Meaningful Data Erasure in the Presence of Dependencies (Long Version)

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@psu.edu

Mohammad Sadoghi
UC, Davis
sadoghi@ucdavis.edu

Nalini Venkatasubramanian
UC, Irvine
nalini@ics.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 it challenging to ensure compliance, especially in databases where data dependencies can lead to erased data being inferred from remaining data. In this paper, we formally define a precise notion of data erasure, where the system guarantees that inference of the value of erased data, given a database that complies with a given set of dependencies, remains the same as it was at the time the data was inserted. We design erasure mechanisms that provide such a precise 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
(Long Version). 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 URL_TO_YOUR_ARTIFACTS.

1 INTRODUCTION

Recently enacted data protection regulations worldwide [15, 23, 42, 45, 57] 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' [42], has propelled recent work in both academic research [7, 49, 50, 52] and industry [19, 58] 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 [29, 38, 41]. 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 [41]. Cascading deletion (beyond those needed for maintaining consistency) has also been considered in works such as [5, 48, 52] 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 [19] and annotation-based deletions in k9db [5] 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 [51], 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. We formally define *Pre-insertion Post-Erasure Equivalence* (P2E2), a deletion principle that restricts inferences about deleted data using dependencies in the

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

database to only those that were possible at the time of its insertion. P2E2 is defined at the cell level, with each cell assigned an expiration time by which it must be deleted. While some data regulations are vague on inference of deleted data using dependencies [16], some regulations [21] require to prevent "reconstruction (of deleted data) in an intelligible form." Our formalization of data deletion in presence of data dependencies aligns with regulatory expectations for safe and effective deletion, selective retention, automated erasure and principles of data minimization and purpose limitation as outlined in CCPA [15] §1798.105(d), PIPEDA [45] Principle 4.5.5, GDPR [42] Art. 5(1) & 17, LGPD [3] Art. 15, HIPAA [1] 45 CFR §164.310(d)(2)(i), and PDPA [2] Sec. 25.

Example 1.1. Consider a relation **R(Name, City, AreaCode)** with the constraint: *if* `City = Irvine`, *then* `AreaCode` $\in \{714, 657, 949\}$ (implemented in PostgreSQL as a check constraint [1]). Starting with the tuple `(Alice, NULL, NULL)`, first insert `AreaCode = 949`, resulting in the tuple `(Alice, NULL, 949)`, followed by `City = Irvine` results in `(Alice, Irvine, 949)`. Deleting `AreaCode` yields `(Alice, Irvine, NULL)`, from which one can now infer that `AreaCode` was between 200 and 500—information that was *not inferable* at the time of `AreaCode`'s insertion. To satisfy P2E2, we must also delete `City`, ensuring no new inferences arise post-deletion. In contrast, if `City = Irvine` is inserted *before* `AreaCode = 949`, the same deletion of `AreaCode` does not require deleting `City`, since the constraint was already active and the inference about `AreaCode` was possible prior to its insertion. This highlights the subtlety of P2E2: deletion must prevent *new* inferences based on post-insertion context, which depends on the sequence of actions.

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[2], such as functional & inclusion dependencies, denial constraints [9, 38] 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 [46] such as (conditional) functional dependencies [11, 12, 25, 33], and similarity inclusion dependencies [34]. 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 [20, 46], consistent query answering [6], to database repair [9]. 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 [29] do not consider. Furthermore, RDRs can express semantic constraints and annotations discussed in recent work motivated by compliance to data regulations [4, 5, 19], 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 [40] and selective data retention obligations in Art. 17.3, GDPR [42].

Minimizing overhead. Given a set of RDRs[3], 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[4]. 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[5]) is related to the problem of minimal repair studied in prior work such as [6, 18]. The minimal repair problem takes as input a set of constraints, such as denial constraints) [18, 38, 54] 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 [7]. For example, WhatsApp lets users set varying retention periods for "disappearing messages" [59]. 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

[1]https://www.postgresql.org/docs/current/ddl-constraints.html

[2]Hard constraints refer to constraints that always hold while, soft constraints are constraints that are likely to hold [18] on the database instance, though are not guaranteed to hold.

[3]We assume a list of RDRs capturing semantic constraints in the data as input. These RDRs may be manually specified or automatically discovered [11, 25, 34].

[4]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.

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

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 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). **Proofs of all theorems are in the Appendix.**

Symbol	Meaning
$\mathcal{S} = (\mathcal{R}, \mathcal{A})$	Schema with sets $\mathcal{R}$ of relations & $\mathcal{A}$ of attributes
$\mathbf{R}_i$, $\mathtt{A}_j$	A relation $\mathbf{R}_i \in \mathcal{R}$ and attribute $\mathtt{A}_j \in \mathcal{A}$
rid_k	Record-ID uniquely identifying records
$\mathcal{D}_t$	Database state, i.e., set of cells in database at time t
$Cells(\cdot)$	Operator returning all cells in the given argument
$A_j(rid_k)$	Cell containing the value of attribute A_j of rid_k
$\kappa(c)$	Creation (insertion) timestamp of cell c
$\eta(c)$	Expiration timestamp of cell c (erasure time)
$Val(\mathtt{A}(\boldsymbol{x}), \mathcal{D}_t)$	Value of attribute function $\mathtt{A}(\boldsymbol{x})$ in $\mathcal{D}_t$
$dep(\mathtt{A}(\boldsymbol{x}))$	Set of dependencies on $\mathtt{A}(\boldsymbol{x})$
δ^-	Instantiated relational dependency rule (RDR)
$\Delta^-(\mathcal{D}_t)$	Set of all instantiated RDRs in $\mathcal{D}_t$
$Head(\delta^-)$	Head of instantiated RDR δ^-
$Tail(\delta^-)$	Tail of instantiated RDR δ^-

Table 1: Notation Table

2 PRELIMINARIES

In this section, we introduce the formal semantics of data erasure and adopt standard notation and concepts from [28, 39]. 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`).

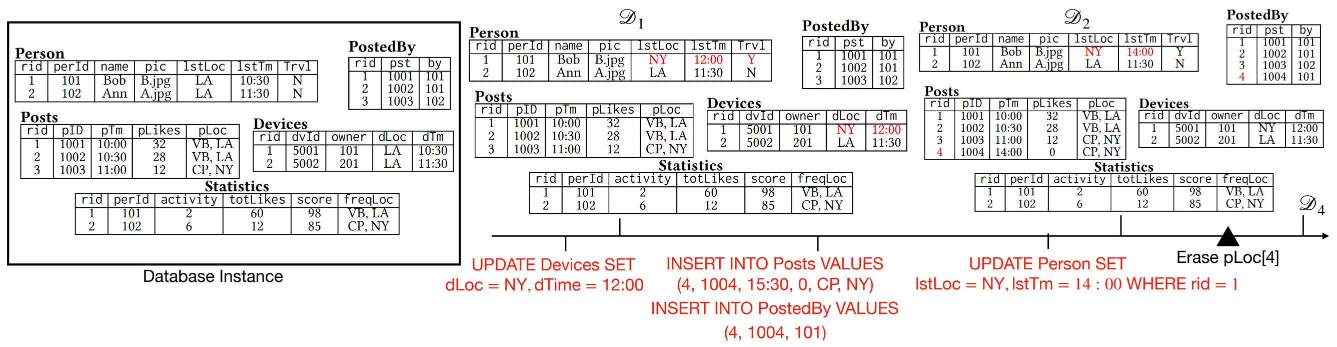
The platform also maintains a `Statistics` table for analytics, which tracks average online activity (`activity`), most frequented location (`freqLoc`), and total likes (`totLikes`) received by all posts authored by a user. The system enforces: (1) When `pLikes` of a post changes, `totLikes` of the corresponding 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 [53] and its recent adaptation [39]. 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 ($\mathtt{rid}$). 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 $\mathtt{rid}$ 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


Database Instance
$\mathscr{D}_1$
$\mathscr{D}_2$
$\mathscr{D}_4$
UPDATE Devices SET dLoc = NY, dTime = 12:00
INSERT INTO Posts VALUES (4, 1004, 15:30, 0, CP, NY)
INSERT INTO PostedBy VALUES (4, 1004, 101)
UPDATE Person SET lstLoc = NY, lstTm = 14 : 00 WHERE rid = 1
Erase pLoc[4]

Database Instance

Person

rid	perId	name	pic	lstLoc	lstTm	Trvl
1	101	Bob	B.jpg	LA	10:30	N
2	102	Ann	A.jpg	LA	11:30	N

PostedBy

rid	pst	by
1	1001	101
2	1002	101
3	1003	102

Posts

rid	pID	pTm	pLikes	pLoc
1	1001	10:00	32	VB, LA
2	1002	10:30	28	VB, LA
3	1003	11:00	12	CP, NY

