

# Differential Privacy and SPARQL

1

2  
3  
4  
5  
6  
7  
8 Carlos Buil-Aranda<sup>a</sup>, Jorge Lobo<sup>b</sup> and Federico Olmedo<sup>c</sup>

9 <sup>a</sup> *Departamento de Informática, Universidad Técnica Federico Santa María and IMFD Chile Avda España 1680,*  
10 *Valparaíso Chile*

11 *E-mail: cbuil@inf.utfsm.cl*

12 <sup>b</sup> *ICREA and Universitat Pompeu Fabra, Roc Boronat 148, Barcelona, Spain*

13 *E-mail: jorge.lobo@upf.edu*

14 <sup>c</sup> *Departamento de Ciencias de la Computación, Universidad de Chile and IMFD, Beauchef 851, Santiago, Chile*

15 *E-mail: folmedo@dcc.uchile.cl*  
16  
17  
18  
19

20 **Abstract.** Differential Privacy is a framework that provides formal tools to develop algorithms to access databases and answer  
21 statistical queries with quantifiable accuracy and privacy guarantees. The notions of Differential Privacy are defined indepen-  
22 dently of the data model and the query language at stake. Most Differential Privacy results have been obtained on aggregation  
23 queries such as counting or finding maximum or average values, and on grouping queries over aggregations such as the creation  
24 of histograms. So far, the data model used by the framework research has typically been the relational model and the query  
25 language SQL. However, effective realizations of Differential Privacy for SQL queries that required joins had been limited. This  
26 has imposed severe restrictions on applying Differential Privacy in RDF knowledge graphs and SPARQL query language. By the  
27 simple nature of RDF data, most useful queries accessing RDF graphs will require intensive use of joins. Recently, new Differ-  
28 ential Privacy techniques have been developed that can be applied to many types of joins in SQL with reasonable results. This  
29 opened the question of whether these new results carry over to RDF and SPARQL. In this paper we provide a positive answer to  
30 this question by presenting an algorithm that can answer counting queries over a large class of SPARQL queries that guarantees  
31 Differential Privacy, if the RDF graph is accompanied with semantic information about its structure. We have implemented our  
32 algorithm and conducted several experiments, showing the feasibility of our approach for large graph databases. Our aim has  
33 been to present an approach that can be used as a stepping stone towards extensions and other realizations of Differential Privacy  
34 for SPARQL and RDF.

35 **Keywords:** Differential Privacy, SPARQL  
36  
37

## 38 1. Introduction

39  
40 As many social norms, privacy, or the right to pri-  
41 vacy, is an evolving term that is invoked in many con-  
42 texts as eloquently described by Louis Menand in [1]:  
43  
44

---

45 <sup>1</sup>Carlos Buil was supported by Fondecyt Iniciacion 11170714  
46 and by ANID - Millennium Science Initiative Program - Code  
47 ICN17\_002. Jorge Lobo was partially supported by the Span-  
48 ish Ministry of Economy and Competitiveness under Grant Num-  
49 bers: TIN-2016-81032-P, MDM-2015-052, and the U.S. Army Re-  
50 search Office under Agreement Number W911NF1910432. Fed-  
51 erico Olmedo was also supported by ANID - Millennium Science  
Initiative Program - Code ICN17\_002

38 “Privacy is associated with liberty, but it is also asso-  
39 ciated with privilege (private roads and private sales),  
40 with confidentiality (private conversations), with non-  
41 conformity and dissent, with shame and embarrass-  
42 ment, with the deviant and the taboo (...), and with sub-  
43 terfuge and concealment”.  
44

45 In order to get some formal underpinning of privacy  
46 in the context of electronic data collection and publish-  
47 ing, Li et al [2] have looked at privacy breaches, stud-  
48 ied their general characteristics, and concluded, that  
49 electronic privacy breaches always ended with giving  
50 an attacker the ability to identify (using public data)  
51 whether an individual is member of a set or class that

1 had been intended to be anonymous (e.g., the class of  
 2 individuals with high cholesterol). Hence, they define  
 3 preservation of privacy as *avoiding privacy breaches*  
 4 in the sense of not disclosing set memberships of indi-  
 5 viduals.

6 For the public good, such as the advance of public  
 7 health, or the fair distribution of government resources,  
 8 such data is frequently made public. There are also sit-  
 9 uations in which governmental and commercial orga-  
 10 nizations collect and analyze data to improve or pro-  
 11 vide new services. Especially in such cases, society ex-  
 12 pects a certain level of privacy on the way these orga-  
 13 nizations use the data. Publishing data with perfect pri-  
 14 vacy means that no assumption can be made about the  
 15 prior knowledge an attacker may have about the sup-  
 16 posedly anonymous set. Under this assumption, there  
 17 would be little utility in published data if perfect pri-  
 18 vacy is expected [2, 3]. Therefore, the research com-  
 19 munity has looked at weaker definitions of "accept-  
 20 able" privacy. Useful concepts like  $k$ -anonymity [4],  
 21  $l$ -diversity [5] and  $t$ -closeness [6] were developed but  
 22 they were shown to have weak privacy guarantees [7].

23 In spite of its limitations [8], a privacy notion that  
 24 has gained a lot of acceptance because of its formal  
 25 properties is *differential privacy*. We will present pre-  
 26 cise definitions later in the paper, but informally, dif-  
 27 ferential privacy tries to hide the identity of individu-  
 28 als that are members of a particular class, while still  
 29 providing some guarantees about the utility of the pub-  
 30 lished data about the class. The basic principle is sim-  
 31 ple. Given a universe  $\mathbb{D}$  of all possible data sets and a  
 32 query  $f : \mathbb{D} \rightarrow \mathcal{R}$  that can be applied to a dataset  $D$   
 33 in  $\mathbb{D}$  and results in a value of an abstract domain  $\mathcal{R}$ ,  
 34 the result  $f(D)$  is differentially private if it is indistin-  
 35 guishable from the query applied to *similar* datasets,  
 36  $D'$ . Differential privacy uses randomized algorithms  
 37 to answer queries, typically by adding noise to the  
 38 true query results. This noise is calibrated according to  
 39 query sensitivity —how much the query result varies  
 40 between similar datasets—, turning the task of query  
 41 sensitivity computation essential for the endeavor of  
 42 differential privacy. In practice, calculating sensitivity  
 43 is not trivial and approximations are used instead [9].

44 Even though the notion of differential privacy is in  
 45 principle independent of the data model and query lan-  
 46 guage at stake, so far most practical, automated imple-  
 47 mentations over well-established languages have been  
 48 in the context of relational databases, over SQL, and  
 49 have been restricted to aggregation queries or group-  
 50 ing. Aggregations are queries such as counting, finding  
 51 maximum or average values over a certain data subset;

1 grouping is the creation of histograms based on aggre-  
 2 gations.

3 Furthermore, to allow for reasonable approxima-  
 4 tions of sensitivity, the support of these implemen-  
 5 tations for queries with *joins* has been rather lim-  
 6 ited [10]. It was only in 2018, when Johnson *et al.* [11]  
 7 introduced a new approach to approximate sensitivity  
 8 that can be applied to a wider class of SQL joins, with  
 9 reasonable results.

10 In the past decades, graph data models have enjoyed  
 11 a growing adoption in comparison to the more tradi-  
 12 tional relational model. One such notable example is  
 13 the RDF data standard, queried over by the SPARQL  
 14 language, which have become extremely popular, in  
 15 particular, for their role in the development of the Se-  
 16 mantic Web. By the simple nature of RDF, it can be  
 17 stored using binary relations [12] and most interesting  
 18 queries will require operations equivalent to joins. This  
 19 raises the question whether Johnson *et al.* approach'  
 20 [11] can also be applied to RDF and SPARQL.

21 In this paper we provide a positive answer to this  
 22 question by presenting an algorithm that can answer  
 23 counting queries over a very large class of SPARQL  
 24 queries that guarantees differential privacy. This re-  
 25 sult has been made possible by introducing the no-  
 26 tion of a *differential privacy schema* that allows us  
 27 to redefine Johnson *et al.*'s sensitivity approximation  
 28 of SQL queries in the appropriate terms for answer-  
 29 ing SPARQL queries. A differential privacy schema  
 30 groups sets of RDF tuples into sub-graphs that can be  
 31 then used as single units for privacy protection. Ex-  
 32 amples show that this type of schema naturally arises  
 33 from the semantics of the data stored in the tuples, and  
 34 it should not be difficult for a database administrator to  
 35 define.

36 We demonstrate the applicability of our approach by  
 37 implementing a differential privacy query engine that  
 38 uses the approximation to answer counting and group-  
 39 ing SPARQL queries, and evaluate the implementa-  
 40 tion running simulations using the Wikidata knowl-  
 41 edge base [13].

42 The rest of the paper is organized as follows: in Sec-  
 43 tion 2 we introduce the readers to the fundamental con-  
 44 cepts of differential privacy. We present in Section 3  
 45 the core concepts of SPARQL used within the paper.  
 46 It is in this section where we introduce the concept of  
 47 differential privacy schema. In Section 5 we prove the  
 48 correctness of our proposed approximation to sensitiv-  
 49 ity and in Section 6 we evaluate the effectiveness of  
 50 our proposed approximation in an implementation that  
 51 we apply to both synthetic and real world datasets and

queries. We present related work in Section 7, and we conclude the paper in Section 8.

## 2. Preliminaries about Differential Privacy

We now describe the framework of Differential Privacy, the problem that arises when applying Differential Privacy to SQL queries with general joins and how it has been addressed by the scientific community.

### 2.1. Definition

Intuitively, a randomized algorithm [14] is differentially private if it behaves similarly on similar input datasets. To formalize this intuition, the framework of Differential Privacy relies on a notion of *distance* between datasets. We model datasets as a *multiset of tuples* and we say that two datasets are *k-far apart* if one can be obtained from the other by changing the value of *k* tuples. Formally, this corresponds to (a mild generalization of) the notion of distance used for defining *bounded* differential privacy [15], which quantifies (only) over pairs of datasets of the same size. In the remainder, we let  $\mathbb{D}$  be the set of all possible datasets, and use  $d(D, D') = k$  to denote that  $D, D' \in \mathbb{D}$  are *k-far apart*. In particular, two datasets  $D, D' \in \mathbb{D}$  that are 1-far apart are called *neighbors*, written  $D \sim D'$ .

**Definition 1.** Let  $\epsilon, \delta \geq 0$ . A randomized algorithm  $\mathcal{A}$  is  $(\epsilon, \delta)$ -differentially private if for every pair of neighbor datasets  $D, D' \in \mathbb{D}$  and every set  $S \subseteq \text{range}(\mathcal{A})$ ,

$$\Pr[\mathcal{A}(D) \in S] \leq e^\epsilon \Pr[\mathcal{A}(D') \in S] + \delta.$$

This inequality establishes a quantitative closeness condition between  $\Pr[\mathcal{A}(D) \in S]$  and  $\Pr[\mathcal{A}(D') \in S]$ , the probabilities that on inputs  $D$  and  $D'$ , the outcome of  $\mathcal{A}$  lies within  $S$ . The smaller the  $\epsilon$  and  $\delta$ , the closer these two probabilities are, and therefore, the less likely that an adversary can tell  $\mathcal{A}(D)$  and  $\mathcal{A}(D')$  apart. In other words, parameters  $\epsilon$  and  $\delta$  quantify the privacy guarantees of the randomized algorithm.

**Multi-table datasets** Our notions of dataset and distance between datasets can be extended to collections of datasets as follows: A dataset formed by multiple sets  $D_1, \dots, D_n$  will contain (tagged) data points belonging to  $D_1, \dots, D_n$  and the distance between two such datasets  $D, D'$  will reduce to  $d(D, D') = \sum_{i=1}^n d(\pi_i(D), \pi_i(D'))$ ,  $\pi_i(D)$  representing the subset of data points from  $D$  belonging to  $T_i$ . In the context of relational databases, the  $T_i$ s correspond to relational tables.

### 2.2. Realization via Global Sensibility

Establishing Differential Privacy for numeric queries of limited sensitivity is relatively simple. The Laplacian mechanism [16] says that we can obtain a differentially private version of query  $f: \mathbb{D} \rightarrow \mathbb{R}$  by simply perturbing its output: On input  $D$ , we return  $f(D)$  plus some noise sampled from a Laplacian distribution. The noise must be calibrated according to the *global sensitivity*  $\text{GS}_f$  of  $f$ , which measures its maximum variation upon neighbor datasets; formally,  $\text{GS}_f = \max_{D, D' | D \sim D'} |f(D) - f(D')|$ .

**Theorem 1.** Given a numeric query  $f: \mathbb{D} \rightarrow \mathbb{R}$  of global sensitivity  $\text{GS}_f$ , the randomized algorithm

$$\mathcal{A}(D) = f(D) + \text{Lap}\left(\frac{\text{GS}_f}{\epsilon}\right)$$

is an  $(\epsilon, 0)$ -differentially private version of  $f$ .

Here  $\text{Lap}(\lambda)$  represents sample from the *Laplacian distribution* with parameter  $\lambda$ , a symmetric distribution with probability density function  $\text{pdf}(x) = \frac{1}{2\lambda} e^{-|x|/\lambda}$ , mean 0 and variance  $2\lambda^2$ . Parameter  $\lambda$  measures how concentrated the mass of the distribution is around its mean 0: The smaller the  $\lambda$ , the less noise we add to the true query result and therefore, the more faithful the mechanism becomes. In the realm of Differential Privacy, this “faithfulness” property is referred to as the mechanism *utility* [7]. An important point here is that utility and privacy are always conflicting requirements: adding more noise results in more private and—at the same time—less useful mechanisms.

