Induced Bipartite Graph). For a cell $\mathtt{A}(\boldsymbol{x})$ in a state $\mathcal{D}_t$, given the set $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x})) = \{\delta_1^-, \ldots, \delta_n^-\}$ of instantiated RDRs, we define the induced bipartite graph $\mathcal{B}(\Delta^-(\mathcal{D}_t, P2E2(\mathtt{A}(\boldsymbol{x})))) = (V = V_L \cup V_R, E = E_H \cup E_T)$ 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 contains, for every $\delta_i^- = c_{i_1} \not\perp c_{i_2}, \ldots, c_{i_{n_i}}$, an edge $(\delta_i, c_{i_1}) \in E_H$ and for, $2 \leq j \leq n_i$, edges $(\delta_i, c_{i_j}) \in E_T$.

Reduction. Consider, for a database state $\mathcal{D}_t$, a cell $\mathtt{A}(\boldsymbol{x})$, denoted with c_d, for which P2E2 must hold, and for the set $\Delta^-(\mathcal{D}_t, \mathtt{A}(\boldsymbol{x}))$ of instantiated RDRs, the induced bipartite graph $\mathcal{B}(\Delta^-(\mathcal{D}_t, \mathtt{A}(\boldsymbol{x}))) = (V_L \cup V_R, E_H \cup E_T)$. 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_i^j; for each edge $(\delta_i, c_j) \in E_T$ a binary variable t_i^j. We specify the constraints in the following.

(1) For P2E2 to hold for unit c_d, it must be erased. So, $a_d = 1$.
(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 $a_j = h_i^j$.
(3) For each instantiated RDR δ_i, if the unit in its head is hidden, then we set $b_i = h_j^i$.
(4) For an 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 $\sum_{j \in \{i_1, \ldots, i_{n_i}\}} t_i^j \geq b_i$.
(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 $a_j = t_i^j$.
(6) $W = \mathtt{min} \sum_{j=1}^{m} a_j$ minimizes the number of units erased.

Let O be the preceding ILP instance. 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 $Cost(j) \in \mathbb{N}$, we use $W = \mathtt{min} \sum_{j=1}^{m} c_j Cost(j)$ to minimizes the total cost of P2E2-guarantee. O can be solved with any off-the-shelf optimizer.

4.3 Dependence Hypergraph

Here, we present a new approach to solve Opt-P2E2. Given a set of instantiated RDRs $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ 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 4.2 (Dependence Hypergraph). For a cell $\mathtt{A}(\boldsymbol{x})$ in a database state $\mathcal{D}_t$, given the set $\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))$ of instantiated RDRs, we define the *dependence hypergraph* $\mathcal{H}(\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))) = (V, E)$ 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 $\mathcal{H}(\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x}))) = (V, E)$, 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 $v \notin Head(\delta_i)$. Fig. 3 shows the dependence hypergraph for the set $\{\delta_1^- : c_1 \not\perp c_5, c_7;\ \delta_2^- : c_5 \not\perp c_2;\ \delta_3^- : c_5 \not\perp c_8;\ \delta_4^- : c_1 \not\perp c_9, c_{10};\ \delta_5^- : c_{10} \perp\!\!\!\perp c_{11};\ \delta_6^- : c_{11} \perp\!\!\!\perp c_{12}\}$. The vertex c_1 is a root and c_9 and c_8 are leafs.

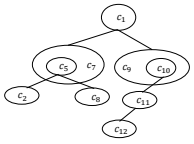


Figure 3: Example of a dependence hypergraph.

Next, we define paths and complete paths in a dependence hypergraph to characterize the P2E2-guarantee.

Definition 4.3. For a dependence hypergraph $\mathcal{H}(\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x})))$, 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.

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.

We can now use the definition of path above to characterize when P2E2 holds. In particular, if there exists a path in $\mathcal{H}(\Delta^-(P2E2, \mathtt{A}(\boldsymbol{x})))$ such that for all vertices on the path, there is a subpath. We show that this is both a necessary and a sufficient condition for P2E2. Note that simply defining a cover [31] would have been sufficient but not necessary for P2E2 to hold. In the dependence hypergraph in Figure 3, it suffices when the units $c_1, c_7,$ and c_9 are set to *NULL* for P2E2 to hold (but they do not cover the graph).