Devices

rid	dvId	owner	dLoc	dTm
1	5001	101	LA	10:30
2	5002	201	LA	11:30

Statistics

rid	perId	activity	totLikes	score	freqLoc
1	101	2	60	98	VB, LA
2	102	6	12	85	CP, NY

$\mathscr{D}_1$

Person

rid	perId	name	pic	lstLoc	lstTm	Trvl
1	101	Bob	B.jpg	NY	12:00	Y
2	102	Ann	A.jpg	LA	11:30	N

PostedBy

rid	pst	by
1	1001	101
2	1002	101
3	1003	102

Posts

rid	pID	pTm	pLikes	pLoc
1	1001	10:00	32	VB, LA
2	1002	10:30	28	VB, LA
3	1003	11:00	12	CP, NY

Devices

rid	dvId	owner	dLoc	dTm
1	5001	101	NY	12:00
2	5002	201	LA	11:30

Statistics

rid	perId	activity	totLikes	score	freqLoc
1	101	2	60	98	VB, LA
2	102	6	12	85	CP, NY

$\mathscr{D}_2$

Person

rid	perId	name	pic	lstLoc	lstTm	Trvl
1	101	Bob	B.jpg	NY	14:00	Y
2	102	Ann	A.jpg	LA	11:30	N

PostedBy

rid	pst	by
1	1001	101
2	1002	101
3	1003	102
4	1004	101

Posts

rid	pID	pTm	pLikes	pLoc
1	1001	10:00	32	VB, LA
2	1002	10:30	28	VB, LA
3	1003	11:00	12	CP, NY
4	1004	14:00	0	CP, NY

Devices

rid	dvId	owner	dLoc	dTm
1	5001	101	NY	12:00
2	5002	201	LA	11:30

Statistics

rid	perId	activity	totLikes	score	freqLoc
1	101	2	60	98	VB, LA
2	102	6	12	85	CP, NY

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

Dependence: $\texttt{totLikes}(R) \not\perp\!\!\!\perp \texttt{pstLikes}(M)$
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: $\texttt{lastLoc}(U) \not\perp\!\!\!\perp \texttt{pLoc}(M)$
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

(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.

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 $\mathbf{S} = (\mathcal{R}, \mathcal{A})$ be a relational functional schema where $\mathsf{A}, \mathsf{A}_1, \ldots, \mathsf{A}_q \in \mathcal{A}$ are attribute functions, the sets $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_p$ of variables and/or constants, and Q is a SQL query over the schema $\mathbf{S}$ 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

$$\text{Dependence: } \mathsf{A}(\boldsymbol{X}) \not\perp\!\!\!\perp \mathsf{A}_1(\boldsymbol{X}_1), \ldots, \mathsf{A}_p(\boldsymbol{X}_p)$$
$$\text{Condition: } Q \qquad (1)$$

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 $\mathsf{A}, \mathsf{A}_1, \ldots, \mathsf{A}_p$ which are dependent. The dependence part of the RDR $\mathsf{A}(\boldsymbol{X}) \not\perp\!\!\!\perp \mathsf{A}_1(\boldsymbol{X}_1), \ldots, \mathsf{A}_p(\boldsymbol{X}_p)$ uses attribute function notation to express the dependency between attribute values amongst the selected records. In particular, the RDR expresses that value the attribute $\mathsf{A}(\boldsymbol{X})$ (*head* of the RDR) can take is **not independent** of the value of the attributes $\mathsf{A}_1(\boldsymbol{X}_1), \ldots, \mathsf{A}_p(\boldsymbol{X}_p)$ (*tail* of the RDR). 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

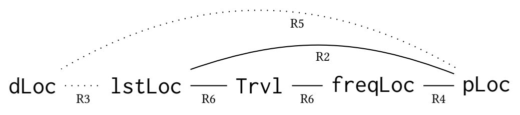


Figure 2: Dependence of attributes in R2 — R5.

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 $\not\perp\!\!\!\perp$ 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) $\not\perp\!\!\!\perp$ pLoc(M).
- **R5**: The location of d dLoc $\not\perp\!\!\!\perp$ pLoc of p if d's location (dLoc) was collected within an hour of the time of p.
- **R6**: For u, Trvl $\not\perp\!\!\!\perp$ 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 [4, 5] which, motivated by data regulations, discovers data dependencies and annotations, and others like [34, 39]) or through data analysis. In our evaluations, we derived dependencies using existing work and manually. Moreover, rules used for data repairs, data cleaning, as well as provenance annotations 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) $\not\perp\!\!\!\perp$ pstLikes(1), δ_3^- : totLikes(1) $\not\perp\!\!\!\perp$ pstLikes(1), and δ_4^- : totLikes(1) $\not\perp\!\!\!\perp$ 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 $\mathsf{A}_k(\boldsymbol{x}_k)$, we associate the *instantiated* RDR, denoted $\delta^-(\mathcal{D}_i, \mathsf{A}_k(\boldsymbol{x}_k))$, obtained by assigning to the variables $\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{X}_1 = \boldsymbol{x}_1, \ldots, \boldsymbol{X}_p = \boldsymbol{x}_p$ such that $Val(\mathsf{A}_1(\boldsymbol{x}_1), \mathcal{D}_i), \ldots, Val(\mathsf{A}_p(\boldsymbol{x}_p), \mathcal{D}_i)$ 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 $\mathsf{A}_k(\boldsymbol{x}_k)$ from $\delta^-(\mathcal{D}_i, \mathsf{A}_k(\boldsymbol{x}_k))$ and write δ^-. An instantiated RDR is of the following form where $\mathsf{A}_i(\boldsymbol{x}_i)$ is the instantiated attribute function for $\mathsf{A}_i(\boldsymbol{X}_i)$.

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

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

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

3.2 Dependency Sets

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

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

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

Indirect Inference. An indirect inference on $\mathsf{A}(\boldsymbol{x})$ exists when a sequence of instantiated RDR can be used to infer $\mathsf{A}(\boldsymbol{x})$ through shared elements between each pair of instantiated RDRs. Continuing Eg. 3.3, let δ_7^- : lastLoc(1) $\not\perp\!\!\!\perp$ 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 $\mathsf{A}(\boldsymbol{x}) \in \mathcal{D}_t$, and a set of $\Delta^-(\mathcal{D}_t)$ of instantiated RDRs, we define the set of dependencies on $\mathsf{A}(\boldsymbol{x})$ in $\mathcal{D}_t$, denoted *dep*$(\mathsf{A}\boldsymbol{x})$, 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:

- $\mathsf{A}(\boldsymbol{x}) \in Cells(\delta_i^-)$
- 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 $\mathsf{A}(\boldsymbol{x}) \in Cells(\delta_1^-)$.

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

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

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 $\mathsf{A}(x)$ 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 $\mathsf{A}(x)$, then P2E2 implies that the probability distributions prior to the insertion of $\mathsf{A}(x)$ 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 $\mathsf{A}(x)$ in $\mathcal{D}_t$. Find a set $\mathcal{T} = \{\mathsf{A}(x)\} \cup \{\mathsf{A}_i(x_i) \mid 1 \leq i \leq |\mathcal{T}|\}$ such that when the value of each $\mathsf{A}_i(x_i)$ is set to *NULL*, P2E2 holds in $\mathcal{D}_{t^+}$ for $\mathsf{A}(x)$ and $\sum_{\mathsf{A}_i(x_i)\in\mathcal{T}} Cost(\mathsf{A}_i(x_i))$ is minimized.

Theorem 3.7. *(Hardness) The Opt-P2E2 problem is NP-hard.*

3.4 Identifying Relevant Dependencies

Given a set of RDRs, and an attribute function $\mathsf{A}(x)$ for which P2E2 has to be guaranteed, where $\kappa(\mathsf{A}(x)) = t_b$ and $\eta(\kappa)(x) = t_e$, we need to instantiate RDRs to determine the sets $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_b})$ and $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e})$ to compute $\Delta^-(P2E2, \mathsf{A}(x)) = dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e}) \setminus dep(\mathsf{A}(x) \mid \mathcal{D}_{t_b})$. Then we address each dependency in $\Delta^-(P2E2, \mathsf{A}(x))$ to ensure that P2E2 holds.

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

With both sets of dependencies constructed as above, the difference $\Delta^-(P2E2, \mathsf{A}(x)) = dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e}) \setminus dep(\mathsf{A}(x) \mid \mathcal{D}_{t_b})$ 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 $\Delta^-(P2E2, \mathsf{A}(x))$. We note that instead of constructing the two sets of dependencies, namely, $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e})$ and $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_b})$, and then computing their difference, we can directly construct $\Delta^-(P2E2, \mathsf{A}(x))$ 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 $\mathsf{A}(x)$ in $\mathcal{D}_{t_b} \setminus E$ are necessarily contained in the set of dependencies on $\mathsf{A}(x)$ 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 $\mathsf{A}(x)$ introduced by the cells in N.

