INSA Centre Val de Loire

ÉCOLE DOCTORALE Mathématiques, Informatique, Physique Théorique et Ingénierie des Systèmes

Laboratoire d'Informatique Fondamentale d'Orléans

THÈSE présentée par :

Sara TAKI

soutenue le : **21 Décembre 2023**
pour obtenir le grade de : **Docteur de l' INSA Centre Val de Loire**

Discipline/ Spécialité : Informatique

Linked Data Sanitization with Differential Privacy

Titre: Assainissement de données liées par Confidentialité Différentielle

Mots clés: Anonymisation; Confidentialité différentielle; RDF; SPARQL; Projection de graphe; Bases de données relationnelles; langage de mappage; RDB2RDF Mappage; Réécriture de Graphes

Résumé: Cette thèse étudie le problème de la protection de la vie privée dans le Linked Open Data (ou « LOD », en français « web des données ouvertes » ou encore « données liées ouvertes »). Ce travail se situe à l'intersection d'une longue série de travaux sur la confidentialité des données et le LOD. Notre objectif est d'étudier l'impact des aspects sémantiques sur la publication des données et sur les fuites éventuelles d'information. Nous considérons RDF comme le format de représentation du LOD et la confidentialité différentielle (DP) comme le principal critère de protection de la vie privée. La DP a été initialement conçue pour définir la confidentialité dans le domaine des bases de données relationnelle. Elle est basée sur une quantification de la difficulté pour un attaquant d'identifier, en observant le résultat d'un algorithme, quelle base de données parmis un voisinage a été utilisée pour le produire.

Les objectifs de cette thèse sont au nombre de quatre: O_1) améliorer la protection des données LOD. En particulier, proposer une approache permettant de construire des méchanismes DP *utilisables* sur RDF ; O_2) étudier comment les définitions des voisinages sur les bases de données relationnelles en présence de contraintes de clés étrangères (FK) peuvent être traduites en RDF : O_3) proposer de nouvelles définitions de voisinages sur des bases de données relationnelles équivalente à des notions existantes de voisinage sur les graphes (avec une sémantique précise) et O_4) proposer un formalisme facilitant la conception et l'implémentation de mécanismes d'anonymisation de données RDF.

Concernant O_1, nous proposons une nouvelle approche basée sur la projection de graphes pour adapter le concept de DP à RDF. Pour O_2, nous déterminons le modèle de protection qui correspond à la traduction de modèles déjà existants pour des bases de données relationnelles sous contraintes FK. Pour O_3, nous introduisons le concept de `restrict deletion neighborhood` (voisinage d'effacement limité) équivalent en voisinage de type "typed-node" (noeud typé). Nous proposons également une relaxation de la définition permettant de traduite les voisinages "typed-outedge" (arc sortant typé). Pour O_4, nous proposons un langage de transformation de graphes basé sur le concept de réécriture de graphes, qui sert de fondation pour construire divers mécanismes d'anonymisation sur des graphes attribués.

L'ensemble de nos contributions théoriques ont été implémentées par des prototypes "preuve de concept" et ont été évalués sur des jeux de données réels, afin de montrer l'applicabilité de nos travaux à des cas d'usage réels.

Title: Linked Data Sanitization with Differential Privacy

Keywords: Anonymization; Differential Privacy; RDF; SPARQL; Graph projection; Relational databases; Mapping language; RDB2RDF mapping; Graph Rewriting

Abstract: This thesis studies the problem of privacy in linked open data (LOD). This work is at the intersection of long lines of work on data privacy and linked open data. Our goal is to study how the presence of semantics impacts the publication of data and possible data leaks. We consider RDF as the format to represent LOD and Differential Privacy (DP) as the main privacy concept. DP was initially conceived to define privacy in the relational database (RDB) domain and is based on a quantification of the difficulty for an attacker observing an output to identify which database among a neighborhood is used to produce it.

The objective of this thesis is four-fold: O_1) to improve the privacy of LOD. In particular, to propose an approach to construct usable DP-mechanisms on RDF; O_2) to study how neighborhood definitions over RDB in the presence of foreign key (FK) constraints translate to RDF; O_3) to propose new neighborhood definitions over relational database translating into existing graph concepts to ease the design of DP mechanisms; and O_4) to support the implementation of sanitization mechanisms for RDF graphs with a rigorous formal foundation.

For O_1, we propose a novel approach based on graph projection to adapt DP to RDF. For O_2, we determine the privacy model resulting from the translation of popular privacy model over RDB with FK constraints to RDF. For O_3, we propose the `restrict deletion neighborhood` over RDB with FK constraints whose translation to the RDF graph world is equivalent to typed-node neighborhood. Moreover, we propose a looser definition translating to typed-outedge neighborhood. For O_4, we propose a graph transformation language based on graph rewriting to serve as a basis for constructing various sanitization mechanisms on attributed graphs.

We support all our theoretical contributions with proof-of-concept prototypes that implement our proposals and are evaluated on real datasets to show the applicability of our work.

Acknowledgment

To be added later

Contents

List of Figures

List of Tables

1 - Introduction

Contents

This Ph.D. thesis studies research issues at the intersection of two existing research fields: the *Semantic Web* and *Privacy*. Historically, these fields are unrelated. In this first chapter, we give an overview of these fields, explain the motivation and research challenges of our work, and state the contributions that will be presented in this manuscript.

1.1 - Introduction

1.1.1 - A short historical overview of the Semantic Web

The first concept we are concerned with in this thesis is the World Wide Web (WWW). It is potentially one of history's primary leading technological achievements. It started at CERN, the European Particle Physics laboratory in Geneva, Switzerland, when the British scientist Tim Berners-Lee proposed its first architecture in 1989 [3]. Initially, it was developed and envisioned to satisfy the need for automated information-sharing between scientists, institutions, and collaborators of a joint project across the world.

Tim Berners-Lee proposed the Semantic Web in 2001 [4]. The term semantic indicates meaning or comprehension. Berners-Lee's vision is to achieve a realization of a Web where the semantics of information play a significant and crucial role. Web 3.0 is the third generation of the World Wide Web and was introduced in 2006 [5, 6] as the Semantic Web.

According to the World Wide Web Consortium (W3C), "The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries" [7].

The Semantic Web is an extension of the classical Web [8]. Semantic Web can also be defined as a web of data [6]. The central point is to make the Web content in formats that can more easily and precisely be processed by machines compared to the initial "Web of Documents" created to be human-friendly, where HTML documents are the main objects and links are established between documents or their parts [6]. The main objective of the Semantic Web is to permit users to simply and easily share, search and find information. The primary layer for representing data on the Semantic Web is the **Resource Description Framework (RDF)** [9].

1.1.2 - RDF, a W3C standard for the Semantic Web

RDF is a standard framework data model for describing resources. Resources have the same meaning as "entity." A resource can be anything with identity: persons, authors, publishers, websites, books, documents, places, stores, hotels, and rooms. An RDF statement, also referred to as an RDF triple, is composed of three ordered components: **subject**, **predicate**, and **object**.

Subject represents the resource being described

Predicate represents the relationship between the subject and object. It indicates a property

Object represents the value of the property. Values can either be resources or literals

A set of RDF triples is named an RDF graph.

1.1.3 - Linked Open Data (LOD)

In 2006, Berners-Lee outlined the principles of Linked Data [10] to publish and connect data on the Web. The linked Data principles give guidelines to utilize fundamental standardized Web technologies to establish links to connect data from diverse sources into a single global data space [11]. By following these principles, data providers can add their data to a unified data space.

Linked Open Data (LOD) refers to Linked Data made available under an open license. Berners-Lee proposed five stars rating system for LOD [10]. It is cumulative in the sense that every star level includes the previous ones. Paraphrased from Berners-Lee:

★ The data is available on the Web, in any format, under an open licence.

★★ The data must be structured and available in a machine-readable format (e.g., XML).

★★★ Non-proprietary format (e.g., CSV instead of Microsoft Excel).

★★★★ Make use of W3C open semantic web standards (e.g., data modeling using RDF format and querying using SPARQL).

★★★★★ Link data to other data sources.

The Linking Open Data project [11] is the best evident illustration of the adoption and application of the Linked Data principles. The project's primary objective was to bootstrap the Web of Data by specifying the openly licensed existing data sets, transforming them into RDF format based on Linked Data principles, and then publishing them on the Web.

The Linked Open Data Cloud (or "LOD cloud" for short) has emerged as a consequence of this project. Figure 1.1 displays the linking Open Data cloud diagram, presenting a broad view of the interconnected data sets as of March 2009[1]. Every colored node denotes a distinct data set, and the arcs signify the presence of links between items in the two linked data sets. The thickness of arcs is strongly proportional to the number of links between two data sets. If the arc is bidirectional, every data set has outward links to the other. LOD cloud continues to grow significantly. The most recent version of the diagram as of September 2023 is depicted in Fig. 1.2. It contains 1314 datasets with 16308 links[2].

1.1.4 - Privacy

The second concept we are concerned with in this manuscript is *privacy*, and more specifically *privacy preserving data publishing* (PPDP). Privacy issues have long existed, but modern privacy in computer science appeared with experimental examples of privacy breaches with the publication of databases containing personal information at the turn of the XXI^{s}t century. Since

[1]https://lod-cloud.net/
[2]https://lod-cloud.net/

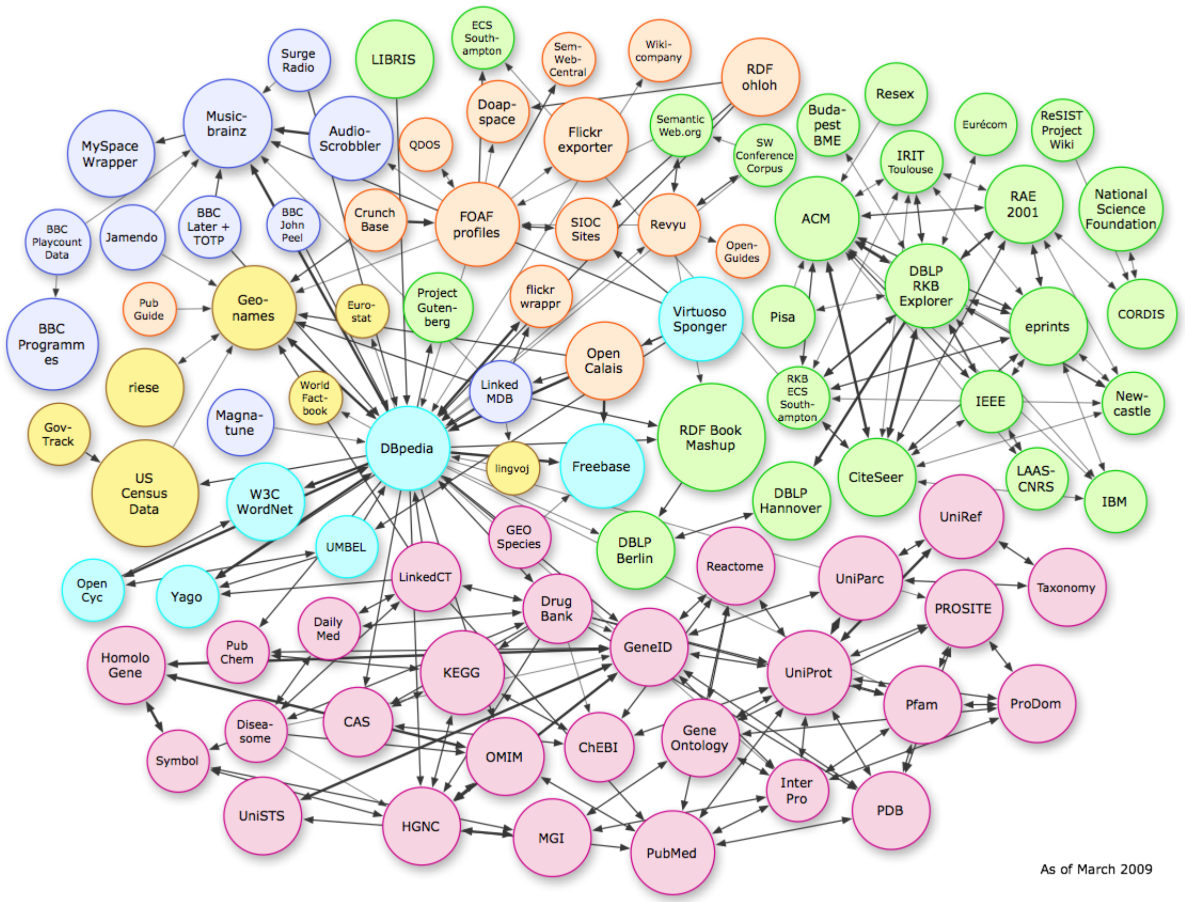


Figure 1.1: The state of Linked Open Data cloud as of March 2009. From lod-cloud.net

then, numerous privacy models have been put forth in the literature to address privacy concerns. The primary objective is to ensure that data subjected to such model does not disclose private information about any individual. The main challenge is to find mechanisms that ensure privacy while preserving the utility of the data. Ideally, we would like to optimize the trade-off between privacy and utility (Pareto optimality).

Two widely recognized privacy methods are k-anonymity [1, 12] and differential privacy (DP) [13, 14]. DP is a formal definition of privacy that permits quantifying the privacy-utility trade-off. Instead of being a privacy property of the released database (such as in k-anonymity and its variants), it is rather a definition that should be respected by a randomized algorithm (referred to as an algorithmic notion of privacy).

Informally, an algorithm is differentially private if observing its output does not permit to determine with strong confidence which of several 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.

DP has emerged as the flagship of data privacy definitions in recent years due to its desir-

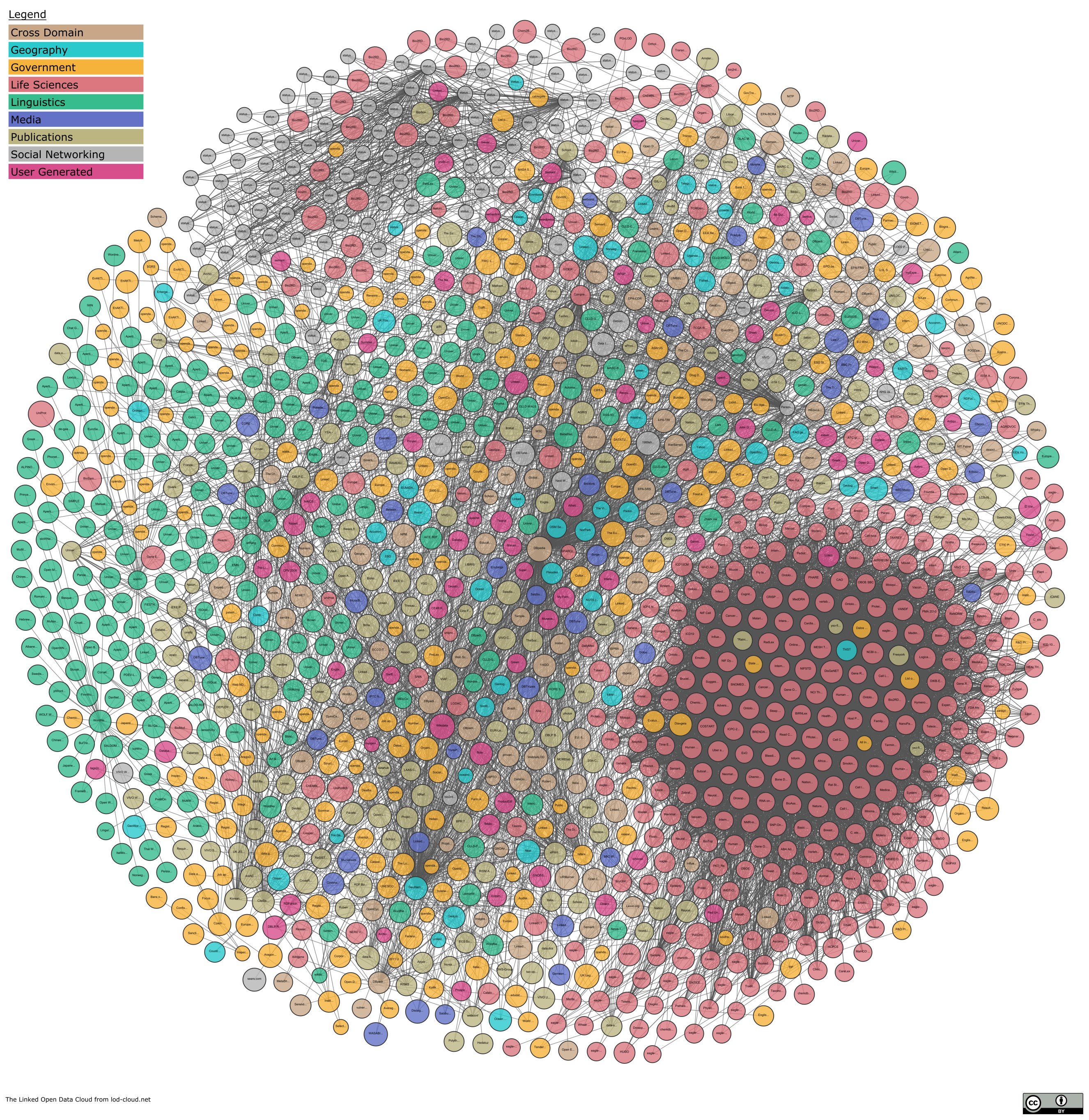


Figure 1.2: The state of Linked Open Data cloud as of September 2023. From lod cloud.net

able formal privacy guarantees. It has received growing attention from organizations. Numerous algorithms and statistical methods were altered to meet DP requirements and used by organizations such as Google [15, 16], Apple [17, 18], Microsoft [19], US Census Bureau [20–22], LinkedIn [23, 24], Facebook [25] , Uber [26–28], etc.

Over the years, DP has grown with two different models: the "central curator" model and the "local" model. These two models essentially provide different guarantees but usually differ on whether the data collector is centralized or decentralized, which can sometimes be seen as the central curator being trusted or untrusted.

1. In the **trusted curator** or **centralized model** for DP, also called the **standard model**, a trusted data curator collects data on individuals and applies a mechanism to achieve DP. Typically, this consists in adding an amount of noise to perturb the query results and then releasing the noisy results.
2. In the **Local model** for DP, there is typically no trusted curator. Instead, each individual applies a mechanism to their data 'locally' before sending it to the untrusted data curator. Afterward, the data curator chooses method of releasing information concerning the data. Consequently, the data curator is no longer required to be trusted. Note however that it is possible to apply the local model in a centralized setting with a trusted curator.

In this thesis, we used DP, particularly the centralized model. Hence, we introduce the centralized model of DP in Chapter 4, then briefly present the local model.

1.2 - Motivation and Research Challenges

Privacy in linked open data sources is becoming an issue, and several recent works have been proposed on the topic [29–31]. Indeed, directly publishing –or allowing direct querying of– graph data may result in the disclosure of sensitive information and therefore to privacy violations. The main motivation for this work is thus to improve the privacy of linked open data. Compared to existing work, which we discuss extensively in Section 4.3.3, our originality is to adopt an approach based on *differential privacy*.

Such an approach poses several research challenges :

RC1 What privacy models should be used to provide DP guarantees on RDF graphs?

RC2 Is it possible to link these models to the models such as those developed for relational databases?

RC3 Is it possible to assist users who want to develop algorithms to sanitize RDF databases?

RC4 Is it possible to implement the solutions proposed to solve the previous research questions?

In order to tackle $RC1$, we must first revisit the definition of neighborhoods in the original RDF context. Indeed, classical definitions of neighborhoods are not adapted to RDF data. We must also test if the solution is usable and does not lead to needing to inject too much noise. This question is studied in our first contribution *using projection to improve differential privacy on RDF graphs*.

Classical DP models for relational databases have been extensively studied. Their extension to multi-relational databases is less popular. $RC2$ leverages this field of work to try to study how privacy guarantees on multi-relational databases with foreign key constraints (FK) can be transposed to RDF databases. This question is studied in our second contribution, *mapping relational databases to RDF, and its impact on privacy*.

Our approach to study $RC3$ is to investigate existing graph transformation formalisms, namely *algebraic graph rewriting*. This framework offers a declarative way of defining transformations on graphs, which we can apply to anonymization algorithms.

$RC4$ is accounted for since we have the objective of proposing *useable* solutions. Thus, each of our contributions is supported by a proof-of-concept prototype implementing our proposals and tested on a real dataset (the Twitter `sentiment140` dataset[3]).

1.3 - Contributions

The main contributions of this thesis are divided into three parts summarized in the following:

1.3.1 - Using projection to improve Differential Privacy on RDF graphs

The first main contribution is a new approach based on graph projection to adapt DP to edge-labeled directed graphs, i.e., RDF graphs, in a usable and useful way. We consider three different privacy definitions: node privacy, outedge privacy and typed-outedge privacy. The main idea behind our approach is to use graph projection within the DP mechanisms to reduce the sensitivity of queries. Informally, this allows to reduce the amplitude of their results' variation over adjacent databases and ultimately reduce the randomness of DP mechanism. For projections to be adequate w.r.t. the privacy definitions and to minimize the information loss they imply, we propose three edge-addition based graph projection methods. They transform the original RDF graph into a graph respecting one of the following constraints: bounded degree, bounded out-degree, and bounded typed-out-degree. We evaluate our contribution analytically and experimentally w.r.t. a real Twitter use-case, showing a significant improvement over a naive approach without projection.

This chapter is based on the published papers: "Using projection to improve differential privacy on RDF graphs" [32] and "It's too noisy in here: using projection to improve Differential Privacy on RDF graphs" [33].

1.3.2 - Mapping relational databases to RDF and its impact on privacy

For the second main contribution, we consider the classical case of cascade deletion in a relational database, where a database which is obtained by deleting one tuple and cascading the others linked by FK constraints. We map both instances to an RDF database and show that it is sometimes similar to an existing DP graph privacy model. Consequently, we tweak this model in the relational world and propose a new model called restrict deletion neighborhood. We show

[3]https://www.kaggle.com/kazanova/sentiment140

that it is always similar to the aforementioned existing DP graph privacy model, facilitating the design and implementation of DP mechanisms.

For the third main contribution, we propose to study how we can translate the typed-outedge privacy model to the relational database setting. We propose a neighborhood definition for a relational database whose translation in the RDF graph world is equivalent to typed-outedge neighborhood.

This chapter is under preparation for a submission.

1.3.3 - Graph Rewriting Primitives for Semantic Graph Databases Sanitization

For the fourth contribution, we propose a graph transformation language to serve as a basis for the construction of various sanitization mechanisms. This language relies on a set of elementary transformation operators formalized using an algebraic graph rewriting approach. Our language takes into account semantic and support the equivalent of WHERE and EXCEPT clauses.

As a proof of concept, we use these operators to implement two mechanisms from the literature, one generic (Local Differential Privacy) and one specifically introduced for semantic graph databases (sensitive attribute masking through anatomization).

We propose an open-sourced tool implementing the elementary operators and the privacy mechanisms we derive from them relying on the Attributed Graph Grammar System (AGG) and its java API, providing a concrete tool implementing formal graph rewriting mechanisms to sanitize semantic graph databases. We present experimental results on this implementation regarding both proposed schemes and discuss its efficiency and scalability.

Note that the work presented in this chapter is collaborative work. My main contributions are on the use of anatomization to achieve anonymization.

The content of this chapter is under I review for publication in the journal Computer Science and Information Systems. It extends the results presented in [34].

1.4 - Thesis Outline

The rest of this manuscript is organized as follows:

The background concepts and existing works necessary to understand the scientific foundations of this thesis at the intersection of Semantic Web and Privacy are presented in Chapter 2 and 3, respectively. Chapter 4 presents the state of the art about various aspects and elements connected and interrelated within the scope of this thesis. Chapter 5 presents the first contribution of this manuscript. Namely, the use of projection to improve DP on RDF graphs. Chapter 6 presents the second and third contributions, around privacy models similarities in relational databases and RDF graphs and the impact of translation from one world to the other. Chapter 7 proposes a graph transformation language to serve as a basis for the construction of different sanitization mechanisms. The last Chapter gives a general conclusion of this work and some perspectives.

1.5 - Publications and Workshops

1. Sara Taki, Cedric Eichler, and Benjamin Nguyen. "Using projection to improve Differential Privacy on RDF graphs." In Actes de la conférence BDA 2021, p. 72. 2021. [32]
2. Sara Taki, Cédric Eichler, and Benjamin Nguyen. "Privacy over RDF datasets." In Actes de la conférence BDA 2021, p. 93. 2021. [35]
3. Sara Taki, Cédric Eichler, and Benjamin Nguyen. "It's Too Noisy in Here: Using Projection to Improve Differential Privacy on RDF Graphs." In European Conference on Advances in Databases and Information Systems (ADBIS), pp. 212-221. Cham: Springer International Publishing, 2022. [33]
4. Sara Taki, Cédric Eichler, and Benjamin Nguyen. "Using projection to improve differential privacy on RDF graphs", In Atelier sur la Protection de la Vie Privée (APVP 2022), Châtenay-sur-Seine, France, 2022. [36]

2 - Background: Semantic Web

Contents

2.1 - Introduction

We present in this Chapter the background concepts and existing works necessary to understand the scientific foundations of this thesis related to the Semantic Web.

We start by providing a brief history of the World Wide Web before introducing the Semantic Web and its most important building blocks: RDF, its standard data model; the dedicated query languages, in particular, SPARQL; the ontologies and vocabularies complimenting data to support inference, such as RDFS and OWL.

Since data is still mostly stored in relational databases, their transformation into RDF is a key growth factor for the Semantic Web. We, therefore, present standard ways of translating relational databases to RDF.

2.2 - World Wide Web (WWW)

Tim Berners-Lee proposed the first architecture of the Web [3]. The proposal established the main concepts and terminologies associated with the Web, in particular: Universal Resource Identifiers (URIs), Hypertext Markup Language (HTML), and Hypertext Transfer Protocol (HTTP). URIs are defined as strings to represent the addresses of objects (e.g., documents, images) on the Web. HTML is a markup language and the basic block for building the Web. It was defined as the standard basic language of exchange for hypertext. HTTP is a network and internet protocol governing information transfer between a server and a client. It does not only transfer HTML documents, but is employed to get documents in wide, unbounded range of formats.

After a period of activity at CERN, in April 1993, CERN made the source code and the Web protocol available on a royalty free basis[1]. This is to enable its widespread distribution. It has permitted its broad use and facilitated the Web's growth.

Since its introduction, the World Wide Web has undergone various revolutions [37] and has become the most significant information platform across the world. Web X.Y is the common phrase utilized to point to the various Web phased, known as Web 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0 [37] and Web 5.0 [5, 38]. The phase numbers indicate the Web's development and its progress from its inception to now.

Web 1.0 was referred to as the first web generation, known as static, read-only Web. Web 2.0, the second generation of the Web, was introduced in 2004, and Web 2.0 is known as 'the social web'. In 2006, Web 3.0 was introduced. It is known as the 'Semantic Web.' The expressions utilized to describe the fourth generation of the Web, Web 4.0, are: 'Ubiquitous web,' 'symbiotic web,' and 'Ultra-Intelligent Electronic Agent' [38]. The fifth generation of the Web is known as the 'sensory-emotion Web,' the latest version. It aims to create computers that can engage and interact with people to produce the emotional Web.

[1]https://home.web.cern.ch/science/computing/birth-web

2.3 - Today's Web

The modern Web benefits us in countless ways in our daily routines. It has revolutionized a wide range of activities we are engaged in. It has changed dramatically the way people communicate with each other. The invention that connected the world made it easier for people to share, get information, connect, and communicate. Even though the success of the Web is unquestionable, it is far from perfect as it has essential limitations.

One limitation of the Web is that web users have to speculate the search terms or keywords they should use to fulfill their information requirement [4] and then assess and analyze the search results that are returned by the search engine.

Another significant limitation of the Web emerges when the requested information is dispersed across multiple web pages and not present on a single one [39]. This requires human intervention to manually collect all the relevant information from diverse sources. Because the human ability and capacity to process vast amounts of information is relatively slow and imprecise, they are often only capable of taking into account a small fraction of the possibly pertinent information out of the huge amount of relevant information when seeking to tackle more complex tasks.

On the other side of the coin, machines are significantly more effective at processing vast amounts of information [39]. Nevertheless, a common algorithm having the potential to browse the Web and directly link the points and data to tackle countless complicated tasks is still far beyond the capabilities of existing technology.

One approach is to use machine learning to build such algorithms. Despite the enormous advances in machine learning nowadays, the task seems too ambitious as it suggests a common form of Artificial Intelligence with no prior equivalent.

The primary problem is that the Web was initially designed for humans to share documents. It was developed for humans to read and understand. On the contrary, machines are mainly utilized to display these documents. However, machines have limitations in processing the semantics.

An alternative approach is to rethink the Web itself. Instead of using Artificial Intelligence to tackle complicated tasks on the Web and imitate human-level understanding of the initially built-in human-readable Web, the Web content may be represented in a more machine-understandable form. This initiative to revolutionize the Web is known as the Semantic Web initiative.

2.4 - Semantic Web

2.4.1 - RDF

RDF [9] is a standard way to model semantic (or linked) data. An RDF data set is a set of triples (subject-predicate-object) which form a labeled directed graph. In an RDF graph, there exists three types of nodes: **IRIs**, **literals**, and **blank nodes**.

IRI (Internationalized Resource Identifier) are viewed as string identifiers for resources. They are expressed in compliance with [40]. IRIs form a global naming convention suitable with the Web, and this is its primary significant aspect. If the same IRI is utilized across several RDF datasets, it is evident that these datasets are referring to the same resource.

IRIs extend **URIs (Uniform Resource Identifiers)** [9], the latter are restricted to a subset of ASCII characters, while the former allow a broader range of Unicode characters. On the other side of the coin, **URIs (Uniform Resource Identifiers)** generalize **URLs (Uniform Resource Locators)**: while URLs refer to the identity and location of a resource on the Web, URIs solely need the identity of a resource [39]. In summary, URLs are a subset of IRIs/URIs.

Literals have a lexical form being a Unicode string and are categorized as either plain or typed. A plain literal is a lexical form that may or may not have an associated language tag, expressed as ("hello" or "hello"@en) while a typed literal is a lexical form with a data type URI, expressed as ("hello" ^^xsd:string). RDF utilizes the XML Schema Definition (XSD) specification to delineate a range of data types and typed literals commonly employ these data types. A data type specifies the set of possible values like strings, numbers, dates, and times. RDF defines two other data types, rdf:XMLLiteral and rdf:HTML.

Blank nodes (anonymous nodes) , rather than identifying a particular resource, represent the existence of a resource. Blank nodes can be viewed as placeholders for resources not referenced by an IRI or a literal [39]. Blank nodes act as variables within a local scope. They are limited to their local original scope, such as an RDF document, and hence can't be referenced outside their local scope. Therefore, their label is only meaningful within their local scope.

IRIs, **literals**, and **blank nodes** are generally referred to as RDF terms. There are some restrictions on where different types of RDF terms can be positioned within a triple:

Subject permits only an IRI or a blank node

Predicate permits only an IRI

Object permits an IRI, a literal or a blank node

As an example, we present a simple RDF graph composed of triples contained in Tab. 2.1 and graphically shown in Fig. 2.1. IRIs from Tab. 2.1 are abbreviated with prefixes in the figure; j.0: replaces http://rdfanon.org/types# and rdf: replaces http://www.w3.org/1999/02/22-rdf-syntax-ns#.

RDF serialization [41] refers to the process of representing the RDF data in a machine-readable format. Principally, the primary distinction among RDF formats lies in the concrete syntax employed for serializing RDF triples.

Table 2.1: Simple RDF graph instance

Subject	Predicate	Object
http://rdfanon.org/types#SGroup	http://rdfanon.org/types#name	"SGroup"
http://rdfanon.org/types#SGroup	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://rdfanon.org/types#person
http://rdfanon.org/types#SGroup	http://rdfanon.org/types#tweeted	http://rdfanon.org/types#19
http://rdfanon.org/types#19	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://rdfanon.org/types#tweet
http://rdfanon.org/types#19	http://rdfanon.org/types#text	"Its freezing "
http://rdfanon.org/types#19	http://rdfanon.org/types#emotion	"4"
http://rdfanon.org/types#19	http://rdfanon.org/types#timestamp	"Sat May 30 00:56:54 PDT 2009"
http://rdfanon.org/types#SGroup	http://rdfanon.org/types#tweeted	http://rdfanon.org/types#20
http://rdfanon.org/types#20	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://rdfanon.org/types#tweet
http://rdfanon.org/types#20	http://rdfanon.org/types#text	" Stuck in traffic"
http://rdfanon.org/types#20	http://rdfanon.org/types#emotion	"0"
http://rdfanon.org/types#20	http://rdfanon.org/types#timestamp	“Sun Jun 07 00:03:06 PDT 2009"

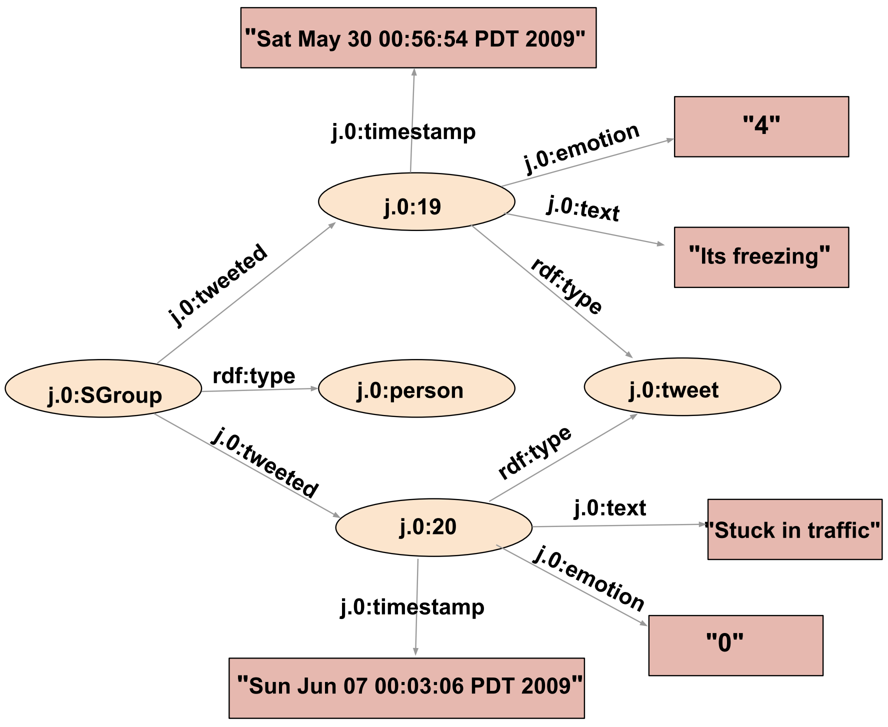


Figure 2.1: Visualization of the RDF graph from Table 2.1

The first standard syntax for RDF is the RDF/XML, which is XML-based syntax [42]. It serializes RDF into XML. The W3C defines concrete syntaxes for RDF, such as N-Triples [43], Turtle [44], RDFa, JSON-LD [45], and TriG [9]. N-Triples is a simple syntax where each line in the syntax represents a single RDF triple. It is plain to parse. Turtle, the Terse RDF Triple Language, gives a way to express RDF graphs in a concise textual form. Due to its compactness and readability, it is regarded as the most human-friendly RDF syntax [39]. Turtle and N-Triples are inspired by Notation3 (N3) [46], a logic language that expresses a superset of

RDF. Interestingly, each valid N-Triples file can also be considered as a Turtle file. Nonetheless, Turtle goes beyond N-Triples by providing a diversity of convenient abbreviations and shortcuts that enable a more compact serialization of RDF. RDFa, the RDF in Attributes, permits the direct embedding of RDF data into XML-based documents. Finally, as its name implies, JSON-LD is a JSON-based format for Linked Data (and also a W3C recommendation).

2.4.2 - SPARQL

Different query languages have been suggested throughout the years, such as RQL [47] and SeRQL [48]. **SPARQL (SPARQL Protocol And RDF Query Language)** [49] is the W3C Recommendation query language for RDF. It was released as a recommendation in 2008, with an extended SPARQL 1.1 version in 2013 that contains additional features. Hence, SPARQL 1.1 is the present version in practice, and this chapter reflects the standard SPARQL 1.1. SPARQL enables the expression of queries over various data sources regardless of whether the data is originally stored in RDF format or perceived as RDF through the use of middleware.

SPARQL draws inspiration from the Structured Query Language (SQL) for relational databases in terms of syntax and semantics, resulting in many shared characteristics between the two [39].

Our thesis heavily focuses on the manipulation of RDF data, and many of these tasks are performed in the experiments through the use of SPARQL. We thus spend some time explaining the functionalities of SPARQL.

2.4.2.1 - Graph Patterns

A basic concept in the SPARQL query language is the triple pattern. A triple pattern in SPARQL is similar to an RDF triple but permits the presence of variables in any position, i.e., in the subject, predicate, or object position, instead of specific RDF terms (such as **IRIs**, **blank nodes**, or **literals**). For example "⟨*http://rdfanon.org/types#SGroup*⟩ ⟨*http://rdfanon.org/types#tweeted*⟩ *?tweet*" is a valid triple pattern where *?tweet* is a variable.

A basic graph pattern (BGP) is a set of triple patterns composing a SPARQL graph pattern, similar to a set of RDF triples composing an RDF graph.

A BGP is expressed as a sequence of its triple patterns and, if needed, separated by a period. A BGP could be interpreted as the combination of its constituent triple patterns. For example: "*?tweet* ⟨*http://rdfanon.org/types#timestamp*⟩ *?timestamp. ?tweet* ⟨*http://rdfanon.org/types#emotion*⟩ *?emotion.*".

A basic graph pattern successfully matches a subgraph of the RDF graph being queried if the RDF terms of the subgraph can be replaced for the variables in the pattern and the output is an RDF graph that is equivalent to the subgraph.

SPARQL enables the definition of prefixes to denote namespaces and incorporate these prefixes within the query pattern. This facilitates query reading by making it shorter.

2.4.2.2 - Query Forms

SPARQL queries can produce either result sets or RDF graphs as their results. It supports four query forms:

SELECT : Returns variable bindings, variables, and their associated values

CONSTRUCT : Returns an RDF graph

ASK : Returns a boolean result which can be either true or false.

DESCRIBE : Returns a single RDF graph comprising RDF data pertaining to resources found.

In the following explanation, we focus on **SELECT** type queries. For the **SELECT** form, the query consists of two primary components: the **SELECT** and **WHERE** clauses. The **SELECT** clause defines the desired projection. It determines which variables and their corresponding values the query should return. This is done by specifying a list of variable names in the SELECT clause. The **WHERE** clause defines the basic graph pattern that is used to match against the data graph. If no exact match is found against the data graph, then no data will be returned.

Listing 1 is an example of a SELECT query. Running it on the graph from Fig. 2.1 and Tab. 2.1 provides the results shown in Tab. 2.2.

SELECT-based result sets are easily presented in tabular form. Binding refers to a pair consisting of a variable and an RDF term. RDF terms (blank nodes, IRIs, or literals) will be bound to variables. In the obtained results in Tab. 2.2, there are two variables: tweet and timestamp that are shown as column headers. The query returns two solutions, every solution being presented as a single row within the table. In the first solution, variable tweet is bound to j.0:20 and the timestamp is bound to "Sun Jun 07 00:03:06 PDT 2009". In the second solution variable tweet is bound to j.0:19 and the timestamp is bound to "Sat May 30 00:56:54 PDT 2009."

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX j.0 : <http://rdfanon.org/types#>
SELECT ?tweet ?timestamp
WHERE {
  ?tweet j.0:timestamp ?timestamp .
}
```

Listing 1: SPARQL query selecting tweet and its corresponding timestamp

Similar to SQL, explicitly specifying specific variables in queries is not always necessary. Using the asterisk character (*) with **SELECT**, i.e., writing **SELECT ***, retrieves all known bindings for all variables. By rewriting the query in Listing with an asterisk character, the query will return all known bindings for all variables.

Table 2.2: Result set of the query in Listing 1 on the graph of Figure 2.1

tweet	timestamp
j.0:20	"Sun Jun 07 00:03:06 PDT 2009"
j.0:19	"Sat May 30 00:56:54 PDT 2009"

2.4.2.3 - Solution Modifiers

SPARQL query solutions are handled as solution sequence. By default, the solutions are unordered. It is possible to apply a solution sequence modifier to make another sequence, which in turn is utilized to produce the SPARQL query results. We present the solution sequence modifiers:

- Order modifier: enables the ordering of solutions based on one or multiple variables with the option to choose either ascending (default) or descending order.
- Projection modifier: enables selecting/projecting specific variables. This is done using the **SELECT** clause.
- Distinct modifier: enables removing duplicate solutions. It guarantees that all solutions in the sequence are unique, avoiding duplicates.
- Reduced modifier: enables the (optional) elimination of duplicate solutions.
- Offset modifier: enables the skipping of a specified number of solutions. An OFFSET of zero does not produce any change.
- Limit modifier: enables limiting the number of solutions, keeping it within bounds.

An example of a query with an order modifier presented as an ORDER BY clause is the query in Listing 2 that can be run on the graph from Fig. 2.1, producing the results in Tab. 2.3. The ordering comparator used here is the ascending specified with the ASC() modifier. It specifies that the solutions should be sorted in ascending order of the ?tweet term. Note that the ascending can also be specified with no modifier, which leads to the same result.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX j.0 : <http://rdfanon.org/types#>
SELECT ?tweet ?timestamp
WHERE {
   ?tweet j.0:timestamp ?timestamp .
}
ORDER BY  ASC(?tweet)
```

Listing 2: SPARQL query with an ORDER BY clause selecting the tweet and its corresponding timestamp

Table 2.3: Result set of the query in Listing 2 on the graph of Figure 2.1

tweet	timestamp
j.0:19	"Sat May 30 00:56:54 PDT 2009"
j.0:20	"Sun Jun 07 00:03:06 PDT 2009"

Listing 3 shows an example of a query with a LIMIT modifier. It can be run on the graph from Fig. 2.1, producing the results in Tab. 2.4. The LIMIT clause specifies that at most one result should be returned.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX j.0 : <http://rdfanon.org/types#>
SELECT ?tweet ?timestamp
WHERE {
   ?tweet j.0:timestamp ?timestamp .
}
LIMIT 1
```

Listing 3: SPARQL query with a LIMIT clause selecting the tweet and its corresponding timestamp

Table 2.4: Result set of the query in Listing 3 on the graph of Figure 2.1

tweet	timestamp
j.0:20	"Sun Jun 07 00:03:06 PDT 2009"

2.4.2.4 - SPARQL Keywords

SPARQL uses the keywords **FILTER** and **OPTIONAL** in the **WHERE** clause. **FILTER** removes solutions where the **FILTER** expressions evaluates to FALSE. The **OPTIONAL** keyword allows to define optional patterns. When data matches the pattern, additional bindings will be added to a result set, and hence it will be added to the solution of the query. However, if no match is found, it generates no bindings yet the query solution is still preserved.

Another used keyword is the **UNION** keyword. SPARQL offers a way of combining graph patterns to enable matching one of various alternative graph patterns. Pattern alternatives are expressed with the **UNION** keyword.

2.4.2.5 - Aggregate functions

SPARQL 1.1 introduced seven aggregate functions that can be performed over groups of solutions, which we list in Tab. 2.5. Initially, a solution set is composed of a single group, including all solutions. The **GROUP BY** clause can be utilized to designate the grouping. Using the **GROUP BY** clause makes it possible to compute aggregate values for a solution.

In this context, the solution will be split, forming group(s), and afterward for every group, the aggregate value is computed.

The aggregate functions can as well be employed in conjunction with the **HAVING** clause. HAVING works across grouped solutions in the same manner as FILTER works across non-grouped solutions.

Table 2.5: SPARQL Aggregate functions

Function Name	**Meaning**
Count	count the number of values present within the aggregate group
Sum	sums the values across the aggregate group and returns a numeric value
Min	returns the minimum value among a set of values within a group
Max	returns the maximum value among a set of values within a group
Avg	returns the average numeric value across a group
GroupConcat	do a string concatenation for the values within a group
Sample	returns random value from a given set of values/ samples one value from a given set of values

To count the total number of tweets in our example, we can write the query in Listing 4. It returns the value two over our running example of Fig. 2.1. The aggregate function count (*) counts the number of current solutions. If required, we can ask the number of solutions for a specific variable, as in the example in Listing 5. This will also yield the value 2 for ?c.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX j.0 : <http://rdfanon.org/types#>
SELECT (count(*) AS ?c)
WHERE {
   ?tweet rdf:type j.0:tweet .
}
```

Listing 4: SPARQL query that counts the total number of tweets

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX j.0 : <http://rdfanon.org/types#>
SELECT (count(?tweet) AS ?c)
WHERE {
   ?tweet rdf:type j.0:tweet .
}
```

Listing 5: SPARQL query that counts the total number of tweets

To count the number of tweets tweeted by each user, we can use the GROUP BY clause as in the example of Listing 6. This will return the number of tweets for each user. The results are shown in Table 2.6. There is only one user in the considered example, so the query returns the number of tweets solely for this user.