Optimization. Given a set Δ^- of RDRs, a database state $\mathcal{D}_t$, an instantiated attribute function $\mathtt{A}(\boldsymbol{x})$, Algorithm 1 can be easily adapted to construct a dependence hypergraph. Given a dependence hypergraph, Algorithm 2 produces an optimal solution. We assume that the graph is cycle-free. This assumption is necessary only for proving optimality[5]. For a hypergraph, with an attribute function $\mathtt{A}(\boldsymbol{x})$ as its root, a bottom-up traversal is required to determine the minimum cost for P2E2 to hold for $\mathtt{A}(\boldsymbol{x})$.

In practice, we implement 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

[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.

Algorithm 2 Optimizing the Dependence Hypergraph

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

with minimal cost (Line 24). Thus, the algorithm guarantees P2E2 at minimal cost but has to traverse the tree twice.

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 the 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 (a 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. For example, 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 determining 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.

[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.

Algorithm 3 Reconstruction Scheduler

```
1:  procedure createSchedule(c, freq(c), depSet)
2:      κ(c) ← time.now()
3:      Sch[0] ← maxOverlap(depSet, κ(c) + freq(c))
4:      depSet ← depSet \ {c_i | Sch[0] ∈ (η_i^b, η_i^e)}
5:      i ← 1
6:      while depSet ≠ ∅ do
7:          Sch[i] ← maxOverlap(depSet, Sch[i − 1] + freq(c))
8:          depSet ← depSet \ {c_i | Sch[0] ∈ (η_i^b, η_i^e)}
9:          i ← i + 1
10: procedure updateSchedule(c, freq(c), Sch, depSet)
11:     for (c_i, η_i^b, η_i^e) ∈ depSet do
12:         if η_i^b > Sch[0] ∧ η_i^e < Sch[1] then
13:             createSchedule(c, freq(c), depSet)
```

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.

6 EVALUATION

In this section, we evaluate our approaches for guaranteeing P2E2. We compare the ILP approach (Sec. 4.2) with the graph-based algorithm (HGr in Sec. 4.3) and its approximate version, Apx, (Sec. 4.4). 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 (Sec. 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. The first four columns of Table 4b shows the characteristics of the five datasets we used to evaluate our approaches — the number of cells, the number of RDRs (and where/how they were derived), and the number of base and derived attributes.
(1) *Twitter* [18]. This dataset contains a subset of tweets posted on 𝕏 (formerly Twitter) that represents a real-world instance of our running example. The RDRs were designed manually and express dependencies between individuals and their content.
(2) *Tax* [10]. This dataset is a synthetic dataset created using the real-world distribution of values of American tax records. We discovered all present DCs using [43]. The top-10 DCs that are not entirely comprised of equality predicates are transformed into RDRs. The RDRs include the conditional functional dependencies used in the original publication for data cleaning.
(3) *SmartBench* [28]. This dataset is based on real data collected from sensors deployed throughout the campus at the University of California, Irvine. RDRs capture dependencies between multiple physical sensors which are used to compute derived metrics, e.g., occupancy from Wi-Fi AP locations.
(4) *HotCRP* [32]. This is a dataset containing a sample of real-world conference data. It stores authors, papers, conferences, and their relationships. The RDRs are generated by the method of [2].
(5) *TPC-H* [57]. This well-known benchmark dataset stores transactions of commercial actors: customers and their orders, as well as suppliers that fulfill those. The RDRs capture the links between the tables, i.e., foreign keys. In TPC-H, both customers and suppliers may delete their data. We create two separate scenarios and combine them proportionally to the number of customers and suppliers.

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 [51], we evaluate demand- and retention-driven erasures separately, as well as varying combinations of each.