Observe that not all dependencies in the set $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e})$ are in the desired set $\Delta^-(P2E2, \mathsf{A}(x))$ of dependencies that violate P2E2. More specifically, $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e})$, also contains dependencies that existed in the state $\mathcal{D}_{t_b} - E$. Observe that if all the cells of an instantiated RDR $\delta^- \in dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e})$ were inserted before $\mathsf{A}(x)$, it must have been in $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_b} - E)$, 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 pLoc(4) existed in the state $\mathcal{D}_1$. We show that for an instantiated RDR to be in $\Delta^-(P2E2, \mathsf{A}(x))$ it must contain at least one attribute function that was inserted after $\mathsf{A}(x)$.

Theorem 3.8. *Let $\mathsf{A}(x)$ be a cell with $\kappa(\mathsf{A}(x)) = t_b$ and $\eta(\mathsf{A}(x)) = t_e$. Let E be the set of erased cells between the states $\mathcal{D}_{t_b}$ and $\mathcal{D}_{t_e}$. The following holds:*

- $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_b} - E\} \subseteq dep(\mathsf{A}(x \mid \mathcal{D}_{t_b}\}$.
- $dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e}) - dep(\mathsf{A}(x \mid \mathcal{D}_{t_b} - E) = \{\delta_i^- \mid \delta_i^- \in dep(\mathsf{A}(x) \mid \mathcal{D}_{t_e}) \wedge \exists c \in Cells(\delta_i^-)$ *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 (`pLoc(4)`). Not all the given dependencies hold for `pLoc(4)`. RDR R3 is not applicable in this case. Since the difference between the time of the post (`pTm(4)`) and the time (`dTm(1)`) at which the location of Bob's device (`dLoc(1)`) was updated (dotted edge in Fig. 2), `pLoc(4)` and `dLoc(1)` are not dependent. RDRs R2, R4, and R6 are applicable.

Resolving Violating Dependencies. What remains is to resolve the dependencies in $\Delta^-(P2E2, \mathsf{A}(x))$ to ensure that the P2E2 condition for $\mathsf{A}(x)$ 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 $\mathsf{A}(x)$. 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 $\mathsf{A}(x)$, we denote with $\Delta^-(\mathcal{D}_t, \mathsf{A}(x))$ the set of all instantiated relational dependency rules (RDRs) with $\mathsf{A}(x)$ in the head or tail. Observe that to prevent *direct* and *indirect* inferences on $\mathsf{A}(x)$, we need to consider instantiated RDRs in $\Delta^-(\mathcal{D}_t, \mathsf{A}(x))$ as well as all the instantiated attribute functions in $\bigcup_{\delta^- \in \Delta^-(\mathcal{D}_t, \mathsf{A}(x))} Cells(\delta^-)$ besides $\mathsf{A}(x)$, and recursively so on.

Given $\mathsf{A}(x)$ and a database $\mathcal{D}_t$, Algorithm 1 generates the set $\Delta^-(P2E2, \mathsf{A}(x))$ of instantiated RDRs. The algorithm iteratively instantiates RDRs corresponding to direct or indirect inference of

Algorithm 1 Instantiating RDRs for P2E2

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

$\mathsf{A}(\boldsymbol{x})$. 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 $\mathcal{D}_{\kappa(\mathsf{A}(\boldsymbol{x}))^-}$ (Thm. 3.8) and returns the instantiated RDR.

Theorem 4.1. *For an attribute function $\mathsf{A}(\boldsymbol{x})$ with creation time $\kappa(\mathsf{A}(\boldsymbol{x})) = t_b$ and expirationtime $\eta(\mathsf{A}(\boldsymbol{x})) = t_e$, Algorithm 1 produces all dependencies necessary for ensuring P2E2 holds.*

Asymptotic Analysis. Alg. 1 performs a breadth-first traversal over the attribute function dependency graph, starting from a target cell $\mathsf{A}(\boldsymbol{x})$. Let u be the number of unique attribute functions on which $\mathsf{A}(\boldsymbol{x})$ is dependent and $r = |\Delta^-|$ the number of RDR templates. In the worst case, each of the u attributes may match up to r rules, each involving up to a attribute functions. Thus, the total number of instantiated rules is $O(ur)$, and the number of enqueued attributes is $O(ura)$. Since each attribute is visited at most once and each rule is evaluated once per attribute, the overall time complexity is $O(ura)$, and the space complexity is $O(u + ur)$. In practice, the algorithm is efficient when the dependency graph is sparse and rule arity is small.

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 4.2 (Induced Bipartite Graph). For a cell $\mathsf{A}(\boldsymbol{x})$ in a state $\mathcal{D}_t$, given the set $\Delta^-(P2E2, \mathsf{A}(\boldsymbol{x})) = \{\delta_1^-, \ldots, \delta_n^-\}$ of instantiated RDRs, we define the induced bipartite graph $\mathcal{B}(\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))) = (V = V_L \cup V_R, E = E_H \cup E_T)$ where $V_L = \{\delta_i^- | 1 \leq i \leq n\}$ and $V_R = \{c_j \mid c_j \in Cells(\delta_i^-)\}$ are the bipartition of the vertex set V. The set $E = E_H \cup E_T$ of edges 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 $\mathsf{A}(\boldsymbol{x})$, denoted with c_d, for which P2E2 must hold, and for the set $\Delta^-(\mathcal{D}_t, \mathsf{A}(\boldsymbol{x}))$ of instantiated RDRs, the induced bipartite graph $\mathcal{B}(\Delta^-(\mathcal{D}_t, \mathsf{A}(\boldsymbol{x}))) = (V_L \cup V_R, E_H \cup E_T)$. We introduce the following variables: for $\delta_i \in V_L$ a binary variable b_i; for each $c_j \in V_R$, a binary variable a_j; for each

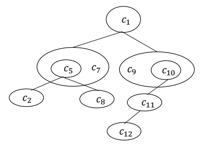


Figure 3: Example of a dependence hypergraph.

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.

Theorem 4.3. *Let $w \in \mathbb{N}$. The following statements are equivalent:*

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

4.3 Dependence Hypergraph

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

Definition 4.4 (Dependence Hypergraph). For a cell $\mathsf{A}(\boldsymbol{x})$ in a database state $\mathcal{D}_t$, given the set $\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))$ of instantiated RDRs, we define the *dependence hypergraph* $\mathcal{H}(\Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))) = (V, E)$ where $V = \bigcup_{\delta_i^- \in \Delta^-(\mathcal{D}_t)} Cells(\delta_i^-)$ is the set of vertices and $E = \{(Head(\delta_i^-), Tail(\delta_i^-)) \mid \delta_i^- \in \Delta^-(\mathcal{D}_t)\}$ is the set of edges.

For a dependence hypergraph $\mathcal{H}(\Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))) = (V, E)$, a vertex $v \in V$ is called a *root* if v is not in the tail of any instantiated RDR, i.e., for all $\delta_i \in \Delta^-(\mathcal{D}_t)$ we have $v \notin Tail(\delta_i)$. Similarly, a vertex $v \in V$ is called a *leaf* if v is not the head of any instantiated RDR, i.e., for all $\delta_i \in \Delta^-(\mathcal{D}_t)$ we have $v \notin Head(\delta_i)$. Fig. 3 shows the dependence hypergraph for the set $\{\delta_1^- : c_1 \not\perp c_5, c_7;\ \delta_2^- : c_5 \perp$

$\not\perp\!\!\!\perp c_2$; $\delta_3^- : c_5 \not\perp\!\!\!\perp c_8$; $\delta_4^- : c_1 \not\perp\!\!\!\perp c_9, c_{10}$; $\delta_5^- : c_{10} \perp\!\!\!\perp c_{11}$; $\delta_6^- : c_{11} \perp\!\!\!\perp c_{12}\}$. The vertex c_1 is a root and c_9 and c_8 are leafs.

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

Definition 4.5. For a dependence hypergraph $\mathcal{H}(\mathcal{D}_t)$, and a sequence $P : v_1, v_2, \dots, v_n$ 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 $P' : v_b, \dots, v_e$ 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 $\mathcal{H}(\mathcal{D}_t)$ 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 [35] 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 $\mathsf{A}(\boldsymbol{x})$, 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[6]. For a hypergraph, with an attribute function $\mathsf{A}(\boldsymbol{x})$ as its root, a bottom-up traversal is required to determine the minimum cost for P2E2 to hold for $\mathsf{A}(\boldsymbol{x})$.