```
PREFIX j.0 : <http://rdfanon.org/types#>
SELECT ?user (count(*) AS ?c)
WHERE {
   ?user j.0:tweeted  ?tweet.
}
GROUP BY ?user
```

Listing 6: SPARQL query that counts the total number of tweets tweeted by each user

Table 2.6: Result set of the query in Listing 6 on the graph of Figure 2.1

user	c
j.0:SGroup	2

2.4.2.6 - SPARQL 1.1 Update

SPARQL 1.1 Update [50] is a W3C standard that enables updating the underlying RDF graphs in a standard declarative manner. An operation is defined as an action to be done that leads to the modification of graph data. Operations are typically used to refer to SPARQL Update queries performed on graphs. Two classes of update operations are provided:

Graph Update Operations : **INSERT DATA** operation adds triples to a graph. If the destination graph does not exist, it is created. **DELETE DATA** removes triples from a graph. **DELETE/INSERT** operation delete triples from a graph or add triples to a graph. This is the primary pattern-based operation for graph updates. The deletion or addition is performed according to bindings detailed in the WHERE clause for a query pattern. **LOAD** operation loads a graph and adds its triples into a given destination graph. If the graph doesn't exist, it is created. **CLEAR** operation deletes all triples contained in the specified graph(s).

Graph Management Operations : **CREATE** create a new empty named graph. **DROP** deletes a specified graph(s). **COPY** copies a graph into a specified destination graph. If the destination graph contains any content or data, this data will be deleted before copying. **MOVE** moves a graph to another. If the destination graph does not exist, it is created. If the latter contains any content or data, this data will be deleted before insertion. The source graph is deleted after insertion. **ADD** adds the data from the input graph to the destination graph. If the destination graph does not exist, it is created. This data will be preserved if the latter contains any content or data. The source graph data is also preserved.

An example of the **DELETE/INSERT** operation is given in Listing 7. Here, it is required to specify either a DELETE or INSERT clause or both. The results for the query pattern will be generated first. After that, deletion is done, and then the insertion is performed. This SPARQL update will change the timestamp of all tweets with "Sat May 30 00:56:54 PDT 2009" to "Sun May 31 00:20:54 PDT 2009". The updated graph resulting from the application of this query to the graph from Fig. 2.1 and Tab. 2.1 is presented in Tab. 2.7.

```
PREFIX j.0 : <http://rdfanon.org/types#>
DELETE { ?tweet j.0:timestamp  "Sat May 30 00:56:54 PDT 2009" }
INSERT {  ?tweet j.0:timestamp  "Sun May 31 00:20:54 PDT 2009" }
WHERE {
   ?tweet j.0:timestamp  "Sat May 30 00:56:54 PDT 2009"
}
```

Listing 7: SPARQL Update query

Table 2.7: Updated RDF graph by the update query in Listing 7

Subject	**Predicate**	**Object**
http://rdfanon.org/types#SGroup	http://rdfanon.org/types#name	"SGroup"
http://rdfanon.org/types#SGroup	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://rdfanon.org/types#person
http://rdfanon.org/types#SGroup	http://rdfanon.org/types#tweeted	http://rdfanon.org/types#19
http://rdfanon.org/types#19	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://rdfanon.org/types#tweet
http://rdfanon.org/types#19	http://rdfanon.org/types#text	"Its freezing "
http://rdfanon.org/types#19	http://rdfanon.org/types#emotion	"4"
http://rdfanon.org/types#19	http://rdfanon.org/types#timestamp	"Sun May 31 00:20:54 PDT 2009"
http://rdfanon.org/types#SGroup	http://rdfanon.org/types#tweeted	http://rdfanon.org/types#20
http://rdfanon.org/types#20	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://rdfanon.org/types#tweet
http://rdfanon.org/types#20	http://rdfanon.org/types#text	" Stuck in traffic"
http://rdfanon.org/types#20	http://rdfanon.org/types#emotion	"0"
http://rdfanon.org/types#20	http://rdfanon.org/types#timestamp	“Sun Jun 07 00:03:06 PDT 2009"

2.4.3 - Reasoning using ontologies such as RDFS and OWL

One of the main advantages of RDF data is that we can apply formal reasoning to it. In order to perform reasoning, additional information on the structure and relations between the concepts represented must be available. This information is contained in *ontologies*. We next describe in more detail what ontologies are, how they are built and represented, and how they can be used to perform inferences.

Ontology, derived from philosophy, points to the branch of knowledge that deals with characterizing and depicting different types of entities present in the world and their relationships.

Gruber introduced the notion of ontology in 1993 as an “explicit specification of a conceptualization” [51]. In 1997, Borst [52] added to this definition that the conceptualization has to convey a joint perspective among multiple groups. Moreover, Borst emphasized that such conceptualization should be specified in a formal machine-readable format. Consequently, he introduced ontology as a “formal specification of a shared conceptualization.”

Vocabularies establish the definition of terms, concepts, and the relationships that hold among them employed to represent a particular subject or domain that exists in some area of interest. The distinction between "vocabularies" and "ontologies" is not well-defined. It is common to refer to an intricate and potentially fully formal collection of terms as an ontology, while "vocabulary" is employed when rigorous formalism may not be required. Vocabularies serve as the fundamental foundation for inference mechanisms on the Semantic Web.

The W3C designed a wide range of techniques to present, explain, and define various types of vocabularies in a standard manner. The primary ones are RDF Schema (RDFS) and the Web Ontology Language (OWL).

RDFS [53] is a lightweight extension of the primary RDF vocabulary with a collection of terms that compose the RDFS vocabulary. For RDF data, RDF Schema offers a vocabulary for data modeling. It offers means to define groups of resources among with their relationships. RDFS introduces terms for classes, properties, and other constructs to describe well-defined relationships between classes and properties, serving as the essential core of RDF Schema. In particular, it permits the definition of a hierarchy of concepts through the definition of subclasses and subproperties. RDFS introduces terms detailing the deliberated manner of utilizing classes and properties in conjunction within RDF data. For instance, it allows the specification of the domain and range properties that enable the association of a class with a property's subject and object.

OWL [54] is a language designed to define ontologies on the Web. It extends the essence RDFS vocabulary with a set of novel terms to express much richer meaning and semantics than RDFS alone.

OWL introduces additional vocabulary for detailing classes and properties. OWL defines two distinct classes of properties, namely ObjectProperty and DatatypeProperty. It also enables the definition of property semantics (TransitiveProperty, SymmetricProperty, FunctionalProperty, InverseFunctionalProperty,...) and semantic relations between different properties (equivalent-Property, inverseOf, ...).

Similarly, it proposes to relate classes with each other (equivalentClass, disjointWith). New classes can be defined by applying set operations such as intersection, union, or complement to other classes (intersectionOf, unionOf, complementOf). OWL also offers a vocabulary that allows the declaration of a diverse types of restrictions to define new classes (someValues-From, allValuesFrom, hasValue,...). It supports number-based restrictions on class members (minCardinality, maxCardinality, cardinality,..).

OWL has sublanguages, each characterized by its expressiveness. From least to most expressive, these sublanguages are OWL Lite, OWL DL, and OWL Full. Every sublanguage extends its former in terms of expression power and conclusions correctly derived. OWL 2 [55] has been W3C recommendation since 2009.

2.4.4 - Inference

The term "inference" on the Semantic Web can be summarized by the discovery of new relationships.

Semantic Web data intends to describe different resources and the relations between them. Inferences denote the process that can produce new relationships derived from the data and extra information given as vocabulary or rule sets. Depending on the specific implementation approach, the newly established relationships are directly included to the existing set of data or retrieved during query execution.

The supplementary information relied on to make inferences can be specified through the use of vocabularies or rule sets. Ontologies foremost focus on classification procedures. They provide a manner to define groups of resources among their relationships, stress on introducing classes, subclasses, properties, class instances, and the well-defined relationships between classes and properties. On the contrary, rule sets focus on establishing a broad technique for finding out and producing new relationships by building upon existing ones.

Within the scope of W3C recommendations pertaining to the Semantic Web, RDFS, OWL, or SKOS (Simple Knowledge Organization System) are the preferred tools for defining ontologies. On the other hand, the Rule Interchange Format (RIF)- a format for exchanging rules over the Web- addresses rule-based practices.

Leveraging inference in the Semantic Web context is a distinguished way to enrich the degree of data integration. It involves uncovering new relationships, automated analysis of data content, and handling and regulating Web knowledge. In addition, methods built upon inference are crucial for detecting any inconsistencies in the integrated data.

Another term that one could come across is 'reasoning.' Reasoning generally aims to find what results from what and is thereby related to inference. Nonetheless, reasoning is a delicately broader term comprising different forms of reasoning, such as deductive, inductive, and abductive reasoning. Diverse precise types of inference can be defined for these different forms of reasoning.

Regarding reasoning, our primary focus centers around deductive reasoning that enriches the RDF graph by leveraging a portion of the formal semantics of RDFS and OWL to infer novel information. RDF graphs can be fodder for automated deductive reasoning systems. One of the key characteristics of the RDF graphs lies in their capacity to employ automated deductive reasoning to infer new information from pre-existing triples. There exist two major strategies of reasoning: forward chaining and backward chaining. Forward chaining involves the iterative application of inference rules on existing facts to deduce all possible new facts. In contrast, backward chaining commences from the objective and proceeds backward by employing inference rules to find supporting facts or evidence.

It is worth mentioning that as the expressiveness of an ontology language increases, so does the computational costs for inferring new information. This is why reasoning in OWL Full is typically challenging, with the most crucial tasks for the language being undecidable. Researchers in this field concentrate on investigating the trade-offs between expressiveness and reasoning complexity, targeting to reach a good compromise.

2.5 - Mapping Relational Databases to RDF

The vast amount of data underpinning the Web is stored in relational databases, which are excellent tools to store and manage huge amounts of data efficiently. Relational databases have a demonstrated history of reliability and optimized query execution. However, their potential to describe the semantics of the data is limited. RDF, on the other hand, has more expressive power. This enables machines to understand the meaning or 'semantics' of data, process, interpret, and reason over these data. In this next section, we discuss existing technologies that allow mapping relational data to RDF data. We will use these techniques in our prototype and experimentation.

Mapping these bulk of data from relational databases to RDF [56] has been an active field of research during the last two decades. Historically, in March 2008, the W3C launched the RDB2RDF (Relational Database to Resource Description Framework) incubator group[2], part of the Semantic Web Activity, to investigate the issues associated with this transformation and to standardize languages for mapping relational data and relational database schemas into RDF and OWL [57]. In September 2012, the RDB2RDF Working Group published two Recommendations: Direct Mapping (DM) [58] and customized mapping (CM) R2RML [59].

2.5.1 - Direct Mapping (DM)

The W3C recommendation DM defines direct mapping good practices. It suggests a formalization of the rules defined by Tim Berners-Lee [60]. Direct mapping, simple and automatic mapping of relational data to RDF, converts relational database (data and schema) to RDF graphs. The RDF generated straightforwardly is based on the structure of the database schema. URIs are automatically generated [60]. The W3C DM recommendation defines simple mapping rules to map relational data to RDF as follows [61]:

- The subject URI is formed from the combination (base URI, table name, primary key column, the symbol =, and primary key value). Every row in a database table produces a set of triples with a common subject; this shared subject URI. If there is no primary key, blank nodes are created.
- The literal triples are formed from the subject URI as a subject, the combination (base URI, table name, the symbol #, and the column name) as the predicate, and the column value as the object.
- The case where the row in a database table contains a FK. It is stated that [3] "Each FK produces a triple with a predicate composed from the FK column names, the referenced table, and the referenced column names. The object of these triples is the row identifier for the referenced triple". This will produce the subject URI as the subject, the combination (base URI, table name, the string #ref-, column name) as the predicate, and the combination (referenced table name, primary key column, the symbol =, and primary key value) as object.

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

[2]http://www.w3.org/2001/sw/rdb2rdf/
[3]https://www.w3.org/TR/rdb-direct-mapping/#lead-ex

is to represent many-to-many relation as simple triples [62]. When DM is applied, the join table will be translated into a distinct class. The result of RDF is not actually what many-to-many relationship means.

2.5.2 - R2RML Mapping

Customized mapping (CM) R2RML [59] is a RDB to RDF mapping language that allows to manually customize the mapping. The expert user has to know the relational database (schema and data) and the domain ontology to express the schema utilizing an existing target ontology and therefore translate relational database to RDF datasets. The W3C RDB2RDF Working Group discussed requirements regarding a RDB2RDF mapping language [63]. They proposed a set of core requirements for the R2RML. One of the core and mandatory requirements is to expose many-to-many join tables as simple triples [62].

An R2RML mapping document is written in RDF in Turtle syntax[4]. An R2RML mapping refers to logical tables to retrieve data from the input database. The logical table can be a base table, an SQL view, or a valid SQL query. Using TriplesMaps, every logical table is mapped to RDF triples. The TriplesMap is a rule that maps every logical table row into a number of RDF triples. A TriplesMap is constituted of exactly one SubjectMap and multiple PredicateObjectMaps. PredicateObjectMaps are composed of PredicateMaps and ObjectMaps. Triples are generated by combining the subject map with every predicate from the PredicateMap and every value from its corresponding ObjectMap.

[4]http://www.w3.org/TR/turtle/

3 - Background: Privacy

Contents

3.1 - Introduction

This Chapter is dedicated to the second fundamental field this thesis belongs to: Privacy.

We start by introducing general concerns and background of privacy and privacy-preserving data publication in particular. We then develop on the key related concepts that are attack models, privacy models, and general operations for anonymization.

Finally, we give an overview of the two principal classes of privacy schemes: those deriving from k-anonymity and the probabilistic privacy models, of which DP is the most representative.

3.2 - Privacy-preserving data publishing

In the present Big Data era, the dramatic growth in the volume and variety of data collected, especially personal data scattered in numerous data sources, presents critical concerns about the privacy of individuals [64, 65]. An adversary can take advantage of these personal data to get additional information and gain knowledge that might possibly not be permitted to get, therefore violating individual's privacy [66–68].

A common practice to address this issue is to remove the explicit identifiers attributes that hold information that explicitly identifies an individual, e.g., name, social security number. Nevertheless, these practices have been demonstrated as inadequate because an individual can still be uniquely identified by erasing explicit identifiers attributes. The data left can be linked or matched with another datasets or examined to find distinct features and attribute to re-identify individuals.

From a legal standpoint, the interpretation of personally identifiable information varies based on the jurisdiction in question. For instance, under California Senate Bill 1386, personal identifying information includes Social Security numbers (SSN), driver's license numbers, account numbers, debit/credit card numbers, and any necessary security or access code or password enabling access to an individual's financial account. On the other hand, the European Union employs a more comprehensive definition:

"Any information relating to an identified or identifiable natural person [...] [A]n identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity. [...] [A]ccount should be taken of all the means likely reasonably to be used either by the controller or by any other person to identify the said person." [69]

Many legal regulations have been introduced to safeguard private user information, like California Privacy Rights Act[1] in the United States, the official European Union's General Data Protection Regulation (GDPR) [70], and the Singapore Personal Data Protection Act in Singapore [71]. Given the introduction of GDPR, that every organization must comply with,

[1]https://oag.ca.gov/privacy/ccpa.

handling personal information has turned essential, pivotal, and critical in Europe.

In what follows, we present the most prominent approaches that are primarily dedicated to traditional relational legacy systems. Consequently, these approaches cannot be applied directly to the RDF data context. These approaches typically rely on variants or adaptations of two well-known privacy methods: k-anonymity and Differential Privacy (DP).

The development of techniques and tools for publishing data in increasingly antagonistic settings is a work of extreme significance, in order to keep the released data essentially useful while protecting privacy. This undertaking is named privacy-preserving data publishing (PPDP).

PPDP offers techniques and means that enable disclosing valuable information while protecting privacy. The body of work in PPDP was already significant in 2010 [72], and the interest in the field research community has continued growing. Numerous approaches have been suggested to accommodate various data publishing scenarios, resulting in a wide range of privacy models and, consequently, anonymization methods. When considering each of these methods, many attack models, such as record linkage, attribute linkage, table linkage, and probabilistic attacks are taken into account. These have given rise to two key assumptions: firstly, the extent of knowledge the attacker has concerning the victim and secondly, the conditions that give rise to a potential privacy leak.

We split privacy models following [72] into two classes following their attack standard: the first group postulates that a privacy breach arises when an attacker successfully establishes a link between a record owner and a record, a record owner and a sensitive attribute, or the released data table. These linkages are respectively referred to as record linkage, attribute linkage, and table linkage. On the other hand, the second group's objective is to satisfy the uninformative principle [73]. It considers the occurrence of probabilistic attacks if the latter is not satisfied.

3.3 - Privacy models against linkage attacks

3.3.1 - The historical Sweeney privacy breach

The concept of privacy is commonly linked to the relational (tabular) data model where datasets are structured in relations (tables) comprising columns, known as table attributes, and rows, also known as records. In the privacy setting, the following terminology is used to categorize attributes.

- Explicit identifiers (EID) attributes, including information that directly identifies table record owners, e.g., name, social security number, and driver's license.
- Quasi-identifiers (QID) are sets of attributes that in combination can potentially identify records owner and can easily be known to an adversary, e.g., date of birth, gender, zip code, and ethnic group.
- Sensitive attributes (SA) are highly critical attributes composed of individual sensitive

information, e.g., disease, religion, salary, and political opinion.

- Non-sensitive attributes (NSA) are attributes that do not fall into any of the former categories.

EID, QID, and SA can be deemed to be private attributes, whereas NSA are basically not concerned with privacy issues.

In 2002, L. Sweeney [1] demonstrated a privacy breach to the Massachusetts state Governer at that time, William Weld. Sweeney succeeded in linking a record to a particular individual: William Weld. This was done by having access to two datasets. The first dataset included the voter registration list for Cambridge Massachusetts. It had information about their name, address, etc. The group insurance company (GIC) released the second dataset, a supposedly anonymized dataset that included patients' medical data. While their names and social security numbers were removed, it had information about their ethnicity, visit date, diagnosis, etc. .As shown in Fig 3.1, Sweeney linked these two datasets through the combination of zip code, birth date, and sex. Thus, linking diagnosis and medical records to specific individuals was possible, in particular concerning William Weld himself. Thereupon, Sweeney proved using a counter-example, a re-identification by straightforward linking on common attributes, that simply removing EID was not an acceptable PPDP strategy.

Based on 1900 Census summary data [74] regarding the United States (US) population, Sweeney found that 87 % of the population in the US had recorded characteristics that probably considered them unique through these three specific attributes: zip code, birth date, and gender. It turns out that 87 % of the US population could be uniquely identified using zip code, birth date, and gender.

The set of such attributes that, in combination, can potentially identify individuals has been called the quasi-identifiers by Dalenius [75].

3.3.2 - k-anonymity

Anonymization [75, 76] is the most popular form of PPDP that aims to conceal the identity and/or sensitive information of record holders, supposing the keeping of sensitive data for data analysis purposes. Obviously, erasing the EIDs of record holders is a first, yet insufficent, step towards anonymization.

To thwart record linkage through QID, Samarati and Sweeney [12] and Sweeney [1] proposed the k-anonymity privacy notion. The objective is to release individual particular information, ideally structured in tables composed of rows (or records) and columns (fields or attributes), in such a manner that restricts the capability to link it with other external information through the QID.

A table release satisfies k-anonymity if every released table record is indistinguishable from at least k−1 other table records with respect to the QID attributes. A table adhering to this requirement is said to be k-anonymous. In other words, in a k-anonymous table, there exist at least k records for any set of QID's values. A group of records with the same QID collectively

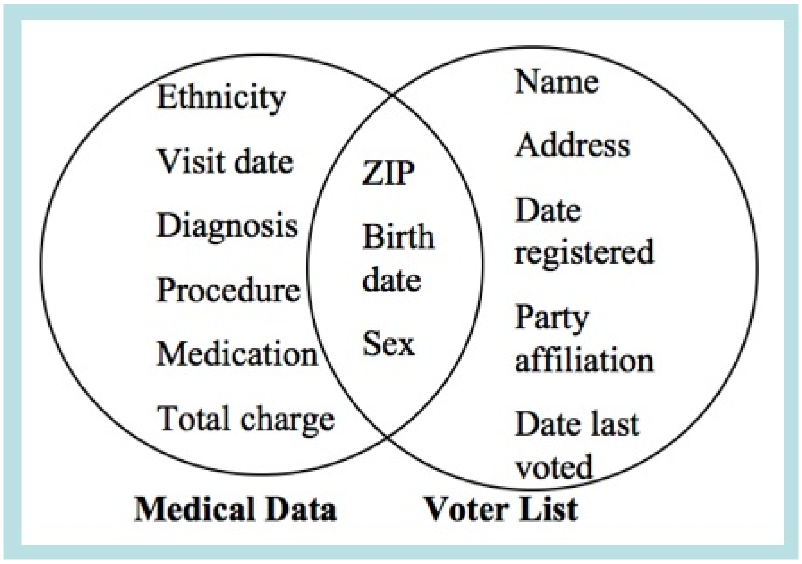


Figure 3.1: Re-identification by directly linking on quasi-identifiers from Sweeney [1]

.

constitute an equivalence class (EC). For example, if $k = 5$, every EC must have at least five identical records. Hence, the probability of an adversary to successfully link a victim to a particular record via QID is at most $1/k$. In this example, at most $1/5$. This is the probability knowing that the victim is presented in the released table and knowing its exact QID values. Practically, each data publisher would select the parameter k proportionate to the probability of re-identification deemed acceptable [77]. Increasing the values of k suggests a decrease in re-identification probability. However, it means additional data distortion and more information loss. Broadly speaking, immoderate anonymization would induce extremely high information loss, making the data essentially useless.

Generalization and suppression techniques can be employed to provide k-anonymity. Generalization is one of the widespread anonymization operations wherein QID values are substituted with more general values that are semantically compatible. For instance, age may be substituted with an age group (i.e., age:25 into 'Age Group: 20-29'). Suppression is simply erasing particular values such that they are not released. There are several generalization schemes, such as full-domain generalization, subtree generalization, sibling generalization, cell generalization, and multidimensional generalization schemes. Similarly, there are several suppression schemes, such as record suppression, value suppression, and cell suppression [72].

Different studies have demonstrated that searching for optimal anonymization is an NP-hard problem [78–81]. Samarati [78] demonstrated that it demands significant costs to achieve the optimal k-anonymity through full-domain generalization. [81] showed that the optimal (α, k)-anonymity through cell generalization is NP-hard. Various algorithms have been proposed for the problem [79, 80, 82, 83].

The popular formal mathematical protection model k-anonymity is among the first anonymization models and definitely the most popular as it remains a point of reference to this day.

3.3.3 - L-diversity

Attribute linkage attack is similar to record linkage. The attacker is assumed to know the exact QIDs of the victim and that the victim's record is presented in the released table. In the attribute linkage attack, the attacker attempts to link a particular attribute value to an individual. In this context, the attacker could infer the individual's sensitive attributes values [72].

The attribute Linkage attack can be established in many ways. In this regard, Machanavajjhala et al. [73] presented examples of two attacks on k-anonymous datasets: the homogeneity attack and the background knowledge attack. These attacks permit an adversary to infer the value of the sensitive attribute of an individual and thus can be employed to jeopardize a k-anonymous dataset.

In their work, the authors explained the homogeneity attack, where an attacker can infer the sensitive attribute value of a victim by leveraging the case where a unique or not diverse sensitive attributes values being presented in an equivalence class. Even though the attacker can not know which exact record relates to the target individual, it may infer its sensitive attributes by identifying the equivalence class she belongs to. The background knowledge attack is where an attacker possesses particular or general information concerning an individual or group regardless of its source. Such knowledge allows to discard possible sensitive values for an individual within an equivalence class and, therefore, infer the correct value. The attack occurs when the attacker uses this background knowledge to find out sensitive information, compromising the privacy of an individual or group of individuals. Thus, the absence of diversity in the values of the sensitive attributes within k-anonymized datasets enables the occurrence of these attacks.

The authors of [73] proposed the diversity principle, named L-diversity, to overcome homogeneity and background knowledge attacks. L-diversity is among the first endeavors [72] to counter attribute linkage attack.

The central point of L-diversity is to guarantee that sensitive attributes are well represented within each group. In other words, it necessities every quasi-identifier group to have at least l "well-represented" sensitive values.

There are many instantiations of the l-diversity principle that vary on what "well represented" means. The looser interpretation of "well represented" is the existence, ensuring the presence of at least l distinct values for the sensitive attribute in each equivalence class. This is known as distinct l-diversity.

Machanavajjhala et al. [73] provided two stronger instantiations of the L-diversity principle: entropy L-diversity and recursive (c, L)-diversity. Besides, they proposed two additional instantiations, named positive disclosure-recursive (c,L)-diversity and negative/positive disclosure-recursive (c,L)-diversity.

K-anonymity and L-diversity have strong structural similarities, allowing for the adaptation of k-anonymity algorithms to serve L-diversity. The anonymization operations used here are also generalization and suppression, similar to k-anonymity. However, due to the consideration of sensitive attributes while the creation of groups, additional modifications are typically required as compared to k-anonymity.

3.3.4 - Anatomization

In contrast to generalization and suppression, anatomization by Xiao and Tao [84] does not change the quasi-identifiers or the sensitive values. Instead, it breaks the correlation between quasi-identifiers and sensitive values. In particular, their approach involves releasing two separate tables: a quasi-identifier table (QIT) that holds the QID attributes and a sensitive table (ST) that holds the sensitive attributes, and both tables conserve the unique common attribute, Group-ID.

Anatomy [84] provides a key benefit: it keeps the data unmodified in both QIT and ST tables. The authors demonstrated through broad experimentations that their technique permits notably more efficient data analysis than the traditional publication techniques that rely on generalization. In particular, compared to the generalization techniques, it can more accurately respond to aggregate queries, including domain values of the QID and sensitive attributes.

3.3.5 - t-closeness

The authors in [85] showed that l-diversity has some shortcomings and is not enough to prevent attribute linkage attacks. They introduced two attacks on l-diversity, namely skewness attack and similarity attack. The skewness attack is possible when the distribution is skewed, showing that l-diversity is not enough in such cases to prevent attribute disclosure. On the other hand, similarity attack can be triggered when the sensitive attribute values in an equivalence class are different yet semantically related, allowing the attacker to infer crucial information. To counter skewness attack, they proposed a privacy notion named t-closeness, which demands the sensitive values distribution within each equivalence class to be adequately close to its general distribution in the whole table. t-closeness guarantees no greater than a threshold t distance between the two distributions. They select the Earth Mover Distance measure [86] to achieve this task.

Achieving t-closeness would considerably reduce the data utility as it demands the distribution of sensitive values to be close in all equivalence classes. Hence, this considerably deteriorates the correlation between QID and sensitive attributes. To address this, [87] proposed to loosen the condition by adapting the threshold in response to the increased threat to skewness attack. Another approach is to use the probabilistic privacy models [72].

3.4 - Probabilistic privacy models againts probabilistic attacks

There exists a class of privacy models that concentrates on how the attacker would alter his/her probabilistic belief regarding the victim's sensitive information when viewing published data, as opposed to what the attacker can precisely associate to a specific victim from records, attributes, and tables [72]. Broadly speaking, this category of privacy models strives to fulfill the uninformative principle [73]. The attacker should gain minimal additional information from the published table compared to his/her existing underlying background knowledge. The objective is to minimize the variation between the prior and posterior beliefs. A probabilistic attack occurs if there is a large difference between such beliefs. Several works strive to achieve the uninformative principle [73].

3.4.1 - (c,t)-Isolation

Chawla et al. [88] are among the first works targeting establishing a privacy framework designed for statistical databases. Their work served as a fundamental basis for probabilistic methods and DP. They indicated that the attacker's –called in their work "isolator"– ability to isolate any record should not increase after accessing the sanitized database. Accordingly, they introduced a privacy framework to prevent (c,t)-isolation in a statistical database. This implies that for an attacker's target value P and the inferred data Q for a specific point in the dataset, given d the distance between P and Q, a ball centered on Q of radius c $\times$ d encompasses fewer than t points in the database.

3.4.2 - Differential privacy

3.4.2.1 - Intuition

Consider a database containing information about whether a person suffers from the flu (e.g., Fig. 3.2). Each row corresponds to an individual. For the column 'Flu', 0 indicates that the person does not suffer from flu, and 1 indicates it does. We may learn statistics concerning the underlying population from this database and release these statistics publicly. Nevertheless, individual privacy may be jeopardized in this case due to possible information leakage.

Let's consider that the database offers a query interface Q_j(D) that gives the sum of the second column, 'Flu,' of the first j rows. The query outputs an aggregate outcome and does not target any particular individual.

Let's consider a scenario where an attacker wants to infer whether or not Alice has the flu, having the background knowledge that Alice's record is the last. The attacker can pose two queries, Q_5(D) and Q_4(D), and then calculate Q_5(D)$-Q_4$(D), which is the difference between their outputs. If the result is 1, this indicates that Alice has the flu; conversely, she is deemed free from Flu. This demonstrates that even when the database is not publicly released, releasing exact query outputs may compromise individual privacy.

The intuition of differential privacy (DP) is to hide the presence or absence of a single individual, i.e., make it difficult to determine whether her data contributed to the result of the query. A typical way to achieve this goal is to add some noise to the output of the query. To preserve Alice's privacy in the example presented in Fig. 3.2, we can add some noise to Q_5 and Q_4 outputs, ensuring that the difference between Q_5 and Q_4 and whether or not Alice has the flu are independent with high probability.

3.4.2.2 - Differential Privacy Definition

In 2006, DP emerged as a solution to address the privacy concerns of individuals in structured datasets (like census datasets), where the disclosure of data could potentially cause

Figure 3.2: A Database Example

Name	Flu
Bob	1
Umeko	0
Carine	1
Joe	1
Alice	0

privacy risks. Dwork et al. [13] admited the certainty of privacy breaches in any valuable data release and introduced a privacy definition established on 'plausible deniability' for individuals, controlled by an epsilon (ε) parameter that dictates the level of plausible deniability.

Two natural models for releasing differential private data are interactive and non-interactive [89]. In the non-interactive or offline setting, the data curator, or holder, publishes certain types of objects, like "synthetic database," statistics, or "sanitized database." Following this publication, the curator can shut down the initial data. In the interactive or online setting, users can pose queries adaptively and get (possibly noisy) answers.

The interactive setting is considered in this manuscript, which corresponds to a way of achieving DP through adding noise to the query output. A framework of output perturbation in DP is shown in Fig. 3.3, where privacy is achieved by adding random noise that hides the private information. Query sensitivity is the key parameter that defines the magnitude of added noise. The added noise ideally should preserve the data's utility. The challenge, therefore, is to maximize the utility while protecting the privacy of individuals.

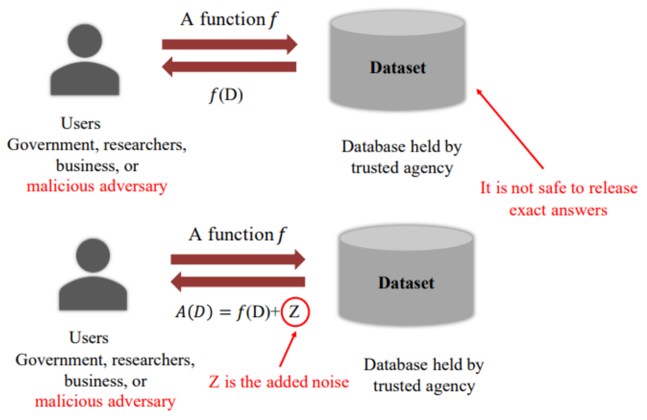

A function f
f(D)
Dataset
Users
Government, researchers, business, or
malicious adversary
Database held by trusted agency
It is not safe to release exact answers
A function f
Dataset
$A(D) = f(D) + Z$
Users
Government, researchers, business, or
malicious adversary
Z is the added noise
Database held by trusted agency

Figure 3.3: Output perturbation in differential privacy [2]

Although a multiset of records can be used to model databases, DP research frequently takes into account alternative models for suitability. Two popular representations are the histogram and the vector representation, which are explained below.

Intuitively, the goal of DP is to ensure that an attacker is not able to infer (beyond a certain probabilistic threshold) whether an individual contributed to the result of a query over a database. The exact protection and the notion of individual contributions are defined based on the concept of neighboring (or adjacent) databases- databases that differ by the data of a single individual. Given a distance on databases, we say that two databases are neighbors (or adjacent) if they are at a distance 1. In this section, we note d the distance on the considered space. An algorithm is differentially private if it is likely to yield the same output on neighboring databases, promising that one's participation in the database will not in itself be disclosed. DP tackles the paradox of learning about a population while learning nothing about single individuals [89]. The robustness of DP is quantified by a positive parameter ε, called privacy budget or privacy loss.

The popular definition of neighboring databases[2] supposes that each individual contributes to precisely one row in the database, meaning that neighboring databases differ in one row. Two flavors of DP have been defined [26]: unbounded and bounded.

In unbounded DP [14], a neighboring database is obtained by the addition or deletion of a single record in the database. Under this setting, neighboring databases differ in size. Research in this context usually represents the database as a histogram h(D) $\in \mathbb{N}^{|\mathcal{X}|}$ where every entry h(D)[i] denote the number of elements in the database D of type i $\in \mathcal{X}$. Under this representation, the distance between two databases D_1 and D_2 is $d_{unbounded}$ (D_1, D_2)= $\|$ h(D_1) - h(D_2) $\|_1$ where $\|\|_1$ denotes the L1 norm.

In bounded DP [13], a neighboring database is obtained by changing the value of exactly one record. Under this setting, neighboring databases have the same size, n. Research in this context usually represents the database as a vactor D $\in \mathcal{X}^n$ where D[i] denotes the data contributed by individual i. The distance between two databases D_1 and D_2 is $d_{bounded}$ (D_1, D_2)= $|\{i | D_1[i] \neq D_2[i]\}|$.

A formal definition of DP is given in the following.

Definition 3.1 (ε, δ-differential Privacy [89])**.** *Given $\varepsilon > 0$ and $0 \leqq \delta < 1$, a randomized mechanism K: $\mathcal{D} \rightarrow \mathbb{R}$ preserves (ε, δ)-differential privacy if for any pair of databases D_1 and $D_2 \in \mathcal{D}$ such that d(D_1, D_2) = 1, and for all sets S of possible outputs:*

$$Pr[K(D_1) \in S] \leq e^{\varepsilon} Pr[K(D_2) \in S] + \delta \tag{3.1}$$

where the probability is taken over the randomness of K.

When δ >0, $(\varepsilon, \delta) - DP$ relaxes ε-DP by a small probability controlled by parameter ε. Typically, a common choice of δ is to set it considerably smaller than 1/n where n is the number of users in the database [13]. In this manuscript, we consider ε-DP which is $(\varepsilon, \delta) - DP$ with $\delta = 0$.

The ratio between the output probability distributions for neighboring databases D_1 and D_2 is bounded by e^{ε}. A smaller ε will yield a higher level of privacy preservation, but it

[2]We stress here that the definition of neighboring databases in the context of RDF is not trivial, and poses one of the research questions of the thesis.

comes at the expense of decreased data accuracy. Observe that when ε is set to zero, both distributions become equal, resulting in no information leakage. The privacy preservation level, in this case, reaches its peak, achieving 'perfect' protection. This aligns with the privacy objective articulated by Dalenius [90] in 1977, which highlights that accessing a statistical database should not provide the means to learn anything about an individual that would not be attainable without such access. Nevertheless, as demonstrated by Dwork [14], complying with this objective makes it impossible to achieve any level of utility. Thus, we should admit the inevitability of disclosing certain information about individuals to achieve some level of utility, which corresponds to raising the value of ε. Therefore, the choice of ε should balance the trade-off between privacy and data utility.

Randomized response is a technique introduced by Warner [91], aiming to offer individuals a means of plausible deniability when answering sensitive or embarrassing questions. It is probably the first example of a privacy-preserving algorithm in this spirit. Survey participants flip a coin. If 'heads,' they throw the coin again and answer "Yes" if heads and "No" if tails. If "tails," they answer honestly. Warner's technique precedes the emergence of DP by several decades, yet it encapsulates the intuition that underlines the pledge of DP. Possibly not by mere coincidence, the randomized response algorithm was among the pioneering's that demonstrated compliance with the principles of DP [89].

DP spans many research areas, and considerable work has been conducted across various application domains, encompassing location privacy [92–94], recommender systems [95, 96], social networks [97–101], and different applications.

3.4.2.3 - Properties of Differential Privacy

DP is currently the gold standard for data privacy, mainly for its mathematical properties (it composes well and is robust to post-processing) and its resilience against background knowledge of the attacker (its assurance remains valid across all prior knowledge). These properties distinguish it from anonymization-based definitions, which are vulnerable to adversarial attacks when joined with additional datasets.

The privacy breach demonstrated by L. Sweeney and explained in detail in Section 3.3.1 is one example of such an attacks. DP asserts resistance against such forms of attacks [13].

A significant and crucial property of DP is composability. It guarantees that running two differentially private mechanisms still satisfies DP, yet at an increased privacy cost. We introduce below two types of composition.

Theorem 3.1 (Sequential Composition [89]). *Let K_1 be an ε_1-DP mechanism and K_2 be an ε_2-DP mechanism. Then the mechanism K defined by K($\mathcal{D}$) = ($K_1(\mathcal{D})$,$K_2(\mathcal{D})$) is (ε_1+ε_2)-DP*

Theorem 3.2 (Parallel Composition [89]). *Let K_1 be an ε_1-DP mechanism and K_2 be an ε_2-DP mechanism. For any dataset $\mathcal{D}$, let D_1 and D_2 be an arbitrary disjoint subsets of the input domain $\mathcal{D}$. Then the mechanism K defined by K($\mathcal{D}$) = ($K_1(D_1)$, $K_2(D_2)$) is max (ε_1, ε_2)-DP*

3.4.2.4 - Sensitivity

As stated earlier, one of the crucial parameters that will influence how precisely we can answer queries is their sensitivity. Below, we revisit the definitions of l_1-sensitivity/global sensitivity (GS), then l_2-sensitivity/global sensitivity.

One way of achieving DP for a query q is to add an appropriate amount of noise to its results, calibrated by the *global sensitivity* (GS) of q. GS measures the maximal variation of the query result when evaluated upon any two neighboring databases. GS depends only on the type of query q, the considered space of databases, and the distance it is associated with (i.e., that identify neighbouring databases). It is independent of the database itself.

Definition 3.2 (l_1-sensitivity / GS [89]). *For a function f : $\mathcal{D} \rightarrow \mathbb{R}$ and all $D_1, D_2 \in \mathcal{D}$, the l_1-sensitivity of f is*

$$\Delta f = GS_f = \max_{D_1,D_2:d(D_1,D_2)=1} \parallel f(D_1) - f(D_2) \parallel_1 \tag{3.2}$$

where $\|\|_1$ denotes the L1 norm.

GS establishes an upper bound on the noise magnitude that must be injected to satisfy DP. Computing the GS for certain functions like *sum*, *count*, and *max* is easy. For example, the GS of the *count* query usually is one because only one record is changed for any two neighboring databases. Since most existing literature focuses on GS, sensitivity refers to GS in this manuscript unless specified otherwise.

Certain mechanisms for achieving DP compute sensitivity using various norms like l_2 distance, referred to as l_2-sensitivity.

Definition 3.3 (l_2-sensitivity [89]). *For a function f : $\mathcal{D} \rightarrow \mathbb{R}^k$ and all $D_1, D_2 \in \mathcal{D}$, the l_2-sensitivity of f is*

$$\Delta_2 f = \max_{D_1,D_2:d(D_1,D_2)=1} \parallel f(D_1) - f(D_2) \parallel_2 \tag{3.3}$$

where $\|\|_2$ denotes the L2 norm.

For queries with low GS, a small magnitude of noise has to be added to respect DP. On the other hand, when the GS is high, a substantial amount of noise must be injected to achieve DP, which will impair data utility. To address this issue, Nissim et al. [102] proposed Local sensitivity (LS), which measures the maximum difference between the query's results on the initial database and any neighborhood of this database.

LS depends on the type of query q, the considered space of databases, the considered distance, but also a database itself. The LS is defined as below.

Definition 3.4 (Local Sensitivity (LS) [89]). *For a function f : $\mathcal{D} \rightarrow \mathbb{R}$ and $D_1 \in \mathcal{D}$, the local sensitivity of f at D_1 is*

$$LS_f(D_1) = \max_{D_2:d(D_1,D_2)=1} \parallel f(D_1) - f(D_2) \parallel_1 \tag{3.4}$$

where $\|\|_1$ denotes the L1 norm.

For many queries, for example, the *median*, the LS is much smaller than the GS. However, LS does not preserve DP guarantees since the noise magnitude itself might reveal information about the database. To address that, Nissim et al. [102] proposed a smooth upper bound on the LS to decide the noise magnitude.

Definition 3.5 (A Smooth Bound [102]). *For β >0, a function S : $\mathcal{D} \rightarrow \mathbb{R}$ is a β-smooth upper bound on the local sensitivity of f if it satisfies the following requirements:*

$$\forall D : S(D) \geq LS_f(D); \forall D, D', d(D, D') = 1 : S(D) \leq e^{\beta} S(D'). \tag{3.5}$$

Smooth sensitivity is an example of a function that fulfills Definition 3.5:

Definition 3.6 (Smooth Sensitivity [102]). *For β >0, the β-smooth sensitivity of function f is*

$$S^*_{f,\beta}(D) = \max_{D'}(LS_f(D').e^{-\beta.d(D,D')}) \tag{3.6}$$

There are numerous other variants of the original concept of GS, encompassing restricted sensitivity [100], empirical sensitivity [103], elastic sensitivity [27], derivative sensitivity [104], etc.

3.4.2.5 - Differential Privacy Mechanisms: Laplace and Gaussian

Any mechanism that adheres to Definition 3.1 can be considered differentially private. Two commonly employed mechanisms for achieving DP for numerical queries (i.e., functions f : $\mathcal{D} \rightarrow \mathbb{R}$) are the Laplace mechanism [13] and the Gaussian mechanism [89]. On the other hand, the Exponential mechanism [89] is employed for non-numeric queries and when the query output makes no sense after adding noise. For instance, setting a price in an auction. The Exponential mechanism is beyond the scope of this manuscript as we will not focus on such queries in our work. Below we explain the Laplace Mechanism and Gaussian Mechanism.

Laplace mechanism, a noise mechanism is proven to preserve ε-DP [13].

Theorem 3.3 (Laplace Mechanism [13]). *The Laplace distribution centered at μ with scale b being the distribution with probability density function*

$$h(x) = \frac{1}{2b} exp(\frac{-|x-\mu|}{b}) \tag{3.7}$$

In the Laplace mechanism, in order to publish f(D) where f: $\mathcal{D} \rightarrow \mathbb{R}$ and D $\in \mathcal{D}$ while satisfying ε-DP, one publishes

$$K(D) = f(D) + Lap(\frac{\Delta f}{\varepsilon}) \tag{3.8}$$

where Lap ($\frac{\Delta f}{\varepsilon}$) represents a random draw from the Laplace distribution centered at $\varnothing$ with scale $\frac{\Delta f}{\varepsilon}$.

Theorem 3.4 (Gaussian Mechanism [89]). *In the Gaussian mechanism, in order to publish f(D) where f :$\mathcal{D} \rightarrow \mathbb{R}$ and D $\in \mathcal{D}$ and σ= $\frac{\Delta_2 f}{\varepsilon}$ $\sqrt{2\ln(1.25/\delta)}$ while satisfying (ε, δ)-DP, one publishes*

$$K(D) = f(D) + \mathcal{N}(0, \sigma^2) \tag{3.9}$$

where $\mathcal{N}$(0, σ^2) is the the normal distribution centered at 0 with variance σ^2. For $\varepsilon \in$ (0,1), the Gaussian mechanism preserves (ε, δ)-DP.

3.4.3 - Local Differential privacy

The centralized model of DP that we have been discussing so far assumes that the data curator is trustworthy. However, this assumption may not always hold true in real life [105]. To address this issue, local differential privacy (LDP) [106] emerges as an alternative approach that does not rely on the presence of any trusted third-party data curator. In the LDP decentralized setting, individuals apply a mechanism to their data before uploading it to an untrusted curator. In fact, we already introduced such a mechanism in Section 3.4.2.2 : the randomized response mechanism [91].

Note that while LDP does not require a trusted curator and can be realized in a distributed way, it can still be applied centraly by such a curator. An example of centralized LDP is proposed in Chapter 7.

LDP was first formalized in [106]. There is an extensive literature on LDP models, e.g. [107–111] to cite just few.

Definition 3.7 (Local DP (Duchi *et al.*) [107]). *Let χ be a set of possible values and Y the set of noisy values. A mechanism $\mathcal{M}$ is ε-locally differentially private (ε-LDP) if for all $x, x' \in \chi^2$ and for all $y \in Y$ we have*

$$Pr[\mathcal{M}(x) = y] \leq e^{\varepsilon} \times Pr[\mathcal{M}(x') = y]$$

LDP-mechanisms outputing a value in a discrete space Y achieve optimal utility for a given ε by giving an answer randomly drawn from a staircase distribution over Y, with the most probable value being the *real* value -whose probability depends solely on $|Y|$ and ε- and all other values being equiprobable [112].

3.4.4 - (d,y)-Privacy

Rastogi et al. [113] introduced a probabilistic privacy definition called (d,γ)-privacy. This definition bounds the difference of the prior probability (before accessing the published data) and the posterior probability (after accessing the published data). It offers a demonstrated promise on both privacy and utility. They indicated that attaining an optimal balance between privacy and utility is possible only when the prior probability is low. However, it provides protection exclusively against attacks that are d-independent; that is where the attacker has knowledge that a particular record is present in the dataset, and obfuscating its presence is not required. Machanavajjhala et al. [73] highlighted that the d-independent hypothesis may not be applicable in certain practical scenarios.

3.4.5 - Distributional privacy

Blum et al. [114] introduced a privacy model inspired from the learning theory, known as distributional privacy for a non interactive query model. The basic concept behind this model is to sample a data table from a distribution; the table should solely disclose information pertaining to the underlying distribution and should not reveal any additional information or detail.

3.5 - Conclusion

Several privacy models within this group do not categorize data table attributes into sensitive and QID attributes in an explicit manner. However, certain privacy models within this group can prevent sensitive linkages presented in the first group, resulting in an overlap between the capabilities of the two groups.

4 - Related work

Contents

4.1 - Introduction

We have presented in the two previous chapters the background concepts and existing works necessary to understand the scientific foundations of this thesis. We now present related works to our research. We start with an overview of existing graph anonymization methods. Indeed, due to its graph structure, we expect graph anonymization techniques to be more adequately suited to RDF than classical tabular data approaches. However, graph anonymization approaches are based on adaptation of tabular database anonymization concepts, such as k-anonymity or DP. We will discuss how different neighborhood concepts define different privacy models and show how this will help us choose a good neighborhood model to use in the RDF context.

We then describe existing works that specifically focus on RDF and linked data anonymization. We describe the results of these approaches and show how they are different or complementary from the approach described in this manuscript.

Finally, we study the application of DP and the neighborhoods definitions in the multi-table relational context.

4.2 - Graph anonymization

To address the aspects and characteristics of LOD, it is pertinent to explore the privacy techniques that are tailored to graph databases because RDF is densely formed on graphs. In particular, we discuss herein privacy mechanisms over social network graphs. Generally, social network data is modeled as a graph that consists of nodes and edges. Nodes represent actors, individuals, or other social entities, and edges represent ties or relationships between nodes. Social network platforms such as Twitter, Facebook, WhatsApp, Instagram, and LinkedIn are notably revolutionizing how people connect, communicate, socialize and engage with each other. The increasing popularity of social networks has sparked a rising interest in collecting social network data. However, given that social network data commonly contain sensitive information related to its users, it is indispensable to guarantee that disclosing such data does not compromise user's privacy. Hence, privacy in social networks has gained increasing interest. It has been surveyed many times [115–117, 117–119], and the recently published survey on privacy-preserving data publishing includes a specific section devoted to graph data [120]. Other recent work on privacy preserving publishing of social network [121] and a survey on private graph data release [122].

Anonymization is the prevailing technique employed in privacy-preserving graph publication (PPGP) to protect the privacy of social network users. From a broad perspective, these PPGP anonymization methods can be generally classified into three categories [121]: Graph modification techniques, Graph Generalization/Clustering techniques, and DP-based graph anonymity techniques. Graph modification techniques modify the graph's structure by adding and/or deleting nodes or edges. In this regard, in the next section, we only focus on k-anonymization techniques in which the model gives anonymity by modifying the graph structure depending on the scenario. The principle of graph generalization/clustering techniques lies in clustering nodes and edges into groups or classes and generalizing the clusters to form super nodes/edges. On the other hand, DP-based graph anonymity techniques usually do not publish the graph as the former two techniques. Instead, DP-based approaches run queries on the graph and release the perturbed output to provide a privacy guarantee. This is done by adding random noise to the query result.

Most of the techniques used for anonymizing social networks are driven by the methods introduced initially for anonymizing relational databases—for example, the methods mentioned in earlier sections, such as [1]. In the following, we do not discuss graph generalization/clustering techniques as they do not directly relate to our work. We first introduce Graph modification techniques, particularly the k-anonymity-based techniques in Section 4.2.1. Then, we present DP models in Section 4.2.2 and DP-based graph anonymity techniques in Section 4.2.3.

4.2.1 - Graph modification: k-anonymization approaches

Several research works proposed implementations or adaptations of k-anonymity to graph databases [115, 123]. The structural characteristics of a graph, particularly the degree of the nodes, can potentially disclose the identities of individuals. In this context, a node can be identified if the attacker has certain prior knowledge of its degree. Liu et al. [123] proposed k-degree anonymity to tackle this concern. A graph is said to be k-degree anonymous if every node has the same degree as at least k-1 other nodes in the graph. This can be attained by computing the minimum number of edge additions (or deletions) that need to be added (or deleted) to obtain a k-degree anonymous graph.

The research paper from Zhou et al. [115] identifies a vital type of privacy attack over social networks: neighborhood attacks. Under this attack, a node can be identified if the attacker has some knowledge about its neighbor nodes. To combat neighborhood attacks, they proposed k-neighborhood anonymity, which adapts k-anonymity to anonymize social networks. To fulfill k-anonymity, nodes with similar neighborhoods must be grouped together. However, they addressed only the one-neighborhoods in their work (i.e., one-hop neighborhoods). While it may be valuable to protect the d-neighborhood where $d > 1$, this raises a significant computational difficulty. Their method is classified as a greedy graph modification strategy because it greedily adds edges to similar neighborhoods to make them identical. Furthermore, the method searches for the similarity between neighborhoods to decrease information loss. Another research work that addresses neighborhood-based attacks on social network is [124]. Nevertheless, there is still a possibility of privacy breaches in k-anonymous social network [115]. Suppose the attacker can identify a specific vertex in a group of anonymized vertices, and all the vertices in this group are linked to some sensitive information. In that case, the attacker can still determine the sensitive attribute of the specified vertex. To address this issue, techniques similar to l-diversity [73] can be employed. Other k-anonymity-based anonymization schemes are k-automorphism [125] and k-isomorphism [126].

K-anonymity-based graph anonymization approaches are relatively vulnerable to modern structure-based de-anonymization attacks [127–135]. This is due to the fact that when research works extend k-anonymity to the graph databases, they rely on classic graph structural properties such as degree, neighborhood, and subgraph. Even if users cannot be differentiated based on particular structural properties, they may still be vulnerable to de-anonymization through alternative properties like path length distribution, closeness centrality, and betweenness centrality. Hypothetically, to achieve indistinguishability among users in terms of all structural semantics is to have a complete or wholly disconnected graph. However, this would result in the total loss of data utility.

4.2.2 - Differential Privacy Models

A social network can be modeled as a graph G = (V, E), where V is a set of nodes, and E is a set of edges. Recall that DP aims at hiding the contribution of an individual, assumed to be a single entry, on the outcome of any analysis. But, what makes a single database entry in graph databases? Which data belongs to a specific individual?

There are at least three primary challenges to be addressed when applying DP to graphs. Since the semantic interpretation of DP lies in the definition of neighboring (adjacent) databases, we must first select this definition of neighboring and comprehend its privacy semantics. Second, we should tackle the problem of high query sensitivity in such complex graphs. Third, there is a trade-off between privacy and utility where stronger privacy guarantees (i.e., "stronger" neighborhood definitions) typically necessitate the injection of more noise to achieve DP, which can impair query outcomes.

Two interpretations of the neighboring definition for social networks and graphs in general have been suggested [136]: node- and edge-DP. Another interesting variant of DP was proposed in [137] in the context of social graphs, called out-link privacy. Since the privacy model relies on neighborhood, each definition provide different kinds of privacy protection.

Definition 4.1 (Node Privacy [137]). *A privatized query Q preserves node privacy if it satisfies DP for all pair of graphs G_1 =(V_1, E_1) and G_2 =(V_2, E_2) such that $V_2 = V_1$-x and $E_2 = E_1$ -{(v_1,v_2) | v_1 = x $\vee$ v_2 = x} for some x $\in V_1$*

In node privacy, a neighboring graph G_2 of a given social network G_1 is the one in which an arbitrary node and all of its adjacent edges are added or removed from G_1. Node-DP attempt to prevent an attacker from asserting the presence or absence of a particular node in the graph (as well as the presence or absence of any incident edge). The privacy guarantee completely protects all nodes as well as their adjacent edges.This imposes a significant restriction on the type of queries that can be computed. A differential private algorithm must hide the worst-case difference between neighboring graphs, which might be significantly considerable under the node privacy model. Indeed, if the number of nodes of the graph is unbounded, two neighbours may differ by an unbounded number of edges. Broadly speaking, it is difficult to achieve high utility under node privacy because of the high query sensitivity. However, node privacy offers desirable privacy guarantees, as stated in [136]. To be mentioned here, in many scenarios or use cases, node privacy can be an excessively stringent requirement that is not needed.

Definition 4.2 (k-edge Privacy [136]). *A privatized query Q preserves k-edge privacy if it satisfies DP for all pair of graphs G_1 =(V_1, E_1) and G_2 =(V_2, E_2) such that $V_1 = V_2$ and $|(E_2 \cup E_1) \backslash (E_2 \cap E_1)| \leq k$*

In k-edge privacy, a neighboring graph G_2 of a given social network G_1 is one in which k arbitrary edges are added or removed from G_1. 1-edge privacy is simply called edge privacy and the most commonly employed in the literature. Edge-DP endeavors to prevent an attacker from infering the presence or absence of a particular edge in the graph. Compared to node privacy, edge privacy is limited to protecting the relationships between nodes. However, nodes having high degrees continue to exert a noticeable impact on query outcomes, although the relationships between these nodes are protected. In many cases, edge privacy can be sufficient, depending on the application. An example of the application of edge privacy can be seen in the work of Kossinets and Watts [138], who utilized edge privacy for safeguarding email relationships.

Definition 4.3 (Out-Link Privacy [137]). *A privatized query Q preserves out-link privacy if it satisfies DP for all pair of graphs G_1 =(V_1, E_1) and G_2 =(V_2, E_2) such that $V_1 = V_2$ and $E_2 = E_1$ -{(v_1,v_2) | v_1 = x} for some x ∈ V_1*

In outlink privacy, a neighboring graph G_2 of a given social network G_1 is one in which an arbitrary node is chosen and all of its out-links (outedges) are added (if the node has no out-links) or removed from G_1. Outlink-DP endeavors to prevent an attacker from disclosing the presence or absence of the out-links (outedges) of a particular node in the graph. The privacy guarantee protects all the out-links of a node. Out-link privacy enhances on edge privacy by decreasing the identifiable properties associated with high degree nodes. A high-degree node can deny that the friendships are mutual in a query output even if others assert being friends with this particular node.

Out-link privacy guarantee is strictly weaker than node privacy, yet for some specific query types, it can provide better performance compared to edge privacy, as indicated in [137]. The authors [137] mentioned that out-link privacy is comparable to edge privacy in many applications. Furthermore, they argue that this model simplifies the calculation of sensitivity and the amount of noise injected to satisfy DP, consequently allowing several queries that would be considered impractical under both node and edge privacy [137].

Throughout this manuscript, we will use **Outedge Privacy** as a synonym for **Out-Link Privacy** because the word "outedge" presents better the considered model.

4.2.3 - Differential Privacy Approaches

A lot of strides have been conducted in the field of differential private publication of social networks. Several algorithms that satisfy the definition have been proposed to release statistics concerning social networks.

A primary challenge in the differential private analysis of social networks lies in the fact that computing the GS for several natural queries can be complex, quite high, or unbounded. For instance, the GS of the median function can be high [102]. Another common example is counting triangles in a graph. The GS of this query under node and edge privacy is unbounded because the change of triangle counting depends on the graph size. In other words, the sensitivity is a function of the graph size [137].

Common tasks in social network analysis encompass degree distribution computation, triangle counting, k-star counting, etc. Degree distribution is one of the most extensively examined graph characteristics. It provides insights about the graph's structure and can be used to delineate the fundamental structure of the graph, construct graph models, and estimate graph similarities. Various techniques have been suggested to obtain differential private outcomes in degree distribution analysis. These techniques include post-processing [101, 136] and projection methods (also known as bounded degree) [97–99, 139], Lipschitz extension [140], and Erdos-Renyi graph [141]. Furthermore, several techniques have been suggested to achieve DP in triangle counting [102, 142–145] or the cut of a graph [144].

Edge privacy has been studied more extensively for providing different graph statistics. Node privacy is a much stronger privacy guarantee, yet it is much more challenging to achieve since it protects against more changes in the input. In the following subsections, we provide a detailed discussion of some of these methods under their specific category, along with other methods.

4.2.3.1 - Node Differential Privacy

In this section, we present projection (also known as bounded degree) methods [97–99, 139], Lipschitz extension [140], and Erdos-Renyi graph [141] for degree distribution analysis. Then, we introduce the notion of restricted sensitivity [100]. We end the section by citing some relevant works under this model unrelated to degree distribution.

Node privacy offers a stronger privacy guarantee than edge privacy, but it is hard to satisfy. One issue is that, in the worst-case scenario, it is challenging to satisfy node privacy for several common statistics while obtaining precise answers. The issue is that node private algorithms must be resilient to the addition of a single node to the graph. However, adding a node with all its edges can notably change the characteristics of a sparse network. For instance, in sparse networks, the alteration can dominate the statistic's value for popular graph statistics, like the number of edges or the frequency of a certain subgraph [97].

Kasiviswanathan et al. [97] proposed several techniques for designing node-DP algorithms and presented a methodology for analyzing the accuracy of such algorithms on realistic networks. The primary concept underlying their techniques is to "project" (in one of several senses) the input graph onto the set of graphs with a maximum degree below a certain threshold. In a bounded degree graph, node privacy is easier to fulfill because adding a single node primarily influences a limited portion of the graph; hence certain query sensitivity can be much smaller on restricted graphs with a particular degree. Additionally, when the degree threshold is cautiously selected, such a transformation loses slightly little information.

The challenge lies in the fact that the projection itself can be highly sensitive to a change of one node in the input graph. They address this challenge by employing two distinct techniques.

The first technique designs tailored projection operators, which have low sensitivity and protect information for specific statistics. Employing these projections, they provide algorithms to release the number of edges in a graph, the counts of small subgraphs such as triangles, k-cycles, and k-stars in a graph, and certain estimators for power law graphs.

In the second technique, they analyze the "naive" projection method, which involves removing high-degree nodes from a graph. They design algorithms that can bound the LS of this projection. Employing this, they develop a generic reduction technique that enables to apply any differentially private algorithm for bounded-degree graphs to an arbitrary graph. They used this to develop algorithms for releasing the entire degree distribution of a graph under node privacy. For the generic reduction part, they consider a projection operator T: $\mathcal{G}_n \rightarrow G_{n,D}$ that takes an arbitrary graph (where $\mathcal{G}_n$ denotes the set of all n-node graphs) and outputs a D-bounded graph, and define the (local, global, smooth) sensitivity of T in terms of the node distance d_{node}(T(G_1), T(G_2)) where G_1 and G_2 differ in one node. Given a query f defined on graphs with degree bound D, they declare that the local sensitivity of the composed query (f ∘ T) is bounded by the product LS_T(G) · $\Delta_D f$ where $\Delta_D f$ is the l_1-global node sensitivity on D-bounded graphs of a function f (discrete analogue of the chain rule from calculus).

If there is a smooth upper bound on the local sensitivity of F_f∘ T and mechanisms that add noise tailored to the smooth upper bound, then a private algorithm for releasing F_f on all graphs in $\mathcal{G}_n$ can be obtained.

Day et al. [98] studied how to publish the degree distribution (or, equivalently, degree histogram) of a graph under node-DP by exploring the projection method to reduce the sensitivity on the projected graph. A primary technique to achieve node-DP is "graph projection," which transforms the graph into a θ-degree-bounded graph, where the maximum degree in the graph is no more than θ. Inspired by graph projection, they [98] proposed an edge-addition based graph projection that keeps as much edges of a given graph as possible. It is the state-of-the-art in this area. The projection algorithm necessitates a stable ordering of all edges within an input graph G, which is represented by Λ(G). This edge ordering must be stable in that given two node neighboring graphs G and G', if two edges appear in G and G', their relative order must be the same in Λ(G) and Λ (G'). The projection first creates a graph with the same nodes as G but with no edges. It then inserts edges from Λ following the ordering whenever inserting the considered edge is possible. That is, do not raise the degree of the associated nodes to be greater than θ. They proved this projection is maximal, implying that adding any further edge would result in the graph no longer being θ-degree-bounded. They [98] proved that the degree histogram over the projected graph has a sensitivity $2\theta + 1$, and the cumulative degree histogram has a sensitivity $\theta + 1$. Based on this sensitivity bound, two approaches, namely (θ, Ω)-Histogram and θ-Cumulative Histogram, were proposed for publishing degree histograms.

The work by Macwan et al. [99] adopt the same edge-addition method as Day et al. [98] to decrease the sensitivity of the node degree histogram. It should be noted that existing projection-based techniques are not satisfactory in providing good utility for continual privacy-preserving releases of graph statistics. To address this limitation, Song et al. [139] proposed a differentially private solution to continually release standard graph statistics such as degree distributions and subgraph counts while considering privacy-accuracy tradeoff. They assumed that there is an upper bound on the maximum degree of the nodes in the whole graph, allowing the release of such graph statistics.

Raskhodnikova et al. [140] suggested an approximation to the degree distribution of a graph under node-DP. The authors employed their Lipschitz extension and the generalized exponential mechanism. They demonstrated that their algorithm surpasses the accuracy of algorithms proposed in former research.

Sealfon et al. [141] introduced a simple and computationally efficient algorithm under node privacy designed to estimate the parameter of an Erdos-Renyi graph. The algorithm provides an optimal estimation of the edge density of any graph with degree distribution concentrated on a small interval.

Blocki et al. [100] presented the notion of restricted sensitivity as an alternative to GS and smooth sensitivity to improve accuracy in differentially private data analysis. The definition of restricted sensitivity [100] is closely similar to GS but is different because it considers any belief about the dataset possessed by the querier. This enables quantifying over a restricted class of datasets rather than considering all possible datasets. In their work [100], they analyzed two classes: subgraph counting queries (e.g., number of triangles) and local profile queries. They showed that the restricted sensitivity of these two kinds of queries are much lower than their smooth sensitivity. Therefore, employing restricted sensitivity can ensure privacy while giving more accurate results.

Other works under node privacy [103,146]. In [103], Chen and Zhou proposed the recursive mechanism. This new differential private mechanism supports unrestricted joins in the context of graph analyses, enabling the release of an approximation of a linear statistic of the outcome of some positive relational algebra calculation on a sensitive database. In [146], the authors utilized the Lipschitz extension method in conjunction with the exponential mechanism to propose node differential private algorithm.

4.2.3.2 - Edge Differential Privacy

In this section, we start by presenting post-processing techniques [101, 136] for degree distribution analysis. Then we briefly present work relying on smooth sensitivity [102] and other work edge-DP mechanisms [142–144].

Hay et al. [101] proposed a post-processing technique to improve the accuracy of existing differentially private algorithms significantly. They then adapted this technique on graphs [136] and presented an efficient DP algorithm to get an estimation of degree distribution.

As discussed earlier, relying on GS is generally inefficient for several queries, such as the number of triangles in a graph, the cost of the minimum spanning tree of a graph, and the median function. To tackle this, Nissim et al. [102] proposed the idea of LS. Unfortunately, LS does not ensure DP since the noise magnitude itself leaks information about the database.

To bridge the gap, the authors [102] proposed a generic framework that permits one to release functions f with instance-based additive noise, meaning that the magnitude of the noise does not depend only on the function being released but also on the database itself. However, a key challenge is to guarantee that the magnitude of noise does not reveal information about the database. To address this, they introduced a smooth upper bound on the LS and calibrate noise proportional to the smooth sensitivity of function f on database x, which measures the variability of f in the neighborhood of instance x. They presented an approach to compute the smooth sensitivity of the number of triangles in a graph and provide the cost of a minimum spanning tree under edge privacy.

Karwa et al. [142] introduced an algorithm that provides approximate answers to subgraph counting queries, including triangle counting, k-star counting, and k-triangle counting. These algorithms fulfill edge privacy and can be considered as an extension of the algorithms proposed in prior work [102] to encompass a broader range of subgraph counting problems while offering privacy guarantees and improved accuracy.

Zhang et al. [143] provided a ladder function and employed it to k-star queries under edge privacy. The work in [144] addresses privacy concerns in network data publication under edge privacy.

4.2.3.3 - Out-Link Differential Privacy

Task et al. [137] proposed out-link privacy, a new standard for DP over social network data. The authors introduced two out-link private algorithms designed for standard network analysis techniques [137] [147]. These network analyses were previously deemed infeasible to realize under existing DP standards (node and edge privacy) due high GS. Specifically, they introduced algorithms that satisfy out-link privacy to release private triangle counting and clustering coefficient information. Moreover, they presented an approach to gather and privatize information pertaining to influential nodes within a network under out-link privacy. In this regard, they presented an algorithm to privatize centrality measures. They also consider degree distribution, particularly the distribution of out-degrees, and compute the sensitivity under out-link privacy.

4.3 - RDF and Linked Data anonymization

The thesis subject is concerned specifically with Linked Data and RDF anonymization. However, the available literature on this topic is notably scarce and extremely limited. Most of the literature addressing the topic appears to be theoretical, whereas we have the ambition to create useable software.

Some models initially developed for relational databases and discussed in the preceding sections have been adapted to the Semantic Web context in recent years [148]. Other works adapt the well-established k-anonymity and its derivative to the RDF setting [149, 150], and other few works adapt DP to the RDF setting.

In line with existing works on the topic, Delanaux et al. [29, 151] introduced a query-based Linked Data Anonymization, and Thouvenot et al. [30, 152] combined k-anonymity with semantic anatomy in the context of the RDF data model.

4.3.1 - k-anonymity-based RDF graph anonymization approaches

Radulovic et al. [149] introduced an anonymization framework for RDF that considers the specific characteristics of RDF specification. The framework aims to protect the privacy of particular entities of interest within RDF graphs. The framework encompasses an anonymization model, various anonymization operations, and an anonymization metric adjusted to align with the features and characteristics of RDF specification. They proposed an anonymization model named k-RDFanonymity. The k-anonymity model introduced by Samarati and Sweeney for tabular data [1] served as a model motivation for this one. Their approach focuses on a subset of resources called entities of interest (EOI). The idea is that a resource representing an EOI can't be distinguished from at least k-1 other resources representing EOI, considering the aspect of the QIDs. They introduced different anonymization operations that can be employed to implement such a model. The operations are generalization, suppression, atomization, and perturbation. In addition, a metric to assess the precision and distortion of the resulting anonymized RDF data is introduced. However, their approach does not include any pseudo code or particular references to implementation details.

K-RDF-Neighborhood anonymity is an approach presented by Heitmann et al. [150]. They stated that the anonymization of heterogeneous graphs is much more complicated than a homogenous graph due to multiple edge and vertex types in the heterogeneous graph. Their work joins and builds upon other works conducted for homogeneous graphs, specifically by Zhou et al. [115], as well as on heterogeneous graphs, such as RDF, specifically by Radulovic et al. [149]. The central point behind their proposal is that the one hop neighborhood of a resource must be indistinguishable from the one hop neighborhood of at least k $-$ 1 other resource. The anonymization algorithm considers only the one hop neighborhood of resources with type foaf:Person. Additionally, their graph modification algorithm deletes edges and does not add edges. They state that this option aligns with the open world assumption, which suggests that a missing statement can be considered true even if it is not present in the dataset. Their algorithm is designed to perform both partial and full anonymization.

4.3.2 - Differential privacy-based RDF graph anonymization approaches

An edge-labeled directed graph is a graph G = (V, E) where V is a set of vertices, E is a set of edges such that E $\subseteq$ V $\times$ L$\times$ V, with L the set of possible edge labels. Let QL be a subset of L.

In the last decade, adapting DP to graphs has received growing attention. However, most efforts have been dedicated to unlabeled, homogeneous graphs –as seen in Sections 4.2.2 and 4.2.3– while labeled graphs with an underlying semantic have seldom been addressed. In fact, to the best of our knowledge, the only works investigating DP in the context of RDF that directly provides experiments are [153, 154].

Reuben [155] studied the adaptation of DP to edge-labeled directed graphs, RDF graph being one of the applications of these graphs. Reuben introduced the notion of QL-edge-label neighboring graphs: graphs that differ by a set of outedges of a node with specific labels. The underlying idea behind this definition is that, for example, in RDF graphs, some relations of an entity may be innocuous and some should be considered sensitive, shown by particular labels. Given this neighboring definition, the author presented **QL-edge-labeled privacy**. It is similar to **Out-Link privacy** –introduced for social networks and explained in Section 4.2.2– but considers edges' semantics by only protecting edges of a given set QL (i.e., *sensitive* labels). Hence it only protects a predetermined subset of outgoing edges.

Let's note an edge-labeled directed graph G = (V, E) where V is a set of vertices, E is a set of edges such that E $\subseteq$ V $\times$ L$\times$ V, and L the set of possible edge labels.

The formal definition is as follow:

Definition 4.4 (QL-edge-labeled Privacy)**.** *A privatized query Q preserves QL-edge-labeled privacy for a set $QL \subseteq L$ if it satisfies DP for all pair of graphs G_1 =(V_1, E_1) and G_2 =(V_2, E_2) such that $V_1 = V_2$ and $E_2 = E_1$-{(v_1,l,v_2) | v_1 = x ∧ l ∈ QL} for some x ∈ V_1*

The author argues that edge labels from a particular domain-specific can 'uniquely' identify a node in such graphs. Reuben considers the outedges of a node since it denotes the contributions made by that node within the graph. This semantically grabs the idea of the presence of an individual in a graph while not being present in another graph, analogous to how private data is modeled as a tuple in the relational model. Reuben's work is considered theoretical. As a future work, Reuben presents a plan to study particular graph statistics using this model.

Throughout this manuscript, we will use **QL-Outedge Privacy** as a synonym for **QL-edge-labeled Privacy** because the word "QL-outedge" presents better the considered model.

In 2017, the authors of [153] studied privacy-preserving statistic queries containing sensitive information about relationships between individuals. They presented an approach to apply ε-DP for RDF data based on node privacy. They consider two datasets D and D' to be neighbors if D' is obtained from D by removing the outgoing and incoming edges of a node. However, no formal definition to define neighboring datasets was provided. Also, they presented an index-like data structure to efficiently compute the actual value and GS for queries with the relationship-based property. However, their work gives a DP realization via LS without the use of a smoothing function hence failing to comply with the privacy guarantees stated in [89].

In 2018, Johnson et al. [27] proposed a novel method to approximate the sensitivity, called elastic sensitivity, which can be applied in a wide range of SQL joins. They demonstrated that elastic sensitivity is an upper bound on LS and thus can be employed to satisfy DP by utilizing any LS-based mechanism.

The emergence of this method has prompted the question of its applicability to RDF and SPARQL. Buil-Aranda et al. [154] introduced the concept of differential privacy schema to enable the redefinition of the sensitivity approximation of SQL queries presented by Johnson et al. [27]. This redefinition is tailored to address SPARQL queries effectively. More precisely, they presented an algorithm that answers count queries over a large class of SPARQL queries while satisfying DP.

The algorithm requires the RDF graph to be supplemented with some semantic information about its structure, which form the differential privacy schema. In particular, authors mentioned that it is essential to determine the various types of entities in the graph and the set of individuals associated with each entity type. In a more formal sense, the objective is to characterize the RDF graph G as a set $\{G_1, . . . , G_n\}$ of sub-graphs. Each subgraph G_i denotes the contribution of an individual, and G= $\uplus_i\ g_i$ is the disjoint union of all these subgraphs. The goal is to protect the contribution of the individuals designated by each G_i.

The authors defined the notion of distance between RDF graphs through direct adaptations from the relational setting, wherein the induced subgraphs serve as table rows. They defined two graphs as being k far apart if one can be obtained from the other by replacing k of its induced sub-graphs.

They implemented the algorithm and carried out various experiments. They showed that their approach is effective over large databases. Nevertheless, the elastic sensitivity of a query that involves join can still be relatively high.

4.3.3 - Query-based Linked Data Anonymization: Utility and privacy policies

Motivated by the logical framework presented by [156], which expresses privacy in a declarative manner using queries that form policies, Delanaux et al. [29] extended this declarative framework by incorporating utility using queries likewise, along with modeling anonymization operations as update queries.

Delanaux et al. [29] proposed a query-based approach to privacy-preserving RDF data publishing. Their declarative framework is based on two crucial concepts: privacy and utility policy. Essentially, a policy is a set of SPARQL queries. The purpose of a policy within the framework pertains to whether it is a privacy policy, which designates the data to be masked in the anonymized RDF graph, or a utility policy, which designates the data to be retained and revealed. As per [156], a privacy policy is deemed to successfully fulfill the anonymization process when no sensitive responses are present when running on an anonymized graph. This objective is attained when all the designated privacy queries either yield no answer or, instead, provide answers containing blank nodes. On the other hand, a utility policy is considered to meet the anonymization process requirements when it ensures maintaining the responses of all of its underlying explicitly specified utility queries.

Their novel framework is declarative as it enables the users to define their privacy and utility prerequisites as policies. Figures 4.1 and 4.2 provide examples of privacy and utility policies, respectively, on public transportation data in a given city. The privacy policy has two SPARQL queries, P1 and P2. Privacy query P1 specifies that the postal addresses of travelers are considered sensitive and thus need protection. P2 expresses that the identifier of a user linked to their geolocation information is a potential risk. The utility policy also has two SPARQL queries, U1 and U2. It specifies that user's ages and location information pertaining to their journeys should be preserved.