In practice, when implementing the Laplacian mechanism we approximate the global sensibility of queries by exploiting their structures: Numeric queries are typically constructed by first transforming the original dataset using some standard transformers and by returning as final result some aggregation on the obtained dataset. For example, we join two tables, filter the result (dataset transformations) and return the count (aggregation) of the obtained table. The global sensitivity of such a query can be estimated from the so-called stability properties of the involved transformers. Intuitively, a stable transformer can increase the distance between nearby datasets at most by a multiplicative factor. Formally, we call a dataset transformer  $T: \mathbb{D} \rightarrow \mathbb{D}$   *$\alpha$ -globally-stable* if  $d(T(D), T(D')) \leq \alpha d(D, D')$  for every  $D, D' \in \mathbb{D}$ . Transformers with bounded global stability yield bounded global sensitivities:  $\text{GS}_{f \circ T} \leq \alpha \text{GS}_f$  whenever  $T$  is  $\alpha$ -globally-stable.

Conversely, the use of transformers with unbounded stability might result in queries of unbounded sensitivity. A prominent example of a transformer exhibiting this problem is join. Assume we join two tables, say  $t_1$  and  $t_2$ , by matching a pair of their attributes. A modification in a mere tuple from  $t_1$  may result in the addition and/or deletion of an unpredictable number of tuples in the result of the join, leaving elementary queries such as counting the number of tuples in a join already out of the scope of the Laplacian mechanism. The applicability of Differential Privacy approaches based on query *global* sensitivity is thus rather limited.

### 2.3. Realization via Local Sensibility

To handle queries that involve transformers of unbounded stability, such as joins, we require the use of more advanced techniques. The Laplacian mechanism calibrates noise according to the query, overlooking the fact that queries are done on concrete datasets, hence the employed noise could be potentially customized for each dataset. Nissim *et al.* show how to exploit this idea of instance-based noise [17]. Their approach relies on the notion of local sensitivity.

**Definition 2.** The local sensitivity  $\text{LS}_f(D)$  of a numeric query  $f: \mathbb{D} \rightarrow \mathbb{R}$  on dataset  $D \in \mathbb{D}$  is defined as

$$\text{LS}_f(D) = \max_{D' \mid d(D, D')=1} |f(D) - f(D')|.$$

The local sensitivity  $\text{LS}_f^{(k)}(D)$  at distance  $k \in \mathbb{N}_0$  of  $D$  is defined as

$$\text{LS}_f^{(k)}(D) = \max_{D' \mid d(D, D')=k} \text{LS}_f(D').$$

Observe that  $\text{LS}_f^{(0)}(D)$  coincides with  $\text{LS}_f(D)$ . Similar to global stability, a dataset transformer  $T: \mathbb{D} \rightarrow \mathbb{D}$ , is  $\alpha$ -local-stable for a dataset  $D$  if  $d(T(D), T(D')) \leq \alpha d(D, D')$  for every  $D' \in \mathbb{D}$ . And as with global sensitivity,  $\text{LS}_{f \circ T}(D) \leq \alpha \text{LS}_f(D)$  whenever  $T$  is  $\alpha$ -local-stable for  $D$ .

For answering a query  $f$  on dataset  $D$ , we cannot simply use noise calibrated according to  $\text{LS}_f(D)$  because the noise level itself may reveal information about  $D$  [11]. Instead, we should use an approximation of  $\text{LS}_f$  that is insensitive to small variations of its input dataset. This is captured by the notion of *smooth* upper bound.

**Definition 3.** A function  $\mathcal{U}_f: \mathbb{D} \rightarrow \mathbb{R}_{\geq 0}$  is called a  $\beta$ -smooth upper bound of the local sensitivity  $\text{LS}_f: \mathbb{D} \rightarrow \mathbb{R}_{\geq 0}$  of query  $f: \mathbb{D} \rightarrow \mathbb{R}$  if it satisfies the following requirements:

1.  $\mathcal{U}_f(D) \geq \text{LS}_f(D)$  for all dataset  $D$ , and
2.  $\mathcal{U}_f(D) \leq e^\beta \mathcal{U}_f(D')$  for all neighbor datasets  $D$  and  $D'$ .

We can readily achieve Differential Privacy by adding noise calibrated according to a smooth upper bound of the query local sensitivity [18, Corollary 2.4].

**Theorem 2.** Let  $f: \mathbb{D} \rightarrow \mathbb{R}$  be a numeric query and let  $\mathcal{U}_f: \mathbb{D} \rightarrow \mathbb{R}_{\geq 0}$  be a  $\beta$ -smooth upper bound of its local sensitivity  $\text{LS}_f$ . Moreover, let  $\delta \in (0, 1)$  and let  $\beta \leq \frac{\epsilon}{2 \ln(2/\delta)}$ . Then, the randomized algorithm

$$\mathcal{A}(D) = f(D) + \text{Lap}\left(\frac{2\mathcal{U}_f(D)}{\epsilon}\right)$$

is an  $(\epsilon, \delta)$ -differentially private version of  $f$ .

The benefits of this mechanism are twofold. On the one hand, it allows handling queries that fail to have a bounded *global* sensitivity, but *do* have a bounded *local* sensitivity. These include *e.g.* the query we considered earlier, consisting of the count of the join between two tables. On the other hand, it does not require computing the local sensitivity of the queries them self, but only a smooth upper bound thereof. This is key for its practical adoption since calculating the local sensitivity of queries is computationally prohibitive: As observed by Johnson *et al.* [11], “it requires running the query on every possible neighbor of the original dataset”.

To apply the mechanism from Theorem 2, we must provide a smooth upper bound for the local sensitivity of queries. We can construct the smooth upper bound using approximations for the local sensitivity at fixed distances.

**Lemma 1.** Let  $f: \mathbb{D} \rightarrow \mathbb{R}$  be a numeric query and assume that  $\mathcal{U}_f^{(k)}$  is a pointwise upper bound of the local sensitivity  $\text{LS}_f^{(k)}$  of  $f$  at distance  $k$ , that is,

$$\mathcal{U}_f^{(k)}(D) \geq \text{LS}_f^{(k)}(D) \quad \text{for all } D \in \mathbb{D}.$$

Then,

$$\mathcal{U}_f(D) = \max_{0 \leq k \leq \text{size}(D)} e^{-\beta k} \mathcal{U}_f^{(k)}(D)$$

1 is a  $\beta$ -smooth upper bound of the local sensitivity  
 2  $LS_f(D)$  of  $f$  on  $D$ , where  $size(D)$  denotes the number  
 3 of rows (in all the tables) in  $D$ .

4 The goal of Section 5 is to apply the Differential Pri-  
 5 vacy mechanism from Theorem 2 to SPARQL count-  
 6 ing queries. To do so, we will use Lemma 1 to derive  
 7 smooth upper bounds of the local sensitivity of queries.  
 8 In turn, this requires constructing upper bounds for the  
 9 local sensitivity of queries at fixed distances, for which  
 10 we will leverage *local* stability properties of SPARQL  
 11 dataset transformers.  
 12

### 13 3. Toward Differential Privacy over RDF Graphs

14 In this section we examine the semantic informa-  
 15 tion that is necessary considering RDF graphs, in or-  
 16 der to answer counting queries in a differentially pri-  
 17 vate manner. This comprises a data schema and upper  
 18 bounds on the predicate multiplicities.  
 19

#### 20 3.1. Privacy schema

##### 21 Motivation

22 As mentioned in the previous section, the goal of  
 23 Differential Privacy is to protect the (possibly sensible)  
 24 contribution of each individual within a dataset when  
 25 publicly releasing aggregate information about the  
 26 dataset—in our case, the result of counting queries.  
 27 In the relational model, individuals are typically *iden-*  
 28 *tified with rows* of the database which significantly  
 29 simplifies all the technical development. For instance,  
 30 if the database at stake consists of a *single* table, we  
 31 consider two instances of the database neighboring,  
 32 *i.e.* differing in the contribution of a single individual,  
 33 if they differ in a single row. On the other hand, if  
 34 the database consists of *multiple* tables, we consider  
 35 two database instances neighboring if they differ in a  
 36 row of some of the tables (see paragraph *Multi-table*  
 37 *datasets* in Section 2). The underlying assumption be-  
 38 hind this is that each table represents a type of en-  
 39 tity, *e.g.* people, political parties or companies, or part  
 40 thereof, that we want to protect.  
 41

42 To be able to apply Differential Privacy to a dataset  
 43 in the form of an RDF graph, we must thus begin by  
 44 identifying the different types of entities present in the  
 45 graph, and the set of individuals for each type of entity.  
 46 Consider, for instance, the RDF graph  $\mathcal{G}$  in Figure 1,  
 47 which will be the running example of our presenta-  
 48 tion. This graph contains information about three types  
 49 of entities: people, companies and cites. In particular,  
 50 it contains information about two people (depicted in  
 51 blue), two companies (depicted in red) and two cities  
 (depicted in green). Said otherwise, there are two indi-  
 viduals of each entity type, adding up six individuals  
 in all. When querying the graph, we will be interested  
 in protecting the contribution of all these individuals,  
 and when applying Differential Privacy techniques to  
 this end, we will then consider as a neighbor any other  
 graph that differs in the contribution of either of them.

We refer to the semantic information necessary to  
 identify the individuals in an RDF graph  $G$  as a *Dif-*  
*ferential Privacy schema*. More formally, its goal is  
 to characterize  $G$  as a set  $\{g_1, \dots, g_n\}$  of sub-graphs,  
 where each  $g_i$  represents the contribution of an individ-  
 ual, and  $G = \bigsqcup_i g_i$  is the disjoint union of all these sub-  
 graphs. For example, the graph in Figure 1 is decom-  
 posed as the disjoint union of the pair of sub-graphs  
 in blue, the pair of sub-graphs red and the pair of sub-  
 graphs green. Observe that in the relational model, this  
 corresponds to nothing more than understanding a (set  
 of) table(s) as the disjoint union of its (their) rows. Our  
 interest is to protect the identity of the entity contrib-  
 uting each  $g_i$ . In Figure 1, this means, for example, the  
 identity of Alice or Bob.

##### 22 Formal definition

To hint how we formally identify entities within an  
 RDF graph, observe that the six sub-graphs in Fig-  
 ure 1 all have a star shape [19], consisting of a center  
 with outgoing and/or incoming edges, *i.e.* predicates.  
 Moreover, sub-graphs representing individuals of the  
 same entity type, are built from the same set of pred-  
 icates. For example, both (blue) sub-graphs represent-  
 ing people are built from predicates phone, livesIn and  
 member. Therefore, we characterize each type of en-  
 tity through a set of triple patterns, *i.e.* a basic graph  
 pattern (BGP), which share a common or “join” vertex.  
 Formally, a *join vertex* in a BGP is a variable that ap-  
 pears either as a subject or as an object multiple times  
 in the BGP. The type of BGP’s that we employ have  
 further restrictions, which are captured by the notion  
 of *star BGP* below:

**Definition 4** (Star BGP). *A BGP is called a star if*

1. both the subject and the object of all its triple pat-  
 terns are variables,
2. it contains no repeated predicate, and
3. it consists either of
  - (a) a single triple pattern with no join vertex,  
*i.e.* a triple pattern whose subject and object  
 are distinct variables, or of

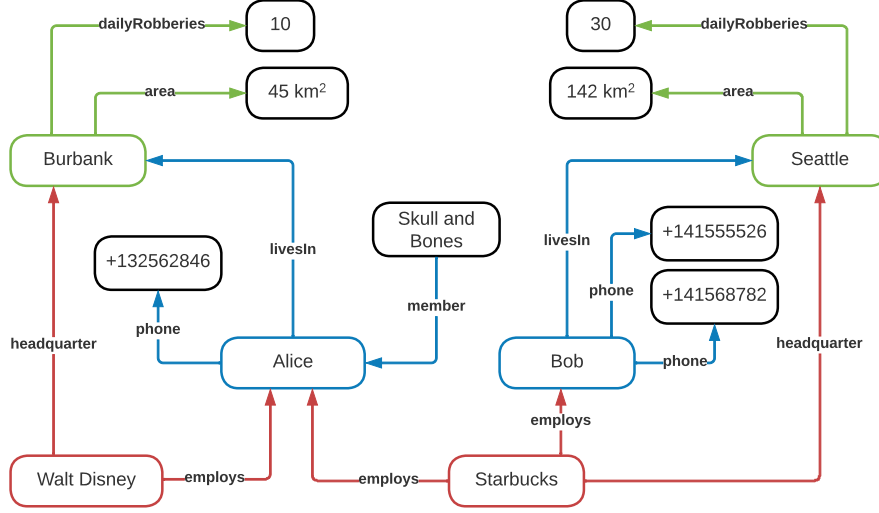


Figure 1. RDF graph  $\mathcal{G}$  containing information about three types of entities: people, companies and cites.

(b) multiple triple patterns with a single join vertex, which appears once and only once in every triple pattern

**Example 3.1.** The three stars employed to identify the different entities in our running example (Figure 1) are (modulo variable renaming):

$$S_1 = \{(?c_1, \text{livesIn}, ?o_1), (?c_1, \text{phone}, ?o_2), (?s_2, \text{member}, ?c_1)\}$$

$$S_2 = \{(?c_2, \text{employs}, ?o_3), (?c_2, \text{headquarter}, ?o_4)\}$$

$$S_3 = \{(?c_3, \text{area}, ?o_5), (?c_3, \text{dailyRobberies}, ?o_6)\}$$

Observe that in each  $S_i$ , the vertex, denoted  $?c_i$ , plays the role of the star center, joining all the triple patterns in  $S_i$ . Formally, we define the *center* of a star as the join vertex if the star contains multiple patterns, or the variable appearing in the subject of the triple pattern in case it consists of a single triple pattern.<sup>2</sup> Furthermore, observe that there are no common predicates across  $S_1$ ,  $S_2$  and  $S_3$ . We say that a set of stars are *pairwise predicate-disjoint*, if no pair of stars in the collection share a common predicate. They thus define what we call a Differential Privacy schema:

<sup>2</sup>The notion of *star BGP* that we use here is similar to that of *star query* from [19], except that in a star query the center of the star must always appear as a subject.