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

Theorem 4.6. *When Algorithm 2 terminates, the set $\mathcal{T}$ contains cells which, when set to NULL, guarantees P2E2 for the input $\mathsf{A}(\boldsymbol{x})$. Moreover, $\sum_{\mathsf{A}_i(\boldsymbol{x}_i)\in\mathcal{T}} Cost(\mathsf{A}_i(\boldsymbol{x}_i))$ is minimized.*

The algorithm consists of two phases: a bottom-up cost propagation to assign minimal deletion costs, followed by a top-down traversal to extract the optimal deletion set. Let $n = |\mathcal{I}|$ be the number of instantiated RDRs, a the maximum rule arity, and $C_{\max}$ the maximum cell cost. The bottom-up phase runs in $O(n \cdot a \cdot \log C_{\max})$ time, and the top-down phase adds $O(C_{\max} \cdot a \cdot \log C_{\max})$ for path extraction. Hence, the total time complexity is $O\left((n + C_{\max}) \cdot a \cdot \log C_{\max}\right)$, with space complexity $O(n \cdot a)$. In practice, the algorithm is efficient due to typically low arity and shallow dependency paths.

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

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).

Theorem 4.7. *Let D_η be the database state at deletion time, and let $c_d = A(x)$ be the cell to be erased. The approximation algorithm in Sec. 4.4 produces a deletion set $\mathcal{T}$ such that P2E2 holds for c_d.*

Asymptotic Analysis. The algorithm greedily constructs a partial dependence hypergraph top-down and avoids the bottom up traversal as in Alg. 2. Since each of the n RDRs (each with maximum arity a) is instantiated at most once for each of its a cells, the total number of instantiated cells is $O(an)$. As the hypergraph is traversed only once, the time complexity is $O(an)$ and the space complexity is also $O(an)$.

Observe that greedily selecting the attribute function with the lowest deletion cost can lead to suboptimal outcomes—for example, forcing the deletion of a high-cost function later. Let $\mathcal{T}$ denote the deletion set from the greedy algorithm and $\mathcal{T}^*$ the optimal set. In general, the cost ratio $\frac{Cost(\mathcal{T})}{Cost(\mathcal{T}^*)}$ can be unbounded. However, when minimizing the cardinality of the deletion set, if the number of cells per instantiated RDR is bounded by arity $a = \max_{\delta^- \in \Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))} |Cells(\delta^-)|$, the approximation ratio is bounded by a. Moreover, in the same setting, if a cell is in at most d instantiated RDRs, then our greedy algorithm provides a $O(\log d)$ approximation.

Theorem 4.8. *Given a dependence hypergraph $\mathcal{H}(\Delta^-(P2E2, \mathsf{A}(\boldsymbol{x})))$= (V, E), let $\mathcal{T}^*$ be the minimal set of cells that need to be deleted to guarantee P2E2 for $\mathsf{A}(\boldsymbol{x})$ and $\mathcal{T}$ be the set of cells to be deleted to guarantee P2E2 with the approximation algorithm (in Sec. 4.4). The following holds: (1) $\frac{|\mathcal{T}|}{|\mathcal{T}^*|} \leq a$ where $a = \max_{\delta^- \in \Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))} |Cells(\delta^-)|$; (2) $\frac{|\mathcal{T}|}{|\mathcal{T}^*|} \leq O(\log d)$ where $d = \max_{c \in \cup_{\delta^- \in \Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))} Cells(\delta^-)} |\{\delta^- | c \in Cells(\delta^-)\}|$;*

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[7] $(\eta^b_1, \eta^e_1), (\eta^b_2, \eta^e_2), \ldots, (\eta^b_n, \eta^e_n)$, 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.

Algorithm 3 Reconstruction Scheduler

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

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 $\{(c_i, \eta^b_i, \eta^e_i) \mid 1 \leq i \leq n\}$ of all its dependencies, their insertion time, and their erasure time, respectively. The MAXOVERLAP function is a standard algorithm to find the maximum overlap given a set of time intervals.

Step 1: The first step finds a reconstruction time ρ that maximizes for $1 \leq i \leq n$ the overlap between the erasure time intervals (η^b_i, η^e_i) 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 $\rho \in (\eta^e_i, \eta^e_i)$. 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.

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

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.

Let n be the number of dependent base cells for a derived cell c. Since the algorithm utilizes a greedy interval scheduling problem, the time complexity is $O(n \log n)$ due to sorting, and the space complexity is $O(n)$ to store intervals and the resulting schedule.

THEOREM 5.1. *Let c be a derived cell, which needs to be reconstructed at least once in freq(c), and has dependencies $c_1, \ldots, c_n$ with erasure time intervals (η_i^b, η_i^e), for $1 \leq i \leq n$. The reconstruction schedule Sch$(c) = [\rho_1, \rho_2, \ldots, \rho_m]$ such that c is erased and reconstructed at time ρ_j, generated in Algorithm 3 ensures the following:*

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

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 4a 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* [22]. 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* [12]. This dataset is a synthetic dataset created using the real-world distribution of values of American tax records. We discovered all present DCs using [44]. 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* [32]. 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* [36]. 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 [4].
(5) *TPC-H* [55]. 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 [50], 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. 4b. 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 and compare our three P2E2 algorithms against three baseline implementations inspired by cascading deletions [27, 29]. These baselines guarantee P2E2 by identifying and deleting dependent cells but, unlike our approaches, they cannot account for the state of the database at the time of insertion of the data being deleted. Each baseline scans the entire dataset but varies in how it selects which cells to delete: INST deletes all dependent (i.e., instantiated) cells, mimicking traditional cascading deletes; OPR improves this and deletes exactly one cell per instantiated RDR; finally, MINSET computes a minimal set of cells such that at least one cell is deleted for each RDR (see Table 4a).

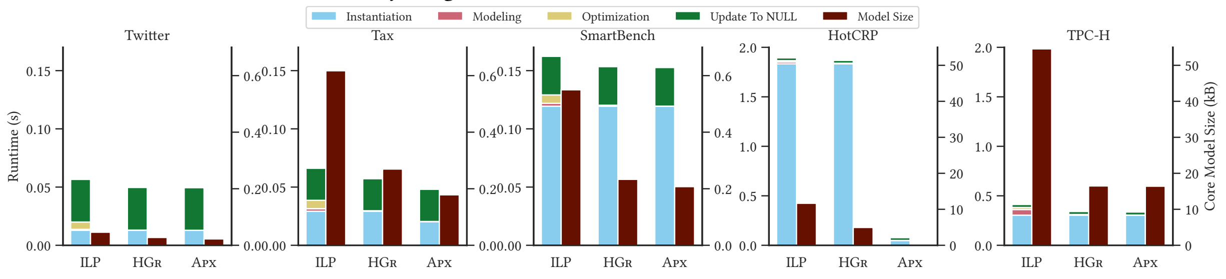
Comparison with baselines: As expected, all three baseline methods delete significantly more data than our P2E2 algorithms on every dataset except Tax. This is because, to guarantee P2E2 for a cell, we only need to consider data that was inserted after the cell and that is dependent on it. In high-volume datasets like Twitter and SmartBench, where a user generates many entries, the overhead of baselines is especially pronounced. In contrast, the Tax dataset does not incur additional deletions under MINSET because all user data is inserted simultaneously.

Analysis of our P2E2 mechanisms: 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

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

Dataset	# Cells	# RDRs (Source)	# base, # derived	Instantiated cells ILP	Instantiated cells HGr	Instantiated cells Apx	Deleted cells ILP	Deleted cells HGr	Deleted cells Apx	Deleted cells for Baselines Inst	Deleted cells for Baselines OpR	Deleted cells for Baselines MinSet
𝕏 (Twitter)[8]	21 926 096	11*	7, 11	0.29	0.29	0.29	0.29	0.29	0.29	837.21	517.8	321.1
Tax [12]	16 000 004	10 ([44])	15, 4	6.93	6.93	5.14	4.07	4.07	4.2	6.93	6.6	4.07
SmartBench [32]	94 424	5*	5, 1	5.43	5.43	5.43	5.43	5.43	5.43	1021.8	1021.8	1021.8
HotCRP [36]	122 697	3([4])	6, 3	134.4	134.4	4.81	3.78	3.78	3.78	815.0	721.7	74.5
TPC-H [55]	5 825 639	6 ([29])	8, 4	17.63	17.63	17.63	17.63	17.63	17.63	722.2	722.2	722.2

(a) Summary of datasets and average number of instantiated & deleted cells. Inst = all instantiated cells, OpR = one cell per rule, MinSet = minimal set, *manually designed.



(b) Average runtime and space overhead

Figure 4: Evaluation of demand-driven erasures

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 the SmartBench, HotCRP, and TPC-H dataset. In contrast, the instantiation time of the Tax dataset is lower although the number of instantiated cells is higher than in the SmartBench dataset. The RDRs in the Tax dataset specify connections only within one row of the data. For each deleted cell, we repeatedly query the same row, which is faster than scanning larger parts of the data as observed in SmartBench. 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 4a).