```
# Privacy query P1
SELECT ?ad
WHERE {
  ?u  a   tcl:User.
  ?u  vcard:hasAddress  ?ad.
}
```

```
# Privacy query P2
SELECT ?u ?lat ?long
WHERE {
  ?c  a   tcl:Journey.
  ?c  tcl:user        ?u.
  ?c  geo:latitude    ?lat.
  ?c  geo:longitude   ?long.
}
```

Figure 4.1: Privacy policy on public transportation data

.

They addressed the compatibility between privacy and utility policy prior to tackling the generation of anonymization operations. There are a lot of cases in which anonymization becomes infeasible due to the considerable overlap between the policies, resulting in cases where one desires to "keep what you desire to hide" or vice versa. They demonstrated that the compatibility between the two policies is established when, for any privacy query, there exists at least one RDF triple pattern in the body of this query that is not unifiable with any triple of

```
# Utility query U1                # Utility query U2
SELECT ?u ?age                    SELECT ?c ?lat ?long
WHERE {                           WHERE {
  ?u  a   tcl:User.                 ?c  a   tcl:Journey.
  ?u  foaf:age  ?age.               ?c  geo:latitude   ?lat.
}                                   ?c  geo:longitude  ?long.
                                  }
```

Figure 4.2: Utility policy on public transportation data

.

a utility query.

They designed two algorithms that take as input the privacy and utility policy and generate as output a set of anonymization operations satisfying the privacy policy while preserving the utility policy. Their first algorithm is designed for a unitary privacy policy containing a single query. Subsequently, the second algorithm extends the first to the general global anonymization process. The generated anonymization operations are in the form of SPARQL update queries of DELETE type (deletion of triples) and DELETE/INSERT (triples update).

The basis of the algorithm is to traverse the set of all triples of the queries of the privacy policy and examine the feasibility of unifying each triple with those in the utility queries of the utility policy. A triple cannot be deleted if is unifiable with one in a utility query as doing so would contradict the utility policy. On the other hand, if there is no conflict with the utility queries, meaning that the considered triple is absent from all the utility queries, then this particular triple becomes eligible for possible deletion.

The operations employed by the algorithm involve the deletion of triples and possible update operations. Update operations are presented in two ways: either by substituting the subject of the triple with a blank node or by replacing the object with a blank node in the case the object is an IRI. In either case, the algorithm explores three possible options:

The triple is included in a path of length ≥ 2, and consequently, the update operation cuts the path, thereby fulfilling the privacy policy.

The substituted subject appears as a subject or object of another triple in the privacy query, respectively the replaced object appears as the subject or the object of another triple, and the update operation cuts the bond between these triples, fulfilling the privacy policy.

The substituted subject of the triple, respectively, the substituted object, is included in the distinguished variables. This results in a blank value in the query results, effectively fulfilling the privacy policy.

In Fig. 4.3 and 4.4, we provide an illustrative example of these operations related to transportation data, taken from [29].

The anonymization process is noteworthy as it is flexible and can be customized to the user's specific requirements. This concept was briefly mentioned in the context of k-RDF Neighborhood Anonymity [150], where a "percentage of anonymization" could be specified. In such cases, the algorithm would partially traverse the graph without allowing for the targeting

```
DELETE   { ?u  vcard:hasAddress   ?ad. }
WHERE    { ?u  a tcl:User.
             ?u  vcard:hasAddress   ?ad.}
```

Figure 4.3: Deleting the addresses of users

.

