# Cohesive database neighborhoods for differential privacy: mapping relational databases to RDF

Sara Taki, Adrien Boiret, Cédric Eichler, Benjamin Nguyen

# Cohesive database neighborhoods for differential privacy: mapping relational databases to RDF*

Sara Taki[1], Adrien Boiret[1], Cédric Eichler[1], and Benjamin Nguyen[1]

Laboratoire d'Informatique Fondamentale d'Orléans, INSA Centre Val de Loire, Université d'Orléans, Inria, Bourges, France
{sara.taki,adrien.boiret,cedric.eichler,benjamin.nguyen}@insa-cvl.fr

**Abstract.** The Semantic Web represents an extension of the current web offering a metadata-rich environment based on the Resource Description Format (RDF) which supports advanced querying and inference. However, relational database (RDB) management systems remain the most widespread systems for (Web) data storage. Consequently, the key to populating the Semantic Web is the mapping of RDB to RDF, supported by standardized mechanisms. Confidentiality and privacy represent significant barriers for data owners when considering the translation and subsequent utilization of their data. In order to facilitate acceptance, it is essential to build privacy models that are equivalent and explainable within both data formats.
Differential Privacy (DP) has emerged to be the flagship of data privacy when sharing or exploiting data. Recent works have proposed DP-models tailored for either multi-relational databases or RDF. This paper leverages this field of work to study how privacy guarantees on RDB with foreign key constraints can be transposed to RDF databases and vice versa.
We consider a promising DP model for RDB related to cascade deletion and demonstrate that it is sometimes similar to an existing DP graph privacy model, but inconsistently so. Consequently, we tweak this model in the relational world and propose a new model called restrict deletion. We show that it is equivalent to an existing DP graph privacy model, facilitating the comprehension, design and implementation of DP mechanisms in the context of the mapping of RDB to RDF.

**Keywords:** Differential Privacy · Relational Databases · Graph Databases · RDB2RDF mapping.

## 1 Introduction

The Semantic Web represents an extension of the current web standardized by the World Wide Web Consortium. It relies on the Resource Description Format (RDF) and provides metadata-rich, reusable, and shareable data. RDF can be

* Supported the French National Research Agency, under grants ANR-18-CE23-0010 'SENDUP', ANR-23-CMAS-001-CYBERINSA and ANR-22-PECY-0002 'iPoP

coupled with ontologies such as OWL, thereby enhancing the semantic value of the data and inference capabilities. Currently, vast volumes of data still reside in relational databases (RDB), and relational databases management systems (RDBMS), such as Oracle or PostgreSQL, remain largely the most popular systems to manage data[1].

Mapping these bulk of data from relational databases to RDF is therefore a key to populate the Semantic Web. It has been an active field of research during the last two decades [1,2,3] initiated by the RDB2RDF (Relational Database to Resource Description Format) incubator group[2].

Data collected (whether stored in RDB or RDF) can contain sensitive information. With the increasing attention on data privacy and the development of privacy regulations (e.g., the General Data Protection Regulation in the European Union), it is becoming increasingly important to protect sensitive information when sharing or allowing the utilization of data. Such concerns are a significant obstacle for RDB holders in accepting the translation and subsequent utilization of their data, as protection models vastly differ between RDF and RDB formats. *It is therefore crucial to construct models that are equivalent and explainable within both formats, easing the comprehension of the guarantees provided in RDF within the familiar context of RDB.*

Differential Privacy (DP) [4] provides a form of indistinguashibility: it is difficult for an entity observing the output of a DP algorithm to determine which of several *adjacent* or *neighboring* databases was used as input. If two neighboring databases differ by the contribution of an individual, an external entity may therefore not know with high confidence whether the data pertaining to a particular individual has been used. Hence, it may not infer anything significant on such data. The concept of adjacency is thus a cornerstone of DP, defining what is protected. In the most simple context, a database is a single, monolithic table of records (or tuples) that holds private data. In this context, neighboring databases are those that differ by one record, meaning that DP protects (or "hides") the presence or absence of any single record in the database [4]. The intuition here is that each individual participates in, at most, one database record and therefore DP indeed protects the contribution of each individual.

Defining neighborhoods for multi-relational databases, i.e., databases composed of many tables, is challenging for many reasons (see, for instance, [5]). Indeed, the introduction of several relations usually comes with *constraints*, each constraint stemming from the semantics of the database. It is thus no longer possible to define an adjacent database simply by adding or removing a tuple in a table since this may violate the database constraints and thus not be an acceptable instance. In this paper, we consider an important type of constraint, foreign key (FK) constraints, sometimes associated to cardinality constraints.

Therefore, this paper focus on distances and neighborhoods in the RDB and RDF worlds that are equivalent through standard translation mechanisms to

[1] See : https://db-engines.com/en/ranking
[2] http://www.w3.org/2001/sw/rdb2rdf/

build DP-models that are equivalent in both. More specifically, we focus on two research questions:

RQ1 What does transitive deletion, a state of the art DP model for multi-RDB with FK constraints, leads to in RDF?
RQ2 Is it possible to construct a meaningful DP model in RDB equivalent to a state-of-the art model in RDF, making existing RDF DP mechanism explainable in the context of RDB?