**Definition 5** (dp-schema). A Differential Privacy schema (dp-schema, for short)  $\mathcal{P}$  is a finite set of pairwise predicate-disjoint stars.

If no ambiguity is created, we will refer to a set of pairwise predicate-disjoint stars, simply as pairwise disjoint.

The set consisting only of  $S_1$  and  $S_2$  forms a dp-schema, as well. However, this schema falls short of describing the whole graph  $\mathcal{G}$ , as it leaves out the information related to cities. To formally capture this completeness condition, we require the notion of *induced sub-graph*. Given a star BGP  $S$ , and a solution mapping  $\mu$  over an RDF graph  $G$ , we can take the star  $S$  and replace each variable in  $S$  according to  $\mu$  and drop any triple pattern that contains a variable after the replacement. The resulting set is a subset of the RDF triples defining  $G$ . We call any subset constructed this way an induced sub-graph over  $G$  by the star BGP  $S$ . There will be one induced sub-graph per solution mapping.

**Example 3.2.** The star  $S_1$  induces two sub-graphs  $g_1$  and  $g_2$  over  $\mathcal{G}$ , i.e.  $S_1(\mathcal{G}) = \{g_1, g_2\}$ . These correspond to the sub-graph depicted in blue in Figure 1,

which are formally defined as:

$$g_1 = \{(Alice, livesIn, Burbank), \\ (Alice, phone, +132562846), \\ (Skull and Bones, member, Alice)\}$$

$$g_2 = \{(Bob, livesIn, Seattle) \\ (Bob, phone, +141555526), \\ (Bob, phone, +141568782), \}$$

Likewise,  $S_2(\mathcal{G}) = \{g_3, g_4\}$ , where

$$g_3 = \{(Walt Disney, employs, Alice), \\ (Walt Disney, headquarter, Burbank)\}$$

$$g_4 = \{(Starbucks, employs, Alice), \\ (Starbucks, employs, Bob), \\ (Starbucks, headquarter, Seattle)\}$$

Intuitively, the set of induced sub-graphs  $S(G)$  can be recovered by evaluating  $S$  over  $G$ , but assuming that the triple patterns in  $S$  are optional. This assumption is already exposed by the above example, where the triple pattern  $(?s_2, member, ?c_1)$  belongs to  $S_1$ , but it is not materialized in  $g_2$ . From the privacy point of view, the values of the star center are the unique identifiers of the entities contributing the data in each sub-graph, and must be kept confidential.

The notion of induced sub-graphs naturally extends from single stars, to dp-schemas, *i.e.* sets of stars. Concretely, we let  $\mathcal{P}(G) = \bigcup_{S \in \mathcal{P}} S(G)$  be the set of sub-graphs of  $G$  induced by dp-schema  $\mathcal{P}$ . For example,  $\{S_1, S_2\}(\mathcal{G}) = \{g_1, g_2, g_3, g_4\}$ .

Now we have all the prerequisites to define the core concept of this subsection:

**Definition 6** (Dp-schema compliance). *We say that an RDF graph  $G$  complies with a dp-schema  $\mathcal{P}$  iff  $G$  coincides with the graph induced by  $\mathcal{P}$  in  $G$ , *i.e.* if  $G = \bigcup_{g \in \mathcal{P}(G)} g$ .*

**Example 3.3.** *Our running example  $\mathcal{G}$  complies with dp-schema  $\{S_1, S_2, S_3\}$ . In contrast, it would not comply with dp-schema  $\{S_1, S_2\}$ .*

In summary, if a graph  $G$  complies with a dp-schema  $\mathcal{P}$ , the schema partitions the graph into a finite set  $\mathcal{P}(G)$  of sub-graphs, which intuitively model the different individuals in the graph. The fact that sub-

graphs are disjoint follows from the definition of dp-schema, which requires stars in the schema to be pairwise predicate disjoint, and that all the occurrences of variables in a star are different except for the center of the star. The fact that the set of sub-graphs cover the entire graph follows from the definition of compliance.

#### Discussion

We now address a few key points about dp-schemas.

*No shared attribute between entities.* At first sight, the requirement that stars in a dp-schema be disjoint might seem a limitation, as it requires that each attribute belong to a single entity type. For instance, it might seem natural to consider that predicate *employs* should be part of both the employer and the employee, and it should be thus present in both  $S_2$  (which identifies companies) and  $S_1$  (which identifies people). However, for the sake of protection it makes no difference to which star it belongs, since our application of Differential Privacy will protect the contribution of all individuals, regardless of its type.

*Existence of dp-schemas.* RDF graphs always admit compliant dp-schemas. In particular, every graph complies with a *trivial* dp-schema consisting in the union of all singleton BGP's of the form  $\{(?x, p, ?y)\}$ , where  $p$  ranges over the set of predicates appearing in  $G$ . Intuitively, this schema indicates that each RDF triple is the contribution of a different individual. Even though this is a valid dp-schema, it will yield very weak privacy guarantees. In general, database administrators should provide dp-schemas with stars having all their triples since, as we will see later, it will allow better approximations of the local sensitivity of queries, and thus, better privacy guarantees.

*Compliance verification.* Checking whether an RDF graph complies with a given schema  $\mathcal{P}$  is algorithmically straightforward as it amounts to verifying that all predicates in the graph appear also in (some star of) the schema (which can be done in  $O(N)$  in the worse case using a hashing algorithm).

*Dp-schema provision.* For practical purposes, we assume that the database administrator of the RDF graph at stake is responsible for designing the dp-schema the graph shall comply with, and for ensuring the compliance as the graph evolves. In this latter regard, observe that removing an RDF triple from the graph always preserves the dp-schema compliance, and adding a triple also preserves compliance provided the predicate in the triple already appears in the dp-schema.



We believe this is a natural assumption, as in the relational data model this would correspond to changing the schema of the database by adding a new attribute to a relation if the new predicate is incorporated into an existing star of the Dp-schema or creating a new table if the predicate is added to the Dp-schema as a new star.

### 3.2. Predicate multiplicity

Automatic approaches to answer dataset queries in a differentially private manner are typically obtained by adding noise to the query results, calibrated according to their sensitivity. Thus, a prerequisite to apply Differential Privacy to counting queries over RDF graphs is that they have bounded sensitivity. Unfortunately, this does not occur in the general case.

To see this, consider graph  $\mathcal{G}$  of our running example and query “How many phone numbers are currently in use?”. If we evaluate the query over  $\mathcal{G}$ , the answer is 3. Now assume we consider a neighboring graph  $\mathcal{G}'$ , where Bob’s contribution (*i.e.* sub-graph  $g_2$ ) is replaced by somebody else’s contribution. The query answer over this neighboring graph can certainly be any integer  $n \geq 1$ , since a priori we do not know how many phone numbers this new individual might have. Therefore, the sensitivity of the query becomes unbounded.

This problem arises because of the presence of predicates that are not *one-to-one*. To recover bounded sensitivities, we have to restrict ourselves to predicates that have bounded multiplicity. For instance, if the administrator of graph  $\mathcal{G}$  requires that individuals declare at most 5 phone numbers, then the above query will have a local sensitivity of at most 5 (recall that if  $n_1, n_2 \leq \alpha$ , then  $|n_1 - n_2| \leq \alpha$ ). This approach was already taken by other authors [9, 20] to bound the sensitivity of counting queries (in the presence of *joins*), and is the price one has to pay to apply Differential Privacy over RDF graphs.

On the formal level, we associate such bounds to triple patterns rather than to predicates. This is because in the presence of compliant dp-schemas, predicates are identified with triple patterns (every predicate within a dp-schema occurs in a single triple pattern, in a single star).

**Definition 7** (Triple-pattern multiplicity). *Let  $G$  be an RDF graph that complies with a dp-schema  $\mathcal{P}$ . A multiplicity bound  $\kappa$  associates to each triple pattern  $tp$  in a star  $S$  of  $\mathcal{P}$  an integer  $\kappa(tp)$  that upper-bounds the number of solution mappings  $\mu \in \llbracket tp \rrbracket_G$  with the same image for the center of  $S$ .*

Observe that many predicates (or equivalently, triple patterns) have a natural multiplicity bound of 1. For instance, a city has a unique area and a unique number of dailyRobberies. Likewise, we can assume that a person lives in a single city (at least for formal purposes). On the other hand, if a predicate does not admit a natural bound for its multiplicity, think *e.g.* of predicate friend, we can either a) choose an upper bound that cover most of the cases in practice or b) establish an upper bound in accordance with the size of the graph that the administrator is willing to support.

Henceforth, in the remainder we assume that every graph  $G$  is accompanied by a compliant dp-schema  $\mathcal{P}$ , which is provided by the graph administrator. Furthermore, we assume that the administrator establishes an upper bound  $\kappa(tp)$  for each triple pattern  $tp$  in  $\mathcal{P}$ . The graph space that we consider for the purposes of Differential Privacy will be that of graphs that comply with both  $\mathcal{P}$  and  $\kappa$ .

For bounding the local sensitivity of queries in Section 5, it will suffice a coarser notion of multiplicity, at the level of stars rather than triple patterns. The required generalization is straightforward:

**Definition 8** (Star multiplicity). *Let  $G$  be an RDF graph that complies with a dp-schema  $\mathcal{P}$  and has a multiplicity bound  $\kappa$ . We call the multiplicity of a star  $S \in \mathcal{P}$ , by notation convenience written  $\kappa(S)$ , to the product of the multiplicity bound of the triple patterns in  $S$ , *i.e.*  $\kappa(S) = \prod_{tp \in S} \kappa(tp)$ .*

**Example 3.4.** *Assume that a graph administrator adopts dp-schema  $\mathcal{P} = \{S_1, S_2, S_3\}$  and requires that each individual lives in (at most) a single city, declare at most five phone numbers and be member of at most 3 secret societies. Then the star  $S_1$  will have multiplicity  $\kappa(S_1) = 1 \times 5 \times 3 = 15$ .*

## 4. Queries

In this section we describe the subset of queries over RDF graphs for which we provide Differential Privacy, and show how dp-schemes enable a decomposition result for the evaluation of such queries.

### 4.1. Supported queries

We develop Differential Privacy for counting queries over the SPARQL fragment of basic graph patterns with filter expressions, also known as *constrained ba-*



1 *sic graph pattern* (CBGP) [19]. In this fragment, a  
2 query is denoted by a pair

$$3 \quad \bar{B} = \langle B, F \rangle,$$

4 where  $B$  is a finite set of triple patterns, *i.e.* a BGP,  
5 and  $F = \{f_1, \dots, f_n\}$  is a finite (possibly empty) set  
6 of filter expressions.  $\bar{B}$  represents the SPARQL graph  
7 pattern  
8

$$9 \quad P = ((\dots (B \text{ FILTER } f_1) \dots) \text{ FILTER } f_n),$$

10 and its meaning over an RDF graph  $G$ , denote by  $\llbracket \bar{B} \rrbracket_G$ ,  
11 is the multiset of solution mappings  $\llbracket P \rrbracket_G$  as defined by  
12 the standard semantics of SPARQL queries [21].

13 For simplicity, we consider only CBGPs that are se-  
14 mantically valid. We also assume that in a graph  $G$  that  
15 complies with a dp-schema  $\mathcal{P}$ , all predicates appearing  
16 in triple patterns of a CBGP also appear in  $\mathcal{P}$ .

17 **Example 4.1.** Take RDF graph  $\mathcal{G}$ , which complies  
18 with dp-schema  $\mathcal{P} = \{S_1, S_2, S_3\}$ . Now assume we  
19 want to know how many people have a coworker that  
20 lives in a city with over 20 daily robberies. The query  
21 can be cast in terms of the CBGP  $\bar{B} = \langle B, F \rangle$ , where

$$22 \quad B = \{(?x, \text{employs}, ?p_1), (?x, \text{employs}, ?p_2),$$

$$23 \quad \quad (?p_2, \text{livesIn}, ?c), (?c, \text{dailyRobberies}, ?n)\}$$

$$24 \quad F = \{?n \geq 20\}$$

25 Triple patterns in an RDF graph compliant with  
26 a dp-schema  $\mathcal{P}$  naturally inherit the notion of center  
27 from the star they “belong to”. Specifically, for a  
28 star  $S \in \mathcal{P}$  and a triple pattern  $tp = (s, p, o)$  such  
29 that  $(X, p, Y)$  belongs to  $S$ , for some variables  $X, Y$ , we  
30 let  $center(tp) = s$  if  $X$  is the center of  $S$ ; otherwise  
31  $center(tp) = o$ . For instance, in the above example we  
32 have

$$33 \quad center((?x, \text{employs}, ?p_1)) = ?x$$

$$34 \quad center((?x, \text{employs}, ?p_2)) = ?x,$$

35 since star  $S_2$  which contains the triple pattern

$$36 \quad (?c_2, \text{employs}, ?o_3)$$

37 has center  $?c_2$ . Note that center is a well-defined func-  
38 tion because the predicate of any triple pattern can only  
39 appear in single star of the dp-schema. Finally, a *user*  
40 *query* is a CBGP  $\bar{B}$  wrapped by either of the two fol-  
41 lowing aggregation operations:  
42

1.  $\text{COUNT}^{?x}(\bar{B})$ , where  $center(t) = ?x$  for a triple  
2 pattern  $t \in \bar{B}$  with its semantics,  $\llbracket \text{COUNT}^{?x}(\bar{B}) \rrbracket_G$ ,  
3 defined as the cardinality of the multiset  $\llbracket \bar{B} \rrbracket_G$ .
4.  $\text{COUNT}^{\text{DISTINCT } ?x}(\bar{B})$ , where  $center(t) = ?x$  for  
5 a triple pattern  $t \in \bar{B}$  with its semantics,  
6  $\llbracket \text{COUNT}^{\text{DISTINCT } ?x}(\bar{B}) \rrbracket_G$ , defined as the cardinal-  
7 ity of the set  $\{\mu(?x) \mid \mu \in \llbracket \bar{B} \rrbracket_G\}$ .
8.  $\text{COUNT}_{?x_1, \dots, ?x_n}^{?x}(\bar{B})$ , where  $?x, ?x_1, \dots, ?x_n$  are vari-  
9 ables appearing in  $\bar{B}$ , and where  $center(t) = ?x$   
10 for a triple pattern  $t \in \bar{B}$  with its semantics,  
11  $\llbracket \text{COUNT}_{?x_1, \dots, ?x_n}^{?x}(\bar{B}) \rrbracket_G$ , defined by grouping the  
12 solution mappings from  $\llbracket \bar{B} \rrbracket_G$  according to (the  
13 values they assign to) variables  $?x_1 \dots ?x_n$  and re-  
14 turning the number of mappings within each of  
15 the resulting groups. Loosely speaking, this cor-  
16 responds to a histogram over tuples grouped by  
17 keys created by the different combinations of the  
18 values assigned to variables  $?x_1 \dots ?x_n$  by the so-  
19 lution mappings from  $\llbracket \bar{B} \rrbracket_G$ .