```
DELETE   { ?c  tcl:user    ?u. }
INSERT   { ?c  tcl:user    []. }
WHERE    { ?c  a tcl:Journey.
             ?c  tcl:user         ?u.
             ?c  geo:latitude     ?lat.
             ?c  geo:longitude    ?long. }
```

Figure 4.4: Substituting users identifiers related to a transport ticket validation by a blank node

.

of specific portions.

Their approach is distinguished as data-independent and solely focuses on checking the privacy and utility policies to produce the sequence of anonymization operations. Therefore, performance is independent of the size of the graph being anonymized. Instead, it is solely affected by the size of queries that need to be analyzed.

It is essential to consider an additional factor: including a new dataset in the LOD could risk privacy breaches since this new dataset can be potentially linked to existing objects in other LOD datasets. The paper [151] continues the previous work presented in [29], and concentrates on the challenge of constructing safe anonymizations of an RDF graph. The objective is to ensure that the union between the anonymized graph and any external RDF graph does not compromise privacy.

By considering a set of privacy queries as input, they investigated the data-independent safety problem and the precise required sequence of anonymization operations needed to ensure its enforcement. They gave adequate conditions to ensure the safety of an anonymization instance, considering the provided set of privacy queries. Moreover, they demonstrated the robustness of their proposed algorithms for RDF data anonymization when confronted with the existence of sameAs links in the data. These links can be either explicit, indicated by the presence of sameAs triples, or implicit, deduced from other triples, inferred by additional knowledge. Similar to the first approach [29], the proposed anonymization algorithms generate anonymization operations as SPARQL update queries. The operations employed by the algorithm come with an assurance that their application to any RDF graph results in compliance with the privacy policy. Once again, the presented algorithms analyze the triples contained in the privacy policy to generate the update queries. The performance is independent of the size of the graph. The anonymization method replaces IRIs and literals with blank nodes.

The work of Asghar et al. [31] extends the framework proposed in [29] to formalize privacy and utility policies as temporal aggregate conjunctive queries. They introduced a framework for a formal specification and verification of compatibility between privacy and utility policies. Their framework is characterized as data-independent and is adequate for assisting data producers in maintaining control over safeguarding their data in various real-world scenarios, such as collecting of sensitive data by mobile personal devices or smart environments and sharing these data over the internet. Compared to the approach of Delanaux et al. [29, 151], the work of the latter is limited to handle simple conjunctive queries. They do not take into account aggregates and do not consider temporal data.

4.3.4 - Combining k-anonymity and semantic anatomy for knowledge graph anonymization

Thouvenot et al. [30] adapted the anatomization approach originally used in the relational data model to the context of the RDF data model. In the context of knowledge graphs (or RDF graphs), anatomization involves breaking the relationships between the QIDs and their SAs by introducing in-between blank nodes between entities (and their QIDs) and their SAs. Instead of generalizing or suppressing entity QIDs, anatomization alters the graph's structure by introducing additional nodes and edges. The retaining of the original QID values without any transformation maintains the correlation and consequently facilitates a high-quality data analysis of the published anonymized data.

They presented a detailed example in Fig. 4.5 demonstrating the religion aspect of the Lehigh University Benchmark (LUBM) extension. Fig. 4.5(a) shows a portion of the original initial graph depicting three people and certain corresponding QIDs. Fig. 4.5(b) shows the anatomized version of this particular graph portion. As shown, the QID remains unchanged and the EIDs (John and Jane's URIs) have been substituted with blank nodes. The "hasReligion" property no longer refers directly to a religion concept but points to a sub-graph group employed to break sensitive relationships. Additionally, many properties have been introduced, namely inGroup, value, and cardinality. The value and cardinality properties allow the retention of useful information about SAs, namely which values appear in a group and how many times each value does.

The goal of [30] is to prevent revealing new information individuals. For doing so, they proposed semantic anatomization, an anonymization technique that concentrates on grouping of sensitive attributes while considering their semantic context. They presented a semantic anatomization algorithm to achieve this task. Their approach is distinguished as semantic because it exploits a domain ontology accompanied with the graph. Precisely, grouping of SAs according to their closeness in an underlying ontology. By this, semantic anatomization ensures utility improvement in the published datasets compared to standard anatomization. Their semantic-based approach allows to utilize l-diversity to measure the degree of privacy preservation. Their algorithm guarantees a minimum diversity level with a value of $l=2$.

Building upon [30], they enhance this utility-centered framework by integrating the widely popular k-anonymity in their follow-up work [152] to boost privacy. Their work considers the privacy utility trade-off in the context of RDF data model. They presented two algorithms, denoted global and group. The global algorithm in which k-anonymity is run on top of the entire anatomized dataset, whereas the group algorithm where k-anonymity is run separately on each group formed by anatomization.

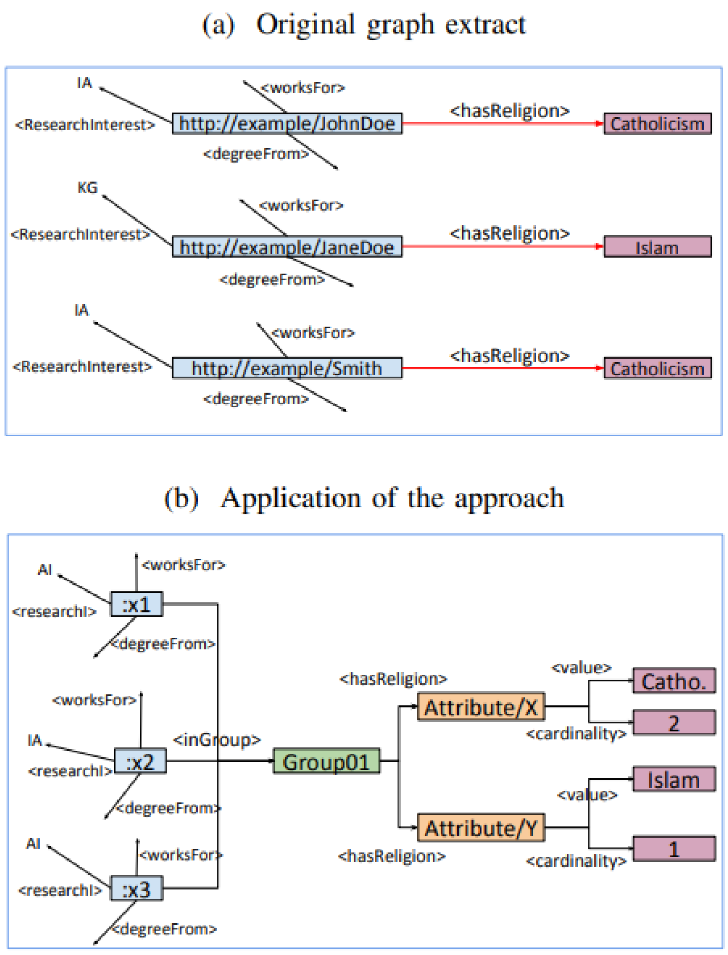


Figure 4.5: Anatomization: breaking the relationships between EoI and SA

.

Their evaluation highlights that applying both anatomy and k-anonymity maintains the utility characteristics of the former work [30] besides improving the privacy of the published data sets.

However, it is still challenging to determine the optimal value of k for achieving k-anonymity. Evidently, there is no universal approach in anonymization solutions, and it is crucial to have a thorough understanding of the data within original datasets so that it is possible to publish safe, private data.

Nevertheless, in realistic scenarios, the data is always exposed to modifications through insertions or deletions. Addressing anonymization within a dynamic context is challenging as it introduces potentials to various attacks that standard anonymization techniques (e.g., k-anonymity [12], l-diversity [73], or t-closeness [85]) are not suitable to address. This is the subject of the follow-up work of Thouvenot et al. [157], in which they proposed Kgastor (KG anonymized store), an anonymization framework specifically intended to be integrated directly with a RDF database management system to anonymize knowledge graphs.

By employing a partitioning strategy in conjunction with an access control solution and an anonymization strategy tailored to the dynamic context, they were capable of handling knowledge graphs along with protecting the personal data belonging to the entities it encompasses.

The partitioning strategy partitions the original graph into two distinct sub-graphs: private and default graph. The private graph contains non-anonymized QID values and is reserved only for privileged users whereas the default graph is an anonymized graph accessible to all users. The joining of the two graphs is guaranteed through the introducing of blank nodes. For the anonymization strategy, they adapted the state-of-the-art m-invariance [158], initially developed for the relational data model and capable of handling both insertions and deletions.

4.4 - Differential Privacy over Multi-Relational Databases

In the DP literature [89], 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 [27, 159–161].

The notion of DP [14] is defined based on the concept of neighboring databases. Accordingly, one should define this notion in the context of multi-relational database to construct appropriate DP mechanisms. Two DP policies have been introduced in the relational setting based on whether foreign key (FK) constraints are considered while defining neighboring databases or not.

4.4.1 - Neighboring databases: One-row neighbors

PINQ [159] and FLEX [27] 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.

4.4.2 - Beyond one-row neighbors: multi-relations with constraints

PrivateSQL system [160] introduces a richer notion of neighboring databases that considers constraints in the schema, in particular primary and FK constraints. When we delete 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.

The formal DP definition in databases with FKs is as follow [160]:

Definition 4.5 (Neighborhood with FK)**.** *Consider a database schema **R**. One relation is designated as the primary private relation R_p, and any relation having a direct or indirect FK referencing R_p is called a secondary private relation. Assuming **I** a database instance over **R**. For any R $\in$ **R**, we define **I**(R) as the relation instance of R in **I**. The referencing relationship over the tuples is defined as follows:*

*For t $\in$ **I**(R) and t$'$ $\in$ **I**(R$'$), we say that t$'$ reference t if (1) R$'$ references R and the FK of t$'$ equals to the PK of t; or (2) $\exists$ another t$''$ such that t$'$ references t$''$ and t$''$ references t.*

*Two database instances **I** and **I'** are neighbors if **I'** can be obtained from **I** by deleting some tuple t $\in$ **I**(R) and all tuples referencing t.*

Starting from [160], researchers begin to consider FK constraints when defining neighboring databases [162, 163]. The privacy guarantees provided by these two DP policies cannot be compared. While the DP policy that takes into account FK constraints provides a stronger privacy guarantee for the primary private relation, it just supports that particular relation. On the contrary, the DP policy that ignores FK constraints offers relationship-level protection for multiple relations. If the goal is to protect the entities and information about the entity accessible via FK constraints, then the former is more suitable. If the relationships, for instance, friendship, or seller-buyer relationships, are sensitive, then it is more appropriate to opt for the latter.

4.4.3 - DP Systems for Multi-Relational Databases

Several DP systems for multi-relational databases has been proposed. We present briefly some systems from the literature.

PINQ. In 2009, McSherry proposed Privacy Integrated Queries (PINQ) [159], a platform built on top of LINQ declarative query language. It gives a DP guarantee for counting queries expressed in an augmented SQL dialect. PINQ supports only a restricted join operator that enables linking unique records. This join operator corresponds to the usual meaning for one-to-one joins. On the contrary, for many-to-many and one-to-many joins, a statistical count query does not count the number of joined results but rather, it counts the number of unique join keys.

FLEX. Johnson et al. [27] proposed a new notion of sensitivity called elastic sensitivity that supports general equijoins besides the whole range of join relationships. By utilizing only precomputed database metrics and the query, elastic sensitivity will be computed effectively. They verified that elastic sensitivity is an upper bound on local sensitivity and can be used with any local sensitivity-based mechanism to provide DP. They presented an end-to-end system called Flex that uses elastic sensitivity to enforce DP for queries expressed in standard SQL.

PrivateSQL. PrivateSQL [160] adjusts DP to multi-relational schemas with constraints. It enables differentially private SQL query answering and supports only count queries. It supports the expression of complex privacy policies. PrivateSQL generates private synopses of many views over the base table to allow queries to be answered under a fixed privacy budget. It utilizes a diversity of new techniques, such as truncation, policy-aware rewriting, and constraint-oblivious sensitivity analysis, to guarantee high accuracy.

GoogleDP. Differentially private SQL with bounded user contributions was proposed in [161]. It addresses a particular sort of privacy policy called user-level DP for multi-relational databases with constraints. It supposes that a single individual (stated as a user) can contribute to multiple rows in the underlying database. They proposed a generic method to bound user contribution to enforce DP to a rich class of aggregate SQL functions.

4.5 - Conclusion

In this Chapter, we have presented works that we identified as closest to our scientific problematic. To the best of our knowledge, we have shown that the only work investigating DP in the context of RDF databases that directly provides experiments is by [154]. This work presents an algorithm that answers count queries over a large class of SPARQL queries while satisfying DP. Furthermore, the proposed algorithm requires the RDF graph to be supplemented with some additional semantic information about its structure.

Existing results of the use of DP in graph or multi-relation databases will be used as inspiration for some of our work, in particular, how privacy models are impacted by neighborhood definitions.

5 - Using projection to improve Differential Privacy on RDF graphs

Contents

5.1 - Motivation and approach

We want to adapt DP to edge-labeled directed graphs with an underlying semantic, typically RDF graphs. This demands a complete and extensive analysis of many aspects. First, formalize the distance(s) over such graphs to define neighboring graphs. Subsequently, establish the privacy semantics associated with each specific distance. Then, to provide an intuition on usability, calculate the sensitivity of various queries over the RDF graphs w.r.t the various defined distances under the specific privacy model. This leverages two challenges. First, check if the sensitivity of the query is high and determine whether directly employing a Laplacian mechanism to perturb the query results is a good option or will destroy utility. Secondly, under the hypothesis that the latter is true, study or analyze how to construct DP-mechanisms over RDF graphs while keeping data utility as desired.

Building upon this, we can state the aim of this chapter as follows: We are interested in adapting DP to edge-labeled directed graphs with an underlying semantic. We want to propose methods to query RDF graphs while respecting DP constraints and contribute to designing, implementing, and evaluating algorithms that guarantee privacy while preserving data utility.

Sketch of our approach

Given these requirements and the challenges mentioned above, we explain in detail the contributions below in the following sections.

- We consider three privacy definitions: node, outedge and QL-outedge privacy. We formalize three distances over RDF graphs to define the corresponding notions of neighborhood. (Section 5.2);

- We present a new approach based on graph projection to adapt DP to RDF graphs while reducing the sensitivity of many queries (Section 5.3). In detail, we introduce three edge-addition based graph projection methods that transform the original RDF graph into a graph of bounded degree, bounded out-degree, and bounded typed-out-degree (QL-out-degree) (Section 5.3.1). We demonstrate that two of the proposed projections preserve neighborhoods, allowing to expand the domain of any differentially private algorithm from graphs with bounded out- or typed-out-degree to any arbitrary RDF graph (Section 5.3.4);

- We evaluate our approach analytically and experimentally w.r.t. a real Twitter use-case (Section 5.4), showing significant improvement over a naive approach without projection.

To the best of our knowledge, as outlined in Chapter 4, our work is the first to provide algorithms to query RDF graphs while reducing the amplitude of the randomized noise and respecting DP constraints.

A prototype has been implemented that performs the proposed projections and apply DP mechanisms to various queries.

Notation	Definition
d_n (node)	$\lvert V\rvert$ where $V = \{v \in V_1 \cup V_2 \lvert v \notin V_1 \cap V_2 \vee$ $(\exists e \in (E_1 \cup E_2) \setminus (E_1 \cap E_2), \exists l \in L, \exists u \in V_1 \cap V_2$: $e = (v, l, u) \vee e = (u, l, v))$
d_O (outedge)	∞ if $V_1 \neq V_2$ else $\lvert V\rvert$ where $V = \{v \in V_1 \lvert \exists l \in L_1, \exists u \in V_1 :$ $(v, l, u) \in (E_1 \cup E_2) \setminus (E_1 \cap E_2)\}$
d_{QL} (QL-outedge)	∞ if $V_1 \neq V_2 \vee \exists (u, v) \in (V_1 \cap V_2)^2, \exists l \in L \setminus QL$ such that $(u, l, v) \in (E_1 \cup E_2) \setminus (E_1 \cap E_2)$ else $\lvert V\rvert$ where $V = \{v \in V_1 \lvert \exists l \in QL, \exists u \in V_1 :$ $(v, l, u) \in (E_1 \cup E_2) \setminus (E_1 \cap E_2)\}$

Table 5.1: Distances: notations and definitions

5.2 - Models and distances for DP on Graphs

The privacy model is tightly related to a distance on the considered database space when using DP. In chapter 2, we present three DP models defined over social graphs, namely: node privacy, edge privacy, and outedge privacy (refer to Section 4.2.2), and the QL-outedge privacy model defined over edge-labeled directed graphs (refer to Section 4.3.2). Now, we present the meaning of these models over RDF graphs.

Node Privacy defines neighboring graphs as graphs that differ by one node and all its incident edges. In the case of RDF, an incident edge represents a triple involving the node as a subject or object. In RDF, node-DP therefore protects the presence/absence of node (i.e., an IRI or a blank) and protect the contribution of all the triple it is the subject or object of.

In **Edge Privacy**, neighboring graphs differ by an edge. The model therefore protects the contribution of a single edge. In RDF, this means protecting a single triple.

The adaptation of **Outedge Privacy** [137] to the context of edge-labeled directed graphs is straightforward. This privacy model protects *all* the outedges of a node. In the context of RDF, this means protecting all the triples a node is the subject of.

QL-Outedge Privacy [155] is similar to outedge privacy but considers edges' semantics by only protecting the edges whose labels belong to a given set QL (i.e., *sensitive* labels). In RDF, this means protecting a subset of the triples a node is the subject of: those whose predicate is deemed sensitive.

Notations. An edge-labeled directed graph is a graph G = (V, E) where V is a set of vertices, E is a set of edges such that E $\subseteq$ V $\times$ L$\times$ V, with L the set of possible edge labels. We note $\mathcal{G}$ the set of such graphs.

Formal definitions for the considered distances (related to node, outedge, and QL-outedge privacy) between two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are reported in Table 5.1.

5.3 - Proposed Approach: from a subspace with low-sensitivity queries to $\mathcal{G}$

The main challenge when developing node-DP, outedge-DP, and QL-outedge-DP algorithms is that the sensitivity of many queries can be very high, or even unbounded, in $\mathcal{G}$. Consider the three following motivating queries:

Motivating Examples

Q1: Compute the maximum degree in a graph.

Q2: Compute the maximum out-degree in a graph.

Q3: Compute the maximum typed-out-degree in a graph. In other words, compute the maximum out-degree of a specific label(s).

For all three privacy models, the GS of the three queries is in terms of the number of nodes in the graph, which is unbounded. For these three queries, any naive DP mechanism tuned with an infinite sensitivity will produce random noise, destroying the utility regardless of the used metric. Technically, many queries may have much lower sensitivity when considered in a space of graphs with bounded degree, out-degree, or QL-out-degree.

Therefore, the main idea behind our approach is *graph projection* in order to transform the original graph G into a graph of bounded degree, out-degree, or QL-out-degree. We show that such projections can significantly reduce the sensitivity of a query and, consequently, the magnitude of the noise added to achieve DP. This reduction may compensate the data loss inherent to these projections, allowing them to improve utility ultimately.

In this section, we first introduce projection methods. We then introduce the necessary notations and concepts to study DP for queries within the projected space. Finally, we show how to make the whole mechanism (i.e., a projection followed by a query over the projected space) differentially private.

5.3.1 - Proposed Projection Methods

For projections to be adequate w.r.t. the three considered privacy models, we propose three edge-addition based graph projection methods named T_n, T_O and T_{QL}. Projection by edge-addition was introduced by [98] for unlabeled, undirected graphs and is herein expanded to edge-labeled directed graphs.

Notations. We note $\mathcal{G}^D$ the set of graphs with maximum degree D and $\mathcal{G}_o^D$ the set of graphs with maximum out-degree D. Finally, we note $\mathcal{G}_{QL}^D$ the set of graphs with maximum QL-out-degree D for a given QL $\subseteq L$; i.e. the set of graphs whose vertices are the source of at most D edges whose labels are in QL. Note that $\mathcal{G}_{QL}^D \subseteq \mathcal{G}_o^D \subset \mathcal{G}^D \subset \mathcal{G}$.

We notice that using the bound corresponding to the privacy model seems favorable for many queries: sensitivities are bounded considering node privacy, outedge privacy and QL-outedge privacy on degree, out-degree and QL-out-degree bounded graphs. Refer back to the motivating examples presented in the previous section, with D the bound (on degree, outdegree, or QL-out-degree).

Motivating Examples

One may initially intuit that the kind of bounds to be used should depend on the query. The three motivating examples are in relation to the three kind of bounds that are proposed. This deliberate choice allows to stress that, in fact, **the optimal kind of bound does not depend on the query but instead on the privacy model**. In what follows, we will show that the kind of bound related to each query is in fact sub-optimal if it does not correlate with the considered privacy model.

Q1: Compute the maximum degree in a graph.

The GS of this query under node privacy on $\mathcal{G}^D$ is D.
The GS of Q_1 under node privacy on $\mathcal{G}_O^D$ and $\mathcal{G}_{QL}^D$ is still ∞. For example, a graph with $|V|-1$ nodes of degree 1, pointing toward the $|V|$th node would be out- and QL-out-degree bound. Yet, removing the $|V|$th node would make the result of the query go from $|V|-1$ to 0.
Under out-edge and QL-out-edge privacy, the GS of Q_1 on $\mathcal{G}^D$ is not immediate, and equal $D + min(D, |L|)$ and $D + min(D, |QL|)$, respectively. Indeed, in $\mathcal{G}$ two neighbours in such models vary by at most D edges. An edge can contribute to the degree of a node by 0,1, or 2. Since E is a set (and not a bag), two edges with the same sources and the same destination must have different labels. Therefore, a node may not be both the source and destination of more than $|L|$ edges, contributing to 2 to its degree. Under out-edge and QL-out-edge privacy, the GS of Q1 is D on $\mathcal{G}_O^D$ and $\mathcal{G}_{QL}^D$ respectively. Under outedge privacy, the sensitivity is ∞ on $\mathcal{G}_{QL}^D$.

Q2: Compute the maximum out-degree in a graph.

The GS of this query under outedge privacy on $\mathcal{G}_O^D$ is D. While this would still be true on $\mathcal{G}^D$, the projection on this space may lead to unnecessary edge deletion and information loss. On the contrary, the GS is ∞ on $\mathcal{G}_{QL}^D$ (since the out-degree may be arbitrarily high considering edges considered non-sensitive).
Under QL-outedge privacy, while the GS of $Q2$ is D on $\mathcal{G}^D$, $\mathcal{G}_O^D$, and $\mathcal{G}_{QL}^D$, the projection on $\mathcal{G}_{QL}^D$ is the one that will lead to less deletion and that will provide the best result. Under node-privacy, the GS of $Q2$ is D on $\mathcal{G}^D$ and $\mathcal{G}_O^D$. It is ∞ on $\mathcal{G}_{QL}^D$.

Q3: Compute the maximum typed-out-degree in a graph.

The GS of this query under Ql-outedge privacy on $\mathcal{G}_{QL}^D$ is D. Similarly, this remains true with the other two kinds of bounds but projecting considering those may lead to unnecessary data loss.

The sensitivity of Q1, Q2, and Q3 is low (since D can be arbitrarily low) and bounded on (degree, out-degree, QL-out-degree) bounded graphs, respectively. In what follows, we present the projection algorithms.

Projection algorithms Projection methods T_n: $\mathcal{G} \rightarrow \mathcal{G}^D$, T_O: $\mathcal{G} \rightarrow \mathcal{G}^D_o$, $T_{QL} : \mathcal{G} \rightarrow \mathcal{G}^D_{QL}$ are described in Algorithm 1, 2, and 3, respectively. They transform the original graph G into one of its sub-graphs $\tilde{G}$, such that the maximum degree, out-degree, or QL-out-degree of a node in $\tilde{G}$ is less than or equal to D.

First, the projection creates a graph with the same nodes as G but without any edges. It then tries to insert each edge of G following an *edge ordering function* –noted A– that takes a graph and outputs an ordered lists of its edges. An edge $e = (v_1, \ell, v_2)$ is successfully inserted whenever its insertion preserves the constraint, i.e. for T_n (resp. T$_O$, T$_{QL}$) inserting this edge will not raise the ***degree*** **of either** v_1 **or** v_2 (resp. the ***out-degree*** **of** v_1, the ***QL-out-degree*** **of** v_1) over D.

Algorithm 1: T_n : projection by edge-addition, Bound Degree

```
   Input: A graph G = (V, E) ∈ $\mathcal{G}$, a bound D, a stable edge ordering A
   Output: An output D-degree bounded graph
1  $\tilde{E} \leftarrow \varnothing$;
2  foreach $v \in V$ do toBound(v)←0;
3  foreach e=($v_1$,$l$,$v_2$) ∈ A(G) and following A's order do
4      if toBound($v_1$) < D ∧ toBound($v_2$) < D then
5          $\tilde{E} \leftarrow \tilde{E} \cup$ {e};
6          toBound($v_1$)++;
7          toBound($v_2$)++;
8      end if
9  end foreach
10 return $G^D$ = (V, $\tilde{E}$)
```

Algorithm 2: T_O : projection by edge-addition, Bound Out-degree

```
   Input: A graph G = (V, E) ∈ $\mathcal{G}$, a bound D, a stable edge ordering A
   Output: An output D-out-degree bounded graph $T_O$(G)
1  $\tilde{E} \leftarrow \varnothing$;
2  foreach $v \in V$ do toBound(v)←0;
3  foreach e=($v_1$,$l$,$v_2$) ∈ A(G) and following A's order do
4      if toBound($v_1$) < D then
5          $\tilde{E} \leftarrow \tilde{E} \cup$ {e};
6          toBound($v_1$)++;
7      end if
8  end foreach
9  return $G^D_o$ = (V, $\tilde{E}$)
```

Edge ordering As seen above, the algorithm attempts to insert the edge in some predetermined order. Using a different order may produce a different result. This edge ordering must be stable in the sense that given two neighboring graphs, G_1 and G_2, if two edges appear in G_1 and G_2, then their relative order must be the same in A(G_1) and A(G_2). We can construct a stable edge ordering quite easily. Indeed, as E $\subseteq$ V $\times$ L$\times$ V, a first intuition is to consider

Algorithm 3: T_{QL} : projection by edge-addition, Bound QL-out-degree