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.

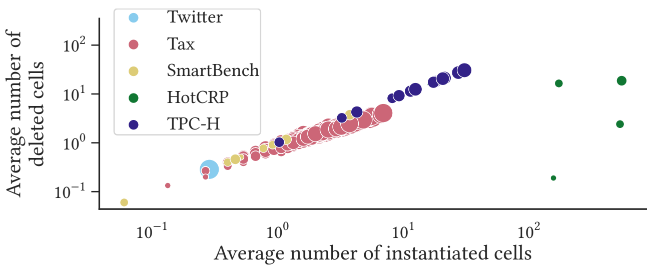


Figure 5: Impact of dependencies (log-axis) on P2E2 overhead; the size of data points signals number of RDRs

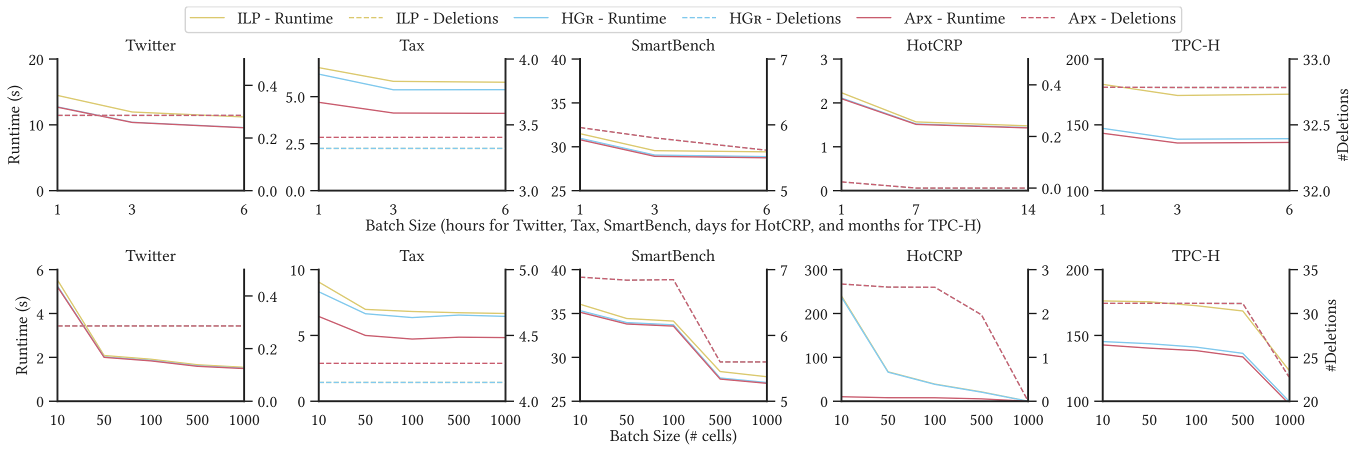


Figure 6: Evaluation of the batching

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 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.

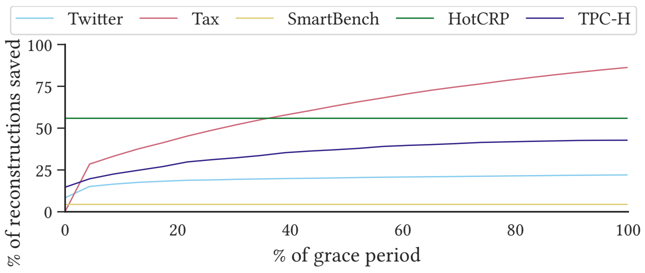


Figure 7: Number of saved reconstructions vs. grace period

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 period, Γ (see Table **??**). 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

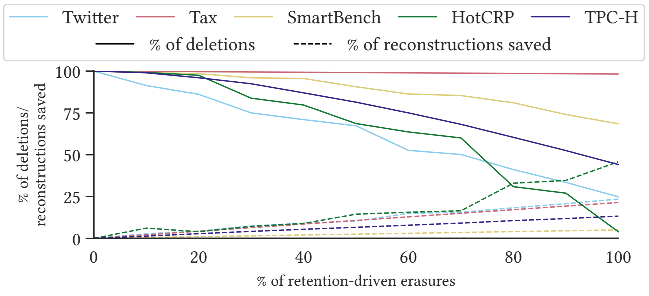


Figure 8: P2E2 overheads for fractions of demand- and retention-driven erasures

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 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 Semantics. Regulation-driven deletion semantics have been studied in several contexts in several prior works: for timely deletion in LSM-tree [50], for cascading deletes in relationship graphs [19], for schema and annotations containing personal data [5], for deletion in blockchains [37], and in SQL context [48] that explores SQL extensions to specify when data should be deleted. None of the work above considers the role of data inference in deletion. The need for formal specification of deletion semantics for regulation compliance has been discussed in [7, 16, 47].

Dependency rules. RDRs draw inspiration from several lines of work in databases that explored dependency frameworks. These include: specification and reasoning frameworks for functional dependencies & denial constraints [46] in data cleaning [46] and consistent query answering [6, 8, 18, 24, 26], correlation constraint and causal constraints in causal databases, delta rules for generalized reasoning for a large class of dependencies (e.g., denial constraints, correlation, and causal constraints) for deletion-based data repair [29], and provenance/lineage dependencies using semiring structures and annotations [17, 30, 31, 56] when available. RDRs extend these to include aggregate dependencies (like summation and max), and cell level dependencies which are essential to define fine grained data deletion. While reasoning frameworks based on above dependency specifications can be exploited to identify minimal deletion sets to make a database consistent to given constraints, they do not provide mechanisms to reason with relative changes in inferences across different database states (e.g., insertion and deletion states of a data). Techniques for discovering dependencies from data have also been explored extensively [4, 10, 11, 34, 43] which RDRs can express.

Deletion in ML. Recent regulatory efforts, such as the AI Act [23], raise deletion requirements in machine learning contexts. Naively deleting training data implies model retraining, which is often impractical [14]. Approaches such as SISA [13] propose partial retraining, while others apply differential privacy [60] to bound information leakage. Extending RDRs and P2E2 to cover dependencies between data and learned models remains an open and promising direction.

8 CONCLUSIONS AND FUTURE WORK

We formalize safe data erasure as Pre-Insertion Post-Erasure Equivalence (P2E2), which resolves semantic ambiguity by providing strong guarantees on deleted data—filling a key gap in current systems [7]. We implement P2E2 using Relational Dependency Rules (RDRs), developing both exact and approximate algorithms.

While P2E2 provides a principled foundation for compliant data deletion, its adoption faces two key challenges. First, data dependencies may necessitate deleting more than what is requested to be erased. However, our evaluation across five domains—including highly dependent datasets—shows that our approaches effectively reduces this overhead. Future extensions may incorporate selectively applying P2E2 to data subsets, weighting dependencies, domain-specific inference, or relaxed variants of P2E2 enabling flexible trade-offs between deletion cost and retention obligations

Second, P2E2 relies on a specified set of dependencies through which deleted data can be inferred. While RDRs provide a structured way to encode known dependencies, discovering them may incur an overhead. In practice, such dependencies often exist elsewhere in the pipeline—e.g., business logic, analytics, consistency checks, or data cleaning—or can often be learned from data [4, 11, 43]. This requirement is consistent with the "reasonableness" standard in data regulations, which calls for reasonable, cost-effective conformative measures given current technology and implementation constraints. An interesting future direction is to extend P2E2 to protect against not only specified dependencies but any potential inferences, perhaps offering weaker, probabilistic guarantees.

Beyond refining P2E2 extensions and evaluating its applicability across domains, a key direction for future work is to extend deletion guarantees to data processing pipelines, where dependencies span multiple system components.

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] 2003. HIPAA Security Rule - 45 CFR §164.310(d)(2)(i). https://www.ecfr.gov/current/title-45/subtitle-A/subchapter-C/part-164/subpart-C/section-164.310 U.S. Code of Federal Regulations.

[2] 2012. Personal Data Protection Act 2012 (PDPA) - Part IV: Retention Limitation Obligation. https://sso.agc.gov.sg/Act/PDPA2012 Singapore Statutes Online.

[3] 2018. Lei Geral de Proteção de Dados (LGPD) - Article 18(IV). https://www.gov.br/cidadania/pt-br/acesso-a-informacao/lgpd Brazilian Federal Law No. 13,709/2018.

[4] 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

[5] 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.

[6] 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.

[7] 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).

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

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

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

[11] 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.

[12] 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

[13] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine Unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*. 141–159. https://doi.org/10.1109/SP40001.2021.00019

[14] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. 2022. Membership inference attacks from first principles. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 1897–1914.

