# Answering and Explaining SQL Queries Privately

by

Yuchao Tao

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ashwin Machanavajjhala, Supervisor

_____
Sudeepa Roy

_____
Jerome Reiter

_____
Bolin Ding

Dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2022

# ABSTRACT

# Answering and Explaining SQL Queries Privately

by

Yuchao Tao

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Ashwin Machanavajjhala, Supervisor

_____
Sudeepa Roy

_____
Jerome Reiter

_____
Bolin Ding

An abstract of a dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
Duke University

2022

# Abstract

Data privacy has been receiving an increasing amount of attention in recent years. While large-scale personal information is collected for scientific research and commercial activities, a privacy breach is not acceptable as a trade-off. In the last decade, differential privacy has become a gold standard to protect data privacy and has been applied in many organizations. Past work focused on developing a differentially private SQL query answering system as an infrastructure for wider applications. However, answering counting queries with joins under differential privacy appears as a challenge. The join operator allows any user to have an unbounded impact on the query result, which impedes hiding the existence of a single user in the worst case. On the other hand, the introduction of differential privacy to the query answering also prevents the users from understanding the query results correctly, since she needs to distinguish the effect of differential privacy from the contribution of data.

In this thesis, we study two problems about answering and explaining SQL queries privately. First, we present efficient algorithms to compute local sensitivities of counting queries with joins, which is an important premise for answering these queries under differential privacy. We track the sensitivities contributed by each tuple, based on which we propose a truncation mechanism that answers counting queries with joins privately with high utility. Second, we propose a formal framework DPXPlain, a three-phase framework that allows users to get explanations for group-by COUNT/SUM/AVG query results while preserving DP. We utilize confidence intervals to help users understand the uncertainty in the query results introduced by differential privacy, and further provide top-k explanations under differential privacy to explain the contribution of data to the results.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

*Data collection is everywhere.* In the modern society, our data is collected from anywhere at anytime. It includes daily search logs from our web browsers, posts from our social media, transactions from our credit cards, commute routes from our mobile GPS, medical history from our hospital visits, and so on. The data is collected by a wide spectrum of organizations, and is processed for a variety of uses. Although it boosts a significant increase to the economy and technology development from the collection and processing of personal data, it also raises up a new concern from the public: privacy.

*The issue of privacy.* Privacy is not a new concept in the recent decades, but it receives more attention nowadays due to the vast collection of personal data. With the accurate and sufficient data about a single person, it allows the people who controls the data to do two things: to know who you are and to know what you are doing. For example, with the collection of facial data, a well trained model can identify the person from the camera [121], and further extract her behaviors from the videos and link them to that person. Another example is targeting advertising [92]. The advertising companies collect the user portraits, which includes their demographic features, biological profiles, and personal habits, and then advertise to the users by different advertisements that are best tailored or believed to be best matched, to increase the click-through rate and grow business. Accurate targeting advertising not only makes the users worry about that their personal information is overly collected, but also has a potential risk of leaking the user data to the uncontrolled third party

[104]. More and more examples indicates that privacy is an urgent and important issue that needs to be immediately solved in the big data era.

*Past attempts.* There are many attempts to solve the issue of privacy in the past decades. The most well-known technique before 2006 is k-anonymity [151]. The fundamental concept before k-anonymity is obfuscation through similarity. For example, if there are four digits in the database are used to identify a single person, k-anonymity ensures each identifier stored in the database have at least $k-1$ other identifiers that are the same. Intuitively, a higher value of $k$ protects better privacy, but studies show that it is still vulnerable to attacks [120] and it cannot provide privacy guarantee without the assumption to the attackers.

Another attempt of privacy protection is through regulation. GDPR, or General Data Protection Regulation, is proposed by Europe in 2018 to regulate the activities of the companies about the collection and usage of personal data [162]. For example, it requires the organization to provide "the right to be forgotten", which allows users to delete their data and remove all the related evidences. Users are also given more options to control their data. For example, mobile applications need to ask permissions to have access to the specific functionalities of mobile phones, and websites need to ask permissions of cookies to track the browser activities.

These attempts either protects the privacy from the input (e.g., user access control) or from the output (e.g., k-anonymity), but none of them provides a provable privacy guarantee.

*Differential Privacy is a gold standard.* Differential privacy (DP) [67, 69, 70, 30] is the gold standard for protecting sensitive databases. It has been widely adopted by organizations like the U.S. Census Bureau [6, 65, 146, 103] and companies like Google [75, 164], Microsoft [55] and Apple [154]. The basic concept behind differential privacy is that the released statistics does not change significantly by adding or removing

a single person from the data. Given a query, it is replaced by a mechanism $\mathcal{M}$ and the output is no longer determined. It becomes random, and its distribution depends on the underlying data; i.e., each data $x$ defines an output distribution $\mathcal{M}(x)$. Differential privacy ensures that for any two data $x$ and $x'$ that only differs by a single row, the output distribution is similar. In particular, for any number $o$, we can find a constant $\epsilon$ such that $Pr[\mathcal{M}(x) = o] \leqslant e^{\epsilon} Pr[\mathcal{M}(x') = o]$. Since this constant $\epsilon$ only depends on the mechanism $\mathcal{M}$, $\epsilon$ is also called *privacy loss*, which becomes an odometer to measure the degree of privacy protection. Recent works have made significant advances in the usability of differential privacy, allowing for complex query support [106, 164, 107, 94, 122, 155, 60], and employing DP in different settings [60, 155, 77, 137, 81, 170]. These works assist in bridging the gaps between the functionality of non-DP databases and databases that employ DP.

## 1.2 Problem Settings

*The importance of a differential private database.* Differential privacy formalize the privacy guarantee, but also increases the challenge of its application to the real world. Unlike other data privacy techniques that perturb the data itself and allows unlimited downstream tasks, differential privacy keeps the data unchanged and requires the process of querying the data to be perturbed. This is a fundamental difference between differential privacy and other privacy techniques, and is also the reason that mechanism design becomes challenging for differential privacy. Given different queries, each needs a specific differential private mechanism that is tailored to the query so that it both satisfies differential privacy and also provides similar utility as the original query. However, the number of queries is infinite and it is impossible to design the mechanism one by one. Therefore, it is important to regulate the model of data and the way of processing it.

Relational database and SQL queries is a natural choice as a standard framework for data collection and processing [150]. Past work has studied specific mechanisms for a subset of SQL queries, e.g. histogram and range count [88]. After Flex [94], a complex solution of answering SQL queries differentially privately that is applied in Uber, we have also developed PrivateSQL [106] with a smarter architecture. Such a system allows people who are familiar with relational database and SQL queries quickly to deploy differential privacy into the existing products with a low learning curve.

*The challenge of join-count queries.* One of the basic mechanism design logic is to find the global sensitivity of the query, and apply Laplace noise scaled to the global sensitivity to achieve differential privacy. Basic SQL queries such as a counting query that counts the number of all the tuples in a table has global sensitivity as 1, since adding or removing a single tuples can only change the count by at most 1. However, when combining joins into a counting query, the global sensitivity can become unbound. This is because in such a case, adding or removing a single tuple can affect multiple tuples due to the join operator. For example, consider the following SQL query:

SELECT COUNT(*) FROM User NATURAL JOIN Transaction;

assuming the schema is given as User(<u>id</u>, sex), Transaction(id, item), where <u>id</u> indicates that is a primary key. Suppose a user has bought 1000 items, removing this user from the database will reduce the result of this query by 1000. Since this number can be arbitrarily large in theory, the global sensitivity of such a query is infinite.

Although for join-count queries, the global sensitivity is a negative result, research has shed light on the local sensitivity. Unlike global sensitivity that seeks for the worst case from all the possible databases, local sensitivity only focus on the local change of the current database. For example, continue with the previous user-transaction

join counting case, if the maximum size of items bought by a single user is 1000 in the current database, the local sensitivity is 1000. However, calibrating noise to the local sensitivity does not follow differential privacy, since the sensitivity itself leaks the information about the database. Several works have proposed solutions to adapt local sensitivity to satisfy differential privacy [133, 173], therefore computing local sensitivity of join-count queries connects the differential privacy to these queries.

*Explain differential privacy.* The introduction of differential privacy to the SQL query results also raises a new challenge: how to interpret the result? Due to the noise injected to the query results, for people who have no background about differential privacy, she is not aware that the result is perturbed and may raise doubts when several conflicting results are shown. For example, consider the following three SQL queries

$q_1$ : SELECT COUNT(*) FROM User;

$q_2$ : SELECT COUNT(*) FROM User WHERE sex = "Male";

$q_3$ : SELECT COUNT(*) FROM User WHERE sex = "Female";

A typical differential private mechanism for releasing these queries is to add a Laplace noise to the original query results and round to integers. We assume the results are given as follows: $\hat{q}_1 = 101$, $\hat{q}_2 = 50$, $\hat{q}_3 = 52$, where the truth are $q_1 = 100$, $q_2 = 51$, $q_3 = 49$ hidden from the user. First, from the user perceptive, there is a clear conflict about the query results: $\hat{q}_2 + \hat{q}_3 \neq \hat{q}_1$, where the equality should instead hold according to the semantics of the queries. In this case, user expects an explanation about how the incorporation of differential privacy results in such a conflict. Second, user may expect that $\hat{q}_2 > \hat{q}_3$ by some prior knowledge, which is not observed from the results. In this case, user expects an explanation about whether the differential privacy or the data results in such an absence of expectation.

In conclusion, the problem of explaining differential privacy is to track how the

adoption of differential privacy affects the phenomenon described by the user after observing the differential private results. The difficulty of explanation depends on both the complexity of the phenomenon and of the mechanism.

*Explain differentially privately.* Although SQL query explanation has been explored by the database community in multiple previous works [51, 91, 90, 110, 34, 158, 52, 128, 129, 117, 140, 130, 115, 166, 144, 143], there is yet no work applying differential privacy to the SQL query explanation. Since this problem is to mine the reason from the database to explain the query results under differential privacy, it is fundamentally different from explaining the differential privacy. In other words, one is to *explain differential privacy*, and the other is to *explain differentially privately*. To stop the loop of chicken-and-egg problem about differential privacy and explanation, it is important to find self-explained explanations under differential privacy.

## 1.3 Contributions

In this section, we summarize the major contributions in this work.

*Computing local sensitivity of join-count queries.* Local sensitivity of a query Q given a database instance D, i.e. how much the output Q(D) changes when a tuple is added to D or deleted from D, has many applications including query analysis, outlier detection, and in differential privacy. However, it is NP-hard to find local sensitivity of a conjunctive query in terms of the size of the query, even for the class of acyclic queries. Although the complexity is polynomial when the query size is fixed, the naive algorithms are not efficient for large databases and queries involving multiple joins. In this work, we present a novel approach to compute local sensitivity of counting queries involving join operations by tracking and summarizing tuple sensitivities – the maximum change a tuple can cause in the query result when it is added or removed. We give algorithms for the sensitivity problem for full acyclic join queries using join

trees, that run in polynomial time in both the size of the database and query for an interesting sub-class of queries, which we call 'doubly acyclic queries' that include path queries, and in polynomial time in combined complexity when the maximum degree in the join tree is bounded. Our algorithms can be extended to certain non-acyclic queries using generalized hypertree decompositions. We evaluate our approach experimentally, and show applications of our algorithms to obtain better results for differential privacy by orders of magnitude.

*A formal framework to explain SQL queries under differential privacy.* Differential privacy (DP) has become a gold standard to answer database queries while preserving user privacy; however, it also presents challenges for users to understand the query results: Is the unexpected answer due to the data itself, or is it due to the extra DP noise? For the first case, explanations mining from the sensitive data is required, which poses unique challenges if privacy is protected. In this paper, we present DPXPlain, a three-phase framework that allows users to get explanations for group-by COUNT/SUM/AVG query results while preserving DP. In Phase-1, the system evaluates the query in a DP manner and returns the results. In Phase-2, the system allows users to ask a comparative question about the relation between groups in the result, and then assesses, with high probability, whether this relation is genuine or flipped by the DP noise. Finally, in Phase-3, DPXPlain outputs a table containing the approximately 'top-k' explanation predicates regarding to their influences to the user question, along with their influences and ranks in the form of intervals to help understand the approximation. We further perform an extensive experimental analysis of our approach that includes multiple use-cases, as well as performance experiments, showing that DPXPlain provides insightful explanations with good accuracy and utility.

## 1.4 Organizations

We present the rest of the work in the structure as follows. Chapter 2 includes the notations and definitions of differential privacy, common mechanisms and database. Chapter 3 discuss the computation of local sensitivity for join-count queries. Chapter 4 presents the framework to explain SQL queries under differential privacy. Chapter 5 gives a brief literature review over the related work. Chapter 6 concludes this work with future directions. Appendix A provides extra supporting materials.

# Chapter 2

# Preliminary

## 2.1 Differential Privacy

Differential privacy (DP) [64] is considered the gold standard for private data analysis. An algorithm satisfies DP if its output is insensitive to adding or removing a tuple in the input database. Formally,

**Definition 2.1.1** (Differential Privacy)**.** *A mechanism $\mathcal{M} : I \to \Omega$ is $\epsilon$-differentially private if for any two neighbouring relational database instances $\mathcal{D}, \mathcal{D}' \in I$ and $\forall O \subseteq \Omega$:*

$$|\ln(\Pr[\mathcal{M}(\mathcal{D}) \in O] / \Pr[\mathcal{M}(\mathcal{D}') \in O])| \leqslant \epsilon$$

When $\mathcal{D}$ is a single relation, all neighboring relations are of the form $\mathcal{D}' = \mathcal{D} - \{t\}$. When $\mathcal{D}$ is a multi-relational database with foreign key constraints, then a neighboring instance $\mathcal{D}'$ is gotten by deleting one tuple $t$ in $\mathcal{D}$'s *primary private relation* and cascadingly deleting other tuples that depend on $t$ through foreign keys [106].

Differential privacy has been successfully used to publish summary statistics, synthetic data, machine learning models, and answer SQL queries [136, 168, 4, 159, 127, 94, 106] . It has also been adopted at government [6] and commercial organizations [75, 165, 24, 54, 19].

*Laplace Mechanism.* Laplace Mechanism is a fundamental building block of DP algorithms [68]. It answers a query $Q$ by adding noise drawn from a Laplace distribution scaled to the ratio of the *global sensitivity* of $Q$ and the privacy loss parameter $\epsilon$.

**Definition 2.1.2** (Global Sensitivity)**.** *Given a counting query $Q : I \to \mathcal{R}$, the global sensitivity $GS$ is defined as the max difference of query result from any two neighbouring relational database instances $\mathcal{D}, \mathcal{D}' \in I$ :*

$$GS(Q) = \max_{(\mathcal{D}'-\mathcal{D})\cup(\mathcal{D}-\mathcal{D}')=\{t\}} |Q(\mathcal{D}) - Q(\mathcal{D}')|$$

**Definition 2.1.3** (Laplace Mechanism)**.** *Given a counting query $Q : I \to \mathcal{R}$, a database instance $\mathcal{D} \in I$ and a privacy parameter $\epsilon$, the following noisy query result satisfies $\epsilon$-DP: $Q(\mathcal{D}) + \eta$, where $\eta \sim exp(-\frac{|\eta|\cdot\epsilon}{GS(Q)})$.*

*Composition.* Composition [127] allows a combined privacy guarantee by connecting multiple mechanisms together.

- **Parallel composition:** if two mechanisms take disjoint data as input, the total privacy loss is the maximum privacy loss from each.
- **Sequential composition:** if we run two mechanisms in a sequence on overlapping inputs, the total privacy loss is the sum of each privacy loss.
- **Postprocessing:** if we run a mechanism and postprocess the result without accessing the data, the total privacy loss is only the privacy loss from the mechanism.

# Chapter 3

# Computing Local Sensitivities of Join-Count Queries

## 3.1 Introduction

Understanding how adding or removing a tuple to the relations in the database affects the query output is an important task to many applications [97, 127, 106]. For instance, airline companies need to search for a new flight that can meet the requirements of popular trips. Sales companies should identify the critical part in the production to minimize the number of orders affected by this part. Besides these examples for query explanations, applications of the state-of-the-art privacy guarantee – differential privacy [68] – also need to add sufficient amount of noise to hide the change in the query output due to adding or removing a tuple. In particular, given a database instance $\mathcal{D}$, the maximum change to the query output when one of the given tables in the database adds or deletes a tuple is known as the *local sensitivity* of query on $\mathcal{D}$, and the tuple that matches this maximum change is known as *the most sensitive tuple* in the domain of this database.

Computing the local sensitivity of queries that involves joins of multiple relations is challenging. These queries join several relations (the base relations or transformed relations) into a single table and count the number of tuples in the join output that satisfy certain predicates. For instance, to compute the number of possible connecting flights for a multi-city trip requires a join of flights from the given cities. Similarly, given a database of relations (Region, Nation, Customer, Orders, Lineitem, Part), to compute the number of orders from Europe which purchase lineitem that uses a

particular part requires the join of these six relations with corresponding predicates on the region name and the part name. Prior work on provenance for queries and deletion propagation [11, 31] focus on removing a tuple in the database, but adding new tuples from the full domain is equally important and even harder especially for complex queries over large domains.

Therefore, we are motivated to study the *local sensitivity problem for counting queries with joins*. In particular, given a conjunctive counting query $Q$ and a database instance $\mathcal{D}$, we would like to find the local sensitivity of $Q$ on $\mathcal{D}$ and find a tuple $t^*$ from the full domain whose sensitivity matches the local sensitivity. We make the following contributions for this local sensitivity problem.

- We show that it is NP-hard to find local sensitivity of a conjunctive query in terms of the size of the query, even for the class of acyclic queries.
- We find an efficient algorithm to solve the sensitivity problem and find the most sensitive tuple for *path join queries*, in polynomial time in *combined complexity* [160], *irrespective of the output size*. This is particularly interesting as the well-known algorithms for acyclic and path join queries [171] run in polynomial time in both the size of the input and also the output.
- We present an algorithm, TSens, that efficiently finds the most sensitive tuple for full acyclic conjunctive queries without self-joins using *join trees*, and for a sub-class of general conjunctive queries though extensions using *generalized hypertree decompositions*. TSens runs in polynomial time in both the size of the database and query for an interesting sub-class of queries, which we call 'doubly acyclic queries' that generalizes path queries, and in polynomial time in combined complexity when the maximum degree in the join tree is bounded.

This paper also shows an *application of our proposed technique* TSens *for dif-*

*ferential privacy*. An algorithm satisfies differential privacy if its output is insensitive to adding or removing a tuple in any possible input database. This is usually achieved by injecting sufficient amount of noise to the mechanism in order to hide the changes caused by the most sensitive tuples from the domain. Hence, the utility of the mechanisms crucially depends on the upper bound of the local sensitivity. Prior work [134, 108, 14, 71, 106, 127] for differential privacy focus on a global version of sensitivity — the maximum local sensitivity of all possible database instances. These global sensitivity algorithms either are restricted to a special type of joins [127, 71] or output a much larger value than the true local sensitivity for general join queries on unconstrained database instances [134, 108, 14, 106]. Another line of differentially private algorithms [133, 40, 173, 94] depend on the local sensitivity directly. However, these efforts [133, 40, 173] either offer no efficient and systematic solution for computing the precise bound or derive very loose bound for the local sensitivity of general queries. Moreover, some queries are highly sensitive to adding or removing a tuple, and approaches that just add noise calibrated to the sensitivity fail to offer any utility.

In this paper, we combine TSens with an effective and general purpose technique for DP query answering, called *truncation*. Here the query is run on a truncated version of the database where tuples resulting in high sensitivity are removed. While this introduces error in the query answer (bias), it decreases the sensitivity and the noise added, and thus, the overall error. While prior work has used truncation [127, 106], obtaining high accuracy is challenging as it is nontrivial to determine which tuples to truncate. We show TSens can solve this challenge:

- Our algorithm TSens is able to compute the sensitivity of each tuple in the domain. This allows us to develop a new truncation-based differentially private mechanism (called TSensDP) to answer complex SQL queries by truncating

a proper set of sensitive tuples.

- TSens provides tight estimates on the local sensitivity (as much as 2.2 million times better than the state of the art techniques for sensitivity estimation [95]). Moreover, TSensDP answers queries with significantly lower error than PrivSQL [106], a state of the art method for answering SQL queries.

**Organization:** We discuss preliminaries in Section 3.2. We discuss the problem in Section 3.3. Section 3.4 and 3.5 respectively give algorithms for path join and acyclic conjunctive queries with possible extensions. These algorithms are used to construct a differentially private mechanism in Section 3.6. Section 3.7 presents an experimental evaluation of our approach. Related work and future direction are discussed in Sections 3.8 and 3.9.

## 3.2 Preliminaries

We consider a database instance $\mathcal{D}$ with $m$ tables $R_1, \ldots, R_m$. Relation $R_i$ has attributes $\mathbb{A}_i$ where $k_i = |\mathbb{A}_i|$, and $n_i$ tuples. Database $\mathcal{D}$ has attributes $\mathbb{A}_\mathcal{D} = \cup_{i=1}^{m} \mathbb{A}_i$, and $k = |\mathbb{A}_\mathcal{D}|$ and $n = \sum_{i=1}^{m} n_i$ denotes the total number of attributes and tuples respectively in $\mathcal{D}$. For any attribute $A \subseteq \mathbb{A}_\mathcal{D}$, we use $\Sigma^A$ as the domain of $A$. For multiple attributes $\mathbb{A} = \{A_1, \ldots, A_\ell\} \subseteq \mathbb{A}_\mathcal{D}$, the domain is $\Sigma^\mathbb{A} = \Sigma^{A_1} \times \ldots \times \Sigma^{A_\ell}$. For a tuple $t \in R_i$ and attribute $A \in \mathbb{A}_i$, $t.A$ denotes the value of attribute in $t$, and for $\mathbb{A} \subseteq \mathbb{A}_i$, $t.\mathbb{A}$ denotes a list of values of the attributes in $\mathbb{A}$ with an implicit order.

*Full conjunctive queries without self-joins.* We focus on counting queries that counts the number of output tuples (in bag semantics) for the class of full conjunctive queries (CQ) without self-joins[1], which is equivalent to the *natural join in the SQL semantics* (equal values of common attributes in different relations) and has been extensively

[1]Note that CQs can include the *selection* operator by adding predicates of the form $A = a$, which we discuss in Section A.2.

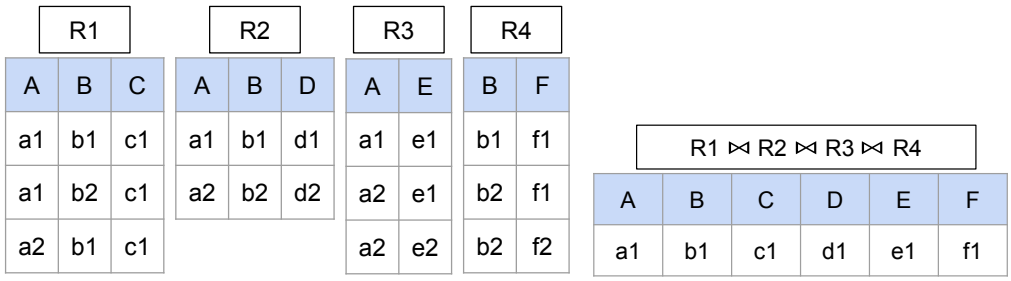
studied in the literature [37, 83, 14]. A CQ $Q$ can be written as a datalog rule as:

$$Q(\mathbb{A}_{\mathcal{D}}) \text{: } -R_1(\mathbb{A}_1), R_2(\mathbb{A}_2), \ldots, R_m(\mathbb{A}_m).$$

Here all the attributes $\mathbb{A}_{\mathcal{D}}$ appear in the *head* of the datalog rule, and $\mathbb{A}_i \cap \mathbb{A}_j \neq \varnothing$ captures natural join. We also call attributes as variables and relations as atoms. We interchangeably use the equivalent relational algebra (RA) form:

$$Q = R_1 \bowtie \cdots \bowtie R_m.$$

where $\bowtie$ with no subscripts refer to natural joins. We denote $Q(\mathcal{D})$ as the query result about $Q$ on $\mathcal{D}$. For example, in Figure 3.1, given 4 relations $(R_1, R_2, R_3, R_4)$ and attributes $(A, B, C, D, E, F)$, where each attribute has a domain of size 2, the natural join of these relations $Q(A, B, C, D, E, F)$ has an output of a single tuple $(a1, b1, c1, d1, e1, f1)$ (Figure 3.1(b)).




(a) a database instance (b) the join result

**Figure 3.1**: An example for a full conjunctive query

*Acyclic Queries.* One sub-class of CQs that has been studied in depth in the literature is the class of *acyclic queries* [20, 76, 5], which we consider as one of the classes of queries in this paper. There are different notions of acyclicity [76], however, in this paper we will use one of the standard notions based on *GYO decompositions* (from Graham-Yu-Ozsoyoglu) [5].

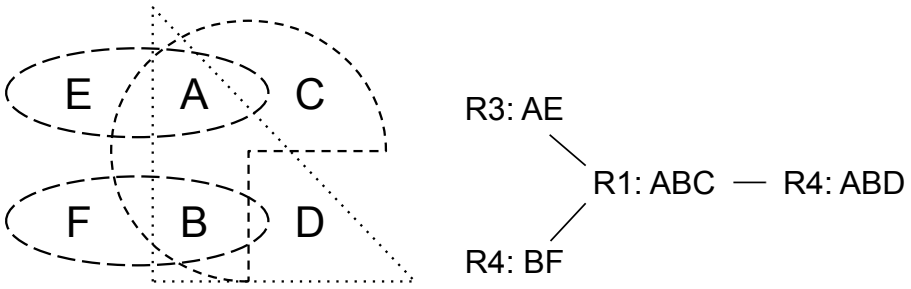
Given a CQ $Q$, the query hypergraph has all the variables or attributes as vertices, and relations appearing in the body of the query as edges. An *ear* is a hyperedge $h$

whose vertices can be divided into two groups that (i) either exclusively belong to $h$, or (ii) are completely contained in another hyperedge $h'$. The GYO-decomposition algorithm repeatedly picks an ear from the hypergraph, removes the vertices that are exclusively in the ear, and then removes the ear from the hypergraph, until the hypergraph is empty or no more ears are found. A CQ is acyclic if the GYO-decomposition algorithm returns an empty hypergraph. Further, the decomposition algorithm results in a *join-tree*, which will be described next (assuming the query hypergraph is connected, otherwise a join-forest is obtained).

*Join-trees.* A *join-tree* $T$ for a CQ whose hypergraph is connected satisfies the following property: for any two relations $R_i, R_j$ appearing in the body of the query such that $\mathbb{A}_i \cap \mathbb{A}_j \neq \varnothing$, all attributes in the intersection appear on a unique path from $R_i$ to $R_j$ in the tree. A join-tree can be obtained for an acyclic query from a GYO-decomposition by adding an edge from relation $R_i$ to another relation $R_j$, when the hyperedge for $R_i$ is being eliminated as an ear, and all the vertices that do not exclusively belong to $R_i$ belong to $R_j$. It is well-known that joins on acyclic queries can be computed in polynomial time in the size of the query and the input (combined complexity, see Section 3.3). The output can be generated by two passes on a join-tree using semi-join operators [5]. Figure 3.2 shows the hypergraph of the query $Q(A, B, C, D, E, F)$ in Figure 3.1 and its GYO decomposition.

## 3.3 Local Sensitivity

*Tuple and Local Sensitivity.* Tuple and local sensitivity of a counting query measure the (maximum) possible change in the number of output tuples when a tuple is added to the database or is removed from the database, and are defined as follows. For two relations $R, R'$ with the same set of attributes, $R \Delta R' = (R - R') \cup (R' - R)$ is the symmetric set difference.




(a) the hypergraph (b) the join tree

**Figure 3.2**: The GYO decomposition of the query in Figure 3.1 (left) and the resulted join tree (right). Here R3(AE), R4(BF) and R4(ABD) are all ears of R1(ABC), so we remove them from the hypergraph and connect them to R1(ABC) in the join tree.

**Definition 3.3.1** (**Tuple Sensitivity).** *Given a tuple $t$ from the domain of any table as $t \in \Sigma^{\mathbb{A}_1} \cup \Sigma^{\mathbb{A}_2} \ldots \cup \Sigma^{\mathbb{A}_m}$, a query $Q$, and a database instance $\mathcal{D}$,*

- upward tuple sensitivity *is:*

  $\delta^+(t, Q, \mathcal{D}) = |Q(\mathcal{D} \cup \{t\}) \ \Delta \ Q(\mathcal{D})|$

- downward tuple sensitivity *is:*

  $\delta^-(t, Q, \mathcal{D}) = |Q(\mathcal{D}) \ \Delta \ Q(\mathcal{D} \backslash \{t\})|$

- tuple sensitivity *is:*

  $\delta(t, Q, \mathcal{D}) = \max\left(\delta^+(t, Q, \mathcal{D}),\ \delta^-(t, Q, \mathcal{D})\right)$

*We often drop $t$, $Q$, and $\mathcal{D}$ and simply use $\delta^+$, $\delta^-$, or $\delta$.*

**Example 3.3.2.** *In Figure 3.1, the tuple $(a1, b1, c1)$ in $R_1$ has a downward tuple sensitivity of 1 as removing this tuple will decrease the join output size by 1. Similarly, the tuple $(a2, b2, c1)$ from the full domain of $R_1$ has a downward tuple sensitivity of $0$ as no such tuple exists in the given database instance. On the other hand, the tuple $(a2, b2, c1)$ has an upward tuple sensitivity $4$, as adding this tuple will increase the output size by 4.*

*Local Sensitivity.* By defining tuple sensitivity, local sensitivity of a query on a given database is equivalent to the maximum tuple sensitivity over all possible tuples.

**Definition 3.3.3** (**Local Sensitivity).** *Given a query $Q$ and a database instance $\mathcal{D}$, the local sensitivity is defined as the maximum tuple sensitivity:*

$$LS(Q, \mathcal{D}) = \max_{t \in \Sigma^{\mathbb{A}_1} \cup \Sigma^{\mathbb{A}_2} \ldots \cup \Sigma^{\mathbb{A}_m}} \delta(t, Q, \mathcal{D})$$

**Example 3.3.4.** *Continue Example 3.3.2 (Figure 3.1), to compute the local sensitivity of this query on the database instance given in Figure 3.1, we need to find the largest possible change to the output size when adding or removing any possible tuple from the domain. The local sensitivity of this query equals to 4, and the most sensitive tuple is $(a2, b2, c1)$ in $R_1$.*

*Sensitivity Problem.* Below we define the sensitivity problem as looking for the most sensitive tuple from the full domain for the general join query and it's sensitivity as the local sensitivity.

**Definition 3.3.5** (**The Local Sensitivity Problem).** *Given a query $Q$ and a database instance $\mathcal{D}$, the local sensitivity problem aims to find the local sensitivity $LS(Q, \mathcal{D})$ of $Q$ on $\mathcal{D}$, and also find a tuple $t^*$ whose tuple sensitivity matches the local sensitivity.*

The problem is trivial when there is only one relation $R$ in the database and $Q(R) = R$, since the local sensitivity is always 1 and any tuple can be the most sensitive tuple. In this paper, we focus on full CQs on multiple relations involving multiple joins.

*Query, Data, and Combined Complexity.* For evaluation of database queries, both the query size (the number of relations and attributes as $m$ and $k$) and the instance size (the number of tuples $n$) are inputs, and therefore based on the parameter that

is considered as variable, three different notions of complexity are considered [160]. *Query or expression complexity* treats the query size $(m, k)$ as a variable and the data size $(n)$ as a constant. *Data complexity* treats data size as a variable and query size as a constant, whereas *combined complexity* treats both query and data size as variables. It is known that even query evaluation for general CQs is NP-hard for query and combined complexity (e.g., by a simple reduction from clique), but has polynomial data complexity.

*Polynomial Data Complexity.* The naive solution of computing the local sensitivity is to check the tuple sensitivity of all possible tuples from all tables. While for downward tuple sensitivity, we need to consider deletions of at most $n$ tuples from the database, for the upward tuple sensitivity when we consider inserting a tuple, the domain of a possible tuple can be arbitrarily larger than $n$ (and even infinite if any attribute has infinite, e.g., integer domain). However, we show below that we can always have a polynomial data complexity by narrowing down the domain of interest.

*Active domain* of an attribute with respect to a given instance typically refers to the set of values of that attribute appearing in the instance. In our context, given an instance $\mathcal{D}$, a relation $R_i$ in $\mathcal{D}$, and an attribute $A \in \mathbb{A}_i$, we use $\Sigma_{act}^{A,i} = \cup_{t \in R_i} t.A \subseteq \Sigma^{\mathbb{A}_i}$ to denote the active domain of $A$ with respect to $R_i$ in $\mathcal{D}$. If an attribute $A$ appears in two relations $R_i, R_j$, it may happen that $\Sigma_{act}^{A,i} \neq \Sigma_{act}^{A,j}$.