```
Input: A graph G = (V, E) ∈ $\mathcal{G}$, a bound D, a stable edge ordering A, a set of
       labels QL ⊆ L
Output: An output D-Ql-out-degree bounded graph $T_{QL}$(G)
1  $\tilde{E} \leftarrow \varnothing$;
2  foreach $v \in V$ do toBound(v)←0;
3  foreach e=($v_1$,$l$,$v_2$) ∈ A(G) and following A's order do
4      if $l \in$ QL ∧ toBound($v_1$) < D then
5          $\tilde{E} \leftarrow \tilde{E} \cup$ {e};
6          toBound($v_1$)++;
7      end if
8      if $l \notin$ QL then $\tilde{E} \leftarrow \tilde{E} \cup$ {e} ;
9  end foreach
10 return $G^D_{QL}$ = (V, $\tilde{E}$)
```

orders on the space of sources, labels, and destination (e.g., lexicographical order) and to define a total edge order by combining the three.

Many orders may be considered in this context. For instance, we can decide that we want to order according to alphabetical order, lexicographical order, priority label order (user-defined priority of certain labels over others), etc. Once we decide the order on each space, we choose one of the six possible combinations (S-L-D, S-D-L, L-S-D,L-D-S,D-S-L, D-L-S). For example, we choose the lexicographical order over each space and S-L-D (sources, labels, and then destination). Edges are ordered according to the lexicographical order of their sources. If both sources are the same, they are ordered w.r.t. the lexicographical order of their labels. If both labels are the same, they are ordered w.r.t. the lexicographical order of their destination. Note that if their destination is the same, then the two edges are, in fact, equal. We provide many different orders with detailed explanations in our experimentation part in Section 5.4.2.1.

5.3.2 - Privacy on Bounded (degree, Out-degree, Ql-out-degree) Graphs

Note that most concepts related to DP depend on the considered space of databases and its associated distance. Since we consider in this paper various subspaces of $\mathcal{G}$ and various distances, this subsection introduces unambiguous notations and definitions.

Definition 5.1 (Restricted ε, δ-differential Privacy)**.** *A randomized mechanism K: $\mathcal{G} \to S$ is $(\varepsilon, \delta)_R$ differentially private over $R \subseteq \mathcal{G}$ w.r.t. a distance d over R, if for all pairs (G_1, G_2) $\in R^2$,*

$$d(G_1, G_2) = 1 \implies Pr[K(G_1) \in S] \leq e^{\varepsilon} Pr[K(G_2) \in S] + \delta$$

By definition, the GS of a query is the maximum variation of its results over any neighboring graphs in the considered space.

Definition 5.2 (Global sensitivity on Bounded Graphs)**.** *For any f : $\mathcal{G} \to \mathbb{R}^k$, the global sensitivity of f on $R \subseteq \mathcal{G}$, w.r.t a distance d over R is:*

$$\Delta_d^R f = \max_{(G_1,G_2)\in R^2 : d(G_1,G_2)=1} \parallel f(G_1) - f(G_2) \parallel_1 \tag{5.1}$$

By convention, $\Delta_d^{\mathcal{G}}$ is noted Δ_d. The sensitivity over the projected spaces can simply be seen as the sensitivity of the restriction of the original function to the projected spaces. *Considering the definitions it is trivial that for any f, R, and d, $\Delta_d^R f \leq \Delta_d f$.*

5.3.3 - Privacy and projections

These notations having been introduced, we study in what follows the privacy guarantees of the mechanism composed by a projection followed by a query. Its sensitivity depends on the sensitivity of the projection, i.e., the maximal distance between any two neighboring graphs after projection.

Definition 5.3 (Global sensitivity of a projection [97]). *The global sensitivity of a projection T: $\mathcal{G} \to R$ w.r.t. a distance d over $\mathcal{G}$ and d_R over R is:*

$$\Delta_{(d,d_R)} T = \max_{(G_1,G_2)\in \mathcal{G}^2 : d(G_1,G_2)=1} d_R(T(G_1), T(G_2)) \tag{5.2}$$

In what follows, we assume that the same distance d is used in $\mathcal{G}$ and R, and note $\Delta_d T$ instead of $\Delta_{(d,d)} T$. The sensitivity of the composed function $f \circ T$ is bounded by the sensitivity of T times the sensitivity of f on the projected space:

Theorem 5.1 (Sensitivity of the composed mechanism [97]). *Given a projection $T : \mathcal{G} \to R \subseteq \mathcal{G}$, a function $f : R \to \mathbb{R}^k$, and a distance d over $\mathcal{G}$:*

$$\Delta_d(f \circ T) \leq \Delta_d^R f \times \Delta_d T \tag{5.3}$$

5.3.4 - Privacy on unbounded graphs through projection

Our context involves three privacy models (node, outedge, and QL-outedge) related to three distances and three projections introduced in Sections 5.2 and 5.3.1, respectively. In principle, one would want to study the global sensitivity of each projection w.r.t. all distances. However, there is an obvious relation between projection and distances, and we believe that a preliminary study may be restricted to (i) $\Delta_{d_n} T_n$, (ii) $\Delta_{d_O} T_O$ (iii) $\Delta_{d_{QL}} T_{QL}$. Moreover, in this manuscript, we focus on (ii) and (iii) as they seem more interesting to us in the context of RDF.

Lemma 5.1 (Global sensitivity of T_O). $\Delta_{d_O} T_O = 1$

Intuitively, by removing or adding all the out-edges of some node v, v is the only impacted node in term of out-degree. Since the constraint of T_O concerns out-degree, and since the edge ordering is stable, outedges of other nodes are handled in the same way by T_O, the sole difference between the two projected graphs being the out-edges of v. Therefore, the projected graphs are still neighbors w.r.t. d_O.

Lemma 5.2 (Global sensitivity of T_{QL}). $\Delta_{d_{QL}} T_{QL} = 1$

The proof is similar to Lemma 5.1 since in the worst case scenario, outedge and QL-outedge are identical.

Hence, the global sensitivity of T_O and T_{QL} w.r.t. their related distance (d_O and d_{QL}, respectively), is 1. Therefore:

- They preserve neighborhood, i.e., the projection of two neighboring graphs through the use of T_O and T_{QL} results in two neighboring graphs (w.r.t. d_O and d_{QL}, respectively).
- According to Thm. 5.1, for any function f, the global sensitivity of the composed mechanism is no greater than the global sensitivity of f over the projected space (w.r.t. d_O and d_{QL}, respectively).

It directly follows that any algorithm DP on $\mathcal{G}_o^D$, or $\mathcal{G}_{QL}^D$ can be transformed into an algorithm DP on $\mathcal{G}$ without any extra privacy budget (i.e., while preserving ε).

Proposition 5.1. *Given any mechanism f whose domain is $\mathcal{G}_o^D$ (resp. $\mathcal{G}_{QL}^D$), if f is ε-DP w.r.t. d_O (resp. d_{QL}) then $f \circ T_O$ (resp. $f \circ T_{QL}$) is ε-DP on $\mathcal{G}$ w.r.t. d_O (resp. d_{QL}).*

Proof is immediate considering that T_O and T_{QL} preserve neighborhood according to Lemmas 5.1 and 5.2.

5.4 - Analytical and Experimental Evaluation

This section is dedicated to the evaluation and discussion of our proposal. We design several metrics dedicated to the evaluation of the expected utility loss due to DP, the information loss due to projection, and the overall interest of projection, in particular when compared to a naive approach. We demonstrate experimentally the usability of the approach and show how projection can be used in practice to answer SPARQL queries with DP guarantees with reference to a realistic use case. Before introducing the practical evaluation settings, we discuss hereafter the evaluation itself: the used metrics and their meaning.

5.4.1 - Metrics: Utility and Information Loss

This section introduces metrics to analytically evaluate the approach and confront them to a real use case. Analytical expectations are experimentally confirmed, demonstrating the feasibility and interest of the approach.

Fig. 5.1 provides an overview of the functions and values considered during our evaluation w.r.t. a query Q, a projection T, and a distance d. To simplify, we consider that $Q : \mathcal{G} \to \mathbb{R}$ and the laplacian mechanism is used to achieve DP.

We are interested in particular in evaluating: (1) the overall utility loss due to privacy, comparing $\bar{qn}$ and q; (2) the information loss due to projection, by comparing q and $\bar{q}$; (3) the

$$\begin{tikzcd}
\text{G} \in \mathcal{G} \arrow[r, "\text{Q}"'] \arrow[d, "\text{T}"] & \text{q} \arrow[r, "+\text{Lap}(\frac{\Delta_d Q}{\varepsilon})"'] & \text{qn} \\
\bar{G} \in R \arrow[r, "\text{Q}"] & \bar{q} \arrow[r, "+\text{Lap}(\frac{\Delta_d^R Q}{\varepsilon})"] & \bar{qn}
\end{tikzcd}$$

Figure 5.1: Available data for evaluation : overview

interest of projection w.r.t. providing a DP answer without projection, by comparing qn and $\bar{qn}$.

For (1), we have this metric from literature:

$$RE = \frac{|\bar{qn} - q|}{q} \tag{5.4}$$

Since $\bar{qn}$ is not deterministic, an experimental evaluation of *RE* should be made over several runs where some value would be averaged. Various values to average could be considered, e.g. $\bar{q}_n$, which would in fact average to $\bar{q}$. We argue that the most meaningful metric is the average or expected difference between q and $\bar{qn}$ which we formalize as E as follows:

- Expected utility loss E, which is the expected difference between $\bar{qn}$ and q. E $= \int_0^\infty xG(x)\,dx$ with $G(x)$ the probability of answering $\bar{qn}$ such as $|\bar{qn} - q| = x$:

$$\begin{aligned} E = \int_0^\infty x(\frac{1}{2b}exp(\frac{-|q-x-\bar{q}|}{b}) + \frac{1}{2b}exp(\frac{-|q+x-\bar{q}|}{b}))dx \\ = b * exp(\frac{-(q-\bar{q})}{b}) + q - \bar{q} \end{aligned} \tag{5.5}$$

  with b $= \frac{\Delta_d^R}{\varepsilon}$

For (2), we propose, two metrics, one generic from the literature and one tailored to our use case:

- Preserved Edge Ratio (PER) [98], which is the ratio of number of edges preserved in D-bounded graph (in our case whether it's D-out-degree or D-QL-out-degree bounded graph) to the number of edges in the original graph. For a given original graph G = (V, E) and projected graph G^D = (V, E^D), it is defined as

$$PER = \frac{|E^D|}{|E|} \tag{5.6}$$

- Information loss due to projection, defined as

$$Ploss = \frac{|q - \bar{q}|}{q} \tag{5.7}$$

Finally, regarding interest of projection (3), note that if $\Delta_d Q = \infty$, we can either consider that DP is impossible on the original space, or that qn should be considered pure noise (uniformly random over $\mathbb{R}$), which makes the projection's interest immediate. In what follows, in this context, we also compare the approach with a naive one consisting in using restricted DP without projection. When $\Delta_d Q$ is finite, we propose the following metric:

- Projection interest for queries with bounded global sensitivity in $\mathcal{G}$, defined as

$$PI = \frac{|qn - \bar{qn}|}{qn} \tag{5.8}$$

5.4.2 - Implementation and evaluation settings

Dataset The dataset used is the Sentiment140 dataset with 1.6 million tweets[1], which we have parsed and serialized in RDF/XML format. Its schema is shown in Fig. 5.2.

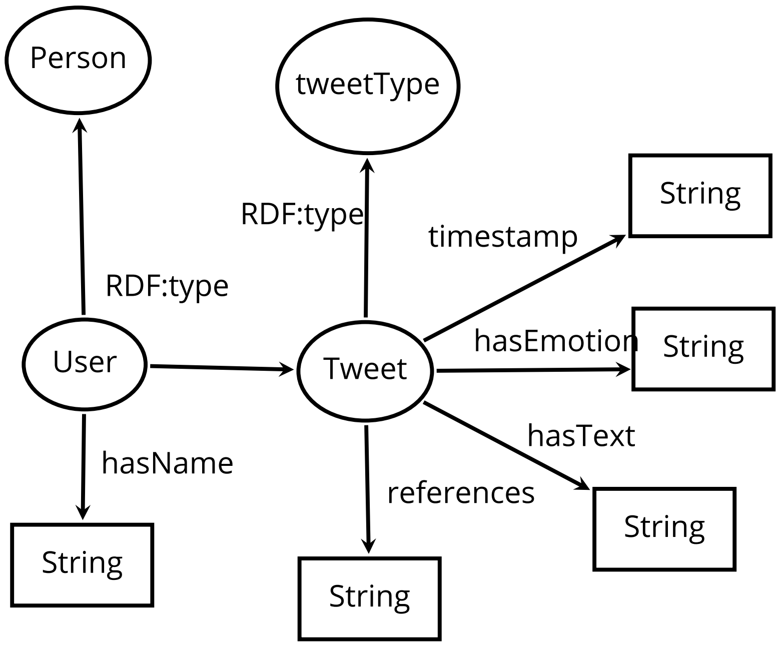


Figure 5.2: RDF schema for Sentiment140

5.4.2.1 - Projections and edge orderings

We propose an implementation of each projection: T_n, T_O, and T_{QL} using Java 1.8. Each projection may be performed w.r.t. several edge ordering to investigate the effect of edge ordering on utility. The implemented edge ordering are detailed hereafter.

Code for the implementation of our projection algorithms and the different edge ordering is available at :`https://github.com/saratakì/dp-projection-queries`

Due to the large size of the dataset used, and consequently the projection algorithms requirements to load the dataset and then perform the projection, we run the queries on a server Intel Xeon Gold 5215 2.5GHz of 64 GB RAM.

[1]`https://www.kaggle.com/kazanova/sentiment140`

Edge ordering We implement many *stable* orders to study the effect of edge ordering on utility.

Order S-D-L: We choose the lexicographical order over each space and consider sources, destinations and then labels. If both sources are the same, they are ordered w.r.t. the lexicographical order of their destinations. If both destinations are the same, they are ordered w.r.t. the lexicographical order of their labels. Note that if their labels are also the same, then the two edges are, in fact, equal.

Order S-L-D: We choose the lexicographical order over each space and consider sources, labels, and then destinations.

Order PriorityLabel"label1": We consider some label ("label1") to have priority, and use the L-S-D order if a decision can be taken w.r.t. the prioritized label. Considering two edges:

- If exactly one of the labels is "label1", we prioritize the edge with this label at the beginning of the ordering.
- If both labels are "label1", edges are ordered w.r.t. the L-S-D order.
- If neither labels are "label1", edges are ordered according to the L-S-D order.

Order PriorityLabels"label1,label2": A label ("label1") has priority over all other labels, while a second label ("label2") has priority over all labels that are not "label1". When a decision cannot be taken using this rule, we use the L-S-D order.

- If both or no labels are "label1" or "label2", edges are ordered w.r.t. the L-S-D order.
- If one label is "label1" and the other is "label2", we give priority to the edge with "label1" to be at the beginning of the ordering.
- If exactly one of the labels is "label1" or "label2", we prioritize the edge with this label at the beginning of the ordering.

5.4.2.2 - Implemented Queries

We study and implement numerous SPARQL queries under Outedge Privacy with T_O and QL-Outedge Privacy with T_{QL}. Code for the numerous queries we run using Apache Jena is available at at :`https://github.com/sarataki/dp-projection-queries`.

However, in this manuscript, we only select and show five queries due to space restriction:

QA : compute maximum node out-degree.

QB and QC : compute maximum label-specific-out-degree, i.e., the maximum number of edge with a certain label a node is the source of. QB consider "tweeted" edges and QC "references" edges.

QD : count how many users tweeted more than 25 tweets.

QE : count the number of users "Garythetwit" has referenced.

For each query we study the impact of:

- Out degree bounds under Outedge Pivacy with T_O, QL-Out-degree bounds and QL label(s) under QL-Outedge Privacy with T_{QL}.
- Edge ordering, whether it is important or not and the choice of the order.
- ε values, set to 0.01, 0.1, 0.5, 1, 1.5 and 10.

5.4.2.3 - Laplacian Mechanism

To achieve DP with the aforementioned queries, we implemented the laplacian mechanism.

Lap Noise without projection . If the GS of query Q on $\mathcal{G}$ is bounded under Outedge privacy and/or QL-Outedge privacy, then it is possible to achieve DP on $\mathcal{G}$ using a laplacian mechanism.

In this context, we average the noise over 100 runs. More precisely, we sample the laplacian distribution with the relevant parameters 100 times. Every time, we calculate the absolute value of the distance from q to q +Lap, Lap being the result of the sample, denoted by |q-(q+Lap)| |Lap|. We get 100 distances. Then, we sum up these 100 distances to obtain a single value, which we name dis. After that, we compute the avg distance which is defined as avg distance=$\frac{dis}{100}$. Finally, we add the avg distance to q to obtain an estimate of the expectancy of the value of qn. This, in turn can be used to compute the expectancy of *PI*.

Lap Noise with projection . Noise is drawn from Lap($\frac{\Delta^{\mathcal{G}_o^D}_{d_O} Q}{\varepsilon}$) under Outedge privacy with T_O and from Lap($\frac{\Delta^{\mathcal{G}_{QL}^D}_{d_{QL}} Q}{\varepsilon}$) under QL-Outedge privacy with T_{QL}.

We draw LaplaceNoise 100 times. Every time, we calculate the absolute value of the distance from $\bar{q}$ to $|\bar{q}$ +Lap$|$ denoted by $|\bar{q}$-$|\bar{q}$+Lap$||$. We get 100 distances. Then, we sum up these 100 distances to obtain a single value, which we name dis. After that, we compute the avg distance which is defined as avg distance=$\frac{dis}{100}$. Finally, we add the avg distance to $\bar{q}$ to obtain an estimate of the expectancy of $\bar{qn}$, which allows us to approximate the expectancy of *PI*.

5.4.3 - Experimental evaluation

After projection, we run the query on the projected graph to get $\bar{q}$. Then, noise is added to $\bar{q}$, as explained in Section 5.4.2.3. At the end, we obtain $\bar{qn}$ values corresponding to each bound D with all considered ε's.

Used metrics: For QA we use PER, E and Ploss. For the rest of queries, we use E and Ploss. For QD exclusively, we use PI.

5.4.3.1 -QA- Compute maximum out-degree

On the original dataset, QA outputs 551, q = 551.

Interest of the approach. It is immediate that $\Delta_{d_O} QA$ and $\Delta_{d_{QL}} QA$ (given that QL $\neq \emptyset$) is infinite. Thus, it would not be possible to use a laplacian mechanism on the original query without reducing its sensitivity.

Let us first consider T_O. $\Delta^{\mathcal{G}^D_o}_{d_O} QA = D$, i.e., the sensitivity of QA w.r.t. d_O restricted to the space of graphs with maximal out-degree D is D.

Edge ordering is not important because we intend to compute the maximum node out-degree and T_O bound out-degree of nodes. The nodes of the projected graph will have the same out-degrees regardless of the chosen edge ordering. We consider one case: case 1_O. We take the *Order S-D-L* and make the out-degree bound vary from 2 to 560. We select these bounds because QA outputs 551 on the original graph, so we pick the maximum D to be 560, slightly greater than 551. Clearly, when $D > 551$, the projection has no effect. Such a situation would therefore be similar to a naive restricted DP mechanism (i.e., a DP mechanism on a particular subset of $\mathcal{G}$).

We report in Fig. 5.3 the analytical value of E. Ploss and PER are reported in Table 5.2.

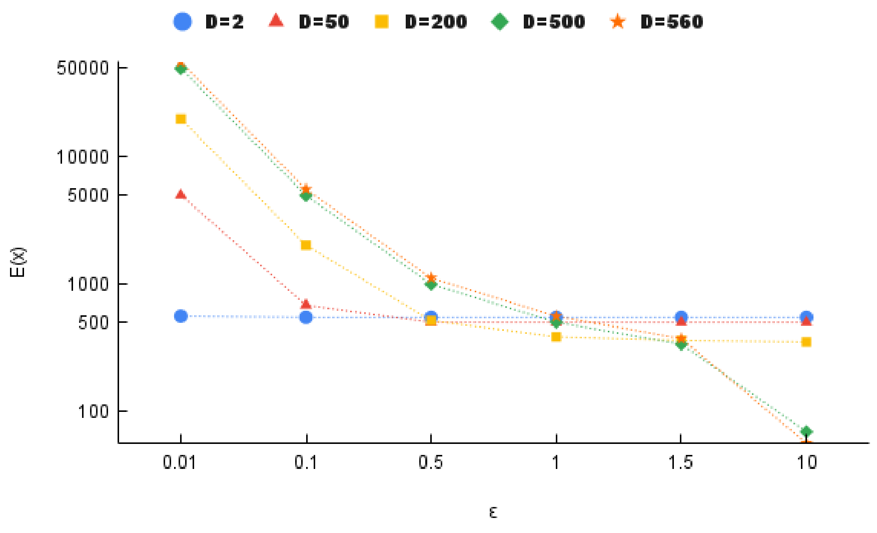


Figure 5.3: E for QA case 1_O

Degree bound D	2	50	200	500	560
Ploss	0.99	0.9	0.63	0.09	0
PER	0.446	0.997	0.999	0.999	1

Table 5.2: Ploss and PER for QA case 1_O

Regarding T_{QL}, $\Delta^{\mathcal{G}^D_{QL}}_{d_{QL}} QA$=D (provided that QL $\neq \emptyset$) i.e., the sensitivity of QA w.r.t. d_{QL} restricted to the space of graphs with maximal QL-out-degree D is D.

Edge ordering is not important also because we intend to compute the maximum node out-degree and T_{QL} bound QL-out-degree of nodes. Regardless of the edge order chosen, nodes of the projected graph will have exactly the same number of non-sensitive outedges as before projection and either the same number of sensitive outedges or D if they had more than D sensitive outedges. Consider three cases. For all the cases, we take the *Order S-D-L*.

1. Case 1_{QL} where QL={tweeted}. QB outputs 549 on the original graph, i.e. the maximum number of tweets tweeted by a single person on the original graph is 549. Therefore, we choose the Ql-out-degree bounds to go from 2 to 560. We report in Fig. 5.4 the analytical value of E. Ploss and PER are reported in Table 5.3.

2. Case 2_{QL} where QL={references}. We choose the Ql-out-degree bounds to go from 2 to 13. We select this last bound to since the maximum references outdegree of a node is 12.

3. Case 3_{QL} where QL={tweeted, references}. We choose the Ql-out-degree bounds to go from 2 to 560. We select these bounds to be compatible with the chosen bounds in the case 1_{QL} and case 2_{QL}. As also, references and tweeted are not outedges of the same node. We report in Fig. 5.6 the analytical value of E. Ploss and PER are reported in Table 5.5.

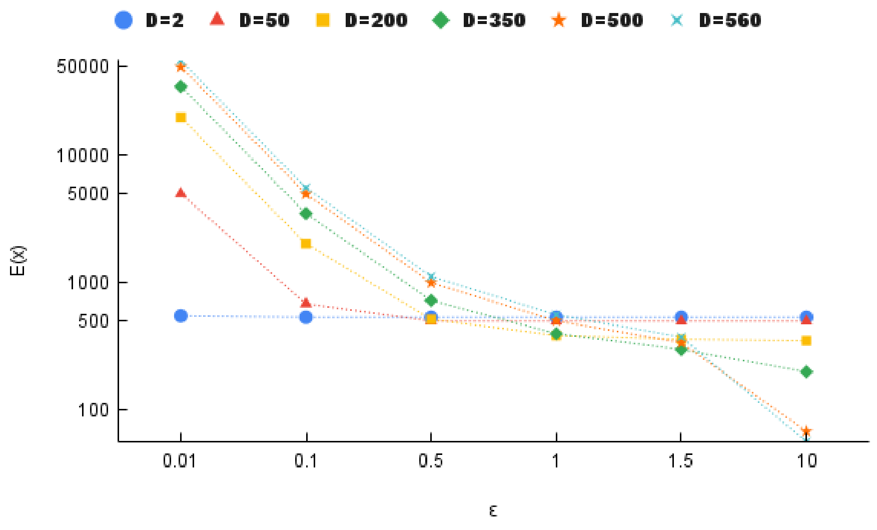


Figure 5.4: E for QA case 1_{QL}

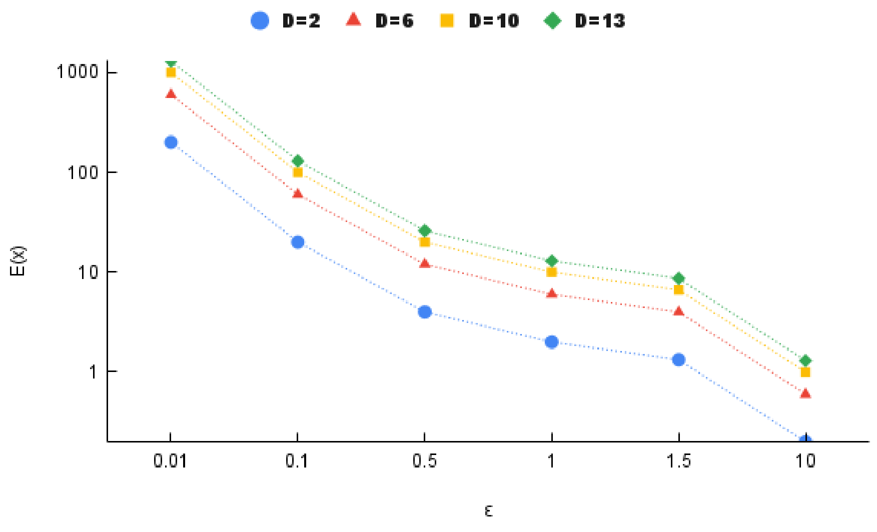


Figure 5.5: E for QA case 2_{QL}

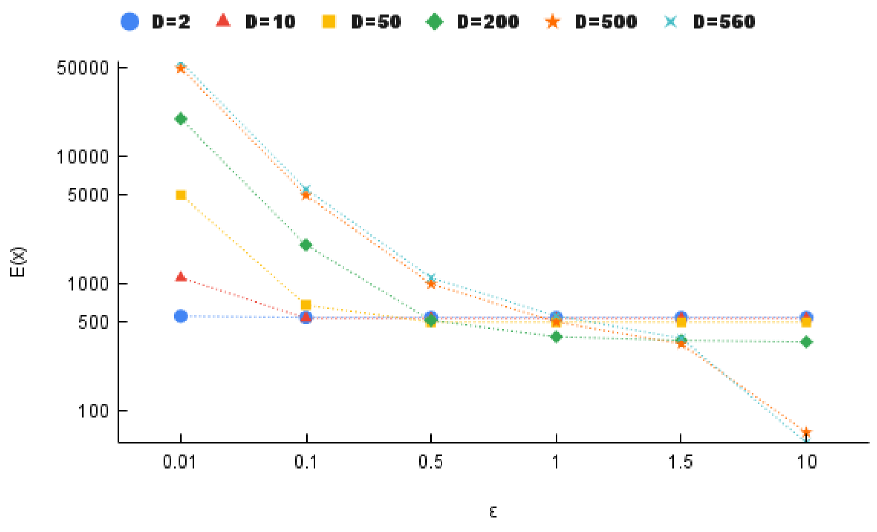


Figure 5.6: E for QA case 3_{QL}

Degree bound D	2	50	200	350	500	560
Ploss	0.97	0.9	0.63	0.36	0.08	0
PER	0.967	0.997	0.999	0.99	0.99	1

Table 5.3: Ploss and PER for QA case 1_{QL}

Degree bound D	2	6	10	13
Ploss	0	0	0	0
PER	0.998	0.999	0.999	1

Table 5.4: Ploss and PER for QA case 2_{QL}

Degree bound D	2	10	50	200	500	560
Ploss	0.98	0.97	0.9	0.63	0.08	0
PER	0.930	0.979	0.997	0.999	0.999	1

Table 5.5: Ploss and PER for QA case 3_{QL}

Discussion We start by discussing E values. In case 1_O, as the out-degree bound increases, $\bar{q}$ will be closer to q. This is obvious because QA computes max node out-degree and T_O bounds out-degree. We have b= $\frac{\Delta^O}{\varepsilon}$; with the same out-degree bound D, as ε increases, b decreases. Therefore, E decreases. With D = 560 > 551, q = $\bar{q}$, then E = b.

In case 1_{QL}, case 2_{QL}, and case 3_{QL}, QB outputs 549 on the original graph. According to our schema, tweeted outedges contribute significantly to the maximum out-degree in the graph. We have b= $\frac{\Delta^{QL}}{\varepsilon}$; with the same out-degree bound D, as ε increases, b decreases. Thus, E decreases.

In case 1_{QL}, we are bounding tweeted outedges of a node. Therefore, as Ql-out-degree bound increases, $\bar{q}$ will be closer to q (with D=500, $\bar{q}$=502). With D = 560 > 549, q = $\bar{q}$, then E = b.

In case 2_{QL}, bounding references do not affect the answer of QA on the projected graph. With all Ql-out-degree bounds, we get $\bar{q}$ = q. Therefore, E = b.

For case 3_{QL}, we are bounding tweeted and references outedges of nodes. As Ql-out-degree bound increases, $\bar{q}$ will be closer to q (with D=500, $\bar{q}$=502). With D= 560 > 549, q = $\bar{q}$, then E = b.

We start first by discussing E results. For cases 1_O, 1_{QL}, and 3_{QL}, With D<200, E decreases while ε increases until it reaches a constant. With D >= 200, as expected, E decreases while ε increases: utility increases as privacy guarantees weaken. More interestingly, E decreases with D, meaning that *increase of information loss due to a tighter bound is compensated by the decrease in the amplitude of noise added to obtain DP guarantees*, for small values of ε.

As we can notice for both cases 1_{QL} and 3_{QL}, with ε=1, E values are very close with a maximum difference <178, i.e., $\frac{E(x)withD=560}{E(x)withD=200}$ is roughly 1.46. This means that ε=1 is the point where things start to shift and bigger bounds become more interesting than lower bounds: the information loss is no longer compensated by the reduction of the amplitude of added noise.

Interestingly, for cases 1_O, 1_{QL}, and 3_{QL}, $D = 560$ is an extremal case where the graph is not modified during projection. We have also seen that $\Delta_{d_O}Q = \infty$ and $\Delta_{d_{QL}}Q = \infty$ meaning that no DP mechanism can be trivially constructed over $\mathcal{G}$. A straightforward –but somewhat weak– approach would be to construct a restricted DP mechanism over some subspace of $\mathcal{G}$, typically $\mathcal{G}_o^D$ or $\mathcal{G}_{QL}^D$ with $D = 560$. This would provide exactly the same results as our approach with $D = 560$, which provides utility -roughly- ten times worse than a regular parametrization of our approach with a bound $\leq$ 500. For example, for case 1_{QL} with $\varepsilon = 0.1$, $\frac{E(x)withD=560}{E(x)withD=50}$ is roughly 8.19, meaning that the expected distance between the private answer and the real value is 8.19 times greater with bound D = 560 than 50.

Regarding Ploss, in case 1_O, bounding node out-degree results in Ploss values that are slightly greater compared to case 1_{QL} and case 2_{QL}. This shows that bounding node out-degree, unsurprisingly, leads to more loss of information than bounding specific QL-out-degree. On the other hand, case 1_{QL} where QL={tweeted} produces slightly better results in terms of Ploss as compared with case 3_{QL} where QL={tweeted, references}. This is due to the database schema.

Therefore, the choice of QL is important in contributing to the max out-degree. Bounding references, as in case 2_{QL} where QL={references}, don't affect the query results. As we can see in this case, with all Ql-out-degree bounds, we get $\bar{q}$ = q. This is mainly due to the database schema, as the node with high out-degree has mostly *tweeted* typed edges (QA outputs 551 on the original graph, and QB outputs 549).

5.4.3.2 -QB- Compute maximum label specific (tweeted) out-degree

On the original dataset, QB outputs 549, q=549.

Interest of the approach. It is immediate that $\Delta_{d_O} QB$ and $\Delta_{d_{QL}} QB$ (given that tweeted $\in$ QL) are infinite. Thus, in this case, it would not be possible to construct a DP mechanism directly from the original query without reducing its sensitivity.

Let us first consider T_O. $\Delta_{d_O}^{\mathcal{G}_o^D} Q = D$, i.e., the sensitivity of QB w.r.t. d_O restricted to the space of graphs with maximal out-degree D is D.

Edge ordering is important because we intend to compute maximum tweeted out-degree. We are concentrating on edges with specific labels "tweeted" and according to our schema nodes with "tweeted" outedges have other outedges.

We consider two orders:

1. Case 1_O. Choose the *Order S-D-L*. We report in Fig. 5.7 the analytical value of E.
2. Case 2_O. Choose the *Order S-L-D*. We report in Fig. 5.8 the analytical value of E. Ploss for both cases are reported in Table 5.6.

Regarding T_{QL}, if tweeted $\in$ QL, then $\Delta_{d_{QL}}^{\mathcal{G}_{QL}^D} QB$=D , i.e., the sensitivity of QB w.r.t. d_{QL} restricted to the space of graphs with maximal QL-out-degree D is D. Note that if tweeted $\notin$ QL then $\Delta_{d_{QL}}^{\mathcal{G}_{QL}^D} QB$=0. We consider two cases in which tweeted $\in$ QL.

1. Case 1_{QL} where QL={tweeted}. Edge ordering is not important because we are interested in computing the maximum tweeted out-degree and we are bounding solely the tweeted outedges. Choose the Ql-out-degree bounds to go from 2 to 560. We select these bounds because QB outputs 549 on the original graph and we are bounding "tweeted" outedges here.

   Regarding E, we obtain the same results as in case 1_O. We mean the same $\bar{q}$ values by the same results, leading to the same E values. (we take the same bounds D in both cases). Refer to Fig. 5.7, which shows the analytical value of E. Ploss is also the same as in case 1_O. Refer to Table 5.6.

2. Case 2_{QL} where QL={tweeted, name}. Edge ordering matters because, according to our schema, tweeted outedges play an important role in contributing to the maximum out-degree. Both tweeted and name are outedges of the same node. Take the *Order S-L-D.*

   Regarding E, we obtain the same results as in case 2_O. By same results, we mean same $\bar{q}$ values, leading to same E values. Refer to Fig. 5.8, which shows the analytical value of E. Ploss is also the same as in case 2_O. Refer to Table 5.6.

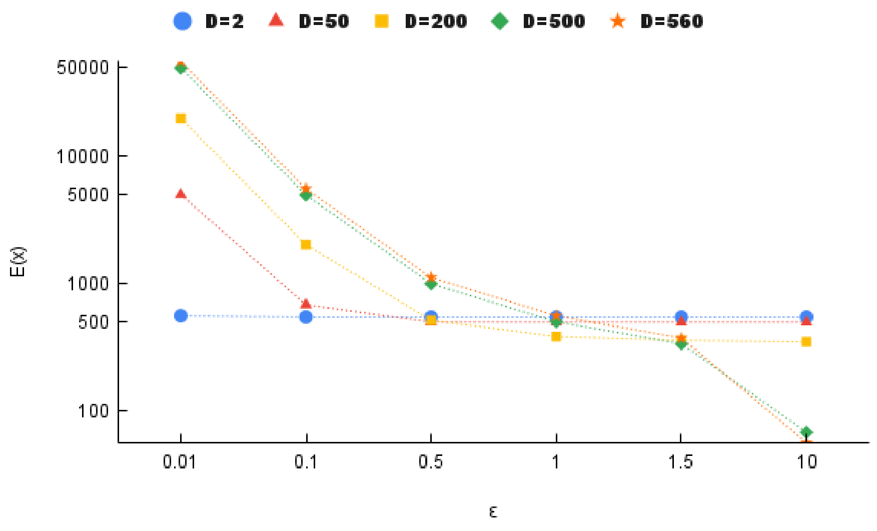


Figure 5.7: E for QB case 1_O

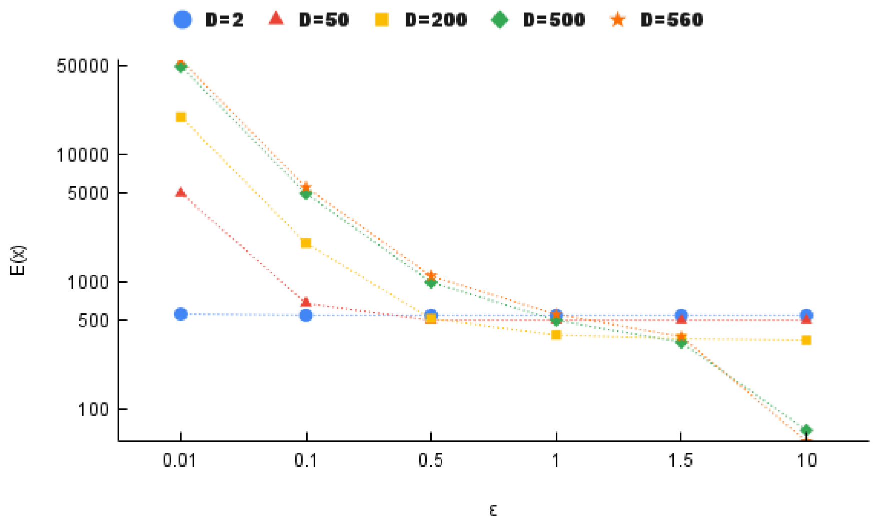


Figure 5.8: E for QB case 2_O

Discussion We first start by discussing E values. In case 1_O and case 2_O and according to our schema, nodes with tweeted outedges have two other outedges. With D = 2, 50, 200,

Degree bound D	2	50	200	500	560
Ploss case 1_O	0.99	0.90	0.63	0.08	0
Ploss case 2_O	0.99	0.91	0.63	0.09	0

Table 5.6: Ploss for QB case 1_O and case 2_O

500, Order S-D-L preserves D-tweeted outedges, whereas Order S-L-D preserves D-1 tweeted outedges. Therefore, Order S-D-L gives better results than Order S-L-D (although the results are quite close). With these bounds, E values in case 1_O are slightly smaller than E values in case 2_O (it is negligible when comparing Fig. 5.7 and 5.8). With D = 560, $\bar{q}$ = q with both orders. So, E = b.

The optimal order is *Order S-D-L* under T_O. We should note here that the $\bar{q}$ values for the two chosen orders are quite close because nodes with tweeted outedges have two other outedges. So, if D > 2, in the worst case scenario (worst order), will preserve the two non-tweeted outedges with D-2 tweeted.

In case 1_{QL}, we obtain the same results as in case 1_O. This is expected because with D = 2, 50, 200, 500, case 1_O preserves D-QL outedges and under this case where QL={tweeted} we are preserving D-tweeted outedges. With D = 560, $\bar{q}$ = q. Consequently, E = b. In case 2_{QL}, we obtain the same results as in Case 2_O. This projection preserves D-1 tweeted.

If our objective is to protect the tweeted outedges of a node, then it is sufficient to bound tweeted out-degree (rather than bounding the out-degree) to get the optimal results for Q_B. We are here under QL-outedge privacy with T_{QL}. On the contrary, if we want to protect all outedges of a node, then it is necessary to bound the out-degree and search for the best order. In our scenario, it is case 1_O (we might find other optimal order(s)). We are here under outedge privacy with T_O.

Regarding Ploss results, Ploss values are quite close when comparing case 1_O and case 2_O.

5.4.3.3 -QC- Compute maximum label specific (references) out-degree

This query can also be written as compute the maximum number of references referenced by a single tweet.

On the original dataset, QC outputs 12, q = 12.

Interest of the approach. It is immediate that $\Delta_{d_O} QC$ and $\Delta_{d_{QL}} QC$ (given that tweeted $\in$ QL) are infinite.

Let us first consider T_O. $\Delta_{d_O}^{\mathcal{G}_o^D} QC = D$, i.e., the sensitivity of QC w.r.t. d_O restricted to the space of graphs with maximal out-degree D is D.

Edge ordering is important because we intend to compute maximum references out-degree. We are concentrating on edges with specifc labels "references", and according to our schema nodes with "references" outedges have other outedges.

Choose the out-degree bounds to go from 2 to 16. We select these bounds because QC outputs 12 on the original graph. So, we pick the maximum D to be 16, slightly greater than 12. Consider two cases:

1. Case 1_O. Choose the *Order S-D-L*. Fig. 5.9 reports the analytical value of E.
2. Case 2_O. Choose *Order PriorityLabel"references"*. Fig. 5.10 reports the analytical value of E.
   Ploss for both cases are reported in Table 5.7

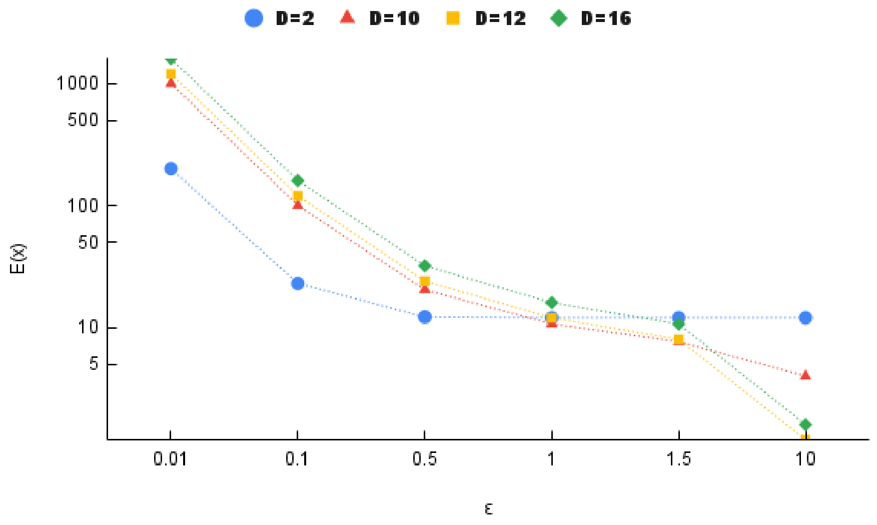


Figure 5.9: E for QC case 1_O

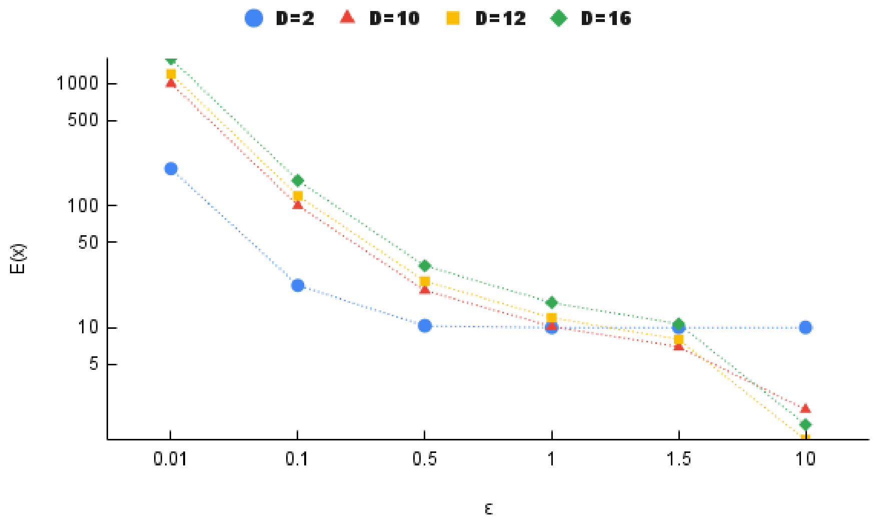


Figure 5.10: E for QC case 2_O

Regarding T_{QL}, if references $\in$ QL, then $\Delta^{\mathcal{G}^D_{QL}}_{d_{QL}} QC = D$, i.e., the sensitivity of QC w.r.t. d_{QL} restricted to the space of graphs with maximal QL-out-degree D is D. Consider two cases:

Degree bound D	2	10	12	16
Ploss case 1_O	1	0.33	0	0
Ploss case 2_O	0.8	0.16	0	0

Table 5.7: Ploss for QC case 1_O and case 2_O

1. Case 1_{QL} where QL={timestamp, emotion, text, references}. Edge ordering is important because we intend to compute maximum references out-degree, and according to our schema, nodes with "references" outedges have other outedges, including timestamp, emotion, and text. Take two orders under this case:

   - Case 11_{QL}. Where we take the *Order S-D-L*. We obtain the same results as in case 1_O. By the same results, we mean same $\bar{q}$ values, leading to same E values. Refer to Fig. 5.9. Ploss is also the same as in case 1_O. Refer to Table 5.7.
   - Case 12_{QL}. Where we take the *Order PriorityLabel"references."* We obtain the same results as in Case 2_O. By the same results, we mean same q(D) values, leading to same E values. Refer to Fig. 5.10. Ploss is also the same as in case 2_O. Refer to Table 5.7.

2. Case 2_{QL} where QL={references}. Edge ordering is not important because we are bounding references out-degree. So, whatever chosen edge ordering we will insert at the end of projection at most D-references outedges for each node. Take the *Order S-L-D*.

   We obtain the same results as in case 2_O. By the same results, we mean same $\bar{q}$ values, leading to same E values. Refer to Fig. 5.10. Ploss is also the same as in the case 2_O. Refer to Table 5.7.

Discussion We start first by discussing E results. In case 1_O and case 2_O, with D = 2, $\bar{q}$ equals 0 and 2 in case 1_O and case 2_O respectively. With D = 10, $\bar{q}$ equals 8 and 10 in case 1_O and case 2_O, respectively. Therefore, with these bounds, E values in case 1_O are slightly greater than E values in case 2_O (it is negligible when comparing Fig. 5.9 and Fig. 5.10). With D = 12, 16, $\bar{q}$ = q = 12 thus E = b.

The optimal order is the *Order PriorityLabel"references"* under outedge privacy with T_O. We should note here that the $\bar{q}$ values for the two chosen orders are quite close.

In case 11_{QL}, we obtain the same results as in case 1_O. In case 12_{QL}, we obtain the same results as in case 2_O.

Suppose our objective is to protect the references outedges of a node. In that case, it is sufficient to bound references out-degree (rather than bounding the out-degree) of a node to get the optimal results for QC. We are here under QL-outedge privacy with T_{QL}.

Moreover, in case 1_{QL} where QL={timestamp, emotion, text, references}, prioritizing references labels produces better results than the *Order S-D-L*. Clearly, the optimal order is the *Order PriorityLabel"references"* that is Case 12_{QL}.

On the contrary, if we want to protect all outedges of a node, then it is necessary to boud the out-degree and search for the best order. In our scenario, it is case 2_O (we might find other optimal order(s)). We are here under outedge privacy with T_O.

5.4.3.4 -QD- Count how many users tweeted more than 25 tweets

On the original dataset, QD outputs 3782, q = 3782.

$\Delta_{d_O}QD$= $\Delta_{d_{QL}}QD$ (given that tweeted ∈ QL)=1. The sensitivity is low and bounded on $\mathcal{G}$. There is no need to project the graph to reduce the sensitivity. Nonetheless, we project the graph, and we study different metrics.

Let us first consider T_O. $\Delta_{d_O}^{\mathcal{G}_o^D}QD = 1$, i.e., the sensitivity of QD w.r.t. d_O restricted to the space of graphs with maximal out-degree D is 1.

Edge ordering is important because we intend to count the number of users who tweeted more than 25 tweets. We are concentrating on edges with specific labels "tweeted". Hence, edge ordering is important.

Choose the out-degree bounds to go from 2 to 560. We select bounds 26 and 27 in particular with experimentation as by analyzing the results with other bounds we couldn't realize the importance of ordering. Consider three cases:

1. Case 1_O. Choose the *Order S-D-L*.

2. Case 2_O. Choose the *Order S-L-D*.

3. Case 3_O. Choose the Order PriorityLabel"tweeted".

Ploss for case 1_O, case 2_O, and case 3_O are reported in Table 5.8. For this query, we study the interest of projection. As stated before, $\Delta_{d_O}QD$= $\Delta_{d_{QL}}QD$ =1. We can achieve DP in $\mathcal{G}$. Noise is added to q, as explained in Section 5.4.2.3. We show PI for case 1_O only. It is reported in Table 5.9

Fig. 5.11 shows how E varies as a function of ε with D = 2 for the three cases. Fig. 5.12 shows how E varies as a function of ε with D = 26 for the three cases. Fig. 5.13 shows how E varies as a function of ε with D = 27 or with D = 50 or D = 200 or D = 500 or D = 560 for the three cases.

Degree bound D	2	26	27	50	200	500	560
Ploss case 1_O	1	0.69	0	0	0	0	0
Ploss case 2_O	1	1	0	0	0	0	0
Ploss case 3_O	1	0	0	0	0	0	0

Table 5.8: Ploss for QD case 1_O

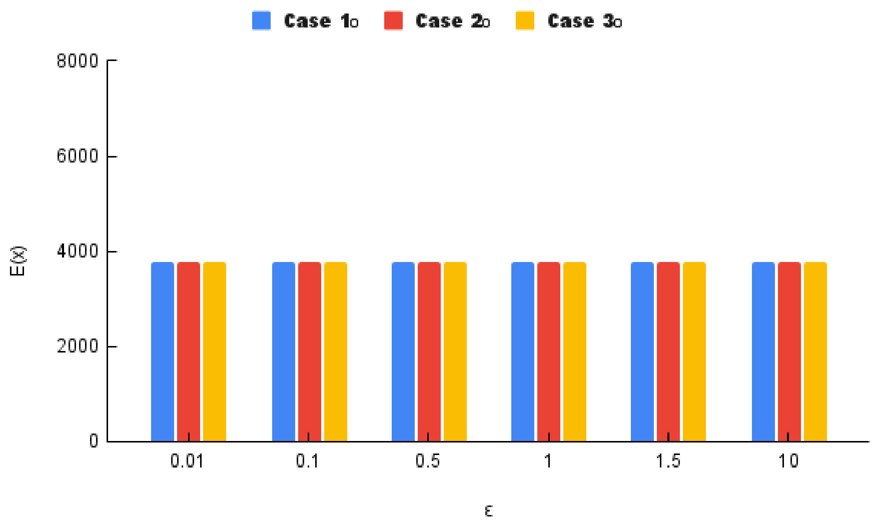


Figure 5.11: E with Out-degree Bound D = 2

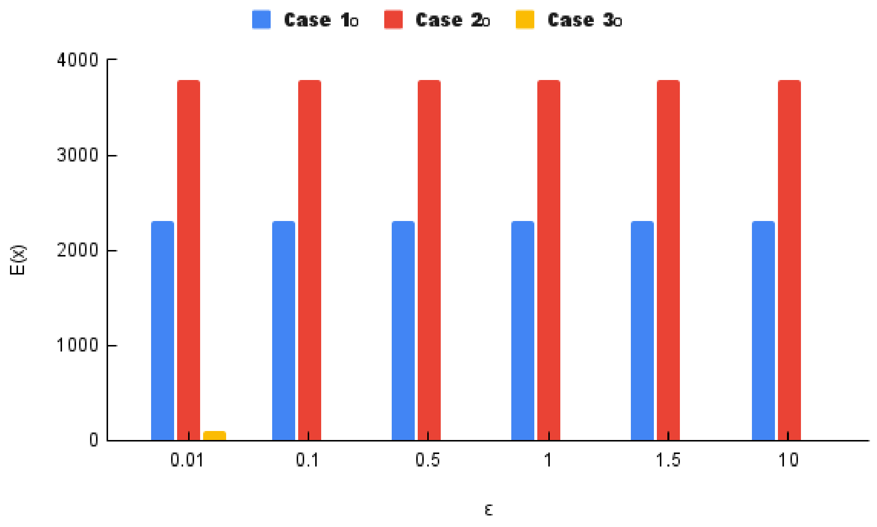


Figure 5.12: E with Out-degree Bound D = 26

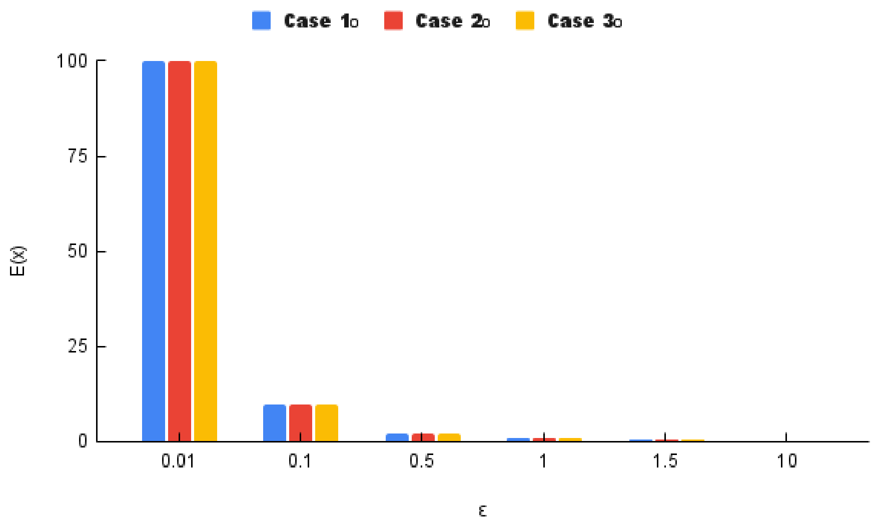


Figure 5.13: E with Out-degree Bound D = 27 or D = 50 or D = 200 or D = 500 or D = 560

epsilon	0.01	0.1	0.5	1	1.5	10
D = 2	0.97	0.99	0.99	0.99	0.99	0.99
D = 26	0.59	0.6	0.6	0.6	0.6	0.6
D = 27, 50, 200, 500, 560	0	0	0	0	0	0

Table 5.9: PI for QD case 1_O

Regarding T_{QL}, if tweeted $\in$ QL, then $\Delta_{d_{QL}}^{\mathcal{G}_{QL}^{D}} QD = 1$. Here, we only consider one case: case 1_{QL} where QL={tweeted}. Edge order is not important because we will insert at the end of projection at most D-QL-Outedges for each node no matter in which order "tweeted "outedges are chosen. Take any order. We take the *Order S-L-D*.

Choose the Ql-out-degree bounds to go from 2 to 560. We obtain the same results as in case 3_O. By the same results, we mean the same $\bar{q}$ values, leading the same E values (we are taking the same bounds D in both cases). Therefore, E for case 1_{QL} is the same as in case 3_O; Refer to Fig. 5.11, 5.12, 5.13. Ploss is also the same as case 3_O. Refer to Table 5.8.

Discussion We start first by discussing E results. In case 1_O, case 2_O, and case 3_O, we have b=$\frac{\Delta^O}{\varepsilon} = \frac{1}{\varepsilon}$. With D $= 2$, $\bar{q} = 0$ in the three cases. As a result, E $=$ q $= 3782$. Furthermore, with D $= 26$, $\bar{q} = 1477$ in case 1_O and $\bar{q} = 0$ in case 2_O. As a result, E $= 2305$ and 3782 (equal to q) in case 1_O and case 2_O, respectively. Conversely, in case 3_O with D $= 26$, we get $\bar{q} =$ q $= 3782$. As a result, E $=$ b (corresponding to each epsilon value). With D ≥ 27, we obtain $\bar{q} =$ q $= 3782$ in the three cases. So, E $=$ b (corresponding to each epsilon value).

The results for the three chosen orders are quite close. We notice the importance of edge ordering with D $= 26$, seeing that in case 3_O, where we give priority to tweeted outedges, we get $\bar{q} =$ q with this bound (optimal results). On the other hand, in case 2_O we get $\bar{q} = 0$

(worst results). The optimal order is the *Order PriorityLabel"tweeted"* under Outedge privacy with T_O.

In case 1_{QL}, we obtain the same results compared to case 3_O. This means that giving priority to tweeted outedges while bounding out-degree of nodes (case 3_O) yield the same results as bounding "tweeted" outedges under QL-outedge privacy with T_{QL}.

Suppose the projection is a must, and our objective is to protect the tweeted outedges of a node. In that case, bounding a node's tweeted out-degree (rather than bounding the out-degree) is sufficient to get the optimal results for QD. We are here under QL-outedge privacy with T_{QL}.

On the other hand, if we want to protect all outedges of a node, then it is necessary to bound the out-degree and search for the best order. In our scenario, it is Case 3_O. We are here under outedge privacy with T_O.

Now regarding Ploss, in case 1_O, case 2_O, and case 3_O: In case 3_O, giving priority to "tweeted" labels results in optimal results (Ploss=0) with D > 25. With D $\geq$ 27, Ploss = 0 in the three cases. This implies that at least 26 tweets are preserved with the projection following the orders in each case.

5.4.3.5 -QE- counts the number of users user "Garythetwit" has referenced

On the original dataset, QE outputs 55, q = 55.

This query leverages the dataset's semantics, refers to a path of size greater than 1, and showcase several interesting properties.

Interest of the approach. It is immediate that $\Delta_{d_n} QE$ and $\Delta_{d_O} QE$ are infinite. Thus, in this case, it would not be possible to construct a DP mechanism directly from the original query without reducing its sensitivity. We will see in what follows that the approach also provides significant improvement in utility if one were to construct a restricted mechanism without projection.

If neither *tweeted* nor *references* are considered sensitive (i.e., are in QL), $\Delta_{d_{QL}} QE$ is 0 and QE do not provide any insight to an attacker. If at least one of the two is sensitive, $\Delta_{d_{QL}} QE$ is also infinite. In what follows, we consider $QL = \{$*references*, *tweeted*$\}$.

QE considers solely *tweeted* and *references* outedges. According to the schema represented in Fig. 5.2, a node with a *tweeted* outedge has two other outedges related to its type and name. A node with *references* outedges has four other outedges related to its *timestamp, emotion, text* and *query term*.

Let us first consider T_O. An immediate heuristic to reduce information loss during projection is to prioritize *tweeted* and *references* edges over the other types. Assuming a bound D greater than the maximal number of users referenced in a tweet plus 4, it is immediate that a projection bounding the out-degree of the graph to D will not have any impact on the out-degree of tweets. If the projection gives priority to *tweeted* outedge, it leads to the same data loss as a projection bounding it to $D + 2$ while giving lowest priority to *tweeted*. We propose two orders:

1. Case 1_O. Choose the *Order PriorityLabel"references"*
2. Case 2_O. Choose the *Order PriorityLabels"tweeted,references"*

Regarding *Order PriorityLabels"tweeted, references"*, the relative order between *references* and *tweeted* edges does not matter; according to the schema, these type of edges originate from different types of nodes. Note that such a use of lexicographical order for non-prioritized edge labels means that *tweeted* edges have the lowest rank according to *Order PriorityLabel"references"*. According to the previous remark, and assuming that the maximal out-degree of tweets is lower than the maximal out-degree of users, there exists a couple (D_m, D_M) such that for any D, $D_m \leq D \leq D_M$ Ploss of T_O with $o_{r,t}$ and D is equal to Ploss of T_O with o_r and $D+2$.

Regarding T_{QL} with $QL = \{$*references*, *tweeted*$\}$, since edges that are neither *references* nor *tweeted* do not count during projection, there is no difference between *Order PriorityLabel"references"* and *Order PriorityLabels"tweeted,references"* and in both cases, T_{QL} behaves like T_O with *Order PriorityLabels"tweeted,references"* that is case 2_O. Our experimental evaluation of Ploss, reported in Table 5.10, confirms these considerations. Name this case under T_{QL} by case 1_{QL}.

Here, $\Delta_{d_{QL}}^{\mathcal{G}_{QL}^D} QE = \Delta_{d_O}^{\mathcal{G}_o^D} QE = D^2$, i.e., the sensitivity of QE w.r.t. d_{QL} (resp. d_O) restricted to the space of graphs with maximal QL-out-degree (resp. out-degree) D is D^2. Note that this is quite pessimistic and does not consider the database schema. Indeed, we consider that for any value of D, D tweets can reference D users each. In reality, and due to character limits, the number of users referenced in a tweet is limited. Considering the database schema and constraints could lead to further reduction of the query's sensitivity over the projected space.

Table 5.10: Ploss for QE

Degree bound D	2	4	10	50	500	560
Ploss case 1_O	1	1	0.9	0.27	0	0
Ploss case 2_O	1	0.96	0.89	0.27	0	0
Ploss case 1_{QL}	1	0.96	0.89	0.27	0	0

We compute the analytical value of E. Since the sensitivities in the projected space are equal, and since we have seen previously that the projected graphs are quite similar, we report in Fig. 5.14 only the analytical value for case 1_{QL} (which is the same as case 2_O).

Discussion We start first by discussing E results. As expected, E decreases while ε increases: utility increases as privacy guarantees weaken. More interestingly, E decreases with D, meaning that *increase of information loss due to a tighter bound is compensated by the decrease in the amplitude of noise added to obtain DP guarantees*. With $\varepsilon = 1$, $\frac{EwithD=560}{EwithD=50}$ is roughly 125, meaning that the expected distance between the private answer and the real value is 125 times greater with bound $D = 560$ than 50. Interestingly, as said before, $D = 560$ is an extreme case where the graph is not modified during projection. We have also seen that $\Delta_{d_O} Q = \infty$,

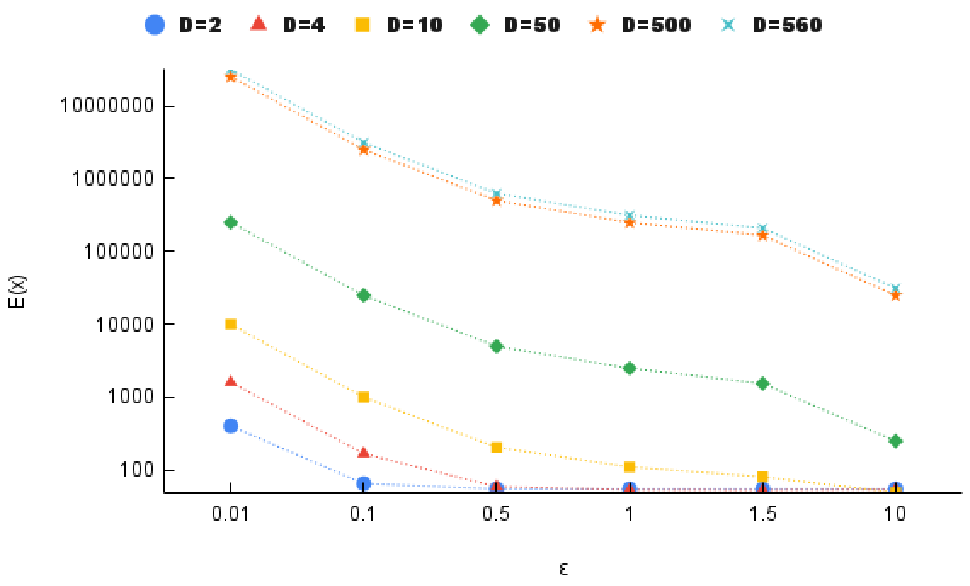


Figure 5.14: Analytical value of E for case 1_{QL}

meaning that no DP mechanism can be trivially constructed over $\mathcal{G}$. A straightforward – but somewhat weak– approach would be to construct a restricted DP mechanism over some subspace of $\mathcal{G}$, typically $\mathcal{G}_o^D$ or $\mathcal{G}_{QL}^D$ with $D = 560$. This would provide exactly the same results as our approach with $D = 560$, which provides utility several orders of magnitude worse than a regular parametrization of our approach with a bound $\leq$ 50.

Regarding Ploss, experimental bounds go from 2 to 560, meaning that we preserve at minimum up to 2 (sensitive) outedges per node and at most 560. $D = 2$ obviously leads to an inoperable database with an information loss of 1. 560 is an extremal. Indeed, the maximum out-degree of a node in the dataset is 551; therefore, a projection with $D = 560$ leads to 0 modification and 0 information loss. As expected:

1. Case 2_O is equivalent to case 1_{QL} with $QL = \{$*references*, *tweeted*$\}$.

2. Case 1_O leads to slightly more loss than these two projections with a small bound: from 1 to 0.96 and 0.9 to 0.89 information loss with D equal to 4 and 10, respectively.

Note that Ploss with $D = 50$ is $0, 27$, $\bar{q}$ in this case being 40. Indeed, the projections keep the edge between *Garythetwit* and some of its edges that do not contain any reference. With $D > q$, an optimal projection (w.r.t. Ploss) would lead to a $\bar{q}$ between D (worst case scenario where each tweet contains a single reference) and D^2 (there exist at least D tweets with at least D references).

5.5 - Conclusion

This chapter presents a new approach based on *graph projection* to adapt differential privacy to edge-labeled directed graphs –e.g., RDF graphs– while reducing the amplitude of the randomized noise.

The main idea is to use *graph projection* to reduce the sensitivity of queries. We propose three edge-addition based graph projection methods that transform an RDF graph into a graph of bounded degree, out-degree, or typed-out-degree. We show that two of these projections preserve neighborhood w.r.t. two different privacy models. Consequently, for said projections and models, the global sensitivity of the composition (query $\circ$ projection) is at most equal to the global sensitivity of the query over the projected space. Thus, we obtain a general method to expand the domain of any DP mechanism over a restricted projected space to the space of RDF graphs. We experimentally and analytically demonstrate the feasibility and interest of the approach on a real Twitter dataset w.r.t. five queries, most of which have infinite sensitivity on the original space. In such cases, we also show that our approach provides a utility several orders of magnitude better than a naive approach relying on restricted DP without projection.

6 - Mapping relational databases to RDF and its impact on privacy

Contents

6.1 - Motivation and approach

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 system to manage data[1].

The Semantic Web [164] is an extension of the current Web. Its goal is to create a metadata-rich Web based on the Resource Description Framework (RDF) data model. RDBMS are the most widespread systems for data storing (in particular for Web data) [165,166]. Thus, mapping

[1]See : https://db-engines.com/en/ranking

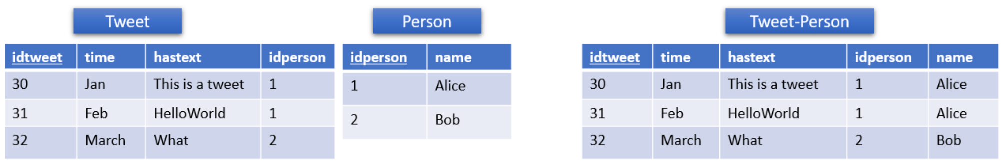
relational databases to RDF (usually referred to as RDB2RDF) is the key to creating a metadata-rich Web. This has been an active field of research during the last decades [165, 167–169].

Data collected (whether stored in relational databases 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 (GDPR) [70] in the European Union), it becomes necessary to use or share data without compromising the privacy of data contributors.

DP [13, 14] has emerged to be the flagship of data privacy when publishing and sharing data. In the initial definition of DP by Dwork [89], a database is commonly a single, monolithic table of records (or tuples) that holds private data. The classical definition of a neighborhood defines neighboring databases as those that differ by one record. DP protects the presence or absence of any single record in the database. The intuition here is that each individual participates in, at most, one database record.

Defining neighborhoods for multi-relational databases, i.e., databases composed of many tables, is challenging for many reasons (see for instance [26]) and as we will see in below. 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 chapter, we consider an important type of constraints, FK constraints, sometimes associated to cardinality constraints.

Definition 6.1 (Foreign Key (FK))**.** *In a relational database, a FK constraint is a set of attributes K_{FK} of a table T_1 referencing a set of attributes K_{PK} (of same cardinality and type) of table T_0 such that K_{PK} is a primary key of T_0 and all values of K_{FK} that appear in T_1 must appear in F_{PK} in T_0. A primary key defines a unicity constraint on the table : a given value of the primary key can only appear once.*



Tweet

idtweet	time	hastext	idperson
30	Jan	This is a tweet	1
31	Feb	HelloWorld	1
32	March	What	2

Person

idperson	name
1	Alice
2	Bob

Tweet-Person

idtweet	time	hastext	idperson	name
30	Jan	This is a tweet	1	Alice
31	Feb	HelloWorld	1	Alice
32	March	What	2	Bob

Figure 6.1: A multi-relational database under FK constraints (left); a single table database (right)

FK constraints model cardinality dependencies between tables, and are pivotal to adding basic semantics to relational databases. Common cardinalities are one-to-one (e.g. a person has a social security number, and a social security number only belongs to one person), one-to-many (e.g. a person can post many tweets, but a tweet is only posted by a single person), and many-to-many (e.g. a student can follow several courses and a course can be followed by several students) relationships. We are interested in studying one-to-many relationships and many-to-many relationships. Indeed, one-to-one cardinalities do not create any new problems.

In order to circumvent the multi-table approach, one idea would be to build a single table by joining all the tables in the database. While this works for one-to-one cardinalities (since this does not generate any new lines), this leads to space and query processing time problems. Moreover, in some cases this solution is inadequate to provide enough privacy protections. For instance, consider the database presented in Fig. 6.1 with the schema Tweet (idtweet,time, hastext, idperson) and Person (idperson, name). Tweet.idperson is a FK referencing Person. Notice that in the single table case, we can only represent a neighboring database which differs by a row of Tweet but not by a row of a Person. We would need to introduce a new definition of neighborhood if we wanted to take model that an adjacent database differs by a person : we would need to consider that a database is adjacent if it differs by all the rows containing a given person name. Also note that deleting a record from one table may result in (cascading) deletions [160] in other tables that are linked to it via FKs, thus making it impossible to respect the initial DP definition if we want to satisfy these constraints.

Mapping relational databases to RDF and then applying a privacy model in the mapped RDF graph and consequently defining the privacy semantics will depend on the generated RDF graph. It is indispensable to do the mapping according to the needed privacy protection in RDF so that when applying the privacy model in the mapped RDF graph, the privacy guarantees are appropriate.

Defining a privacy model in RDF to fit a known model can ease the specification of DP mechanism (in RDF), and having the privacy model making sense in the relational database allows the data owner that has knowledge in relational database to still comprehend the relational privacy model.

Sketch of our approach

In this chapter, we propose to study how to map (or *translate*) a relational database into a RDF database, in order to achieve existing neighborhood definitions on graphs (such as node-DP, or more semantic definitions such as QL-Outedge-DP), and how these neighborhood definitions can be expressed on a relational database. In other words, we seek to provide meaningful definitions of neighborhoods on relational databases under FK constraints, that translates into existing privacy model on graphs. The chapter is organized as follow:

- We present the necessary background concerning mapping relational database to RDF (Section 2.5);
- We discuss existing work on the difficulties of defining neighborhoods and DP in the multi-relation context (Section 4.4);
- We present the structure of the database used as running example (Section 6.2);
- We present our first contribution: To define meaningful concepts on relational database, we start by considering the classical case of cascade deletion in a relational database, where a database which is obtained by deleting *one* tuple and cascading the others linked by FK constraints. We translate both instances to an RDF database and see if it corresponds to an existing graph privacy model (Section 6.3). We tweak this classical model and propose to use restrict deletion in a relational database and show it translates in a meaningful RDF privacy model (Section 6.4);

- We present our second contribution: We propose how to define neighborhoods in a relational database that translate into QL-Outedge privacy, which we believe is well adapted to the context of RDF databases, and thus provides interesting semantics for the neighborhoods (Section 6.5);

6.2 - Use case database

In this chapter, we use as an example a simple Twitter database using MYSQL to map relational databases to RDF. We use this example both to illustrate our propositions and also in our tests in order to verify our implementation.

The Twitter schema is inspired by Sentiment140 dataset composed of 1.6 million tweets [2]. The ER diagram is shown in Fig. 6.2. We create a database instance of the Twitter schema, as shown in Fig. 6.3. The IDE used is the Eclipse IDE.

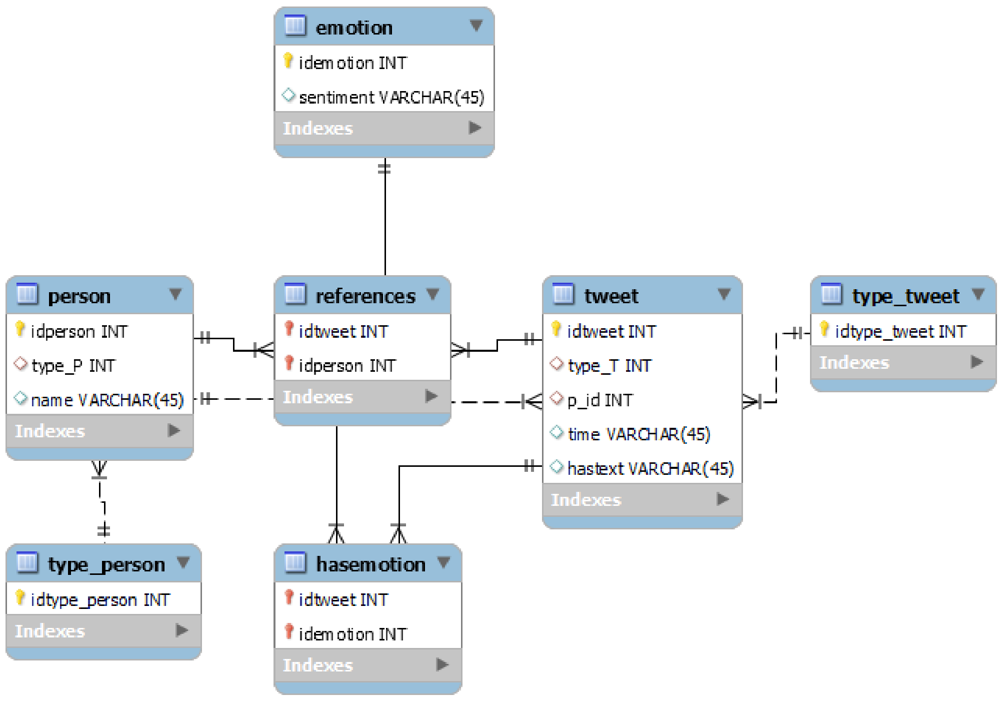
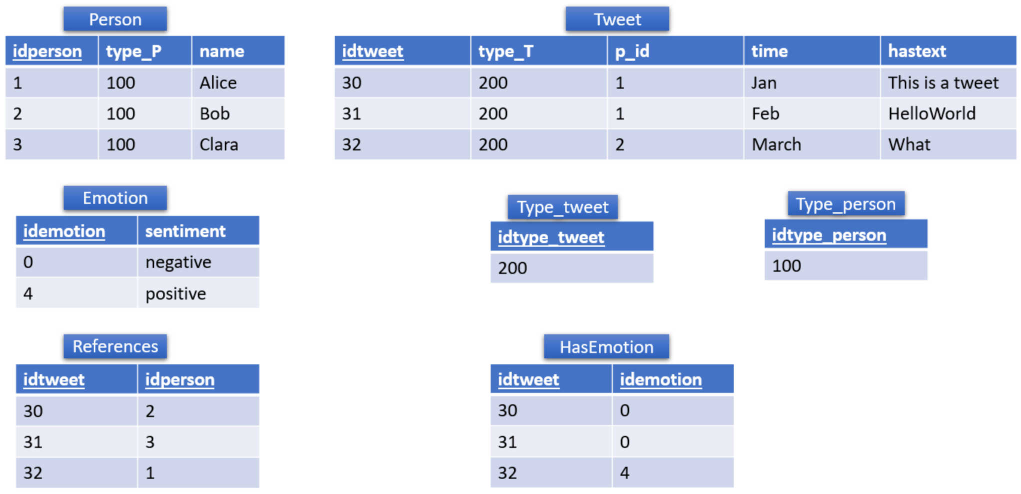


Figure 6.2: ER Diagram

6.2.1 - Direct Mapping

DB2Triples[3],[4] is an implementation of W3C DM and R2RML Mapping. It is developed by the Antidot company and delivered as a Java library, available under the LGPL 2.1 open source licence. However, it's no longer maintained by Antidot. It is validated with PostgreSQL and

[2]https://www.kaggle.com/kazanova/sentiment140
[3]https://github.com/antidot/db2triples
[4]https://antidot.net/fr/2011/10/07/db2triples-une-implementation-de-r2rml-et-directmapping-en-c



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

Figure 6.3: A database instance of the Twitter schema

MySQL back-ends. The DM process in this paper is done with DB2Triples with the stages shown in Fig. 6.4.

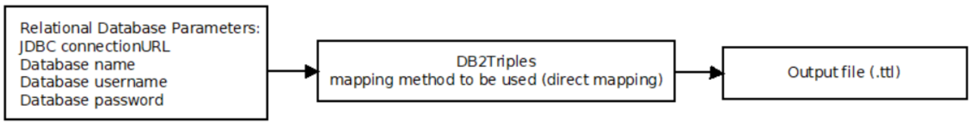


Figure 6.4: Direct mapping process

6.2.2 - R2RML Mapping

We select R2RML-F [170], an R2RML implementation available on Github[5]. The R2RML mapping process is done with R2RML-F with the flow diagram shown in Fig. 6.5. R2RMl-F engine takes as input connectionURL, a JDBC connection URL to a database, username and password for the user connecting to the database, the R2RML mapping file, and the format of the output file to generate RDF data. We manually write R2RML mappings, which are tailored to our database schema.

As stated in Chapter 2, one key advantage of R2RML is to represent many-to-many relations as simple triples. In our database example, we have a many-to-many relationship between Person and Tweet. This many-to-many relationship is captured by the content of the References table. Also, we have a many-to-many relationship between Tweet and Emotion. This many-to-many relationship is captured by the content of the HasEmotion table. When translating to RDF, one can see that there will be two options to choose the direction of the link in the R2RML file for each References and HasEmotion.

[5]https://github.com/chrdebru/r2rml

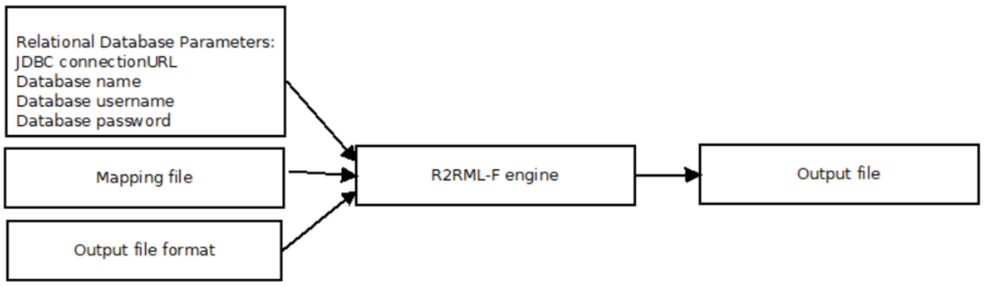


Figure 6.5: R2RML mapping process

For References:

1. The direction of link from Tweet to Person.
2. The direction of link from Person to Tweet.

For HasEmotion:

1. The direction of link from Tweet to Emotion.
2. The direction of link from Emotion to Tweet.

Throughout this chapter, IRIs are simplified using prefixes. In lisiting 8 we present all the prefixes used. For instance rdf: is a shorthand for http://www.w3.org/1999/02/22-rdf-syntax-ns#.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX tweet: <http://foo.example/DB/tweet/>
PREFIX person: <http://foo.example/DB/person/>
PREFIX references: <http://foo.example/DB/references/>
PREFIX reference: <http://foo.example/DB/references#>
PREFIX pr: <http://foo.example/DB/tweet#>
PREFIX t: <http://foo.example/DB/type_tweet/>
PREFIX r: <http://example.com/resource/>
```

Listing 8: Prefixes

6.3 - Investigating a good neighborhood concept for multi-relational databases

In this Chapter, we are interested in multi-relational databases with FK constraints and in the notion of neighborhood beyond one-row neighbors. Our goal is to allow the data owner to designate which entities in the schema require privacy, with the most flexibility possible. We adopt the notion of neighboring databases introduced by Kotsogiannis et al. [160], presented in Section 4.4.2, and we analyze this notion on the relational database example, Twitter. After that, we analyze what this neighborhood means in the mapped RDF graph. Throughout this chapter, we will name this neighborhood the **cascade delete neighborhood**.