20 **Example 4.2.** The query from the previous example  
21 can be expressed as  $\text{COUNT}^{?x} \bar{B}$  using the previous  
22 CBGP.  
23

#### 24 4.2. Evaluation decomposition

25 Continuing with the previous example, assume we  
26 want to evaluate  $B$  (from Example 4.1) over  $\mathcal{G}$  (from  
27 Figure 1), that is, to compute  $\llbracket B \rrbracket_G$  (observe that if  
28 we are interested in obtaining  $\llbracket \langle B, F \rangle \rrbracket_G$  instead, we  
29 simply add a FILTER operation on top of the eval-  
30 uation of  $\llbracket B \rrbracket_G$ ). We can do this in a compositional  
31 fashion, leveraging the partition that dp-schema  $\mathcal{P} =$   
32  $\{S_1, S_2, S_3\}$  induces on  $\mathcal{G}$ . For instance, we could split  
33  $B$  as  
34

$$35 \quad B_1 = \{(?x, \text{employs}, ?p_1), (?x, \text{employs}, ?p_2)\}$$

$$36 \quad B_2 = \{(?p_2, \text{livesIn}, ?c)\} \quad (1)$$

$$37 \quad B_3 = \{(?c, \text{dailyRobberies}, ?n)\}$$

38 Hence, for any query  $\bar{B} = \langle B, F \rangle$ , we can formally  
39 define a split  $B_{\div \mathcal{P}}$  of  $B$  from a dp-schema  $\mathcal{P}$ , as fol-  
40 lows:  $B_{\div \mathcal{P}} = \{B_1, \dots, B_n\}$  iff  
41

- 42 1. every  $B_i \in B_{\div \mathcal{P}}$  is a maximal subset of  $B$   
43 for which there exists a star  $S \in \mathcal{P}$  such that  
44  $pred(B_i) \subseteq pred(S)$ , and
- 45 2. for any two triple patterns  $tp, tp' \in B_i$ ,  $center(tp) =$   
46  $center(tp')$ .

Because the stars in  $\mathcal{P}$  are predicate disjoint and the  $B_i$ 's in  $B_{\div\mathcal{P}}$  are maximal, the split  $B_{\div\mathcal{P}}$  is unique. In the context of Condition 1 above, we call  $S$  the *covering star* of  $B_i$ . Moreover, we call a BGP  $B$  *elementary* if  $|B_{\div\mathcal{P}}| = 1$ .

**Example 4.3.** For the CBGP from Example 4.1, we have  $B_{\div\mathcal{P}} = \{B_1, B_2, B_3\}$ , where  $B_1, B_2, B_3$  are defined in Equation 1 above. Moreover, the covering stars of  $B_1, B_2$  and  $B_3$  are  $S_2, S_1$  and  $S_3$ , respectively.

If  $B$  were extended e.g. with triple pattern

$$tp = (\text{Skull and Bones}, \text{member}, ?p_1),$$

the splitting  $B_{\div\mathcal{P}}$  would contain a fourth member  $B_4 = \{tp\}$ , with covering star  $S_1$ . In this case, note that despite sharing the same covering star,  $S_1$ ,  $B_2$  and  $B_4$  remain different members of  $B_{\div\mathcal{P}}$  because their triple patterns have different centers ( $?p_2$  is the center of triple patterns in  $B_2$  and  $?p_1$  the center of the triple pattern in  $B_4$ ). Finally,  $B_1, B_2, B_3$  and  $B_4$  (and all their non-empty subsets) are elementary BGPs.

In general, all members of a split will be elementary. Note also that the elementary BGP where a triple pattern belongs to is determined by the center of the triple pattern, all triple patterns in the same split must share the same variable or RDF term as their center. The interest in  $B_{\div\mathcal{P}} = \{B_1, \dots, B_n\}$  resides in that it lets us isolate the fragment of the graph necessary to answer each  $B_i$ . Assume we denote by  $G_{S_i}$  the subgraph induced by star  $S_i$ , i.e.  $G_{S_i} = \bigcup_{g \in S_i(G)} g$ .

**Lemma 2.** If  $S_i$  is the covering star of  $B_i$  then

$$\llbracket B_i \rrbracket_G = \llbracket B_i \rrbracket_{G_{S_i}}$$

Then, following the terminology defined in [22] for joins between multisets of solution mappings, we can extend the lemma to  $B$  as follows:

**Lemma 3.**

$$\llbracket B \rrbracket_G = \llbracket B_1 \rrbracket_{G_{S_1}} \bowtie (\llbracket B_2 \rrbracket_{G_{S_2}} \bowtie \dots \bowtie (\llbracket B_{n-1} \rrbracket_{G_{S_{n-1}}} \bowtie \llbracket B_n \rrbracket_{G_{S_n}}))$$

Recall that  $\bar{B}$  is such that  $\llbracket B \rrbracket_G$  can be evaluated using only equi-joins. Therefore, there must exist an ordering of the elements in  $B_{\div\mathcal{P}}$  such that  $\llbracket B_i \rrbracket_{G_{S_i}} \bowtie \llbracket B_{i+1} \rrbracket_{G_{S_{i+1}}}$  can also be done with equi-joins. In other words, an ordering where  $|\text{var}(B_i) \cap \text{var}(B_{i+1})| = 1$  for all  $1 \leq i < n$ . We call this order a *normal order-*

*ing* of  $B_{\div\mathcal{P}}$ , and without loss of generality denote by  $?x_i$  the variable in the equi-join  $\llbracket B_i \rrbracket_{G_{S_i}} \bowtie \llbracket B_{i+1} \rrbracket_{G_{S_{i+1}}}$ . For convenience, in the remainder we assume that the indexing used for  $B_{\div\mathcal{P}}$  follows a normal order. Note that this is already the case for the splitting from Example 4.3 ( $B_1$  and  $B_2$  share variable  $p_2$ , and  $B_2$  and  $B_3$  share variable  $c$ ).

We are now in a position to establish Differential Privacy for SPARQL count and histogram queries.

## 5. Towards Differential Privacy for SPARQL

In this section we develop all the prerequisites to extend Lemma 1 from the relational model to the graph model, in terms of the SPARQL queries over RDF graphs described in the previous section.

### 5.1. Preliminary notions

We begin defining the notion of size and distance between RDF graphs. These are straightforward adaptations of the relational case, where the induced subgraphs play the role of table rows. Concretely, the size of a graph refers to the number of individuals present in it. Formally, given an RDF graph  $G$  that complies with a dp-schema  $\mathcal{P} = \{B_i\}_{i \in I}$ , we define the *size* of  $G$  w.r.t.  $\mathcal{P}$  as  $\text{size}(G)_{\mathcal{P}} = \sum_{i \in I} |B_i(G)|$ .

Moreover, we say that two graphs are *k far apart* if one can be obtained from the other by replacing  $k$  of its induced sub-graphs. Formally, given a pair of RDF graphs  $G_1, G_2$  that comply with a dp-schema  $\mathcal{P} = \{B_i\}_{i \in I}$  and such that  $\text{size}(G_1)_{\mathcal{P}} = \text{size}(G_2)_{\mathcal{P}}$ , their *distance* is defined as the size of their difference, i.e.  $d(G_1, G_2)_{\mathcal{P}} = \text{size}(G_1 \setminus G_2)_{\mathcal{P}}$ . Note that in the general case where the sizes of  $G_1$  and  $G_2$  need not coincide, their distance is defined as the size of their symmetrical difference  $(G_1 \setminus G_2) \cup (G_2 \setminus G_1)$ , but when the sizes coincide, this reduces to the size of either their differences, making distance commutative as expected.

Finally, this notion of distance between RDF graphs readily induces a notion of *local sensitivity* (at distance  $k$ )  $\text{LS}_Q(G)$  ( $\text{LS}_Q^{(k)}(G)$ ) of SPARQL query  $Q$  over RDF graph  $G$ , as given by Definition 2.

In order not to clutter the presentation, we usually omit the underlying dp-schema when referring to the size of an RDF graph, the distance between a pair of RDF graphs, and the local sensitivity of a SPARQL query if the dp-schema is understood from the context.

## 5.2. Elastic sensitivity

Our next step is, given a user query  $Q$  and an RDF graph  $G$  that complies with a dp-schema  $\mathcal{P}$ , to compute an upper bound of the local sensitivity  $\text{LS}_Q^{(k)}(G)$  (denoted by  $\mathcal{U}_Q^{(k)}(G)$  in Lemma 1).

To this end, observe that the naive approach of evaluating the query on every neighbor of  $G$  is not a feasible solution, since the number of neighbors can be extremely large. To address this problem in the relational setting, Johnson *et al.* [11] has introduced the notion of *elastic sensitivity*, which leverages (maximum) frequency values of the join keys (that can be precomputed or statistically estimated) to provide more efficient upper bounds for the local sensitivity of queries with joins.

In the remainder of the section we adapt Johnson *et al.*'s approach to the case of RDF graphs and SPARQL. Intuitively, our notion of elastic sensitivity of a SPARQL query  $Q$  at distance  $k$  of a concrete graph  $G$  (that complies with the dp-schema  $\mathcal{P}$ ) regards the evaluation of  $Q$  as the composition of successive transformations applied to  $G$ , and is defined in terms of the stability properties of such transformations.

We thus introduce the auxiliary notion of *elastic stability*. In our case, the elastic stability of a SPARQL transformation CBGP is determined by its BGP part, only. A key property of this notion is that it allows bounding the local sensitivity of counting queries: Given a BGP  $B$ , its elastic stability at distance  $k$  with respect to a graph  $G$  that complies with a dp-schema  $\mathcal{P}$ , bounds the local stability for any graph  $G'$  that also complies with  $\mathcal{P}$  and is at distance  $k$  of  $G$ . Hence, it bounds the local sensitivity at distance  $k$  of  $\text{COUNT}^{?x}(B)$  (for any  $B = \langle B, F \rangle$ ) over  $G'$ .

The formal definition of elastic stability relies on the frequency of most popular values. More precisely, if  $?x$  is a variable occurring in a BGP  $B$ , we use  $mpv(?x, B, G)$  to denote the frequency of the most popular value to which  $?x$  is mapped, when evaluating  $B$  over  $G$ . In practice, we use SPARQL itself to determine  $mpv(?x, B, G)$  through the query

```
SELECT (COUNT(?x) as ?c) WHERE B
GROUP_BY ?x ORDER_BY ?c DESC LIMIT 1
```

Loosely speaking, this corresponds to first evaluating  $\text{COUNT}^{?x}(B)$ , and then selecting the value with the largest count. Observe that this result is also an upper bound for the frequency of the most popular value

yielded by the CBGP  $\bar{B} = \langle B, F \rangle$  for any  $F$ , since we are ignoring the filter  $F$  that reduces the size of the result. Finally, observe that these frequencies can be pre-computed as they are independent of any query.

To compute the the elastic stability of BGP  $B$  at distance  $k$  of graph  $G$ , written  $\mathcal{S}_B^{(k)}(G)$ , we start by applying Lemma 3 to decompose  $B$  as a sequence of elementary BGPs. Once we fix a normal ordering, we have  $B_{\dot{\mathcal{P}}} = B_1 \bowtie (B_2 \bowtie (\dots \bowtie B_n) \dots)$ . This decomposition allows estimating the frequency of most popular values of graphs  $k$  far apart from  $G$ . Formally, the frequency of the most popular values for variable  $?x$  in a BGP  $B$  for graphs at distance  $k$  of  $G$ , written  $mpv^{(k)}(?x, B, G)$ , is defined by induction on the length  $|B_{\dot{\mathcal{P}}}|$  as follows:

**Base case:** If  $|B_{\dot{\mathcal{P}}}| = 1$ , we let

$$mpv^{(k)}(?x, B, G) = mpv(?x, B, G) + k \times \kappa(S),$$

where  $S$  is the covering star of  $B_1$  (recall that, in this base case,  $B_{\dot{\mathcal{P}}} = \{B_1\}$ ).

**Inductive case:** If  $|B_{\dot{\mathcal{P}}}| > 1$ , we let

$$mpv^{(k)}(?x, B, G) = mpv^{(k)}(?x_1, B_{\dot{\mathcal{P}}} \setminus \{B_1\}, G) \\ \times mpv^{(k)}(?x, B_1, G),$$

where  $?x_1$  is the common variable shared by  $B_1$  and  $(B_2 \bowtie (\dots \bowtie B_n) \dots)$ , used for their equi-join.