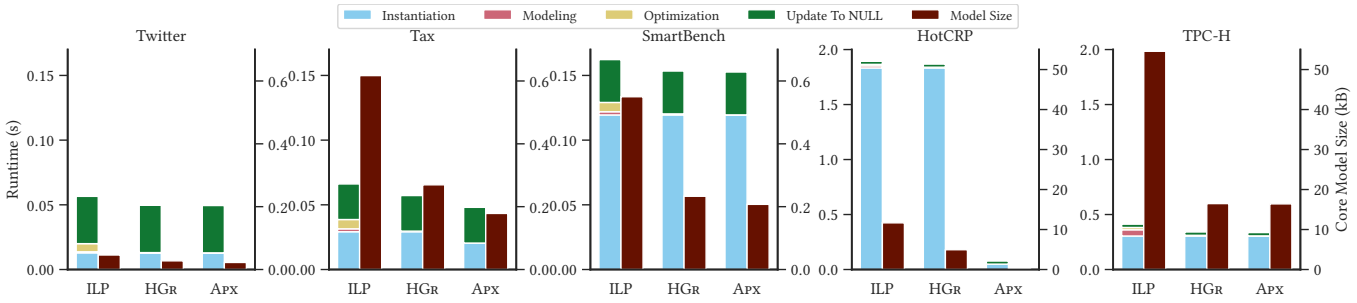
6.2 Experiments

Experiment 1. To test the cost of demand-driven erasures, we erase 100 random cells for each base data 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 HGr and Apx and defining the ILP instance); (iii) model optimization (traversing the graph and keeping track of the minimal cost erasure for HGr and solving the ILP; there is no optimization phase in Apx, 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 (see Table 4b).

We observe a stark difference between Twitter, Tax, SmartBench, and HotCRP and TPC-H. We highlight the similarity within these groups by using the same axis scaling. The methods to create the RDRs for the HotCRP and TPC-H dataset relied on schema constraints (or IND discovery) to discover the dependencies in the data. DBMSs already include a mechanism to delete data linked by foreign keys. Thus, there is no overhead to guarantee P2E2.

Interestingly, neither the number of rules, nor the dataset size determine the deletion complexity. P2E2 is more sensitive to the amount of related data, i.e., the number of instantiated cells. Consequently, the instantiation time takes the most amount of time for

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



(a) Average runtime and space overhead

Dataset	# Cells	# RDRs (Source)	# base, # derived	Instantiated cells ILP	Instantiated cells HGR	Instantiated cells APX	Deleted cells ILP	Deleted cells HGR	Deleted cells APX
𝕏 (Twitter)[7]	21 926 096	11*	7, 11	0.29	0.29	0.29	0.29	0.29	0.29
Tax [10]	16 000 004	10 ([43])	15, 4	6.93	6.93	5.14	4.07	4.07	4.2
SmartBench [28]	94 424	5*	5, 1	5.43	5.43	5.43	5.43	5.43	5.43
HotCRP [32]	122 697	3([2])	6, 3	134.4	134.4	4.81	3.78	3.78	3.78
TPC-H [57]	5 825 639	6 ([25])	8, 4	17.63	17.63	17.63	17.63	17.63	17.63

(b) Summary of datasets and average number of instantiated & deleted cells. *manually designed

Dataset	*freq*	grace period
𝕏	1 day	0 - 23 hours
Tax	1 day	0 - 23 hours
SmartBench	1 hour	0 - 55 minutes
HotCRP	1 week	0 - 6 days
TPC-H	1 month	0 - 30 days

(c) Grace Period

Figure 4: Evaluation of demand-driven erasures

the SmartBench, HotCRP, and TPC-H dataset. The instantiation time of the Tax dataset is proportionally lower although the number of instantiated cells is higher, because the RDRs specify connections within one row of the data. Thus, we benefit from the database's caching. In the Twitter dataset, the final update step dominates the cost, as the number of instantiated cells is low.

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, HGR 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 HGR is negligible compared to the ILP approach. APX instantiates fewer cells, so it outperforms the other approaches for all datasets. This behavior is especially noticeable for HotCRP. The optimal models have to instantiate a long chain of RDRs that turn out to be irrelevant to identify the cheapest deletion set. Due to its greedy nature, APX avoids instantiating all those RDRs and significantly outperforms the rest of the algorithms. Since it does not keep track of an erasure cost, it also consumes less memory. However, it cannot guarantee optimality for its result set, as exemplified in the Tax dataset (see Table 4b).

Experiment 2. To investigate the influence of the degree of dependence (determined by the count of the number of cells that are part of instantiated RDRs) in the data on the number of deleted cells, we conducted the following experiment. For each dataset, we randomly sample 100 erasures. Each sampled erasure is processed by HGR given every subset of RDRs. In Fig. 5, we plot the average number of instantiated cells compared to the average number of deleted cells. The size of the data points signals the number of RDRs present in the processed subset. Intuitively, the more dependent cells a cell has, the more needs to be deleted. The number of rules does not have an immediate effect on the number of deleted cells, as larger and smaller points are mixed along the general trend line. However, the HotCRP dataset is an outlier. Due to the aforementioned long chain of instantiated RDRs, the number of instantiated cells is high, while the necessary deletions remain low.