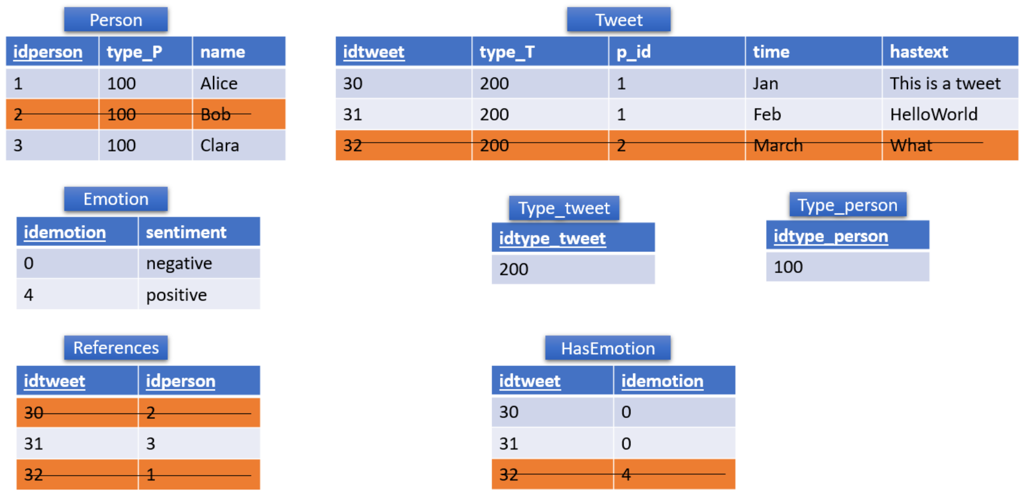
Approach. Our approach is to map both the relational database and its neighbor to RDF, and see what characteristics their images in the RDF domain have.

We next present a case study where we define the privacy policy as P = (Person, ε), P = (Tweet, ε), and P = (Emotion, ε), and analyze what the corresponding neighborhoods means in the mapped RDF graph. The mapped RDF graph is computed according to the proposed approach in Section 6.5.3. Note that the same analysis holds if the mapped RDF graph is done according to the default R2RML mapping explained in Section 6.5.1.2. The only difference will be in the synopsis shown of the mapped RDF graph in which the direction of *tweetedby* outedge will be from *Tweet* node to *Person* node.

6.3.1 - Case 1- Privacy policy P = (Person, ε)

Person is the primary private relation, implying that Tweet, References, and HasEmotion will be secondary private relations. Fig. 6.6 shows a neighborhood instance of the original database under privacy policy P = (Person, ε). Removing Person with idperson=2 results in deleting one row in Tweet, two in References, and one in HasEmotion. In this case, neighboring databases differ in the primary private relation Person and the secondary private relations. In the mapped RDF graph, the defined neighborhood in relational databases means:

- Deleting the node *Person*, its inedges and outedges, plus isolated Literals.
- Deleting node *Tweet*, its inedges and outedges, plus isolated Literals.



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~~

Figure 6.6: Cascade neighboring under Person policy

Fig. 6.18 shows a synopsis of the mapped RDF graph, and Fig. 6.7 shows a neighboring graph of the mapped RDF graph. The neighboring graph is obtained by deleting nodes *Person* and *Tweet*, their inedges, outedges, plus isolated Literals.

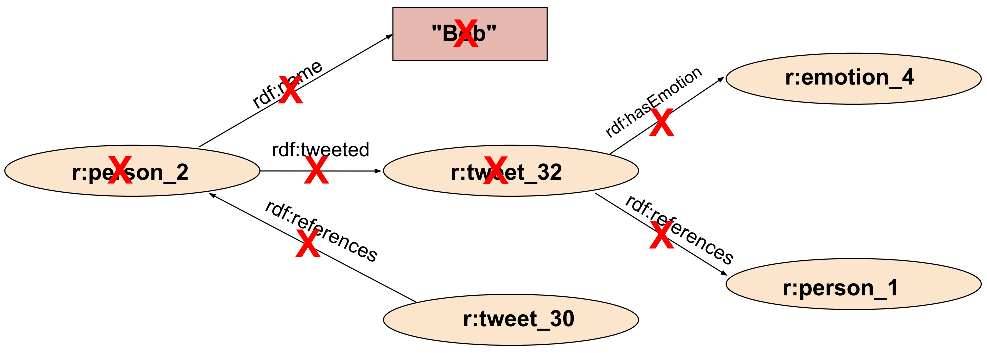


Figure 6.7: Case 1: Neighboring of the mapped RDF Graph

6.3.2 - Case 2- Privacy policy P = (Tweet, ε)

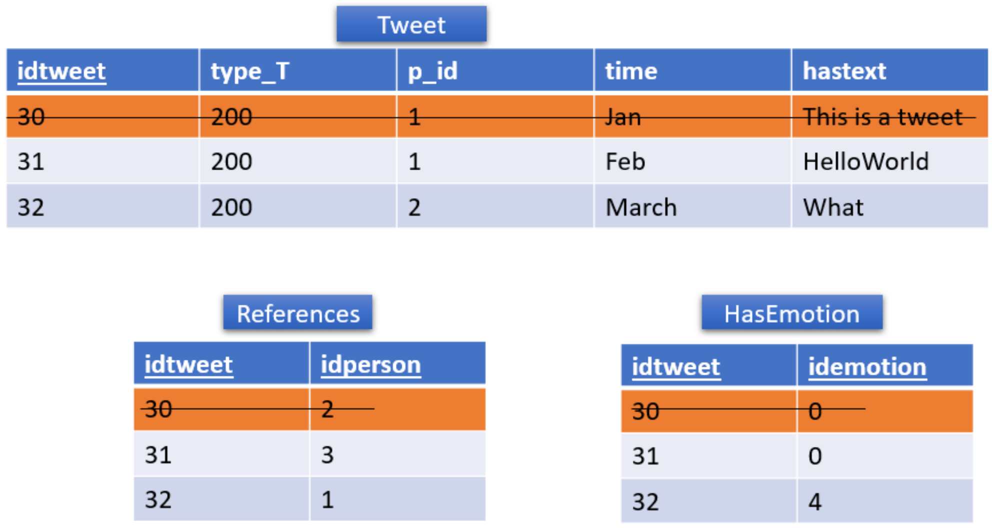
Tweet is the primary private relation which implies that References and HasEmotion will be secondary private relations. Figure 6.8 shows a neighboring instance of the original database under privacy policy P = (Tweet, ε). Removing idtweet=30 results in deleting one row in both References and HasEmotion tables. In this case, neighboring databases differ in the primary private relation Tweet and secondary private relations.

In the mapped RDF graph, this translates to deleting the node *Tweet*, its inedges and outedges, plus isolated Literals.

Fig. 6.21 shows a synopsis of the mapped RDF graph, and Fig. 6.9 shows a neighboring graph of the mapped RDF graph.

An immediate adaptation of node-DP to typed graphs would be typed node-DP that restricts node-DP to node of certain type(s) typed node, similar to how QL-outedge-DP restrict outedge-DP to edges of a certain type. In this case, neighboring graphs differ by a single node of a certain type.

Here, the obtained neighborhood in RDF is similar to typed-node privacy, where we concentrate on nodes of type tweet, where the node itself is deleted as well as its inedges and



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

References

idtweet	idperson
~~30~~	~~2~~
31	3
32	1

HasEmotion

idtweet	idemotion
~~30~~	~~0~~
31	0
32	4

Figure 6.8: Cascade neighboring under Tweet policy

outedges. The only difference here is that isolated Literals are deleted also.

6.3.3 - Case 3- Privacy policy P = (Emotion, ε)

Emotion is the primary private relation, implying that HasEmotion will be the secondary private relation. Figure 6.10 shows a neighboring instance of the original database under privacy policy P = (Emotion, ε). Removing idemotion=0 from Emotion results in deleting two rows in HasEmotion table. In this case, neighboring databases differ in the primary and secondary private relation.

In the mapped RDF graph, this translates to deleting the node *Emotion*, its inedges and outedges, plus isolated Literals.

Fig. 6.25 shows a synopsis of the mapped RDF graph, and Fig. 6.11 shows the corresponding neighboring graph.

The obtained neighborhood in RDF is similar to typed-node privacy, where we concentrate on nodes of type emotion where the node itself is deleted, its inedges and outedges. The only difference here is that isolated Literals are also deleted.

6.3.4 - Equivalency in RDF

After analyzing the notion of neighborhood in the relational database and studying it in RDF, we see that the neighborhood obtained in the mapped RDF is similar to typed node-DP in cases 2 and 3 but not in case 1. In the latter case, more nodes are deleted through cascading. The problem arises in case 1 due to the presence of Tweet as a secondary relation and specifically the one-to-many relationship between Person and Tweet that results in deleting one

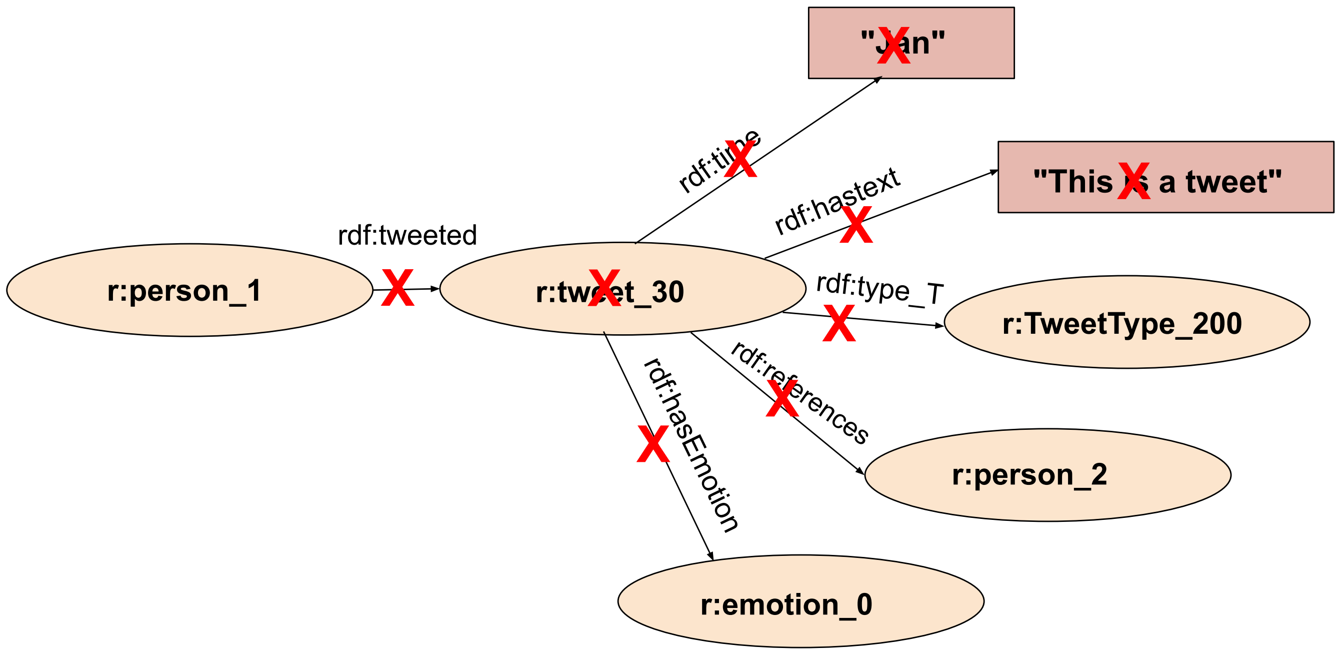


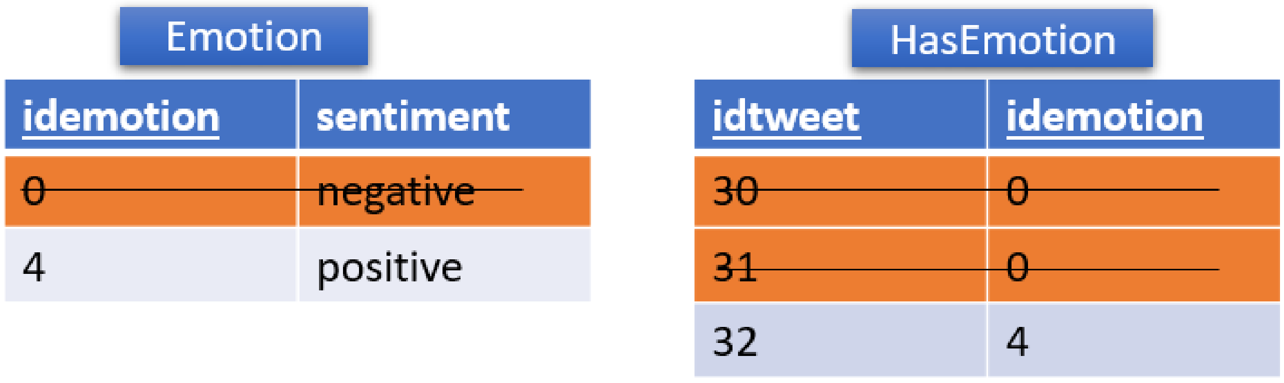
Figure 6.9: Case 2-Neighboring of the mapped RDF Graph

row in Tweet. On the other hand, case 2 and 3 leads to a neighborhood similar to typed-node DP , because the secondary relations are join tables.

Note that node-DP is usually considered the most robust privacy model in graph and that it can lead to high sensitivities, as discussed in the previous chapter.

We study if we can make more like typed node-DP to fit a known model and ease the specification of DP mechanism (in RDF) while still making sense in relational database (so that the data owner that has knowledge in relational database can still understand the relational privacy model). We thus propose next a slight modification of the neighborhood definition in the relational database so that the obtained neighborhood, and consequently, the privacy model in RDF deletes less. We call it *restrict delete neighborhood*.

6.4 - Proposition: Restrict deleting neighboring



Emotion

idemotion	sentiment
~~0~~	~~negative~~
4	positive

HasEmotion

idtweet	idemotion
~~30~~	~~0~~
~~31~~	~~0~~
32	4

Figure 6.10: Cascade neighboring under Emotion policy

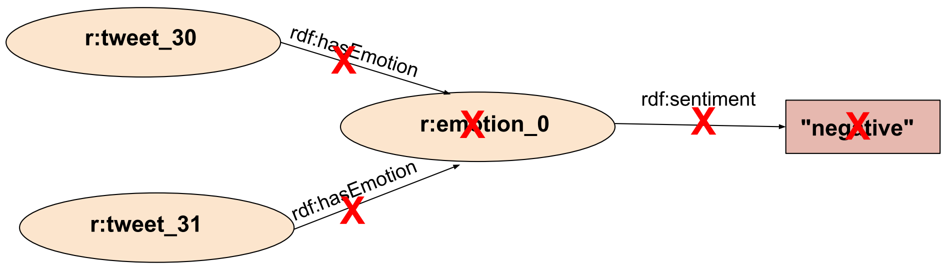


Figure 6.11: Case 3-Neighboring of the mapped RDF Graph

We next detail our first proposal: the **restrict delete neighborhood** for a relational database. Let **R** be the database schema. The key idea is that one relation is designated to be the primary private relation R_p and the secondary private relations are defined as relations with a direct FK referencing R_p. Let **I** be a database instance over **R**. For any R ∈ **R**, let **I**(R) be the relation instance of R in **I**. The referencing relationship over the records is defined as follows: for $t \in \mathbf{I}(R)$ and $t' \in \mathbf{I}(R')$, we say that t' reference t if R' references R and the FK of t' equals the PK of t.

Definition 6.2 (Restrict delete neighborhood)**.** *We define the neighborhood as select a record t in the primary private relation, delete the entire row, cascade deletion to the secondary relations as follows:*

- *If the secondary relation represents a many-to-many relationship, we cascade delete to the entire row.*
- *If the secondary relation is in a one-to-many relationship or in a one-to-one relationship with the primary one (the PK of the primary relation appears as an FK in the secondary relation), then we restrict the deletion in the secondary relation to just the FK entry and not the entire row. In other words, we set the FK entry to NULL.*

Analyzing this **restrict delete neighborhood** definition in the mapped RDF graph means deleting the node, its inedges and outedges, plus isolated Literals. The obtained model in RDF is similar to typed-node privacy as we concentrate on nodes of a specific type that depends on the selected primary private relation.

Now we refer back to the Twitter example to illustrate **restrict deleting neighboring**. Cases 2 and 3 remain unchanged while considering cascade or restrict delete, because the secondary relations are join tables. The change in definition impacts case 1.

6.4.1 - Case 1- Privacy policy P = (Person, ε)

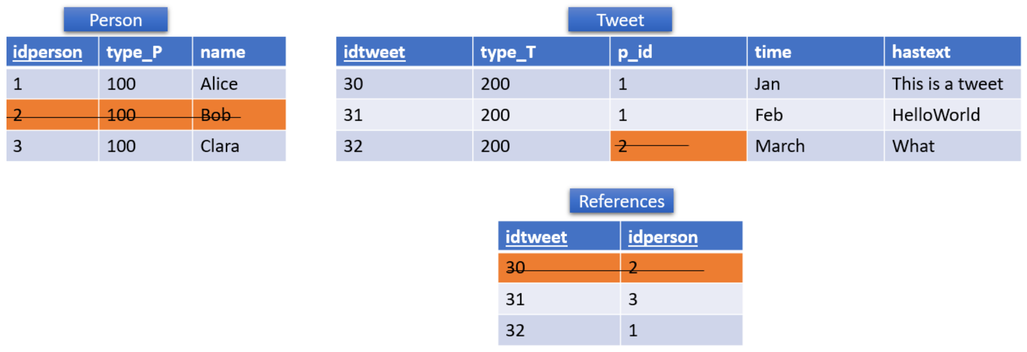
With the privacy policy P = (Person, ε), Person is the primary private relation. idperson appears as a FK in Tweet and References. Then secondary private relations are Tweet and References. Since References represent many-to-many relationship, we cascade deletion of the entire row. In Tweet, we restrict deletion to the FK entry.

Fig. 6.12 shows a neighboring instance of the original database under privacy policy P = (Person, ε). Removing a person with idperson=2 results in deleting the person FK entry in Tweet and one row in References. In this case, neighboring databases differ in the primary private relation and the secondary private relations. Note that we are not deleting the entire tweet row; we restrict the deletion to just the FK entry.

In the mapped RDF graph, the defined neighborhood in relational databases means deleting the node *Person*, its inedges and outedges, plus isolated Literals.

Fig. 6.18 shows a synopsis of the RDF graph. We take the same synopsis as in Case 1 in Section 6.5.4.1. Fig. 6.13 shows a neighboring of the mapped RDF graph under restrict deletion.

The obtained neighborhood in RDF is similar to the typed-node privacy neighborhood definition where we concentrate on nodes of type person, and where the node itself is deleted, its inedges and outedges. The only difference here is that Literals are deleted.



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

References

idtweet	idperson
~~30~~	~~2~~
31	3
32	1

Figure 6.12: Restrict Neighboring DB instance under Person policy

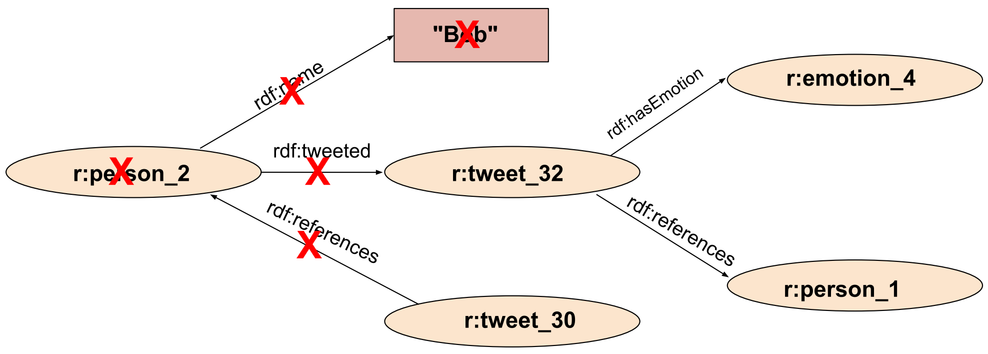


Figure 6.13: Restrict deleting neighboring of the mapped RDF graph

In conclusion, inspired by [160], we have proposed the **restrict delete neighborhood** for relational databases, which captures privacy policies and key constraints and deletes less information when compared to [160] (Refer to Section 6.3.1). **We show that by accepting NULL as a FK** and analyzing this neighborhood definition in the mapped RDF graph, we get a model similar to typed node privacy.

6.5 - QL-Outedge Privacy over Relational databases

In the previous sections, we have seen it is possible to propose a simple and meaningful adaption of a classical privacy model in relational databases, that when transformed to an

RDF database, correspond to typed-node privacy. However, as discussed in Chapter 5, other privacy models on graphs are better adapted to RDF graphs, such as QL-Outedge privacy, which we proposed to use in [33]. Thus in this section, we propose to study how we can translate the QL-Outedge privacy model to the relational database setting. The final goal is to propose a neighborhood definition for a relational database which would be the equivalent of the QL-Outedge neighborhood defined over RDF graphs.

Approach. Our approach is to map the relational database to RDF, apply QL-outedge privacy in RDF, then return to relational world to define the corresponding neighborhood. When mapping a relational database to RDF, there is the edge direction to be tailored in order to provide the appropriate privacy guarantee under QL-outedge privacy. Thereupon, we propose a model where one can choose the direction of edges according to what to protect.

6.5.1 - Mapping relational data to RDF

We first start by mapping the relational database to RDF. As mentioned earlier, W3C propose DM and R2RML mapping. In the following, we analyze the result of DM and highlight its shortcomings. Then, we propose a model where one can decide what to protect by leveraging R2RML mapping capabilities to control privacy protection. To achieve this goal, we write R2RML mapping to be able to control the privacy protection.

6.5.1.1 - Direct Mapping then applying QL-outedge privacy

Direct mapping process The DM of the Twitter database is done by Db2triples. The generated output file is available online[6]. In what follows, we analyze the RDF graph produced from the W3C DM, particularly the direction of the edge, which corresponds to the one-to-many relationship between Person and Tweet. In table Tweet in Fig. 6.3, idtweet is the PK, and Tweet.p_id is a FK to Person. Let us take as an example the first row in this table. We have idtweet=30 and p_id=1. This signifies that idtweet=30 is *tweetedby* person with idperson=1. We plot a synopsis of the DM result of this particular example in Fig. 6.14. The direction of edge is from node *Tweet* to node *Person*. Applying QL-outedge privacy over the obtained RDF graph means protecting the QL-outedges of a node. If QL=L, it means protecting all the triples it is the subject of. We are interested in the specific outedge from node *Tweet* to node *Person* i.e the *tweetedby* outedge represented as a predicate pr:p_id. In this case, QL-outedge privacy protects the author of one tweet. Therefore, the default privacy protection with the DM is protecting the author of one tweet. The direction of edges plays a significant role in defining the neighboring databases under QL-outedge privacy model.

The default privacy protection with DM is to protect the author of one tweet. However, it would be interesting to protect all the tweets of a person, of an author, rather than the author of one tweet. This implies that node *Person* must have a *tweeted* outedge and not *tweetedby* inedge.

We also analyze the translation of many-to-many relationships, for instance, between Person and Tweet, which is captured by the content of the References table. In References, we have a row with idtweet=30 and idperson=2. This signifies that idtweet=30 *references* person with

[6]https://github.com/sarataki/mapping/blob/main/directmapping/output.ttl

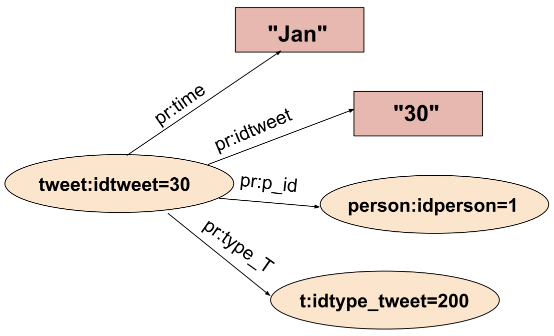


Figure 6.14: DM: translation of one-to-many relationship

idperson=2. DM generates triples in listing 6.1. We draw this part of the RDF graph to better visualize the results in Fig. 6.15. As we can see, many-to-many relations are not translated to simple triples, but rather, an intermediate node is created.

Listing 6.1: Generated triples

```
references:Twid=30,Peid=2   reference:Peid
person:ideperson=2;  reference:Twid  tweet:idtweet=30.
```

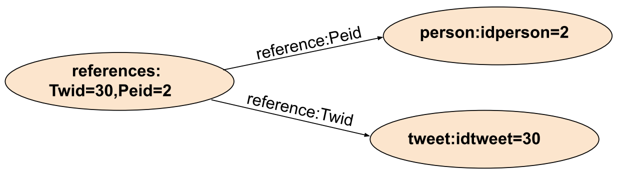


Figure 6.15: Translation of many-to-many relationship

6.5.1.2 - Default R2RML Mapping then applying QL-outedge privacy

We manually write R2RML mapping file according to W3C R2RML recommendation[7]. In our Twitter example, References and HasEmotion represent many-to-many relationships. As stated in Section 6.2.2, we have two options for choosing the direction of edge in the R2RML file. We choose to translate both relations as outedges of Tweet.

R2RML mapping process The R2RML-F engine takes the Twitter relational database and the R2RML mapping file as input and generates the output file (.ttl) available online [8].

We study the one-to-many relationship between Person and Tweet. The direction of the edge in the mapped RDF graph is from referencing table to referenced table, from *Tweet* to *Person*, as explained in Section 2.5.1, precisely the case where the row contains a FK. The translation of one-to-many relationship is the same as in the DM, as depicted in Fig. 6.14. The privacy guarantees are therefore also the same.

6.5.2 - Motivating examples

We now present two motivating examples of two queries. We study the GS of these queries over the RDF graphs obtained from DM and/or default R2RML mapping.

Motivating Example 1: Consider query Q1: Find the maximum number of tweets tweeted by a single person (maximum outdegree of *tweeted* outedges). The GS of this query under QL-outedge privacy over the RDF graph obtained from DM in Section 6.5.1.1 and over the RDF graph obtained from this default R2RML mapping is 1, assuming that *tweetedby* $\in$ QL. The neighboring graphs differ by the QL-outedges of an arbitrary node. The *Tweet* node has exactly one *tweetedby* outedge (along with other outedges). So, one possible neighboring graph differs by the outedges of node *Tweet*. The privacy protection is protecting the *tweetedby* outedge that is protecting the author of one tweet.

Motivating Example 2: Consider query Q2: Count how many users "Alice" has referenced. The GS of this query under QL-outedge privacy over the RDF graph obtained from this default R2RML mapping is infinite, assuming *tweetedby* $\in$ QL and references $\in$ QL. The node *Person* has one *name* outedge with Literal value "Alice" (along with other outedges). The neighboring graphs differ by the QL-outedges of an arbitrary node. The *Tweet* node has exactly one *tweetedby* outedge plus some references outedges. So, one possible neighboring graph differs by the outedges of node *Tweet*. The number of references outedges could be unbounded and hence the GS is unboudned.

We want to protect all the tweets of a person. With DM and the default R2RML mapping we cannot get the choice what we want to protect. Using DM or default R2RML mapping will restrict the possible privacy models.

Our contribution is to write R2RML where we define our own mappings to control the privacy protection. To this end, we propose a model where one can choose the direction of the edge according to what to protect. This decision is to be taken by the person modeling

[7]https://github.com/sarataki/mapping/tree/main/defaultR2RML/r2rml.ttl
[8]https://github.com/sarataki/mapping/tree/main/defaultR2RML/output.ttl

the system. To do so, we manually write R2RML mapping tailored to our Twitter example to be able to control the privacy protection. In the following, we will explain how we achieve this goal.

6.5.3 - Proposed R2RML Mapping then applying QL-outedge privacy

For this Twitter example, to protect all the tweets of a person, we change the direction of edges for the one-to-many relationship between Person and Tweet. To do so, we again manually write a R2RML mapping file[9] in such a way that we decide the direction of the edge to be from Person to Tweet, from referenced table to referencing table. We take the same example as in Section 6.5.1.1 "In table Tweet: idtweet=30 and p_id=1", and plot a synopsis of the output of the R2RML mapping in Fig. 6.16. We get *tweeted* outedge represented as a predicate rdf:tweeted.

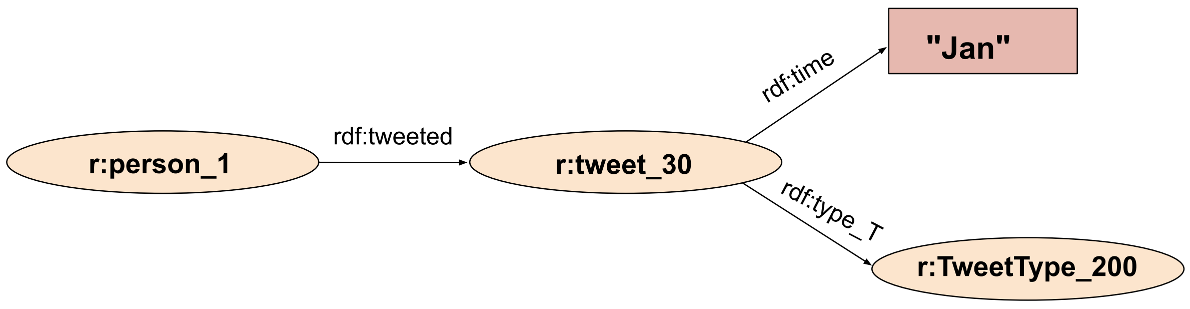


Figure 6.16: Proposed R2RML: translation of one-to-many relationship

After that, for the many-to-many relationships, we also choose the direction of the edge in the R2RML file according to what we want to protect. For References and HasEmotion, we

[9]https://github.com/saratak i/mapping/tree/main/propR2RML/r2rml.ttl

have two options to choose the direction of edge in the R2RML file, as mentioned earlier in Section 6.2.2, each providing different privacy guarantees:

For References:

1. The direction of edge from Tweet to Person. Privacy guarantee is protecting *references* outedges of node *Tweet*.
2. The direction of edge from Person to Tweet. Privacy guarantee is protecting *referencedby* outedges of node *Person*.

For HasEmotion:

1. The direction of edge from Tweet to Emotion. Privacy guarantee is protecting *hasEmotion* outedges of node *Tweet*.
2. The direction of edge from Emotion to Tweet. Privacy guarantee is protecting *ofhasEm* outedges of node *Emotion*.

We choose to protect the *references* and *hasEmotion* outedges of node *Tweet*. The R2RML mapping file is available online [10].

R2RML mapping process The R2RML-F engine takes as input the Twitter relational database, the R2RML mapping file, and generates the output file[11].

We present a synopsis of the translation of References. We refer back to the same example of Section 6.5.1.1 where we took the row with idtweet=30 and idperson=2 for References. This signifies that idtweet=30 *references* person with idperson=2. We draw a part of the RDF graph to visualize the R2RML mapping output in Fig. 6.17. We obtain simply that *Tweet* of idtweet=30 has an outedge *references* represented as a predicate rdf:references to the *Person* with idperson=2. As we can see, R2RML exposes many-to-many relationships as simple triples.

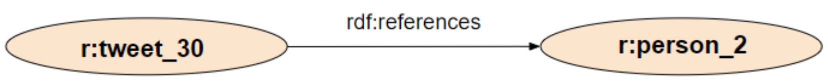


Figure 6.17: R2RML: translation of many-to-many relationship

Comparison between default R2RML and proposed R2RML mapping: The only difference is in the edge direction between node *Person* and *Tweet*. In the default R2RML mapping result, the RDF graph part is shown in Fig. 6.14, whereas in the proposed R2RML mapping, the RDF graph part is shown in Fig. 6.16.

[10] https://github.com/saratakі/mapping/tree/main/propR2RML/r2rml.ttl
[11] https://github.com/sarataki/mapping/tree/main/propR2RML/output.ttl

Now, we refer back to the same two motivating examples of Section 6.5.1.2. We study again the GS with our proposed R2RML mapping.

Motivating Example 1 Consider query Q1: find the maximum number of tweets tweeted by a single person (maximum outdegree of *tweeted* outedges). Applying QL-outedge privacy over the RDF graph obtained from our R2RML mapping means protecting the QL-outedges of a node. In what follows, we assume *tweeted* $\in$ QL (otherwise, it is immediate that the GS of Q1 is 0).

For computing the GS of Q1, one possible neighboring graph differs by the QL-outedges of node *Person*. If the number of *tweeted* out-edges of *Person* is unbounded, then the GS is infinite. However, let us consider that we are projecting the graph with the projection method introduced in our previous work [33] with maximum QL-out-degree bound D. Node *Person* will have maximum D QL-outedges. Therefore, the GS of Q1 is bounded by D compared to the GS of the same query over the RDF graph obtained from the DM, which is 1.

Nevertheless, the privacy protection over the RDF graph obtained from our R2RML mapping protects all the tweets of a person compared to the privacy protection over the RDF graph obtained from DM which protects the author of one tweet.

Motivating Example 2 Consider query Q2: Count how many users "Alice" has referenced. For computing the GS of Q2, one possible neighboring graph differs by the QL-outedges of some nodes. *Tweet* (resp. *Person*) nodes may have an arbitrary number of references (resp. tweeted) outedges.

Again, assuming *tweeted* $\in$ QL or references $\in$ QL, we have GS of Q2 equal to ∞. Let us consider that we are projecting the graph with the projection method introduced in our previous work [33] with maximum QL-out-degree bound D. Node Tweet will have maximum D outedges. Therefore, the GS of Q2 is bounded by D.

6.5.4 - Defining neighborhood in relational database

We have mapped the relational database to RDF and applied QL-outedge privacy in RDF in Section 6.5.3. Now, we come back to the relational database to define the neighborhood for QL-outedge privacy. Inspired by [160], we present a notion of neighboring databases which captures privacy policies and key constraints to some extent and do present the neighborhood equivalent to the QL-outedge privacy model.

We define the neighborhood in the mapped RDF graph (according to default or proposed R2RML mapping). Our approach permits the data owner to flexibly designate which nodes or edges in RDF need privacy.

Accordingly, for defining the neighborhood definition in the relational database, one designates the node's corresponding relation in the relational schema that need privacy. This relation will be called the primary private relation. In a relational database, we define possible secondary private relations as relations that refer directly to the primary relation through FK constraints. We select secondary relations from the possible secondary relations (if they exist) if the direction of the linking edge in the mapped RDF graph is from the node that corresponds to the primary private relation (outedge of the private node).

The formal definition is as follows:

Definition 6.3 (Referencing relation)**.** *Let **R** be the database schema. The designated primary private relation is R_p, and the possible secondary private relations are relations that have a direct FK referencing R_p. Let **I** be a database instance over **R**. For any $R \in$ **R**, let **I**(R) be the relation instance of R in **I**. The referencing relationship over the records is defined as follows: for $t \in$ **I**(R) and $t' \in$ **I**(R'), we say that t' reference t if R' references R and the FK of t' equals the PK of t.*

Definition 6.4 (QL-Outedge neighborhood)**.** *The QL-outedge neighborhood in relational database can be defined as select a record t in the primary private relation, keep the PK, delete the entire row, cascade deletion to the selected secondary relations according to the following list:*

- *If the secondary relation represents a many-to-many relationship, then we cascade deletion to the entire row.*
- *If the secondary relation is in a one-to-many relationship or in a one-to-one relationship with the primary one (the PK of the primary relation appears as a FK in the secondary relation), then we restrict the deletion in the secondary relation to just the FK entry and not the entire row.*

On the other hand, if we have changed the direction of edge in the R2RML file for the one-to-many relationship to be from referenced table to referencing table as we did in our proposal in Section 6.5.3, then when choosing the referencing table as a primary private relation, we define the QL-outedge neighborhood in relational database as select a record t in the primary private relation, keep the PK, keep the corresponding FK in the primary relation, delete the entire row, cascade deletion to the selected secondary relations as defined in the previous list.

Case Study: In the following, we present the QL-outedge neighborhood in the mapped RDF graph obtained from our proposed R2RML mapping described in detail in Section 6.5.3. We present a case study to illustrate our proposed QL-outedge neighborhood in a relational database and to illustrate the decision on which relations to be selected as secondary (if they exist). In this case study, we define the Ql-Outedge neighborhood in the mapped RDF graph as select an arbitrary node and delete all of its QL-outedges (Note that another case study can be done to define Ql-outedge neighborhood as select an arbitrary node and add all of its QL-outedges). We focus on particular types of nodes to define the corresponding neighborhood in the relational database. We also consider QL = L.

6.5.4.1 - Case 1- Choose node *Person*

Neighborhood definition in RDF In QL-outedge privacy, if the true world is a given RDF graph G, the neighboring possible worlds can be obtained by deleting the QL-outedges of node *Person*. Figure 6.18 shows a synopsis of the mapped RDF graph, and Fig. 6.19 shows a neighboring graph of the mapped RDF graph.

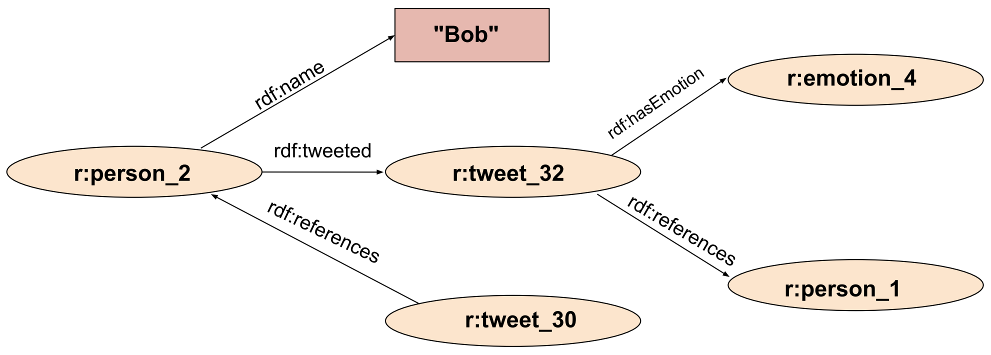


Figure 6.18: Case 1: Synopsis of the mapped RDF Graph

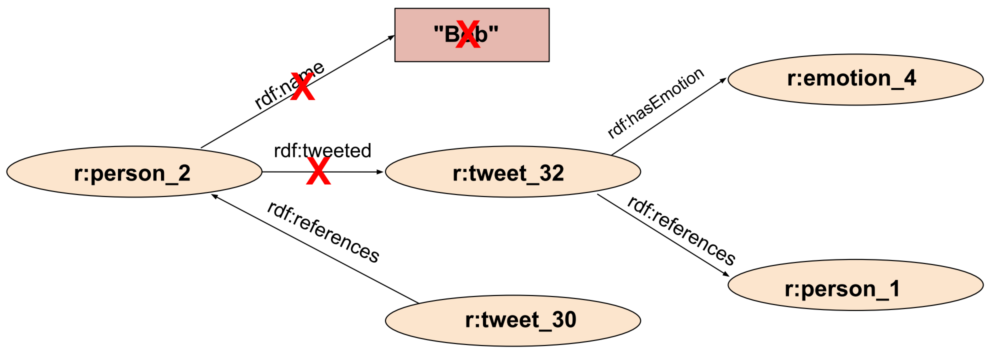


Figure 6.19: Case 1- QL-Outedge neighboring graph

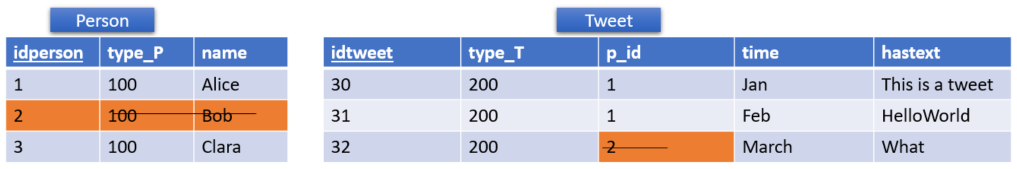
Neighborhood definition in relational database Accordingly, for defining the neighborhood definition in the relational database, we specify the privacy policy P = (Person, ε), where Person is the primary private relation. idperson appears as a FK in Tweet and References. Then the possible secondary private relations are Tweet and References.

For Tweet, we have specified in the R2RML file the direction of edge to be from Person to Tweet, then it is an outedge of *Person*, and then it is a secondary relation.

For References, we have specified in the R2RML file the direction of edge to be from Tweet to Person. It is not an outedge of *Person* and then not a secondary relation. Hence, Tweet is the only secondary relation.

The neighborhood in a relational database can be defined as select a record t in Person, keep the PK idperson, delete the Person row (because QL=L we do not consider specific labels), and cascade deletion to Tweet i.e., just delete p_id entry in Tweet. We just delete p_id entry in Tweet because we have one-to-many relationship between Person and Tweet and we have specified the direction of edge to be from Person to Tweet. So, we will not delete the entire row of Tweet.

Fig. 6.20 shows a neighboring instance of the original database under privacy policy P = (Person, ε). It is the equivalent of the RDF of Fig. 6.19. Notice that we are keeping the idperson=2 in Person, deleting the Person row, and just deleting the p_id=2 entry in Tweet. In this case, neighboring databases differ in the primary private relation Person and the secondary private relation Tweet.