The intuition behind the base case is easy to grasp. For  $k = 1$ , we take a star instance from  $G$  and replace it from a different one. The largest effect this change can have on the count of the most popular mapping value of  $?x$  is adding  $\kappa(S)$ , which is an upper bound of all the mappings that can be produced by the instance due to the triple-pattern multiplicity, if the instance removed didn't have an instance of the value and a new instance of the same value is added. Hence,  $k$  changes will at most increment the count by  $k \times \kappa(S)$ . For the inductive case, we need to worry about the most frequent mapping value of  $?x_1$  in  $B_{\dot{\mathcal{P}}} \setminus \{B_1\}$  since for every mapping of  $?x$  obtained from  $B_1$ , if this mapping maps  $?x_1$  in  $B_1$  to the same value of the most frequent value of  $x_1$  in  $B_{\dot{\mathcal{P}}} \setminus \{B_1\}$ , the value of  $?x$  will be repeated as many times in the combined mapping of  $B$ . Hence, the most frequent value of  $?x$  in  $\llbracket B_1 \rrbracket_{S_1}$  can be duplicated, in the worst case, as many as  $mpv^{(k)}(?x_1, B_{\dot{\mathcal{P}}} \setminus \{B_1\}, G)$  times in the multiset of solution mappings  $\llbracket B \rrbracket_G$ , giving us a safe upper bound

for the count. Importantly, observe that because the multiplication operation is commutative, the frequency is not affected by the selected normal ordering.

We are now ready to define the elastic stability of a BGP  $B$  at distance  $k$  of graph  $G$ , denoted by  $\mathcal{S}_B^{(k)}(G)$ . The definition also proceeds by induction on the cardinality of a fixed normal ordering of  $B_{\dot{\mathcal{P}}} = B_1 \bowtie (B_2 \bowtie (\dots \bowtie B_n) \dots)$ :

**Base case:** If  $|B_{\dot{\mathcal{P}}}| = 1$ , we let

$$\mathcal{S}_B^{(k)}(G) = \kappa(S),$$

where  $S$  is the covering star of  $B_1$ .

**Inductive case:** If  $|B_{\dot{\mathcal{P}}}| > 1$ , let  $B' = B_{\dot{\mathcal{P}}} \setminus \{B_1\}$ . We have two cases. If the covering star  $S$  of  $B_1$  is not the covering star of any other  $B_j \in B'$ , we let

$$\mathcal{S}_B^{(k)}(G) = \max \left\{ \begin{array}{l} mpv^{(k)}(?x_1, B_1, G) \times \mathcal{S}_{B'}^{(k)}(G) \\ mpv^{(k)}(?x_1, B', G) \times \mathcal{S}_{B_1}^{(k)}(G) \end{array} \right\} \quad (2)$$

If the covering star  $S$  of  $B_1$  is also the covering star of another  $B_j \in B'$ , we let

$$\begin{aligned} \mathcal{S}_B^{(k)}(G) &= mpv^{(k)}(?x_1, B_1, G) \times \mathcal{S}_{B'}^{(k)}(G) + \\ &\quad mpv^{(k)}(?x_1, B', G) \times \mathcal{S}_{B_1}^{(k)}(G) + \\ &\quad \mathcal{S}_{B_1}^{(k)}(G) \times \mathcal{S}_{B'}^{(k)}(G) \end{aligned}$$

Loosely speaking, this definition captures the amount of changes that the transformations (i.e. the joins) within the query, add to the final result, when modifying a single element in the induced schema.

As so defined, the local stability bounds the local sensitivity of counting queries:

**Lemma 4.** For any CBGP query  $\bar{B} = \langle B, F \rangle$ , any  $k \in \mathbb{N}$  and any graph  $G$  compliant with  $dp$ -schema  $\mathcal{P}$ :

$$\mathcal{S}_B^{(k)}(G) \geq \text{LS}_{\text{COUNT}^{?x}(\bar{B})}^{(k)}(G).$$

The main intuition behind the proof is that changes made to a graph to get a new graph at distance 1, are limited to a sub-graph  $g_i$ , that must be covered by a single star pattern  $S$  in  $\mathcal{P}$ . Then the maximum number of RDF tuples that can change to get the graph at distance 1 is limited by the multiplicity of the predicates in  $S$ . Therefore, the change in the number of

mappings obtained from the new graph of an elementary BGP covered by  $S$  is bounded by  $\kappa(S)$ . If these RDF triples contribute in the result mappings of a join vertex, the number of new mappings can increase by as much as the frequency of the most popular result mapping of the joining triple pattern. For example, if  $B = \{(?v_0, p, ?u), (?u, p', ?v_1)\}$ , and the triple  $(s, p, o)$  is part of  $g_1$  and  $o$  happens to be the most popular result mapping for  $(?u, p', ?v_1)$ , then there will be at most  $mpv^{(1)}(?u, (?u, p', ?v_1), G)$  new mappings in the result.

*Proof.* The proof of this lemma follows the same strategy as the proof in [11, Lemma 2], and is by induction on the length of  $B_{\dot{\mathcal{P}}}$ .

- **Case**  $|B_{\dot{\mathcal{P}}}| = 1$ . Let  $S$  be the covering star of  $B$ . Hence, its elastic stability is  $\kappa(S)$ , a parameter given by the DBA. Thus, we have

$$\kappa(S) = \mathcal{S}_B^{(k)}(G) \geq \text{LS}_{\text{COUNT}^{?x}(\bar{B})}^{(k)}(G)$$

since the local sensitivity at distance  $k$  is calculated as the max of the sensitivities of all graphs at distance 1 of all graphs at distance  $k$  or less of  $G$ , meaning the modification of a single star, which may change by at most  $\kappa(S)$  tuples and the filter in  $\bar{B}$  doesn't affect its local stability.

- **Case**  $|B_{\dot{\mathcal{P}}}| = n + 1$ : we have a covering star  $S_1$  for partition  $B_1$  and a set with  $n$  covering stars for partitions  $B' = \{B_2, \dots, B_{n+1}\}$ . We want to bound the number of RDF triples that can change in graphs  $G'$  at distance  $k$  of  $G$  to get a graph at distance 1, based on the star multiplicities. First, let's assume  $S_1$  is not the covering star of any other  $B_i$  in  $B'$ . Hence, changes can happen in either  $G'_{S_1}$  or in a graph  $G'_{S'}$ , induced by a star  $S'$  different from  $S_1$  that covers some other other  $B_i \in B'$ , but not in both graphs since the new graph must be at distance 1 from  $G'$ . Thus, either  $\mathcal{S}_{B_1}^{(k)}(G) = 0$  or  $\mathcal{S}_{B'}^{(k)}(G) = 0$ :

1. When  $\mathcal{S}_{B_1}^{(k)}(G) = 0$ , the changes in  $G'_{S'}$ , by induction hypothesis using Eq(2), produce at most  $mpv^{(k)}(?x_1, B_1, G) \times \mathcal{S}_{B'}^{(k)}(G)$  changed mappings since one change in  $G'_{S'}$  might affect at most  $mpv^{(k)}(?x_1, B_1, G)$  triplets in the same join when applied to a graph at distance 1 of  $G'$ .
2. In the symmetric case, when  $\mathcal{S}_{B'}^{(k)}(G) = 0$ ,  $G'_{S_1}$  may contain  $\mathcal{S}_{B_1}^{(k)}(G) = \kappa(S)$  changed triplets, producing at most

1  $mpv^{(k)}(?x_1, B', G) \times \mathcal{S}_{B_1}^{(k)}(G)$  changed map-  
 2 pings in the joined SPARQL pattern.

3 We chose the largest of the two values when cal-  
 4 culating  $\mathcal{S}_B^{(k)}(G)$ .

5 On the other hand, if  $S_1$  also covers another  $B_i \in$   
 6  $B'$ , a change in  $G'_{S_1}$  can also imply changes in  
 7  $G'_{S'}$ . This, in the worst case, can cause  $mpv^{(k)}(?x_1, B_1, G) \times$   
 8  $\mathcal{S}_{B'}^{(k)}(G)$  changed mappings for the change hap-  
 9 pening in  $G'_{S'}$ , plus  $mpv^{(k)}(?x_1, B', G) \times \mathcal{S}_{B_1}^{(k)}(G)$   
 10 changed mappings caused by the change in  $G_{S_1}$ ,  
 11 which may contain  $\mathcal{S}_{B_1}^{(k)}(G) = \kappa(S)$  changed  
 12 triplets. We also need to consider that the change  
 13 may cause new joins between then new triplets in  
 14 both  $G'_{S_1}$  and  $G'_{S'}$ , for a total of  $\mathcal{S}_{B_1}^{(k)}(G) \times \mathcal{S}_{B'}^{(k)}(G)$   
 15 changed mappings in the joined SPARQL pattern.  
 16 The sum of these three values is what the defini-  
 17 tion of  $\mathcal{S}_B^{(k)}(G)$  uses.  $\square$

18 Now we have all the prerequisite to define the *elastic*  
 19 *sensitivity* of user queries (at fixed distances of a given  
 20 graph):

$$\begin{aligned} \text{ES}_{\text{COUNT}^{?x}(\bar{B})}^{(k)}(G) &= \mathcal{S}_B^{(k)}(G) \\ \text{ES}_{\text{COUNT}^{\text{DISTINCT } ?x}(\bar{B})}^{(k)}(G) &= \mathcal{S}_B^{(k)}(G) \\ \text{ES}_{\text{COUNT}^{?x_0, ?x}(\bar{B})}^{(k)}(G) &= 2\mathcal{S}_B^{(k)}(G) \end{aligned}$$

21 And as in Johnson *et al.* [11], the above lemma read-  
 22 ily leads us to the desired bound for (the three kinds  
 23 of) user queries:

24 **Lemma 5.** For any user query  $Q$ , and any graph  $G$   
 25 compliant with schema  $\mathcal{P}$ :

$$\text{ES}_Q^{(k)}(G) \geq \text{LS}_Q^{(k)}(G).$$

26 *Proof.* By case analysis on the type of user query  $Q$ :

- 27 – For plain counting queries ( $Q = \text{COUNT}^{?x}(\bar{B})$ ):  
 28 the result follows directly from Lemma 4 since  
 29 the result of the counting query is given by the  
 30 application of the sensitivity calculation for the  
 31 *CBGPs* in the query.
- 32 – For plain counting queries ( $Q = \text{COUNT}^{\text{DISTINCT } ?x}(\bar{B})$ ):  
 33 it can be noted that *DISTINCT* reduces the elas-  
 34 tic stability of the elementary BGP,  $B'$ , containing  
 35  $?x$  from  $\kappa(S')$  to 1, where  $S'$  is the star covering  
 36  $B'$ .

- 37 – For counting queries after grouping ( $Q = \text{COUNT}^{?x_0, ?x}(\bar{B})$ ):  
 38 During the grouping, each changed triple affects  
 39 *two* result mappings in the query since one modi-  
 40 fied triples will generate a mapping that may fall  
 41 into one group and losing that mapping another  
 42 group.  $\square$

43 Lemma 5 readily establishes our main result, which  
 44 allows applying Differential Privacy to SPARQL queries  
 45 over RDF graphs:

46 **Theorem 3.** Assume that our universe of (valid) RDF  
 47 graphs is composed by the graphs that comply with *dp*-  
 48 schema  $\mathcal{P}$  and multiplicity bound  $\kappa$ . Let  $Q$  be a user  
 49 query and let

$$\mathcal{U}_Q(G) = \max_{0 \leq k \leq \text{size}(G)_{\mathcal{P}}} e^{-\beta k} \text{ES}_Q^{(k)}(G),$$

50 where  $\epsilon > 0$ ,  $0 < \delta < 1$ ,  $\beta \leq \frac{\epsilon}{2 \ln(2/\delta)}$  and the elas-  
 51 tic sensitivity  $\text{ES}_Q^{(k)}(G)$  of  $Q$  is computed, as previously  
 52 described, from multiplicity bound  $\kappa$  and the frequen-  
 53 cies of most popular values mapped to the variables in  
 54  $Q$  as specified by function  $mpv^{(k)}$ . Then, the random-  
 55 ized algorithm

$$\mathcal{A}(G) = \llbracket Q \rrbracket_G + \text{Lap}\left(\frac{2\mathcal{U}_Q(G)}{\epsilon}\right)$$

56 is an  $(\epsilon, \delta)$ -differentially private version of  $Q$ .

57 The theorem follows immediately from Theorem 2  
 58 and Lemmas 1 and 5.

## 6. Evaluation

59 Having characterized formally how an algorithm  
 60 can be implemented to enforce Differential Privacy on  
 61 SPARQL queries based on privacy schemes, in this  
 62 section, we present an empirical evaluation of how the  
 63 algorithm would behave in real scenarios.

64 *Setup* We conducted our evaluation on a 2018 Mac-  
 65 book Pro with 16 GB of RAM memory having in-  
 66 stalled a Fuseki instance on a 2 AMD Opteron server  
 67 with an SSD drive and 64GB of RAM memory. We  
 68 used Java 1.17 to implement our proof of concept. We  
 69 also used the `SecureRandom` Java class to generate  
 70 the random numbers to calculate the Laplacian prob-  
 71 ability distribution since that class implements a well-

tested random number generator<sup>3</sup>, an essential component for ensuring the correctness of our privacy guarantees algorithm. The code and all the queries used for this evaluation are available in GitHub<sup>4</sup>.