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 number of erasures in such a batching strategy is workload-dependent. We further create batches based on 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 100 record updates per hour. After sampling the erasures, we use three different grace periods: one, three, and six hours. The HotCRP and TPC-H dataset operate on a different timescale. Therefore, we sample one month for the HotCRP dataset and one year for the TPC-H dataset. They use the batch sizes of one, seven, fourteen days, and one, three, and six months, respectively. The cumulated

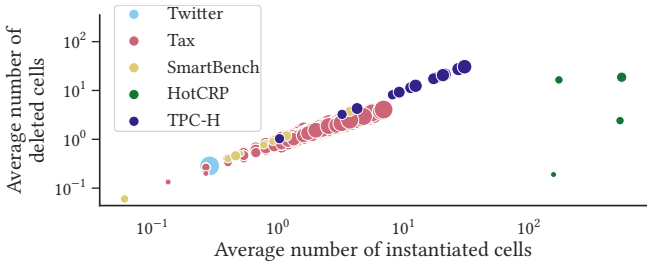


Figure 5: Impact of dependencies (log-axis) on P2E2 overhead; the size of data points signals number of RDRs

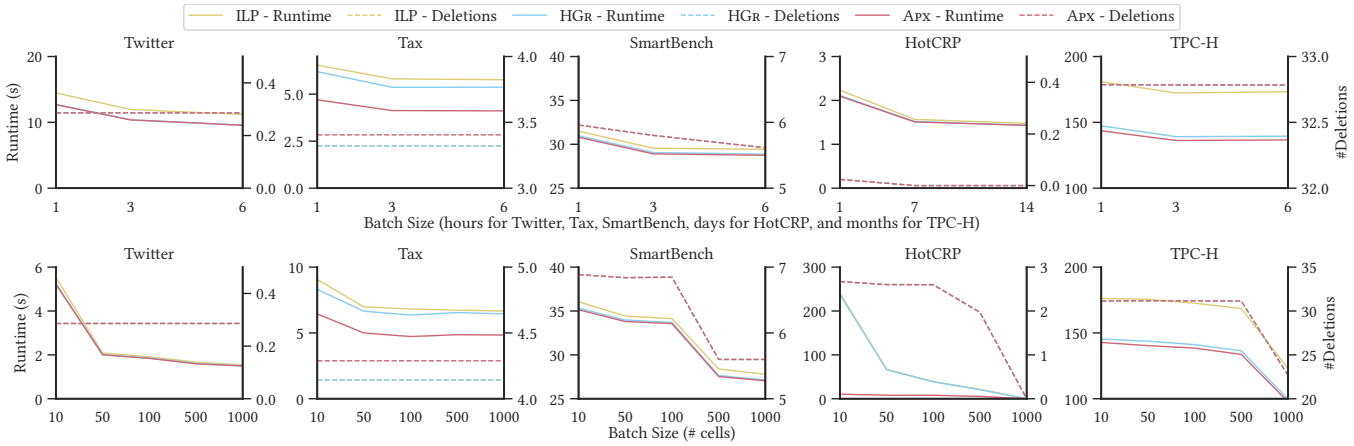


Figure 6: Evaluation of the batching

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 processed faster. Initially, batching provides a larger benefit, as the impact of finding the first cells that are already instantiated is larger. In the Twitter and Tax dataset, the number of additionally deleted cells stays constant. In contrast, it scales similar to the runtime improvement in the HotCRP and TPC-H dataset. 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.

Batching based on cardinality: We randomly sampled 1000 erasures from the entire dataset, and 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. In the Twitter, Tax, and HotCRP datasets, the scaling trend is similar to the previous experiment. For SmartBench, the biggest improvement occurs when enlarging the batch size from 100 to 500 cells. As the erasures are sampled uniformly from the entire dataset, they are less likely to overlap in the instantiated cells and improve runtime. Enlarging the batch size increases this likelihood. Similarly, in the larger TPC-H dataset, this improvement only happens for the largest batch size. In the HotCRP dataset, HGR and ILP manage to become as performant as APX because there is no need to instantiate the long chain of RDRs, if parts of it are already in the batch.

Experiment 4. Here, we evaluate the effectiveness of SCHEDULER in reducing extra reconstructions of derived data for retention-driven erasures. As none of the datasets except SmartBench contained derived data, we manually added aggregate statistics and applicable RDRs to them. These represent real-world examples of typical derived data, e.g., the total likes of a Twitter profile, or the total sales of a supplier in TPC-H. On the one hand, they need to be updated regularly to reflect the underlying base data. On the other hand, it is cost-prohibitive to compute them on the fly, so minimizing the number of necessary reconstructions is desirable.