For the upward tuple sensitivity, we only consider tuples that can possibly change the result after the insertion, so its attribute values should appear in all other relations. We define representative domain to capture this intuition:

**Definition 3.3.6** (**Representative domain**). *Given an instance $\mathcal{D}$, a relation $R_i$ in $\mathcal{D}$, and an attribute $A \in \mathbb{A}_i$, we define the* representative domain *of $A$ with respect to $R_i$ as $\Sigma_{repr}^{A,i} = \bigcap_{j \,:\, A \in \mathbb{A}_j, j \neq i} \Sigma_{act}^{A,i}$, if $A$ appears in at least two relations, and set it as $\{a\}$ for any arbitrary value $a \in \Sigma_{act}^{A,i}$, if $A$ does not appear in any other relation.*

*The representative domain for a relation $R_i$, denoted by $\Sigma_{repr}^{\mathbb{A}_i} \subseteq \Sigma^{\mathbb{A}_i}$, is defined as $\Sigma_{repr}^{\mathbb{A}_i} = \Sigma_{repr}^{A_1,i} \times \ldots \times \Sigma_{repr}^{A_{k_i},i}$ where $\mathbb{A}_i = \{A_1, \cdots, A_{k_i}\}$ are the attributes in $R_i$.*

**Example 3.3.7.** *From Figure 3.1, the representative domain of $A$ in $R_1$ is $\Sigma_{repr}^{A,1} = \Sigma_{act}^{A,2} \cap \Sigma_{act}^{A,3} = \{a1, a2\} \cap \{a1, a2\} = \{a1, a2\}$*

We show the following theorem with the proof in appendix.

**Theorem 3.3.8.** *The local sensitivity of a full CQ $Q$ with respect to a given instance $\mathcal{D}$ can be computed in polynomial time in data complexity.*

*Proof.* **(Algorithm)** The algorithm works as follows. First, compute the maximum downward tuple sensitivity $\delta^{-*} = \max_{t \in Q(\mathcal{D})} \delta^-(t, Q, \mathcal{D})$ (see Definition 3.3.1), and note the tuple giving the max value. Next, compute the maximum upward tuple sensitivity as $\delta^{+*} = \max_{i \in \{1,\cdots,m\}} \max_{t \in \Sigma_{repr}^{\mathbb{A}_i}} \delta^{+*}(t, Q, \mathcal{D})$, again noting the tuple giving the maximum value. Return $\delta^* = \max(\delta^{+*}, \delta^{-*})$ along with the tuple that led to this highest value.

**(Correctness)** Clearly, if the local sensitivity is obtained by deleting a tuple from $D$, the above algorithm correctly finds that tuple and the sensitivity value by iterating over all tuples in the database instance $D$. The same holds if the local sensitivity is obtained by adding a tuple from the representative domain of any relation. Therefore, we argue that for any relation $R_i$, for any tuple $t \in \Sigma^{\mathbb{A}_i} \backslash \Sigma_{repr}^{\mathbb{A}_i}$, the result size remains unchanged by adding $t$ to $R_i$, i.e., $Q(D \cup \{t\}) = Q(D)$. In other words, for all such tuples $t$, $\delta(t, Q, D) = 0$.

Fix any relation $R_i$ and any tuple $t \in \Sigma^{\mathbb{A}_i} \backslash \Sigma_{repr}^{\mathbb{A}_i}$. Since $t \notin \Sigma_{repr}^{\mathbb{A}_i}$, (i) either there is an attribute $A \in \mathbb{A}_i \cap \mathbb{A}_j$ for some $j \neq i$ such that $t.A \notin \Sigma_{act}^{A,i}$, in which case by adding $t$ the set of result tuples from the full CQ $Q$ does not change as $t$ does not join with any existing tuple in $R_i$, or, (ii) for all shared attributes $A \in \mathbb{A}_i \cap \mathbb{A}_j$ for

some $j$, $t.A \in \Sigma_{act}^{A,i}$, but there exists an attribute $A$ that does not appear in any other relation such that $t.A \neq a$, the value of $A$ chosen in the definition of representative domain. In that case, $t$ can be converted into another equivalent tuple $t'$ from the representative domain by changing $t.A$ with $a = t'.A$ for all such attributes $A$ that do not appear in any other relation, and $\delta(t, Q, D) = \delta(t', Q, D)$, which is considered by the above algorithm.

**(Polynomial data complexity)** Finally we argue that the algorithm runs in time polynomial in $n = |D|$. Note that the active domain of any single attribute $A \in \mathbb{A}_D$ in any relation $R_i$ can be computed in time polynomial in $n$ (in $O(n \log n)$ time if we use sorting to remove duplicates), and $|\Sigma_{act}^{A,i}| \leqslant n$. Since each relation $R_i$ has at most $k$ attributes, $|\Sigma_{repr}^{\mathbb{A}_i}| \leqslant n^k$. Hence the above algorithm iterates over polynomial number of choices for $t$, for each $t$ it evaluates the query $Q(D \cup \{t\})$ or $Q(D - \{t\})$, which can be done in polynomial time in $n$. Hence the total time of the above algorithm is also polynomial in $n$. □

*Combined Complexity: NP-hardness.* Theorem 3.3.8 shows that the sensitivity problem has polynomial data complexity, but the algorithm may run in $O(mn^k)$ time, which is inefficient even for a small number of relations and attributes. Therefore, in this section, we study the combined complexity for the problem and show that the exponential dependency on the query size is essential under standard complexity assumptions not only for general CQs, but also for acyclic queries, thereby motivating the study of efficient, practical algorithms for the sensitivity problem discussed in the subsequent sections.

**Theorem 3.3.9.** *Given a CQ $Q$ and a database $\mathcal{D}$ as input, the sensitivity problem is NP-hard in combined complexity. In particular, checking whether $LS(Q, \mathcal{D}) > 0$ is NP-hard for combined complexity even if $Q$ is acyclic.*

*Proof.* The local sensitivity of a CQ query is defined as the maximum change to the query output size if adding to one of the tables or removing a tuple from one of the tables. The changes to the query output size when removing a tuple is upper bounded by the changes to the query output when adding a tuple. Hence, we will just show that checking if the changes to the query output size is greater than 0 if adding a tuple to one of table is NP-hard.

We give a reduction from the 3SAT problem. Consider any instance of 3SAT $\phi$ with $s$ clauses $(C_1, \ldots, C_s)$ and $\ell$ variables $(v_1, \ldots, v_\ell)$, where each clause is disjunction three literals (a variable or its negation), and the goal is to check if the formula $\phi = C_1 \wedge \cdots \wedge C_s$ is satisfiable. We create an instance of the sensitivity problem $LS(Q, \mathcal{D})$ with $s+1$ relations and $\ell$ attributes in total. For each clause $C_i$ that involves variables $v_{i_1}, v_{i_2}, v_{i_3}$, we add a table $R_i$ with three Boolean attributes $A_{i_1}, A_{i_2}, A_{i_3}$, and insert all possible triples of Boolean values that satisfy the clause $C_i$ into $R_i$ in $\mathcal{D}$. For example, if $C_i$ is $v_2 \vee \bar{v}_5 \vee \bar{v}_7$, then $R_i(A_2, A_5, A_7)$ contains seven Boolean triples $(0,0,0), (0,0,1), \ldots, (1,1,1)$ *except* $(0,1,1)$. In addition, we create an empty relation $R_0(A_1, \ldots, A_\ell)$, which does not contain any tuple in $\mathcal{D}$. The query is:

$$Q(A_1, \ldots, A_\ell) = R_0(A_1, \cdots, A_\ell) \wedge \bigwedge_{i=1,\ldots,m} R_i(A_{i_1}, A_{i_2}, A_{i_3})$$

Note that $Q$ is acyclic, as all of $R_1, \cdots, R_s$ correspond to ears (see Section 3.2). Further, the reduction is in polynomial time in the number of variables and clauses in $\phi$. Next we argue that $\phi$ is satisfiable if and only if $LS(Q, \mathcal{D}) > 0$.

**(only if)** Suppose $\phi$ is satisfiable, and $\mathbf{v} = (v_1 = b_1, \cdots, v_\ell = b_\ell)$ is a satisfying assignment. Then the join of $R_1 \bowtie \cdots \bowtie R_s$ is not empty and $\mathbf{v}$ belongs to their join result. However, $Q(\mathcal{D}) = \varnothing$ as $R_0 = \varnothing$ in $\mathcal{D}$. Now, if we add a tuple corresponding to $\mathbf{v}$ to $R_0$, then $Q(\mathcal{D} \cup \{\mathbf{v}\})$ is no longer empty (at least contains $\mathbf{v}$), and therefore $LS(Q, \mathcal{D}) > 0$.

**(if)** Suppose $LS(Q, \mathcal{D}) > 0$. Hence there exists at least one tuple $t$ such that if it is added to one of the relations $R_0, R_1, \cdots, R_s$, then $|Q(\mathcal{D} \cup \{t\})| > |Q(\mathcal{D})|$. Since

$Q(\mathcal{D}) = \varnothing$ as $R_0$ is empty, this tuple must be inserted to $R_0$ to have a non-empty output. Further, the projection of this tuple to $A_{i_1}, A_{i_2}, A_{i_3}$ for relation $R_i$ must match one of the existing seven tuples of $R_i$ in $\mathcal{D}$ to have a non-empty join result. Therefore, this tuple (Boolean values for $v_1, \cdots, v_\ell$) gives a satisfying solution for $\phi$ by satisfying all the clauses, and makes $\phi$ satisfiable. □

Some intuitions of hard acyclic queries are discussed in Section 3.5.2. Although Theorem 3.3.9 gives a negative result even for acyclic queries, the proof suggests that we may get polynomial-time for special classes of acyclic queries. Indeed, as we show in Sections 3.4 and 3.5, we can get polynomial combined complexity for the sensitivity problem for *path queries*, and for an interesting sub-class that we call *doubly acyclic queries*. The algorithm uses join trees and works for other full acyclic CQ. It gives polynomial running time in combined complexity when the max degree in the join tree is bounded and can be extended to certain non-acyclic CQs.

## 3.4 Path Join Query

In this section, we give an efficient algorithm for a special class of acyclic queries called *path join queries* or *path queries* that run in polynomial time in combined complexity, *irrespective of the output size* (note that the polynomial combined complexity for query evaluation of acyclic and path queries is polynomial in input query, input database, and also the output size, which can be exponential in the query size). A path join query has the following form:

$$Q_{\texttt{path}}(\mathbb{A}_{\mathcal{D}})\text{: } -R_1(A_0, A_1), R_2(A_1, A_2), \ldots, R_m(A_{m-1}, A_m)$$

where $\mathbb{A}_{\mathcal{D}} = \{A_0, A_1, \ldots, A_m\}$ and each relation $R_i$ contains two attributes: $A_{i-1}$ and $A_i$. Note that the above form can be used when two adjacent relations share more than one attributes, since we can replace multiple attributes with a single attribute

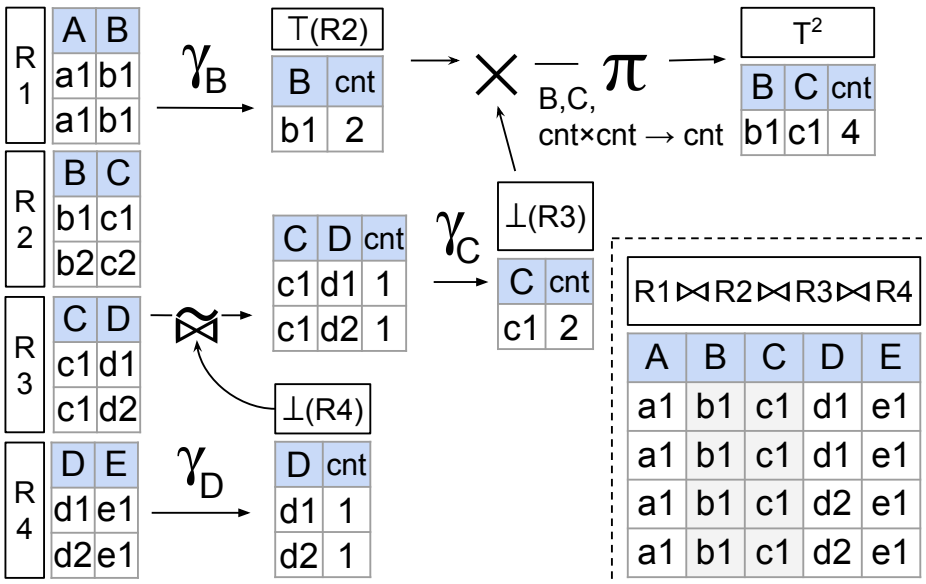



**Figure 3.3**: A path join query $Q_{\texttt{path-4}}(A, B, C, D, E) := -R_1(A, B), R_2(B, C), R_3(C, D), R_4(D, E)$ and the procedure of computing tuple sensitivities from $R_2$. Topjoin $\top(R_1)$ is a group-by count on $B$ from $R_1$, and botjoin $\bot(R_4)$ is a group-by count on $D$ from $R_4$. The join of $R_3$ and $\bot(R_4)$ gives an intermediate table, which is equivalent to a group-by count on attributes $C, D$ from the join between $R_3$ and $R_4$. The following group-by count on $C$ gives the botjoin $\bot(R_3)$. The right bottom table is the result after full join. Now consider a tuple $(b1, c1)$ from $R2$, we found 2 "incoming" paths ending at $b1$ from $\top(R1)$ and 2 "outgoing" paths starting at $c1$ from $\bot(R3)$, so the tuple sensitivity of $(b1, c1)$ is the product of number of "incoming" and "outgoing" paths as $2 \times 2 = 4$.

using combinations of values for multiple attributes. Path joins can capture natural joins in many scenarios, like joining Students, Enrollment, Courses, TaughtBy, Instructors, $\cdots$ relations, or, joining Region, Nation, Customer, Orders, and Lineitem (e.g., in TPC-H data, see Section 3.7). In addition, our algorithm for path join queries will give the basic ideas of our algorithms that can handle general CQs discussed in Section 3.5.

### 3.4.1 Intuition

First, we discuss the basic idea of our algorithm using a toy example of a path query in Figure 3.3 with four relations:

$$Q_{\texttt{path-4}}(A, B, C, D, E) : -R_1(A, B), R_2(B, C), R_3(C, D), R_4(D, E)$$

The number of output tuples affected by adding or removing a tuple $t$ to any of the relations $R_i$ depends on the number of ways in which $t$ can combine with tuples, or in this case 'join-paths', from the remaining relations. Recall that we are using bag semantics from Section 3.2, so a 'join-path' can repeat multiple times and lead to multiple output tuples.

**Example 3.4.1.** *In Figure 3.3, if removing the tuple $R_2(b1, c1)$, all the 4 tuples in the current answer of $Q_{\texttt{path-4}}(\mathcal{D})$ will be removed. These 4 tuples are formed by the join between the 2 tuples from $R_1$ (the "incoming" paths ending at $b1$) and the 2 tuples from $R_3 \bowtie R_4$ (the "outgoing" paths starting at $c1$). On the other hand, if the initial $R_2$ does not have the tuple $(b1, c1)$, inserting $(b1, c1)$ to $R_2$ will add 4 new tuples to the query answer.*

It is easy to see that the sensitivity of adding or removing a tuple $(a_i, b_i) \in R_i$ is the product of the number of incoming paths ending in $a_i$ and the number of outgoing paths starting in $b_i$. However, computing sensitivity by enumerating all join paths is inefficient since the number of incoming/outgoing paths can be exponential in the number of relations (and thus not polynomial in combined complexity). We also need to consider tuples from all the relations including tuples that are not in the active domain but can possibly connect a new path, further worsening the runtime. Hence, we propose the following algorithm to avoid repeated query evaluation and capture the effects of adding and removing tuples simultaneously.

### 3.4.2 Efficient Algorithm for Path Queries

To efficiently represent the data, we first append each relation with an additional attribute `cnt` to record the multiplicity of the other attribute values in that relation. To keep track of the multiplicity of the incoming paths and outgoing paths for the

tuples in $R_i$, we define the following terms.

*Topjoin and botjoin. .* We define *topjoin* $\top(R_i)$ and *botjoin* $\bot(R_i)$ for $R_i$ as follows, which respectively compute the multiplicities of the values of attribute $A_{i-1}$ for the partial path joins from $R_1$ to $R_{i-1}$, and $R_i$ to $R_m$.

$$\top(R_i) = \gamma_{A_{i-1}}\big(\widetilde{\bowtie}(R_1, \ldots, R_{i-1})\big) \tag{3.1}$$

$$\bot(R_i) = \gamma_{A_{i-1}}\big(\widetilde{\bowtie}(R_i, \ldots, R_m)\big) \tag{3.2}$$

The notation $\widetilde{\bowtie}(R_i, \ldots, R_j)$ for $j > i$ used above is a shorthand of two steps: (a) a natural join among $R_i, R_{i+1}, \ldots, R_j$ except the attributes `cnt`, and (b) a projection of the product of these multiplicity attributes `cnt` to a new multiplicity column[2], i.e., abusing RA expressions: $\widetilde{\bowtie}(R_i, \ldots, R_j) = \pi_{\mathbb{A}_i,\ldots,\mathbb{A}_j,(R_i.\texttt{cnt}\times\cdots\times R_j.\texttt{cnt})\to\texttt{cnt}}(R_i \bowtie \ldots \bowtie R_j)$. The group-by operation $\gamma_{\mathbb{A}}(R)$ computes groups according to a set of attributes $\mathbb{A} \subseteq \mathbb{A}_R$, and also sums the counts as the new count attribute, i.e., $\gamma_{\mathbb{A}}(R) = \gamma_{\mathbb{A},\ sum(\texttt{cnt})\to\texttt{cnt}}(R)$.

**Example 3.4.2.** *In Figure 3.3, the topjoin for* $R_2$ *is* $\top(R_2) = \gamma_B(R_1) = \{(B : b1, \texttt{cnt} : 2)\}$ *and the botjoin for* $R_3$ *is* $\bot(R_3) = \gamma_C(\widetilde{\bowtie}(R_3, R_4)) = \{(C : c1, \texttt{cnt} : 2)\}$. *In order to compute the maximum change to the query output by adding/removing a tuple* $(b1, c1)$ *to/from* $R_2$*, we can multiply the* `cnt` *of* $b1$ *from* $\top(R_2)$ *and the* `cnt` *of* $c1$ *from* $\bot(R_3)$*, i.e.,* $2 * 2 = 4$*. This is the largest possible change to the query answer if adding or removing a tuple to* $R_2$*, as the multiplicities of the other values are all smaller than the* `cnt` *of* $b1$ *and the* `cnt` *of* $c1$*.*

Hence, to compute the most sensitive tuple $t_i^*$ within each $R^i$ just requires the tuple $t_i^\top$ from $\top(R_i)$ with the largest multiplicity and the tuple $t_i^\bot$ from $\bot(R_i)$ with the largest multiplicity, i.e.,

$$t_i^\top = \arg\max_{t\in\top(R_i)} t.\texttt{cnt} \text{ and } t_i^\bot = \arg\max_{t\in\bot(R_{i+1})} t.\texttt{cnt} \tag{3.3}$$

[2]A more systematic way to propagate the multiplicity for arbitrary queries has been discussed in the literature, e.g., [8, 11].

Then $t_i^*$ takes $(t_i^\top.A_{i-1}, t_i^\perp.A_i)$ and its sensitivity takes $(t_i^\top.\texttt{cnt} * t_i^\perp.\texttt{cnt})$. For $R_1$, the most sensitive tuple $t_1^*.A_1$ can be derived from the most sensitive tuple in $\perp(R_2)$ and $t_1^*.A_0$ can take any values. Similarly, for $R_m$, the most sensitive tuple $t_m^*.A_{m-1}$ can be derived from the most sensitive tuple in $\top(R_m)$ and $t_m^*.A_m$ can take any values. The most sensitive tuple can be identified from these $m$ tuples $(t_1^*, \ldots, t_m^*)$.

The two relations $\top(R_i)$ and $\perp(R_i)$ for deriving $t_i^*$ do not share any attribute, so their join is equivalent to a cross product. Hence, we are not only getting the tuples in the active domain of $R_i$, but also considering all the tuples from its representative domain (Definition 3.3.6) that can lead to a non-zero local sensitivity by joining with tuples in the other relations, which takes care of both *upward* and *downward* tuple sensitivities.

*Algorithm.* Explicitly computing topjoin (3.1) and botjoin (3.2) can require exponential combined complexity, so we give an iterative approach in Algorithm 1 to compute them in polynomial combined complexity. We first compute $\top(R_2)$ as a base case in the way as topjoin is defined in equation (3.1). Next, we iteratively compute $\top(R_i)$ for $i = 3$ to m in the algorithm. As in the efficient query evaluation for acyclic queries[5], we use *sort-merge joins* to compute the pairwise joins and the then groupby (sort both relations on the join column, join together, then groupby and add the `cnt` values). which can be implemented in $O(n_i \log n_i)$ time as $R_{i-1}$ can join at most one tuple in $\top(R_{i-1})$. We apply a similar approach to compute botjoin for all relations. In total it takes $O(n \log n)$ time.

After preparing topjoin and botjoin, the third step is to combine them together and find the most sensitive tuple. We first find the tuple $t_i^\top$ from $\top(R_i)$ with the highest count and the tuple $t_i^\perp$ from $\perp(R_i)$ with the highest count (Eqn. (3.3)). Then using these tuples, we can construct the most sensitive tuple $t_i^*$ and its sensitivity for each $T^i$ and identify the most sensitive tuple. Note that finding the tuple with the highest count in any of these relations can be done in $O(n_i)$ time, taking $O(n)$ time in total. Therefore, the following theorem holds (formal correctness and complexity

proofs are deferred to the full version due to lack of space):

**Theorem 3.4.3.** *Algorithm 1 solves the sensitivity problem and finds the most sensitive tuple for path join queries. The time complexity is $O(n \log n)$ where $n$ is the size of the database instance irrespective of the size of the output.*

*Connection with Yannakakis's algorithm [171].* Algorithm 1 is inspired by Yannakakis's algorithm [171] that computes join results for acyclic queries in (near)-linear time in the size of the input and output, and can be adapted to compute the join size in near-linear $O(n \log n)$ time only in the input size $n$ in a single pass. In Algorithm 1, we make two passes to compute intermediate topjoins and botjoins, and hence have a similar complexity. We note that, however, this is the *total* time complexity for sensitivity computation considering all possible tuple additions and deletions from all relations. If we naively repeat the $O(n \log n)$ time algorithm inspired by [171] to compute the output join size for all possible tuple deletions, the time would be multiplied by $n$. Further, if we repeat this algorithm for all possible tuple insertions using the representative domain in Definition 3.3.6, the time would be (approximately) multiplied by a factor of $O(n^2)$. Algorithm 1 provides an approach to the sensitivity problem using ideas from [171] in $O(n \log n)$ time for path queries (we compare these experimentally in Section 3.7.2.)

However, the above theorem is not generalizable to even all acyclic queries (recall Theorem 3.3.9). In the next section, we give algorithms that can handle acyclic CQs in parameterized polynomial time and run in sub-quadratic time for a class called 'doubly acyclic queries' that generalizes path queries.

## 3.5 Acyclic and Other Join Queries

In this section, we discuss a general solution to acyclic queries (Section 3.5.1), and then present an efficient algorithm with additional parameters in the running time

**Algorithm 1** Compute Local Sensitivity of a Path Join Query and the corresponding Most Sensitive Tuple

**Input:** Path query $Q_{\texttt{path}}(A_0 \dots A_m)$, the database instance $\mathcal{D}$
**Output:** $LS(Q, \mathcal{D})$, and a most sensitive tuple $t^*$

1: **procedure** LSPathJoin
   I) Prepare topjoin
2: $\top(R_2) = \gamma_{A_1} R_1$ /* *also adds the* `cnt` *values* */
3: **for** $i = 3, \dots, m$ **do**
4: $\top(R_i) = \gamma_{A_{i-1}} \widetilde{\bowtie}(\top(R_{i-1}), R_{i-1})$ /* *also multiplies and adds the* `cnt` *values* */
5: **end for**
   II) Prepare botjoin
6: $\bot(R_m) = \gamma_{A_{m-1}} R_m$ /* *also adds the* `cnt` *values* */
7: **for** $i = m-1, \dots, 2$ **do**
8: $\bot(R_i) = \gamma_{A_{i-1}} \widetilde{\bowtie}(\bot(R_{i+1}), R_i)$ /* *also multiplies and adds the* `cnt` *values* */
9: **end for**
   III) Select most sensitive tuple
10: **for** $i = 1, \dots, m$ **do**
11: $t_i^{\top} = \arg\max_{t \in \top(R_i)} t.\texttt{cnt}$
12: $t_i^{\bot} = \arg\max_{t \in \bot(R_{i+1})} t.\texttt{cnt}$
13: $t_i^* = (t_i^{\top}.A_{i-1}, t_i^{\bot}.A_i)$ with sensitivity $\texttt{cnt} = (t_i^{\top}.\texttt{cnt} * t_i^{\bot}.\texttt{cnt})$ /* *when* $i = 1$ *(or* $i = m$*),* $A_0$ *and* $A_m$ *takes any value and* $t_1^{\top}.\texttt{cnt} = t_m^{\bot}.\texttt{cnt} = 1$. */
14: **end for**
15: $t^* = \arg\max_{i=1,\dots,m} t_i^*.\texttt{cnt}$
16: $LS = t^*.\texttt{cnt}$
17: **return** $LS, t^*$
18: **end procedure**

complexity of the algorithm (Section 3.5.2). In Section 3.5.3, we show that a class of queries that we call *doubly-acyclic queries* has a polynomial combined complexity. Last, we discuss several extensions of this algorithm to general cases.

We consider queries with no self joins; i.e., there are no duplicated relations in the query body. For simplicity, we assume an acyclic full CQ of the following form:

$$Q_{\texttt{acy}}(\mathbb{A}_{\mathcal{D}}) : -R_1(\mathbb{A}_1), \dots, R_m(\mathbb{A}_m).$$

We assume that the query hypergraph (Section 3.2) is connected. We also assume that each attribute appears in at least two relations in the body; further, there are no selections in the body, and no projections in the head of the query, i.e., $\mathbb{A}_{\mathcal{D}} = \cup_{i=1}^{m} \mathbb{A}_i$, and the total number of attributes $|\mathbb{A}_{\mathcal{D}}|$ is $k$. These assumptions,

except the no-self-join assumption (which introduces new challenges and we leave it as a future direction), are without loss of generality as how they can be relaxed using our algorithm is discussed in Section 3.5.4.

### 3.5.1 Basic Idea Using Join Trees

Similar to a path join query, the sensitivity of adding or removing a tuple in a relation depends on the number of the incoming/outgoing paths to/from this tuple. To compute the multiplicity of these paths, we represent an acyclic query using a join tree constructed from its GYO decomposition (Section 3.2). For example, given the join tree for a join between 12 relations in Figure 3.4, in order to compute the sensitivity of tuples in $R_8$ (node 8), we need to construct the join between two groups of relations: (i) the set of relations that are not the descendants of node 8, i.e., $\{11, 12, 9, 10, 1, 2, 7, 3, 4\}$ — the incoming paths and (ii) the relations rooted at node 8, i.e., $\{5, 6\}$ — the outgoing paths.

Formally, we denote this join tree derived based on GYO decomposition as $\mathcal{T}(V, E)$, where the nodes in the tree $V = \{R_1, \ldots, R_m\}$ correspond to relations in the query. Let $p(R_j)$ denote the parent of node $R_j$ in the join tree, $C(R_j)$ denote the children of $R_j$, and $N(R_j)$ denote the *neighbors* or *siblings* of $R_j$, i.e. $N(R_j) = C(p(R_j))\backslash\{R_j\}$. We denote $\mathcal{T}(R_j)$ as the relations in the subtree rooted at $R_j$, while $\mathcal{T}^c(R_j)$ is the set of relations in the complement of $\mathcal{T}(R_j)$ on the tree $\mathcal{T}(V, E)$.

**Example 3.5.1.** *In Figure 3.4, the complementary tree of $R_8$, $\mathcal{T}^c(R_8)$, includes $\{11, 12, 9, 10, 1, 2, 7, 3, 4\}$ and the subtrees at the children of $R_8$ are leaf node 5 and leaf node 6. Computing the joins of these relations can be exponential in the number of the relations or the number of attributes (and thus not have a polynomial combined complexity). We propose an algorithm to make two passes on $\mathcal{T}$ to efficiently track the incoming/outgoing paths.*

### 3.5.2 Efficient Algorithm for Acyclic Queries

**Topjoin and botjoin.** To compute the sensitivity of the tuples in $R_i$, we need to evaluate the join between two groups of relations: (i) the complementary tree of $R_i$, and (ii) the subtrees rooted at the children of $R_i$. These two groups of relations can be represented as *topjoin* $\top$ and *botjoin* $\bot$:

$$\top(R_i) = \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}}(\widetilde{\bowtie}(\mathcal{T}^c(R_i))) \tag{3.4}$$

$$\bot(R_i) = \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}}(\widetilde{\bowtie}(\mathcal{T}(R_i))) \tag{3.5}$$

The operators $\widetilde{\bowtie}$ and $\gamma$ are the same as the ones used for path queries in Section 3.4.2 which take into account multiplicities.

Since $\mathcal{T}$ is a join tree, for each attribute $A$, the relations that contain $A$ always form a connected subtree. Hence, all the attributes of $R_i$ that appear in the join tree should be either in the attributes of its complementary tree or the attributes of its descendants. Then applying group by according to the attributes in $R_i$, $\mathbb{A}_{R_i}$, on the join between all the remaining relations gives us the sensitivities of all the tuples in representative domain of $R_i$, i.e.,

$$T^i = \gamma_{\mathbb{A}_{R_i}}\big(\widetilde{\bowtie}(\top(R_i), \{\bot(R_j) \mid R_j \in C(R_i)\})\big) \tag{3.6}$$

We name $T^i$ the $m$ultiplicity table of $R_i$. The expression for $T^i$ is simpler if $R_i$ is the root or a leaf. We will discuss it in the algorithm below.

*Algorithm.* Algorithm 2 takes as input the join tree $\mathcal{T}$ of the acyclic query and database $\mathcal{D}$ . It first prepares botjoin and topjoin for each node with an iterative approach. To prepare botjoin $\bot(R_i)$ (3.5), we start from the leaf nodes. The botjoin of a leaf node $R_i$ is simply a group by on the common attributes between $R_i$ and its parent node $p(R_i)$ on $R_i$. Next, we compute $\bot$ for other nodes in a post-order

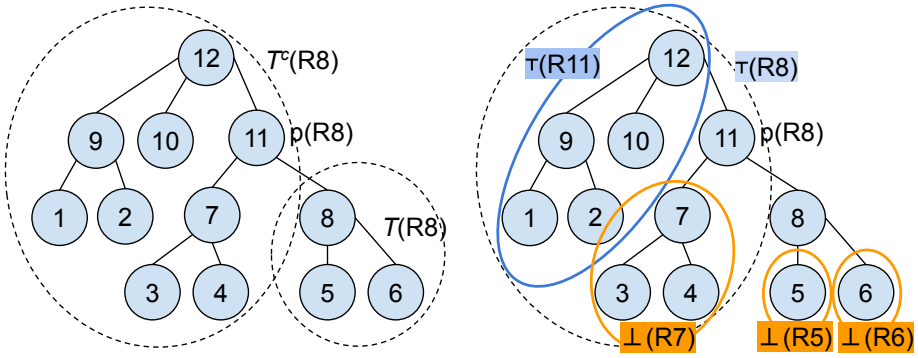



**Figure 3.4**: The given join tree consists of 12 relations. The node with number $i$ means $R_i$. The left figure circles the subtree rooted at $R_8$, $\mathcal{T}(R_8)$, and the complementary subgraph of $\mathcal{T}(R_8)$, $\mathcal{T}^c(R_8)$. The right figure highlights that the multiplicity table of $R_8$, $T^8$, requires the join between the topjoin of $R_8$, $\top(R_8)$ and the botjoins of all its children, $\{\bot(R_5), \bot(R_6)\}$. The topjoin of $R_8$ can be iteratively computed from the join between the topjoin of the parent of $R_8$, i.e., $\top(R_{11})$, and the botjoins of the neighbors of $R_8$, i.e., $\{\bot(R_7)\}$.

traversal of the tree with this iterative formula:

$$\bot(R_i) = \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}}(\widetilde{\bowtie}(R_i, \{\bot(R_j) \mid R_j \in C(R_i)\})) \tag{3.7}$$

For each $\bot(R_i)$, the join starts with $R_i$ and follows by the children of $R_i$ one by one.