*Data* In the evaluation we used real world data and queries from Wikidata [13]. Wikidata is a collaboratively edited knowledge base hosted by the Wikimedia Foundation. It is a common source of data for Wikimedia projects such as Wikipedia, and it has been made available to the general public under a public domain license. Wikidata stores 86,671,701 items (RDF resources), and 1,084,935,969 statements (triples<sup>5</sup>). We selected a subset of the Wikidata Truthy from 2021-06-23, which has all but direct properties (i.e [http://www.wikidata.org/prop/direct/P\\*](http://www.wikidata.org/prop/direct/P*)) removed [23, 24]. The data is available to download from Google Drive<sup>6</sup>. We also provide the scripts to generate this Wikidata version in our Github repository<sup>7</sup>. We use the following prefixes from Wikidata along this section:

```
wdt: <http://www.wikidata.org/prop/
    direct> # prefix for referring to
    properties
wd: <http://www.wikidata.org/entity> #
    prefix for referring to data
```

### 6.1. Privacy schema

Our Differential Privacy schema is defined based on three pairwise disjoint stars,  $\mathcal{P}_1$ ,  $\mathcal{P}_2$ , and  $\mathcal{P}_3$ , representing data from Humans, Organizations, and Professions, built around Wikidata’s P31 property (the *instance of* property). We extracted URIs from all the instances of the classes Human, Organization and Profession using queries like:

```
SELECT ?center WHERE {
    ?center wdt:P31 wd:Q5
    # wd:Q5 represents the Human class
}
```

<sup>3</sup><https://docs.oracle.com/javase/8/docs/api/java/security/SecureRandom.html>

<sup>4</sup>Repository <https://github.com/cbuil/DPSparql>

<sup>5</sup><https://tools.wmflabs.org/wikidata-todo/stats.php>

<sup>6</sup>[https://drive.google.com/u/0/uc?id=1oDkrHT68\\_v7wfxTxaRg40F7itb7tVEZ](https://drive.google.com/u/0/uc?id=1oDkrHT68_v7wfxTxaRg40F7itb7tVEZ)

<sup>7</sup>[https://github.com/MillenniumDB/benchmark/blob/master/src/database\\_generation/filter\\_direct\\_properties.py](https://github.com/MillenniumDB/benchmark/blob/master/src/database_generation/filter_direct_properties.py)

This gathers the instances of the class Human. Star instances were formed by selecting a subset of properties of the three classes using star queries centered in the URIs (each instance representing either a Human, an Organization or a Profession) to define three sub-graphs covered by three stars,  $\mathcal{P}_1$ ,  $\mathcal{P}_2$ , and  $\mathcal{P}_3$ . In other words, we used the mappings of `?center` from the initial queries as star centers and for each center we retrieved a few of their properties and used the resulting RDF graph as the basis for our queries. Differential Privacy is applied to protect the privacy of the centers. We present the schema with all the properties we used for each star in the following example.

*Evaluation Privacy Schema* The three stars employed to identify the different entities in our evaluation schema are (modulo variable renaming):

$$\begin{aligned} S_1 &= \{(?c_1, P569, ?o_1), (?c_1, P570, ?o_2), (?c_1, P106, ?c_3)\} \\ &\quad (?c_1, P108, ?c_3), (?c_1, P2002, ?o_2), (?c_1, P21, ?o_3)\} \\ &\quad (?c_1, P40, ?c_3)\} \\ S_2 &= \{(?c_3, P1963, ?o_5), (?c_3, P101, ?o_6), (?c_3, P425, ?o_6)\} \\ S_3 &= \{(?c_2, P106, ?o_3), (?c_2, P178, ?o_4), (?c_2, P112, ?o_5)\} \end{aligned}$$

These three stars shouldn’t be used directly as a privacy schema because  $S_1$  and  $S_3$  share a property, P106. Nevertheless, the sets of instances of the property in the sub-graphs induced by  $S_1$  and  $S_3$  are disjoint because instances of `?c1` (human URIs) and `?c2` (organization URIs) are disjoint. Hence, for the purpose of our evaluation, we consider this partition of P106 as two different properties that we refer to as P106 Profession in  $S_1$  and P106 Occupation in  $S_3$ , keep  $S_3$  as it is, and rename them  $\mathcal{P}_1$ ,  $\mathcal{P}_2$ , and  $\mathcal{P}_3$  respectively.<sup>8</sup>

The Wikidata properties that allow joining data from two different stars are P108 Employer from Humans to Organizations, P106 Profession from Humans to Professions, P106 Occupation from Organizations to Professions and P112 Founded by from Organizations to Humans. Table 1 shows statistics about the instances of the stars in our data, including star size (on top of the table).

<sup>8</sup>Note that this observation suggests a more subtle definition of privacy scheme would be useful.



Humans: 9,181,487				
P569	P570	P106	P108	P2002
5,109,648	2,511,719	6,446,811	1,085,617	159,194
P21	P40			
7,223,891	707,747			
Professions: 7,786				
P1963	P101	P425		
97	95	3,018		
Organizations: 72,879				
P106	P178	P112		
50	16	2,789		

Table 1

Table showing key statistics about the data in our Privacy Schema (the largest schema is by far the Humans star)

## 6.2. Queries

We selected 26 queries from the query logs in [23, 24] containing the predicates used in our Privacy Schema. Since the amount of COUNT queries in the query logs is small [25] for each of these queries we added the COUNT keyword and we removed the triples from the query that were not accessing our schema. In addition, these queries were modified to get a diverse set of query results and types.

We consider star queries which are queries covered by a single start from the dp-schema. These queries can only add filters and remove triple patterns from the star. Therefore, a start query is centered around a single join vertex  $?x_0$ , corresponding to the center of the star (Figure 2). We also have linear queries describing a path that must include a join variable that appears in a place that is not the center of any star from the dp-schema (Figure 3), and snowflake queries (Figure 4), a concatenation through a join variable of a star query with other queries of different shapes, as defined in [19]. The queries are listed in Appendix B and they can also be found in the companion Github repository for this article. Table 3, also in Appendix A summarizes their characteristics. We show  $Q_3$  in Figure 4, which queries the Humans star, accessing sex, birth and death dates for each human as well as their professions. We use the `?professions` variable for connecting to the Professions star. From that star we

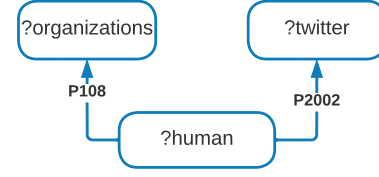


Figure 2. Star-shaped query  $Q_{16}$  accessing the Humans privacy star

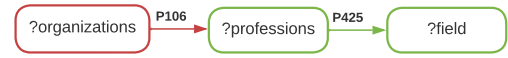


Figure 3. Linear query  $Q_{15}$  accessing the Humans privacy star

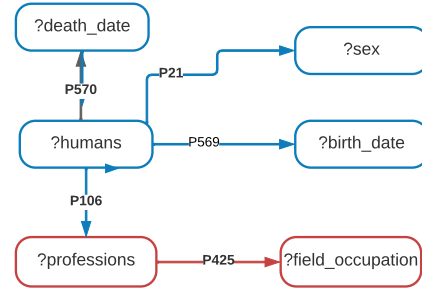


Figure 4. Snowflake query  $Q_3$  accessing the Humans and Organization privacy schemas

retrieve each profession's field of work (property P425), obtaining a snowflake-shaped query.

## 6.3. Results

We report the results of our evaluation in Table 2, showing the actual result to our evaluation queries and the average result to these queries with added noise (calculated applying the method described in Section 5). We report the results using two values for  $\epsilon$ , 0.1 and 1.0. We also report the median error percentage for  $\epsilon = 1.0$  in Figure 5 in which we compare the existence of join operations in the query with the private results obtained. To calculate that error introduced to the actual result of the testing queries we used the following Equation:

$$\text{median} \left( \frac{(ActualCount - NoiseResult_i) * 100}{ActualCount} \right)_{i=1...100}$$

We use  $\delta = n^{-\epsilon \ln n}$  where  $n$  is the size of the dataset (i.e. the sum of all the different RDF resources in the schema that the queries access) and  $\beta = \epsilon / (2 \times \log(2/\delta))$  for the  $\beta$  parameter.

Figure 5 compares the median error obtained using our approach (Y axis) with the original result for each query as well as its query shape, using a value of  $\epsilon = 1.0$  as the privacy parameter. The (blue) squares represent Star queries (accessing a single Star within the privacy schema), have a very low error (and thus a high utility) compared to others.

However, the lower the result from the COUNT, the more error introduced. (Red) Triangles represent path queries and thus queries with join operations. Only query  $Q_9$  is close to the 10% error threshold for considering a query with enough utility. That query is only two triple patterns that retrieve all data from the Professions and Organizations stars. (Grey) Circles represent Snowflake queries, which are queries that join two or more stars in the schema, and also access several properties from each star pattern. Only those queries returning a result greater than 1,000,000 (queries  $Q_3$  and  $Q_6$ ) have a high utility result.

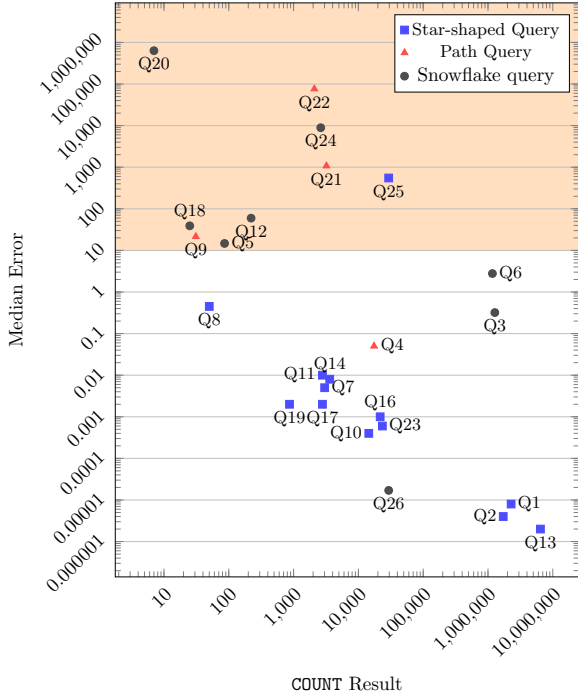


Figure 5. This plot shows that Star-shaped queries (blue squares) have the greatest utility, since they have no joins between stars in the schema. Next, only queries accessing large amounts of data have high utility.

Table 2 shows the results of the execution of the queries in our evaluation set. Columns *Average Private Result Epsilon 0.1 and 1.0* show the result adding noise to the results in column *Actual result* (using values of 0.1 and 1.0 for  $\epsilon$ ). The *Sensitivity* and *Stability* columns show that the larger the Stability polynomial the greater the sensitivity, and thus, the higher the error introduced. Note that in most cases the derived queries produce smaller results than the simpler queries, thus the errors are larger. The only case that this is not the case is for queries  $Q_1$  and  $Q_2$ , but in this case the errors are so small that to really establish that the errors are different much more data should be collected. The queries with a larger amount of noise introduced are those queries that contain at least one join and the result from the COUNT operator is small as in queries  $Q_9$  or  $Q_{20}$ . In general, the more joins in the query and the smaller the result size, the worse results. Joins directly affect the Stability polynomial, more joins imply larger degrees in the polynomials. The effect is exacerbated if the value representing the most popular mapping is large.

Our Differential Privacy method introduces larger amounts of noise to results from queries associated with a polynomial of degree 2 or more. Examples of this are queries  $Q_{21}$  and  $Q_{22}$ , which access data in 3 different stars, and even though the results of the original queries are not small ( $>2,000$ ), the error introduced is very high. Compare that to the single join query  $Q_9$  that produces a small result and although it has a high error, the error is much smaller than the errors of  $Q_{21}$  and  $Q_{22}$  with much larger results. Results from queries accessing a single Privacy Star are good as expected, since without joins they have low query sensitivity and, hence, small errors.

## 7. Related Work

The study of how to guarantee the privacy of individuals contributing personal data to datasets is a long studied problem. In this work we have focused on how to guarantee this privacy in RDF data graphs accessed through SPARQL queries using Differential Privacy. The related work can be roughly classified into those that provide some privacy guarantees to accesses to data stored in (social) graphs and those that guarantee privacy over the results returned by SPARQL queries. We briefly look over these works in this section.

Query Id	Actual Result	Private Result using Epsilon = 0.1	Private Result using Epsilon = 1.0	Sensitivity	Stability
Q1	2,275,177	2,275,176	2,275,176	1.0	x
Q2	1,717,945	1,717,940	1,717,945	1.0	x
Q3	1,274,788	1,189,636	1,270,649	290,415	$(x+290,863) * 1$
Q4	17,440	17,464	17,319	245	$(x+18) * 1$
Q5	86	110.8	203.1	245	$(x+18) * 1$
Q6	1,170,315	3,860,469	1,137,687	400,363	$(x+400,981) * 1$
Q7	3,018	3019	3,017	1.0	x
Q8	50	57	50	1.0	x
Q9	31	1577	37	241.3	$(x+8) * 1$
Q10	14,477	14,472	14,477	1.0	x
Q11	2,789	2,792	2,789	1.0	x
Q12	221	1,144	89	1,162.2	$(x+1,164) * 1$
Q13	6,446,811	6,446,812	6,446,810	1.0	x
Q14	3,615	3,613	3,614	1.0	x
Q15	0	71	8	250.8	$(x+33) * 1$
Q16	21,683	21,682	21,682	1.0	x
Q17	2,789	2,788	2,788	1.0	x
Q18	25	465	34	248.5	$(x+27) * 1$
Q19	865	864	865	1.0	x
Q20	7	7,087,181	44254	1,656,501	$(x+6,189) * (x+8) * 1$
Q21	3,213	1,739,549	31357	1,651,883	$(x+6) * (x+6,189) * 1$
Q22	2,092	377,638,493	1,600,610	408,594,207	$(x+7) * (x+1,694,747) * 1$
Q23	23,450	23,449	23449	1.0	x
Q24	2,626	1,071,418	231,278	628,018	$(x+628,987) * 1$
Q25	29,352	1,593,488	42,317	628,018	$(x+628,987) * 1$
Q26	29,352	29,350	29,351	1.0	x

Table 2

Results of the execution of Wikidata queries using our Differential Privacy method. Those queries with Sensitivity equal to “1.0” are Star queries since the sensitivity of a COUNT query over a single star schema is 1 (a COUNT query over a table), and their elastic stability is “x” as described in Section 5.2. When there are joins between different stars within the schema, the sensitivity increases based on the stability polynomial, calculated using our Theorem 3.