[15] 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.

[16] 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).

[17] 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

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

[19] 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.

[20] 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

[21] Ghana Data Protection Commission. 2012. Data Protection Act. https://nca.org.gh/wp-content/uploads/2020/09/Data-Protection-Act-2012.pdf Accessed: 2025-04-02.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2009. *Database systems - the complete book (2. ed.)*. Pearson Education.

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

[29] 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.

[30] 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

[31] 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.

[32] 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.

[33] 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

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] 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

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

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

[42] 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.

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

[44] 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.
[45] 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.
[46] 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
[47] 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.
[48] 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.
[49] 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
[50] 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
[51] 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
[52] 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
[53] David W Shipman. 1981. The functional data model and the data languages DAPLEX. *ACM Transactions on Database Systems (TODS)* 6, 1 (1981), 140–173.
[54] Slawomir Staworko. 2007. Declarative Inconsistency Handling in Relational and Semi-Structured Databases. *PhD Thesis* (2007).
[55] Transaction Processing Performance Council. 2021. *TPC-H Benchmark Specification, Version 2.17.3.* Technical Report. Transaction Processing Performance Council. http://www.tpc.org/tpch/
[56] 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
[57] 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.
[58] 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
[59] WhatsApp. 2020. About Disappearing Messages. https://faq.whatsapp.com/673193694148537/?helpref=uf_share, last accessed on 2025-01-10.
[60] Lefeng Zhang, Tianqing Zhu, Haibin Zhang, Ping Xiong, and Wanlei Zhou. 2023. FedRecovery: Differentially Private Machine Unlearning for Federated Learning Frameworks. *IEEE Transactions on Information Forensics and Security* 18 (2023), 4732–4746. https://doi.org/10.1109/TIFS.2023.3297905

A APPENDIX

We present the technical results in the paper and prove them.

- Section A.1 presents the hardness proof for the Opt-P2E2 problem defined in Section 3.3 of the paper.
- Section A.2 presents two technical results used in Section 3.4.
- Section B presents the technical results related to the dependence hyper graph characterization of P2E2 (Section 4.3)
- Finally, Section C contains the correctness proofs of all the algorithms.

A.1 Hardness of Opt-P2E2.

We reduce an instance of the Set Cover problem to an instance of the Opt-P2E2 problem to show NP-hardness.

Theorem A.1. *(Reproduced Thm. 3.7)*
The Opt-P2E2 problem is NP-hard.

Proof. We prove this by reduction from the Set Cover problem. Consider an instance of Set Cover with universe U and collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$. We construct an instance of our minimal deletion set problem as follows.

For each element $u \in U$, we create a dependency δ_u^- such that: $Head(\delta_u^-) = \{\mathsf{A}(\boldsymbol{x})\}$ $Tail(\delta_u^-) = \{c_u\} \cup \{s_i \mid u \in S_i\}$ where c_u is a unique element with $\kappa(c_u) > t_b$, and each s_i corresponds to set S_i with $\kappa(s_i) > t_b$.

Let Δ^- be the set of all such dependencies. By construction:

(1) Each $\delta_u^- \in \Delta^-$ must be resolved to satisfy P2E2
(2) Resolving δ_u^- requires removing at least one cell from $Tail(\delta_u^-)$
(3) The cells s_i correspond to sets in the original Set Cover instance

We now prove that there exists a set cover of size k if and only if there exists a valid deletion set $\mathcal{T}'$ of size k.

($\Rightarrow$) Suppose there exists a set cover C of size k. Let $\mathcal{T}' = \{s_i \mid S_i \in C\}$ Since C is a set cover, each $u \in U$ is contained in some $S_i \in C$. Therefore, each dependency δ_u^- has at least one cell from its tail in $\mathcal{T}'$, making $\mathcal{T}'$ a valid deletion set of size k.

($\Leftarrow$) Suppose there exists a valid deletion set $\mathcal{T}'$ of size k. Let $C = \{S_i \mid s_i \in \mathcal{T}'\}$ Since $\mathcal{T}'$ must Resolve each dependency δ_u^-, it must contain at least one s_i from each $Tail(\delta_u^-)$. Therefore, C covers all elements in U and has size at most k.

This polynomial-time reduction from Set Cover, which is NP-hard, proves that finding the minimum size deletion set is also NP-hard. □

A.2 Construction of $\Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))$

In this section we present the two technical results related to the construction of the set of instantiated RDRs (Section 3.4).

Theorem A.2. *(Reproduced Thm 3.8)*
Let $\mathsf{A}(\boldsymbol{x})$ be a cell with $\kappa(\mathsf{A}(\boldsymbol{x})) = t_b$ and $\eta(\mathsf{A}(\boldsymbol{x})) = t_e$. Let E (and N) be the set of erased (inserted, respectively) cells between the states $\mathcal{D}_{t_b}$ and $\mathcal{D}_{t_e}$. The following holds:

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

Proof. We will prove both parts of the theorem separately, starting with the dependency subset relationship and then proving the equality for new dependencies.

First, we prove that $dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b} - E) \subseteq dep(\mathsf{A}(\boldsymbol{x} \mid \mathcal{D}_{t_b})$. Consider any dependency δ that exists in $dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b} - E)$. This dependency must use only cells that exist in state $\mathcal{D}_{t_b} - E$. Since $\mathcal{D}_{t_b} - E$ is a subset of $\mathcal{D}_{t_b}$, all cells involved in δ must also exist in $\mathcal{D}_{t_b}$. The removal of cells cannot create new probabilistic dependencies that didn't exist in the original state. Therefore, δ must have been a valid dependency in $\mathcal{D}_{t_b}$, proving that $\delta \in dep(\mathsf{A}(\boldsymbol{x} \mid \mathcal{D}_{t_b})$.

Next, we prove the equality concerning new dependencies:

$$dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e}) - dep(\mathsf{A}(\boldsymbol{x} \mid \mathcal{D}_{t_b} - E) =$$
$$\{\delta_i^- \mid \delta_i^- \in dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e}) \wedge \exists c \in Cells(\delta_i^-) \text{ s.t. } \kappa(c) > t_b\}$$

We will prove this equality by demonstrating both set inclusions. For the forward inclusion, consider any dependency δ in the left-hand side of the equation. Since δ is not in $dep(\mathsf{A}(\boldsymbol{x} \mid \mathcal{D}_{t_b} - E)$, it must involve at least one cell that is not present in $\mathcal{D}_{t_b} - E$. This cell must either be in the set E of erased cells or must have been created after time t_b. If all such cells were in E, then δ would have been a dependency in $\mathcal{D}_{t_b}$. Therefore, there must exist at least one cell c in $Cells(\delta)$ such that $\kappa(c) > t_b$, placing δ in the right-hand side of the equation.

For the reverse inclusion, consider any dependency δ that involves a cell c where $\kappa(c) > t_b$. By definition, δ is in $dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e})$. Since δ involves a cell created after time t_b, it cannot exist in $dep(\mathsf{A}(\boldsymbol{x} \mid \mathcal{D}_{t_b} - E)$. Therefore, δ must be in the set difference on the left-hand side of the equation.

This completes the proof of both set inclusions, establishing the equality. The theorem demonstrates two fundamental properties of our dependency model: the erasure of cells can only remove dependencies, never create new ones, and any new dependencies that arise at time t_e must necessarily involve at least one cell that was created after time t_b. □

Theorem A.3 (Resolving Dependencies). *Given an attribute function $\mathsf{A}(\boldsymbol{x})$ with $\kappa(\mathsf{A}(\boldsymbol{x})) = t_b$ and $\eta(\mathsf{A}(\boldsymbol{x})) = t_e$, let $\Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})) = \{\delta_i^- \mid \delta_i^- \in dep(\mathsf{A}(\boldsymbol{x})) \mid \mathcal{D}_{t_e}) \cup dep(\mathsf{A}(\boldsymbol{x})) \mid \mathcal{D}_{t_b})$ and $\exists c \in Cells(\delta_i^-)$ s.t. $\kappa(c) > t_b\}$ and $\mathcal{T}^* = \bigcup_{\delta_i^- \in \Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))} \big((Head(\delta_i^-) \cap \mathcal{D}_{t_e}) \cup (Tail(\delta_i^-) \cap \mathcal{D}_{t_e})\big)$. Then P2E2 holds for $\mathsf{A}(\boldsymbol{x})$.*

Proof. Let us prove that removing at least one cell from each dependency in Δ^- is sufficient to ensure P2E2 holds for $\mathsf{A}(\boldsymbol{x})$. We will construct an alternative set $\mathcal{T}'$ that contains just one cell from each dependency and show this is sufficient.