We randomly sampled 100 cells of each derived data attribute and sequentially deleted all their associated base data. Based on the time-frame of the dataset, we varied the *freq* and the grace

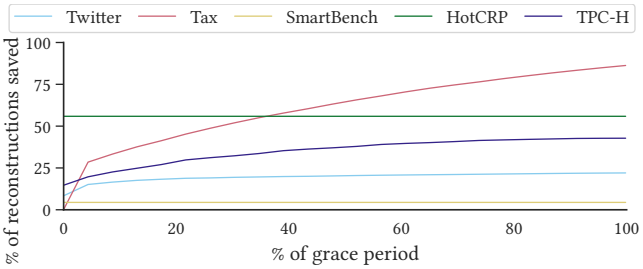


Figure 7: Number of saved reconstructions vs. grace period

period, Γ (see Table 4c). In Fig. 7, we depict the average number of reconstructions that we save compared to the baseline. We observe the effectiveness of SCHEDULER in all cases, but it differs depending on the update characteristic of the dataset. There are three distinct patterns visible. First, HotCRP and SmartBench are insensitive to an increase of the grace period because only base data from the same time is aggregated. Thus, the maximal saving is reached as soon as we allow scheduling. The actual improvement (55.8% for HotCRP, 4.4% for SmartBench) differ based on the number of base data cells that are aggregated into a derived cell. This difference is also apparent between the Twitter and TPC-H dataset. Both these datasets experience "bursty" updates, so initially increasing the grace period reduces the number of reconstructions significantly, but the effect flattens off. The third update characteristic is exhibited in the Tax dataset. It is continuously updated, so there is a steady reduction of necessary reconstructions. Since no two updates happen at the same time, there is no benefit without a grace period. Given the large amount of base data cells per derived data cell, the overall improvement is the largest in this dataset.

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. We vary the fraction of profiles that 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

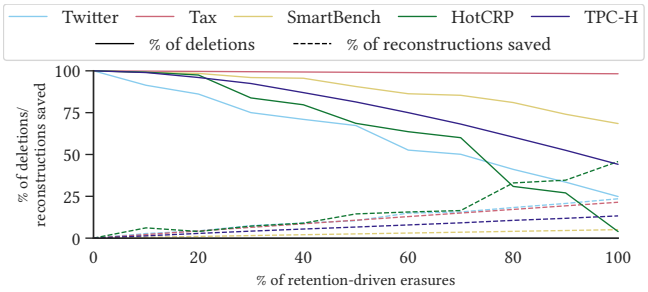


Figure 8: P2E2 overheads for fractions of demand- and retention-driven erasures

and the expiration time. For the entire experiment, we allow a grace period (Γ) of one hour for Twitter and Tax, one minute for SmartBench, one day for HotCRP, and one week for TPC-H. In both erasure methods, we adopt a time-based batching.

We present the average number of deleted cells and the necessary reconstructions in Fig. 8. The figure shows that the number of deleted cells and the number of reconstructions reduce as the share of the retention-driven erasures increases. The improvement largely depends on the update characteristic, as explained for Experiment 4. Thus, the HotCRP dataset profits the most, while we cannot reduce the number of deleted cells for the Tax dataset. The variation in the amount of change depends on the number of base data cells per derived data cells. Some derived data cells that aggregate more data are more impactful, depending on the method used to delete them. When comparing 0% to 100% retention-driven erasures, we can save between 25% (SmartBench) and 90% (HotCRP) of deletions, and between 50% (HotCRP) and 20% (Twitter) of reconstructions.

7 RELATED WORK

Deletion in Systems. Challenges of data erasure compliance in databases have been addressed in recent work [5, 12, 48, 54], and the lack of, and need for, meaningful guarantee of data erasure in systems have been noted in [1, 5]. Database deletions often involve cascades and triggers [23, 38, 45, 49], with varying semantics [25]. Not only are these approaches limited, but also the semantics cannot model P2E2. Recent work like Lethe [51] which enables delete-aware LSMs, [34] which explores blockchain erasure, and DelF [15] which supports Facebook engineers in retention management, do not provide any guarantees or formally define what data erasure is.