**Example 3.5.2.** *In Figure 3.4, we first compute the botjoins for all the leaf nodes including* $\bot(R_1)$, $\bot(R_2)$, $\bot(R_3)$, $\bot(R_4)$,$\bot(R_5)$, $\bot(R_6)$, $\bot(R_{10})$. *Next, if all the children of a node has a computed botjoin, then we can compute the botjoin of this node, e.g.* $\bot(R_8) = \gamma_{\mathbb{A}_{R_8} \cap \mathbb{A}_{R_{11}}} \left(\widetilde{\bowtie} \left(\widetilde{\bowtie}(R_8, \bot(R_5)), \bot(R_6)\right)\right)$, *where* $R_5$ *and* $R_6$ *are the children of* $R_8$, *and* $R_{11}$ *is the parent of* $R_8$.

To prepare topjoin $\top(R_i)$, we start with the children of the root node. The topjoin of each child $R_i$ of the root is the join between the root and the botjoins of all its neighbors followed by a group by on the common attributes between $R_i$ and the root. Next, we compute topjoin $\top$ for other nodes in a pre-order traversal of the tree with this iterative formula:

$$\top(R_i) = \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}}(\widetilde{\bowtie}(p(R_i), \top(p(R_i)), \{\bot(R_j) \mid R_j \in N(R_i)\})) \tag{3.8}$$

For each $\top(R_i)$, the join starts with $p(R_i)$ and $\top(p(R_i))$ and follows by the botjoin of the neighbors of $R_i$ one by one. For example, computing the topjoin of $R_8$ in Figure 3.4 requires the join of its parent $R_{11}$, the topjoin of its parent $\top(R_{11})$, and the botjoins of all its neighbors, here $\{\bot(R_7)\}$.

After preparing all the topjoins and botjoins, we combine these results to obtain the multiplicity tables $T^i$ for $i = 1, \ldots, m$ based on Eqn. (3.6). For instance, to compute $T^8$ in Figure 3.4, we join the topjoin of $R_8$ and the botjoins of all its children, $\{\bot(R_5), \bot(R_6)\}$. This does not require the topjoins of the root node or the botjoins of the leaf nodes. We iterate all the multiplicity tables $T_i$ and find the tuple with maximum `cnt`. This tuple is returned as the most sensitive tuple with its `cnt` as the local sensitivity of this query on the given database instance.

The runtime of the algorithm depends on the *max degree* of the tree, which is the maximum children size + 1 (for a non-root node including the parent) of any node in the tree.

**Theorem 3.5.3.** *Algorithm 2 computes the local sensitivity of an acyclic CQ and also finds the corresponding most sensitive tuple. Given $m$ tables with $k$ attributes in total, $n$ tuples in the database instance, and a join tree of the query with max degree $d$, the time complexity is $O(m\ d\ n^d\ \log n)$.*

*Proof.* (sketch) If two nodes $R_i$ and $R_j$ share a common attribute $A$, then all the nodes on the path between $R_i$ and $R_j$ in the join tree also contain $A$. Hence, the iterative equations (3.7) and (3.8) correctly compute the botjoin (3.5) and topjoin (3.4) by tracking multiplicities through common attributes.

Now we analyze the running time of the algorithm. Notice that all joins in any topjoin equation (3.8) and botjoin equation (3.7) have at least one common join attribute, according to the definition of join tree and the fact that the projection of $\top(R_i)$ and $\bot(R_i)$ is always the subset of $\mathbb{A}_i$ and $\mathbb{A}_{p(R_i)}$. For botjoin (3.7), we join

**Algorithm 2** Compute Local Sensitivity of an acyclic CQ and the corresponding Most Sensitive Tuple

**Input:** Acyclic CQ $Q_{\texttt{acy}}(\mathbb{A}_{\mathcal{D}})$ as a join tree $\mathcal{T}$, the database $\mathcal{D}$
**Output:** $LS(Q_{\texttt{acy}}, \mathcal{D})$, and the most sensitive tuple $t^*$

1: **procedure** LSAcyclicJoin
2: I) Compute $\bot(R_i)$ in post-order (leaf to root)

$$\begin{cases} \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}}(R_i), & \text{if } R_i \text{ is leaf} \\ \gamma_{\mathbb{A}_i} \widetilde{\bowtie}(R_i, \{\bot(R_j) \mid R_j \in C(R_i)\}), & \text{if } R_i \text{ is root} \\ \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}} \widetilde{\bowtie}(R_i, \{\bot(R_j) \mid R_j \in C(R_i)\}), & \text{o.w.} \end{cases}$$

3: II) Compute $\top(R_i)$ in pre-oder (root to leaf)

$$\begin{cases} \varnothing, & \text{if } R_i \text{ is root} \\ \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}} \widetilde{\bowtie}(p(R_i), \{\bot(R_j) \mid R_j \in N(R_i)\}), & \text{if } p(R_i) \text{ is root} \\ \gamma_{\mathbb{A}_i \cap \mathbb{A}_{p(R_i)}} \widetilde{\bowtie}(p(R_i), \top(p(R_i)), \{\bot(R_j) \mid R_j \in N(R_i)\}), & \text{o.w.} \end{cases}$$

4: III) Prepare multiplicity tables $T^i$ of nodes for $i = 1, \ldots, m$

$$\begin{cases} \gamma_{\mathbb{A}_i}(\top(R_i)), & \text{if } R_i \text{ is leaf} \\ \gamma_{\mathbb{A}_i} \widetilde{\bowtie}(\{\bot(R_j) \mid R_j \in C(R_i)\}), & \text{if } R_i \text{ is root} \\ \gamma_{\mathbb{A}_i} \widetilde{\bowtie}(\top(R_i), \{\bot(R_j) \mid R_j \in C(R_i)\}), & \text{o.w.} \end{cases}$$

IV) Select the most sensitive tuple
5: $t^* = \arg\max_{t: t \in T^i, i=1,\ldots,m} t.\texttt{cnt}$ and $LS = t^*.\texttt{cnt}$
6: **return** $LS, t^*$
7: **end procedure**

relations with $R_i$ one at a time using sort-merge-join and then do groupby count. The size of each join is always $\leqslant n_i$ since each tuple $R_i$ can join at most one tuple from any botjoin of its children. In total, it takes $O(d_i n_i \log n_i)$ for each botjoin and $O(mn \log n)$ for all botjoins since the summation of $d_i$ is m and $n_i \leqslant n$.

Next we discuss the running time for step III) in Algorithm 2. Unlike the computation for topjoins and botjoins, this step requires joining the botjoins of all the children of a node with the topjoin of that node $R_i$, and all these partial joins may not share any attributes in general (although all the join attributes are still subsets of $\mathbb{A}_i$). Hence, for arbitrary acyclic joins, there can be at most $d - 1$ joins in

this step for each $R_i$ where $d$ is the max degree in $\mathcal{T}$, which can be computed in $O(n^d \log n^d) = O(dn^d \log n)$ time even by the brute force approach. The total time to compute the multiplicity tables $T^i$ for $m$ relations is $O(mdn^d \log n)$. Hence, the total time complexity is $O(mdn^d \log n)$. □

Similar to the discussion in Section 3.4.2, the computation of botjoins $\perp(R_i)$ and topjoins $\top(R_i)$ in Algorithm 2 is inspired by Yannakakis's algorithm [171], which can track counts of intermediate tuples from the leaves to the root in a bottom-up pass, whereas in the second top-down pass, we need to traverse the join tree to compute the topjoins $\top(R_i)$. As explained earlier for path queries, Algorithm 2 computes changes in the join size for *all possible* tuple deletions and additions, and naively repeating [171] to evaluate query on all possible databases formed by adding or removing a tuple does not give the desired complexity. In fact, [171] works in near-linear time in the input size $n$ to output the output join size for any acyclic join query (and has polynomial combined complexity), whereas the sensitivity problem is NP-hard in combined complexity even for acyclic queries as stated in Theorem 3.3.9.

Given the NP-hardness result in Theorem 3.3.9, we next show an example acyclic query that may take $\omega(mn)$ time for the $T^i$ step. Suppose we have an acyclic query as $Q(A, B, C) : -R_1(A, B, C), R_2(A, B), R_3(B, C), R_4(C, A)$ and we want to compute the multiplicity table $T^1$ for $R_1$. Given botjoins of $R_2(A, B)$, $R_3(B, C)$ and $R_4(C, A)$, we have a cyclic join among them, and in worst the join size is $O(n^{3/2})$ according to the AGM bound [17]. In general, if we replace the children with more complex queries, and if the number of relations (or the degree) is larger, the time to compute this join may be larger. Note that some of the complexity of this problem comes from the bag semantics considered in our model (that is also relevant for applications of sensitivity related to differential privacy), as for set semantics, changes in the join size can be computed more efficiently when a tuple is added or removed from a table.

However, for bag semantics, adding any tuple, say to $R_1$, may increase the sensitivity significantly for $Q$ (product of the multiplicities of the edges forming the triangle), which adds to the complexity.

### 3.5.3 Doubly Acyclic Join Queries

For an acyclic query, if there exists a join tree $\mathcal{T}$ constructed from the GYO decomposition such that for each node $R_i$ in $\mathcal{T}$, the join between its parent $p(R_i)$ and its children $C(R_i)$ is also acyclic, then we say this query is a *doubly acyclic join query*. Given this property, the computation of the multiplicity table $T_i$ for $R_i$ involves an acyclic join between the topjoin and botjoins and hence has a time complexity $O(d_i n_i \log n_i)$, where $d_i$ is the node degree of $R_i$ in $\mathcal{T}$. Since the sum of all node degrees is $m$ and $n_i \leqslant n$, the total time complexity to compute $T^i$ for all nodes is $O(mn \log n)$. When $d_i$ is a constant, such as at most 2, the complexity is written as $O(n \log n)$, which also matches the total runtime of Algorithm 2 including the computation of topjoins and botjoins.

Notice that a path join query is a special case of doubly acyclic join query, because for each $R_i$, $\top(R_i)$ and $\bot(R_{i+1})$ (assuming $R_{i+1}$ is the child of $R_i$ in $\mathcal{T}$) share no attributes. and therefore is an acyclic join. The time complexity of path join queries in Algorithm 1 also matches the time complexity of doubly acyclic join queries.

### 3.5.4 Extensions

In this section we briefly discuss how to extend our framework relaxing the assumptions listed at the beginning of Section 3.5.

*Selections:* We can easily extend Algorithm 2 to handle queries with arbitrary selection conditions (that can be applied to each tuple individually in any relation) in the body of the query by assigning 0 sensitivity to the tuples fail the selection

condition.

*Disconnected join trees:* If the hypergraph of a query is not connected, Algorithm 2 can be applied to each join tree and merge them back to update each tuple sensitivity.

*General joins*: For a non-acyclic join query, if there exists a *generalized hypertree decompositions*[7] such that we can find a join plan from this tree by assigning each relation to one node, Algorithm 2 can be extended by computing the multiplicity table as including other relations within the same node, and the time complexity is parameterized by the max number $p$ of relations within a node as $O(mpdn^{pd} \log n)$. This is implemented in our experiments; q3 from TPC-H queries, and $q_{\triangle}$, $q_{\square}$ from Facebook queries are all non-acyclic queries and their generalized hypertree decompositions are shown in Figure 3.5.

*Efficient approximations:* We can extend our algorithm to tradeoff accuracy in the sensitivity for better runtime. As our experiment will show, the multiplicity tables that topjoins and botjoins compute can grow quadratically or faster in the input size depending on the query. To make the computation scalable, we can maintain the top $k$ frequent values instead of all the frequencies in the top and botjoins. We can set the frequencies of the rest of the active values in the top and botjoins to the kth largest frequency. This approach gives an upper bound of tuple sensitivity but can speed up runtime.

*Self Joins:* Acyclic join queries with self-joins can not be captured by our algorithms, because we only allow a relation to appear once in the query. For each relation, we compute the joins for the rest of relations to summarize how tuples from this relation can affect the full join. A possible workaround is to join the repeated base relations as a single and combined relation, run our algorithm, and then link the effect of adding or removing a tuple from the base relation to the combined relation

and the effect of adding or removing a tuple from the combined relation to the rest. However, it is challenging to find all possible insertions to the base relation that allows the combined relation to join all possible pairs of "incoming" and "outcoming" path. We defer this line of research to future work.

*Other:* For attributes that appear only once in the query body, we ignore them in Algorithm 2 but in the end we extrapolate a value for these attributes.

## 3.6 Truncation

In this section, we will show how to use our algorithm TSens (section 3.5.2) for computing sensitivity to develop accurate differentially private algorithms. Section 3.6.2 discusses how the tuple sensitivity measures can be used to develop accurate DP algorithms.

### 3.6.1 Challenge

We first consider applying Laplace mechanism to answer a join counting query, which adds a noise scaled to the global sensitivity. Unlike the local sensitivity of a query which depends on the given database instance, the global sensitivity of a query finds the largest possible local sensitivity among all possible database instances. Consider the join query in Figure 3.1(b), Example 3.3.4 shows that it has a local sensitivity of 4 on the database instance shown in Figure 3.1(a). However, there exist other database instances with a much larger local sensitivity than 4. For example, if $R_2$ in Figure 3.1(a) has 1000 copies of $(a_1, b_1, d_1)$ which results in 1000 copies of $(a_1, b_1, c_1, d_1, e_1, f_1)$ in the join output, removing $(a_1, b_1, c_1)$ from $R_1$ can result in a change of 1000 in the output size.

If there is no *a priori* bound on the number of tuples that share the same join key, the global sensitivity of the query will be unbounded. Hence, Laplace mechanism

cannot be directly applied to query with unbounded global sensitivity. Prior work takes two approaches to answering queries with joins under DP – smooth sensitivity [94] and Lipshitz extensions [100, 25, 40, 106]. The former approach tunes noise to a smooth upper bound on the local sensitivity. However, the smooth sensitivity approach was shown to be too inaccurate [106].

The Lipshitz extension approach is to rewrite the original query $Q$ that has an unbounded sensitivity into a different query $Q'$ that (a) has a bounded global sensitivity, and (b) evaluates to roughly the same answer. These techniques are superior to smooth sensitivity based techniques as they introduce some bias (as $Q(\mathcal{D}) \neq Q'(\mathcal{D})$) to reduce the noise added to the answer. Many Lipshitz extension techniques are query specific [100, 25] or fairly inefficient in practice [40]. One class of efficient and general purpose Lipshitz extension techniques is *truncation* that execute the query $Q$ on a truncated version of the database $T(\mathcal{D})$ [127, 106]. The truncation is done in such a way that $Q(T(\cdot))$ has bounded global sensitivity. For a join, this might mean removing rows from the database such every join key has a bounded selectivity. We will next show a truncation based algorithm for answering SQL aggregation queries with joins based on the tuple sensitivities computed by TSens.

One effective and general purpose technique from prior work is *truncation* that executes the query $Q$ on a truncated version of the database $T(\mathcal{D})$ [127, 106]. The truncation is done in such a way that $Q(T(\cdot))$ has a bounded global sensitivity. For a join, this might mean removing rows from the database such every join key has a bounded selectivity. We will next show a truncation based algorithm for answering SQL aggregation queries with joins based on the tuple sensitivities.

### 3.6.2 Truncation via Sensitivity

TSens *Truncation.* The idea behind our algorithm is to (a) identify tuples in the database (i.e., in the primary private relation) that have a sensitivity greater than a *sensitivity threshold*, and (b) remove all tuples with sensitivity greater than the sensitivity threshold.

**Definition 3.6.1** (TSens Truncation)**.** *Given a query $Q$, a database $\mathcal{D}$ with primary private relations $\mathcal{PR}$, and a sensitivity threshold $i$, the truncation operator $T_{\text{TSens}}$ transforms the database as:*

$$T_{\text{TSens}}(Q, \mathcal{D}, i) = \{t \in \mathcal{D} \mid t \in \mathcal{PR} \Rightarrow \delta(t, Q, \mathcal{D}) \leqslant i\}$$

The global sensitivity of $Q(T_{\text{TSens}}(Q, \cdot, \tau))$ is $\tau$. If we add or remove a tuple with sensitivity more than $\tau$, the query result does not change as the new tuple will be truncated or has already been truncated. Since the largest possible tuple sensitivity is $\tau$ for any database, the global sensitivity is $\tau$. Hence, given a join query $Q$ with high global sensitivity, we can first apply $Q(T_{\text{TSens}}(Q, \cdot, \tau))$ to the database and then apply Laplace mechanism with smaller noise (due to smaller global sensitivity) on the transformed database. However, the transformed database also introduces bias if too many tuples are truncated. Hence, we would like to find a truncation threshold that minimizes the expected sum of bias and noise.

*Finding truncation threshold.* If setting $\tau$ to be the local sensitivity of the query $Q$, then $Q(\mathcal{D}) = Q(T_{\text{TSens}}(Q, \cdot, \tau))$, i.e., no bias is introduced. However, using local sensitivity directly violates DP. Moreover, the global sensitivity of querying the local sensitivity of a join query is unbounded, we cannot use Laplace mechanism to release a noisy local sensitivity. Instead, line in PrivSQL [106], we apply the *sparse vector technique* (SVT) [118] to find the optimal truncation threshold that is close to the local sensitivity.

**Algorithm 3** Learning truncation threshold ($Q$, $\mathcal{D}$, $l$, $\epsilon_{tsens}$)

$\epsilon_1 \leftarrow \epsilon_{tsens}/10$
$\epsilon_2 \leftarrow \epsilon_{tsens} - \epsilon_1$
$\hat{Q} \leftarrow Q(\mathcal{D}) + Lap(l/\epsilon_1)$
$q \leftarrow \{\frac{Q(T_{\text{TSENS}}(Q,\mathcal{D},i))-\hat{Q}}{i} \mid i = 1, \ldots, l-1\}$
Set $\tau \leftarrow SVT(q, 0, \epsilon_2)$

For a query $Q$ and a database $\mathcal{D}$, let $\ell$ be an upper bound on the local sensitivity. We first release a noisy version of $Q(T_{\text{TSENS}}(Q, \mathcal{D}, l))$ as $\hat{Q}$ using the Laplace mechanism with global sensitivity as $\ell$. Next, we run the SVT method that checks whether $q_i > 0$ for $i = 1, \ldots, \ell - 1$, where

$$q_i = \frac{Q(T_{\text{TSENS}}(Q, \mathcal{D}, i)) - \hat{Q}}{i}$$

Since the global sensitivity of $Q(T_{\text{TSENS}}(Q, \mathcal{D}, i))$ is $i$, the global sensitivity of each $q_i$ is a constant 1. SVT stops the first time (noisy) $q_i$ is above the (noisy) threshold 0 and reports $i$. We take this $i$ as the truncation threshold $\tau$, and answer the query $Q$ using $Q(T_{\text{TSENS}}(Q, \mathcal{D}, i))$. A part of the privacy budget $\epsilon_{tsens}$ is used to release $\hat{Q}$ and run SVT for finding the truncation threshold $\tau$. The rest $\epsilon - \epsilon_{tsens}$ is used to answer the query.

**Theorem 3.6.2.** *The algorithm that finds the truncation threshold satisfies $\epsilon_{tsens}$-DP and releasing a noisy answer as $Q(T_{\text{TSENS}}(Q, \mathcal{D}, \tau)) + Lap(\frac{\tau}{\epsilon-\epsilon_{tsens}})$ satisfies $(\epsilon-\epsilon_{tsens})$-DP. Together the mechanism satisfies $\epsilon$-DP.*

*Discussion.* Our solution is inspired by Wilson et al [165], but they handle can only handle a single join (and not self joins), while we can handle a wider sub-class of full conjunctive queries without self joins. Moreover, Wilson et al set the sensitivity threshold manually, while we automatically identify the threshold given an estimated upper bound.

Our algorithm truncates primary private tables while in PrivSQL [106], truncation happens at non-primary private tables. PrivSQL truncates tuples with high

frequencies, but it doesn't mean that they join with the tuple of the highest tuple sensitivity. In contrary, truncation by tuple sensitivity is a finer truncation strategy which reduces global sensitivity and bias at the same time.

Our algorithm for finding the most sensitive tuples can be easily extended for TSens by storing the multiplicity table for the primary private table. Our truncation algorithm takes in estimated upper bound of tuple sensitivity $\ell$. Our algorithm will still ensure DP regardless of the value for $\ell$, but the value of $\ell$ can affect the accuracy. We illustrate the impact of $\ell$ on the accuracy in the evaluation.

## 3.7 Experiments

We evaluate the efficiency and accuracy of TSens. Experiments are designed to answer following questions:

- How tight is the local sensitivity computed by TSens compared to other algorithms like elastic sensitivity [95]?
- How does TSens' runtime compared to that of (a) the elastic sensitivity algorithm and (b) query evaluation?
- Does the truncation with TSens mechanism result in more accurate differentially private query answering than prior work like PrivSQL [106]?

We use synthetic datasets from TPC-H benchmark [3] and real world datasets of Facebook ego-networks from SNAP [113] and designed seven full conjunctive queries with different query complexities to evaluate the performance of TSens. These queries are also used to evaluate the performance of DP mechanism supported by TSens. The results are compared with the sensitivity engine Elastic from 'Flex' [95] and with the differentially private SQL answering engine PrivSQL from 'Pri-

vateSQL' [106]. We name our sensitivity algorithm as TSens and its DP application as TSensDP.

TSens is implemented by 1.5k lines of python and it uses Postgres as the database engine. We extend 'Elastic Sensitivity' so it can compute the sensitivity of a cross product. As to the DP experiment, we also borrow the concept of policy and primary private relation from 'PrivateSQL' and assume the query has already been rewritten according to the policy.

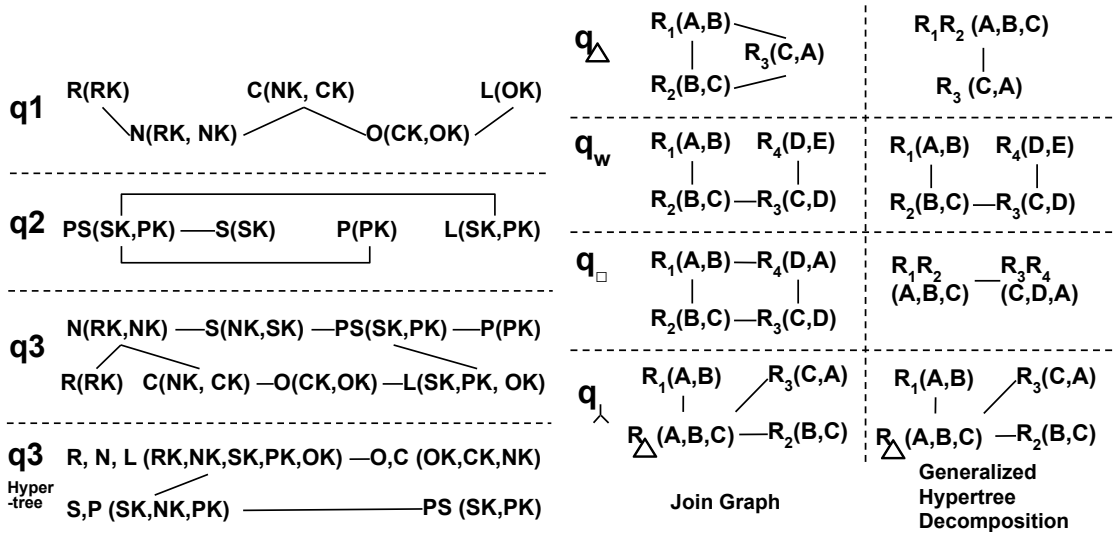
A summary of our key findings:

- TSens achieves at most 2,200,000 times smaller local sensitivity compared to Elastic for a simple cyclic query for a database with 866,602 tuples.
- TSens has on average 80% - 320% overhead compared to query evaluation for different queries. It is 2 - 60 times slower than Elastic, but returns a local sensitivity value that is 6 - 60,000 times smaller on average.
- PrivSQL has more than 99% relative error (almost worse than just returning 0 as the answer) for four of the seven queries. TSensDP answers 8 queries with $\leqslant 8\%$ relative error and the last query with $\leqslant 20\%$ relative error.

### 3.7.1 Setup

*Dataset.* We evaluate our algorithms on synthetic TPC-H datasets [135] and real world Facebook dataset [113].

*TPC-H.* We consider synthetic datasets generated from TPC-H benchmark [135] with the following schema:
Attributes: RegionKey(RK), NationKey(NK), CustKey(CK), OrderKey(OK), SuppKey(SK), PartKey(PK)




(a) TPC-H queries (b) Facebook queries

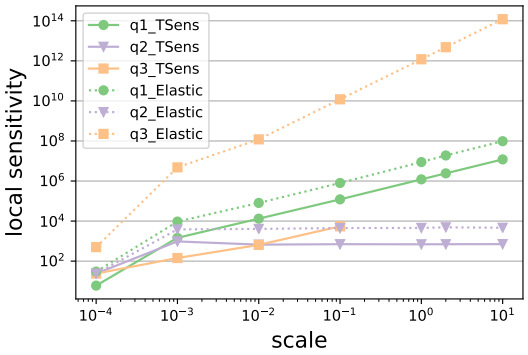
**Figure 3.5**: The join plan for each query.

Relations: Region(R:RK), Nation(N:RK,NK), Customer(C:NK,CK), Orders(O:CK,OK), Supplier(S:NK,SK), Part(P:PK), Partsupp(PS:SK,PK), Lineitem(L:OK,SK,PK).

We evaluate the scalability of our algorithm on TPC-H datasets at different scales $\{0.0001, 0.001, 0.01, 0.1, 1, 2, 10\}$. At scale 1, the sizes of for these relations are 5, 25, 1e4, 1.5e5, 2e5, 8e5, 1.5e6, 6e6 respectively. The same schema and datasets were used to evaluate prior work on differentially private SQL query answering [94, 106].

*Facebook*. We use the Facebook ego-networks from SNAP (Stanford Network Analysis Project) [113]. An ego-network of a user is a set of "social cirles" formed by this user's friends [114]. This dataset consists 10 ego-networks, 4233 circles, 4039 nodes and 88234 edges. We choose the ego-network of user 348 who has 567 circles, 225 nodes and 6384 edges, create edge tables $E_i(x, y)$ for each circle $i$ such that both users of each edge is from the circle $i$ and sort them by table size in descending order. We further create tables $R_1(x, y)$, $R_2(x, y)$, $R_3(x, y)$, $R_4(x, y)$ and insert $E_j$ into $R_i$ if the rank of $E_j$ mod 4 = i. We also create a 3-column table $R_\triangle(x, y, z)$ :

$-R_4(x,y), R_4(y,z), R_4(z,x)$ as a triangle table. All edges are bi-directed.




(a) Local sensitivity reported by TSens and Elastic for queries q1, q2 and q3 on datasets of differing scales

| Relation | TSens | | Elastic |
|---|---|---|---|
| | Most Sensitive Tuple | Tuple Sensitivity | Elastic Sensitivity |
| Region | regionkey(2) | 647 | 120350000 |
| Nation | regionkey(4), nationkey(16) | 179 | 24070000 |
| Supplier | suppkey(51), nationkey(3) | 46 | 51000000 |
| Customer | nationkey(16), custkey(154) | 18 | 11200000 |
| Part | partkey(1311) | 7 | 2550000 |
| Orders | orderkey(57410), custkey(117) | 5 | 350000 |
| Partsupp | partkey(1580), suppkey(81) | 4 | 637500 |
| Lineitem | skip | 1 | 50000 |

(b) Most sensitive tuples and their tuple sensitivities for each relation of q3 when scale = 0.01.

**Figure 3.6**: Local sensitivity reported by TSens versus Elastic for TPC-H queries

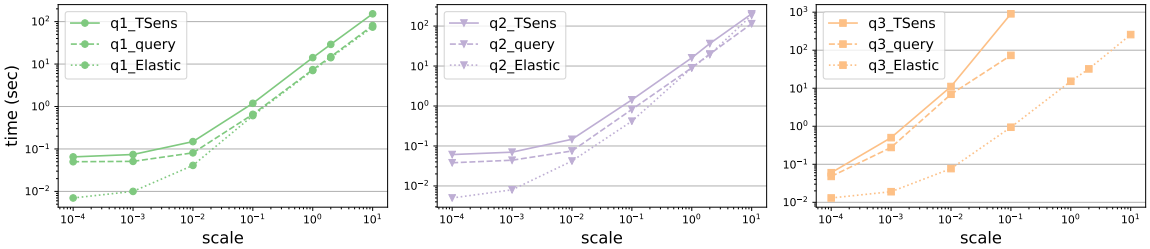



**Figure 3.7**: The trend of time usage in terms of various scales for queries q1, q2 and q3 and algorithms TSens and Elastic for TPC-H queries. The base line 'query' shows the query evaluation time.

*Queries.* We consider 3 TPC-H queries and show their query plan in Figure 3.5a which include a path join query q1, an acyclic join query q2, and a cyclic join query q3. The third query q3 is a cyclic join query that builds a universal table with an extra constraint that the supplier and customer should be from the same nation. We also consider 4 Facebook queries as shown in Figure 3.5b including a triangle query $q_{\triangle}(A,B,C)$, a path join query $q_w(A,B,C,D,E)$, a 4-cycle query $q_{\square}(A,B,C,D)$, and a star join query $q_{\curlywedge}(A,B,C)$. We also show the generalized hypertree decomposition for all non-acyclic queries in the same figure.

We use a machine with 2 processors, 512G SSD and 16G memory to run experiments. Each query is repeated 10 times.

### 3.7.2 Local Sensitivity

**Baseline**. We compare the accuracy and runtime of our TSens algorithm with prior technique Elastic [95] for finding the local sensitivity of a given query. As the original Elastic algorithm requires the maximum frequency of the join attributes to derive the upper bound of the local sensitivity, we first let Elastic pre-process the database to obtain the max frequency for its sensitivity analysis. We also extend Elastic algorithm to support cross-product by assigning the max frequency of empty attributes as the size of the table and to take the join plan as input so that the join order in the experiment is the same. We define the join order as a post-traversal of the join plan.

We also compare the algorithm runtime to the query evaluation time. We apply Yannakakis algorithm to compute the size of query output. For queries that are not acyclic, we first compute the join for each node in the generalized hypertree, and then apply Yannakakis algorithm. The time of running Elastic is also reported. We run each algorithm 10 times to report the average time.

*Result and Analysis.*. Figure 3.6a shows the local sensitivity trend in aspect to the scale for TPC-H. Notice that after scale 0.001, TSens has on average 7x smaller and 6x smaller of the local sensitivity for q1 and q2 than Elastic has. Moreover, for q3, TSens achieves 2,200,000x smaller value for the local sensitivity than Elastic does when scale equals to 0.1. We didn't run q3 for scale larger than 0.1 due to the memory limit issue. The multiplicity tables for this cyclic query grows nearly quadratically with the input table size. Our future work will extend our algorithm to maintain the top $k$ frequent values instead of all the frequencies which can reduce

the intermediate size and further speed up runtime (Section A.2).

Figure 3.6b shows the most sensitive tuple found by TSens for each relation for q3 at scale = 0.01. Unlike TSens, Elastic can only obtain a local sensitivity upper bound, but cannot find the most sensitive tuple. Hence, we report the most sensitive tuple for each relation while also reports its elastic sensitivity by setting this relation as the only sensitive table for Elastic. Each tuple sensitivity found by TSens is below 1,000 while the least elastic sensitivity reported by Elastic is beyond 10,000. We skip finding the most sensitive tuple in LINEITEM since it has the superkey in the query head and thus the tuple sensitivity is at most 1.

Figure 3.7 shows the time cost for both TSens and Elastic for different queries and scales for TPC-H. The second line $q_i$*_query* is the query evaluation time. Notice that we skip computing the multiplicity table of Lineitem in q3 since the tuple sensitivity is at most 1 due to FK-PK joins. For q1 and q2, both TSens and Elastic shows a tight relation to the time of query evaluation, which takes on average 1.8x and 0.9x of the query evaluation time after scale 0.001. For q3, although Elastic is much faster, TSens only takes on average 4.2x of the query evaluation time to find on average 60,000x smaller local sensitivity before scale 1.

We also report the accuracy and the runtime of TSens and Elastic for Facebook queries in Table 3.1. The sensitivity bound improvement ranges from $\times 3$ to $\times 80k$. Although TSens spends $\times 25$ to $\times 60$ more time than Elastic, its runtime is comparable to query evaluation time. The local sensitivity can also be computed by repeating query evaluation over databases which are formed by removing a tuple from active domain or inserting a tuple from representative domain one at a time (using a variation of [171] as discussed in Sections 3.4.1 and 3.5.2). However, the size of the active domain and representative domain is above 10k. This approach will take $\times 10k+$ time than TSens.