Let us define $\mathcal{T}'$ as follows: For each $\delta_i^- \in \Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x}))$, select any cell $c_i \in (Head(\delta_i^-) \cup Tail(\delta_i^-)) \cap \mathcal{D}_{t_e}$. Then: $\mathcal{T}' = \{c_i \mid \delta_i^- \in \Delta^-(\mathcal{D}_t, P2E2(\mathsf{A}(\boldsymbol{x})))\}$

We will prove that this construction ensures: $dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e} - \mathcal{T}') \subseteq dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$

Consider any dependency δ in $dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e} - \mathcal{T}')$. As in our previous proof, there are two cases to consider:

Case 1: If δ involves no cells created after t_b, then δ must have existed in $\mathcal{D}_{t_b}$ and thus belongs to $dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$.

Case 2: If δ involves any cell created after t_b, then $\delta \in \Delta^-$. By our construction of $\mathcal{T}'$, we have removed at least one cell required for this dependency. Since dependencies require all cells to be present (by the complete set property), removing even one cell resolves the dependency. Therefore, δ cannot exist in $\mathcal{D}_{t_e} - \mathcal{T}'$.

This proves that any dependency remaining in $\mathcal{D}_{t_e} - \mathcal{T}'$ must have existed in $\mathcal{D}_{t_b}$. Furthermore, this construction may be more efficient than removing all cells from each dependency, as it requires removing only one cell per dependency while still ensuring P2E2 holds.

The key insight is that dependencies require all their cells to be present to exist, so removing any single cell from a dependency is sufficient to resolve it. This allows us to maintain P2E2 while potentially removing fewer cells than the previous construction. □

B DEPENDENCE HYPERGRAPH

Results corresponding to Section 4.3 in the paper.

B.1 Sufficient and Necessary Conditions

Theorem B.1. *For a database state $\mathcal{D}_t$, cell $c_d \in \mathcal{D}_t$, and a set Δ^- of RDRs, let $\mathcal{H}(\Delta^-(\mathcal{D}_t, P2E2(c_d))) = (V, E)$ be a dependence hypergraph with c_d as its root, let $\mathcal{P}$ be a set of complete paths in $\mathcal{H}(\Delta^-(\mathcal{D}_t, P2E2(c_d)))$, and let $\mathcal{T} = Cells(\mathcal{P})$. P2E2 holds for c_d when all such units c in $\mathcal{T}$ are set to NULL if the following holds:*

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

Proof. To prove Theorem B.1, we will proceed in two parts. First, we will prove that if the conditions stated in the theorem hold, then P2E2 holds for c_d. Then we will prove the converse - that is, if P2E2 holds for c_d, then the conditions in the theorem must be true.

Part 1: Conditions $\implies$ P2E2

Assume the conditions in the theorem hold. Consider any instantiated rule δ_i^- that could potentially be used to infer the value of c_d. Let $P = v_1, v_2, \ldots, v_n$ be the complete path corresponding to applying the sequence of rules to infer c_d, with $v_1 \in Tail(\delta_i^-)$ and $v_n = c_d$.

By the second condition, since v_1 is in a complete path, there must be some $v' \in Tail(\delta_i^-)$ such that v' is the second vertex in some complete path $P' \in \mathcal{P}$. But by definition of $\mathcal{T}$, this means $v' \in \mathcal{T}$ and thus v' has been set to *NULL*.

Therefore, the rule δ_i^- cannot be applied, because one of the cells in its tail has a *NULL* value. Since this holds for any such rule δ_i^-, there is no way to infer a value for c_d. Thus, P2E2 holds for c_d.

Part 2: P2E2 $\implies$ Conditions

Now assume P2E2 holds for c_d when all cells in $\mathcal{T}$ are set to *NULL*. We will show the two conditions in the theorem must hold.

First, suppose there is no complete path from c_d. Then there would be no way to infer a value for c_d even if all cells were non-*NULL*, contradicting the assumption that setting $\mathcal{T}$ to *NULL* was necessary and sufficient for P2E2. Thus, the first condition must hold.

For the second condition, suppose there is some $v \in V$ in a complete path, and some edge $(v, T) \in E$, such that for all complete paths P containing v, the second vertex after v is not in T. Let δ be the rule corresponding to edge (v, T).

Consider the database state where all cells in $\mathcal{T}$ are *NULL*, but v and all cells in T have non-*NULL* values (this is possible since by assumption, T does not intersect the second vertex of any complete path from v). Then the rule δ can be applied to infer a value for the head cell of δ, say v'.

If v' is not on any complete path, this does not immediately contradict P2E2 for c_d. However, by applying the same argument repeatedly, we can construct a sequence of inferences that eventually reaches c_d, since v was assumed to be on a complete path. This contradicts the assumption that P2E2 holds for c_d. Thus, the second condition must hold.

By proving both directions, we have shown that the conditions stated in the theorem are both necessary and sufficient for P2E2 to hold for c_d when the cells in $\mathcal{T}$ are set to *NULL*. This completes the proof. □

C PROOFS FOR ALGORITHMS

In this section, we present the technical results for the algorithms presented in the paper.

C.1 Instantiation: Algorithm 1

Theorem C.1. *(Reproduced Thm. 4.1) For an attribute function* $\mathsf{A}(\boldsymbol{x})$ *with creation time* $\kappa(\mathsf{A}(\boldsymbol{x})) = t_b$ *and expirationtime* $\eta(\mathsf{A}(\boldsymbol{x})) = t_e$*, Algorithm 1 produces all dependencies necessary for ensuring P2E2 holds.*

Proof. We will prove the correctness of the DepInst algorithm by demonstrating that it discovers all dependencies required for P2E2 while maintaining computational feasibility. Our proof builds on previous results about the structure of dependencies required for P2E2.

First, recall from our earlier characterization theorem that the dependencies required for P2E2 are precisely those $\delta \in dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_e}) \cup dep(\mathsf{A}(\boldsymbol{x}) \mid \mathcal{D}_{t_b})$ where there exists a cell $c \in Cells(\delta)$ such that $\kappa(c) > t_b$. We will show that Algorithm 1 discovers exactly these dependencies through three key properties.

The first property concerns proper dependency instantiation. The algorithm employs the $eval()$ function which instantiates only those dependencies where all required cells are non-null in the current database state and at least one cell was created after time t_b. This ensures that every dependency instantiated by the algorithm satisfies our theoretical requirements for P2E2 dependencies.

For the second property of complete exploration, let us prove by contradiction that the algorithm discovers all relevant dependencies. Assume there exists a necessary dependency δ that the algorithm fails to discover. By the nature of dependencies, δ must either directly involve $\mathsf{A}(\boldsymbol{x})$ or be reachable through a chain of dependencies starting from $\mathsf{A}(\boldsymbol{x})$. In the first case, δ would be discovered during the initial processing of $\mathsf{A}(\boldsymbol{x})$, as the algorithm evaluates all rules in Δ^- for this attribute.

In the second case, there must exist a sequence of dependencies $\delta_1^-, \delta_2^-, ..., \delta_n^- = \delta$ where $\mathsf{A}(\boldsymbol{x}) \in Head(\delta_1^-)$ and $Head(\delta_{i+1}^-) \cap Tail(\delta_i^-) \neq \emptyset$ for all i. The breadth-first search nature of the algorithm, implemented through the queue Q, ensures that all tail attributes of discovered dependencies are explored. Therefore, if such a chain exists, the algorithm would systematically discover each δ_i^- in sequence, eventually reaching δ. This contradicts our assumption.

The third property ensures termination. The algorithm maintains the set $\mathcal{V}$ of visited attributes and processes each attribute exactly once. Since the set of attributes is finite, each attribute is processed exactly once, and rule evaluation produces a finite number of tail attributes, the algorithm must terminate in finite time.

Combining these properties, we can conclude that upon termination, the sets $\mathcal{V}$ and $\mathcal{I}$ contain exactly those attributes and dependencies that are reachable from $\mathsf{A}(\boldsymbol{x})$ through dependency chains, involve at least one cell created after t_b, and have all required cells present in the current database state. By our previous theorem characterizing P2E2 dependencies, these are precisely the dependencies required to ensure P2E2 holds for $\mathsf{A}(\boldsymbol{x})$. Therefore, Algorithm 1 correctly identifies all dependencies necessary for maintaining P2E2. □

C.2 ILP-Reduction

In this section, we present the proof of correctness of our reduction in Section 4.2. We want to show that a 0-1 solution to O with $W = w$, where $w \in \mathbb{N}$, is the optimal number of cells that need to be erased to ensure that P2E2 holds for c_d.

Theorem C.2. *(Reproduced Thm. 4.3) Let* $w \in \mathbb{N}$*. The following statements are equivalent:*

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

Proof. We will prove the correctness of the ILP formulation by demonstrating that any feasible solution to the ILP corresponds to a valid solution to the P2E2 problem, and that an optimal solution to the ILP yields an optimal solution to P2E2.