Deletion Rules. Many rule-based approaches related to deletion of data exist (database repair, consistent query answering, and data cleaning [4, 6, 14, 20, 22, 35, 47, 47]) which have different settings and goals from our problem. These and work on dependency discovery [2, 8, 30, 42] support RDR specification. Although the syntax of RDRs is inspired by relational causal rules [36] due to their ease in understanding and familiar SQL-like structure, RDRs express semantic data dependencies, whereas relational causal rules express causal dependencies. RDRs can express data annotations explored in [3, 15]. RDRs, unlike delta-rules [25] (which express dependencies and facilitate only tuple deletion), can support aggregates like averages, maximum, and summation and cell-level dependencies necessary for fine-grained deletions. Delta-rules express constraints interpreted under varying semantics, including SQL-inspired cascade rules [25], but none capture P2E2 with its temporal guarantees. Independence semantics [25] minimize tuple deletions for consistency, whereas Opt-P2E2 considers database states at insertion and deletion times to infer dependencies and determine the minimal cells required for erasure. While not directly related to deletion, works on provenance [13, 26, 27] have explored expressing data lineage and data dependencies often used for data regulation compliance [58].

Inference Control. Inference control has been studied through query and access control policies [17, 39, 40, 46]. These methods construct views by selectively removing sensitive data to enforce constraints, but, unlike P2E2, none formally define data erasure or guarantee its outcomes [3, 15, 33, 53].

8 CONCLUSIONS AND FUTURE WORK

We model the semantics of safe *data erasure* in databases as an inference problem, formalizing it as Pre-Insertion Post Erasure Equivalence (P2E2). P2E2 overcomes the challenge of ambiguity in interpreting deletion (discussed in [12, 54]) by providing strong formal guarantees about the deleted data. It can serve as a meaningful user-facing definition that databases can support which, as noted in recent work [5, 51], is currently lacking. We develop exact and approximate algorithms to implement P2E2 in databases using RDRs. While P2E2 provides a principled approach to implementing the right to be forgotten, we envision two potential hurdles in its widespread adoption: First, P2E2 may incur an overhead of deleting additional data (other than the data user wishes to be deleted). Our experiments over five different domains, some of which contain a large proportion of semantically dependent data, demonstrate that using the minimization framework we develop the need for deleting such additional data is low. We believe these results show promise making P2E2 a candidate for supporting deletion in databases. We note P2E2 can also be adopted judiciously to only part of the data and not all of it, to reduce overheads as well as accommodate retention requirements for legal reasons. We could consider generalizations that assign different weights with different dependencies, exploit the specific nature of dependencies using which more precise nature of data inferences can be determined, and/or develop weaker notions of P2E2 that reduce deletions.

Second, P2E2 assumes that data dependencies are either known a priori, or can be learned from data. While generating such knowledge adds to the complexity of data collection, we believe that such complexity will be acceptable to data collectors who, today, are striving to meet regulatory requirements. We note significant prior work on data cleaning, causal learning, and consistent query answering, which too have posed similar requirements on data analysts. Indeed, if data dependencies and rules, were already being used for data analytics, they can be reused for our purpose.

As future work, in addition to exploring extensions of P2E2 discussed above, and exploration of its practicality in various domains, it remains to be explored if similar well-defined guarantees of data deletion can be developed for data processing pipelines where dependencies may exist across different components of a system.

Acknowledgments. The first author was supported by fellowship from HPI@UCI. This work was supported by NSF Grants No. 2032525, 1545071, 1527536, 1952247, 2008993, 2133391, and 2245372.

REFERENCES

[1] 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. 2022. The Seattle Report on Database Research. *Commun. ACM* 65, 8 (jul 2022), 72–79. https://doi.org/10.1145/3524284

[2] Archita Agarwal, Marilyn George, Aaron Jeyaraj, and Malte Schwarzkopf. 2022. Retrofitting GDPR Compliance onto Legacy Databases. *PVLDB* 15, 4 (2022), 958–970. https://doi.org/10.14778/3503585.3503603

[3] Kinan Dak Albab, Ishan Sharma, Justus Adam, Benjamin Kilimnik, Aaron Jeyaraj, Raj Paul, Artem Agvanian, Leonhard Spiegelberg, and Malte Schwarzkopf. 2023. K9db:{Privacy-Compliant} Storage For Web Applications By Construction. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 99–116.

[4] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. 68–79.

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

[6] Leopoldo Bertossi. 2006. Consistent Query Answering in Databases. *ACM SIGMOD Record* 35, 2 (2006), 68–76.

[7] Leopoldo Bertossi. 2011. *Database repairing and consistent query answering*. Vol. 20. Morgan & Claypool Publishers.