**Table 3.1**: Local sensitivity and runtime of 4 query types for TSens and Elastic for Facebook queries. It also reports the query evaluation time for counting the output size. Gray cells have tighter local sensitivities.

| | **Local Sensitivity** | | **Time (seconds)** | | |
|---|---|---|---|---|---|
| | TSens | Elastic | TSens | Elastic | Evaluation |
| $q_{\triangle}$ | 87 | 7524 | 0.405 | 0.007 | 0.431 |
| $q_w$ | 178923 | 511632 | 0.237 | 0.010 | 0.182 |
| $q_{\square}$ | 2014 | 511632 | 0.618 | 0.009 | 0.465 |
| $q_{\curlywedge}$ | 34 | 2723688 | 0.604 | 0.012 | 0.175 |

### 3.7.3 Differential Privacy

*Baseline.* PrivSQL is a differentially private SQL answering engine and it introduces the concept of policy that given primary private relations, the sensitivity of other related relations should be updated to be non-zero according to the database key constraints. For TPC-H datasets, we consider CUSTOMER is the primary private relation for q1 and q3, and SUPPLIER is the primary private relation for q2, so the sensitivity of ORDERS is affected by CUSTOMER and the sensitivity of PARTSUPP is affected by SUPPLIER. The sensitivity of LINEITEM is affected by either of them. For Facebook dataset, we consider $R_2$ is the primary private relation.

PrivSQL uses the maximum frequency as the truncation threshold, which is different from using tuple sensitivity to truncate the database. Any tuple whose frequency is beyond the max frequency will be dropped from the database. PrivSQL runs SVT to learn the truncation threshold for each relation; however, the noise scale of SVT depends on the sensitivity of the relation while it is constantly 1 in TSensDP.

Although the privacy budget allocation strategy affects the performance of DP algorithm, we skip exploring this effect and assume PrivSQL and TSensDP divide the privacy budget into two halves, one for the threshold learning and the other for reporting the query result after truncation. We disable the synopsis generation phase

**Table 3.2**: Application to DP: Comparison between TSensDP and PrivSQL for TPC-H and Facebook queries. Time is in seconds. Gray cells achieve lower errors.

| Query | $\lvert Q(D)\rvert$ | Algorithm | Error | Bias | Global Sens. | Time |
|---|---|---|---|---|---|---|
| q1 | 60175 | TSensDP | 3.56% | 3.44% | 119 | 0.693 |
| | | PrivateSQL | 1.34% | 1.02% | 220 | 0.292 |
| q2 | 60175 | TSensDP | 7.71% | 7.62% | 640 | 0.554 |
| | | PrivateSQL | 99.03% | 100.00% | 774 | 0.231 |
| q3 | 2333 | TSensDP | 2.84% | 0.00% | 14 | 23.063 |
| | | PrivateSQL | 1293% | 2.14% | 12375k | 0.546 |
| $q_{\triangle}$ | 30699 | TSensDP | 1.50% | 1.47% | 49 | 0.562 |
| | | PrivateSQL | 19.12% | 0.00% | 6732 | 0.230 |
| $q_w$ | 17555419 | TSensDP | 5.59% | 5.69% | 17440 | 0.843 |
| | | PrivateSQL | 2.25% | 0.00% | 289476 | 10.340 |
| $q_{\square}$ | 142903 | TSensDP | 2.00% | 1.77% | 167 | 0.792 |
| | | PrivateSQL | 100% | 0.00% | 289476 | 2.232 |
| $q_{\curlywedge}$ | 786 | TSensDP | 19.02% | 16.16% | 13 | 0.670 |
| | | PrivateSQL | 30K% | 0.00% | 2437K | 0.290 |

of PrivSQL and just use Laplace mechanism to answer the SQL query directly.

*Result and Analysis.*. Table 3.2 shows the statistics of releasing differential private query results by TSensDP or PrivSQL for TPC-H and Facebook datasets. Output below 0 is truncated to 0. We report the median of global sensitivity, the median of relative absolute bias, the median of relative absolute error and the average time for each query over 20 runs. We assume the table size is given. For TPC-H, we assume the maximum tuple sensitivity of q1 is 100, of q2 is 500 and of q3 is 10. TSensDP has $\leqslant 4\%$ error for $q_1$ and $q_3$, and $\leqslant 8\%$ error for $q_2$. In contrast, PrivSQL has more than 99% error on $q_2$ and $q_3$. This means that the error in PrivSQL answers for these queries is worse than returning 0 as the answer without looking at the data. The reasons for the poor error are different. In $q_2$ PrivSQL truncates too much of the data, while in $q_3$ it estimate a very loose bound on sensitivity.

For Facebook dataset, we assume the maximum tuple sensitivity of $q_{\triangle}$ is 70, of $q_w$ is 25k, of $q_{\square}$ is 200 and of $q_{\curlywedge}$ is 15. TSensDP achieves $< 6\%$ error for $q_{\triangle}$, $q_w$, $q_{\square}$, while PrivSQL get $> 100\%$ error for $q_{\square}$ and $q_{\curlywedge}$. Since there is no FK-PK join for Facebook queries, we have only one primary private table, which means no table truncation and thus has 0 bias in PrivSQL. However, PrivSQL has $\times 10$ to $\times 180k$ larger global sensitivity than TSensDP, which dominates the error.

*Parameter Analysis..* To find how the upper bound parameter for tuple sensitivity $\ell$ affects the performance, we vary $\ell$ through 1, 10, 30, 50, 100, 1000 and repeat TSensDP 20 times for the star query $q_{\curlywedge}(A, B, C)$ whose true local sensitivity is 13 when $R_2$ is the primary private relation for DP. For each bound, the median global sensitivity, which is also the tuple sensitivity threshold learned from the SVT routine, is [11, 13, 9, 4, 48, 160], the median bias error is [3%, 1%, 13%, 55%, 0%, 0%], and the median relative error is [5%, 4%, 17%, 56%, 32%, 98%]. The optimal $\ell$ in this case is 10, with the corresponding error as 4%, while the worst error is 98% when $\ell = 1000$.

Notice that as $\ell$ increases, the noise added to $\hat{Q}$ in the SVT routine gets larger. This causes the learned tuple sensitivity threshold to deviate more from the local sensitivity, which is considered the optimal threshold by the rule of thumb. When $\ell$ is too small, the learned tuple sensitivity threshold could also be small, which increases the bias.

## 3.8 Related Work

Sensitivity analysis for SQL queries is important to the design of differentially private algorithms. The focus of existing work [134, 108, 14, 71, 106, 127] is to compute the global sensitivities of SQL queries or their upper bounds. The earliest work by McSherry along this line [127] applies static analysis on a given relational algebra and

then combines the sensitivities of the operators in the relational algebra to obtain the maximum possible change to the query output for all possible database instances. This analysis is independent of the database instance, so the result can be much larger than the local sensitivity. In particular, for join operator, the global sensitivity can be unbounded. The analysis either considers a restricted form of join [127, 71] or constrained database instances [134, 108, 14]. For general join queries on unconstrained databases, Lipschitz extension [100, 25, 40, 106] is usually applied to transform the original query $Q$ that has an unbounded sensitivity into a different query $Q'$ that (a) has abounded global sensitivity and (b) has a similar answer as $Q$. In particular, the transformed query in PrivateSQL [106] require to truncates the sensitive tuples. Hence, our work offers efficient ways to identify the most sensitive tuples to complement PrivateSQL to achieve differential privacy.

Smooth sensitivity [133, 94] is another important sensitivity notion for achieving differential privacy. This sensitivity is a smooth upper bound of the local sensitivity of databases at a distance from the given database instance. This requires the computation of local sensitivity of exponentially number of database instances. For SQL queries, elastic sensitivity [94] provides efficient static analysis rule to estimate the upper bound of local sensitivity, but this bound can be still very loose. For example, even if the local sensitivity for a query with selection operator is small, the elastic sensitivity algorithm will output the same value as for a query without the selection operators. In addition, the computation of elastic sensitivity requires additional constraint cardinality information of the given database instance.

Smooth sensitivity [133, 173] or Lipschitz extension [40, 100, 25] have been mainly applied to release graph statistics. However, these algorithms either require customized analysis for each new query [173] or suffer from high performance cost [133, 40, 100, 25]. We will extend our study to graph queries (involving self-joins)

in the future.

Sensitivity analysis has also been studied for non-SQL functions [80, 139, 36], with a focus on global sensitivity. Related topics also include sensitivity analysis for probabilistic queries [97] and finding responsibility of tuples [128], where the goal is different from ours. Prior work on provenance for queries and deletion propagation (e.g., [11, 31]) provide analysis for a rich set of queries and explanations for query results, but the analysis is mainly for removing a tuple in the database (downward tuple sensitivity). Our work also considers upward tuple sensitivity which involves adding new tuples from the domain. Our future study will consider general aggregates and functions.

## 3.9 Conclusions

We studied the local sensitivity problem for counting queries with joins – an important task for many applications like differentially private query answering and query explanations. We showed that the problem is NP-hard in combined query and data complexity even for full conjunctive queries that have an acyclic structure – queries for which the combined complexity of query answering is PTIME. We develop algorithms for full acyclic join queries using join trees, that run in linear in the number of relations and near linear in the number of tuples for interesting sub-classes of acyclic queries including path queries and "doubly acyclic queries", and in PTIME in combined complexity when the maximum degree in the join tree is bounded. Our algorithms can be extended to handle related queries that include selection predicates as well as non-acyclic queries with a certain property on generalized hypertree decompositions. The local sensitivity output by our algorithms is shown to be orders of magnitude tighter than prior work. Our algorithm can also be used to construct differentially private query answering methods that are more accurate than the state

of the art. Extending the framework to handle general non-acyclic queries involving self-joins, projections, negations, and other aggregate functions would be an interesting direction of future work.

# Chapter 4

# Explaining Query Results Under Differential Privacy

## 4.1 Introduction

One major challenge about differential privacy that remains open is the ability to provide explanations for query answers. For non-DP databases, multiple works have proposed approaches for explaining query answers throughout the years, including provenance [51, 91], Shapley values [117, 140], and causal responsibility [128, 129]. However, no work thus far has incorporated DP into explanations. When attempting to solve this problem, one must tackle two distinct challenges that are specific to DP: (1) the answers shown to users are distorted due to the noise added to them, so it is unclear how to separate the contribution of the noise from the data, and (2) even after removing the effect of noise, there is a need to provide data-based explanations for the query answer without violating DP.

In this work, we propose DPXPlain a three-phase framework that generates explanations for query results while ensuring DP, which surmounts the two aforementioned challenges. We next illustrate DPXPlain through an example.

### DPXPlain Through Example

**Example 4.1.1.** *Consider the* Adult *(a subset of census) dataset [61] with 48,842 tuples. We consider the following attributes:* age, workclass, education, marital-status, occupation, relationship, race, sex, native-country, and high-income*, where* high-income *is a binary attribute indicating whether the income of a person is above*

| marital-status | occupation | ... | education | high-income |
|---|---|---|---|---|
| Never-married | Machine-op-inspct | ... | 11th | 0 |
| Married-civ-spouse | Farming-fishing | ... | HS-grad | 0 |
| Married-civ-spouse | Protective-serv | ... | Assoc-acdm | 1 |
| Married-civ-spouse | Machine-op-inspct | ... | Some-college | 1 |
| ... | ... | ... | ... | ... |

**Figure 4.1**: Example of the Adult dataset.

*50K or not (changed to 1/0 from* true/false *for easy aggregation); some relevant columns are illustrated in Figure 4.1.*

We now demonstrate the DPXPlain framework using a concrete query and user question. In the *first phase (Phase-1)*, the user submits a query and gets the results as shown in Figure 4.2. This query is asking the fraction of people with high income in each marital-status group. As Figure 4.2 shows, the framework returns the answer with two columns: group and Priv-answer. Here group corresponds to the group-by attribute marital-status. However, since the data is private, instead of seeing the actual aggregate values avg-high-income, the user sees a perturbed answer Priv-answer for each group as output by some differentially private mechanism with a given privacy budget (here computed by the Gaussian mechanism with privacy budget $\rho = 0.1$ [30]). The third column True-answershown in Figure 4.2 corresponds to the *true aggregated output* for each group; it is hidden from the user and is shown here only for illustration.

In the *second phase (Phase-2)*, the user selects two groups to compare their aggregate values and asks for explanations. However, unlike standard explanation frameworks [167, 144, 115, 74] where the answers of a query are correct and hence the question asked by the user is also correct, in the private setting, the answers that the users see are perturbed. Therefore, the user question and the direction of comparison may not be valid. Hence, instead of getting an explanation to their question, the users get an error bound on the validity of question based on how the noise is added

**Question-Phase-1:**
SELECT marital-status, AVG(high-income) as avg-high-income
FROM R
GROUP BY marital-status

**Answer-Phase-1:**

| **group** marital-status | **Priv-answer** avg-high-income | **True-answer** (hidden) |
|---|---|---|
| Never-married | 0.045511 | 0.045480 |
| Separated | 0.064712 | 0.064706 |
| Widowed | 0.082854 | 0.084321 |
| Married-spouse-absent | 0.089988 | 0.092357 |
| Divorced | 0.101578 | 0.101161 |
| Married-AF-spouse | 0.463193 | 0.378378 |
| Married-civ-spouse | 0.446021 | 0.446133 |

**Figure 4.2**: Phase-1 of DPXPlain: Run a query and receive noisy answers by DP. True-answers are not visible to the user.

to the query answers. Below we illustrate one such comparison:

**Question-Phase-2:**
Why avg-high-income of group **Married-AF-spouse** > that of group **Married-civ-spouse**?

**Answer-Phase-2:**
The 95% confidence interval of group difference $(-0.259, 0.460)$ indicates that the noise in the query is possibly the reason.

**Figure 4.3**: Phase-2 of DPXPlain (Comparison-A)

Although we cannot judge the validity from the question itself, the confidence interval shown in Figure 4.3 indicates a range of true difference between two groups that are highly possibly true, based on which the user can conclude that the added noise might have possibly changed the direction of comparison from the ground truth since this range includes numbers on the negative axis. This is actually in line with the truth: from the hidden True-answercolumn in Figure 4.2, the group Married-AF-spouse has smaller avg-high-income than the group Married-civ-spouse, which is totally the opposite from the question. *Since the validity of the user question is*

*uncertain, we know that any further explanation might not be meaningful and the framework stops here.* In other words, the explanation of the comparison is primarily attributed to the added noise by the DP mechanism.

Now we consider the comparison between other two points shown in Figure 4.4 below. In this case, the confidence interval about the difference between two groups does not include zero and is tight around a positive number 0.4, which indicates that the direction between two groups as the question is asking is correct with high probability. Since the question is valid, the user may continue with the next phase for Comparison-B.

**Question-Phase-2:**
Why avg-high-income of group **Married-civ-spouse** $>$ that of group **Never-married**?

**Answer-Phase-2:**
The 95% confidence interval of group difference $(0.399, 0.402)$ indicates that the noise in the query is possibly not the reason.

**Figure 4.4**: Phase-2 of DPXPlain (Comparison-B)

In the *third phase (Phase-3)*, for Comparison-B, the user wants to know more explanations about the question from the data. To achieve this again with differential privacy, our framework reports an "*Explanation Table*" to the user as Figure 4.5 shows, which includes the top-5 *explanation predicates*. In the simplest form, explanation predicates are singleton predicates of the form "attribute = ¡value¿", while in general, our framework supports more complex predicates involving conjunction, disjunction, and comparison ($>, \geqslant$ etc.) . In Figure 4.5, top-5 explanation predicates are shown out of 103 singleton predicates, according to their (noisy) influences to the question that we discuss later. Once the top-5 predicates are selected, the Explanation Table also shows both their *relative influence* (how much they affect the difference of noisy aggregates of the groups in the question) and their *ranks* in the

form of confidence interval (upper and lower bounds) to preserve differential privacy. The total privacy loss in this phase is $\rho = 2$ (which includes the mechanisms for privately selecting top-k explanation predicates, privately computing confidence interval of influence and privately computing confidence interval of rank).

**Phase-3:**

| explanation predicate | Rel Influ 95%-CI | | Rank 95%-CI | |
|---|---|---|---|---|
| | L | U | L | U |
| occupation = "Exec-managerial" | 3.33% | 8.28% | 1 | 5 |
| education = "Bachelors" | 2.26% | 7.21% | 1 | 9 |
| age = "(40, 50]" | 1.99% | 6.93% | 1 | 8 |
| occupation = "Prof-specialty" | 0.68% | 5.63% | 1 | 18 |
| race = "White" | -2.67% | 2.27% | 1 | 99 |

**Figure 4.5**: Phase-3 of DPXPlain: Receive further explanations from data for Comparison-B (Figure 4.4) that passed Phase-2.

From this table, occupation = Exec-managerial is returned as the top explanation predicate to explain why the average high-income rate in the group Married-civ-spouse is higher than that of the group Never-married. The 95%-confidence interval of its rank is narrowed down to [1, 5], so we know this is highly likely to be among the true top ones. This explanation predicate can be interpreted in this way:
*If we exclude the people whose occupation is* Exec-managerial*, it is less likely that* Married-civ-spouse *has higher* avg-high-income *than* Never-married*.*
In other words, managers tend to earn more if they are married than those who are single, which probably can be attributed to the intuition that married people might be older and have more seniority.

We summarize our framework in Figure 4.6.

## Our contributions

Our main contributions can be summarized as follows.

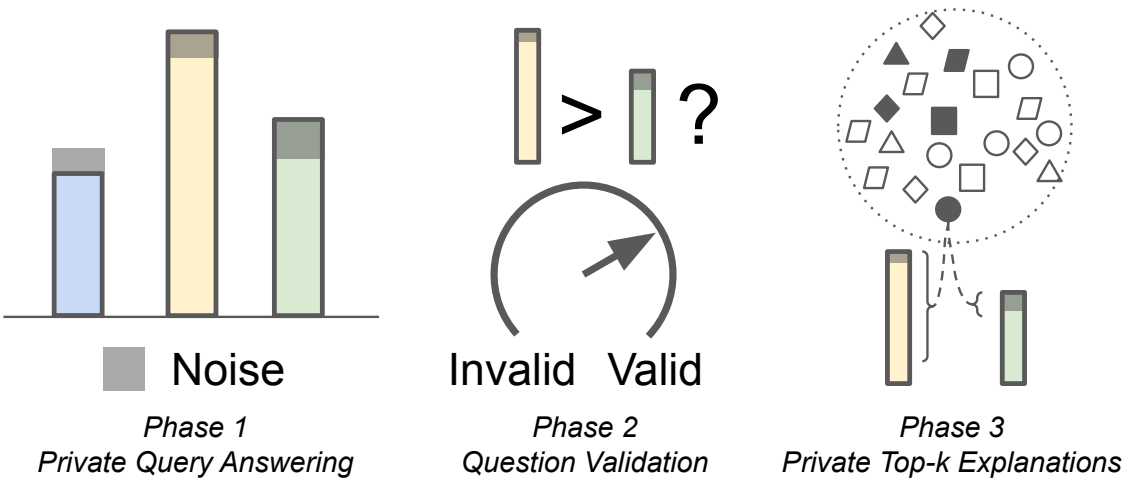



**Figure 4.6**: DPXPLAIN framework.

- We propose a first-of-its-kind framework DPXPLAIN to enable SQL query explanation under differential privacy. It explains a user question over a group-by aggregate query answer (COUNT, SUM, or AVG) under DP with three phases: private query answering, question validation and private explanation table. This approach is inspired by previous work on explanations for aggregate queries [144, 166], but adapted to satisfy DP.

- We develop multiple novel techniques to allow DPXPLAIN to provide explanations under differential privacy:

  - We design a low-sensitive influence function adapted from previous work on non-private explanations [166], which allows it possible to find top-k explanations accurately even under differential privacy.

  - We devise a binary search based algorithm to find the confidence interval of rank, which overcomes the high sensitivity challenge of the rank function.

- We have implemented a prototype of DPXPLAIN, demonstrated two extra use cases based on two real-world datasets, and preformed a comprehensive experimental evaluation that shows DPXPLAIN can correctly suggests the validity of

the question with 100% accuracy for 8 out of 10 questions, select at least 80% of the true top-5 explanation predicates correctly for 8 out of 10 questions with precise descriptions about their influences and ranks.

## 4.2 Preliminaries

We now give the necessary background for our model.

*Databases and Queries.* The DPXPlain framework supports single-block SELECT - FROM - WHERE - GROUP BY queries with aggregates on single tables[1]. Hence the database schema $\mathbb{A} = (A_1, \ldots, A_m)$ is a vector of attributes of a single relational table. Each attribute $A_i$ is associated with a domain $\texttt{dom}(A_i)$. We consider both continuous and categorical domains. A database (instance) $D$ over a schema $\mathbb{A}$ is a bag of tuples (duplicate tuples are allowed) $t_i = (a_1, \ldots, a_m)$, where $a_i \in \texttt{dom}(A_i)$ for all $i$. The domain of a tuple is denoted as $\texttt{dom}(\mathbb{A}) = \texttt{dom}(A_1) \times \texttt{dom}(A_2) \times \ldots \times \texttt{dom}(A_m)$. For an attribute $A_i$, we denote $A_i^{max} = \max\{|a| \mid a \in \texttt{dom}(A_i)\}$ as the maximum absolute value of $A_i$. The value of the attribute $A_i$ of tuple $t$ is denoted by $t.A_i$.

$q$ = SELECT $A_{gb}$, agg($A_{agg}$) FROM D WHERE $\phi$ GROUP BY $A_{gb}$;

**Figure 4.7**: Group-by query with aggregates supported by DPXPlain. The true results are denoted by $(\alpha_i, o_i)$ and the noisy results released by a DP mechanism are denoted by $(\alpha_i, \hat{o}_i)$ where $\alpha_i$ is the value of the group-by attribute $A_{gb}$ and $o_i, \hat{o}_i$ are aggregate values.

In this paper, we consider group-by aggregate queries $q$ of the form shown in Figure 4.7. Here $A_{gb}$ is the group-by attribute and $A_{agg}$ is the aggregate attribute, $\phi$ is a predicate without subqueries, and agg $\in \{COUNT, SUM, AVG\}$ is the aggregate

[1]Unlike some standard explanation framework [166], in differentially private settings, we cannot consider materialization of join-result for multiple tables, since the privacy guarantee depends on *sensitivity*, and removing one tuple from a table may change the join and query result significantly. We leave it as an interesting future work.

function. When query $q$ is evaluated on database $D$, its result is a set of tuples $(\alpha_i, o_i)$, where $\alpha_i \in \texttt{dom}(A_{gb})$ and $o_i = agg(\{t.A_{agg} \mid t \in D, \phi(t) = true, t.A_{gb} = \alpha_i\})$. For brevity, we will use $\phi'(D)$ to denote $\{t \mid \phi'(t) = true\}$ for any predicate $\phi'$, and $agg(A_{agg}, D')$, or simply $agg(D')$ when it is clear from context, to denote $agg(\{t.A_{agg} \mid t \in D'\})$ for any $D' \subseteq D$. Hence, $o_i = agg(A_{agg}, g_i(D))$, where $g_i = \phi \wedge (A_{gb} = \alpha_i)$.

**Example 4.2.1.** *Consider Example 4.1.1. The schema is* $\mathbb{A}$ = (marital-status, occupation, age, relationship, race, workclass, sex, native-country, education, high-income). *All the attributes are categorical attributes and the domain of* high-income *is* $\{0, 1\}$. *The query is shown in Figure 4.2 and the true results for each group is shown in the True-answer column. Here* $A_{gb} = \texttt{marital} - \texttt{status}$, $A_{agg} = \texttt{high} - \texttt{income}$, *and* agg = $AVG$.

*Differential Privacy.* In this work, we consider query-answering and providing explanations using *differential privacy (DP)* [69] to protect private information in the data. In standard databases, a query result can give an adversary the option to find the presence or absence of an individual in the database, compromising their privacy. DP allows users to query the database without compromising the privacy by guaranteeing that the query result will not change too much (defined in the sequel) even if it is evaluated on any two different but *neighboring* databases defined below.

**Definition 4.2.2** (Neighboring Database)**.** *Two databases* $D$ *and* $D'$ *are neighboring if* $D'$ *can be transformed from* $D$ *by adding or removing* [2] *a tuple in* $D$. *We use* $D \approx D'$ *to denote that* $D$ *and* $D'$ *are neighboring databases.*

[2]There are two variants of how neighboring databases are defined. The definition by addition/deletion of tuples is categorized as "unbounded DP", and by updating tuples is categorized as "bounded DP", since the size of data is fixed in this case. In this work, we assume the unbounded version, while our framework can be adapted also for the bounded version by adapting the noise scale.

To achieve DP, it is necessary to randomize the query result such that given any two neighboring databases, it is highly possible that the answers are the same. Informally, the more similar the two random distributions are, the harder it is to distinguish which database is the actual database, therefore the privacy is better protected.

In this paper, we consider a relaxation of DP called *$\rho$-zero-concentrated differential privacy (zCDP)* [30, 70] for several reasons. First, we use Gaussian noise to perturb query answers and derive confidence intervals, which does not satisfy pure $\epsilon$-DP [69] but satisfies approximate $(\epsilon, \delta)$-DP [69] and $\rho$-zCDP. Second, $\rho$-zCDP only has one parameter $\rho$, comparing to $(\epsilon, \delta)$-DP which has two parameters, so it is easier to understand and control. Third, $\rho$-zCDP allows for tighter analyses for tracking the privacy loss over multiple private releases, which is the case for this framework. The parameter $\rho$ is also called the *privacy budget* of the mechanism. A lower $\rho$ value implies a lower privacy loss.

**Definition 4.2.3** (Zero-Concentrated Differential Privacy (zCDP) [30])**.** *A mechanism $\mathcal{M}$ is said to be* $\rho$-zero-concentrated differential private*, or $\rho$-zCDP for short, if for any neighboring datasets $D$ and $D'$ and all $\alpha \in (1, \infty)$ it holds that*

$$D_\alpha(\mathcal{M}(D) \| \mathcal{M}(D')) \leqslant \rho\alpha$$

*where $D_\alpha(\mathcal{M}(D) \| \mathcal{M}(D'))$ denotes the Rényi divergence of the distribution $\mathcal{M}(D)$ from the distribution $\mathcal{M}(D')$ at order $\alpha$ [131].*

*Unless otherwise stated, from now on, we will refer to zero-concentrated differential privacy simply as DP.*

A popular approach for providing zCDP to a query result is to add Gaussian noise to the result before releasing it to user. This approach is called *Gaussian mechanism* [69, 30].

**Definition 4.2.4** (Gaussian Mechanism)**.** *Given a query $q$ and a noise scale $\sigma$, Gaussian mechanism $\mathcal{M}^G$ is given as:*

$$\mathcal{M}^G(D; q, \sigma) = q(D) + N(0, \sigma^2)$$

*where $N(0, \sigma^2)$ is a random variable from a normal distribution*[3] *with mean zero and variance $\sigma^2$.*

**Example 4.2.5.** *Suppose there is a database $D$ with 100 tuples. Consider a query $q$ = "*SELECT COUNT(*) FROM R*", which counts the total number of tuples in a database $D$. Here $q(D) = 100$. Now we use Gaussian mechanism to release $q(D)$, which is to randomly sample a noise $z$ from distribution $N(0, \sigma^2)$. Here we assume $\sigma = 1$. Finally, we got a noisy result $\hat{q}(D) = 102.32$, which we may round to an integer in postprocessing without sacrificing the privacy guarantee (Proposition 4.2.9 below).*

The privacy guarantee from the Gaussian mechanism depends on both the noise scale it uses and the sensitivity of the query. Query sensitivity reflects how sensitive the query is to the change of the input. More noise is needed for a more sensitive query to achieve the same level of privacy protection.

**Definition 4.2.6** (Sensitivity)**.** *Given a scalar query $q$ that outputs a single number, its sensitivity is defined as:*

$$\Delta_q = \sup_{D \approx D'} |q(D) - q(D')|$$

**Example 4.2.7.** *Continuing Example 4.2.5, since the query $q$ returns the database size, for any two neighboring databases, their sizes always differ by 1, so the sensitivity of $q$ is 1.*

[3]The probability density function of a normal distribution $N(\mu, \sigma^2)$ is given as $exp(-((x - \mu)/(\sigma\sqrt{2}))^2/2)/(\sigma\sqrt{2\pi})$.

The next theorem provides the bound on DP guaranteed by a Gaussian mechanism.

**Theorem 4.2.8** (Gaussian Mechanism [30])**.** *Given a query $q$ with sensitivity $\Delta_q$ and a noise scale $\sigma$, its Gaussian mechanism $\mathcal{M}^G$ satisfies $(\Delta_q^2/2\sigma^2)$-zCDP. Equivalently, given a privacy budget $\rho$, choosing $\sigma = \Delta_q/\sqrt{2\rho}$ in Gaussian mechanism satisfies $\rho$-zCDP.*

*Composition Rules.* In our analysis, we will use the following standard composition rules and other known results from the literature of DP [127] (in particular, zCDP [30]) frequently:

**Proposition 4.2.9.** *The following holds for zCDP [127, 30]:*

- **Parallel composition:** *if two mechanisms take disjoint data as input, the total privacy loss is the maximum privacy loss from each.*
- **Sequential composition:** *if we run two mechanisms in a sequence on overlapping inputs, the total privacy loss is the sum of each privacy loss.*
- **Postprocessing:** *if we run a mechanism and postprocess the result without accessing the data, the total privacy loss is only the privacy loss from the mechanism.*

*Private Query Answering.* Recall that we have group-by aggregation query of the form $q$ = SELECT $A_{gb}$, agg($A_{agg}$) FROM R WHERE $\phi$ GROUP BY $A_{gb}$, and it returns a list of tuples $(\alpha_i, o_i)$ where $\alpha_i \in \texttt{dom}(A_{gb})$ and $o_i$ is the corresponding aggregate value. Since no single tuple can exist in more than one group, adding or removing a single tuple can at most change the result of a single group. As mentioned earlier, Phase-1 returns noisy aggregate values $\hat{o}_i$ for each $\alpha_i$ instead of $o_i$. The following holds:

**Observation 4.2.1.** *According to the* parallel composition rule *(Proposition 4.2.9), if for each $\alpha_i$, its (noisy) aggregate value $\hat{o}_i$ is released under $\rho_q$-zCDP, the entire release of results including all groups $\{\alpha_i, \hat{o}_i \ : \ \alpha_i \in \texttt{dom}(A_{gb})\}$ satisfies $\rho_q$-zCDP.*

For a $COUNT$ or $SUM$ query, we use the Gaussian mechanism for each group $\alpha_i$: $\hat{o}_i = o_i + N(0, \sigma^2)$, where the noise scale $\sigma = \Delta_q/\sqrt{2\rho_q}$ to satisfy $\rho_q$-zCDP by Theorem 4.2.8. The sensitivity term $\Delta_q$ is 1 for $COUNT$ and $A_{agg}^{max}$ for $SUM$, the maximum absolute value of the aggregation attribute in its domain. For an $AVG$ query, since $AVG = SUM/COUNT$, we decompose the query into a $SUM$ query and a $COUNT$ query, privately answer each of them by half of the privacy budget $\rho_q/2$, get $\hat{o}_i^S$ and $\hat{o}_i^C$ for each group $\alpha_i$ for the $SUM$ query and $COUNT$ query separately [4], and release $\hat{o}_i = \hat{o}_i^S/\hat{o}_i^C$ as a post-processing step. The noisy query answers of the group-by query with AVG satisfies $\rho_q$-zCDP by the sequential composition rule (Proposition 4.2.9), since each of $SUM$ and $COUNT$ queries satisfies $\rho_q/2$-zCDP.

*Confidence Level and Interval.* Confidence intervals are commonly used to determine the error margin in uncertain computation and are used in various fields from estimating the error in predictions by Machine Learning models [93] to providing query results with added noise due to DP [78]. In our context, we use confidence intervals to estimate the influence of each explanation (in the form of a predicate as we detail in the next section).

**Definition 4.2.10** (Confidence Level and Interval [163])**.** *Given a confidence level $\gamma$ and an unknown but fixed parameter $\theta$, a random interval $\mathcal{I} = (\mathcal{I}^L, \mathcal{I}^U)$ is said to be its confidence interval, or CI, with confidence level $\gamma$ if the following holds:*

$$Pr[\mathcal{I}^L \leqslant \theta \leqslant \mathcal{I}^U] \geqslant \gamma$$

Notice that $\theta$ is a fixed quantity and $\mathcal{I}^L, \mathcal{I}^U$ are random variables. One interpre-

[4]The intermediate releases $\hat{o}_i^S$ and $\hat{o}_i^C$ are stored in the framework for computing the confidence interval of question, which will be discussed in Section 4.4.1.

tation of a confidence interval is that with probability at least $\gamma$, a random draw of the pair $(\mathcal{I}^L, \mathcal{I}^U)$ as an interval will contain the unknown parameter $\theta$. Two bounds are sampled together unless they are independent.