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

Figure 6.20: Neighboring instance under Person policy

6.5.4.2 -Case 2- Choose node *Tweet*

Neighborhood definition in RDF In QL-outedge privacy, if the true world is a given RDF graph G, the neighboring possible worlds can be obtained by deleting the QL-outedges of node *Tweet*. Since QL=L, the privacy guarantee protects the outedges of node *Tweet*. Fig. 6.21 shows a synopsis of the mapped RDF graph, and Fig. 6.22 shows a neighboring graph of the mapped RDF graph.

Neighborhood definition in relational database Accordingly, for defining the neighborhood definition in the relational database, we specify the privacy policy P = (Tweet, ε), where Tweet is the primary private relation. idtweet appears as a FK in References and hasEmotion. Then the possible secondary private relations are References and HasEmotion. For both, References and HasEmotion, we have chosen option 1 in the R2RML file. Then, the direction of edge is from Tweet to Person and from Tweet to Emotion in References and HasEmotion,

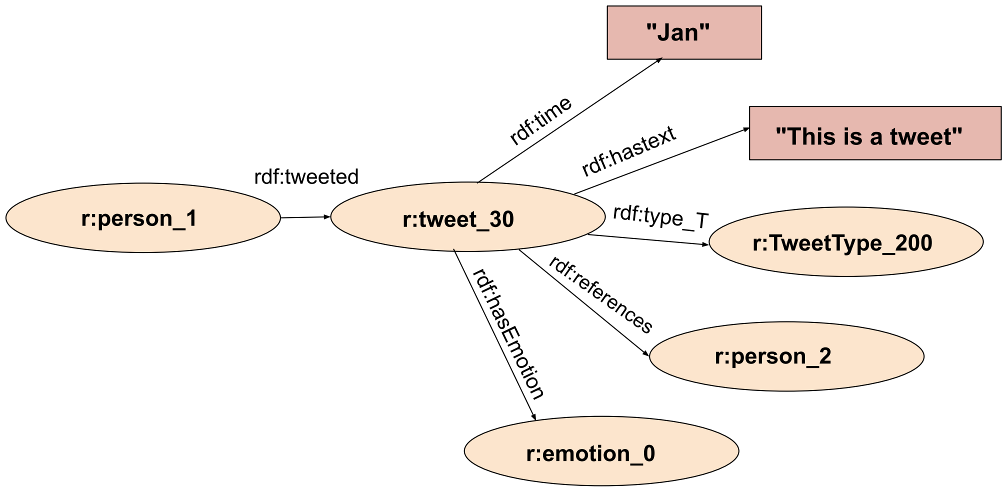


Figure 6.21: Case 2- Synopsis of the mapped RDF Graph

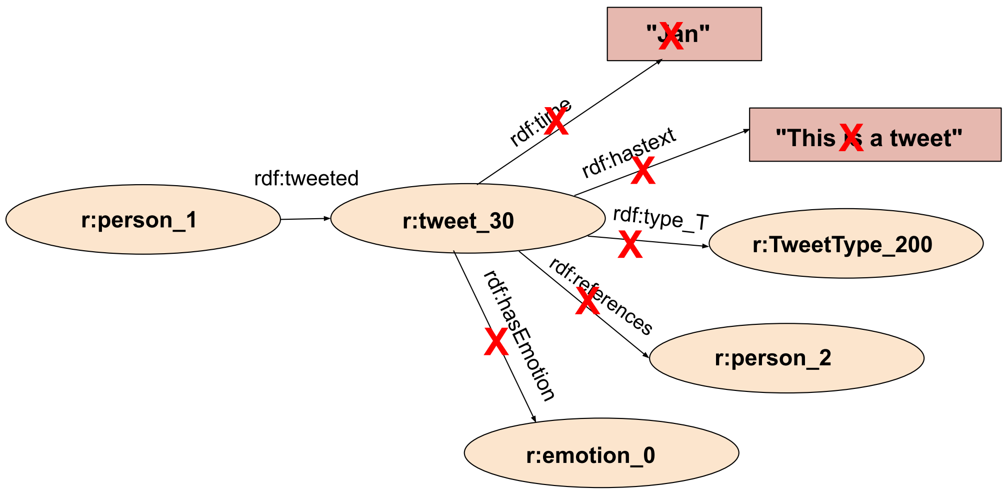
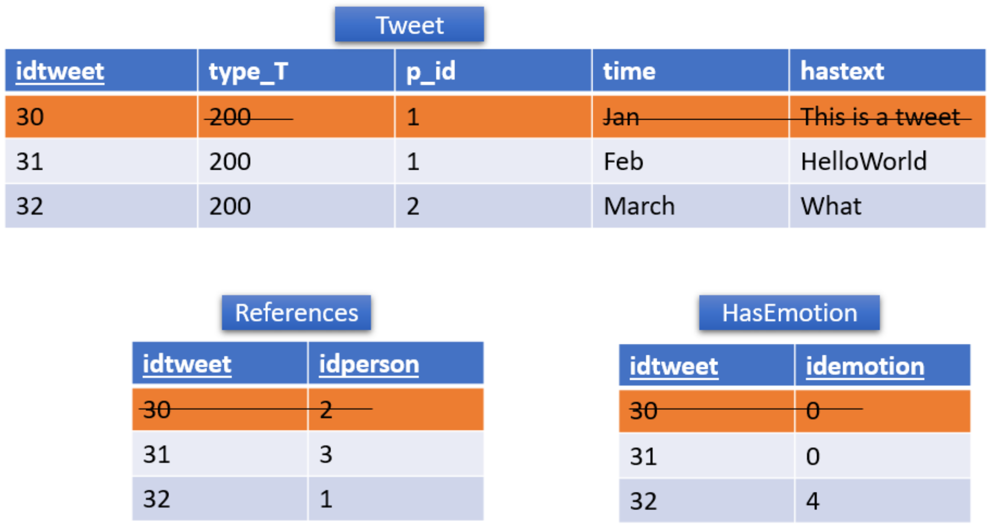


Figure 6.22: Case 2-QL-outedge neighboring graph

respectively. Hence, both References and HasEmotion are secondary relations.

The neighborhood in relational database can be defined as select a record t in Tweet, keep the PK idtweet, delete row Tweet except the p_id entry, and cascade deletion to References and HasEmotion i.e. delete the entire row that corresponds to this idtweet. The reason why we keep p_id entry because as stated before, we have a one-to-many relationship between Person and Tweet, and we have changed the direction of edge to be from Person to Tweet.

Fig 6.23 shows a neighboring instance of the original database under privacy policy P = (Tweet, ε). In this case, neighboring databases differ in the primary private relation Tweet and the secondary private relations References and HasEmotion.



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

References

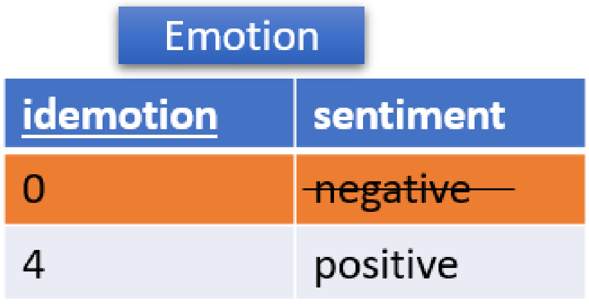
idtweet	idperson
~~30~~	~~2~~
31	3
32	1

HasEmotion

idtweet	idemotion
~~30~~	~~0~~
31	0
32	4

Figure 6.23: Neighboring instance under Tweet policy

6.5.4.3 - Case 3- Choose node Emotion



Emotion

idemotion	sentiment
0	~~negative~~
4	positive

Figure 6.24: Neighboring instance under Emotion policy

Neighborhood definition in RDF In QL-outedge privacy, if the true world is a given RDF graph G, the neighboring possible worlds can be obtained by deleting the QL-outedges of node *Emotion*. Since QL=L, the privacy guarantee protects the outedges of node *Emotion*. Fig. 6.25 shows a synopsis of the mapped RDF graph, and Fig. 6.26 shows a neighboring graph of the mapped RDF graph.

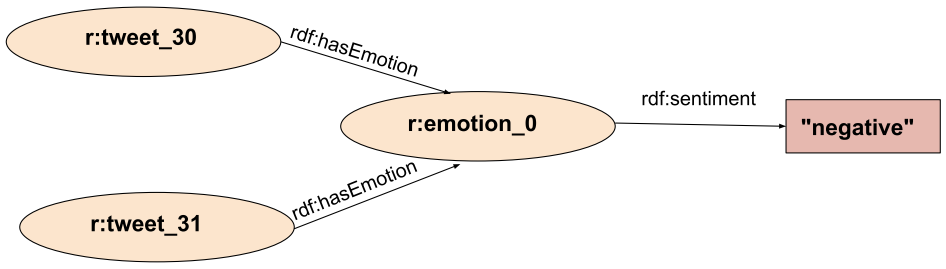


Figure 6.25: Case 3- Synopsis of the mapped RDF Graph

Neighborhood definition in relational database Accordingly, for defining the neighborhood definition in the relational database, we specify the privacy policy P = (Emotion, ε), where Emotion is the primary private relation. idemotion appears as a FK in HasEmotion. Then the possible secondary relation is HasEmotion. For HasEmotion, we the direction of the edge is from Tweet to Emotion. Hence, it is not a secondary relation.

The neighborhood in the relational database can be defined as select a record t in Emotion, keep the PK idemotion, delete the row. Fig 6.24 shows a neighboring instance of the original database under privacy policy P = (Emotion, ε). In this case, neighboring databases differ in only the primary private relation Emotion.

In conclusion, we present a case study where we define QL-outedge neighborhood in the mapped RDF graph obtained from our proposed R2RML mapping. We focus on specific types of nodes in the mapped RDF graph. Then, we could define the corresponding QL-outedge neighborhood in the relational database. When mapping relational databases to RDF, we propose a model where one can choose what they want to protect. This is done by deciding the edge direction in the R2RML mapping. The decision on which direction to choose will affect the creation of secondary relations (from the possible secondary relations, we will select the

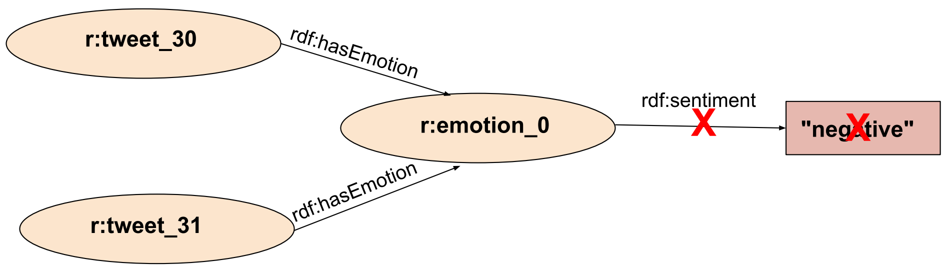


Figure 6.26: Case 3-QI-Outedge neighboring graph

secondary if they exists) and, therefore, the QI-outedge neighborhood definition in the relational database.

6.6 - Implementation

A core element in our two approaches presented in Sections 6.3, 6.4 and 6.5 is to map the relational database to RDF. As stated earlier in Section 6.2, we use DB2Triples for the DM and R2RML-F for the R2RML mapping.

Antidot no longer updates DB2Triples. So, we updated the code and installed the necessary libraries/packages. In details, we install db2triples from Maven site[12]. The needed dependencies are: Commons-cli, Commons-logging and MySQL Connector and openRdf sesame. We install them from Maven except for the OpenRdf Sesame[13] which is a Java framework for processing and handling RDF data. OpenRDF Sesame it changed its name, after officially forking into the Eclipse project RDF4J[14] in May 2016. We install a single jar file for sesame available online[15]. Regarding R2RML-F, we also update the code and install the needed dependencies. The updated code for both DB2Triples and R2RML-F is available at :https://github.com/sarataki/mapping/tree/main/code.

[12]https://mvnrepository.com/artifact/net.antidot/db2triples
[13]http://www.openrdf.org/
[14]https://rdf4j.org/
[15]https://sourceforge.net/projects/sesame/

The DM process is simple, as depicted in Fig. 6.4. It takes as input the relational database parameters and outputs the generated mapped database.

The R2RML mapping process, as shown in Fig. 6.5, takes as input the relational database parameters, output file format, and an R2RML mapping file tailored to the database schema. Here lies part of our contribution (Refer to Section 6.5) in which we choose what to protect by leveraging R2RML mapping capabilities to control privacy protection. To accomplish this objective, we write manually R2RML mapping tailored to our Twitter example to be able to control the privacy protection.

6.7 - Conclusion

In this chapter, we showed that a popular privacy model in the relational database word sometimes translates to typed-node DP in RDF, a natural adaptation of node DP 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. Once we are in the RDF world, we can use the results of Chapter 5 to reduce the GS of some queries using the projection algorithms.

Furthermore, we propose how to define neighborhoods in relational databases equivalent to the concept of QL-Outedge privacy, which was previously defined over RDF graphs. We believe this approach is particularly suitable to the context of RDF setting, offering meaningful semantics for the neighborhoods.

7 - Graph Rewriting Primitives for Semantic Graph Databases Sanitization

Contents

7.1 - Motivation and approach

In this chapter, we consider the sanitization of a *graph databases* prior to their release. During their sanitization, graph databases are transformed following graph transformations that are usually described informally or through ad-hoc processes.

In general, and despite the importance of graphs in databases and ontology representations, the use of formal graph rewriting techniques to model database evolutions is seldom studied.

Formal graph rewriting techniques are usually based on *category theory*, an abstract way to deal with different algebraic mathematical structures and the relationships between them. Algebraic approaches of graph rewriting allow a formal yet visual specification of rule-based systems characterizing both the effect of transformations and the contexts in which they may be applied. To the best of our knowledge, there exists no proposal relying on algebraic graph rewriting to support graph sanitization.

This chapter is the first effort to bridge the gap between rigorous algebraic graph rewriting and graph sanitization. Its content is under review for publication in the journal *Computer Science and Information Systems*.

Sketch of our approach We formalize a language of basic operators using attributed graph rewriting rules to serve as a basis for different graph sanitization mechanisms. We formalize and implement basic operators using AGG – The Attributed Graph Grammar System [171], one of the most mature development environments supporting the definition and application of typed graph rewriting systems [172]. These operators demonstrate the feasibility of the approach and should be enriched with other operators to build a library of operators supporting various anonymisation schemes. The code, and examples of how sanitization algorithms can be built with this approach is available at `https://github.com/ceichler/granon/`.

We choose to focus on eight basic operators that create nodes, delete nodes, copy, cut, or merge edges, randomize the targets of a relation, and help better identify sets of nodes. Each of these operators can be represented as a single graph rewriting rule. Together, they can be combined into procedures expressive enough to have allowed the implementation of two privacy schemes: randomization providing LDP guarantees and sensitive attribute anatomization. The chapter is organized as follow:

- We present the formalism of graph databases and graph rewriting we base our approach on, and the adopted visual conventions that stem from the AGG rewriting tool. We also provide a running example for the rest of the chapter (Section 7.2);
- We describes the semantic and syntax of eight basic operators, among which `JoinSet` that processes the keywords WHERE and EXCEPT for restriction and exclusion in the scope of the other operators (Section 7.3);
- We presents the description of two privacy procedures through our operators (Section 7.4);
- We presents the results of a preliminary experimental evaluation made by applying one of the procedures of Section 7.4 to a sampling of the Sentiment140 dataset containing information about tweets (Section 7.5);
- Finally, we conclude and highlight the future work (Section 7.6);

7.2 - Background and setting

7.2.1 - Attributed multigraphs

We consider databases to be modelled as attributed oriented multigraphs. In such models, it is customary for nodes and edges to have properties (among a finite set) and attributes (as words on a signature). In [173], RDFS databases are modelled as a typed graph with 4 node types and 6 edge types. These types are inherent to RDF and thus the model can not be applied natively to arbitrary graph databases (*e.g.* neo4j).

We argue that considering a single node type and a single edge type having a single attribute (named *att* and *prop*, respectively) is in fact at least as expressive. Indeed, typing and additional properties can be encoded via special kinds of relation (see Section 7.2.2). We believe this model to be able to capture most –if not all- graph database representations.

7.2.2 - Running example

As a running example, we consider a graph database that contains information on travels, both professional and personal.

It has nodes for relevant entities, people and travels, whose attributes are an identifier. It also has nodes for every literal describing informations on those entities, e.g. last name, first name and address for people, date and destination for travels. We do not differentiate nodes representing entities or literals.

Its edges describe both relations between entities, e.g. "this person participated in this travel", represented by an edge of attribute ''attends'', but also relations between entities and their information, e.g. "this person's name is in this literal", represented by an edge of attribute ''name''. Typing falls within this second case e.g. "this node is a person" or "this literal is a city", represented by an edge of attribute ''type''.

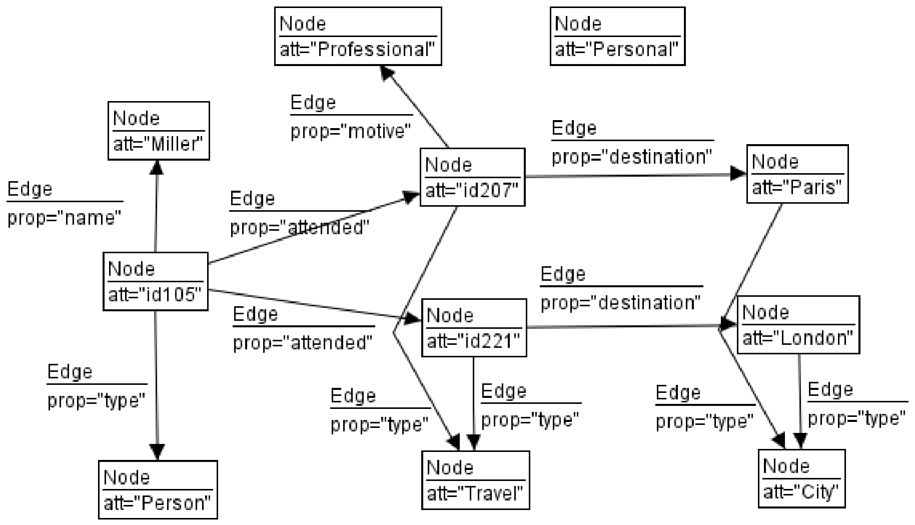


Figure 7.1: Running example: instance of a database

An example of such a database is provided in Fig. 7.1. In this instance, id105 (named Miller) attended travel id207 to Paris for professional reasons.

We give two target examples consisting in the application of two privacy mechanisms in this database, whose implementation will be shown to be possible in Section 7.4. In these examples, we will note that the described procedures should only apply on specific nodes and relations.

- We want to provide plausible deniability with regard to the relation "destination" between personal travels and cities. To do so, we want to randomize this relation for every personal trip specifically, to preserve privacy, with a bias towards correct answers to preserve utility. This corresponds to guaranteeing local differential privacy (LDP [106,107]) on trips with a "motive" edge leading to "personal". More precisely, we want to modify the database such that querying it to output the destination of personal trips would be locally differentially private.

- For professional trips, we want to obfuscate the relation "destination" between travels and cities, as to hide precise dates and frequence of collaboration between the database's company and its collaborators, for instance. This can be done, for instance, by grouping trips in certain cities together (*e.g.* "Paris", "Bordeaux", "Toulouse" all grouped in the more nebulous group "France") and rerouting the "destination" edges towards those groups rather than a precise value. This would means that we want to apply anatomization [30, 149] where the "destination" attributes of travels with attribute "motive" set to "professional" is considered sensitive.

7.2.3 - Graph Rewriting Rules

We adopt the Single Push Out (SPO) formalism [174], an algebraic approach based on category theory, to specify rewriting rules. In the SPO approach, a rule formalize both the applicability of a rule and the effects of its application. We use two extensions of SPO to specify additional application conditions and restrict their applicability: *Negative Application Conditions* (NACs) [175] and *Positive Application Conditions* (PACs).

These rules may be *fully specified graphically*, enabling an easy-to-understand yet formal graphical view of the graph transformation.

The SPO approach relies on graph rewriting rules defined by two graphs – the Left and Right Hand Side of the rule, denoted by L and R – and a partial morphism m from L to R (*i.e.*, an edge-preserving morphism m from a subgraph of L to R).

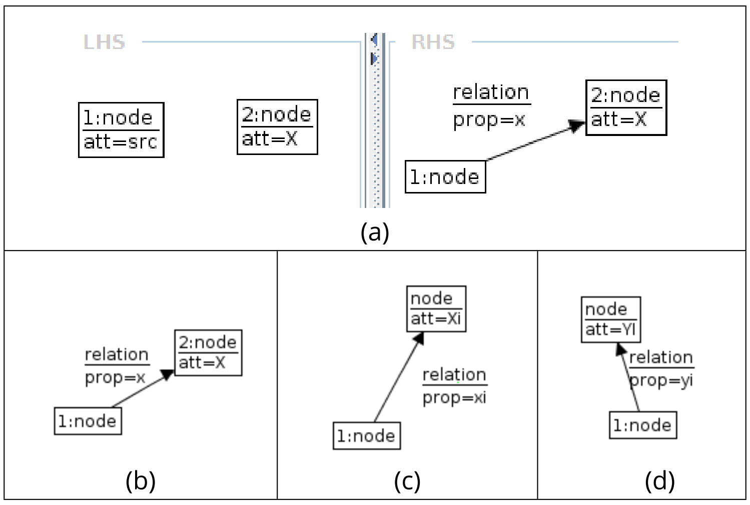


Figure 7.2: JoinSet: Creating an edge *x* from *src* to *X*: (a) SPO core (b) NAC forbidding duplicate edges, (c) NAC related to an EXCEPT clause, (d) PAC related to a WHERE clause.

Example 7.1. *Fig. 7.2 formalizes the JoinSet operator. Its SPO core is illustrated in Fig. 7.2a; its L is composed of two nodes attributed src and X, respectively; its R also has two nodes plus a new edge attributed x from one to the other. Note that the attribute of a node of R is not represented. In general, this stems from three possibilities: (i) it does not matter, e.g. an unattributed node in L, PAC or NAC will match any node; (ii) it can be infered, e.g. a node in R* *or NAC has the same attributes as the node it is matched with in R; (iii) a node in R is created without an attribute value.*

The partial morphism from L to R is specified in the figure by tagging graph elements - nodes or edges - in its domain and range with a numerical value. An element with value i in L is part of the domain of m and its image by m is the graph element in R with the same value i. For instance, in Fig. 7.2a, the notation 1: for the node attributed src in L and the unattributed node in R indicates that they are mapped through m. Hence, it can be infered that the node with an implicit attribute actually has an attribute src.

By convention, in rules, an attribute value within quotation mark (e.g. "Paris") is a fixed constant, while *a value noted without quotation mark (e.g. src) is a variable that is either a wildcard matching any value or whose value is given as input*. A variable appearing solely in the R of a rule (*e.g.* *x*) must be given as input.

A graph rewriting rule $r = (\mathsf{L}, \mathsf{R}, m)$ is applicable to a graph G iff there exists a total morphism $\tilde{m} : \mathsf{L} \rightarrow G$. The result of the application of r to G w.r.t. $\tilde{m}$ is the object of the push-out of the diagram composed by L, R, G, m, and $\tilde{m}$. Informally, the application of r to G with regard to $\tilde{m}$ consists in modifying G by (1) removing the image by $\tilde{m}$ of all elements of L that are not in the domain of m (*i.e.*, removing $\tilde{m}(\mathsf{L} \backslash Dom(m))$); (2) removing all dangling edges (*i.e.*, deleting all edges that were incident to a node suppressed in step (1)); (3) adding an isomorphic copy of all elements of R that are not mapped through m.

Example 7.2. *The SPO rule specified in Fig. 7.2a is applicable to any couple of nodes whose attributes match src and X. If a parameter is not specified, it can match any attribute, otherwise it matches attributes having the same value.*

For example, with the input (X="Miller", x = "name"), any node n_1 can match the one attributed src but the second node in L can only be mapped to the node n_2 whose attribute is "Miller". The application of the rule consists in adding a "name" edge from n_1 to n_2.

NACs* and *PACs are well-studied extensions that restrict rule application by forbidding and demanding certain patterns in the graph, respectively. EXCEPT clauses are encoded through NACs, while WHERE clauses are encoded through PACs. A NAC or a PAC for a rule r is defined as a constraint graph which is a super-graph of its L. An SPO rule $r = (L, R, m)$ with NACs and PACs is applicable to a graph iff: (i) there exists a total morphism $\tilde{m} : \mathsf{L} \rightarrow G$ (this is the classical SPO application condition); (ii) for all NAC N associated with r, there exists *no* total morphism $\bar{m} : N \rightarrow G$ whose restriction to L is $\tilde{m}$; (iii) for all PAC P associated with r, there exists a total morphism $\bar{m} : P \rightarrow G$ whose restriction to L is $\tilde{m}$.

By convention, since NACs and PACs are super-graphs of L, unnecessary parts of L are not depicted when illustrating these constraint graphs. Graph elements common to L and NAC or PAC are identified by a numerical value similarly to elements mapped by m.

Example 7.3. *Figures 7.2b, 7.2c, and 7.2d represent two NACs and a PAC associated to the SPO core of Fig. 7.2a, respectively. The first specifies that there must not already be an edge x between nodes src and X. This ensures that the recursive application of the rule w.r.t. all* *possible morphisms terminates. The second and third applications conditions are similar yet opposite. Figure 7.2c (resp. Fig. 7.2d) specifies a NAC (resp. a PAC) that forbids (resp. ensures) the presence of an edge xi (resp. yi) between src and Xi (resp. Yi). They correspond to* except *and* where *clauses, respectively. For instance, the parametrization of the SPO rule discussed in example 7.2 gives the name "Miller" to any node n_1, which would be semantically unsound. One may want to ensure that n_1 can only matches unnamed persons. This can be done by (1) parametrizing the PAC with (yi, Yi) = ("type", "Person") to guarantee type soundness and (2) parametrizing the NAC with xi = "name", ensuring that n_1 does not already have a name. Note that, since Xi is left unspecified, it acts as a wildcard and can therefore be matched with any name.*

With such a parametrization, the rule would in fact not be applicable to the graph of Fig. 7.1 as any morphism satisfying the PAC violates the NAC and vice versa.

A rewriting procedure –or rule sequence– as we consider it here is a succession of steps. Each step is the application of a rewriting rule as long as the rule applies or a specified number of times. We consider that *when a rule is applicable w.r.t. several morphism, one is chosen uniformly at random.*

7.3 - Our Language

This section introduces the language we build and its design choices: procedures composed of elementary operators applied to simple cases, and the special operator `JoinSet` that allows it to preserve its expressiveness.

7.3.1 - Design principle

Simple operators Our main design choice is to keep the basis of our language as simple as possible. We propose a set of eight operators, each described by a single graph rewriting rule, with an easy-to-match pattern. They allow to create and delete nodes, copy, modify, cut or merge edges, randomize the sources or destination of a relation. All these operators work on very basic patterns, and the main role of the eighth operator, `JoinSet`, is to reduce complex application cases to the simple cases handled by our operators.

Need for pre-processing Our language's goal is to support the specification of privacy procedures and mechanisms that will apply operators (e.g. randomization) on some particular sets of nodes and edges. In its general form, we would expect an instruction to be the combination of an operator, a set of sources (or subject), a set of destinations (or objects), and potentially additional operator- or procedure-specific requirements (e.g. set of eligible new targets for randomization, or lists of identifiers and sensitive attributes).

As discussed in Section 7.2.2, these sets can be defined as types, can exclude types, or can require the presence or absence of a certain property at a certain value. It is possible to specify these kinds of restrictions directly in graph rewriting rules, using L, PACs, and NACs. However, this would have the important drawback of necessitating numerous versions of the same rule and/or complicating the pattern matching phase. We therefore believe a pre- and post-processing procedure to construct temporary add-hoc sets to be necessary.

Node and Edge identification As a base case, we use simple identification tools for nodes and edges.

To identify *nodes*, our language considers either the label of the node (*e.g.* the node.s labelled "id207"), or the existence of labels with a given edge towards nodes of a certain label (*e.g.* the node.s with edges "type" heading to nodes labelled "City"). As a shorthand, we sometimes say that the nodes with a label S and an edge labelled p towards a node labelled O as "matching (S, p, O)". Such a triplet (S, p, O) can be called a **set**, and identified with a single letter $\mathcal{X}$.

We note that we can specify two special values in these sets:

- $*$ corresponds to any label we want
- `null` corresponds to no expectation, *i.e.*not only any label but the existence is not necessary either

For instance, $(\text{City}, null, null)$ would match any node whose label is "City", whether they have an outgoing edge or not, which includes one node in our running example. $(\text{City}, *, *)$ would match any node whose label is "City" with any outgoing edge at all, which includes no node of our example. Note that `null` differs from $*$ only when both p and O are equal to `null`. Indeed, $(\text{City}, *, null)$ would not constraint O but imposes the existence of some p, and thus of some O.

To identify *edges*, our language considers their labels and identifiers of their source or target (*e.g.* all edges labelled "destination" from a node matching $(*, \text{"type"}, \text{"Travel"})$ to a node matching $(*, \text{"type"}, \text{"City"})$).

Pre-processing using `JoinSet` To reduce a wide variety of operators' scopes and targets to the basic case, we only define operators on very simple cases, i.e. by considering simple node sets, and when applicable the edges that link them. For this to not hinder the expressiveness of our language, we need a way to create or populate sets that corresponds to nodes matching intricate conditions. For this, we use the following instruction:

`JoinSet` (x, X) `Where` $\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ `Except` $\{\mathcal{Y}_1, \ldots, \mathcal{Y}_m\}$

This primitive matches nodes that match every $\mathcal{X}_i$ in the `Where` section and match none of the $\mathcal{Y}_i$ in the `Except` section. When it does, it create from it an edge labelled x towards a node labelled X.

Graph rewriting primitives: We present here the few primitives, formalized and implemented directly using AGG, that create or delete nodes, copy, divide or merge edges, and randomize specific relations[1].

[1]Rules specifications are available at `https://github.com/ceichler/granon/blob/master/anonOperator.ggx`

- `NewNode`(X) creates a new node labelled X.
- `DeleteNode`$(\mathcal{X})$ matches all nodes matching $\mathcal{X}$ and deletes them.
- `JoinSet` (x, X) `Where` $\{\mathcal{X}_1, \dots, \mathcal{X}_n\}$ `Except` $\{\mathcal{Y}_1, \dots, \mathcal{Y}_m\}$ (explained above).
- `EdgeCopy`$(\mathcal{S}, p, \mathcal{O}, p')$ matches couples of source nodes matching $\mathcal{S}$ and destination nodes matching $\mathcal{O}$, where there is an edge from the source to the destination labelled p.

  When a match is found, creates an edge labelled p' from the source to the destination.
- `EdgeReverse`$(\mathcal{S}, p, \mathcal{O}, p')$ matches couples of source nodes matching $\mathcal{S}$ and destination nodes matching $\mathcal{O}$, where there is an edge from the source to the destination labelled p.

  When a match is found, creates an edge labelled p' from the destination to the source.
- `EdgeCut`$(\mathcal{S}, p, \mathcal{O}, p_I, M, p_O)$ matches pairs of source nodes matching $\mathcal{S}$ and destination nodes matching $\mathcal{O}$, where there is an edge from the source to the destination labelled p.

  When a match is found, creates a new intermediary node of label M. Then, it creates an edge labelled p_I from the source to the intermediary, an edge labelled p_O from the intermediary to the destination, and finally it deletes the edge p from the source to the destination.
- `EdgeChord`$(\mathcal{S}, p_I, \mathcal{M}, p_O, \mathcal{O}, p)$ is the converse of `EdgeCut`, and matches triplets of source nodes matching $\mathcal{S}$, intermediary nodes matching $\mathcal{M}$ and destination nodes matching $\mathcal{O}$ where there is an edge from the source to the intermediary labelled p_I and there is an edge from the intermediary to the destination labelled p_O.

  When a match is found, creates an edge labelled p from the source to the destination.
- `RandomTarget`$(\mathcal{S}, p, \mathcal{O}, \mathcal{T})$ matches all edges labelled p between a source matching $\mathcal{S}$ and a destination matching $\mathcal{O}$, and it reroutes this edge by picking a new target uniformly among nodes matching $\mathcal{T}$.

Procedures: A procedure is a sequence of instructions, executed in order. They can be as simple as intermediary operators, or complex enough to describe graph manipulations guaranteeing certain types of privacy. For instance, a procedure `DeleteEdge`$(\mathcal{S}, p, \mathcal{O})$ meant to delete all edges labelled p between a source matching $\mathcal{S}$ and a destination matching $\mathcal{O}$ would be:

```
1: EdgeCut$(\mathcal{S}, p, \mathcal{O}, p_I, "ToBeDeleted", p_O)$
2: DeleteNode$("ToBeDeleted", null, null)$
```

This procedure first cuts every occurrence of p between $\mathcal{S}$ and $\mathcal{O}$ in two with a new node labelled "ToBeDeleted" (line 1). Then, it immediately deletes all those "ToBeDeleted" nodes (line 2), thus erasing the original edges.

7.4 - Privacy Procedures

This section introduces procedures that provide privacy guarantees: Local Differential Privacy [107] and anatomization [149]. These procedures are expressed as combinations of our primitives.

7.4.1 - Randomized edge selection for local differential privacy

In this section we propose a randomizing procedure that can be used to achieve local differential privacy (LDP) as defined and used in [106, 107] (see Section 3.4.3). Assuming a query outputing the value.s of a property for a particular node or a set of nodes, we wish to modify the graph to make the query LDP.

To achieve ε-LDP for a set R of relations, our random operator should therefore transform each $(s,t) \in R$ into a relation $(s,t') \in R'$ under the appropriate staircase probability distribution:

$$P(t') = \begin{cases} 0 & t' \notin T \\ \frac{K}{|T|-1+K} & t' = t \\ \frac{1}{|T|-1+K} & t' \in T \wedge t' \neq t \end{cases}$$

with K an integer approximation of e^{ε}, $K = \lfloor e^{\varepsilon} \rfloor$.

To do so, we pick a new target at random, but, to obtain a staircase distribution from a uniform distribution –used to choose the morphism with regard to which the transformation rule is applied–, we skew the odds by creating $K-1$ dummies. Picking a dummy as a target should ultimately result as giving the true answer to recreate a staircase distribution. The procedure, defined in Alg. 4, and detailed thereafter, has the following arguments:

- $\mathcal{S} = (X_S, s, S)$ to match the sources of the relation to randomize
- p to match the edges of the relation to randomize
- $\mathcal{O} = (X_O, o, O)$ to match the destinations of the relation to randomize
- K the factor by which correct answers are more likely than other values

Recurring example: We illustrate the steps of this procedure in a recurring example presented in Figures 7.3 to 7.8. We present a $\ln(4)$-LDP-providing randomization of persons attending travels, i.e. the procedure LDP((type,Person),attended,(type,Travel),3).

In this example, nodes and edges in black are definitive data that started or are meant to remain in the graph. Nodes and edges in blue are temporary nodes that are only used as intermediary elements in the rewriting process (e.g. the Dummy nodes). Nodes and edges in bold/thick are recently created (e.g. the "attended" edge in Figure 7.6). Nodes and edges that are dashed with reduced opacity are recently deleted (e.g. the "Dummy" nodes in Figure 7.8).

Initialisation: We start by creating a bias for the correct value, by creating $K-1$ dummy nodes thanks to the `NewNode` operator. This rule is to be repeated as many times as required

Algorithm 4: Procedure LDP(S,p,O,K)

```
1  for $i$ from $1$ to $K-1$ do
2      NewNode$(Dummy)$
3  end for
4  JoinSet$(o, O)$Where$\{(Dummy, *, *)\}$Except$\{\}$
5  EdgeCut$(\mathcal{S}, p, \mathcal{O}, p_I, \text{"}Intermediary\text{"}, p_O)$
6  EdgeCopy$((*, \text{null}, \text{null}), p_O, (*, \text{null}, \text{null}), p_N)$
7  RandomTarget$((*, \text{null}, \text{null}), p_N, (X_O, o, O), (*, o, O))$
8  EdgeChord$(\mathcal{S}, p_I, (*, p_N, Dummy), p_O, \mathcal{O}, p)$
9  EdgeChord$(\mathcal{S}, p_I, (*, \text{null}, \text{null}), p_N, \mathcal{O}, p)$
10 DeleteNode$(Dummy, *, *)$
11 DeleteNode$(Intermediary, *, *)$
```

to obtain ε-LDP, then they are matched to $(*, o, O)$ with JoinSet (lines 1 to 4). Creating two dummies makes the truth three times likelier, and suits $\varepsilon \geq \ln(3)$. If we create 0 dummies, then we are 0-LDP as the edges' targets will be uniformly randomized. This step is illustrated in Figure 7.3 with the creation of 3 dummy nodes.

Edge Cut: We want to use RandomTarget to reroute edges p towards a new target matching $(*, o, O)$ at random. However, picking a dummy as a new target means we want to keep the original target. This means that instead of rerouting the edges p directly, we would like to create "forks" that lead both to the original and new targets. Since hyperedges are not objects of our graphs, we use EdgeCut and EdgeCopy to emulate this behaviour (line 5 and 6):

- First, we cut the edges p from $\mathcal{S}$ to $\mathcal{O}$ into p_I and p_O edges coming from a middle node labelled "Intermediary"
- Then, we create a copy of p_O edges labelled p_N

This is pictured on Fig. 7.4.

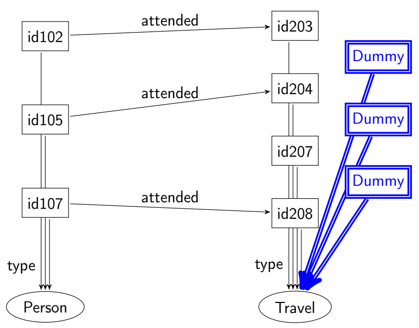


Figure 7.3: Create dummy travel nodes

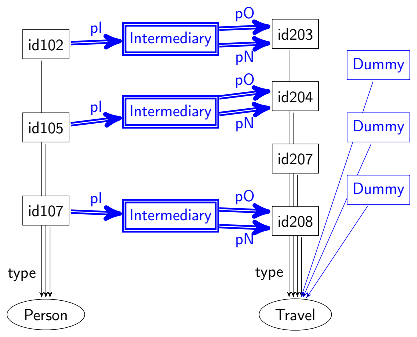


Figure 7.4: Split "attended" edges into pI, pO, pN

Randomizing: We then use RandomTarget (line 7) to redirect the edges p_N towards any target matching $(*, o, O) = \mathcal{O}$, with uniform probability. We have one chance to pick the original node, and $K-1$ chances to pick a dummy. This means we are K times more likely to pick one of these nodes than any other real node matching $\mathcal{O}$. This is pictured on Fig. 7.5, where node matching *(*,type,Travel)* are depicted in red.

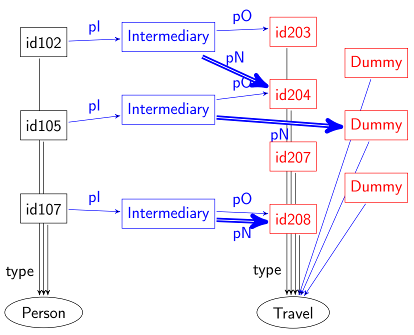


Figure 7.5: Randomize the target of pN edges.

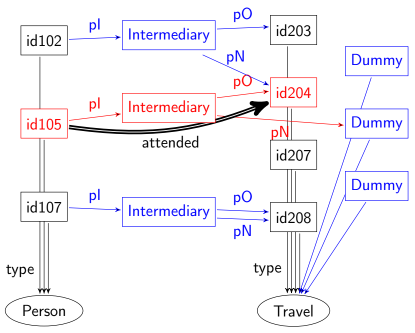


Figure 7.6: Chord rerouting: pN targets a Dummy.

Rerouting: We start by dealing with the case where p_N was rerouted to a dummy (line 8). In that case, the intermediary node matches $(*, p_N, Dummy)$, and we want to keep the old target of p, which means the current target of p_O. We say that if there is a chord from a node matching $\mathcal{S}$ to an intermediary node matching $(*, p_N, Dummy)$ to a destination node matching $\mathcal{O}$ with edges p_I, p_O, then we create the edge p from the source to the old destination. This is pictured on Fig. 7.6, the matched chord being depicted in red.

We then deal with the case where p_N was rerouted to a real target (line 9). We say that if there is a chord from a node matching $\mathcal{S}$ to any intermediary node to a destination node matching $(*, o, O)$ with edges p_I, p_N, then we create the edge p from the source to the new destination. This is pictured on Fig. 7.7, the matched chords being depicted in red.

We note that this also creates unnecessary edges from sources to dummies. This is not a problem, as dummies will be deleted shortly, as pictured on Fig. 7.8.

Termination: Since dummies and intermediary nodes are artefacts we created rather than real nodes of the graph, we end the procedure by using `DeleteNode` to delete all nodes of label "Dummy" and "Intermediary" (line 10 and 11). This also deletes all edges p_I, p_O, p_N linked to intermediary nodes, as well as edges p unduly targeting dummies generated during rerouting.

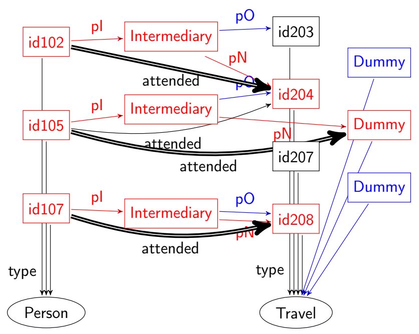


Figure 7.7: Chord rerouting: every path pI, pN generates a new "attended" edge.

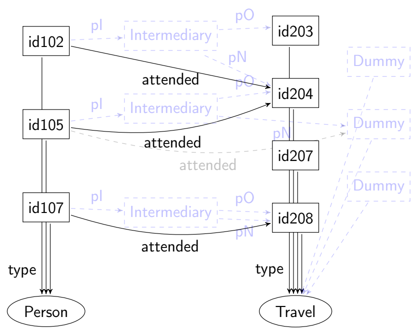


Figure 7.8: Cleanup.

7.4.2 - Anonymization through Anatomization

In this section we show that sorting sensitive attributes in groups to prevent inferences can be naturally expressed in our elementary operators.

We consider that some attributes of our graphs are sensitive, as described in [30, 149]. These papers describe an anonymization process that involves the deletion of explicit identifiers (*e.g.* first and last name), and the separation of nodes with QIDs (*e.g.* date of birth, zip code) to a set of sensitive attributes (*e.g.* religion, sexual orientation). To cut links between QIDs and the sensitive attributes, the possible values of sensitive attributes are put into groups through a process called semantic anatomization [30]. Their approach is qualified as semantic as it concentrates on semantic-aware grouping of sensitive attributes. Furthermore, it retains the correlation between entity QIDs and semantically related sensitive values.

While the creation of these groups itself is a potentially involved and complex process that goes beyond the scope of graph rewriting, the redirection of links from identifiers to sensitive attributes is possible if the groups are provided.

We consider that our graph contains explicit identifiers (edge labels $e_1, \ldots, e_n$), quasi-identifiers (edge labels $q_1, \ldots, q_m$), sensitive attributes (edge labels $p_1, \ldots, p_k$), and group nodes that aggregate attribute values. A value X is part of a group if nodes of label X have an edge of label "inGroup" pointed at this group's node. The anonymization is made as follows:

Recurring example: The algorithm is detailed thereafter and illustrated through a toy example where the destinations of travels are considered sensitive $(p_i) = (destinatiation)$, the identifiers are names $(e_i) = (name)$, and the QIDs are types $(q_i) = (types)$.

More specifically, we illustrate the steps of sensitive attributes redirection in a recurring example presented in Fig. 7.9 to 7.14.

Erasing explicit identifiers: Lines 1 to 3 delete the edges towards explicitly identifying attributes. The values are preserved, but the links are cut for anonymization. In the example, this would lead to the deletion of every edges whose attribute is "name".