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

[9] Tobias Bleifuß, Thorsten Papenbrock, Thomas Bläsius, Martin Schirneck, and Felix Naumann. 2024. Discovering Functional Dependencies through Hitting Set Enumeration. *Proceedings of the International Conference on Management of Data (SIGMOD)* 2, 1 (2024), 1–24.

[10] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional Functional Dependencies for Data Cleaning. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. 746–755. https://doi.org/10.1109/ICDE.2007.367920

[11] CCPA. 2018. 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 2025-01-10.

[12] Vishal Chakraborty, Stacy Ann-Elvy, Sharad Mehrotra, Faisal Nawab, Mohammad Sadoghi, Shantanu Sharma, Nalini Venkatsubhramanian, and Farhan Saeed. 2024. Data-CASE: Grounding Data Regulations for Compliant Data Processing Systems. *Proceedings of the 27th International Conference on Extending Database Technology (EDBT), Italy* (2024).

[13] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (April 2009), 379–474. https://doi.org/10.1561/1900000006

[14] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation* 197, 1-2 (2005), 90–121.

[15] Katriel Cohn-Gordon, Georgios Damaskinos, Divino Neto, Joshi Cordova, Benoît Reitz, Benjamin Strahs, Daniel Obenshain, Paul Pearce, and Ioannis Papagiannis. 2020. DELF: Safeguarding Deletion Correctness in Online Social Networks. In *Proceedings of the 29th USENIX Conference on Security Symposium (SEC)*. USENIX, USA, Article 60, 18 pages.

[16] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: A Commodity Data Cleaning System. *Proceedings of the International Conference on Management of Data (SIGMOD)* (2013), 541–552. https://doi.org/10.1145/2463676.2465327

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

[18] enryu43. 2023. Twitter 100 Million Tweets Dataset. https://huggingface.co/datasets/enryu43/twitter100m_tweets Accessed: 2025-01-25.

[19] European Parliament and Council of the European Union. 2024. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending certain Union legislative acts (Artificial Intelligence Act). https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng. Accessed: 2025-01-10.

[20] Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. 2016. Declarative Cleaning of Inconsistencies in Information Extraction. *ACM Transactions on Database Systems (TODS)* 41, 1 (2016), 1–44.

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

[22] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. 2001. Declarative Data Cleaning: Language, Model, and Algorithms. In *Proceedings of the International Conference on Very Large Databases (VLDB) (2013)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 371–380.

[23] Hector Garcia-Molina, Jeffrey Ullman, and Jennifer Widom. 2008. *Database Systems: The Complete Book*. Pearson Education India.

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

[25] Amir Gilad, Daniel Deutch, and Sudeepa Roy. 2020. On Multiple Semantics for Declarative Database Repairs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 817–831.

[26] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. 2007. Provenance Semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '07)* (Beijing, China). ACM, New York, NY, USA, 31–40. https://doi.org/10.1145/1265530.1265535

[27] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. 2007. Update Exchange with Mappings and Provenance. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. VLDB.

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

[29] Ihab F. Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: automatic discovery of correlations and soft functional dependencies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, New York, NY, USA, 647–658. https://doi.org/10.1145/1007568.1007641

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

[31] Subhash Khot and Oded Regev. 2008. Vertex cover might be hard to approximate to within 2- ε. *J. Comput. System Sci.* 74, 3 (2008), 335–349.

[32] Eddie Kohler. 2024. HotCRP: Conference Review System. https://github.com/kohler/hotcrp Accessed: 2025-01-25.

[33] Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner. 2019. SchengenDB: A Data Protection Database Proposal. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 24–38.

[34] Michael Kuperberg. 2020. Towards enabling deletion in append-only blockchains to support data growth management and GDPR Compliance. In *Proceedings of the IEEE International Conference on Blockchain (Blockchain)*. IEEE, 393–400.

[35] Andrei Lopatenko and Leopoldo Bertossi. 2007. Complexity of Consistent Query Answering in Databases under Cardinality-Based and Incremental Repair Semantics. In *Proceedings of the International Conference on Database Theory (ICDT)*. Springer, 179–193.

[36] 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. 2020. Causal Relational Learning. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020), 241–256. https://doi.org/10.1145/3318464.3389759

[37] Meta. 2017. Permanently Delete Your Facebook Account. https://www.facebook.com/help/224562897555674. Accessed: 2025-01-14.

[38] MySQL. 2019. MySQL Triggers. https://dev.mysql.com/doc/refman/9.0/en/trigger-syntax.html. Accessed:2025-01-10.