**Contributions**. To answer these research questions, this paper:

- formalizes the notion of encoding (or mapping) from RDB to RDF in a standard-compliant fashion consistent with R2RML mapping.
- formalizes a generalized notion of cascade deletion in RDB covering both transitive deletion and our proposal.
- studies the translation in RDF of the DP model relying on transitive deletion.
- introduces a meaningful relaxation of this model and demonstrates that it is equivalent to an existing graph privacy model through mapping.

The remainder of this paper is structured as follows. The next section details relevant related works on mapping and DP. Section 3 introduces an illustrative example as well as the formalization of the considered databases and mappings. Section 4 proposes an analysis of the translation of transitive deletion in RDF and details our proposed relaxation, demonstrating its equivalence to a well known DP model in graphs through mapping. Finally, section 5 concludes this paper and discuss future work.

## 2 Related Work

This section introduces background on RDB to RDF mapping, before introducing DP and its adaptations to RDB and RDF databases. To the best of our knowledge, this paper is the very first at the intersection of these two fields, focusing on the impact of the RDB to RDF translation on DP-models and the definition of equivalent DP-models in both worlds.

### 2.1 Mapping RDB to RDF

In September 2012, the RDB2RDF Working Group published two Recommendations: Direct Mapping (DM) [6] and customized mapping (CM) R2RML [7].

The W3C DM recommendation defines simple mapping rules to map relational data to RDF [8]. The RDF generated straightforwardly is based on the structure of the database schema. URIs are automatically generated [9]. Many-to-many relations in relational databases are generally represented as a join table where all its columns are FKs to other tables (n-ary relations). One missing part from the DM is to represent many-to-many relation as simple triples [10]. When

DM is applied, the join table will be translated into a distinct class, which conflicts with the canonical representation of many-to-many relationship in RDF.

CM R2RML [7] is a RDB to RDF mapping language that allows to manually customize the mapping. The expert user has to know the RDB and the domain ontology to express the schema utilizing an existing target ontology. The W3C RDB2RDF Working Group proposed a set of core requirements for R2RML [11], including the exposition of many-to-many join tables as simple triples [10].

*Due to the representation of many-to-many join tables as simple RDF triples, we consider mapping mechanisms conform to the R2M2RL specifications.*

### 2.2 Differential privacy

DP proposes a robust mathematical framework for privacy protection [12]. An algorithm respect DP if observing its output does not permit to determine with strong confidence which of several neighboring dataset was used as input.

**Definition 1 ($\epsilon$-differential privacy).** *Given $\epsilon > 0$, a function $f : \mathcal{D} \to \mathcal{S}$ and a distance $\delta$ over $\mathcal{D}$, is $\epsilon$-differentially private if, for any couple of datasets $(D, D') \in \mathcal{D}^2$ such that $\delta(D, D') = 1$, and for any $S \subseteq \mathcal{S}$:*

$$Pr[f(D) = S] \leq e^{\epsilon} * Pr[f(D') = S]$$

*where probability $Pr$ is over the randomness of $f$.*

The parameter $\epsilon$ is also known as the budget of privacy, a smaller value indicating higher privacy. Two datasets at a distance one are said to be neighbors.

**DP for RDF** DP is immediately applicable to any space $\mathcal{D}$ given a proper distance $\delta$ or notion of neighborhood over $\mathcal{D}$. When considering graphs, two models prevail : *k-edge-DP* and *node-DP*.

In node-DP, neighboring graphs differ by a single node and all its incident edges, protecting each nodes and its incident edges. While the strongest of the two models, node-DP poses a particular challenge. Two neighboring graphs can differ by an arbitrary high number of edges, which may lead to high variations of outputs among a neighborhood and low utility DP mechanisms.

k-edge-DP [13] is a looser model that prevents an attacker from inferring the presence or absence of k-edges in the graph. Compared to node privacy, edge privacy is limited to protecting the relationships. In k-edge privacy, two graphs are adjacent if they differ by up to k edges. 1-edge privacy is simply called edge privacy and the most commonly employed in the literature.

Note that this two models do not consider potential attributes or labels of the graph. Reuben [14] proposed the adaptation of DP to edge-labeled directed graphs by defining sets of sensitive labels to which the protection is restricted. This can be transposed to most models, for example, k-typed-edge DP could be defined as the model where two adjacent graphs differ by up to k *sensitive* edges.

**DP for multi-relational databases** In the DP literature [4], a database is commonly a single, monolithic table of records (or tuples) that holds private data. Multi-relational databases, i.e., databases composed of many tables, are less popular. However, DP has also been investigated in this setting [15,16,17].

PINQ [15] and FLEX [16] consider a simple definition of neighboring databases, which does not consider FK constraints. According to their definition, neighboring databases possess the same set of relations and attributes and differ by exactly one tuple in one relation.