### 7.1. SPARQL state of the art

In the context of the Semantic Web and the Web of Linked Data there have been several approaches

1 to address privacy concerns related queries to RDF  
 2 data. A good survey can be found in [26]. There is  
 3 the basic privacy protection that a SPARQL engine  
 4 must provide from queries that directly return individ-  
 5 uals' data. Similar to the case of relational databases  
 6 where this kind of protection can be done by replacing  
 7 some attribute values with nulls, the work presented  
 8 in [27]. The work by Delanoux et al. [28] presents  
 9 a declarative framework with formal guarantees (us-  
 10 ing SPARQL queries) for specifying privacy policies  
 11 and utility queries for dealing with the application of  
 12 anonymization operations on the graph. One of the  
 13 ways they do that is by using blank nodes as generic  
 14 data that replaces the sensitive data within the dataset.  
 15 That framework checks whether the specified policies  
 16 are compatible with each other, and based on a set of  
 17 basic update queries, automatically builds candidate  
 18 sets of anonymization operations that are guaranteed  
 19 to transform any input dataset into a dataset satisfying  
 20 the required privacy and utility policies.

21 All the effort so far has been directed to provide  
 22 tools to efficiently access Linked Data in the seman-  
 23 tic Web. However, this effort has to be balanced with  
 24 the need to protect the privacy of individuals contrib-  
 25 uting to these datasets. The aim of works such as  $k$ -  
 26 anonymity or  $l$ -diversity is to be able to provide an-  
 27 swers to queries about classes;  $k$ -anonymity is used  
 28 in [29, 30] to answer queries in RDF datasets. Unfortu-  
 29 nately, it is well-known that  $k$ -anonymity does not pro-  
 30 vide formal guarantees for privacy. In contrast, Differ-  
 31 ential Privacy precisely prescribes how to characterize  
 32 the privacy guaranties of data query answering algo-  
 33 rithms. The only work known to us that directly uses  
 34 Differential Privacy over RDF datasets is [26]. The au-  
 35 thors propose a method for applying Differential Pri-  
 36 vacy to SPARQL queries. However, they provide a Dif-  
 37 ferential Privacy realization through local sensitivity  
 38 without the use of a smoothing function, violating thus  
 39 the privacy guarantees described in [8]. In [31] the au-  
 40 thors provide a query language that processes RDF  
 41 streams in a privacy-preserving fashion called Sih-  
 42 IQL. Limiting queries to that language permits servers  
 43 to continuously release privacy-preserving histograms  
 44 (or distributions) from online streams. In [32] Han et  
 45 al. provide Differential Privacy for Knowledge Graph  
 46 embedding methods (such as TransE, TransM [33] and  
 47 the like). Their method generates differentially-private  
 48 embeddings (with formal guarantees) that protect the  
 49 presence of any of the former statements in the embed-  
 50 ding space.  
 51

## 7.2. Privacy in Social Graphs

1 One of the most well-known approaches to provide  
 2 differentially private queries over social graphs is by  
 3 the use of Restricted Sensitivity [9]. The authors of Re-  
 4 stricted Sensitivity define adjacency of graphs based  
 5 on two notions: differences on edges and differences  
 6 on vertices. The distance between two graphs,  $G_1$  and  
 7  $G_2$ , is given by the smallest number of changes (either  
 8 on edges or vertices) needed to transform  $G_1$  and  $G_2$   
 9 into the same graph, giving rise to two definitions of  
 10 restricted sensitivity. The authors also provide efficient  
 11 algorithms to calculate these sensitivities for a class  
 12 of social graph queries that involve only one join. The  
 13 authors of [34] extended the previous notion of edge-  
 14 based restricted sensitivity by using weighted datasets.  
 15 Briefly, the aim is to increase the utility of the answer  
 16 by calculating noise with weights added to each edge  
 17 that contributes to the solution. Our notion of Differ-  
 18 ential Privacy schema of sensitivity is a vertex-based  
 19 sensitivity by observing that each  $g_i \in G$  can be in-  
 20 terpreted as a single node. The class of queries for  
 21 which efficient implementations of Restricted Sensi-  
 22 tivity exist corresponds to the execution of joins be-  
 23 tween Start patterns. Our proposed Elastic Sensitiv-  
 24 ity can be then interpreted as a generalization of Re-  
 25 stricted Sensitivity. In Restricted Sensitivity, the poly-  
 26 nomial associated to a query is always of degree 1 and  
 27 the value of  $x$  is bounded by a constant (which is pro-  
 28 vided by the system administrator). Elastic Sensitiv-  
 29 ity is based on a variable selectivity (the values of  $x$   
 30 are obtained directly from the dataset), and general-  
 31 izes to multiple joins. Other works such as [35] pro-  
 32 posed a method to use Differential Privacy for sub-  
 33 graph counting queries with unrestricted joins (through  
 34 node Differential Privacy), however, answering this  
 35 type of queries is computationally difficult (NP-hard).  
 36 The result is then more of theoretical interest and lim-  
 37 ited for general application.  
 38  
 39  
 40  
 41

## 8. Conclusions

1 In this paper we have introduced a framework to de-  
 2 velop Differential Privacy tools for RDF Data reposi-  
 3 tories, and we have used the framework to develop an  
 4  $(\epsilon, \delta)$ -Differential Privacy SPARQL query engine for  
 5 COUNT queries. A crucial component of our frame-  
 6 work is the concept of Differential Privacy Schema.  
 7 Without it, we would have not been able to develop  
 8 a Differential Privacy preserving algorithm to publish  
 9  
 10  
 11

1 data of acceptable quality. The concept is independent  
 2 of the sensitivity approximation used and we hope that  
 3 others can build on the concept to get better query  
 4 answering algorithms.

5 We have implemented our algorithm and tested it  
 6 using the Wikidata RDF database, queries from its log  
 7 files and other example queries found at the Wikidata  
 8 endpoint. The simulations show the approach to be  
 9 effective for queries over large repositories, such as  
 10 Wikidata, and in many cases for queries within the  
 11 10 of thousands answers to aggregate. However, even  
 12 though Elastic Sensitivity has been designed to bind  
 13 the stability of joins, the sensitivity of a query with  
 14 joins can still be very high. In the case of SQL queries  
 15 in relational databases, in order to keep the noise  
 16 under a single percentage digit, the databases should have  
 17 over 1M tuples and  $\epsilon = 1$ . This is also the case for  
 18 SPARQL queries. The evaluation shows though that  
 19 we can safely evaluate Star queries.

20 We can still apply several optimizations to our  
 21 framework. For example, public graphs can be treated  
 22 as public tables. If they participate in joins, we can  
 23 directly use their most popular result mappings dur-  
 24 ing calculation of the query sensitivity. We can also  
 25 consider the approaches described in [11] for SQL to  
 26 add aggregation functions like sum and averages to our  
 27 framework.

28 There are many pending issues to address. From the  
 29 more practical point of view, more operations need to  
 30 be implemented. There are also issues about the im-  
 31 pact that such algorithms will have on SPARQL query  
 32 engines. From the more formal side, it is still impor-  
 33 tant to keep searching for better approximations of lo-  
 34 cal and global sensitivities as well as alternative defi-  
 35 nitions that are less onerous than Differential Privacy.  
 36 One possibility is to find a way to apply Restricted  
 37 Sensitivity to more types of queries by adding more se-  
 38 mantic information to the Differential Privacy Schema.  
 39 It might also be possible to find a more accurate ap-  
 40 proximation of the elastic sensitivity for *DISTINCT*  
 41 queries to make it independent or partially indepen-  
 42 dent of predicate multiplicity. We should point out that  
 43 most queries that will require privacy protection will  
 44 be *DISTINCT* (e.g. how many human exists with  
 45 properties A and B that work in company C?).

## 46 Acknowledgements

47  
 48 We thank the reviewers for their thorough work re-  
 49 vising this paper, which improved the overall qual-  
 50 ity of the paper. C. Buil-Aranda and J. Lobo were  
 51 partially funded by Fondecyt Iniciación 11170714;  
 C Buil-Aranda and Federico Olmedo were supported  
 by ANID – Millennium Science Initiative Program –  
 Code ICN17\_002.

## References

- [1] L. Menand, Why Do We Care So Much About Privacy?, *The New Yorker* **XCIV**(17) (2018), 24–29.
- [2] N. Li, W. Qardaji, D. Su, Y. Wu and W. Yang, Membership privacy: a unifying framework for privacy definitions, in: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM, 2013, pp. 889–900.
- [3] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov and M. Naor, Our data, ourselves: Privacy via distributed noise generation, in: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2006, pp. 486–503.
- [4] L. Sweeney, k-anonymity: A model for protecting privacy, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **10**(05) (2002), 557–570.
- [5] A. Machanavajhala, J. Gehrke, D. Kifer and M. Venkatasubramanian, l-diversity: Privacy beyond k-anonymity, in: *22nd International Conference on Data Engineering (ICDE'06)*, IEEE, 2006, pp. 24–24.
- [6] N. Li, T. Li and S. Venkatasubramanian, t-closeness: Privacy beyond k-anonymity and l-diversity, in: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, IEEE, 2007, pp. 106–115.
- [7] C. Dwork, Differential privacy: A survey of results, in: *International Conference on Theory and Applications of Models of Computation*, Springer, 2008, pp. 1–19.
- [8] C. Dwork, A. Roth et al., The algorithmic foundations of differential privacy, *Foundations and Trends® in Theoretical Computer Science* **9**(3–4) (2014), 211–407.
- [9] J. Blocki, A. Blum, A. Datta and O. Sheffet, Differentially Private Data Analysis of Social Networks via Restricted Sensitivity, in: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, ACM, New York, NY, USA, 2013, pp. 87–96. ISBN 978-1-4503-1859-4. doi:10.1145/2422436.2422449. http://doi.acm.org/10.1145/2422436.2422449.
- [10] F.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*, ACM, 2009, pp. 19–30.
- [11] N. Johnson, J.P. Near and D. Song, Towards practical differential privacy for SQL queries, *Proceedings of the VLDB Endowment* **11**(5) (2018), 526–539.
- [12] R. Cyganiak, D. Wood and M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, 2014.
- [13] D. Vrandečić and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Communications of the ACM* **57**(10) (2014), 78–85.
- [14] R. Motwani and P. Raghavan, Randomized algorithms, *ACM Computing Surveys (CSUR)* **28**(1) (1996), 33–37.

- [15] D. Kifer and A. Machanavajjhala, No Free Lunch in Data Privacy, in: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, ACM, New York, NY, USA, 2011, pp. 193–204. ISBN 978-1-4503-0661-4. doi:10.1145/1989323.1989345. <http://doi.acm.org/10.1145/1989323.1989345>.
- [16] C. Dwork, Differential Privacy, in: *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, 33rd international colloquium on automata, languages and programming, part ii (icalp 2006) edn, Lecture Notes in Computer Science, Vol. 4052, Springer Verlag, 2006, pp. 1–12. ISBN 3-540-35907-9. <https://www.microsoft.com/en-us/research/publication/differential-privacy/>.
- [17] K. Nissim, S. Raskhodnikova and A. Smith, Smooth Sensitivity and Sampling in Private Data Analysis, in: *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, ACM, New York, NY, USA, 2007, pp. 75–84. ISBN 978-1-59593-631-8. doi:10.1145/1250790.1250803. <http://doi.acm.org/10.1145/1250790.1250803>.
- [18] K. Nissim, S. Raskhodnikova and A. Smith, Smooth Sensitivity and Sampling in Private Data Analysis, 2011, Draft full version v1.0. <http://www.cse.psu.edu/~ads22/pubs/NRS07/NRS07-full-draft-v1.pdf>.
- [19] G. Aluç, O. Hartig, M.T. Özsu and K. Daudjee, Diversified stress testing of RDF data management systems, in: *International Semantic Web Conference*, Springer, 2014, pp. 197–212.
- [20] M. Arapinis, D. Figueira and M. Gaboardi, Sensitivity of Counting Queries, in: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani and D. Sangiorgi, eds, LIPIcs, Vol. 55, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 120–112013.
- [21] S. Harris and A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation <http://www.w3.org/TR/2010/WD-sparql11-query-20101014/>, 2012.
- [22] J. Pérez, M. Arenas and C. Gutierrez, Semantics and complexity of SPARQL, *TODS* **34(3)** (2009).
- [23] A. Hogan, C. Riveros, C. Rojas and A. Soto, A Worst-Case Optimal Join Algorithm for SPARQL, in: *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I*, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I.F. Cruz, A. Hogan, J. Song, M. LeFrançois and F. Gandon, eds, Lecture Notes in Computer Science, Vol. 11778, Springer, 2019, pp. 258–275. doi:10.1007/978-3-030-30793-6\_15. [https://doi.org/10.1007/978-3-030-30793-6\\_15](https://doi.org/10.1007/978-3-030-30793-6_15).
- [24] D. Vrgoc, C. Rojas, R. Angles, M. Arenas, D. Arroyuelo, C.B. Aranda, A. Hogan, G. Navarro, C. Riveros and J. Romero, MillenniumDB: A Persistent, Open-Source, Graph Database, *CoRR* **abs/2111.01540** (2021). <https://arxiv.org/abs/2111.01540>.
- [25] A. Bonifati, W. Martens and T. Timm, Navigating the maze of wikidata query logs, in: *The World Wide Web Conference*, 2019, pp. 127–138.
- [26] R.R.C. Silva, B.C. Leal, F.T. Brito, V.M. Vidal and J.C. Machado, A differentially private approach for querying RDF data of social networks, in: *Proceedings of the 21st International Database Engineering & Applications Symposium*, ACM, 2017, pp. 74–81.
- [27] B.C. Grau and E.V. Kostylev, Logical foundations of privacy-preserving publishing of Linked Data, in: *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [28] R. Delanaux, A. Bonifati, M.-C. Rousset and R. Thion, Query-based Linked Data Anonymization, in: *International Semantic Web Conference*, Springer, 2018, pp. 530–546.
- [29] F. Radulovic, R. García Castro and A. Gómez-Pérez, Towards the anonymisation of RDF data (2015).
- [30] B. Heitmann, F. Hermsen and S. Decker, k-RDF-Neighbourhood Anonymity: Combining Structural and Attribute-based Anonymisation for Linked Data., in: *PrivOn ISWC*, 2017.
- [31] D. Dell’Aglío and A. Bernstein, Differentially Private Stream Processing for the Semantic Web, in: *Proceedings of The Web Conference 2020*, WWW '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1977–1987. ISBN 9781450370233. doi:10.1145/3366423.3380265. <https://doi.org/10.1145/3366423.3380265>.
- [32] X. Han, D. Dell’Aglío, T. Grubenmann, R. Cheng and A. Bernstein, A framework for differentially-private knowledge graph embeddings, *J. Web Semant.* **72** (2022), 100696. doi:10.1016/j.websem.2021.100696. <https://doi.org/10.1016/j.websem.2021.100696>.
- [33] Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in: *Twenty-ninth AAAI conference on artificial intelligence*, 2015.
- [34] D. Proserpio, S. Goldberg and F. McSherry, Calibrating data to sensitivity in private data analysis: a platform for differentially-private analysis of weighted datasets, *Proceedings of the VLDB Endowment* **7(8)** (2014), 637–648.
- [35] S. Chen and S. Zhou, Recursive mechanism: towards node differential privacy and unrestricted joins, in: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ACM, 2013, pp. 653–664.