**Example 4.2.11.** *Let $\theta = 0$. Suppose with probability 50% we have $I^L = -1$ and $I^U = 1$, and with another probability 50% we have $I^L = 1$ and $I^U = 2$. Therefore, $Pr[\mathcal{I}^L \leqslant \theta \leqslant \mathcal{I}^U] = 50\%$, and we can conclude that the random interval $\mathcal{I} = (\mathcal{I}^L, \mathcal{I}^U)$ is a 50% level confidence interval for $\theta$.*

## 4.3 Private Explanations in DPXPlain

In this section we provide the model for private explanations of query results in DPXPlain, outline the key technical problems addressed by DPXPlain, and highlight the difference from existing database explanation frameworks.

*User Question and Standard Explanation Framework.* In Phase-2 of DPXPlain, given the noisy results of a group-by aggregation query from Phase-1, users can ask questions comparing the aggregate values of two groups[5]:

**Definition 4.3.1** (User Question)**.** *Given a database $D$, a group-by aggregate query $q$ as shown in Figure 4.7, a DP mechanism $\mathcal{M}$, and two noisy answer tuples $(\alpha_i, \hat{o}_i), (\alpha_j, \hat{o}_j) \in \mathcal{M}(D; q)$ where $\hat{o}_i > \hat{o}_j$, a* user question *has the form* "why is the (noisy) aggregate value $\hat{o}_i$ of group $\alpha_i$ larger than the aggregate value $\hat{o}_j$ of group $\alpha_j$?"*, which is denoted by "why $(\alpha_i, \alpha_j, >)$?".*

**Example 4.3.2.** *The question from Figure 4.4 is denoted as* "why ('Married-civ-spouse', 'Never-married', >)?"*.*

To explain a user question, several previous approaches return predicates that

[5]Our framework can handle more general user questions involving single group or more than two groups; details are deferred to Appendix A.2.5

account for the difference as explanations [167, 144, 74, 115]. We follow this paradigm and define explanation predicates.

**Definition 4.3.3** (Explanation Predicate)**.** *Given a database $D$ with set of attributes $\mathbb{A}$, a group-by aggregation query $q$ (Figure 4.7) with group-by attribute $A_{gb}$ and aggregate attribute $A_{agg}$ and a predicate size $l$, an explanation predicate $p$ is a Boolean expression of the form $p = \varphi_1 \wedge ... \wedge \varphi_l$, where each $\varphi_i$ has the form $A_i = a_i$ such that $A_i \in \mathbb{A} \backslash \{A_{gb}, A_{agg}\}$ is an attribute, and $a_i \in \texttt{dom}(A_i)$ is its value.* [6]

We assume that a set of explanation predicates $\mathcal{P}$ is given (e.g., all singleton predicates of the form $A = a_i$). These explanation predicates are ranked by an influence function [166, 144, 143] and the top-k predicates are reported to the user as the explanations for the question.

However, unlike standard explanation framework on aggregate queries [167, 144, 115, 74], when considering DP, the existing framework is not sufficient to support DP and needs to be adapted: (i) the question itself might not be valid due to the noise injected into the queries, (ii) the selection of top-k explanation predicates needs to satisfy DP, which further requires the influence function to have low sensitivity so that the top-k selection is perturbed at a minimal level, and (iii) since the selected explanation predicates are not guaranteed to be the true top-k, it is also necessary to output extra descriptions under DP for each selected explanation predicate about their actual influences and ranks. We detail the adjustments as follows.

*Question Validation with DP.* While the user is asking "why is $\hat{o}_i > \hat{o}_j$?", in reality, it may be the case that the true results satisfy $o_i \leqslant o_j$, i.e., they have opposite relationship than the one observed by the user. This indicates that $\hat{o}_i > \hat{o}_j$ is the result of the noise being added to the results. In this scenario, one option to explain

[6] We assume $\texttt{dom}(A_i)$ is discrete, finite and data-independent. Our framework can also handle inequality $A_i \circ a_i$ where $\circ \in \{>, <, \geqslant, \leqslant, \neq\}$ when the constant $a_i$ is from a finite and data-independent set. Conjunctions can also be replaced with disjunctions.

the user's observation of $\hat{o}_i > \hat{o}_j$ will be releasing the true values (equivalently, the added exact noise values), which will violate DP. Instead, to provide an explanation in such scenarios, we generate a confidence interval for the difference of two (hidden) aggregate values $o_i - o_j$, which can include negative values (discussed in detail in Section 4.4.1). This leads to the first problem we need to solve in the DPXPlain framework:

**Problem 1** (Private Confidence Interval of Question)**.** *Given a dataset $D$, a query $q$, a DP mechanism $\mathcal{M}$, a privacy budget $\rho_q$, a confidence level $\gamma$, and a user question $(\alpha_i, \alpha_j, >)$ on the noisy query answers output by $\mathcal{M}$ satisfying $\rho_q$-zCDP, find a confidence interval (see Definition 4.2.10) for the user question $\mathcal{I}_{uq} = (\mathcal{I}^L_{uq}, \mathcal{I}^H_{uq})$ for $o_i - o_j$ at confidence level $\gamma$ without extra privacy cost.*

In Phase-2, the framework returns a confidence interval of $o_i - o_j$ to the user. If it is on the negative number axis or it includes zero, it is possible that $o_i \leqslant o_j$, and the user's observation of $\hat{o}_i > \hat{o}_j$ is the result of noise added by the DP mechanism. For such user questions, the user is given the option to stop at Phase-2. If the user is satisfied with the confidence interval for the validity of the user question, she can proceed to Phase-3.

*Influence Function.* When considering DP, the order of the explanation predicates is perturbed by the noise we add to the influences according to the sensitivity of the influence function (discussed in detail in Section 4.4.3). To provide useful explanations, this sensitivity needs to be low, which means the influence does not change too much by adding or removing a tuple from the database. For example, a counting query that outputs the database size $n$ has sensitivity 1, since its result can only change by 1 for any neighboring database. Following this concept, we propose the second problem that lies at the core of the DPXPlain framework, which is also critical to the problems defined below.

**Problem 2** (Influence Function with Low Sensitivity)**.** *Find an influence function* $\mathrm{I} : \mathcal{P} \to \mathcal{R}$ *that maps an explanation predicate to a real number, and has low sensitivity.*

*Private Top-k Explanations.* In DPXPlain, to satisfy DP, in Phase-3 we output the top-$k$ explanation predicates ordered by the noisy influences, and release the influences and ranks of these predicates in the form of confidence intervals to describe the uncertainty. To achieve this goal, we tackle the following three sub-problems.

**Problem 3** (Private Top-$k$ Explanation Predicates)**.** *Given an integer* $k$*, and a privacy parameter* $\rho_{Topk}$*, find the top-k highest influencing predicates* $p_1, p_2, \ldots, p_k$ *from* $\mathcal{P}$ *while satisfying* $\rho_{Topk}$*-zCDP.*

**Problem 4** (Private Confidence Interval of Influence)**.** *Given a confidence level* $\gamma$*,* $k$ *explanation predicates* $p_1, p_2, \ldots, p_k$*, and a privacy parameter* $\rho_{Influ}$*, find a confidence interval* $\mathcal{I}_{influ} = (\mathcal{I}^L_{influ}, \mathcal{I}^U_{influ})$ *for influence* $\mathrm{I}(p_u)$ *at confidence level* $\gamma$ *for each* $u \in \{1, \ldots, k\}$ *satisfying* $\rho_{Influ}$*-zCDP (overall privacy budget).*

**Problem 5** (Private Confidence Interval of Rank)**.** *Given a confidence level* $\gamma$*,* $k$ *explanation predicates* $p_1, p_2, \ldots, p_k$*, and a privacy parameter* $\rho_{Rank}$*, find a confidence interval* $\mathcal{I}_{rank} = (\mathcal{I}^L_{rank}, \mathcal{I}^U_{rank})$ *for rank of* $p_u$ *at confidence level* $\gamma$ *for each* $u \in \{1, \ldots, k\}$ *satisfying* $\rho_{Rank}$*-zCDP (overall privacy budget).*

## 4.4 Computing Explanations Under DP

Next we provide solutions to problems 1, 2, 3, 4, and 5 in Sections 4.4.1, 4.4.2, 4.4.3, 4.4.3, and 4.4.3 respectively, and analyze their properties. We summarize the entire DPXPlain framework in Section 4.4.4.

### 4.4.1 Confidence Interval for a User Question

For **Problem 1**, the goal is to find a confidence interval of $o_i - o_j$ for the user question at the confidence level $\gamma$ without extra privacy cost in Phase-2. We divide the solution into two cases. (1) When the aggregation is COUNT or SUM, the noisy difference $\hat{o}_i - \hat{o}_j$ follows Gaussian distribution, which leads to a natural confidence interval. (2) When the aggregation is AVG, the noisy difference does not follow Gaussian distribution, but we show that the confidence interval in this case can be derived through multiple partial confidence intervals. The solutions below only take the noisy query result as input, which does not incur extra privacy loss according to the post-processing property of DP (Proposition 4.2.9). The pseudo codes can be found in . Appendix A.2.1

*Confidence interval for COUNT and SUM.* For a COUNT or SUM query, recall from Section 4.2 that $\hat{o}_i$ and $\hat{o}_j$ are produced by adding Gaussian noises to $o_i$ and $o_j$ with some noise scale $\sigma$. Therefore, the difference between $\hat{o}_i$ and $\hat{o}_j$ also follows Gaussian distribution with mean $o_i - o_j$ and scale $\sqrt{2}\sigma$ (since the variance is $2\sigma^2$). Following the standard properties of Gaussian distribution, the interval with center $c$ as $\hat{o}_i - \hat{o}_j$ and margin $m$ as $\sqrt{2}(\sqrt{2}\sigma)\,\mathrm{erf}^{-1}(\gamma)$ [7], or (c-m, c+m), is a $\gamma$ level confidence interval of $o_i - o_j$ [163].

*Confidence interval for AVG.* For an AVG query, even the single noisy answer $\hat{o}_i$ does not follow Gaussian distribution, because it is a division between two Gaussian variables as described in Section 4.2: $\hat{o}_i = \hat{o}_i^S / \hat{o}_i^C$. Hence, first we derive partial confidence intervals for $o_i^S$ and $o_i^C$ as discussed above, denoted by $\mathcal{I}^S$ and $\mathcal{I}^C$, individually at some confidence level $\beta$. Let $I^A = \mathcal{I}^S / \mathcal{I}^C := \{x/y \mid x \in \mathcal{I}^S, y \in \mathcal{I}^C\}$ [8] to be the set that includes all possible divisions between any numbers from $\mathcal{I}^S$ and $\mathcal{I}^C$. Especially,

[7] $\mathrm{erf}^{-1}$ is the inverse function of the error function $\mathrm{erf}\, z = (2/\sqrt{\pi}) \int_0^z e^{-t^2} dt$.

[8] In the algorithm, we only need the maximum and the minimum of the set to construct the interval, which can be solved by a numerical optimizer.

if $I^C$ contains zero, we return a trivial confidence interval $(\infty, -\infty)$ that is always valid. Otherwise, $I^A$ is a $2\beta - 1$ level confidence interval for the division, as stated in the following proposition.

**Lemma 4.4.1.** *Given $\mathcal{I}^S$ and $\mathcal{I}^C$ as two $\beta$ level confidence intervals of $o_i^S$ and $o_i^C$ separately, the derived interval $\mathcal{I}^A = \{x/y \mid x \in \mathcal{I}^S, y \in \mathcal{I}^C\}$ is a $2\beta-1$ level confidence interval of $o_i^S/o_i^C$.*

*Proof.* The first inequality below is due to fact that the second event is sufficient for the first event. The next inequality holds by applying the union bound. The full proof is as follows:
$Pr[o_i^S/o_i^C \in \mathcal{I}^A] \geqslant Pr[o_i^S \in \mathcal{I}^S \wedge o_i^C \in \mathcal{I}^C] \geqslant 1 - (Pr[o_i^S \notin \mathcal{I}^S] + Pr[o_i^C \notin \mathcal{I}^C]) \geqslant 1 - ((1 - \beta) + (1 - \beta)) = 2\beta - 1$ □

Furthermore, the difference $\hat{o}_i - \hat{o}_j$ is a subtraction between two ratios of two Gaussian variables, which can be expressed as an arithmetic combination of multiple Gaussian variables: $\hat{o}_i - \hat{o}_j = X_i/Y_i - X_j/Y_j$, where $X_t = N(o_t^S, \sigma_S^2)$ and $Y_t = N(o_t^C, \sigma_C^2)$ for $t \in \{i, j\}$. Similar to Lemma 4.4.1, we can derive the confidence interval for $\hat{o}_i - \hat{o}_j$ based on 4 partial confidence intervals of $o_i^S$, $o_i^C$, $o_j^S$, and $o_j^C$ instead of 2. The confidence level we set for each partial confidence interval is $\beta = 1-(1-\gamma)/4$ by applying union bound on the failure probability $1 - \gamma$ that one of the four variables is outside its interval. After we have 4 partial confidence intervals $\mathcal{I}_i^S$, $\mathcal{I}_i^C$, $\mathcal{I}_j^S$, and $\mathcal{I}_j^C$ for $o_i^S$, $o_i^C$, $o_j^S$, and $o_j^C$ separately, similar to Lemma 4.4.1, we combine them together as $\mathcal{I}^A = \mathcal{I}_i^S / \mathcal{I}_i^C - \mathcal{I}_j^S / \mathcal{I}_j^C$ and derive the confidence interval for $o_i - o_j$ as $(\inf \mathcal{I}^A, \sup \mathcal{I}^A)$, which is guaranteed to be at confidence level $\gamma$. If 0 is included in either $\mathcal{I}_i^C$ or $\mathcal{I}_j^C$, we set the confidence interval to be $(\infty, -\infty)$ instead.

### 4.4.2 Influence Function with Low Sensitivity

For **Problem 2**, the goal is to design an influence function that has low sensitivity. Inspired by PrivBayes [175]. We start by adapting a known influence function to our framework, and then adapt it to have a low sensitivity.

Our influence function of an explanation predicate with respect to a comparison user question is inspired by the Scorpion framework [166], where the user questions seek explanations for outliers in the results of a group-by aggregate query. The Scorpion framework identifies predicates on input data that cause the outliers to disappear from the output results. Given the group-by aggregation query shown in Figure 4.7 and a group $\alpha_i \in \texttt{dom}(A_{gb})$, recall from Section 4.2 that the true aggregate value for $\alpha_i$ is $o_i = agg(A_{agg}, g_i(D))$, where $g_i = \phi \wedge (A_{gb} = \alpha_i)$, i.e., $g_i(D)$ denotes the set of tuples that contribute to the group $\alpha_i$.

Scorpion measures the influence of an explanation predicate $p$ to some group $\alpha_i$ as the ratio between the change of output aggregate value and the change of group size:

$$\frac{agg(g_i(D)) - agg(g_i(\neg p(D)))}{|g_i(p(D))|} \tag{4.1}$$

Here $\neg p(D)$ denotes $D - p(D)$, i.e., the set of tuples in $D$ that do not satisfy the predicate $p$. To adapt this influence function to DPXPlain, the following two differences have to be considered.

1. First, it should measure the influence w.r.t. the comparison from the user question $(\alpha_i, \alpha_j, >)$ instead of a single group. A natural extension is to change the target aggregate on $g_i$ in the numerator in Equation (4.1) to the difference between the aggregate values of two groups $g_i, g_j$ before and after applying the explanation predicate $p$, and change the denominator as the maximum change in $g_i$ or $g_j$ when

$p$ is applied. This leads to the following influence function:

$$\frac{(agg(g_i(D)) - agg(g_j(D))) - (agg(g_i(\neg p(D))) - agg(g_j(\neg p(D))))}{\max(|g_i(p(D))|, |g_j(p(D))|)} \tag{4.2}$$

2. Second and more importantly, in DPXPlain, we need to preserve DP when we use influence function to sort and rank multiple explanation predicates, or to release the influence and rank of an explanation predicate. Therefore, *we need to account for the sensitivity of the influence function*, which is determined by the worst-case change of influence when a tuple is added or removed from the database. If the explanation predicate only selects a small number of tuples, the denominator in the influence function is small and thus changing the denominator in Equation (4.2) by one (when a tuple is added or removed) can result in a big change in the influence as illustrated in the following example, making the adapted influence function in Equation (4.2) unsuitable for DPXPlain.

**Example 4.4.2** (The Issue of the Influence Sensitivity)**.** *Suppose there are two groups $\alpha_i$ and $\alpha_j$ in $D$ with 1000 tuples in each, aggregate function $agg = SUM$ on attribute $A_{agg}$ with domain $[0, 100]$, and the explanation predicate $p$ matches only 1 tuple from the group $\alpha_i$ with $A_{agg} = 100$ and no tuple from $\alpha_j$. Suppose $agg(g_i(D)) = 20,000$, $agg(g_j(D)) = 10,000$, then $agg(g_i(\neg p(D))) = 19,900$ and $agg(g_j(\neg p(D))) = 10,000$. Therefore, from Equation (4.2), the influence of $p$ is $((20,000 - 10,000) - (19,900 - 10,000))/\max\{1, 0\} = 100$ on the original database $D$. However, suppose a new tuple is added again with $A_{agg} = 2$, which also satisfies $p$ and belongs to group $\alpha_i$. Now the influence in Equation (4.2) becomes $((20,002 - 10,000) - (19,900 - 10,000))/\max\{2, 0\} = 102/2 = 51$. Note that while we added a tuple to the database contributing only 2 to the sum, it led to a change of 100-51 = 49 to the influence function because of the small denominator.*

Therefore, we propose a new influence function that is inspired by Equation (4.2)

but has lower sensitivity. Note that the denominator in Scorpion's influence function in Equation (4.2) acts as a normalizing factor, whose purpose is to penalize the explanation predicate that selects too many tuples, e.g., to prohibit removal of the entire database by a dummy predicate. To have a similar normalizing factor with low sensitivity, we multiply the numerator in Equation (4.2) by $\frac{\min(|g_i(\neg p(D))|,|g_j(\neg p(D)|)}{\max(|g_i(D)|,|g_j(D)|)+1}$. From this new normalizing factor, the numerator captures the minimum of the number of tuples that are not removed from each group, and the denominator is a constant, which does not change for different explanation predicates and keeps the normalizing factor in the interval $[0, 1]$. Similar to Scorpion, if $p(D)$ constitutes a large fraction of $D$ (e.g., if $p(D) = D$), then the normalizing factor is small, reducing the value of the influence. Also note that, unlike standard SQL query answering where only non-empty groups are shown in the results, in DP, all groups from the actual domain have to be considered, hence unlike Equation (4.1), $g_i(D), g_j(D)$ could be zero, hence 1 is added in the denominator to avoid division by zero. Specially, when $agg = AVG$, we remove the constant denominator to boost the signal of the influence and keep the sensitivity low, which will be discussed in the sensitivity analysis after Proposition 4.4.4. We formally define the influence as follows.

**Definition 4.4.3** (Influence and Relative Influence of Explanation Predicates)**.** *Given a database $D$, a query $q$ as shown in Figure 4.7, and a user question $(\alpha_i, \alpha_j, >)$, the influence of an explanation predicate $p$ is defined as* $\mathrm{I}(p; (\alpha_i, \alpha_j, >), D)$*, or simply* $\mathrm{I}(p)$ *when it is clear from context:*

$$I(p) = ((agg(g_i(D)) - agg(g_j(D))) - (agg(g_i(\neg p(D))) - agg(g_j(\neg p(D)))))$$
$$\times \begin{cases} \frac{\min(|g_i(\neg p(D))|,|g_j(\neg p(D)|)}{\max(|g_i(D)|,|g_j(D)|)+1} & , agg \in \{COUNT, SUM\} \\ \min(|g_i(\neg p(D))|, |g_j(\neg p(D)|) & , agg = AVG \end{cases} \tag{4.3}$$

*Given noisy answers $\hat{o}_i, \hat{o}_j$ of groups $\alpha_i, \alpha_j$ in the result of $q$ from a DP mechanism $\mathcal{M}$, we use $\tilde{I}(p; (\alpha_i, \alpha_j, >), D, \hat{o}_i, \hat{o}_j)$, or in short $\tilde{I}(p)$ when it is clear from the context,*

*to denote the relative influence:*

$$\tilde{I}(p) = \mathrm{I}(p) \;/\; \begin{cases} |\hat{o}_i - \hat{o}_j| & , agg \in \{COUNT, SUM\} \\ |\hat{o}_i - \hat{o}_j|(|\max(\hat{o}_i^C, \hat{o}_j^C))| + 1) & , agg = AVG \end{cases} \tag{4.4}$$

In DPXPlain, we use the relative influence $\tilde{I}(p)$ only to present the importance of the explanation predicates to the users. Intuitively, it reflects how much the group difference $o_i - o_j$ in the question is changed by the explanation predicate. We only use $\hat{o}_i, \hat{o}_j$ as an approximation since users do not see the true aggregate values $o_i, o_j$ and only see $\hat{o}_i, \hat{o}_j$ that are constants for all explanation predicates given a user question. The true influence value $I(p)$ is used to compute the top-$k$ explanation predicates and their confidence intervals of influence and rank (discussed in Section 4.4.3, 4.4.3 and 4.4.3).

The proposition below guarantees the sensitivity of the influence function. We give an intuitive proof as follows, where the formal proofs can be found at Appendix A.1.1. When $agg = \mathrm{COUNT}$, we combine two group differences $(agg(g_i(D)) - agg(g_j(D))) - (agg(g_i(\neg p(D))) - agg(g_j(\neg p(D))))$ into a single group difference as $agg(g_i(p(D)) - agg(g_j(p(D))$, which is considered as a subtraction between two counting queries. We prove that the sensitivity of a counting query after a multiplication with the normalizing factor will multiply its original sensitivity by 2. Since we have two counting queries, the final sensitivity is 4. When $agg = SUM$, the proof is similar except we need to multiply the final sensitivity by $A_{agg}^{max}$ , the maximum absolute domain value of $A_{agg}$. For AVG, we view it as a summation of 4 AVG queries that times with $\min(|g_i(\neg p(D))|, |g_j(\neg p(D)|)$. Intuitively, we change AVG to SUM, and, therefore, reduce to the case of SUM and bound the sensitivity. This sensitivity now becomes relatively small since we have amplified the influence.

**Proposition 4.4.4.** *[Influence Function Sensitivity] Given an explanation predicate $p$ and an user question with respect to a group-by query with aggregation $agg$, the*

*following holds:*

1. *If* $agg = COUNT$*, the sensitivity of* $\mathrm{I}(p)$ *is 4.*

2. *If* $agg = SUM$*, the sensitivity of* $\mathrm{I}(p)$ *is* $4\ A_{agg}^{max}$.

3. *If* $agg = AVG$*, the sensitivity of* $\mathrm{I}(p)$ *is* $16\ A_{agg}^{max}$.

Intuitively, the sensitivity of $\mathrm{I}(p)$ is low. When $agg = COUNT$, $\mathrm{I}(p)$ is $O(n)$ and $\Delta_{\mathrm{I}}$ is $O(1)$, where $n$ is the size of database. When $agg \in \{SUM, AVG\}$, $\mathrm{I}(p)$ is $O(nA_{agg}^{max})$ and $\Delta_{\mathrm{I}}$ is $O(A_{agg}^{max})$. Therefore, the sensitivity of influence $\Delta_{\mathrm{I}}$ is low comparing to the influence itself. However, as the example below shows, if we define the influence function for $AVG$ the same way as $COUNT$ or $SUM$, both $\mathrm{I}(p)$ and $\Delta_{\mathrm{I}}$ will become $O(A_{agg}^{max})$, which makes the sensitivity (relatively) large.

**Example 4.4.5** (The Issue with $AVG$ Influence.)**.** *Consider an* $AVG$ *group-by query with aggregation on an attribute with domain* $[0, 100]$*, and an explanation predicate* $p$ *such that for group* $\alpha_i$ *we have 2 tuples with* $AVG(g_i(D)) = 100/2 = 50$*,* $AVG(g_i(\neg p(D))) = 0/1 = 0$*, and for group* $\alpha_j$ *we have two tuples with* $AVG(g_j(D)) = 100/2 = 50$ *and* $AVG(g_j(\neg p(D))) = 100/2 = 50$*. Suppose we define the influence function for* $AVG$ *the same way as* $COUNT$ *or* $SUM$*, therefore the influence of* $p$ *in Equation* (4.3) *is* $\mathrm{I}(p) = ((50-50)-(0-50))(\min(1,2)/(\max(2,2)+1) = 50/3$*. However, suppose we remove a tuple and* $g_i(\neg p(D))$ *becomes 0, now the influence in Equation* (4.3) *becomes 0. Note that a single removal of a tuple completely changes the influence to 0, and this change is equal to the influence itself, which is (relatively) large.*

There are two other observations about the influence functions. First, recall that the user question "why $(\alpha_i, \alpha_j, >)$" is asked based on the noisy results $\hat{o}_i > \hat{o}_j$, while the influence function uses the true results, i.e., even if they show the reverse trend

$o_i \leqslant o_j$, still we consider $agg(g_i(D)) - agg(g_j(D))$ in the influence function. Second, $\mathrm{I}(p)$ could be either positive or negative. When it is positive, it means that removing the tuples satisfying $p$ makes the group difference smaller, and therefore it helps explain the question. When it is negative, it has the opposite effect. We find that the influence function is not monotone and will discuss in Appendix A.2.4.

### 4.4.3 Private Top-k Explanations

In this section, we discuss the computation of the top-k explanation predicates and the confidence intervals of influences and ranks.

#### Problem 3: Private Top-k Explanation Predicates

The goal is to find with DP the top-$k$ explanation predicates from a set of explanation predicates $\mathcal{P}$ in terms of their (true) influences $\mathrm{I}(p)$, which is the first step in Phase-3 of DPXPlain (Figure 4.6). Note that simply choosing the *true* top-$k$ explanation predicates in terms of their $\mathrm{I}(p)$ is not differentially private.

In DPXPlain, we adopt the One-shot Top-k mechanism [63, 62] to privately select the top-$k$. It works as follows. For each explanation predicate $p \in \mathcal{P}$, it adds a Gumbel noise [9] to its influence with scale $\sigma = 2\Delta_{\mathrm{I}}\sqrt{k/(8\rho_{Topk})}$, where $\Delta_{\mathrm{I}}$ is the sensitivity of the influence function (discussed in Proposition 4.4.4), reorders all the explanation predicates in a descending order by their noisy influences, and outputs the first $k$ explanation predicates. It satisfies $\rho_{Topk}$-zCDP [63, 59, 33, 141, 62], since it is equivalent to iteratively applying $k$ exponential mechanisms [69], where each satisfies $\epsilon^2/8$-zCDP [63, 59, 33, 141] and $\epsilon = \sqrt{8\rho_{Topk}/k}$ [63, 62]. Therefore, in total it satisfies $(k\epsilon^2/8)$-zCDP by the sequential composition property (Proposition 4.2.9) which is also $\rho_{Topk}$-zCDP. The returned list of top-k predicates is close to that of

[9] For a Gumbel noise $Z \sim Gumbel(\sigma)$, its CDF is $Pr[Z \leqslant z] = \exp(-\exp(-z/\sigma))$.

the true top-k in terms of their influences, of which the proof is based on the utility proposition of exponential mechanism in Theorem 3.11 of [69]. We summarize the properties of this approach in the following proposition and defer the pseudo codes and proof to Appendix A.2.1 and Appendix A.1.2.

**Proposition 4.4.6.** *Given an influence function* I *with sensitivity* $\Delta_{\mathrm{I}}$*, a set of explanation predicates* $\mathcal{P}$*, a privacy parameter* $\rho_{Topk}$ *and a size parameter* $k$*, the following holds:*

1. *One-shot Top-k mechanism finds* $k$ *explanation predicates while satisfying* $\rho_{Topk}$*-zCDP.*

2. *Denote by* $OPT^{(i)}$ *the* $i$*-th highest (true) influence, and by* $\mathcal{M}^{(i)}$ *the* $i$*-th explanation predicate selected by the One-shot Top-k mechanism. For* $\forall t$ *and* $\forall i \in \{1, 2, \ldots, k\}$*, we have*

$$Pr\left[\mathrm{I}(\mathcal{M}^{(i)}) \leqslant OPT^{(i)} - \frac{2\Delta_{\mathrm{I}}}{\sqrt{8\rho_{Topk}/k}}(\ln(|\mathcal{P}|) + t)\right] \leqslant e^{-t} \tag{4.5}$$

**Example 4.4.7.** *Reconsider the user question in Figure 4.4. For this question, we have in total 103 explanation predicates as the set of explanation predicates. The privacy budget* $\rho_{Topk} = 0.05$*, the size parameter* $k = 5$*, and the sensitivity* $\Delta_{\mathrm{I}} = 16$*. For each of the explanation predicate, we add a Gumbel noise with scale* $\sigma = 113$ *to their influences. For example, for the predicates shown in Figure 4.5, their noisy influences are 990, 670, 645, 475, 440, which are the highest 5 among all the noisy influences. The true influences for these five ones are 547, 501, 555, 434, 118. To see how close it is to the true top-5, we compare their true influences with the true highest five influences: 555, 547, 501, 434, 252, which shows the corresponding differences in terms of influence are 8, 46, 54, 0, 134. By Equation* (4.5)*, in theory the probability that such difference is beyond 864 is at most 5%. Finally, we sort explanation predicates*

*by their noisy scores and report the top-k. The order of the explanation predicates will be further sorted by their relative influence and rank confidence intervals, which will be discussed in Section 4.4.4.*

**Problem 4: Private Confidence Interval of Influence**

The goal is to generate a confidence interval of influence $\mathrm{I}(p)$ (Definition 4.4.3) of each explanation predicate $\mathrm{I}(p_1)$, $\mathrm{I}(p_2)$, ..., $\mathrm{I}(p_k)$ from the selected top-k (Section 4.4.3). For each $\mathrm{I}(p_i)$, we apply the Gaussian mechanism (Theorem 4.2.8) with privacy budget $\rho_{Influ}/k$ to release a noisy influence $\hat{\mathrm{I}}_i$ with noise scale $\sigma = \Delta_{\mathrm{I}}/\sqrt{2\rho_{Influ}/k}$. The sensitivity term $\Delta_{\mathrm{I}}$ is determined by Proposition 4.4.4. Following the standard properties of Gaussian distribution, for each $\mathrm{I}(p_i)$, we set the confidence interval by a center $c$ as $\hat{\mathrm{I}}_i$ and a margin $m$ as $\sqrt{2}\sigma\,\mathrm{erf}^{-1}(\gamma)$, or (c-m, c+m), as a $\gamma$ level confidence interval of $\mathrm{I}(p_i)$ [163]. Together, it satisfies $\rho_{Influ}$-zCDP according to the composition property by Proposition 4.2.9. Pseudo codes can be found in Appendix A.2.3.

**Problem 5: Private Confidence Interval of Rank**

The goal is to find the confidence interval of the rank of each explanation predicate from the selected top-k (Section 4.4.3).

Given an explanation predicate $p \in \mathcal{P}$, we denote the rank of $p$ w.r.t. the database $D$ by $\mathrm{rank}(p; D, \mathcal{P}, \mathrm{I})$, where the rank is the natural ordering of the predicates imposed by their (true) influences according to the influence function I. For simplicity, we omit $D, \mathcal{P}, \mathrm{I}$ from the notation, leaving us with $\mathrm{rank}(p)$. Since rank is assumed to be a one-to-one function, we also consider $\mathrm{rank}^{-1}(t; D, \mathcal{P}, \mathrm{I})$, or simply $\mathrm{rank}^{-1}(t)$, for an integer $1 \leqslant t \leqslant |\mathcal{P}|$ that returns the predicate ranked in the the $t$-th place according to I. DPXPlain aims to privately report a confidence interval of $\mathrm{rank}(p)$. For example, $\mathcal{I}_{rank} = [1, |\mathcal{P}|]$ is a trivial confidence interval which has no privacy loss and

always includes the true rank.

Unlike the sensitivity of the influence function, the sensitivity of rank($p$) is high, since adding one tuple could possibly changing the highest influence to be the lowest and vice versa. Fortunately, we can employ a critical observation about rank and influence.

**Proposition 4.4.8.** *Given a database $D$, a set of predicates $\mathcal{P}$, an influence function* I *with global sensitivity $\Delta_{\mathrm{I}}$, and an integer $1 \leqslant t \leqslant |\mathcal{P}|$,* $\mathrm{I}(\mathrm{rank}^{-1}(t))$ *has global sensitivity $\Delta_{\mathrm{I}}$.*

The intuition behind this proof is that, fixing an explanation predicate $p = \mathrm{rank}^{-1}(t)$, for a neighboring database, if its influence is increased, its rank will be moved to the top and push down other explanation predicates with lower influences, so the influence at the original rank of $p$ in the neighboring database is still low.