PrivateSQL system [17] introduces a richer notion of neighboring databases that considers constraints in the schema, in particular primary and FK constraints. Under this model, upon deletion of one tuple from one relation, many tuples in other relations have to be deleted because of the existence of FK constraints. PrivateSQL enables privacy to be designated at multiple resolutions. Their approach permits the data owner to designate which entities in the schema need privacy flexibly. The key idea is that one relation is specified to be the primary private relation. However, privacy protection extends to additional private relations, which are called secondary private relations. Those secondary are linked to the primary one via FKs. Under this DP policy, two database instances are considered neighbors when one can be obtained from the other by deleting a tuple t from the primary private relation and cascade deleting other tuples that depend on t through FKs. One requirement in their approach is that the schema needs to be acyclic. Based on this proposal, researchers began to consider FK constraints when defining neighboring databases [18,19].

*Due to the consideration of FK constraints, the familiarity of cascade deletion on which its neighbourhood concept rely, and the general adoption of the model presented in [17], we adopt this model as the starting point for RQ1 and aim at providing an equivalent model in RDF. The related formal distance definition will be restated in our model in Definition 8.*

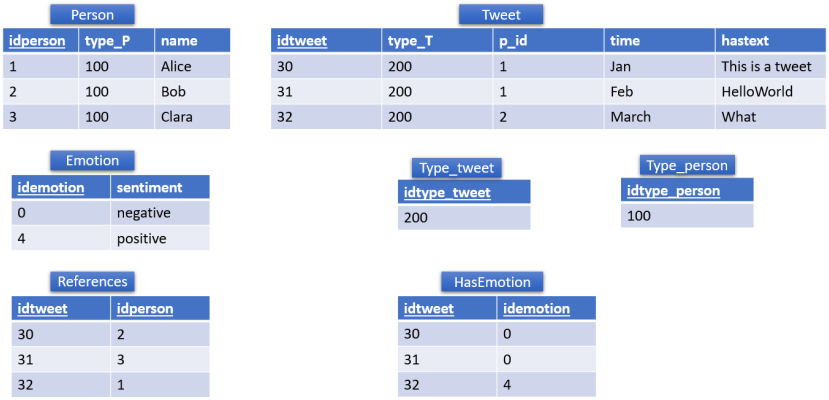
## 3 Setting: formalizing the concepts

This section introduces an illustrative example based on a Twitter dataset that will be used in the remainder of the paper. It then proposes the formalization of RDB, graphs representing RDF databases, and the mapping from RDB to RDF.

### 3.1 Illustrative example

**Illustrative dataset** In this paper, we use as an illustrative example a simple Twitter database, inspired by the Sentiment140 dataset composed of 1.6 million tweets [3]. An instance of the database is illustrated in Fig. 1 and the ER diagram is available online[4].

[3] https://www.kaggle.com/kazanova/sentiment140
[4] https://github.com/sarataki/mapping/blob/main/ER-Diagram.png



Person

| idperson | type_P | name |
|---|---|---|
| 1 | 100 | Alice |
| 2 | 100 | Bob |
| 3 | 100 | Clara |

Tweet

| idtweet | type_T | p_id | time | hastext |
|---|---|---|---|---|
| 30 | 200 | 1 | Jan | This is a tweet |
| 31 | 200 | 1 | Feb | HelloWorld |
| 32 | 200 | 2 | March | What |

Emotion

| idemotion | sentiment |
|---|---|
| 0 | negative |
| 4 | positive |

Type_tweet

| idtype_tweet |
|---|
| 200 |

Type_person

| idtype_person |
|---|
| 100 |

References

| idtweet | idperson |
|---|---|
| 30 | 2 |
| 31 | 3 |
| 32 | 1 |

HasEmotion

| idtweet | idemotion |
|---|---|
| 30 | 0 |
| 31 | 0 |
| 32 | 4 |

Fig. 1: A database instance of the Twitter schema

This example presents two many-to-many relationships: 1) between Person and Tweet, captured by the References table with FK referencing the id of a tweet and the id of the persons it references; 2) between Tweet and Emotion, captured by the HasEmotion table. The Tweet table possesses a FK "p_id" from table Person referencing the person that authored the tweet.

**Our mapping process** We select R2RML-F [20], an R2RML implementation available on Github[5]. The R2RML mapping process is done with R2RML-F whose engine takes as input the RDB, the R2RML mapping file that dictates the direction of relation, and the format of the output file to generate RDF data. We manually write R2RML mappings, which are tailored to our database schema. The R2RML mapping file[6] is available online as well as the output file[7].

Figure 2 shows a synopsis of the RDF database resulting from the R2RML mapping of the RDB illustrated in Fig. 1. As we can see, R2RML exposes many-to-many relationships as simple triples.

### 3.2 Relational Database

We use a conventional notion of schema for relational databases. A **table schema** is a set of attribute names. We assume these sets are disjoint and attributes do not have types or domains for this formalism. A **database schema** is a finite set of tables $\mathcal{T}$ and of constraints $\mathcal{C}$.