First, let us consider a feasible solution $S = \{a_j \mid a_j = 1\}$ to the ILP. We claim that erasing all cells $c_j \in S$ satisfies P2E2 for cell c_d. Constraint 1 ensures that $c_d \in S$, so c_d is erased. For any instantiated RDR δ_i^- with c_d in its head, Constraint 2 sets $b_i = 1$. Then, Constraint 4 ensures that some cell c_j in the tail of δ_i^- has $t_i^j = 1$. By Constraint 5, this cell c_j is in S. Therefore, δ_i^- cannot infer c_d. For any other cell $c_j \in S$, Constraint 2 sets $b_i = 1$ for all δ_i^- with c_j in the head. The same argument as before shows that these δ_i^- cannot infer c_j. Thus, erasing the cells in S satisfies P2E2 for c_d, and any feasible ILP solution corresponds to a valid P2E2 solution.

Now, let us consider an optimal solution S^* to the ILP with objective value W^*. Suppose, for the sake of contradiction, that there exists a solution S' to P2E2 with $|S'| < |S^*|$. We can define an assignment of ILP variables as follows: $a_j = 1$ if and only if $c_j \in S'$, $b_i = 1$ if and only if the head of δ_i^- is in S', $h_i^j = 1$ if and only if $a_j = 1$ and c_j is the head of δ_i^-, and $t_i^j = 1$ if and only if $a_j = 1$ and c_j is in the tail of δ_i^-.

We claim that this assignment is a feasible solution to the ILP. Constraint 1 holds since $c_d \in S'$, Constraints 2 and 3 hold by definition of h_i^j, Constraint 4 holds since for $b_i = 1$, some tail cell of δ_i^- is in S', and Constraint 5 holds by definition of t_i^j. The objective value of this solution is $\sum a_j = |S'| < |S^*| = W^*$, contradicting the optimality of S^*. Thus, no such S' exists, and an optimal solution to the ILP must correspond to an optimal solution to P2E2.

In summary, we have shown that any feasible solution to the ILP yields a valid solution to P2E2, and an optimal ILP solution must be an optimal P2E2 solution. Therefore, the ILP formulation is correct and will find a minimum set of cells to erase to satisfy P2E2 for the given cell c_d. The key steps in the proof were demonstrating how the ILP constraints map to the P2E2 requirements and proving optimality via contradiction. □

C.3 Hypergraph Based Algorithm (Sec. 4.3)

Theorem C.3. *(Reproduced Thm. 4.6) When Algorithm 2 terminates, the set* $\mathcal{T}$ *contains cells which, when set to NULL, guarantees P2E2 for the input* $\mathsf{A}(\boldsymbol{x})$*. Moreover,* $\sum_{\mathsf{A}_i(\boldsymbol{x}_i) \in \mathcal{T}} Cost(\mathsf{A}_i(\boldsymbol{x}_i))$ *is minimized.*

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

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

Optimality: In this proof, for a node v in the hypergraph, $Cost(v)$ refers to that computed in Line 12 of Algorithm 2. For contradiction, assume that there exists a set $\mathcal{T}'$ of cells with $\mathcal{T} \neq \mathcal{T}'$, such that $\sum_{c \in \mathcal{T}'} Cost(c) < \sum_{c \in \mathcal{T}} Cost(c)$ and when cells in $\mathcal{T}'$ are set to *NULL*, they guarantee P2E2 for c_d. Let the complete path corresponding to $\mathcal{T}$ be $P : c_d, v_2, \ldots, v_n$ and that corresponding to $\mathcal{T}'$ be $P' : c_d, v'_2, \ldots, v'_m$. There exists an index k such that for all $2 \leq i < k$, we have $v_i = v'_i$, node $v_k \neq v'_k$. This implies, that $Cost(v'_k) < Cost(v_k)$. Since $v_k \in \mathcal{T}$, it must be the case due to Line 29, that $Cost(v_k)$ is the minimum. Therefore, $Cost(v'_k) > Cost(v_k)$ which is a contradiction. □

C.4 Approximate Algorithm

Theorem C.4. *(Reproduced Thm. 4.7) Let D_η be the database state at deletion time, and let $c_d = A(\boldsymbol{x})$ be the cell to be erased. The approximation algorithm in Sec. 4.4 produces a deletion set T such that P2E2 holds for c_d.*

Proof. Let $\Delta^-(\text{P2E2}, c_d)$ be the set of instantiated RDRs such that for each $\delta^- \in \Delta^-$, we have $c_d \in \text{Head}(\delta^-)$, and there exists $c_i \in \text{Tail}(\delta^-)$ with $\kappa(c_i) > \kappa(c_d)$. Each such δ^- enables a new inference of c_d at time η that did not exist at insertion time κ.

To eliminate the dependency induced by δ^-, it suffices to nullify either the head c_d or any cell $c_i \in \text{Tail}(\delta^-)$. The approximation algorithm iterates over all $\delta^- \in \Delta^-(\text{P2E2}, c_d)$ and, for each rule, greedily deletes the minimum-cost tail cell.

Thus, for each $\delta^- \in \Delta^-$, at least one participating cell is erased in D^+_η, ensuring $\delta^- \notin \text{dep}(c_d \mid D^+_\eta)$. Therefore, all dependencies introduced after $\kappa(c_d)$ are removed, and the condition for P2E2 is satisfied. □

Theorem C.5. *(Reproduced Thm. 4.8) Given a dependence hypergraph $\mathcal{H}(\Delta^-(P2E2, \mathsf{A}(\boldsymbol{x})))= (V, E)$, let $\mathcal{T}^*$ be the minimal set of cells that need to be deleted to guarantee P2E2 for $\mathsf{A}(\boldsymbol{x})$ and $\mathcal{T}$ be the set of cells to be deleted to guarantee P2E2 with the approximation algorithm (in Sec. 4.4). The following holds: (1) $\frac{|\mathcal{T}|}{|\mathcal{T}^*|} \leq a$ where $a = \max_{\delta^- \in \Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))} |Cells(\delta^-)|$; (2) $\frac{|\mathcal{T}|}{|\mathcal{T}^*|} \leq O(\log d)$ where $d = \max_{c \in \cup_{\delta^- \in \Delta^-(P2E2, \mathsf{A}(\boldsymbol{x}))} Cells(\delta^-)} |\{\delta^- | c \in Cells(\delta^-)\}|$;*

Proof. We assume a unit-cost model, so for any deletion set $\mathcal{T} \subseteq \mathcal{V}$, the cost is $|\mathcal{T}|$. Both algorithms aim to delete a target node $t \in \mathcal{V}$ by invalidating all derivations of t via the rules in $\mathcal{I}$.

In the greedy algorithm, starting from t, we select a rule $\delta^- \in \mathcal{I}$ such that $t \in \text{Head}(\delta^-)$, and choose a single node $u \in \text{Tail}(\delta^-)$ to delete. This process recurses: from u, we again pick a rule where u is in the head, select one tail node, and continue until reaching a leaf. Thus, each recursive step contributes exactly one node to $\mathcal{T}_{\text{greedy}}$, and the size of $\mathcal{T}_{\text{greedy}}$ equals the number of steps in this derivation chain.

Let $k = |\mathcal{T}_{\text{greedy}}|$ be the number of such recursive deletions. For each such step, the optimal algorithm could have invalidated the same rule δ^- by deleting *any one* of the up to a tail nodes, since $|\text{Tail}(\delta^-)| \leq a$. That is, for each deletion in the greedy path, the optimal solution could have made a choice among up to a alternatives.

Therefore, each node deleted by the greedy algorithm corresponds to a subset of size at most a in which the optimal solution could choose one node to invalidate the same rule. In the worst case, the optimal solution covers k such steps with disjoint subsets of size a, requiring at least $\lceil k/a \rceil$ deletions. Thus: $|\mathcal{T}_{\text{opt}}| \geq \frac{|\mathcal{T}_{\text{greedy}}|}{a} \Rightarrow \frac{|\mathcal{T}_{\text{greedy}}|}{|\mathcal{T}_{\text{opt}}|} \leq a.$

Hence, the greedy algorithm achieves an a-approximation to the optimal deletion set under the unit-cost model. □

C.5 Algorithm 3

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

Theorem C.6. *(Reproduced Thm. 5.1) Let c be a derived cell, which needs to be reconstructed at least once in $freq(c)$, and has dependencies $c_1, \ldots, c_n$ with erasure time intervals (η^b_i, η^e_i), for $1 \leq i \leq n$. The reconstruction schedule $Sch(c) = [\rho_1, \rho_2, \ldots, \rho_m]$ such that c is erased and reconstructed at time ρ_j, generated in Algorithm 3 ensures the following:*

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

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