*Algorithm for rank confidence interval.* Given this observation, we devise an binary-search based strategy to find the confidence interval of rank for the target explanation predicate. Algorithm 4 summarizes this approach. It takes as input explanation predicates $p_1, p_2, \ldots, p_k$, and outputs confidence intervals of the ranks for all the explanation predicates at confidence level $\gamma$ and satisfies $\rho_{Rank}$-zCDP.

In line 1, RankBound takes four parameters: an explanation predicate $p$, a privacy budget $\rho$, a sub confidence level $\beta$ and a direction $dir \in \{-1, +1\}$. It guarantees that it will find a lower ($dir = -1$) or upper ($dir = +1$) bound of rank with confidence $\beta$ for the explanation predicate $p$ using privacy budget $\rho$. In line 2, we set the maximum depth $N$ of the binary search. In line 3, we set the noise scale $\sigma_{-1}$ or $\sigma_{+1}$, which depends on the sensitivity of $\mathrm{I}(p) - \mathrm{I}(\mathrm{rank}^{-1}(t))$ (in line 8), which is $2\Delta_{\mathrm{I}}$; and the number of Gaussian mechanisms used in the binary search, which is $N$. In line 4, we set the margin $\xi_{+1}$ or $\xi_{-1}$, which will be discussed in line 9. In line 5, we initialize the binary search by setting two pointers, $t_{low}$ and $t_{high}$, as the first and last rank.

In lines 6–12 there is a while loop for the binary search. In line 7, we pick a rank that is at the middle of two pointers. In line 8, we add a Gaussian noise with scale $\sigma$ to the difference between the influence of the target explanation predicate $p$ and the influence of the explanation predicate that has rank $t$. From line 9 to 11 we update one of the pointer according to the relationship between the noisy difference and the margin $\xi_{dir}$. If we are trying to find a rank upper bound ($dir = +1$), we want the binary search to find the rank such that the difference (without noise) is above zero. Due to the noise injected, even if the noisy difference is above zero, the true difference could be negative. To secure the goal with high probability, we requires the noisy difference to be above a margin $\xi_{dir}$, as shown in line 9. In this case, we narrow down the search space by moving $t_{high}$ to $\max\{t-1, 1\}$. The strategy is similar when we are looking for a rank lower bound ($dir = -1$).

Now, we describe the usage of the sub-routine RankBound. We repeat the following for each explanation predicate. In line 16, we allocate an even portion from the total privacy budget $\rho_{Rank}$, and set the sub confidence level to $\beta = (\gamma + 1)/2$ so the final confidence interval has confidence level $2\beta - 1 = \gamma$ by the rule of union bound. In lines 17, we divide the privacy budget $\rho$, and make two calls to the sub-routine RankBound to find a rank upper bound and a rank lower bound for the explanation predicate $p_u$, and finally merge them into a single confidence interval. We spend more budget for the rank upper bank since this is more important in the explanation.

**Example 4.4.9.** *Figure 4.8 shows an example of* RankBound *for finding the upper bound of the confidence interval for* $\text{rank}(p)$ *for some explanation predicate* $p$*. Assume* $\text{rank}(p) = 3$*. The upper part of the figure shows the influences of all the explanation predicates in a descending order, and the lower part shows the status of the binary search pointers in each loop. The search contains three loops starting from* $t_{low} = 1$

**Algorithm 4** Compute Confidence Interval of Rank

**Input:** A dataset $D$, a predicate space $\mathcal{P}$, an influence function I with sensitivity $\Delta_{\mathrm{I}}$, $k$ explanation predicates $p_1, p_2, \ldots, p_k$, a confidence level $\gamma$, and a privacy parameter $\rho_{Rank}$.

**Output:** A list of $\gamma$-level confidence intervals of the influence $\mathrm{rank}(p_u; D, \mathcal{P}, \mathrm{I})$ for $u \in \{1, 2, \ldots, k\}$.

**function** RANKBOUND($p, \rho, \beta, dir$)
  $N \leftarrow \lceil \log_2 |\mathcal{P}| \rceil$
  $\sigma_{dir} \leftarrow (2\Delta_{\mathrm{I}})/\sqrt{2(\rho/N)}$
  $\xi_{dir} \leftarrow \sigma_{dir}\sqrt{2\ln(N/(1-\beta))} \times dir$
  $t_{low}, t_{high} \leftarrow 1, |\mathcal{P}|$
  **while** $t_{high} > t_{low}$ **do**
    $t \leftarrow \lfloor \frac{t_{high}+t_{low}}{2} \rfloor$
    $\hat{s} \leftarrow \mathrm{I}(p) - \mathrm{I}(\mathrm{rank}^{-1}(t)) + \mathcal{N}(0, \sigma^2)$
    **if** $\hat{s} \geqslant \xi_{dir}$ **then** $t_{high} \leftarrow \max\{t-1, 1\}$
    **else** $t_{low} \leftarrow \min\{t+1, |\mathcal{P}|\}$
    **end if**
  **end while**
  **return** $t_{high}$
**end function**
**for** $u \leftarrow 1, 2, \ldots, k$ **do**
  $\rho, \beta \leftarrow \rho_{Rank}/k, (\gamma+1)/2$
  $\mathcal{I}_u \leftarrow (\text{RANKBOUND}(p_u, 0.1\rho, \beta, -1), \text{RANKBOUND}(p_u, 0.9\rho, \beta, +1))$
**end for**
**return** $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k$

*and* $t_{high} = 15$. *Within each loop, to illustrate the idea, it is equivalent to adding a Gaussian noise to* $\mathrm{I}(\mathrm{rank}^{-1}(t))$, *which is shown as a blue circle, compare it with* $\mathrm{I}(p) - \xi$, *which is shown as a dashed line, and update the pointers accordingly. For example, the blue arrows in the upper part illustrates how the pointers are updated from iteration 1 to iteration 2. The pointer* $t_{high}$ *is moved from 15 to 7 since the blue circle in iteration 1 is in the green region. Finally it breaks at* $t_{low} = t_{high} = 5$.

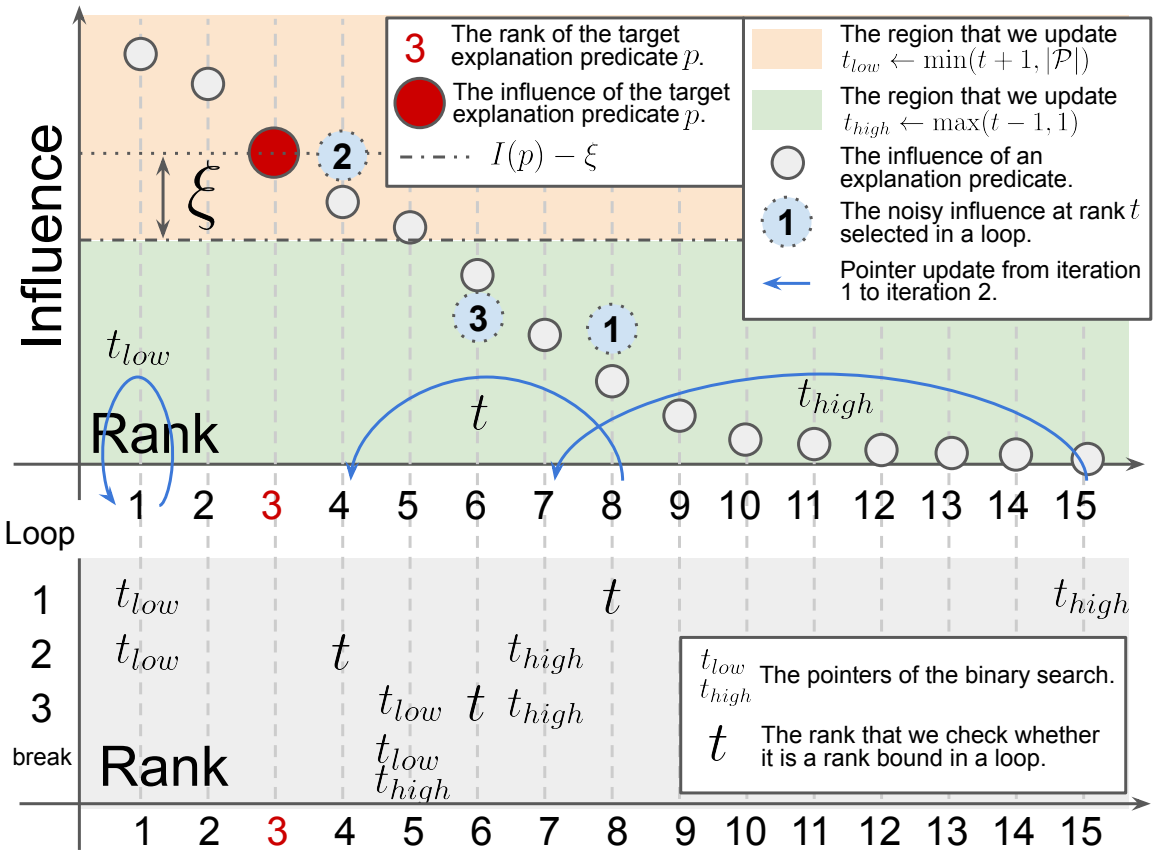



**Figure 4.8**: Illustration of RankBound for finding the upper bound of the confidence interval for $\mathrm{rank}(p)$.

We now show that Algorithm 4 satisfies the privacy requirement, and outputs valid confidence intervals, of which the lower and upper bounds are close to the target explanation predicate in terms of their influences (see Example 4.4.11). In Section 3.7, we show that the interval width is empirically small.

**Theorem 4.4.10.** *Given a database $D$, a predicate space $\mathcal{P}$, an influence function* I *with sensitivity $\Delta_{\mathrm{I}}$, $k$ explanation predicates $p_1, p_2, \ldots, p_k$, a confidence level $\gamma$, and a privacy parameter $\rho_{Rank}$, Algorithm 4 returns a list of confidence intervals $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k$ such that*

1. *Algorithm 4 satisfies $\rho_{Rank}$-zCDP.*
2. *For $\forall u \in \{1, 2, \ldots, k\}$, $\mathcal{I}_u$ is a $\gamma$ level confidence interval of* $\mathrm{rank}(p_u)$.
3. *For $\forall u \in \{1, 2, \ldots, k\}$ and $\forall x \geqslant 0$, the confidence interval $\mathcal{I}_u = (\mathcal{I}_u^L, \mathcal{I}_u^U)$ satisfies $Pr[A \leqslant \mathrm{I}(p_u) \leqslant B] \geqslant 1 - 2e^{-x}$ where $A = \mathrm{I}(\mathrm{rank}^{-1}(\mathcal{I}_u^L)) - (|\xi_{-1}| + \sigma_{-1}\sqrt{2(x + \ln N)})$ and $B = \mathrm{I}(\mathrm{rank}^{-1}(\mathcal{I}_u^U)) + (|\xi_{+1}| + \sigma_{+1}\sqrt{2(x + \ln N)})$.*

**Example 4.4.11** (Rank CI Utility)**.** *Suppose we have 100 explanation predicates with influences 9900, 9800, ..., 0. Fix parameters $\rho_{Topk} = 1.0$, $k = 5$, $\Delta_{\mathrm{I}} = 1$, $\gamma = 0.95$, which gives $N = 7$, $\sigma_{-1} = 26$, $\xi_{-1} = 89$, $\sigma_{+1} = 9$, $\xi_{+1} = 30$. Now we focus on estimating the possible rank confidence interval of $p_3$. Assume the true rank of $p_3$ is 5 and thus $\mathrm{I}(p_3) = 9500$. According to Item 3 from Theorem 4.4.10, we have $Pr[\mathrm{I}(\mathrm{rank}^{-1}(\mathcal{I}_3^L)) - 200 \leqslant \mathrm{I}(p_3) \leqslant \mathrm{I}(\mathrm{rank}^{-1}(\mathcal{I}_3^U)) + 67] \geqslant 99.8\%$. Since $\mathrm{I}(\mathrm{rank}^{-1}(3)) = 9700$ and $\mathrm{I}(\mathrm{rank}^{-1}(5)) = 9500$, for this instance, $Pr[\mathcal{I}_3^L \geqslant 3 \wedge \mathcal{I}_3^U \leqslant 5] \geqslant 99.8\%$, which ensures that the final released confidence interval is tight around the true rank 5.*

The proof of item 1 follows from the composition theorem and the property of Gaussian mechanism [30]. The proof of item 2 and item 3 is based on the property of the random binary search. We defer the formal proofs to Appendix A.1.2.

### 4.4.4 Putting it All Together

After we selected top-$k$ explanation predicates (Section 4.4.3), and constructed the confidence intervals of their influences (Section 4.4.3) and ranks (Section 4.4.3), the final step is to combine them together into a single explanation table as defined

below. Since the absolute influence is hard to interpret, we use relative influence instead to present to the users, which is achieved by dividing the pre-computed bounds by $|\hat{o}_i - \hat{o}_j|$ when $agg \in \{COUNT, SUM\}$ or by $|\hat{o}_i - \hat{o}_j|(\max(\hat{o}_i^C, \hat{o}_j^C)) + 1)$ when $agg = AVG$ according to Definition 4.4.3. Since this table contains the numbers about influences and ranks, users may expect the table to be sorted accordingly. We sort the selected top-k explanations by the upper bound of the relative influence CI in an descending order; if there is a tie, we further sort by the upper bound of the rank confidence interval in an ascending order.

**Definition 4.4.12** (Explanation Table containing top-$k$ explanations)**.** *Given a database $D$, a group-by aggregate query $q$ as shown in Figure 4.7, a user question $(\alpha_i, \alpha_j, >)$, a predicate space $\mathcal{P}$, a confidence level $\gamma$, and an integer $k$, a table of top-$k$ explanations is a list of $k$ 5-element tuples $(p_u, \mathcal{I}^L_{relinflu_u}, \mathcal{I}^U_{relinflu_u}, \mathcal{I}^L_{ranku}, \mathcal{I}^U_{ranku})$ for $u = 1, 2, \ldots, k$ such that $p_u$ is an explanation predicate, $(\mathcal{I}^L_{relinflu_u}, \mathcal{I}^U_{relinflu_u})$ is a confidence interval of relative influence $\tilde{I}(p_u)$ with confidence level $\gamma$, and $(\mathcal{I}^L_{ranku}, \mathcal{I}^U_{ranku})$ is a confidence interval of* $\text{rank}(p_u)$ *with confidence level $\gamma$*

We summarize the privacy guarantee of our DPXPlain framework as follows: (i) the private noisy query answers returned by Gaussian mechanism in Phase-1 satisfy $\rho_q$-zCDP together (see Section 4.2) ; (ii) Phase-2 only returns the confidence intervals of the noisy answers in Phase-1 as defined by the Gaussian mechanism and does not have any additional privacy loss (discussed in Section 4.4.1) ; (iii) Phase-3 returns $k$ explanation predicates and their upper and lower bounds on relative influence and ranks given a required confidence interval, and uses three privacy parameters $\rho_{Topk}, \rho_{Influ}, \rho_{Rank}$ (discussed in Section 4.4.3, 4.4.3 and 4.4.3). The following theorem summarizes the total privacy guarantee.

**Theorem 4.4.13.** *Given a group-by query $q$ and a user question comparing two*

*aggregate values in the answers of $q$, the* DPXPlain *framework guarantees* $(\rho_q + \rho_{Topk} + \rho_{Influ} + \rho_{Rank})$*-zCDP.*

## 4.5 Evaluation

We evaluate the quality and efficiency of the explanations generated by DPXPlain. In particular, we examine the following questions:

1. How accurate is the noise interval generated by DPXPlain in capturing the true answer to the initial query in Phase-2?
2. How accurate is the noisy top-k compared to the true top-k and how precise are the descriptions of these predicates in terms of influence and rank confidence intervals in Phase-3?
3. How efficient is DPXPlain in preparing the explanations?

We have implemented DPXPlain in Python 3.7.4 using the Pandas [157], NumPy [86], and SciPy [161] libraries. All experiments were run on Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz with 32 GB of RAM. The source code can be found in [1].

### 4.5.1 Experiment Setup

We next detail the data, queries, questions, and parameters used in the experiments.

*Datasets.* We consider two datasets in our experiments.

- IPUMS-CPS: A dataset of Current Population Survey from the U.S. Census Bureau [79]. We focus on the survey data from year 2011 to 2019. The dataset contains 8 categorical attributes where domain sizes vary from 3 to 36 and one numerical attribute. The attribute AGE is discretized as 10 years per range, e.g.,

[0,10] is considered a single value. To set the domain of numerical attributes, we only include tuples with attribute INCTOT (the total income) smaller than 200k as a domain bound. The total size of the dataset is over one million.

- Greman-Credit: A corrected collection of credit data [84]. This dataset includes 20 attributes where the domain sizes vary from 2 to 11 and a numerical attribute. Attributes duration, credit-amount, and age are discretized. The domain of attribute good-credit is zero or one. We synthesize the dataset to 1 million rows by combining a Bayesian network learner [13] and XGBoost [29] following the strategy of QUAIL [142].

*Queries and Questions.* The queries and questions used on the experiments are shown in Table 4.1.

**Table 4.1**: Queries and questions for the experiments. The column Valid indicates it is a valid question or not.

| Data | Query | Question | Valid |
|---|---|---|---|
| IPUMS-CPS | $q_1$: AVG(INCTOT) by SEX | I1: Why Male > Female ? | Yes |
| | $q_2$: AVG(INCTOT) by RELATE | I2: Why Grandchild > Foster children ? | Yes |
| | | I3: Why Head/householder > Spouse ? | No |
| | $q_3$: AVG(INCTOT) by EDUC | I4: Why Bachelor > High school ? | Yes |
| | | I5: Why Grade 9 > None or preschool ? | No |
| German-Credit | $q_4$: AVG(good-credit) by status | G1: Why no balance > no chk account ? | Yes |
| | $q_5$: AVG(good-credit) by purpose | G2: Why car (new) > car (used)? | Yes |
| | | G3: Why business > vacation ? | No |
| | $q_6$: AVG(good-credit) by residence | G4: Why "< 1 yr" >">= 7 yrs" ? | Yes |
| | | G5: Why "[1, 4) yrs" > "[4, 7) yrs" ? | No |

*Default setting of* DPXPlain. We consider the following default algorithm parameters: $\rho_q = 0.1$, $\rho_{Topk} = 0.5$, $\rho_{Influ} = 0.5$, $\rho_{Rank} = 1.0$, $\gamma = 0.95$, $k = 5$, and the number of conjuncts $l = 1$, which defines the set of explanation predicates $\mathcal{P}$ including those satisfying Definition 4.3.3 with a fixed $l$.

### 4.5.2 Qualitative Analysis

*Case-1, IPUMS-CPS.* We present a case study with the dataset IPUMS-CPS. All parameters are set by default. In Phase-1, the user submits a query $q_1$ from Table 4.1, and gets a noisy result: (Female, 31135.25) and (Male, 45778.46). The truth are (Female, 31135.78) and (Male, 45778.39) but hidden from the user. Next, in Phase-2, since there is a gap of 14643.21 between the noisy avg-income from two groups, the user asks a question I1 from Table 4.1. The framework then quantifies the noise in the question by reporting a confidence interval of the difference between two groups as (14636.63, 14649.79). Since the interval does not include zero, the framework suggests that this is a valid question, which is correct. Finally, in Phase-3, the framework presents top-5 explanations to the user as Figure 4.9 shows. The last two columns are the true relative influences and true ranks. We correctly find the top-5 explanation predicates, and the first and fourth explanations together suggests that a married man tends to earn more than a married woman, which is supported by the wage disparities in the labor market. The second and third explanations also matches the wage disparities within the educated group and white people. The total runtime for preparing the explanation by Phase-2 and Phase-3 is 67 seconds.

| explanation predicate | Rel Influ 95%-CI | | Rank 95%-CI | | Rel Influ | Rank |
|---|---|---|---|---|---|---|
| | L | U | L | U | (hidden) | (hidden) |
| RELATE = "Head/householder" | 11.49% | 11.82% | 1 | 1 | 11.69% | 1 |
| EDUC = "Bachelor's degree" | 6.81% | 7.13% | 2 | 3 | 6.89% | 2 |
| RACE = "White" | 6.15% | 6.47% | 2 | 5 | 6.16% | 3 |
| RELATE = "Spouse" | 5.53% | 5.86% | 2 | 5 | 5.67% | 4 |
| CLASSWKR = "NIU" | 3.62% | 3.94% | 2 | 6 | 3.98% | 5 |

**Figure 4.9**: Phase-3 of DPXPlain for the case IPUMS-CPS.

*Case-2, German-Credit.* We now present another case study over the dataset German-Credit. All parameters are set by default. In Phase-1, the user submits

a query $q_4$ from Table 4.1, and gets a noisy result: ("no checking account", 0.526571) and ("no balance", 0.574447). The true result is ("no checking account", 0.526574) and ("no balance", 0.574466) but this is hidden from the user. Next, in Phase-2, since there is a gap of 0.047876 between the noisy avg-credit from two groups, the user asks a question G1 from Table 4.1. The framework then quantifies the noise in the question by reporting a confidence interval of the difference between two groups as (0.047786, 0.047967). Since the interval does not include zero, the framework suggests that this is a valid question, which is correct. Finally, in Phase-3, the framework presents top-5 explanations to the user as Figure 4.10 shows. The last two columns are the true relative influences and true ranks. We correctly find the top-5 explanations, and the first explanation suggests that for a person who has already had a credit in the bank, the bank tends to mark its credit as good if she has a checking account even with zero balance with higher probability than the case of no account, which is consistent to the intuition that a person having a credit account but no checking account is risky to the bank. The total runtime for preparing the explanation by Phase-2 and Phase-3 is 40 seconds.

| explanation predicate | Rel Influ 95%-CI | | Rank 95%-CI | | Rel Influ (hidden) | Rank (hidden) |
|---|---|---|---|---|---|---|
| | L | U | L | U | | |
| existing-credits = "1" | 76.18% | 77.25% | 1 | 1 | 76.92% | 1 |
| job = "skilled employee / official" | 69.92% | 70.99% | 1 | 2 | 70.69% | 2 |
| sex-marst = "male : married/widowed" | 53.88% | 54.95% | 2 | 4 | 54.22% | 3 |
| credit-amount = "(500, 2500]" | 48.68% | 49.75% | 2 | 5 | 49.47% | 4 |
| credit-history = "no credits taken/all credits paid back duly" | 47.70% | 48.77% | 4 | 5 | 48.36% | 5 |

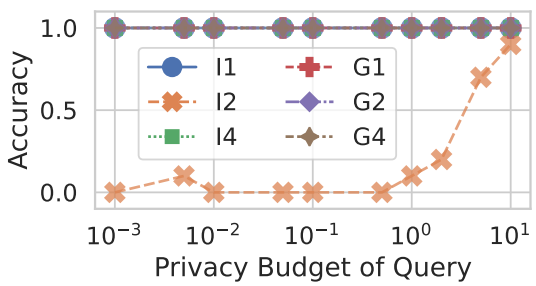
**Figure 4.10**: Phase-3 of DPXPlain for the case German-Credit.

### 4.5.3 Quantitative Analysis

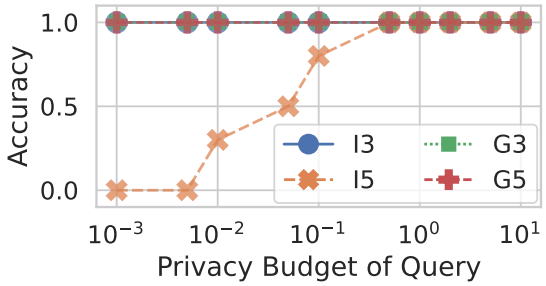
We detail our experimental analysis for the different questions and configurations of DPXPlain. All results are averaged over 10 runs.

*Correctness of noise interval.* In Phase-2 of DPXPlain, user has the option to either stop or proceed based on the confidence interval of the question. We evaluate a simple strategy for judging the validity of the question by whether the confidence interval includes numbers on the negative axis or not. We evaluate the accuracy of this strategy by dividing the questions from Table 4.1 into two categories: valid and invalid, and report the fraction that our framework successfully suggests the correct validity of the question. From Figure 4.11, we find that 8 out of 10 questions, either valid or invalid, are suggested correctly with accuracy 100% given a wide range of privacy budget of query $\rho_q$. However, there are two questions, I2 and I5, only show high accuracy given a large privacy budget as $\rho_q = 10$. One reason is that the minimum group size involved in these questions are small compared to other questions, and therefore the partial confidence intervals at the denominators of the $AVG$ query are low, which makes the final confidence intervals wider and tend to include negative numbers when it should not. For I2, the minimum group size is at least 600 times smaller than other questions about IPUMS-CPS while this number for I5 is 60.

*Accuracy of top-k explanation predicates.* In Phase-3 of DPXPlain, we first select top-k explanation predicates. We measure the accuracy of the selection by Precision@k [89], the fraction of the selected top-k explanation predicates that are actually ranked within top-k. See Appendix A.3.1 for another experiment on the full ranking. From Figure 4.12a, we find that the privacy budget of top-k selection $\rho_{Topk}$ has a positive effect to Precision@k at k = 5 for various questions. When $\rho_{Topk} = 1.0$, all the questions except I2 and I5 have Precision@k $\geqslant 0.8$. The selection accuracy of




(a) Valid Questions. All lines beside I2 are consistently at 100%.



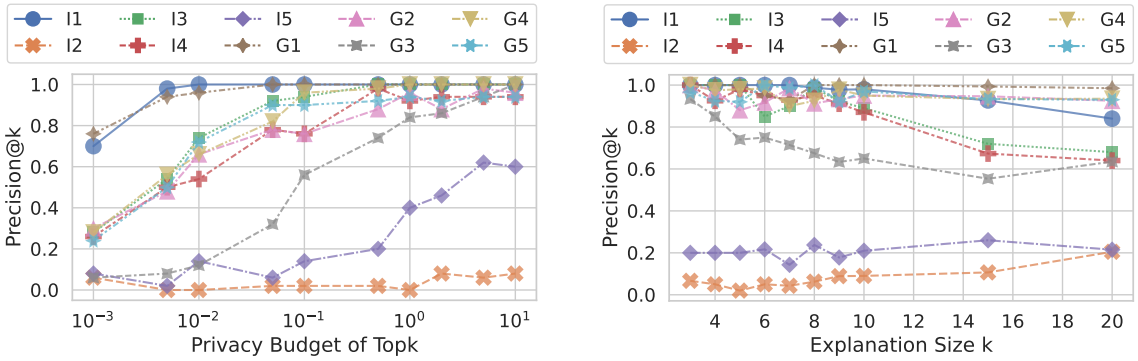

(b) Invalid Questions. All lines beside I5 are consistently at 100%.

**Figure 4.11**: Accuracy of question validation by DPXPlain.

question I2 and I5 are generally lower than other questions. This is because they have small group sizes, and therefore the influences of explanation predicates are small and the ranking are perturbed by the noise more significantly.

From Figure 4.12b, we find that the trend of Precision@k by k is different across questions and there is no clear trend that Precision@k increases as k increases. For example, for G3, it first decreases from k=3 to k=5, but increases from k=5 to k=6. When k = 3, most questions have high Precision@k; this is because the highest three influences are much higher than the others, which makes the probability high to include the true top three. When k is large, for explanation predicates that have similar scores, they have equal probability to be included in top-k and therefore the top-k selected by the algorithm are different from the true top-k selections. The relationship between Precision@k and k depends on the distribution of all the explanation predicate influences. We also study the relationship between Precision@k at k=5 with the conjunction size $l$. For two questions I1 and G1, their Precision@k keep to be 1.0 when $l$ is changed among $1, 2$, and 3. Although the size of explanation predicates grows exponentially with the conjunction size $l$, DPXPlain can select top-k accurately even from a large space.

*Precision of relative influence and rank confidence Interval.* In Phase-3, the last step




(a) Trend by $\rho_{Topk}$.

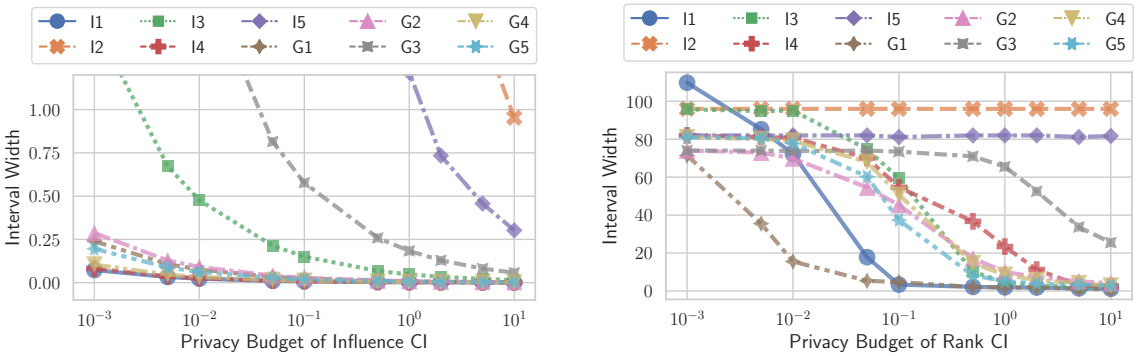
(b) Trend by $k$.

**Figure 4.12**: Precision@k of top-k selection by DPXPlain.

is to describe the selected top-k explanation predicates by a confidence interval of relative influence and rank for each. To measure the precision of the description, we adopt the measure of interval width [78]. Figure 4.13 illustrates the average width of $k$ confidence intervals of relative influence and rank. From Figure 4.13a and 4.13b, we find that the increase of privacy budget $\rho_{Influ}$ and $\rho_{Rank}$ shrinks the interval width of relative influence CI and rank CI separately. In particular, when $\rho_{Influ} \geqslant 0.5$, 6 out of 10 questions have the interval width of relative influence CI $\leqslant 0.015$ ; when $\rho_{Rank} \geqslant 1.0$, 2 questions have the interval width of rank CI $\leqslant 2$ and 6 questions have this number $\leqslant 10$.

We also measure the effect of confidence level $\gamma$ to the confidence interval by changing $\gamma$ from 0.1 to 0.9 by step size 0.1 and from 0.95 and 0.99. The results show that it has a non-significant effect to the interval width, as it changes $< 0.02$ for the influence CI of 6 questions, and changes $< 5$ for the rank CI of 8 questions.

*Runtime analysis.* Finally, we analyze the runtime of DPXPlainfor preparing the explanations from Phase-2 and Phase-3. From Figure 4.14a, a runtime breakdown in average for all the questions from Table 4.1 with total runtime 32 seconds in average, shows that 88% of the time is used for the top-k explanation predicate selection procedure, especially on computing the influences for all the explanation




(a) Relative Influence (b) Rank

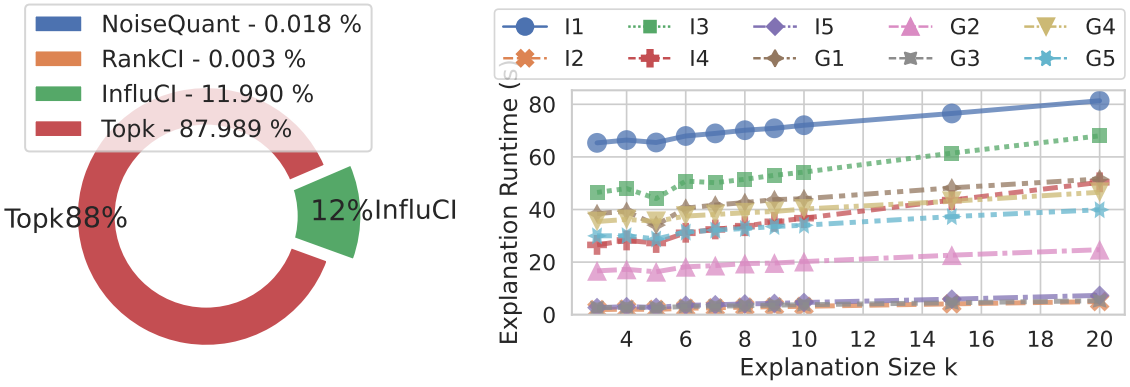
**Figure 4.13**: The width of confidence intervals by DPXPLAIN.

predicates. The next highest time usage is for computing the confidence interval of influence, which needs to evaluate each sub queries. For the step noise quantification and confidence interval of rank, the time usage is not significant since the first only needs to find the image of two intervals and the second is a binary search. From Figure 4.14b, we find the runtime is linearly proportional to the size of explanations $k$, and the difference between questions is due to the difference of group sizes. We also find the runtime is exponentially proportional to the number of conjuncts $l$: for $l = 1, 2, 3$, the runtime about question I1 is 67, 3078 and 79634, and for question G1 it is 40, 1587 and 39922 (in seconds). This is because the number of explanation predicates grows exponentially to the number of conjuncts $l$.