**Definition 2 (Database).** *A* ***database*** *following a certain database schema* $\mathcal{D} = (\mathcal{T}, \mathcal{C})$ *is a set* $D$ *of elements* $x$ *such that:*

[5] https://github.com/chrdebru/r2rml
[6] https://github.com/sarataki/mapping/tree/main/propR2RML/r2rml.ttl
[7] https://github.com/sarataki/mapping/tree/main/propR2RML/output.ttl

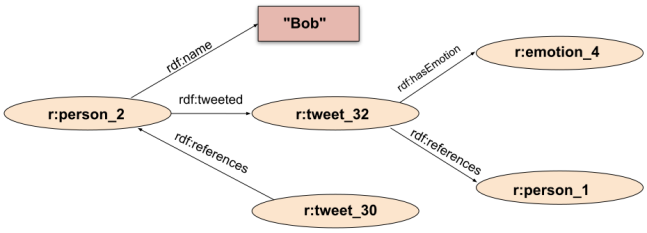



Fig. 2: Extract of the graph database resulting from the mapping of Fig. 1

- *$x$ belongs to exactly one table $T \in \mathcal{T}$, we note $x \in T$*
- *For each $s$ an attribute of $T$, we note $x.s$ the value of attribute $s$ for $x$. If it is undefined, we say $x.s = null$*

*A constraint $C \in \mathcal{C}$ can be of two forms:*

- *Each table $T$ has a primary key constraint $PK_T$, i.e. a nonempty subset $PK_T \subseteq A_T$ that guarantees for $x \in T$, for each $s \in PK_T$, $x.s$ is defined, and if $x \neq x'$ then there exists $s \in PK_T$ such that $x.s \neq x'.s$.*
- *There exist foreign key constraints $\phi$, i.e. a partial injective mapping $\phi$ from the attributes of a table $T_0$ to the attributes of a table $T_1$ such that the domain of $\phi$ is the primary key of $T_0$, $PK_{T_0}$, and for all $x' \in T_1$, there exists an element $x \in T_0$, called its $\phi$-antecedent, such that $x'.\phi(s) = x.s$. Such a key constraint is **from** $T_0$ **to** $T_1$.*

Furthermore, we identify a common type of tables in relational databases. A **relation table** is a table whose primary key contains all its attributes, and is composed of the disjoint union of the domain of two foreign keys. Other tables are called **entities**.

We say that a database schema $\mathcal{D} = (\mathcal{T}, \mathcal{C})$ is **ER-compliant** if for all tables $T_1 \in \mathcal{T}$, either for all key constraint from some $T_0$ to $T_1$, $dom(\phi) \cap PK_{T_1} = \emptyset$ (we say $T_1$ is an entity) or $T_1$ is a relation between two entities, and if all foreign keys start from entities.

Classically, many-to-many relations are stored in relation tables, whereas one-to-many relations are directly embedded in entities through foreign keys. In our example database, the relation "tweeted" is a one-to-many, as each tweet has only one author, and is thus stored as a foreign key in the Tweet table. However, the "references" relation is many-to-many, as a tweet can reference several people, and a person can be mentionned in several tweet. The table References stores this relation as pairs of foreign keys from Person and Tweet.

### 3.3 Graph Database and distances

A RDF dataset is represented as a graph:

**Definition 3 (Graph Database).** *A graph database is a tuple $(A, L, V, E)$ such that:*

- *$A$ is a potentially infinite set of attribute values*
- *$L$ is a potentially infinite set of edge labels*
- *$V$ is a finite set of vertices*
- *$E \subseteq V \times L \times (V \cup A)$ is a set of edges. In an edge $v, l, v'$ we call $v$ the subject, $l$ the predicate, $v'$ the object.*

In this definition, RDF triples are modeled as edges, either between two nodes or from a node to one of its attributes. Attributes here correspond to literals in RDF: they cannot be the subject of a relation and cannot appear isolated. We note that those attributes are not nodes themselves, and will not be counted as such in future distances. $L$ denotes all possible predicates and $A$ denotes the domain of definition of literals that may be object of a predicate.

In the figures, by convention, we represent nodes as yellow ovals and attributes as red rectangle. For example, in Fig. 2, the node "r:person_2" has an out-edge labeled "rdf:name" whose destination is an attribute "Bob". This represent an RDF triple whose object as URI "r:person_2", with predicate "rdf:name" and object the literal string "Bob". From the remainder of the graph, we see that the individual named Bob is the author of "r:tweet_32" which references "r:person_1", etc.

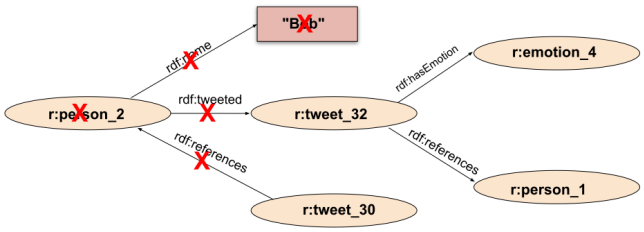



Fig. 3: Node deletion

Since we seek to transpose cascade deletion, which implies the suppression of entities rather than solely relations, we formalize node deletion as follows:

**Definition 4 (Node Deletion).** *Let $G = (A, L, V, E)$. We call $E_{rmv}$ the set of edges incident to $v$: $(v_0, l, v_1) \in E_{rmv}$ iff $v_0 = v$ or $v_1 = v$. The result of the node deletion of $v$ in $G$ is a graph $G' = (A, L, V \backslash \{v\}, E \backslash E_{rmv})$.*

Accordingly, the transposition of the node distance in this formalism is:

**Definition 5 (Node Distance).** *Let $G = (A, L, V, E)$ and $G = (A, L, V', E')$ be two graph databases. We say that $G$ and $G'$ are node-distance neighbors if $G'$ is the result of the deletion of a node $v \in V$ in $G$, or $G$ is the result of the deletion of a node $v \in V$ in $G'$.*
*The node distance is defined over graphs of same labels as the length of a shortest path connecting two databases neighbor by neighbor, if it exists.*

Figure 3 illustrates the deletion of node "r:person_2" from the graph pictured in Fig. 2. All the edges incident to it are deleted (those labeled "rdf:name", "rdf:tweeted", and "rdf:references"). While still formally in $A$, the attribute "Bob" does not appear in the graph anymore since the triple it was the object of has been suppressed. Hence, literal cannot appear while isolated and do not count as the suppression of a node toward the distance. The resulting graph, containing 4 nodes and two edges, is a node-neighbor of the original graph.

### 3.4 Encoding formalizing a mapping

An encoding, or mapping, from RDB to graphs modeling a RDF dataset is defined as follows:

**Definition 6 (ER encoding).** *Let $\mathcal{D} = (\mathcal{T}, \mathcal{C})$ be an ER-database schema. An encoding of relational databases following $\mathcal{D}$ into graphs is an injective function $f$ from the set of all relational databases following $\mathcal{D}$ into the set of graphs, that matches all ER-database $D$ following $\mathcal{D}$ a graph $f(D) = (A, L, V, E)$ such that:*

- *For each entity $T$, for each $x \in T$ in $D$, there is a node $x \in V$ in $f(D)$*
- *For each entity $T$, for each attribute $s$ not in the range of a foreign key, for each $x \in T$ with a defined value for $s$ $x.s = a$ in $D$, there exists a node $a \in A$ and an edge $(x, s, a) \in E$ in $f(D)$*
- *For each relation $T$ with the two foreign keys $\phi_A$ from entity $T_A$ and $\phi_B$ from entity $T_B$, one of these tables is the source (here $T_A$) and the other the object (here $T_B$) such that for each $x \in T$, $x_A \in T_A$ its antecedent by $\phi_A$, $x_B \in T_B$ its antecedent by $\phi_B$ in $D$, there exists an edge $(x_A, T, x_B) \in E$ in $f(D)$*
- *For each foreign key $\phi$ from an entity $T_0$ to an entity $T_1$, for each $x_1 \in T_1$ with a $\phi$-antecedent $x_0 \in T_0$ in $D$, there exists an edge $(x_A, \phi, x_B) \in E$ in $f(D)$*

We note that such encodings correspond to the W3C specification of R2RML for tables, one-to-many relations as foreign keys, and many-to-many relations. An ER encoding is an **isometry** w.r.t. a distance $d$ on relational databases and a distance $d'$ on graph databases iff for all $D, D'$ relational databases following $\mathcal{D}$, if $d(D, D')$ is defined then $d'(f(D), f(D'))$ is defined and equal to $d(D, D')$.

For our recurring example, one possible encoding of some entries of Fig. 1 is the graph of Fig. 2. The entities (e.g. person 2, tweet 32, emotion 4) are translated into nodes. However, relations such as References and foreign key relations such as HasTweeted become labellings in $L$ and are represented as edges, e.g. (r:tweet_32,rdf:references,r:person_1).

## 4 Distances and isometry for DP in RDB and RDF

We now focus on notions of neighboring databases that derive from cascade deletion. First, we present a generalized notion of cascade deletion. Then we

show the special case that corresponds to the transitive deletions introduced by Kotsogiannis et al. [17], and we analyze this notion on the RDB example, Twitter. We show the issue this definition carries through a translation in RDF, and propose an alternative definition, based on a more restricted deletion strategy, whose resulting distance is meaningful in both RDB and RDF worlds.

In general, a cascade deletion is the repercussion of the deletion of an element in a table to all other elements that depended on it in others. The characterization of such dependencies usually revolves around foreign keys, but may vary from a formalization to another. For this reason, we define here the cascade deletion as parameterized by its dependencies.