[39] Kerim Yasin Oktay, Sharad Mehrotra, Vaibhav Khadilkar, and Murat Kantarcioglu. 2015. SEMROD: Secure and Efficient MapReduce over Hybrid Clouds. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 153–166.

[40] Primal Pappachan, Shufan Zhang, Xi He, and Sharad Mehrotra. 2022. Don't Be a Tattle-Tale: Preventing Leakages through Data Dependencies on Access Control Protected Data. *Proceedings of the VLDB Endowment (PVLDB)* 15, 11 (2022), 2437–2449. https://www.vldb.org/pvldb/vol15/p2437-pappachan.pdf

[41] European Parliament and Council of the European Union. 2019. Regulation (EU) 2016/679. https://eur-lex.europa.eu/eli/reg/2016/679/oj, last accessed on 2025-01-10.

[42] Eduardo HM Pena, Fabio Porto, and Felix Naumann. 2022. Fast Algorithms for Denial Constraint Discovery. *PVLDB* 16, 4 (2022), 684–696.

[43] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *PVLDB* 13, 3 (2019), 266–278.

[44] PIPEDA. 2024. 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 2025-01-10.

[45] PostgreSQL. 2019. PostgreSQL Triggers. https://www.postgresql.org/docs/current/trigger-definition.html. Accessed: 2025-01-10.

[46] Xiaolei Qian, Mark E Stickel, Peter D Karp, Teresa F Lunt, and Thomas D Garvey. 1993. 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*. IEEE, 196–205.

[47] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: holistic data repairs with probabilistic inference. *PVLDB* 10, 11 (Aug. 2017), 1190–1201. https://doi.org/10.14778/3137628.3137631

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

[49] Subhadeep Sarkar and Manos Athanassoulis. 2022. Query Language Support for Timely Data Deletion. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 2–429.

[50] Subhadeep Sarkar, Jean-Pierre Banâtre, Louis Rilling, and Christine Morin. 2018. Towards Enforcement of the EU GDPR: Enabling Data Erasure. In *Proceedings of the 11th IEEE International Conference on Internet of Things (iThings 2018)*. Halifax, Canada, 1–8. https://hal.inria.fr/hal-01824058

[51] Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, and Manos Athanassoulis. 2020. Lethe: A Tunable Delete-Aware LSM Engine. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. ACM, New York, NY, USA, 893–908. https://doi.org/10.1145/3318464.3389757

[52] Subhadeep Sarkar, Dimitris Staratzis, Ziehen Zhu, and Manos Athanassoulis. 2021. Constructing and analyzing the LSM compaction design space. *PVLDB* 14, 11 (jul 2021), 2216–2229. https://doi.org/10.14778/3476249.3476274

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

[54] Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. 2020. Understanding and Benchmarking the Impact of GDPR on Database Systems. *Proceedings of the VLDB Endowment (PVLDB)* 13, 7 (mar 2020), 1064–1077. https://doi.org/10.14778/3384345.3384354

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

[56] Slawomir Staworko. 2007. Declarative Inconsistency Handling in Relational and Semi-Structured Databases. *PhD Thesis* (2007).

[57] Transaction Processing Performance Council. 2021. *TPC-H Benchmark Specification, Version 2.17.3.* Technical Report. Transaction Processing Performance Council. http://www.tpc.org/tpch/

[58] Benjamin E. Ujcich, Adam Bates, and William H. Sanders. 2018. A Provenance Model for the European Union General Data Protection Regulation. In *Proceedings of the International Provenance and Annotation Workshop (IPAW)*. Springer, 45–57. https://doi.org/10.1007/978-3-319-98379-0_4

[59] VDPA. 2021. SB 1392 Consumer Data Protection Act (Virginia). https://lis.virginia.gov/cgi-bin/legp604.exe?211+sum+SB1392, last accessed on 2025-01-10.

[60] Yang Wang, Pedro Giovanni Leon, Alessandro Acquisti, Lorrie Faith Cranor, Alain Forget, Norman Sadeh, Norman Sadeh, Lorrie Faith Cranor, Alain Forget, and Norman Sadeh. 2014. A Field Trial of Privacy Nudges for Facebook. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)* (Toronto, Ontario, Canada) *(CHI)*. ACM, New York, NY, USA, 2367–2376. https://doi.org/10.1145/2556288.2557413

[61] WhatsApp. 2020. About Disappearing Messages. https://faq.whatsapp.com/673193694148537/?helpref=uf_share, last accessed on 2025-01-10.