*Summary of our findings.*

1. DPXPLAIN can correctly suggests the validity of the question with 100% accuracy for 8 out of 10 questions at $\rho_q = 0.1$.

2. DPXPLAIN can select at least 80% of the true top-5 explanation predicates correctly for 8 out of 10 questions at $\rho_{Topk} = 0.5$, and the associated confidence intervals of relative influence have width $\leqslant 0.015$ for 6 questions at $\rho_{Influ} = 0.5$ and the confidence intervals of rank have width $\leqslant 10$ for 6 questions at $\rho_{Rank} =$




(a) Breakdown (b) Trend by $k$

**Figure 4.14**: Runtime analysis of DPXPlain.

1.0.

3. The runtime of DPXPlain for preparing the explanations is in average 32 seconds for the 10 examined questions.

## 4.6 Related Work

We next survey related work in the fields of differential privacy and explanations for query results. *To the best of knowledge, this is the first work that considers augmenting query results with explanations while satisfying differential privacy.*

*Explanations for query results.* The database community has explored this subject in multiple previous works [51, 91, 90, 110, 34, 158, 52, 128, 129, 117, 140, 130, 115, 166, 144, 143]. Proposed approaches include provenance [51, 91, 90, 110, 34, 158, 52, 111], causality [128, 129], Shapley value [117, 140], Patterns [130, 115], and predicates [166, 144, 143]. We note that works that consider individual tuples or explicit tuple sets in any form as explanations (e.g., [128, 111, 51, 117]) cannot be applied in our setting since they would directly violate DP, thus, approaches that use predicates as summarized explanations are more practical. Among these, Scorpion [166] has focused on explaining outliers in query results with the most influential predicate,

as mentioned in Section 4.4.2. Our influence function is based on this work, but has been augmented to deliver accurate results while satisfying DP. Another line of work on predicate explanations that inspired this paper [144, 143] has considered explanations for user questions of a slightly different format while measuring the effect of the predicates using causal paths in the database. While these approaches have certainly influenced our work, DPXPlain was developed specifically to satisfy DP, which has required a novel framework and solutions that support it.

*Differential privacy.* The most relevant fields of differential privacy to this work includes private SQL query answering [106, 164, 107, 94, 122, 155, 60], private confidence interval [77, 44, 28], private selection [137, 63, 18, 126, 35, 125, 57, 27, 116, 32, 109, 156, 23], private quantile [81, 98, 10, 43, 66, 112, 149] and private rank aggregation [170, 87, 148]. Private SQL query answering systems such as PrivateSQL [106] considers a workload of selective counting queries with or without joins on a multi-relational database, while our framework focuses on a single group-by query with three different aggregations. The selection of top-k candidates is well-studied by the community, such as one-shot top-k [137], while we focus on designing a low-sensitive influence function to boost the utility of top-k selection. Private confidence interval is a new trend of estimating the uncertainty under differential privacy, however the current bootstrap based methods, such as [77], measure the uncertainty from both the sampling process and the noise injection, while we only focus on the second part. The most relevant work to the private rank estimation in our framework is private quantile, such as [81], and private rank aggregation, such as [87]; however, neither of them can be directly applied to our setting.

*Privacy and provenance.* As mentioned above, provenance is often used for explaining query results. Within the context of provenance privacy [45, 46, 22, 147, 145, 12, 153, 42], one line of work [45, 46, 47] studied the preservation of workflow privacy,

with a privacy criterion inspired by $l$-diversity [119] and $k$-anonymity [152]. A recent work [53] explored what can be inferred about the *query* from provenance-based explanations and found that the query can be reversed-engineered from the provenance in various semirings [82]. To account for this, a follow-up paper [50] proposed an approach for provenance obfuscation that is based on abstraction. This work uses $k$-anonymity [152] to measure how many 'good' queries can generate concrete provenance that can be mapped to the abstracted provenance, thus quantifying the privacy of the underlying query.

## 4.7 Discussions and Future Work

We propose a formal framework to explain a user question over a group-by aggregate query answer (COUNT, SUM, or AVG) under DP with three phases: private query answering, question validation and private explanation table. We adapt the influence function [166] to the setting of differential privacy by reducing its sensitivity. We also propose the algorithms for computing the confidence intervals of questions, relative influences and ranks.

There are a few directions worthy exploring in the future. First, a tighter analysis to the sensitivity of the influence function is the key to the successful application of differential privacy. Second, the complexity of the top-k selection algorithm is high since it needs to iterate over all the explanation predicates, leaving room for future improvements. Finally, extending the framework to more types of queries and questions is also an important direction.

# Chapter 5

# Related Work

In this chapter we give an overview of the application of differential privacy to SQL queries besides the related work discussed by Chapter 3 and Chapter 4. Section 5.1 summarizes two dimensions in this field. Section 5.2 discuss the approaches of answering non-linear SQL queries. Section 5.3 discuss the approaches of synthetic data generation for answering unlimited SQL queries.

## 5.1 Different Dimensions of Problems

It is not yet known whether there is a single mechanism that can provide strong utility guarantee for all problems while satisfying differential privacy. These problems could vary in scope and complexity. For example, one problem is to answer a set of statistical queries, and another problem is to release a synthetic data set such that it could be used to answer a set of statistical queries. Meanwhile, these statistical queries could either be linear queries or non-linear queries. From these examples, we observe that problems could be categorized by different characteristics on different dimensions. In this section, we discuss the problems that are commonly studied in differential privacy community in different dimensions.

- **Linear vs non-linear queries**

  Queries like histogram, marginal and range counting queries are all linear queries. Linear queries are queries that can be expressed as a linear combination of individual weights. The global sensitivity for a linear query is thus bounded by the maximal weight times the maximal factor. However, for non-linear queries, the global sensitivity could be much higher or even unbounded.

For example, a triangle counting query, which is to count the number of triangles in a graph, has global sensitivity unbounded since there always exists a graph such that adding a node or edge can increase the number of triangles by an arbitrary high number. Adding Laplace noise scaled to the global sensitivity is thus meaning less in this case, so for non-linear queries a different mechanism design should be considered.

- **Synthetic vs query answering**

  When we are designing a system that can both protect the data privacy and provide utility for the data analysts, we are facing two choices: one is to make the system interactive, which is to take the queries from the data analysts and return the answers interactively. Another is to make the system non-interactive, which is to release a synthetic data set on which data analysts can use to answer queries. In the interactive setting, system can choose a mechanism tailored to the queries of interest [105]. However, the privacy budget may run out and no future queries could be answered. When synthetic data is released, arbitrary queries or even complex learning tasks can be performed on the synthetic data set. However, the synthetic dataset will not be accurate for all query/analytics tasks.

  There is a large scope of query answering problems that are studied in the differential privacy community, such as range counting queries [88], marginal queries, graph queries, linear regressions and so on. For the synthetic data generation problem, approaches include density estimation, using domain decomposition, probabilistic graphical models and ML approaches like GANs.

## 5.2 Non-linear Query Answering

Non-linear queries are queries that cannot be expressed as the weighted summation of data elements. Examples include quantile queries, join-count queries, and graph queries. Most DP mechanisms are designed for linear queries. For non-linear queries, the global sensitivity is not clear and naively applying Laplace mechanism may fail to follow DP.

### 5.2.1 Smooth Sensitivity

For a linear query, the global sensitivity is determined by the max weight in the query. For example, the global sensitivity is 1 for a common predicate-counting query where each weight is one or zero. To answer such a linear query under $\epsilon$-differential privacy, one approach is apply the Laplace mechanism which adds a Laplace noise scaled to the global sensitivity of the query divided by the privacy budget $\epsilon$. However, for a non-linear query, the global sensitivity is often much larger. For example, for a median query on a data set with domain $[0, \Lambda]$, the global sensitivity is $\Lambda$. Consider a data set with $2n+1$ elements such as $\{0, 0, \ldots, \Lambda, \Lambda\}$ where $n$ are 0 and $n+1$ are $\Lambda$. The median is this case is $\Lambda$. However, under bounded differential privacy, consider a neighboring database by changing the $n+1^{st}$ element from $\Lambda$ to 0; the median drops to 0. This example shows that the global sensitivity of median is $\Lambda$, and thus releasing the median using Laplace mechanism could result in meaningless outputs when $\Lambda$ is very large or the number of elements is small. This is surprising, since the median is less sensitive especially for large datasets, but the noise introduced by Laplace mechanism is not decreasing as we increase the data size.

One way to reduce the noise needed to achieve differential privacy is to replace global sensitivity with a measure of sensitivity on the specific database instance. Local sensitivity is the maximum query difference between the input database and

all its neighboring data sets. For measures like the median, the local sensitivity varies depending on the input database. For the pathological example above, the local sensitivity is still $\Lambda$. However, for a database with $2n+1$ elements with the same value in the positions $n$, $n+1$ and $n+2$, then the local sensitivity is 0! However, the local sensitivity itself is a sensitive information of the data (e.g. for the case of median you can tell the values in positions $n$, $n+1$ and $n+2$ are the same when local sensitivity is 0). Hence, adding Laplace noise scaled to the local sensitivity is does not guarantee differentially privacy. To connect local sensitivity to differential privacy, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith propose the smooth sensitivity mechanism [132]. Smooth sensitivity is a tightest smooth upper bound of the local sensitivity such that itself is $e^{\beta}$-Lipschitz for some $\beta$: i.e., smooth sensitivity $SS(\cdot)$ has the following property: $SS(D) \geqslant LS(D)$ and $|SS(D) - SS(D')| \leqslant e^{\beta} \cdot d(D, D')$ for any data sets $D$ and $D'$, where $LS(D)$ is the local sensitivity of $D$ and $d(D, D')$ is the hamming distance between $D$ and $D'$. Calibrating noise to smooth sensitivity and adding it to the raw query result achieves $(\epsilon, \delta)$-differential privacy.

Deriving the smooth sensitivity $S_f$ from the local sensitivity $LS_f$ for the query $f$ on some data set is not trivial. Taking the median as an example. Consider a data set with $n$ elements and $n$ is odd. Suppose data is ordered as $\{x_1, x_2, \ldots, x_n\}$. Denote $m = \frac{n+1}{2}$ as the index for the median. The local sensitivity $LS_{med}$ is thus equal to $\max(x_m - x_{m-1}, x_{m+1} - x_m)$. To derive the smooth sensitivity, [132] considers an extension of local sensitivity as local sensitivity at distance k, $LS_f^{(k)}$, which is the max local sensitivity for the data set that differs by at most $k$ rows with the ground truth data set. The smooth sensitivity of $f$ is thus equal to the max of $e^{-k\beta} LS_f^{(k)}$ for $k \in \{0, 1, \ldots, n\}$. For the median case, its local sensitivity at distance $k$ is given as $LS_{med}^{(k)} = \max_{0 \leqslant t \leqslant k+1} (x_{m+t} - x_{m+t+k-1})$, which can be further used to derive the smooth sensitivity, and the running time for computing its smooth sensitivity is thus $O(n^2)$.

It is not always clear how to efficiently compute the smooth sensitivity. A naive algorithm to find local sensitivity at distance k is to try all possible data set by changing up to $k$ tuples, and for each data set iterate over all neighboring data sets of that data set to compute the local sensitivity. To deploy smooth sensitivity in a real application, it is important to find a computationally efficient algorithm to compute the smooth sensitivity or its approximate upper bound.

### 5.2.2 Lipschitz Extension

One representative class of non-linear queries is sub-graph counting queries. For example, given an un-directed graph $G(V, E)$, a triangle counting query is to ask the number of triangles in the graph. Other sub-graph counting queries includes counting the number of edges, $k$-stars and so on. If we think about each node in the graph is the basic element, sub-graph counting queries cannot be expressed as a linear combination of nodes since each node may be involved in multiple sub-graphs.

To answer these queries under differential privacy, a basic solution is to apply Laplace mechanism, which adds the Laplace noise scaled to the global sensitivity. Recall that the definition of global sensitivity for a function $f$ is $\max |f(G) - f(G')|$ for all $G$ and $G'$ that are neighbours. Here we consider $G$ and $G'$ are neighbours if they differ by a single node. This is so called node-DP. Under node-DP, for a graph query like counting the number of triangles, the global sensitivity is unbounded. Even if we assume the total number of nodes is $n$, the global sensitivity is $\binom{n-1}{2}$ due to a pair of neighbouring graphs that are $n$-clique and $(n-1)$-clique.

If we assume the graph degree is bounded by $D$ and denote this space as $\mathcal{G}^D$, then the global sensitivity of triangle counting query is $\binom{D}{2}$, which is much smaller than $\binom{n-1}{2}$ since $D \ll n$ is common. Usually $D$ can be selected by some common knowledge or some public information, so that it is close to the real degree bound.

However, we cannot guarantee this assumption is always true. When this assumption does not hold, we still want our query to be of low sensitivity as the case when the assumption is true, but we don't pay much attention to the accuracy. This leads to the solution based on Lipschitz extension, which is to find a new function that gives the same answer in the original scope and also has the same global sensitivity [1]. For example, suppose $f$ is defined on graphs from $G^D$, we can find a new function $\hat{f}$ that takes any graph as input such that $\hat{f}(G) = f(G)$ on $G \in \mathcal{G}^D$ and the global sensitivity $GS(\hat{f}) = GS(f)$.

Finding the Lipschitz extension of graph queries is not trivial. Shiva Prasad Kasiviswanathan, Kobbi Nissim, Sofya Raskhodnikova and Adam Smith consider using max flow of a constructed flow graph for answering the number of edges and using linear programming for answering the number of sub-graphs [101]. Given a graph $G(V, E)$, a flow graph is constructed as a source $s$, a sink $t$ and two copies of $V$ as $V_l$ and $V_r$ such that the source $s$ is connected to each $v_l$ in $V_l$ with capacity $D$, each $v_r$ in $V_r$ is connected to the sink $t$ with capacity $D$ and $v_l$ is connected to $v_r$ with unit capacity if $(v_l, v_r) \in E$. The half of the max flow in this graph is a Lipschitz extension of the number of edges for graphs from $\mathcal{G}^D$. For answering the number of a specific sub-graph in $G$, a Lipschitz extension based on linear programming is considered as follows: Suppose $\Delta_D$ is the global sensitivity of answering the number of sub-graph for graphs from $\mathcal{G}^D$. Create variables $x_c \in [0, 1]$ for each possible sub-graph $c$ in G (no matter whether it exists or not). For each node in $G$, the sum of $x_c$ that is adjacent to that node is bounded by $\Delta_D$. The objective is to maximize the sum of $x_c$. The optimal value by solving this linear programming is a Lipschitz extension of the sub-graph counting for graphs from $\mathcal{G}^D$.

These examples of Lipschitz extension shows the applicability of Lipschitz exten-

[1] If the new global sensitivity is $s$ times the original one, it is called Lipschitz extension with stretch $s$

sion to graph queries with scalar outputs. For graph queries with multi-dimensional outputs, such as degree distribution, another flow graph based solution is considered [138]. Other studies based on Lipschitz extension include [25, 48, 56]. Studies that involves the similar idea of Lipschitz extension includes [41, 174, 106, 155]

### 5.2.3 Answering SQL Queries

SQL queries are widely existing in the real world data analysis tasks. They provide a rich model of query structures that is far beyond simple linear queries. Privately answering SQL queries is thus challenging. On the other hand, SQL queries are often asked on a relational database with multiple relations. Relations are usually associated with each other by some foreign-primary keys. In a multi-relational database, if we remove one tuple in a relation, it is nature to consider cascade delete as removing tuples in other relations that have foreign keys associated with the tuple. In this case, even for a simple linear query, the privacy analysis is not clear since removing a tuple may cause cascade deletions and the sensitivity for the linear query thus is not always a constant.

Kotsogiannis et al. propose a system PrivateSQL for answering SQL queries privately [106]. It extends differential privacy to multi-relational databases with foreign-primary key constraints, which captures Edge- or Node-DP for graphs [99, 101]. It also supports complex SQL queries that include JOINS, GROUPBY and correlated sub-queries. To boost accuracy, PrivateSQL considers decomposing queries into multiple workloads. Within each workload, queries are transformed as a set of linear queries on a complex view. To ensure differential privacy on a multi-relational database, PrivateSQL applies the query rewriting technique to the view so that it is equivalent to the standard differential privacy on a single data set and the stability of the view is also bounded. View stability is the maximum change of rows in the

view after adding or removing a single private tuple. Since it is well studied about answering a workload of linear queries on a view with stability 1 [88, 123], it is natural to extend this problem to a view with a higher stability.

Tracking the stability of a view is not trivial, since a view is equivalent to a tree of SQL operations with arbitrary size. Flex gives a rule-based stability calculation strategy by recursively update the stability of a node in the SQL query tree [94]. To capture the stability update due the JOIN operator, Flex also tracks the frequency of attribute values. Since attribute frequency is a sensitive information, the stability of the view cannot be revealed. Therefore, Flex uses smooth sensitivity [132] to perturb the query answer. PrivateSQL extends this rule-based stability calculator with new rules on the JOIN stability update. Furthermore, PrivateSQL computes the attribute frequency in a differentially private approach through sparse vector technique [69] and enforces the frequency bound by injecting truncation operators into the SQL query tree, which allows the view stability to be publicly accessible. An alternative to find a good bounding frequency is through recursive mechanism [41], which is based on a list of max frequencies related to the subsets of the primary private relation with size ranging from zero to full. Another private SQL engine is proposed by Wilson et al. [164], which considers aggregations other than counting, such as average and quantile. They also consider bounding the user contribution by a fixed number in a join query using reservoir sampling.

## 5.3 Synthetic Data

A typical scenario for data analysis is that one data analyst submits some queries to the data curator, and data curator returns the query answers to the analyst. To ensure data is protected by differential privacy, these query answers are perturbed by some differentially private mechanisms. However, data curator can also choose

another strategy to respond to the request. Instead of returning the noisy query answers, data curator can also generate a synthetic data set that satisfies differential privacy, and then return it to the data analyst. Data analyst can evaluate queries on the synthetic data set to get query answers, and it satisfies differential privacy due to the post-processing rule. The utility of a synthetic data release is associated with a specific task or metric. For example, the utility could be associated with the expected $\ell_2$ error of a set of linear queries evaluating on the original data and the synthetic data.

Avrim Blum, Katrina Ligett and Aaron Roth show that when considering answering a class of counting queries $C$ using the synthetic data set while satisfying $\epsilon$-differential privacy, the lower bound of error only depends on the VC-dimension of the query class $C$ and the privacy factor $\epsilon$ [26]. They define $(\alpha, \delta)$-usefulness for a mechanism with respect to the query class $C$ if the max $L_1$ query error for any query in $C$ is bounded by $\alpha$ with probability $1-\delta$. Based on the reconstruction proof for "blatant non-privacy" [58], they show that given a database with size $\leqslant VCDIM(C)/2$, for any $\epsilon$-differentially private mechanism that is $(\alpha, \delta)$-useful for a query class $C$, we have $\alpha \geqslant \frac{1}{2\exp(\epsilon)+1}$. They also propose the Net mechanism [26] that construct a set of candidate data sets, called $\alpha$-net, such that for any ground truth data set, there always exists a candidate data set that can accurately answers any query from a fixed query class compared to the ground truth one with L1 error less than $\alpha$. The Net mechanism then chooses a data set from the $\alpha$-net using exponential mechanism with score function as the inverse max L1 error for all queries, and thus satisfies differential privacy. However, the size of $\alpha$-net is often large. The bound given in [26] shows that for any counting class $C$ and any data domain $X$, the size of minimal $\alpha$-net is at most $|X|^{O(VCDIM(C)log(1/\alpha)/\alpha^2)}$. It is also computationally infeasible to sample a data set from the $\alpha$-net according to the exponential mechanism.

Another approach for private synthetic data generation uses generative models, which includes probabilistic graphical models and deep generative models. Approaches based on probabilistic graphical models include Bayesian network [175] and Markov network [39, 21, 124, 172]. Most of these approaches first learn the probabilistic graphical model structure of the data, and then learn the parameters for the model. Approaches based on deep generative models include DP-AuGM [38], DP-VaeGM [38], DPGAN [169] and PATE-GAN [96]. These approaches are based on training (variational) autoencoders or generative adversarial networks in a differentially private way. Other approaches includes estimating the data distribution from the noisy DP answers, such as MWEM [85] and PGM estimation [124].

### 5.3.1 Markov Network

There are many ways to express data. It could be expressed as a data frame with $N$ rows, where each row is a tuple with $n$ attributes. Suppose the domain for a tuple is $\mathcal{X}$, we could also express this data as a $|\mathcal{X}|$-dimensional full-domain vector, where each cell in the vector indicates the fraction of rows in the data matching a specific tuple value. Assuming $N$ is public and fixed, this $|\mathcal{X}|$-dimensional vector is equivalent to a contingency table of the data, which lists the number of rows matched for all possible values of $n$ attributes.

To generate a synthetic data set under differential privacy, one approach is to perturb the $|\mathcal{X}|$-dimensional full-domain vector using Laplace mechanism, post-process the noisy vector to be non-negative, normalize it to be a probability distribution and then sample a synthetic data set from it. Roughly speaking, the expected statistical distance between the sanitized distribution and the ground truth grows with the ratio $|\mathcal{X}|/N$, where $|\mathcal{X}|$ is the size of the full-domain vector and $N$ is the data size. When $|\mathcal{X}|$ is much larger than the data size $N$, most cells in the full-domain vector are zero

and few cells have a low non-zero counts, thus adding Laplace noise to all cells results in a huge information loss, which further make the sanitized distribution useless.

An alternative approach is to learn several marginal distributions of the data using differentially private mechanisms, and then infer the full-domain distribution with some principles. Since the full-domain distribution is of high dimension, it is important to find a computationally tractable expression of the distribution. Besides, a principle to infer the full-domain distribution is to find a distribution such that its marginal distributions are close to the sanitized ones generated by the differentially private mechanisms. Based on these ideas, Ryan McKenna, Daniel Sheldon and Gerome Miklau propose a inference principle based on the undirected graphical model of data and the space of marginal polytope [124]. An undirected graphical model, or Markov network, factorizes the data distribution into a product of factor functions over the cliques of a graph with a normalization. In [124], they assume all the attributes from the noisy marginal distributions given as the input for the full-domain distribution inference are the cliques of the graphical model. The goal is to find the parameters in the factor functions such that the loss of marginal distributions is minimized while the entropy of the full-domain distribution is maximized.

To achieve this goal, the first step is to find a valid set of marginals such that the loss between the found marginals and the noisy marginals are minimized, and then the factor parameters can be derived from the marginals using the maximum entropy rule. A set of marginals is valid if there exists a full-domain distribution such that its marginals match the set of marginals. The space of such valid marginals form a marginal polytope. The optimization problem is thus to find the minimum loss on this marginal polytope. The loss is defined as the negative log-likelihood in [124], where the likelihood is about how likely the differentially private mechanism generates the noisy result given a specific marginal. In [124], the optimization problem is solved

by an algorithm based on entropic mirror descent algorithm, and further improved by Nesterov's accelerated dual averaging approach. This algorithm also generate the factor parameters at the same time. To generate a synthetic data set, one can sample from the graphical model using the inferred factor parameters.

Learning a Markov network from the noisy marginals not only provides a tractable expression of the full-domain distribution, but also ensure consistency and some accuracy for the further inference of the distribution. However, it is unknown whether a data distribution can always be factorized using a Markov network, and it is also unknown whether the maximum entropy rule is the best option to infer a Markov network given a set of marginal distributions.

### 5.3.2 Multiplicative Weights Exponential Mechanism

Suppose we are given some initial guess to the full-domain distribution, such as a uniform distribution, and we are also given a query result of $q$, which maps each tuple in the data to $[-1, +1]$ and sums up, how should we update our initial guess of the full-domain distribution according to what we have observed? One approach is to update the weight of each value in the domain using the multiplicative weight rule. The query answer that is released is always based on the latest synthetic data set.

Maintaining a synthetic data set or distribution in this way is also an approach to release a private synthetic data set. Here we consider an offline setting as discussed in MWEM [85]. If all the queries are given at once instead of arriving one by one, we can cherry-pick the query to update the distribution such that otherwise the error of that query is the max among all queries without the update. This query selection is done through the Exponential mechanism, where the score function is the L1 distance between the query answer on the current distribution and the ground truth. Once

a query is selected, the query is answered based on the ground truth data using the Laplace mechanism and the distribution is updated using the multiplicative rule. The algorithm will then repeat again to select a new query and make a new update. If there are in total $T$ such updates, then the privacy budget is divided into $2T$ pieces, one for the EM mechanism and one for the Laplace mechanism in each update. The final distribution or the synthetic data set is the average of all past distribution in the updates. Suppose the domain size of data is $|D|$, the pool size of queries is $|Q|$, the data size is $n$ and the total privacy budget is $\epsilon$, then there exists a $T$ such that with probability at least $1 - 1/ploy(|Q|)$, the error of any query answered by the final synthetic data set is $O\left(n^{2/3}\left(\frac{\log|D|\log|Q|}{\epsilon}\right)\right)$

Since the update of distribution is a pure post-processing step, one can apply the multiplicative weight update multiple times using the existing sanitized query answers. The initial guess of the distribution can also be replaced by some public knowledge or another private mechanism. The final distribution can also be replaced by the final updated distribution instead of the averaged one. These variants of the algorithm may improve the performance of MWEM in a practical sense.

# Chapter 6

# Conclusion

In this thesis we study the problem of answering and explaining SQL queries privately. Relational databases and SQL queries are prevalent and important for data collection and processing. Applying differential privacy to SQL queries allows the possibility of wide application of differential privacy to the real world. We specifically study a non-trivial subset of SQL queries: join-count queries. We propose efficient algorithms to compute the local sensitivity of join-count queries, which lies the core to existing solutions of answering join-count queries privately. We also design new algorithms for this problem based on the analysis to the local sensitivity. On the other hand, due to the introduction of differential privacy, the query results are no longer a simple reflection of the database, and thus it is important to explain the query results to the user to help user understand how differential privacy affects the results. To tackle this problem, we propose a formal framework to explain query results under differential privacy.

## 6.1 Summary of Contributions

We summarize the contributions as follows:

*Local Sensitivity of Join-Count Queries.* In Chapter 3, we analyze the complexity of computing the local sensitivity of a join-count query, which has a negative result, but we also successfully develop algorithms to compute the local sensitivity efficiently for path join queries and 'doubly-acyclic' queries. We also develop a truncation algorithm to bound the global sensitivity of a join-count query, which allows adding Laplace noise possible to satisfy DP and shows empirical advancements than the existing

solutions.

*Explaining Query Results Under Differential Privacy.* In Chapter 4, we study the problem of explaining differentially private query results, and adapt the traditional approach of query explanation to comply with differential privacy. Together, we propose a first-of-its-kind framework based on three phases: private query answering, question validation and private explanation table. It explains a user question over a group-by aggregate query answer (COUNT, SUM, or AVG) under differential privacy. We successfully adapt the influence function of explanation predates from the existing work [166] to a novel form, but augmented to reduce sensitivity. We also propose novel algorithms to compute the confidence interval of rank accurately, which has high sensitivity by nature.

## 6.2 Future Work

We propose several future directions that worth exploration.

*Answering general SQL queries privately.* In the real world applications, SQL queries could be much more complex than a single SELECT - FROM - WHERE - GROUP BY form. It could involve sub-queries which incur correlations between the sub-queries and the root query, self-joins, user-defined functions, having clauses, other operators which includes union, delete and so on. Supporting a larger class of SQL queries under differential privacy with a guaranteed utility is the key to apply differential privacy to the real world with a wider impact.

*Explaining general SQL queries privately.* In this work, we only consider simple mechanisms to answer a basic class of SQL queries, group-by aggregation (COUNT, SUM or AVG) queries without sub-queries and joins. However, there exists multiple advanced mechanism to answer these queries. For example, [88] collected a list of algorithms for answering histogram and range count queries. Answering other aggre-

gations under differential privacy, e.g. quantiles, are also studied [164]. It remains an open question to explain an arbitrary mechanism to the user.

# Appendix A

# Appendix

## A.1 Theorems and Proofs

### A.1.1 Influence Function

**Proposition 4.4.4.** *[Influence Function Sensitivity] Given an explanation predicate $p$ and an user question with respect to a group-by query with aggregation $agg$, the following holds:*

1. *If $agg = COUNT$, the sensitivity of $\mathrm{I}(p)$ is 4.*
2. *If $agg = SUM$, the sensitivity of $\mathrm{I}(p)$ is 4 $A_{agg}^{max}$.*
3. *If $agg = AVG$, the sensitivity of $\mathrm{I}(p)$ is 16 $A_{agg}^{max}$.*

*Proof. (1) CNT.* Recall the definition of influence function:

$$\begin{aligned}\mathrm{I}(p; Q, D) = &\Big( (q(g_i(D)) - q(g_j(D))) \\ &- (q(g_i(\neg p(D))) - q(g_j(\neg p(D)))) \Big) \\ &\times \frac{\min\limits_{t \in \{i,j\}} |g_t(\neg p(D))|}{\max\limits_{t \in \{i,j\}} |g_t(D)| + 1}\end{aligned}$$

We interpret and consider the following equations or notations:

$$
\begin{aligned}
q(D) &= |D| \\
\phi_i &= (\phi \wedge A_{gb} = \alpha_i) \\
g_i(D) &= \phi_i(D) \\
g_i(p(D)) &= (\phi_i \wedge p)(D) \\
g_i(\neg p(D)) &= (\phi_i \wedge \neg p)(D) \\
f(D) &= \min_{t \in \{i,j\}} |g_t(\neg p(D))| \\
g(D) &= \max_{t \in \{i,j\}} |g_t(D)| + 1 \\
h_i(D) &= q((\phi_i \wedge p)(D)) f(D)/g(D)
\end{aligned}
$$

Since $q$ is linear [1] , we have $q(g_i(D)) - q(g_i(\neg p(D))) = q(g_i(p(D)))$, and by replacing $g_i(p(D))$ with $(\phi_i \wedge p)(D)$ we have $q(g_i(D)) - q(g_i(\neg p(D))) = q((\phi_i \wedge p)(D))$. By replacing $q(g_i(D)) - q(g_i(\neg p(D)))$ with $q((\phi_i \wedge p)(D))$ and the last numerator and denominator in the influence function by $f(D)$ and $g(D)$, we have $\mathrm{I}(p; Q, D) = h_i(D) - h_j(D)$.

We prove the sensitivity bound by the following inequality chains.

$$\Delta_{\mathrm{I}} = max_{D \approx D'} |\mathrm{I}(p; Q_{CNT}, D) - \mathrm{I}(p; Q_{CNT}, D'| \tag{A.1}$$

We first replace I according to $\mathrm{I}(p; Q, D) = h_i(D) - h_j(D)$, and then apply Lemma A.1.5 to bound the sensitivity by the sum of sensitivities of $h_i$ and $h_j$.

$$\leqslant \sum_{t \in \{i,j\}} \max_{|D'| = |D|+1} |h_t(D') - h_t(D)| \tag{A.2}$$

The second inequality is by Lemma A.1.8, since $f$ is a non-negative query with sensitivity 1 and $g$ is a monotonic positive and positive query with sensitivity 1.

[1] A query $q$ is linear if it can be expressed as $q(D) = \sum_{t \in D} q(t)$.

$$\leqslant \sum_{t\in\{i,j\}} \frac{2|(\phi_t \wedge p)(D)| + f(D) + 1}{g(D)}\Delta_q \tag{A.3}$$

The next equality is by replacing the variables. Since $q$ is a counting query, it has sensitivity $\Delta_q = 1$.

$$= \sum_{t\in\{i,j\}} \frac{2|(\phi_t \wedge p)(D)| + \min\limits_{s\in\{i,j\}} |(\phi_s \wedge \neg p)(D)| + 1}{\max_{s\in\{i,j\}}|g_s(D)| + 1} \tag{A.4}$$

The third inequality is by the property of $min$ and $max$, since $\min\limits_{s\in\{i,j\}} |(\phi_s \wedge \neg p)(D)| \leqslant |(\phi_t \wedge \neg p)(D)|$ and $\max_{s\in\{i,j\}}|g_s(D)| \geqslant |g_t(D)|$.

$$\leqslant \sum_{t\in\{i,j\}} \frac{|(\phi_t \wedge p)(D)| + |(\phi_t \wedge p)(D)| + |(\phi_t \wedge \neg p)(D)| + 1}{|g_t(D)| + 1} \tag{A.5}$$

The next equality is due to that $\phi_t = (\phi_t \wedge p) \vee (\phi_t \wedge \neg p)$.

$$= \sum_{t\in\{i,j\}} \frac{|(\phi_t \wedge p)(D)| + (|\phi_t(D)| + 1)}{|g_t(D)| + 1} \tag{A.6}$$

The fourth inequality is due to that $|(\phi_t \wedge p)(D)| \leqslant |\phi_t(D)| = |g_t(D)| \leqslant |g_t(D)| + 1$.

$$\leqslant \sum_{t\in\{i,j\}} \frac{(|g_t(D)| + 1) + (|g_t(D)| + 1)}{|g_t(D)| + 1} \tag{A.7}$$

$$\leqslant 4 \tag{A.8}$$

*(2) SUM.* Similar to the proof of the sensitivity of $CNT$ influence, but with $\Delta_q = A_{agg}^{max}$, which should be replaced at Equation (A.3).