**Definition 7 ($\mathcal{C}'$ Cascade Deletion).** *Let $D$ be a database on a schema $\mathcal{D} = (\mathcal{T}, \mathcal{C})$, and $\mathcal{C}' \subseteq \mathcal{C}$ a set of foreign keys. Let $x$ be an element of $D$. The cascade deletion of $x$ alongside $\mathcal{C}'$ defines a set of* ***deleted elements*** *$L_{rm(x)}$ as the smallest set of lines such that:*

- *$x \in L_{rm(x)}$*
- *If $z$ is an element in $T_1$, such that there exists a foreign key $\phi \in \mathcal{C}'$ of $T_1$ on $T_0$, and the $\phi$-antecedent of $z$ is $y \in L_{rm(x)}$, then $z \in L_{rm(x)}$.*

*This is turn defines a set $A_{rm(x)}$ of* ***deleted attributes****, pairs $(y, s)$ such that:*

- *$y \in T_1$*
- *there exists a foreign key constraint $\phi \notin \mathcal{C}'$ from $T_0$ to $T_1$*
- *the antecedent of $y$ by $\phi$ is in $L_{rm(x)}$*

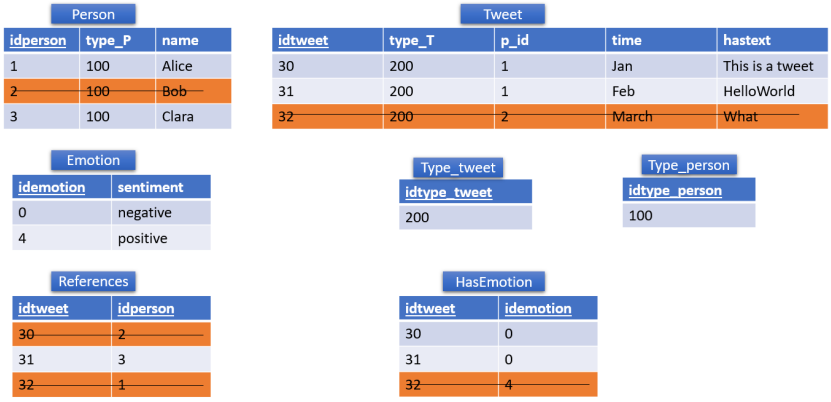
*The result of the cascade deletion of $x$ in $D$ is a database $D'$ of schema $\mathcal{D} = (\mathcal{T}, \mathcal{C})$ whose elements are all the elements of $D$ that are not in $L_{rm(x)}$ where for every $(y, s) \in A_{rm(x)}$, $y.s$ is set to null.*

**Definition 8 ($\mathcal{C}'$ Cascade Distance).** *Let $D, D'$ be two database of same schema, and $\mathcal{C}'$ a set of foreign keys. We say that $D$ and $D'$ are $\mathcal{C}'$ Cascade neighbors if $D'$ is the result of the cascade deletion of an element $x$ in $D$ alongside $\mathcal{C}'$, or $D$ is the result of the cascade deletion of an element $x$ in $D'$ alongside $\mathcal{C}'$.*
*The $\mathcal{C}'$ cascade distance is defined over databases of same schema as the length of a shortest path connecting two databases neighbor by neighbor, if it exists.*

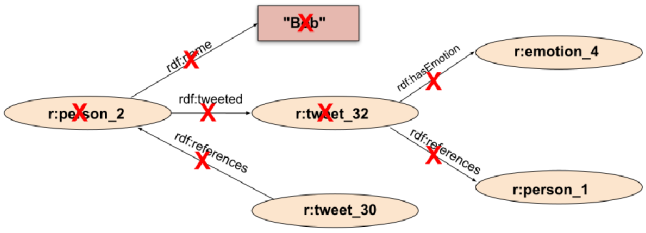
We note that this definition is "eager" in its deletion, which is to say elements are deleted as soon as one of their relevant antecedents is deleted. There exists another, "cautious" cascade deletion, where elements get deleted only if all their relevant antecedents are deleted. The definition of transitive deletions [17] uses this eager deletion strategy, and our own proposal circumvents the problem by limiting deletions in a way this distinction no longer matters.

### 4.1 Mapping the Transitive Deletion Distance

To model cascade deletions in a way that corresponds to [17], we consider that one can compute a join between tables starting from a primary entity $T$ and




(a) in RDB




(b) in RDF

Fig. 4: Transitive cascade deletion with Person as primary entity

following all outgoing foreign key constraints. To study the impact a deletion in the primary table would have on the join, we can delete every line of every table that would no longer occur in it. This corresponds to a transitive deletion alongside those foreign keys.

**Definition 9 ($T$ Transitive Cascade Deletion).** *Let $D$ be a database on a schema $\mathcal{D} = (\mathcal{T}, \mathcal{C})$, and $T \in \mathcal{T}$ a table. A $T$ transitive deletion in $D$ is the $\mathcal{C}$ cascade deletion of an element $x \in T$.*

*The $T$ transitive cascade distance is defined over databases of same schema as the length of a shortest path connecting two databases neighbor by neighbor, if it exists.*

As an example, the transitive deletion of Bob in the Person table, illustrated in Fig. 4, cascades to the Reference table as tweet 30 can no longer reference him, but also leads to the deletion of his tweet (tweet number 32) which in turns deletes two more lines in the database, one in References, one in HasEmotion.

*Limitations.* We now discuss the two limitations of this distance. First, the choice of a primary table is restricting. While the join approach and transitive

deletion as described in [17] necessitates picking a starting point, this has the undesirable side-effect of locking the privacy model towards certain protections and away from other. In RDF, it is possible to use privacy model protecting any node (DP with node distance) or even nodes from one or several tables exclusively (DP with type-node distance). This is not always possible in databases once a primary table is picked. For instance, in the given database, if we pick Person as the primary table, Fig. 4a shows the only possible way to delete tweet 32. It is then impossible to delete a specific tweet without deleting its author and all its other tweets. In turn, choosing Tweet as a primary table would make it impossible to protect a Person. In a privacy setting, this a restriction that (typed) node DP would not exhibit.