Algorithm 5: Procedure Acordinnat($(e_i)_{1\leq i\leq n}, (q_i)_{1\leq i\leq m}, (p_i)_{1\leq i\leq k}, inGroup$)

```
foreach $e_i$ do
    DeleteEdge($(*, \text{null}, \text{null}), e_i, (*, \text{null}, \text{null})$)
end foreach
NewNode($QI$)
foreach $q_j$ do
    JoinSet($hasQI, QI$) Where$\{(*, q_j, *)\}$ Except$\{\}$
end foreach
foreach $p_i$ do
    EdgeChord($(*, hasQI, QI), p_i, (*, \text{null}, \text{null}), inGroup, (*, \text{null}, \text{null}), p'_i$)
end foreach
foreach $p_i$ do
    EdgeCut($(*, hasQI, QI), p_i, (*, \text{null}, \text{null}), p^i_I, "Intermediary", p^i_O$)
end foreach
foreach $p_i$ do
    EdgeChord($("Intermediary", \text{null}, \text{null}), p^i_O, (*, \text{null}, \text{null}), inGroup, (*, \text{null}, \text{null}), p''_i$)
end foreach
foreach $p_i$ do
    EdgeReverse($("Intermediary", \text{null}, \text{null}), p''_i, (*, \text{null}, \text{null}), p''_i$)
end foreach
foreach $p_i$ do
    EdgeChord($(*, \text{null}, \text{null}), p''_i, ("Intermediary", \text{null}, \text{null}), p^i_O, (*, \text{null}, \text{null}), hasOne$)
end foreach
foreach $p_i$ do
    EdgeCopy($(*, \text{null}, \text{null}), p'_i, (*, \text{null}, \text{null}), p_i$)
end foreach
foreach $p_i$ do
    EdgeCut($(*, hasQI, QI), p'_i, (*, \text{null}, \text{null}), p^i_I, "Intermediary", p^i_O$)
end foreach
DeleteNode($"Intermediary", \text{null}, \text{null}$)
DeleteNode($QI, \text{null}, \text{null}$)
```

Flagging QIDs: Lines 4 to 7 create a node QI and relate every node with at least one QIDs to QI through an edge $hasQI$. In the example, all nodes with a *type* are source of an edge $hasQI$ whose target is QI.

Redirecting sensitive attributes: Lines 8 to 10 match cases where a node with QIDs has an edge p_i pointing towards the value of a sensitive attribute, which itself has an edge $inGroup$ designating a group of the semantic anatomization. These matches are depicted in red on Fig. 7.9.

When such a match is detected, an edge p_i' is directly traced from the initial node to the group. The label used is p_i' instead of p_i to avoid undue deletions of redirected edges in line 11.

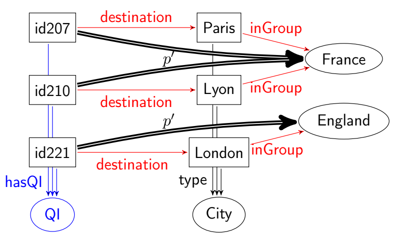


Figure 7.9: Short-circuit sensitive values

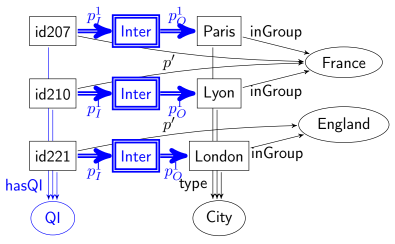


Figure 7.10: Split sensitive edges into p_{I_1}, p_{O_1}

Keep track of sensitive attribute values: Lines 11 through 22 aim to keep track of cardinalities of each sensitive value in the anatomization groups, represented by the edges p_i we delete.

Line 12 creates an intermediary node between the source and the value of its sensitive attribute, as depicted in Fig. 7.10.

Line 15 links those intermediary nodes with the group of their sensitive attributes with an edge labelled p_i'', illustrated in Fig. 7.11. The matched chords are depicted in red.

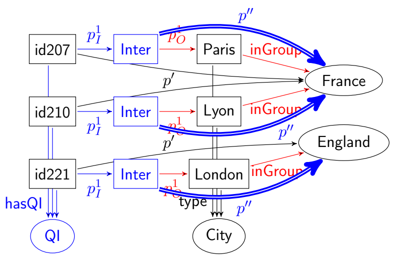


Figure 7.11: Link intermediaries with groups

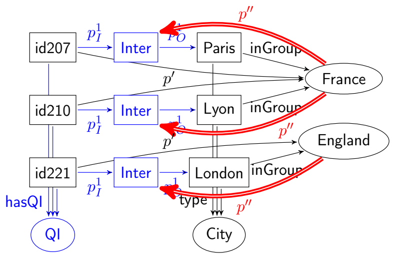


Figure 7.12: Reverse from group to intermediaries

This edge is inverted in line 18, then used in line 21 to create a direct link between the group and the sensitive attribute, as shown if Fig. 7.12 (where the inverted p'' are depicted in red) and Fig. 7.13 (where the p'', p_O^i chords are depicted in red), respectively.

Bringing back p_i: Line 24 creates a copy p_i of each p_i' linking a node with the group its sensitive value is in. Then, line 27 cuts the p'_i edges to prepare them for deletion.

Cleanup: Finally, lines 29 and 30 erase the nodes we created in previous steps. This notably includes the last trace of edges p_i that could not have been redirected –i.e., those that did not belong to an anatomization group–, for which the information is gone for good. This last step is depicted in Fig. 7.14.

Should we want to preserve those sensitive but ungrouped edges, we would only split edges pointing towards grouped values, by replacing lines 12 with:

$$\texttt{EdgeCut}((*, hasQI, QI), p_i, (*, inGroup, null), p_{I,i}, "Intermediary", p_{O,i})$$

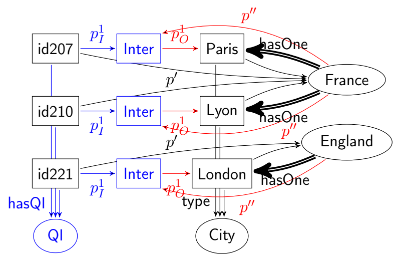


Figure 7.13: Chord from groups to values

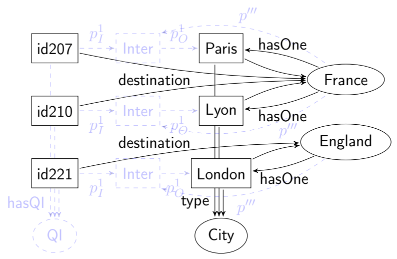


Figure 7.14: Cleanup.

7.5 - Experimental evaluation

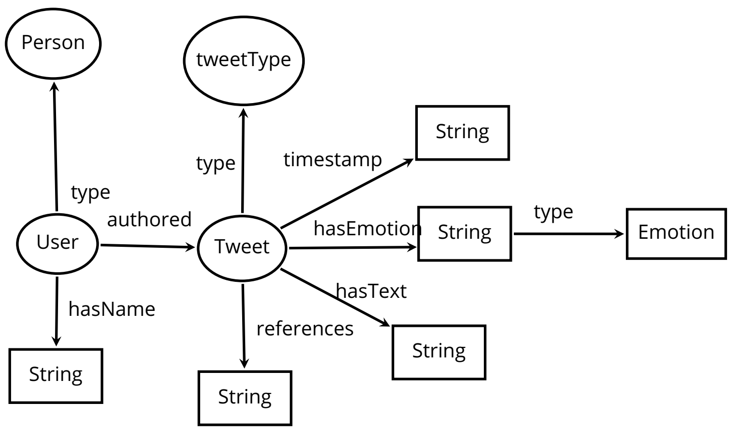


Figure 7.15: Schema for Sentiment140

This section investigates the feasibility of the approach and provides a preliminary experimental evaluation by applying two privacy procedures to a real dataset. We implemented the formal specification of the operators using AGG – The Attributed Graph Grammar System [171], one of the most mature development environment supporting the definition and application of typed graph rewriting systems [172]. Using AGG's Java API, we implemented Granon, a tool in Java[2] to handle their management and on-the-fly modification (*e.g.* the definitions of multiple NACs and PACs in the *JoinSet* operator). Granon also supports the procedures defined

[2]available at `https://github.com/ceichler/granon`

herein. It can be used as a library, through a textual user interface supported by a parser on the language described herein or through a graphical user interface. Experiments are conducted in a single-thread on an Intel Xeon Gold 5215 2.5GHz with 64 GB RAM and a 20Go JVM heap size.

Dataset(s): The experiments are conducted on sampling of the Sentiment140 dataset composed of 1.6 million tweets[3], which we have parsed to load as a knowledge graph conforming to the schema shown in Fig. 7.15. The dataset is composed of *tweets* authored by a named user. Tweets each have a timestamp, a full text, and an emotion (positive, neutral, or negative). They may reference users' name. Both the users, tweets and emotions are typed.

The most important factor regarding runtime is the number of tweets and the size of the graph. Therefore, we apply our experiments on the graphs resulting from the parsing of the t first tweets of the datasets, with $t =$ 200, 400, 600, 800, 1000, 1200, and 2000.

Nature of the experiments: To investigate the scalability of the proposal, we apply an instantiation of both our privacy scheme:

1) *Procedure LDP((*,"type", "tweetType"), "hasEmotion", (*, "type", "Emotion"), ln(2))* as described in Section 7.4.1. We arbitrarily consider the emotion of a tweet to be the sensitive value and use $\varepsilon = ln(2)$, the value of $epsilon$ having a negligible influence on the runtime.
2) *Procedure Anat((hasText), (references), (timestamp), inGroup)* described in 7.4.2. We consider texts to be identifiers, timestamps to be sensitive and references to be QIDs. Therefore, the procedure will delete all "hasText" edges and generalize the timestamp of any tweet that references someone. After considering the dataset, we construct anatomization groups representing a one minute timewindow: for instance a timestamp with value "Mon Apr 06 22:20:19 PDT 2009" belongs to the group "Mon Apr 06 22:20 PDT 2009". Groups are constructed while parsing the experimental datasets, resulting in slightly bigger graphs for the same t.

Experimental results: Average and median execution times (in ms) over 50 runs of the procedures are reported in Fig. 7.16 for the various t. The sizes of the input graphs (*i.e.*their number of vertices #V and edges #E) constructed by parsing the first t tweets of the dataset are reported in Tab. 7.1. The size of the graph is linear in *t*, which is consistent with the schema. The average and median runtimes for the LDP procedure are overlapping as the distribution has a very low standard deviation. The distribution of runtimes for Anatomization procedure consistently comports high outliers for every t.

Experimental interpretation: As expected, the LDP procedure is more time consuming than the Anatomization procedure for $t = 1000$ the median for LDP and Anatomization are 1,47 * 10^6 ms and 2,4 * 10^4ms, respectively). Furthermore, while the asymptotic complexity of the latter is roughly quadratic for $t \geq 2000$, the former is over-quadratic. The median execution time of the LDP procedure roughly triples from $t = 400$ to $t = 500$ (1,2 * 10^5 ms to 3,3 * 10^5ms) when the number of nodes is multiplied by 1,25. The average execution time of the Anatomization procedure is multiplied by 547 regarding $t =$ 100 and t =3000, with 28,7 times the number of nodes.

[3]https://www.kaggle.com/kazanova/sentiment140

Table 7.1: Size of the graphs resulting from parsing the t first tweets of the dataset

	LDP		Anatomization	
t	#V	#E	#V	#E
100	531	743	538	843
200	1042	1465	1054	1665
300	1558	2201	1576	2501
400	2081	2930	2105	3330
500	2605	3661	2636	4161
600	3113	4386	3150	4986
700	3623	5115	3665	5815
800	4147	5889	4195	6659
900	4648	6576	4704	7476
1000	5170	7310	5232	8310
2000			10393	16531
3000			15463	24625

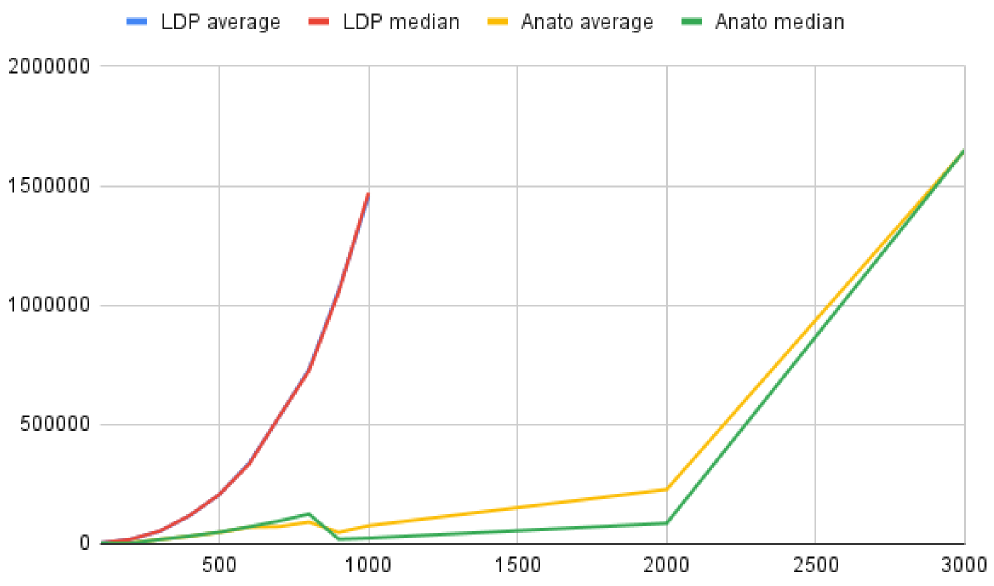


Figure 7.16: Experimental results runtime (ms) with various *t* (# tweet)

This can be explained by the randomization inherent to LDP. Indeed, randomizing the target of a relation requires to restart the matching process from scratch, without benefiting from optimizations (e.g., smart backtracking). Otherwise, the first match would influence all others. Therefore, randomizing the sensitive value of n items requires -in the randomizing step- running n search for graph homomorphisms, which is itself super-linear in the size of the graph.

A 30 minutes run would allow to execute the LDP (resp. anatomization) scheme on a graph slightly bigger than #V = 5170, #E = 7310 (resp. #V = 15463, #E = 24625). While such a runtime is very much reasonable when considering the sanitization of a database prior to its release as the process is expected to be ran infrequently, the asymptotic complexities discourage the application of the proposed graph-rewriting based techniques to big graphs.

7.6 - Conclusion and future work

The work introduced in this chapter is a first effort toward bridging the gap between rigorous algebraic graph rewriting and graph sanitization. We propose a language base on a basic yet expressive set of atomic operators, formalized as simple Single Push-Out graph rewriting rules, providing a generic expressive rigorous definition that can be parametrized with node and set identifiers. We show how they can be used to express privacy-preserving graph databases publication mechanisms, namely local differential privacy and anatomization. These operators and procedures have been subject to an open-source implementation[4]. This work stands as a proof of concept and a first step towards a graph rewriting based approach to graph database sanitization. A lot of considerations and work remain open on the topic.

Firstly, while the current implementation would reasonably allow the sanitization of small to medium graphs, its scalability remains limited due to high asymptotic complexity. A first effort would be required to reduce multiplicative constants, for example, by expanding operators to reduce the number of transformations, e.g., implementing an operator for edge suppression or edge modification rather than relying on two operators to conduct the operation; 2) or even the asymptotic complexity of our procedures, for example by considering a subgraph for the transformation, as procedures work on local properties. Another natural outlook would be to expand the tool to encompass other privacy protocols (e.g., [29]). This might require the use of other operators, that we hope share the simplicity and versatility of our current set.

[4]available at `https://github.com/ceichler/granon`

General Conclusion and Perspectives

This chapter will conclude the study by recalling the key points from each of our contributions. It will also address potential limitations of the work and propose directions for future research.

Privacy models for RDF graphs. The research conducted in this thesis aimed to study privacy over RDF graphs. More specifically, we focused on adapting DP to edge-labeled directed graphs with underlying semantics. The goal was to propose approaches and techniques to query RDF graphs and design, implement, and evaluate algorithms that ensure DP across such graphs while preserving data utility. To this end, we proposed a novel approach that relied on graph projection to adapt DP to RDF graphs. The primary purpose was to decrease the sensitivity of several query types. Analytical and experimental assessment of our approach was performed in the context of a Twitter use case, demonstrating an enhancement of up to several orders of magnitude compared to a naive approach that would not incorporate projection.

Mapping and neighborhoods. The second aim of this thesis was to study how neighborhood definitions over relational databases that adhere to FK constraints translate to RDF. The third aim was to propose new neighborhood definitions over relational database that can be linked to existing neighborhood definitions on graphs, representing well known privacy models, such as node-DP, or even more semantic definitions like QL-Outedge-DP.

Furthermore, we explored the semantics of these neighborhood definitions within a relational database context. Both aims entailed studying how to map a relational database to an RDF database. To this end, we proposed a notion of neighborhood over relational databases, namely `restrict deletion neighborhood`, and studied what it meant in the context of RDF graphs. We also proposed a notion of neighborhood for a relational database, equivalent to the QL-Outedge privacy previously defined over RDF graphs. Finally, we proposed an implementation based on R2RML to demonstrate the applicability of our approach.

Using graph rewriting to support anonymization algorithms. The last objective of this thesis was to support the design and implementation of graph sanitization algorithms, such as anatomization, using a rigorous graph rewriting approach. This last contribution was a graph transformation language that served as a basis for the construction of different sanitization mechanisms. This language relied on a set of elementary transformation operators formalized using a rigorous algebraic graph rewriting approach.

Creating usable software. An important aspect of this Ph.D. work was to create *usable* software. Each contribution presented in this manuscript was implemented and tested on real datasets. I implemented the prototypes for Chapters 5 and 6. I worked on the design of the prototype for Chapter 7. Its implementation was done by one of my supervisors and the textual and graphical user interface by an intern.

Perspectives:

Improving query accuracy in the presence of privacy constraints. The proposed study that adapts DP to RDF graphs in Chapter 5 underlines the importance of several research directions: optimizing projection to reduce information loss, investigating the impact of edge ordering on utility, and exhibiting optimal orders for families of queries. In this context, it would be interesting to develop new methods that allow the prioritization of certain edges above others in the projection process to improve the accuracy of query results. This optimization could be based on prior knowledge of one or several queries to be of interest. Afterward, experimental and/or theoretical studies can be performed to evaluate query accuracy and data privacy.

Another research direction is to systematically evaluate our approach w.r.t. a large set of queries, exhibiting the interest of each projection and privacy models.

Continuing to study the semantics of neighborhoods in the presence of background information. The guarantee provided by DP works best under the assumption that any graph has neighbors to "hide behind." If a graph is isolated from any of its neighbors, then the guarantee provided by DP weakens. We suppose that such situations can arise if the graph databases we consider are known to follow structural constraints (e.g., "every patient has a doctor") or semantic constraints (e.g., "Dr Wilson is an oncologist"). If all possible graphs must follow specific rules, then it is possible that some graphs have no neighbors that an attacker could confuse them with.

In this regard, another line of research would be to study how DP and data privacy evolve in semantic databases when they are known to follow such constraints, for instance, as expressed in an ontology (RDFS or OWL). It is interesting to identify, detect, and quantify the possible loss of privacy incurred, detail how malicious users could exploit this weakness through a semantic-based attack of a DP process, then establish how best to adapt DP (and methods to provide it) to propose countermeasures that ensure privacy is preserved without destroying the querying processes' usefulness in the process.

Moreover, some future work is to formalize and evaluate through experimentation the damage that prior knowledge of a target graph's schema can affect the privacy of a DP-guarantying process. One possible approach is to identify an example of an attack through schema knowledge of a DP guarantying process, evaluate the risk of such attacks on current privacy standards and study mechanisms, such as metric-based d-differential privacy [176], as a means to counter such attacks.

Improving implementation. Another future work is to design and implement mapping methods between graphs and relational databases, as well as neighborhood definitions which would make more sense in this context. Furthermore, it would be interesting to compare the performance (in times of temporal complexity) of our proposed methods and algorithms that guarantee DP in graph databases to those of algorithms directly running on graphs. This involves implementation of relational-to-graph and graph-to-relational database mapping methods. Also, establishing a benchmark to compare the efficiency of different privacy methods through translation would be a useful tool.

Regarding the work on Chapter 7, our language and operators are designed to work on simple relations between two sets of nodes. In real world use cases, most queries are joins of several relations (or, to speak in graph terms, path queries on more than one edge). For a sanitization mechanism to preserve good qualities on such requests while still providing privacy guarantees, it is likely that we will need operators adapted to the preservation of invariants on composite paths. It is possible, but not yet shown, that some such operators can be built as compositions of one-relation operators.

Ultimately, as a more general goal, the intended future work is to expand our current tool to allow a user to specify high-level semantic and privacy constraints. Such a tool would compile these requests as sequences of our operators and offer to perform the resulting graph transformations automatically.

Bibliography

[1] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International journal of uncertainty, fuzziness and knowledge-based systems*, 10(05):557–570, 2002.

[2] Honglu Jiang, Jian Pei, Dongxiao Yu, Jiguo Yu, Bei Gong, and Xiuzhen Cheng. Differential privacy and its applications in social network analysis: A survey. *arXiv preprint arXiv:2010.02973*, 2020.

[3] Timothy J Berners-Lee. Information management: A proposal. Technical report, 1989.

[4] Grigoris Antoniou and Frank van Harmelen. A semantic web primer (second.). *The MIT Press Cambridge, Massachusetts, London, England*, 2008.

[5] Rokhshad Tavakoli and Sarah NR Wijesinghe. The evolution of the web and netnography in tourism: A systematic review. *Tourism Management Perspectives*, 29:48–55, 2019.

[6] Sareh Aghaei, Mohammad Ali Nematbakhsh, and Hadi Khosravi Farsani. Evolution of the world wide web: From web 1.0 to web 4.0. *International Journal of Web & Semantic Technology*, 3(1):1–10, 2012.

[7] World Wide Web Consortium et al. W3c semantic web activity. http://www. w3. org/2001/sw, 2008. Accessed 2023-09-14.

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[9] Richard Cyganiak, David Wood, Markus Lanthaler, Graham Klyne, Jeremy J Carroll, and Brian McBride. Rdf 1.1 concepts and abstract syntax. *W3C recommendation*, 25(02):1–22, 2014.

[10] Tim Berners-Lee. Linked data-design issues. *http://www. w3. org/DesignIssues/LinkedData. html*, 2006.

[11] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011.

[12] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. 1998.

[13] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[14] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.

[15] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[16] Ahmet Aktay, Shailesh Bavadekar, Gwen Cossoul, John Davis, Damien Desfontaines, Alex Fabrikant, Evgeniy Gabrilovich, Krishna Gadepalli, Bryant Gipson, Miguel Guevara, et al. Google covid-19 community mobility reports: anonymization process description (version 1.1). *arXiv preprint arXiv:2004.04145*, 2020.

[17] D Apple. Learning with privacy at scale. *Apple Machine Learning Journal*, 1(8), 2017.

[18] Apple Press Info. Apple previews ios 10, the biggest ios release ever, 2016. *URL https://www. apple. com/pr/library/2016/06/13Apple-Previews-iOS-10-The-Biggest-iOS-Release-Ever. html.*

[19] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. *Advances in Neural Information Processing Systems*, 30, 2017.

[20] Simson L Garfinkel, John M Abowd, and Sarah Powazek. Issues encountered deploying differential privacy. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, pages 133–137, 2018.

[21] Simson Garfinkel. Implementing differential privacy for the 2020 census. *USENIX Association*, 2021.

[22] John M Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2867–2867, 2018.

[23] Krishnaram Kenthapadi and Thanh TL Tran. Pripearl: A framework for privacy-preserving analytics and reporting at linkedin. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2183–2191, 2018.

[24] Ryan Rogers, Subbu Subramaniam, Sean Peng, David Durfee, Seunghyun Lee, Santosh Kumar Kancha, Shraddha Sahay, and Parvez Ahammad. Linkedin's audience engagements api: A privacy preserving data analytics system at scale. *arXiv preprint arXiv:2002.05839*, 2020.

[25] Chaya Nayak. New privacy-protected facebook data for independent research on social media's impact on democracy. *Facebook Research*, 2020.

[26] Joseph P Near, Xi He, et al. Differential privacy for databases. *Foundations and Trends® in Databases*, 11(2):109–225, 2021.

[27] 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.

[28] Joe Near. Differential privacy at scale: Uber and berkeley collaboration. In *Enigma 2018 (Enigma 2018)*, 2018.

[29] Remy Delanaux, Angela Bonifati, Marie-Christine Rousset, and Romuald Thion. Query-based linked data anonymization. In *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17*, pages 530–546. Springer, 2018.

[30] Maxime Thouvenot, Olivier Curé, and Philippe Calvez. Knowledge graph anonymization using semantic anatomization. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 4065–4074. IEEE, 2020.

[31] Hira Asghar, Christophe Bobineau, and Marie-Christine Rousset. Identifying privacy risks raised by utility queries. In *International Conference on Web Information Systems Engineering*, pages 309–324. Springer, 2022.

[32] Sara Taki, Cedric Eichler, and Benjamin Nguyen. Using projection to improve differential privacy on rdf graphs. In *Actes de la conférence BDA 2021*, page 72, 2021.

[33] Sara Taki, Cédric Eichler, and Benjamin Nguyen. It's too noisy in here: Using projection to improve differential privacy on rdf graphs. In *European Conference on Advances in Databases and Information Systems*, pages 212–221. Springer, 2022.

[34] Adrien Boiret, Cédric Eichler, and Benjamin Nguyen. Privacy operators for semantic graph databases as graph rewriting. In Silvia Chiusano, Tania Cerquitelli, Robert Wrembel, Kjetil Nørvåg, Barbara Catania, Genoveva Vargas-Solar, and Ester Zumpano, editors, *New Trends in Database and Information Systems - ADBIS 2022 Short Papers, Doctoral Consortium and Workshops: DOING, K-GALS, MADEISD, MegaData, SWODCH, Turin, Italy, September 5-8, 2022, Proceedings*, volume 1652 of *Communications in Computer and Information Science*, pages 366–377. Springer, 2022.

[35] Sara Taki, Cedric Eichler, and Benjamin Nguyen. Privacy over rdf datasets. In *Actes de la conférence BDA 2021*, page 94, 2021.

[36] Sara Taki, Cedric Eichler, and Benjamin Nguyen. Using projection to improve differential privacy on rdf graph. In *Atelier sur la Protection de la Vie Privée (APVP 2022), Châtenay-sur-Seine, France, 2022*, 2022.

[37] Sebastian Weber and Jörg Rech. An overview and differentiation of the evolutionary steps of the web xy movement: the web before and beyond 2.0. *Handbook of Research on Web 2.0, 3.0, and X. 0: Technologies, Business, and Social Applications*, pages 12–39, 2010.

[38] Karan Patel. Incremental journey for world wide web: introduced with web 1.0 to recent web 5.0–a survey paper. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(10), 2013.

[39] Aidan Hogan. *Web of data*. Springer, 2020.

[40] Martin Dürst and Michel Suignard. Internationalized resource identifiers (iris). Technical report, 2005.

[41] Javier D Fernández and Miguel A Martínez-Prieto. Rdf serialization and archival., 2019.

[42] Fabien Gandon and Guus Schreiber. Rdf 1.1 xml syntax. 2014. *URL: www. w3. org/TR/rdfsyntax-grammar/(Last accessed: November 11, 2014)*, 2016.

[43] David Beckett. Rdf 1.1 n-triples. *URL: https://www. w3. org/TR/n-triples*, 2014.

[44] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. Rdf 1.1 turtle–terse rdf triple language. w3c recommendation. *World Wide Web Consortium (Feb 2014), available at http://www. w3. org/TR/turtle*, 2014.

[45] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, and Niklas Lindström. Json-ld 1.0, a json-based serialization for linked data, w3c recommendation 16 january 2014. *World Wide Web Consortium. Available at https://www. w3. org/TR/json-ld*, 2014.

[46] Tim Berners-Lee and Dan Connolly. Notation3 (n3): A readable rdf syntax. *W3C team submission*, 28, 2011.

[47] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. Rql: a declarative query language for rdf. In *Proceedings of the 11th international conference on World Wide Web*, pages 592–603, 2002.

[48] Jeen Broekstra and Arjohn Kampman. Serql: A second generation rdf query language. In *Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 13–14. Citeseer, 2003.

[49] Steve Harris, Andy Seaborne, and E Prud'hommeaux. Sparql 1.1 query language. w3c recommendation (2013). *URL https://www. w3. org/TR/sparql11-query*, 2013.

[50] P Gearon, A Passant, and A Polleres. Sparql 1.1 update. w3c recommendation. *World Wide Web Consortium (Mar 2013), available at http://www. w3. org/TR/sparql11-update*, 2013.

[51] Thomas R Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[52] W Borst. Construction of engineering ontology. *Ph. D. Dissertation, Institute for Telematica and Information Technology, University of Twente, Enschede, the Netherlands*, 1997.

[53] Dan Brickley, Ramanathan V Guha, and Brian McBride. Rdf schema 1.1. *W3C recommendation*, 25:2004–2014, 2014.

[54] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[55] World Wide Web Consortium et al. Owl 2 web ontology language document overview. 2012.

[56] Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation, 2004.

[57] Satya S Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, and Ahmed Ezzat. A survey of current approaches for mapping of relational databases to rdf. *W3C RDB2RDF Incubator Group Report*, 1:113–130, 2009.

[58] 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.

[59] Das Souripriya, Sundara Seema, and Cyganiak Richard. R2rml: Rdb to rdf mapping language. *W3C Recommendation*, 27, 2012.

[60] Tim Berners-Lee. Relational databases on the semantic web, 1998. *Via https://www. w3. org/DesignIssues/RDBRDF. html, last accessed January*, 2015.

[61] 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.

[62] Franck Michel, Johan Montagnat, and Catherine Faron Zucker. A survey of rdb to rdf translation approaches and tools. 2013.

[63] Sören Auer, L Feigenbaum, D Miranker, A Fogarolli, and J Sequeda. Use cases and requirements for mapping relational databases to rdf. *W3c working draft, W3C*, 2010.

[64] Dinusha Vatsalan, Ziad Sehili, Peter Christen, and Erhard Rahm. Privacy-preserving record linkage for big data: Current approaches and research challenges. *Handbook of big data technologies*, pages 851–895, 2017.

[65] Abid Mehmood, Iynkaran Natgunanathan, Yong Xiang, Guang Hua, and Song Guo. Protection of big data privacy. *IEEE access*, 4:1821–1834, 2016.

[66] Michael Barbaro, Tom Zeller, and Saul Hansell. A face is exposed for aol searcher no. 4417749. *New York Times*, 9(2008):8, 2006.

[67] Judith Duportail. I asked tinder for my data. it sent me 800 pages of my deepest, darkest secrets. *The guardian*, 26, 2017.

[68] Joe Tidy and David Molloy. Twitch confirms massive data breach, 2021. *URL https://www. bbc. com/news/technology-58817658*.

[69] PROCESSING OF PERSONAL DATA. Directive 95/46/ec of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. *Official Journal L*, 281(23/11):0031–0050, 1995.

[70] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[71] Warren B Chik. The singapore personal data protection act and an assessment of future trends in data privacy reform. *Computer Law & Security Review*, 29(5):554–575, 2013.

[72] Benjamin CM Fung, Ke Wang, Rui Chen, and Philip S Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys (Csur)*, 42(4):1–53, 2010.

[73] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[74] Latanya Sweeney. Simple demographics often identify people uniquely. *Health (San Francisco)*, 671(2000):1–34, 2000.

[75] Tore Dalenius. Finding a needle in a haystack or identifying anonymous census records. *Journal of official statistics*, 2(3):329, 1986.

[76] Lawrence H Cox. Suppression methodology and statistical disclosure control. *Journal of the American Statistical Association*, 75(370):377–385, 1980.

[77] Khaled El Emam and Fida Kamal Dankar. Protecting privacy using k-anonymity. *Journal of the American Medical Informatics Association*, 15(5):627–637, 2008.

[78] Pierangela Samarati. Protecting respondents identities in microdata release. *IEEE transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[79] Adam Meyerson and Ryan Williams. On the complexity of optimal k-anonymity. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 223–228, 2004.

[80] Gagan Aggarwal, Tomás Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Anonymizing tables. In *Database Theory-ICDT 2005: 10th International Conference, Edinburgh, UK, January 5-7, 2005. Proceedings 10*, pages 246–258. Springer, 2005.

[81] Raymond Chi-Wing Wong, Jiuyong Li, Ada Wai-Chee Fu, and Ke Wang. (α, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 754–759, 2006.

[82] Roberto J Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *21st International conference on data engineering (ICDE'05)*, pages 217–228. IEEE, 2005.

[83] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *22nd International conference on data engineering (ICDE'06)*, pages 25–25. IEEE, 2006.

[84] Xiaokui Xiao and Yufei Tao. Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32nd international conference on Very large data bases*, pages 139–150, 2006.

[85] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd international conference on data engineering*, pages 106–115. IEEE, 2006.

[86] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99, 2000.

[87] Josep Domingo-Ferrer and Vicenç Torra. A critique of k-anonymity and some of its enhancements. In *2008 Third International Conference on Availability, Reliability and Security*, pages 990–993. IEEE, 2008.

[88] Shuchi Chawla, Cynthia Dwork, Frank McSherry, Adam Smith, and Hoeteck Wee. Toward privacy in public databases. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*, pages 363–385. Springer, 2005.

[89] 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.

[90] Tore Dalenius. Towards a methodology for statistical disclosure control. 1977.

[91] Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69, 1965.

[92] Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. Differentially private spatial decompositions. In *2012 IEEE 28th International Conference on Data Engineering*, pages 20–31. IEEE, 2012.

[93] Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Marco Stronati. Location privacy via geo-indistinguishability. *ACM Siglog News*, 2(3):46–69, 2015.

[94] Xi He, Graham Cormode, Ashwin Machanavajjhala, Cecilia Procopiuc, and Divesh Srivastava. Dpt: differentially private trajectory synthesis using hierarchical reference systems. *Proceedings of the VLDB Endowment*, 8(11):1154–1165, 2015.

[95] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the netflix prize contenders. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–636, 2009.

[96] Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang. Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, 48(1):1–44, 2015.

[97] Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, pages 457–476. Springer, 2013.

[98] Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 123–138, 2016.

[99] Kamalkumar R Macwan and Sankita J Patel. Node differential privacy in social graph degree publishing. *Procedia computer science*, 143:786–793, 2018.

[100] Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 87–96, 2013.

[101] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. Boosting the accuracy of differentially-private histograms through consistency. *arXiv preprint arXiv:0904.0942*, 2009.

[102] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84, 2007.

[103] Shixi Chen and Shuigeng Zhou. Recursive mechanism: towards node differential privacy and unrestricted joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 653–664, 2013.

[104] Peeter Laud, Alisa Pankova, and Martin Pettai. Achieving differential privacy using methods from calculus. *arXiv preprint arXiv:1811.06343*, 2018.

[105] David McCandless, Tom Evans, Miriam Quick, Ella Hollowood, Christian Miles, Dan Hampson, and Duncan Geere. World's biggest data breaches & hacks. *Information is Beautiful*, 2021.

[106] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[107] John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438. IEEE, 2013.

[108] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. Practical locally private heavy hitters. *Advances in Neural Information Processing Systems*, 30, 2017.

[109] Ning Wang, Xiaokui Xiao, Yin Yang, Jun Zhao, Siu Cheung Hui, Hyejin Shin, Junbum Shin, and Ge Yu. Collecting and analyzing multidimensional data with local differential privacy. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 638–649. IEEE, 2019.

[110] Natasha Fernandes, Kacem Lefki, and Catuscia Palamidessi. Utility-preserving privacy mechanisms for counting queries. *Models, Languages, and Tools for Concurrent and Distributed Programming: Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, pages 487–495, 2019.

[111] Graham Cormode, Samuel Maddock, and Carsten Maple. Frequency estimation under local differential privacy [experiments, analysis and benchmarks]. *arXiv preprint arXiv:2103.16640*, 2021.

[112] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Extremal mechanisms for local differential privacy. *Journal of Machine Learning Research*, 17(17), 2016.

[113] Vibhor Rastogi, Dan Suciu, and Sungho Hong. The boundary between privacy and utility in data publishing. In *Proceedings of the 33rd international conference on Very large data bases*, pages 531–542, 2007.

[114] Avrim Blum, Katrina Ligett, and Aaron Roth. A learning theory approach to noninteractive database privacy. *Journal of the ACM (JACM)*, 60(2):1–25, 2013.

[115] Bin Zhou and Jian Pei. Preserving privacy in social networks against neighborhood attacks. In *2008 IEEE 24th International Conference on Data Engineering*, pages 506–515. IEEE, 2008.

[116] Bin Zhou, Jian Pei, and WoShun Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *ACM Sigkdd Explorations Newsletter*, 10(2):12–22, 2008.

[117] Imrul Kayes and Adriana Iamnitchi. Privacy and security in online social networks: A survey. *Online Social Networks and Media*, 3:1–21, 2017.

[118] Jemal H Abawajy, Mohd Izuan Hafez Ninggal, and Tutut Herawan. Privacy preserving social network data publication. *IEEE communications surveys & tutorials*, 18(3):1974–1997, 2016.

[119] Shouling Ji, Prateek Mittal, and Raheem Beyah. Graph data anonymization, de-anonymization attacks, and de-anonymizability quantification: A survey. *IEEE Communications Surveys & Tutorials*, 19(2):1305–1326, 2016.

[120] Abdul Majeed and Sungchang Lee. Anonymization techniques for privacy preserving data publishing: A comprehensive survey. *IEEE access*, 9:8512–8545, 2020.

[121] Baida Ouafae, Ramdi Mariam, Louzar Oumaima, and Lyhyaoui Abdelouahid. Data anonymization in social networks. *EasyChair Preprint*, (2310), 2020.

[122] Yang Li, Michael Purcell, Thierry Rakotoarivelo, David Smith, Thilina Ranbaduge, and Kee Siong Ng. Private graph data release: A survey. *ACM Computing Surveys*, 55(11):1–39, 2023.

[123] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 93–106, 2008.

[124] Bin Zhou and Jian Pei. The k-anonymity and l-diversity approaches for privacy preservation in social networks against neighborhood attacks. *Knowledge and information systems*, 28(1):47–77, 2011.

[125] Lei Zou, Lei Chen, and M Tamer Özsu. K-automorphism: A general framework for privacy preserving network publication. *Proceedings of the VLDB Endowment*, 2(1):946–957, 2009.

[126] James Cheng, Ada Wai-chee Fu, and Jia Liu. K-isomorphism: privacy preserving network publication against structural attacks. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 459–470, 2010.

[127] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *2009 30th IEEE symposium on security and privacy*, pages 173–187. IEEE, 2009.

[128] Shirin Nilizadeh, Apu Kapadia, and Yong-Yeol Ahn. Community-enhanced de-anonymization of online social networks. In *Proceedings of the 2014 acm sigsac conference on computer and communications security*, pages 537–548, 2014.

[129] Mudhakar Srivatsa and Mike Hicks. Deanonymizing mobility traces: Using social network as a side-channel. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 628–637, 2012.

[130] Shouling Ji, Weiqing Li, Mudhakar Srivatsa, and Raheem Beyah. Structural data de-anonymization: Quantification, practice, and implications. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1040–1053, 2014.

[131] Lyudmila Yartseva and Matthias Grossglauser. On the performance of percolation graph matching. In *Proceedings of the first ACM conference on Online social networks*, pages 119–130, 2013.

[132] Shouling Ji, Weiqing Li, Mudhakar Srivatsa, Jing Selena He, and Raheem Beyah. Structure based data de-anonymization of social networks and mobility traces. In *Information Security: 17th International Conference, ISC 2014, Hong Kong, China, October 12-14, 2014. Proceedings 17*, pages 237–254. Springer, 2014.

[133] Nitish Korula and Silvio Lattanzi. An efficient reconciliation algorithm for social networks. *arXiv preprint arXiv:1307.1690*, 2013.

[134] Pedram Pedarsani, Daniel R Figueiredo, and Matthias Grossglauser. A bayesian method for matching two similar graphs without seeds. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1598–1607. IEEE, 2013.

[135] Arvind Narayanan, Elaine Shi, and Benjamin IP Rubinstein. Link prediction by de-anonymization: How we won the kaggle social network challenge. In *The 2011 International Joint Conference on Neural Networks*, pages 1825–1834. IEEE, 2011.

[136] 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.

[137] Christine Task and Chris Clifton. A guide to differential privacy theory in social network analysis. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 411–417. IEEE, 2012.

[138] Gueorgi Kossinets and Duncan J Watts. Empirical analysis of an evolving social network. *science*, 311(5757):88–90, 2006.

[139] Shuang Song, Susan Little, Sanjay Mehta, Staal Vinterbo, and Kamalika Chaudhuri. Differentially private continual release of graph statistics. *arXiv preprint arXiv:1809.02575*, 2018.

[140] Sofya Raskhodnikova and Adam Smith. Efficient lipschitz extensions for high-dimensional graph statistics and node private degree distributions. *arXiv preprint arXiv:1504.07912*, 2015.

[141] Jonathan Ullman and Adam Sealfon. Efficiently estimating erdos-renyi graphs with node differential privacy. *Advances in Neural Information Processing Systems*, 32, 2019.

[142] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, 2011.

[143] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. Private release of graph statistics using ladder functions. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 731–745, 2015.

[144] Anupam Gupta, Aaron Roth, and Jonathan Ullman. Iterative constructions and private data release. In *Theory of Cryptography: 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings 9*, pages 339–356. Springer, 2012.

[145] Zhan Qin, Ting Yu, Yin Yang, Issa Khalil, Xiaokui Xiao, and Kui Ren. Generating synthetic decentralized social graphs with local differential privacy. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 425–438, 2017.

[146] Christian Borgs, Jennifer Chayes, and Adam Smith. Private graphon estimation for sparse graphs. *Advances in Neural Information Processing Systems*, 28, 2015.

[147] Christine Task and Chris Clifton. What should we protect? defining differential privacy for social network analysis. In *State of the Art Applications of Social Network Analysis*, pages 139–161. Springer, 2014.

[148] Sabrina Kirrane, Serena Villata, and Mathieu d'Aquin. Privacy, security and policies: A review of problems and solutions with semantic web technologies. *Semantic Web*, 9(2):153–161, 2018.

[149] Filip Radulovic, Raúl García Castro, and Asunción Gómez-Pérez. Towards the anonymisation of rdf data. 2015.

[150] Benjamin Heitmann, Felix Hermsen, and Stefan Decker. k-rdf-neighbourhood anonymity: Combining structural and attribute-based anonymisation for linked data. *PrivOn@ ISWC*, 1951, 2017.

[151] Remy Delanaux, Angela Bonifati, Marie-Christine Rousset, and Romuald Thion. Rdf graph anonymization robust to data linkage. In *Web Information Systems Engineering–WISE 2019: 20th International Conference, Hong Kong, China, January 19–22, 2020, Proceedings 20*, pages 491–506. Springer, 2019.

[152] Maxime Thouvenot, Olivier Curé, and Philippe Calvez. Preventing attribute and entity disclosures: Combining k-anonymity and anatomy over rdf graphs. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 5460–5469. IEEE, 2021.

[153] A differentially private approach for querying rdf data of social networks. In *Proceedings of the 21st International Database Engineering & Applications Symposium*, pages 74–81, 2017.

[154] Carlos Buil-Aranda, Jorge Lobo, and Federico Olmedo. Differential privacy and sparql.

[155] Jenni Reuben. Towards a differential privacy theory for edge-labeled directed graphs. *SICHERHEIT 2018*, 2018.

[156] Bernardo Cuenca Grau and Egor V Kostylev. Logical foundations of linked data anonymisation. *Journal of Artificial Intelligence Research*, 64:253–314, 2019.

[157] Maxime Thouvenot, Philippe Calvez, and Olivier Curé. Kgastor: a privacy by design knowledge graph anonymized store. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 670–679. IEEE, 2022.

[158] Xiaokui Xiao and Yufei Tao. M-invariance: towards privacy preserving re-publication of dynamic datasets. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 689–700, 2007.

[159] 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.

[160] 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.

[161] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private sql with bounded user contribution. *arXiv preprint arXiv:1909.01917*, 2019.

[162] 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.

[163] 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 2022 International Conference on Management of Data*, pages 759–772, 2022.

[164] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE intelligent systems*, 21(3):96–101, 2006.

[165] 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.

[166] Shufeng Zhou. Exposing relational database as rdf. In *2010 2nd International Conference on Industrial and Information Systems*, volume 2, pages 237–240. IEEE, 2010.

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

[168] Ratan Bahadur Thapa and Martin Giese. Mapping relational database constraints to shacl (extended version). *Research report http://urn. nb. no/URN: NBN: no-35645*, 2022.

[169] Iovka Boneva, Slawomir Staworko, and Jose Lozano. Sherml: mapping relational data to rdf. 2019.

[170] Christophe Debruyne and Declan O'Sullivan. R2rml-f: Towards sharing and executing domain logic in r2rml mappings. *LDOW@ WWW*, 1593, 2016.

[171] Gabriele Taentzer. Agg: A graph transformation environment for modeling and validation of software. In *AGTIVE*, 2003.

[172] Sergio Segura, David Benavides, Antonio Ruiz-Cortés, and Pablo Trinidad. *Automated Merging of Feature Models Using Graph Transformations*, pages 489–505. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[173] Jacques Chabin, Cédric Eichler, Mírian Halfeld Ferrari, and Nicolas Hiot. Graph rewriting rules for RDF database evolution: optimizing side-effect processing. *Int. J. Web Inf. Syst.*, 17(6):622–644, 2021.

[174] Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, 109(1–2):181 – 224, 1993.

[175] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundam. Inf.*, 26(3,4):287–313, December 1996.

[176] Konstantinos Chatzikokolakis, Miguel E Andrés, Nicolás Emilio Bordenabe, and Catuscia Palamidessi. Broadening the scope of differential privacy using metrics. In *Privacy Enhancing Technologies: 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings 13*, pages 82–102. Springer, 2013.