## Appendix A. Query Characteristics

In this Section we present the characteristics of the queries we used in Section 6. The table presents the query ID (which refer to queries Section B in this Appendix), the stars within the Privacy Schema we defined in 6, the query shape, the amount of tripe patterns in the queries as well as the number of variables, the join variables when applicable, including the amount of mappings for each join variable, and data about the Privacy Schema stars in the query.



Query ID	Schemes	Shape	Triple Patterns	Join Variables	Star Info
Q1	Humans	Star	3 tp, 4 vars	No join variables	1,717,255 Humans with Professions, 10,337 Professions at Humans star, 2,991 distinct Professions at Professions star
Q2	Humans	Star	4 tp, 5 vars	No join variables	
Q3	Humans and profs	Snowflake	5 tp, 6 vars	COUNT(?professions) = 3018 (from profs star)	
Q4	Humans	Path	1 tp, 2 vars	COUNT(?humans) = 3615 (from Professions star) COUNT(?humans) = 1,648,629 (from Humans Star)	1,648,629 distinct humans educated at (P69), and 3,615 Humans have Organizations ( $Q_2b$ ), 2,789 professions ( $Q_2c$ 6 developer field)
Q5	Humans and Orgs	Snowflake	2 tp, 3 vars	COUNT(?humans) = 3,615 from Professions Star	
Q6	Humans and Profs	Snowflake	3 tp, 4 vars	COUNT(?professions) = 7,764 from Humans, COUNT(?professions) = 3,018 from professions	
Q7	Profs	Star	1 triple, 2 vars	No join variables	3018 distinct profs 50 distinct organizations with professions, 3018 distinct professions
Q8	Orgs	Star	1 triple, 2 vars	No join variables	
Q9	Orgs and Profs	Path	2 tp 4 vars	COUNT(?professions) = 54 from organizations, ?professions = 3,356 Professions	
Q10	Humans	Star	1 triple 2 vars	No join variables	
Q11	Orgs	Star	1 triple 2 vars	No join variables	14,382 distinct Humans, 2,789 distinct Organizations

Q12	Humans and Orgs	Snowflake	3 tp 6 vars	COUNT(?organizations) = 33,423 from humans star, COUNT(?organizations) = 3,814 form Organizations star	
Q13	Humans	Star	1 triple 2 vars	No join variables	6,446,811 distinct Humans, 2,789 distinct Organizations
Q14	Orgs	Star	1 triple 2 vars	No join variables	
Q15	Humans and Orgs	Path	2 tp 4 vars	COUNT(?organizations) = 8,501,245 at Humans star, COUNT(?organizations) = 3,814 at Organizations star	
Q16	Humans	Star	2 tp 4 vars	No join variables	21,651 Humans with Twitter accounts and a employer, 13,814 distinct Organizations
Q17	Orgs	Star	1 triple 2 vars	No join variables	
Q18	Humans and Orgs	Snowflake	3 tp 4 vars	COUNT(?organizations) = 35,419 from Humans Star, COUNT(?humans=3814) from Organizations star	
Q19	Organizations	Star	2 tp	No join variables	5 distinct organizations
Q20	Profs, Humans and Orgs	Snowflake	3 tp 6 vars	No join variables	3,018 distinct profs, 90,341 Organizations at Humans star, 31 distinct Organizations at Organizations star
Q21	Humans, Occupations, Professions	Path	3 tp, 4 vars	COUNT(?occupationsStar) = 90,341 (humans side), COUNT(?opccupationsStar) = 15,524 (occupationsStar side), 3,018 professions (Professions side)	6,446,860 occupationsStar, 3,018 Professions, 1,448,232 Humans with occupations at Humans Star

Q22	Organizations, Occupations, Professions	Path	3 tp, 4 vars	COUNT(?occupationsStar) = 3,615 (organizations side), 15524 (occupationsStar side), 3018 professions (professions side)	6,446,860 occupationsStar, 2,789 Organizations Star, 3,018 Professions Star
Q23	Humans and profs	Snowflake	3 tp, 4 vars		23,450 Humans with Athlete profession, 4 Professions with "Sport" field.
Q24	Humans	Star	2 tp, 3 vars	?COUNT(?professions) = 23,451 form Humans Star, ?COUNT(?professions) = 4 form Professions Star,	
Q25	Profs, Humans	Snowflake	5 tp, 3 different vars	?COUNT(?professions) = 15,359	1 profession, 4,969,877 humans
Q26	Humans	Star	2 tp, 2 vars	?COUNT(?humans) = 29,352 humans (distinct)	

Table 3: table showing the main characteristics of each query to the privacy schema

## Appendix B. Queries

In this Section we present the queries we used in Section 6. There are 11 base queries with several variations, totaling 26 SPARQL COUNT queries.

Query  $Q_1$ :

```
SELECT (COUNT(DISTINCT ?humans) as ?
count) WHERE {
?humans wdt:P21 ?v1 .
?humans wdt:P569 ?v4 .
?humans wdt:P570 ?v2 .
}
```

Query  $Q_2$ :

```
SELECT (COUNT(DISTINCT ?humans) as ?
count) where {
?humans wdt:P21 ?v1 .
?humans wdt:P569 ?v4 .
?humans wdt:P570 ?v2 .
?humans wdt:P106 ?occupation .
}
```

Query  $Q_3$ :

```
SELECT (COUNT(DISTINCT ?humans) as ?
count) where {
?humans wdt:P31 wd:Q5 .
?humans wdt:P21 ?v1 .
?humans wdt:P569 ?v4 .
?humans wdt:P570 ?v2 .
?humans wdt:P106 ?professions .
?professions wdt:P31 wd:Q28640 .
?professions wdt:P425 ?
field_occupation
}
```

Query  $Q_4$ :

```
SELECT (COUNT(DISTINCT ?humans) as ?
count) WHERE {
?humans wdt:P69 ?v2 .
?organizations wdt:P112 ?humans .
}
```

Query  $Q_5$ :

```
SELECT (COUNT(DISTINCT ?humans) as ?
count) WHERE {
?humans wdt:P69 ?v2 .
?organizations wdt:P178 ?developer .
?organizations wdt:P112 ?humans
}
```

Query  $Q_6$ :

```
SELECT (COUNT(DISTINCT ?humans) as ?
count) WHERE {
?humans wdt:P69 ?v2 .
?humans wdt:P106 ?professions .
?professions wdt:P425 ?field
}
```

Query  $Q_7$ :

```
SELECT (COUNT(DISTINCT ?professions) as
?count) WHERE {
?professions wdt:P425 ?var6 .
}
```

Query  $Q_8$ :

```
SELECT (COUNT(DISTINCT ?organizations)
as ?count) WHERE {
?organizations wdt:P106 ?professions
.
}
```

Query  $Q_9$ :

```
SELECT (COUNT(DISTINCT ?organizations)
as ?count) WHERE {
?professions wdt:P425 ?var6 .
?organizations wdt:P106 ?professions
.
}
```

Query  $Q_{10}$ :

```
Select (COUNT (DISTINCT ?humans) as ?
count) where {
?humans wdt:P40 ?child .
?humans wdt:P108 ?organizations .
}
```

Query  $Q_{11}$ :

```
Select (COUNT (DISTINCT ?organizations)
as ?count) where {
?organizations wdt:P112 ?founded_by
}
```

Query  $Q_{12}$ :

```
SELECT (COUNT (DISTINCT ?humans) as ?
count) WHERE {
?humans wdt:P40 ?child . #with at
least one P40 (child) statement
?humans wdt:P108 ?organizations .
?organizations wdt:P112 ?founded_by
}
```

```

1      Query Q13:                                ?organizations wdt:P106 ?professions      1
2
3      SELECT (COUNT(DISTINCT ?humans) as ?      }
4      count) where {                               .
5      ?humans wdt:P106 ?organizations .          }
6
7      Query Q14:
8
9      SELECT (COUNT(DISTINCT ?humans) as ?
10     count) where {
11     ?organizations wdt:P112 ?humans
12     }
13
14     Query Q15:
15
16     SELECT (COUNT(DISTINCT ?humans) as ?
17     count) where {
18     ?humans wdt:P106 ?organizations .
19     ?organizations wdt:P112 ?founded_by
20     }
21
22     Query Q16:
23
24     SELECT (COUNT (DISTINCT ?humans) as ?
25     count) WHERE {
26     ?humans wdt:P2002 ?twitter . #with
27     at least one P40 (child) statement
28     ?humans wdt:P108 ?organizations .
29     }
30
31     Query Q17:
32
33     SELECT (COUNT (DISTINCT ?organizations)
34     as ?count) WHERE {
35     ?organizations wdt:P31 wd:Q43229 .
36     ?organizations wdt:P112 ?founded_by
37     }
38
39     Query Q18:
40
41     SELECT (COUNT (DISTINCT ?humans) as ?
42     count) WHERE {
43     ?humans wdt:P2002 ?twitter . #with
44     at least one P40 (child) statement
45     ?humans wdt:P108 ?organizations .
46     ?organizations wdt:P112 ?founded_by
47     }
48
49     Query Q19:
50
51     SELECT (COUNT (DISTINCT ?organizations)
52     as ?count) WHERE {
53     # ?organizations wdt:P31 wd:Q43229 .
54     ?organizations wdt:P112 ?founded_by .
55
56     Query Q20:
57
58     SELECT (COUNT(DISTINCT ?humans) as ?
59     count) WHERE {
60     ?professions wdt:P425 ?var6 .
61     ?organizations wdt:P106 ?professions
62     .
63     ?organizations wdt:P31 wd:Q43229 .
64     ?humans wdt:P108 ?organizations .
65     ?humans wdt:P31 wd:Q5 .
66     }
67
68     Query Q21:
69
70     SELECT (COUNT (DISTINCT ?humans) as ?
71     count) WHERE {
72     ?humans wdt:P108 ?occupationsStar .
73     ?occupationsStar wdt:P106 ?
74     professions .
75     ?professions wdt:P425 ?var6 .
76     }
77
78     Query Q22:
79
80     SELECT (COUNT (DISTINCT ?organizations)
81     as ?count) WHERE {
82     ?organizations wdt:P112 ?
83     occupationsStar . # humans link
84     ?occupationsStar wdt:P106 ?
85     professions .
86     ?professions wdt:P425 ?var6 .
87     }
88
89     Query Q23:
90
91     SELECT (COUNT(DISTINCT ?humans) as ?cnt)
92     WHERE {
93     ?humans wdt:P106 wd:Q2066131 .
94     ?humans wdt:P21 ?v2 .
95     }
96
97     Query Q24:
98
99     SELECT (COUNT(DISTINCT ?humans) as ?
100    count) WHERE {
101    ?humans wdt:P106 ?professions .
102    ?humans wdt:P21 ?v2 .
103    ?professions wdt:P425 wd:Q349 .
104    }
105
106    Query Q25:

```

1	<b>SELECT (COUNT(DISTINCT ?humans) as ?</b>	<b>SELECT (COUNT(DISTINCT ?humans) as ?</b>	1
2	<b>count) WHERE {</b>	<b>count) WHERE {</b>	2
3	?humans wdt:P106 wd:Q901 .	?humans wdt:P106 ?professions .	3
4	?humans wdt:P21 ?v2 .	?humans wdt:P21 ?v2 .	4
5	}	?professions wdt:P425 wd:Q336 .	5
6		}	6
7	Query $Q_{26}$ :		7
8			8
9			9
10			10
11			11
12			12
13			13
14			14
15			15
16			16
17			17
18			18
19			19
20			20
21			21
22			22
23			23
24			24
25			25
26			26
27			27
28			28
29			29
30			30
31			31
32			32
33			33
34			34
35			35
36			36
37			37
38			38
39			39
40			40
41			41
42			42
43			43
44			44
45			45
46			46
47			47
48			48
49			49
50			50
51			51