Furthermore, cascade deletion can have a greater or lesser impact based on the chosen starting table. To illustrate this point, in Fig 5, we show a cascade deletion from the Tweet primary relation. In the corresponding RDF graph, this leads to the deletion of a single node and all its adjacent edges, which is coherent with a node distance of 1. However, in Fig 4, we show a cascade deletion from the Person primary relation. In the corresponding RDF graph, this leads to the deletion of two nodes and their adjacent edges, which is coherent with a node distance of 2. While it is possible to define an equivalent distance in graphs and propose DP mechanisms accordingly, they would be at risk of having low utility. Indeed, the number of nodes affected by a single deletion being unbounded is a problem in a DP setting, as it aims to guarantee a protection between neighbors. Providing node-DP while maintaining acceptable utility can be challenging, and in the present case neighboring database would differ even more.

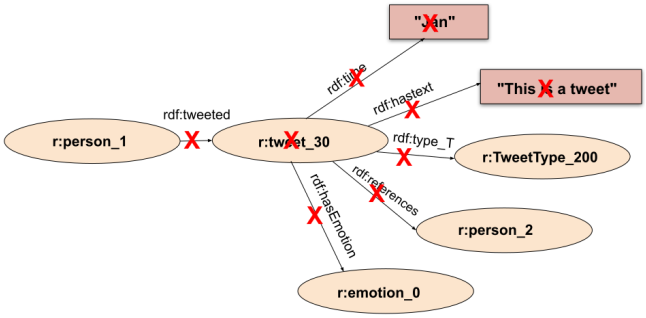



Fig. 5: Transitive cascade deletion of a tweet as shown on a graph

### 4.2 A new meaningful distance : Restrict Cascade Distance

To circumvent the issues highlighted above, we propose another instance of the cascade deletion: the **restrict cascade deletion**. The key idea is that the deletion of elements from an entity only propagates on the neighboring relations.

**Definition 10 (Restrict Cascade Deletion).** *Let $D$ be an ER-compliant database on a schema $\mathcal{D} = (\mathcal{T}, \mathcal{C})$, $T \in \mathcal{T}$ a table, and $x \in T$ be an element of $D$. The restrict cascade deletion of $x$ in $D$ is its $\mathcal{C}'$ cascade deletion, where $\mathcal{C}'$ is the set of foreign keys from entities to relations.*

**Definition 11 (Restrict Cascade Distance).** *Let $D, D'$ be two ER-compliant databases of same schema. We say that $D$ and $D'$ are Restrict Cascade neighbors if $D'$ is the result of the cascade deletion of an element $x$ in $D$, or $D$ is the result of the cascade deletion of an element $x$ in $D'$.*
*The Restrict cascade distance is defined over ER-compliant databases of same schema as the length of a shortest path connecting two databases neighbor by neighbor, if it exists.*

The restrict cascade deletion behaves similarly to the transitive distance except that in some cases it accepts a *null* as FK rather than deleting the concerned line. For example, the restrict cascade deletion is exactly equivalent to the transitive deletion in Fig. 5. In the case of Fig. 4, the FK p_id with value 2 is replaced by a *null*, the line being preserved and the process stopping rather than leading to the suppression of lines in the tables References and HasEmotion.

*Notably, this meaningful notion of distance in the relational world, is, no matter the starting table or the data we are trying to protect, always isometric to RDF node distance.*

**Theorem 1.** *All ER encodings are isometric w.r.t the restrict cascade deletion distance and the node distance*

The proof of this theorem is made by establishing the following lemma:

**Lemma 1.** *In an ER database, restrict cascade deleting an element of an entity is exactly deleting adjacent relations and erasing adjacent foreign keys*

*Proof (Lemma).* Restrict cascade deletion only propagates through foreign keys from an entity to a relation, hence the first part of the lemma: every element of a relation pointing towards the original element are deleted, and it does not propagate further. Erasure concerns foreign keys coming from deleted nodes. However, the only deleted nodes are the original and adjacent relations. Since in an ER database, foreign keys only come from entities, the foreign keys coming from deleted nodes all come from the original.

The proof of the theorem is a direct consequent:

*Proof (Theorem).* Let $D$ be a database, $x$ one of it's elements, $D'$ the database resulting from the cascade deletion of $x$ in $D$, and $f$ an ER encoding. $D$ and $D'$ are neighbors in the restrict cascade deletion distance, from Lemma 1. We will prove $f(D)$ and $f(D')$ are neighbors in the node deletion distance. The only difference between $D$ and $D'$ is:

- The deletion of $x$: this translates as the disappearance of the node $x$ from $V$ and the deletion of all its information, which translates as a deletion of all edges outgoing from $x$ between $f(D)$ and $f(D')$.
- The deletion of every adjacent relation element: this translates as the disappearance of some edges between $x$ and other elements of $V$ that corresponds to the encoding of relations between $f(D)$ and $f(D')$.
- The erasure of every foreign key coming from $x$: this translates as the disappearance of all remaining edges between $x$ and elements of $V$ that corresponds to the encoding of entity-to-entity foreign keys between $f(D)$ and $f(D')$.

As a summary, between $f(D)$ and $f(D')$, we have removed $x$ from $V$, and all edges incident to $x$ from $E$. We note that this is the exact definition of the node deletion of $x$ in $f(D)$, and conclude that $f(D)$ and $f(D')$ are neighbor in the node deletion distance.

Note that this argument goes both ways: any node deletion in $f(D)$ would result in a new graph which is the encoding of another relational database $D''$, which is identical to $D$ save for one restrict cascade deletion.

Since both the cascade deletion distance and the node deletion distance are defined as the shortest distance from neighbor to neighbor between two points, this preservation of neighborhood is sufficient to prove that $f$ is an isometry.

## 5 Conclusion

In this paper, we analyze the transposition to RDF through mapping of a popular DP model for multi-tables relational databases with FK constraints related to transitive deletion [17]. We showed that it sometimes translates to typed-node DP in RDF, a natural adaptation of the classical node DP model to typed graph. To ease the construction of RDF DP mechanisms while remaining explainable in the relational world, we tweak the original privacy model in a meaningful way so that it's translation is always equivalent to typed-node DP. Thus, we proposed the restrict deletion for relational databases, which captures privacy policies and FK constraints. Moreover, we proposed an implementation based on R2RML to illustrate our approach.

For future work, we plan to strengthen and implement relational-to-graph and graph-to-relational database mapping methods, by matching known and useful distances of RDB or RDF as well as neighborhood definitions which would make more sense in this context into corresponding notions in the other formalism. Furthermore, another interesting research direction is establishing a benchmark to compare the efficiency of different privacy methods through mapping. This would lead to a wider choice of comparable options for information stored as RDB or RDF while preserving important privacy guaranteeing properties.

## References

1. Franck Michel, Johan Montagnat, and Catherine Faron Zucker. *A survey of RDB to RDF translation approaches and tools.* PhD thesis, I3S, 2014.

2. Mohamed AG Hazber, Ruixuan Li, Bing Li, Yuqi Zhao, and Khaled MA Alalayah. A survey: Transformation for integrating relational database with semantic web. In *Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences*, pages 66–73, 2019.
3. Iovka Boneva, Slawomir Staworko, and Jose Lozano. Sherml: mapping relational data to rdf. 2019.
4. Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.
5. Joseph P Near, Xi He, et al. Differential privacy for databases. *Foundations and Trends® in Databases*, 11(2):109–225, 2021.
6. Marcelo Arenas, Alexandre Bertails, Eric Prud'hommeaux, Juan Sequeda, et al. A direct mapping of relational data to rdf. *W3C recommendation*, 27:1–11, 2012.
7. Das Souripriya, Sundara Seema, and Cyganiak Richard. R2rml: Rdb to rdf mapping language. *W3C Recommendation*, 27, 2012.
8. Luciano Frontino de Medeiros, Freddy Priyatna, and Oscar Corcho. Mirror: Automatic r2rml mapping generation from relational databases. In *Engineering the Web in the Big Data Era: 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings 15*, pages 326–343. Springer, 2015.
9. Tim Berners-Lee. Relational databases on the semantic web, 1998. *Via https://www. w3. org/DesignIssues/RDBRDF. html, last accessed January*, 2015.
10. Franck Michel, Johan Montagnat, and Catherine Faron Zucker. A survey of rdb to rdf translation approaches and tools. 2013.
11. Sören Auer, L Feigenbaum, D Miranker, A Fogarolli, and J Sequeda. Use cases and requirements for mapping relational databases to rdf. *W3c working draft*, 2010.
12. Cynthia Dwork. Differential privacy. In *Proceedings of the Automata, Languages and Programming, 33rd International Colloquium, ICALP*, 2006.
13. Michael Hay, Chao Li, Gerome Miklau, and David Jensen. Accurate estimation of the degree distribution of private networks. In *2009 Ninth IEEE International Conference on Data Mining*, pages 169–178. IEEE, 2009.
14. Jenni Reuben. Towards a differential privacy theory for edge-labeled directed graphs. *SICHERHEIT 2018*, 2018.
15. Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30, 2009.
16. Noah Johnson, Joseph P Near, and Dawn Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.
17. Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment*, 12(11):1371–1384, 2019.
18. Yuchao Tao, Xi He, Ashwin Machanavajjhala, and Sudeepa Roy. Computing local sensitivities of counting queries with joins. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 479–494, 2020.
19. Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. R2t: Instance-optimal truncation for differentially private query evaluation with foreign keys. In *Proceedings of the International Conference on Management of Data*, 2022.
20. Christophe Debruyne and Declan O'Sullivan. R2rml-f: Towards sharing and executing domain logic in r2rml mappings. *LDOW@ WWW*, 1593, 2016.