*(3) AVG.*

$$
\begin{aligned}
&\mathrm{I}(p; Q_{AVG}, D) \\
=&\Bigg( (\frac{SUM(\phi_i(D), A_{agg})}{|\phi_i(D)|} - \frac{SUM(\phi_j(D), A_{agg})}{|\phi_j(D)|}) - \\
&(\frac{SUM((\phi_i \wedge \neg p)(D), A_{agg})}{|(\phi_i \wedge \neg p)(D)|} - \frac{SUM((\phi_j \wedge \neg p)(D), A_{agg})}{|(\phi_j \wedge \neg p)(D)|}) \Bigg) \\
&\min_{t \in \{i,j\}} |(\phi_t \wedge \neg p)(D)|
\end{aligned}
$$

Now we consider decompose this query into four parts (for example, $\frac{SUM(\phi_i(D), A_{agg})}{|\phi_i(D)|} \min_{t \in \{i,j\}} |(\phi_t \wedge \neg p)(D)|$ as one part) , and analyze the sensitivity for each part and finally sum up. Consider query $q$ as summing up $A_{agg}$ with sensitivity $\Delta_q = A_{agg}^{max}$. By Lemma A.1.8, we can show that the sensitivity of each part is 4 $\Delta_q$. Together, the total sensitivity is bounded by 16 $\Delta_q$. □

### A.1.2 Private Explanations

**Proposition 4.4.6.** *Given an influence function* I *with sensitivity* $\Delta_{\mathrm{I}}$*, a set of explanation predicates* $\mathcal{P}$*, a privacy parameter* $\rho_{Topk}$ *and a size parameter* $k$*, the following holds:*

1. *One-shot Top-k mechanism finds k explanation predicates while satisfying* $\rho_{Topk}$*-zCDP.*
2. *Denote by* $OPT^{(i)}$ *the i-th highest (true) influence, and by* $\mathcal{M}^{(i)}$ *the i-th explanation predicate selected by the One-shot Top-k mechanism. For* $\forall t$ *and* $\forall i \in \{1, 2, \ldots, k\}$*, we have*

$$
Pr\left[ \mathrm{I}(\mathcal{M}^{(i)}) \leqslant OPT^{(i)} - \frac{2\Delta_{\mathrm{I}}}{\sqrt{8\rho_{Topk}/k}} (\ln(|\mathcal{P}|) + t) \right] \leqslant e^{-t} \tag{4.5}
$$

*Proof. (1) Differential Privacy.* it is equivalent to iteratively applying $k$ exponential mechanisms [69] that satisfies $\epsilon^2/8$-zCDP [63, 59, 33, 141] for each, where

$\epsilon = \sqrt{8\rho_{Topk}/k}$ [63, 62], therefore in total it satisfies $(k\epsilon^2/8)$-zCDP which is also $\rho_{Topk}$-zCDP.

*(2) Utility Bound.* It is extended from the utility theorem of EM in Thm 3.11 of [69], which states that

$$Pr\left[\mathrm{I}(\mathcal{M}^{(1)}) \leqslant OPT^{(1)} - \frac{2\Delta_{\mathrm{I}}}{\epsilon}(\ln(|\mathcal{P}|) + t)\right] \leqslant e^{-t}$$

where $\epsilon = \sqrt{8\rho_{Topk}/k}$. To extend from $i = 1$ to $\forall i \in \{1, 2, \ldots, k\}$, we follow the original proof:

$$Pr[\mathrm{I}(\mathcal{M}^{(i)}) \leqslant c] \leqslant \frac{|\mathcal{P}| \exp(\epsilon c/(2\Delta_{\mathrm{I}}))}{\exp(\epsilon OPT^{(i)}/(2\Delta_{\mathrm{I}}))}$$

by giving a upper bound and lower bound of the numerator and denominator. Replacing $c$ with the appropriate value will give this theorem. □

**Proposition 4.4.8.** *Given a database $D$, a set of predicates $\mathcal{P}$, an influence function* I *with global sensitivity $\Delta_{\mathrm{I}}$, and an integer $1 \leqslant t \leqslant |\mathcal{P}|$,* $\mathrm{I}(\mathrm{rank}^{-1}(t))$ *has global sensitivity $\Delta_{\mathrm{I}}$.*

*Proof.* Drop $\mathcal{P}$ and I from $\mathrm{rank}^{-1}(t; D, \mathcal{P}, \mathrm{I})$ for simplicity. Next we show that for any two neighboring datasets $D' \sim D$, we have $|\mathrm{I}(\mathrm{rank}^{-1}(t; D'); D') - \mathrm{I}(\mathrm{rank}^{-1}(t; D); D)| \leqslant \Delta_{\mathrm{I}}$, which is equivalent to showing $-\Delta_{\mathrm{I}} \leqslant \mathrm{I}(\mathrm{rank}^{-1}(t; D'); D') - \mathrm{I}(\mathrm{rank}^{-1}(t; D); D) \leqslant \Delta_{\mathrm{I}}$.

*Case 1, lower bound.* This is to show that for any $D' \approx D$, we have $\mathrm{I}(\mathrm{rank}^{-1}(t; D'); D') - \mathrm{I}(\mathrm{rank}^{-1}(t; D); D) \geqslant -\Delta_{\mathrm{I}}$.

By the definition of global sensitivity, for any explanation predicate $p$, we have $|\mathrm{I}(p; D') - \mathrm{I}(p; D)| \leqslant \Delta_{\mathrm{I}}$, and therefore $\mathrm{I}(p; D') \geqslant \mathrm{I}(p; D) - \Delta_{\mathrm{I}}$. By replacing $p$ with $\mathrm{rank}^{-1}(j; D)$ for some $j$, we have $\mathrm{I}(\mathrm{rank}^{-1}(j; D); D') \geqslant \mathrm{I}(\mathrm{rank}^{-1}(j; D); D) - \Delta_{\mathrm{I}}$. For any $j \leqslant t$, by the property of ranking , we have $\mathrm{I}(\mathrm{rank}^{-1}(j; D); D) \geqslant$

$\mathrm{I}(\mathrm{rank}^{-1}(t;D);D)$. Together, for any $j \leqslant t$, we have $\mathrm{I}(\mathrm{rank}^{-1}(j;D);D') \geqslant \mathrm{I}(\mathrm{rank}^{-1}(j;D);D) - \Delta_{\mathrm{I}} \geqslant \mathrm{I}(\mathrm{rank}^{-1}(t;D);D) - \Delta_{\mathrm{I}}$. This means there are at least $t$ elements in $D'$ such that their scores are above $\mathrm{I}(\mathrm{rank}^{-1}(t;D);D) - \Delta_{\mathrm{I}}$, which implies for the $t$-th largest score in $D'$ we have $\mathrm{I}(\mathrm{rank}^{-1}(t;D');D') \geqslant \mathrm{I}(\mathrm{rank}^{-1}(t;D);D) - \Delta_{\mathrm{I}}$.

*Case 2, upper bound.* This is to show that for any $D' \approx D$, we have $\mathrm{I}(\mathrm{rank}^{-1}(t;D');D') - \mathrm{I}(\mathrm{rank}^{-1}(t;D);D) \leqslant \Delta_{\mathrm{I}}$.

By the definition of global sensitivity, for any explanation predicate $p$, we have $|\mathrm{I}(p;D') - \mathrm{I}(p;D)| \leqslant \Delta_{\mathrm{I}}$, and therefore $\mathrm{I}(p;D') \leqslant \mathrm{I}(p;D) + \Delta_{\mathrm{I}}$. By replacing $p$ with $\mathrm{rank}^{-1}(j;D)$ for some $j$, we have $\mathrm{I}(\mathrm{rank}^{-1}(j;D);D') \leqslant \mathrm{I}(\mathrm{rank}^{-1}(j;D);D) + \Delta_{\mathrm{I}}$. For any $j \geqslant t$, by the property of ranking , we have $\mathrm{I}(\mathrm{rank}^{-1}(j;D);D) \leqslant \mathrm{I}(\mathrm{rank}^{-1}(t;D);D)$. Together, for any $j \geqslant t$, we have $\mathrm{I}(\mathrm{rank}^{-1}(j;D);D') \leqslant \mathrm{I}(\mathrm{rank}^{-1}(j;D);D) + \Delta_{\mathrm{I}} \leqslant \mathrm{I}(\mathrm{rank}^{-1}(t;D);D) + \Delta_{\mathrm{I}}$. This means there are at most $t-1$ elements in $D'$ such that their scores can be above $\mathrm{I}(\mathrm{rank}^{-1}(t;D);D) + \Delta_{\mathrm{I}}$, which implies for the $t$-th largest score in $D'$ we have $\mathrm{I}(\mathrm{rank}^{-1}(t;D');D') \leqslant \mathrm{I}(\mathrm{rank}^{-1}(t;D);D) + \Delta_{\mathrm{I}}$. □

**Lemma A.1.1.** *Given a set of predicates $\mathcal{P}$, an influence function* I *with global sensitivity $\Delta_{\mathrm{I}}$ and a number $t$, then the function $s(D) = \mathrm{I}(p;D) - \mathrm{I}(\mathrm{rank}^{-1}(t;D,\mathcal{P},\mathrm{I});D)$ has global sensitivity $2\Delta_{\mathrm{I}}$.*

*Proof.* The sensitivity of I is $\Delta_{\mathrm{I}}$ by definition and the sensitivity of $\mathrm{I}(\mathrm{rank}^{-1}(t;D,\mathcal{P},\mathrm{I})$ is $\Delta_{\mathrm{I}}$ by Proposition 4.4.8. By Lemma A.1.5, together it has sensitivity $2\Delta_{\mathrm{I}}$. □

**Theorem 4.4.10.** *Given a database $D$, a predicate space $\mathcal{P}$, an influence function* I *with sensitivity $\Delta_{\mathrm{I}}$, $k$ explanation predicates $p_1, p_2, \ldots, p_k$, a confidence level $\gamma$, and a privacy parameter $\rho_{Rank}$, Algorithm 4 returns a list of confidence intervals $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k$ such that*

1. *Algorithm 4 satisfies $\rho_{Rank}$-zCDP.*

2. *For* $\forall u \in \{1, 2, \ldots, k\}$, $\mathcal{I}_u$ *is a* $\gamma$ *level confidence interval of* $\text{rank}(p_u)$.

3. *For* $\forall u \in \{1, 2, \ldots, k\}$ *and* $\forall x \geqslant 0$, *the confidence interval* $\mathcal{I}_u = (\mathcal{I}_u^L, \mathcal{I}_u^U)$ *satisfies* $Pr[A \leqslant \text{I}(p_u) \leqslant B] \geqslant 1 - 2e^{-x}$ *where* $A = \text{I}(\text{rank}^{-1}(\mathcal{I}_u^L)) - (|\xi_{-1}| + \sigma_{-1}\sqrt{2(x + \ln N)})$ *and* $B = \text{I}(\text{rank}^{-1}(\mathcal{I}_u^U)) + (|\xi_{+1}| + \sigma_{+1}\sqrt{2(x + \ln N)})$.

*Proof. (1) Differential Privacy.* Now we discuss why Algorithm 4 satisfies $\rho_{Rank}$-zCDP.

The main structure of Algorithm 4 is a for-loop of $k$ explanation predicates from line 15 to 18, and within each for-loop, we first prepare the parameters at line 16 and 16, make two calls to the sub-routine RankBound and construct the confidence interval by the sub-routine outputs. We first show below that each call to the sub-routine RankBound with parameters $(p, \rho, \beta, dir)$ satisfies $\rho$-zCDP. Given this is true, we then show that our two calls $\text{RankBound}(p_u, 0.1\rho, \beta, -1)$ and $\text{RankBound}(p_u, 0.9\rho, \beta, +1)$ at Line 17 satisfies $0.1\rho$-zCDP and $0.9\rho$-zCDP, which together satisfies $\rho$-zCDP by the composition rule (Lemma A.1.3). By line 16, we set $\rho = \rho_{Rank}/k$, therefore each loop satisfies $(\rho_{Rank}/k)$-zCDP, and after in total $k$ loops, it satisfies $\rho_{Rank}$-zCDP by the composition rule (Lemma A.1.3).

Next we show that $\text{RankBound}(p, \rho, \beta, dir)$ satisfies $\rho$-zCDP, from line 1 to 14. We first prepare some parameters at the start of the sub-routine, which does not touch the data, and then enters a while loop with at most $N = \lceil \log_2 |\mathcal{P}| \rceil$ loops. Denote $s = \text{I}(p) - \text{I}(\text{rank}^{-1}(t))$. Within each loop, we add a Gaussian noise to a secret $s$ at line 8. The value of $s$ touches the sensitive data, but by adding a Gaussian noise to $s$, the release of $\hat{s}$ satisfies zCDP. By Theorem 4.2.8, with noise scale $\sigma$, it satisfies $(\Delta_q^2)/2\sigma^2$-zCDP where $\Delta_q$ is the sensitivity of the function that we want to release. Since we set $\sigma = (2\Delta_{\text{I}})/\sqrt{2(\rho/N)}$ at line 3 and the sensitivity of $s$ is $2\Delta_{\text{I}}$ by Lemma A.1.1, it satisfies $(\rho/N)$-zCDP. Since we have at most $N$ noisy releases of $S$ using the Gaussian mechanism, by composition rule (Lemma A.1.3), the entire while

loop satisfies $\rho$-zCDP, and so is the sub-routine.

*(2) Confidence Interval.* Now we discuss that the confidence interval outputted from the sub-routine RankBound$(p, \rho, \beta, dir)$, from line 1 to 14, is a $\gamma$-level confidence interval.

The sub-routine RankBound with direction as upper is mirror to the sub-algorithm RankBound with direction as lower. We first show that RankBound returns a bound in either upper or lower case such that it is a true bound with probability $\beta = \frac{\gamma+1}{2}$, therefore the target rank is within two bounds with probability $\gamma$. We give the proof for the case when direction is upper for the sub-algorithm RankBound, and skip the proof for the case when direction is lower due to the similarity.

The sub-routine RankBound is a random binary search algorithm with in total $N$ loops. To ensure that the final $t_{high}$ is a rank bound, one sufficient condition is that $t_{high}$ is always an upper bound of rank during all the loops. Recall that in the noisy binary search, in each loop we first find $t$ as the middle of $t_{high}$ and $t_{low}$, check $s = \mathrm{I}(p) - \mathrm{I}(\mathrm{rank}^{-1}(t)) \leqslant 0$, add noise a Gaussian noise to $s$ to get $\hat{s}$ and compare $\hat{s}$ with margin, which is $\xi$ in this case. If $\hat{s} \geqslant \xi$, notice that at line 9, we change $t_{high}$ to $t$. If in this case, $s \leqslant 0$, which means $t$ is not an upper bound of rank, we never have chance to make $t_{high}$ to be a valid upper bound of rank since it will only decrease in the further loops. Therefore, We say a loop is a failure if during that loop, $s \leqslant 0$ but $\hat{s} > \xi$. To have a valid rank upper bound, it is necessary to have no loop failure during the entire noisy binary search. We next show that the probability of no such a failure occur is at least $\beta$.

See the chain of inequalities below.

$$Pr[\mathcal{I}_u^U \text{ is an upper bound of rank}(p_u; D, \mathcal{P}, I)] \tag{A.9}$$

The first inequality is due to the bound of the number of while loops. To be a rank bound, it cannot fail at each loop, therefore it has to success for all the $N$ loops. These are independent events, so we can use a product for all the events happen together.

$$\geqslant (1 - Pr[\text{loop failure}])^N \tag{A.10}$$

The second inequality is due to the bound of $Pr[\text{loop failure}]$. Since any case such that $S \leqslant 0$ but $\hat{s} > \xi$ is considered as a loop failure, $\hat{s}$ is achieved by adding a Gaussian noise to $s$ and $\xi$ is a constant, the probability of a loop failure only depends on the value of $s$. Since here we have a condition about $s \leqslant 0$, $\sup_{s \leqslant 0} Pr[\hat{s} > \xi]$ is an upper bound of $Pr[\text{loop failure}]$.

$$\geqslant (1 - \sup_{s \leqslant 0} Pr[\hat{S} > \xi])^N \tag{A.11}$$

The next equality is because $\sup_{s \leqslant 0} Pr[\hat{s} > \xi] = Pr[N(0, \sigma^2) > \xi]$. Recall that $\hat{s} = s + N(0, \sigma^2)$ in line 8, therefore $\sup_{s \leqslant 0} Pr[\hat{s} > \xi] = \sup_{s \leqslant 0} Pr[s + N(0, \sigma^2) > \xi] = \sup_{s \leqslant 0} Pr[N(0, \sigma^2) > \xi - s]$. Since $Pr[N(0, \sigma^2) > \xi - s]$ increases as $s$ increases, it achieves maximum at $s = 0$ for $s \leqslant 0$. Therefore, $\sup_{s \leqslant 0} Pr[\hat{s} > \xi] = Pr[N(0, \sigma^2) > \xi]$.

$$= Pr[\mathcal{N}(0, \sigma^2) \leqslant \xi]^N \tag{A.12}$$

The third bound is due to Chernoff bound of the Q-function (Lemma A.1.2). Since $Pr[\mathcal{N}(0, \sigma^2) \leqslant \xi] = 1 - Pr[\mathcal{N}(0, 1) > \xi/\sigma]$, by Chernoff bound we have $Pr[\mathcal{N}(0, 1) > \xi/\sigma] \leqslant \exp(-(\xi/\sigma)^2/2)$ and therefore $Pr[\mathcal{N}(0, \sigma^2) \leqslant \xi] \geqslant 1 - \exp(-(\xi/\sigma)^2/2)$.

$$\geqslant \left(1 - \exp(-(\xi/\sigma)^2/2)\right)^N \tag{A.13}$$

The fourth bound is due to $(1+x)^r \geqslant 1 + rx$ for $x \geqslant -1$ and $r \geqslant 1$.

$$\geqslant 1 - N\exp(-(\xi/\sigma)^2/2) \tag{A.14}$$

The final equality is by plugging $\xi = \sigma\sqrt{2\ln(N/(1-\beta))}$.

$$= \beta \tag{A.15}$$

Similarly, we have $Pr[\mathcal{I}_u^L$ is a lower bound of $\text{rank}(p_u; D, \mathcal{P}, I)] \geqslant \beta$. Together, the probability of $\mathcal{I}_u$ is a $\gamma$ level confidence interval of $\text{rank}(p_u; D, \mathcal{P}, I)$ equals to both events $\mathcal{I}_u^U$ is an upper bound of $\text{rank}(p_u; D, \mathcal{P}, I)$ and $\mathcal{I}_u^L$ is a lower bound of $\text{rank}(p_u; D, \mathcal{P}, I)$ happen together, which is greater than or equal to the probability sum of each single event minus one (Lemma A.1.7, which is $\beta + \beta - 1 = 2\beta - 1$. By plugging $\beta = (\gamma+1)/2$ from line 16, we have $2\beta - 1 = \gamma$, which is the confidence interval level for the final confidence interval.

*(3) Utility Bound.* We first show that each utility bound has probability $\geqslant 1 - e^{-x}$, then use the union probability rule to show together it is bounded by $\geqslant 1 - 2e^{-x}$.

Consider the upper utility bound. One sufficient condition for the upper utility bound to be true is that $\text{I}(\text{rank}^{-1}(\mathcal{I}_{rank}^R; D, \mathcal{P}, \text{I}); D) \geqslant \text{I}(p, D) - (\xi + \sigma\sqrt{2(x + \ln N)})$ is always true. Similar to the proof of confidence rank bound, here we say a loop is a failure if $S > \xi + \sigma\sqrt{2(x+\ln N)}$ but $\hat{S} \leqslant \xi$. The proof of the inequality chain below is similar to the proof of confidence interval, except for that the third inequality is

due to that $(1-a)^x \geqslant 1-ax$ for $a \in (0,1)$ and $x \geqslant 1$.

$$
\begin{aligned}
& Pr[\mathrm{I}(\mathrm{rank}^{-1}(\mathcal{I}^R_{rank}; D, \mathcal{P}, \mathrm{I})) - \sigma(\xi + \sqrt{2(x+\ln N)})] \\
\geqslant & (1 - Pr[\text{loop failure}])^N \\
\geqslant & (1 - \sup_{S > \xi + \sigma\sqrt{2(x+\ln N)}} Pr[\hat{S} \leqslant \xi])^N \\
= & (1 - Pr[N(0,\sigma^2) \leqslant -\sigma\sqrt{2(x+\ln N)}])^N \\
\geqslant & 1 - N Pr[N(0,\sigma^2) \leqslant -\sigma\sqrt{2(x+\ln N)}] \\
\geqslant & 1 - N\exp(-(\sqrt{2(x+\ln N)})^2/2) \\
= & 1 - \exp(-x)
\end{aligned}
$$

□

### A.1.3 Supplementary

**Lemma A.1.2** (Chernoff bound of Q function)**.** *Given a Q function: $Q(x) = Pr[X > x]$, where $X \sim N(0,1)$ is a standard Gaussian distribution, if $x \geqslant 0$, we have*

$$Q(x) \leqslant \exp(-x^2/2)$$

*Proof.* By Chernoff bound, we have $Pr[X > x] \leqslant E[e^{tX}]/e^{tx}$ for any $t \geqslant 0$. By the property of Gaussian distribution, we have $E[e^{tX}] = e^{t^2/2}$. Together, we have $Pr[X > x] \leqslant e^{t^2/2 - tx}$. Since $x \geqslant 0$, we can choose $t = x$, and have $Pr[X > x] \leqslant e^{-t^2/2}$. □

**Lemma A.1.3** (Composition [30])**.** *Let $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}$ and $\mathcal{M}' : \mathcal{X}^n \to \mathcal{Z}$ be randomized algorithms. Suppose $\mathcal{M}$ satisfies $\rho$-zCDP and $\mathcal{M}'$ satisfies $\rho'$-zCDP. Define $\mathcal{M}'' : \mathcal{X}^n \to \mathcal{Y} \times \mathcal{Z}$ by $\mathcal{M}''(x) = (\mathcal{M}(x), \mathcal{M}'(x))$. Then $\mathcal{M}''$ satisfies $(\rho + \rho')$-zCDP.*

**Lemma A.1.4** (Postprocessing [30]). *Let $\mathcal{M} : \mathcal{X}^n \to \mathcal{Y}$ and $f : \mathcal{Y} \to \mathcal{Z}$ be randomized algorithms. Suppose $\mathcal{M}$ satisfies $\rho$-zCDP. Define $\mathcal{M}' : \mathcal{X}^n \to \mathcal{Z}$ by $\mathcal{M}'(x) = f(\mathcal{M}(x))$. Then $\mathcal{M}'$ satisfies $\rho$-zCDP.*

**Lemma A.1.5.** *Given two functions $f$ ang $g$ with sensitivities $\Delta_f$ and $\Delta_g$, the sum of two functions have sensitivity $\Delta_f + \Delta_g$*

*Proof.* BY definition, we have $\max_{D \approx D'} |f(D) - f(D')| \leqslant \Delta_f$ and $\max_{D \approx D'} |g(D) - g(D')| \leqslant \Delta_g$. Therefore, $\max_{D \approx D'} |(f(D) + g(D)) - (f(D') + g(D'))| = \max_{D \approx D'} |(f(D) - f(D') + (g(D) - g(D'))| \leqslant \max_{D \approx D'} |(f(D) - f(D')| + \max_{D \approx D'} |(g(D) - g(D'))| = \Delta_f + \Delta_g$. The inequality is due to the property of absolute. □

**Lemma A.1.6** (Gaussian Confidence Interval [163]). *Given a Gaussian random variable $Z \sim N(\mu, \sigma^2)$ with unknown location parameter $\mu$ and known scale parameter $\sigma$. Let $\mathcal{I}^L = Z - \sigma\sqrt{2}\,\mathrm{erf}^{-1}(\gamma)$ and $\mathcal{I}^U = Z + \sigma\sqrt{2}\,\mathrm{erf}^{-1}(\gamma)$, then $\mathcal{I} = (\mathcal{I}^L, \mathcal{I}^U)$ is a $\gamma$ level confidence interval of $\mu$.*

*Proof.* By Theorem 6.16 from the text book [163]. □

**Lemma A.1.7.** *Given events $A_1, A_2, \ldots, A_\ell$, the following inequality holds:*

$$Pr[\bigwedge_{i=1}^{\ell} A_i] \geqslant \sum_{i=1}^{\ell} Pr[A_i] - (\ell - 1)$$

*Proof.* First we show that given events $A$ and $B$, we have $Pr[A \wedge B] \geqslant Pr[A] + Pr[B] - 1$ since $1 \geqslant Pr[A \vee B] = Pr[A] + Pr[B] - Pr[A \wedge B]$. Next we show that

$$Pr[\bigwedge_{i=1}^{\ell} A_i] \geqslant Pr[\bigwedge_{i=1}^{\ell-1} A_i] + Pr[A_\ell] - 1$$

using the previous rule. This gives a recursive expression and can be reduced to the final formula in the lemma. □

**Lemma A.1.8.** *Given a COUNT or SUM query $q$ with sensitivity $\Delta_q$, a predicate $\phi$, a non-negative query $f : \mathcal{D} \to \mathbb{N}_0$ with sensitivity 1 and another monotonic* [2] *and positive query $g : \mathcal{D} \to \mathbb{N}^+$ with sensitivity 1. Denote $h(D)$ as*

$$h(D) = q(\phi(D))\frac{f(D)}{g(D)}$$

*For any two neighboring datasets $D$ and $D'$ such that $|D'| = |D| + 1$, we have*

$$|h(D') - h(D)| \leqslant \frac{2|\phi(D)| + f(D) + 1}{g(D)}\Delta_q$$

*Proof.* Denote $x = q(\phi(D))$, $x' = q(\phi(D'))$, $n = |\phi(D)|$. Since $x$ is the aggregation over tuples from $\phi(D)$ and $x$ has sensitivity $\Delta_q$, we have $|x| \leqslant n\Delta_q$ . Denote $\delta_x = x' - x$. Since $x$ has sensitivity $\Delta_q$, we have $|\delta_x| \leqslant \Delta_q$. Since $g(D)$ is monotonic and has sensitivity 1, we have $g(D) \leqslant g(D') \leqslant g(D) + 1$. Since $f$ has sensitivity 1, we have $|f(D) - f(D')| \leqslant 1$.

$$\begin{aligned}
&|h(D') - h(D)| \\
=&|x'\frac{f(D')}{g(D')} - x\frac{f(D)}{g(D)}| \\
=&|(x + \delta_x)\frac{f(D')}{g(D')} - x\frac{f(D)}{g(D)}| \\
=&|x\left(\frac{f(D')}{g(D')} - \frac{f(D)}{g(D)}\right) + \delta_x\frac{f(D')}{g(D')}|
\end{aligned}$$

Now we divide into two cases depending on the sign of the factor of x in the formula above.

[2] A query $q$ is monotonic if for any two databases $D'$ and $D$ such that $|D'| \geqslant |D|$, we have $q(D') \geqslant q(D)$.

**Case 1, the factor of x is non-negative.**

$$
\begin{aligned}
&|h(D') - h(D)| \\
\leqslant &n\Delta_q \left( \frac{f(D')}{g(D')} - \frac{f(D)}{g(D)} \right) + \Delta_q \frac{f(D')}{g(D')} \\
= &\left[ (n+1)\frac{f(D')}{g(D')} - n\frac{f(D)}{g(D)} \right] \Delta_q \\
\leqslant &\left[ (n+1)\frac{f(D)+1}{g(D)} - n\frac{f(D)}{g(D)} \right] \Delta_q \\
\leqslant &\frac{f(D)+n+1}{g(D)} \Delta_q
\end{aligned}
$$

**Case 2, the factor of x is non-positive.**

$$
\begin{aligned}
&|h(D') - h(D)| \\
\leqslant &n\Delta_q \left( \frac{f(D)}{g(D)} - \frac{f(D')}{g(D')} \right) + \Delta_q \frac{f(D')}{g(D')} \\
\leqslant &\left[ n \left( \frac{f(D')+2}{g(D')} - \frac{f(D')}{g(D')} \right) + \frac{f(D')}{g(D')} \right] \Delta_q \\
\leqslant &\frac{2n+f(D')}{g(D')} \Delta_q \\
\leqslant &\frac{2n+f(D)+1}{g(D)} \Delta_q
\end{aligned}
$$

In conclusion, $|h(D') - h(D)| \leqslant \frac{2n+f(D)+1}{g(D)} \Delta_q$. □

# A.2 Extensions

## A.2.1 Confidence Interval of Question

In this section, we elaborate the algorithm of Section 4.4.1 in the form of pseudo codes.

*Confidence interval for COUNT and SUM.* In Algorithm 5, at line 2, we set the noise scale $\sigma$ according to aggregation as $COUNT$ ($SUM$), and at line 7 and 8, we set the confidence interval from the standard properties of Gaussian distribution by a margin as $\sqrt{2}(\sqrt{2}\sigma)\,\mathrm{erf}^{-1}(\gamma)$ for both bounds [3] [163] .

*Confidence interval for AVG.* In Algorithm 5, at line 10, we set the sub confidence level $\beta = 1 - (1-\gamma)/4$ for each individual confidence interval, so that the final confidence level for $o_i - o_j$ is $\gamma$. At line 11 and 12, we set the noise level $\sigma$ for $SUM$ and $COUNT$. From line 13 to 18, we extract all the intermediate numerators and denominators, and construct individual confidence intervals. At line 19 and 20, we compute the infimum and supremum of the image of the cross product of individual confidence intervals, which is also the confidence interval at level $\gamma$.

### A.2.2 Private Top-k Explanation Predicates

In this section, we restate the One-shot Top-k mechanism based on the exponential mechanism [69] from Section 4.4.3 with pseudo codes.

Given a score function $\mathrm{s} : \mathcal{P} \to \mathcal{R}$ that maps an explanation predicate $p$ to a number, the exponential mechanism (EM) [69] randomly samples $p$ from $\mathcal{P}$ with probability proportional to $exp(\epsilon\,\mathrm{s}(p)/(2\Delta_{\mathrm{s}}))$ with some privacy parameter $\epsilon$ and satisfies $(\epsilon^2/8)$-zCDP [63, 59, 33, 141]. The higher the score is, the more possible that an explanation predicate is selected. In DPXPlain, we use the influence function as the score function.

We denote the exponential mechanism as $\mathcal{M}_E$. To find 'top-$k$' explanation predicate satisfying DP, we can first apply $\mathcal{M}_E$ to find one explanation predicate, remove it from the entire explanation predicate space, and then apply $\mathcal{M}_E$ again until $k$ explanation predicates are found. It was shown by previous work [63, 62] that this

[3] $\mathrm{erf}^{-1}$ is the inverse function of the error function erf.

**Algorithm 5** Compute Confidence Interval of User Question

**Input:** A user question $Q = (\alpha_i, >, \alpha_j)$ with respect to the query SELECT $A_{gb}$, agg($A_{agg}$) FROM R WHERE $\phi$ GROUP BY $A_{gb}$, the noisy results $\hat{o}_i$ and $\hat{o}_j$, the privacy budget $\rho_q$ for the private query answering, and the confidence level $\gamma$.

**Output:** A $\gamma$-level confidence interval of $o_i - o_j$.

**if** agg $= COUNT$ or agg $= SUM$ **then**
  **if** agg $= COUNT$ **then**
    $\sigma \leftarrow 1/\sqrt{2\rho_q}$
  **else if** agg $= SUM$ **then**
    $\sigma \leftarrow A_{agg}^{max}/\sqrt{2\rho_q}$
  **end if**
  $\mathcal{I}^L \leftarrow \hat{o}_i - \hat{o}_j - 2\sigma\,\mathrm{erf}^{-1}(\gamma)$
  $\mathcal{I}^U \leftarrow \hat{o}_i - \hat{o}_j + 2\sigma\,\mathrm{erf}^{-1}(\gamma)$
**else if** agg $= AVG$ **then**
  $\beta \leftarrow 1 - (1-\gamma)/4$
  $\sigma_S \leftarrow A_{agg}^{max}/\sqrt{2\rho_q/2}$
  $\sigma_C \leftarrow 1/\sqrt{2\rho_q/2}$
  **for** $t \in \{i, j\}$ **do** /* *Recall that* $\hat{o}_t = \hat{o}_t^S/\hat{o}_t^C$ */
    $\hat{o}_t^S \leftarrow$ numerator of $\hat{o}_t$.
    $\mathcal{I}_t^S \leftarrow (\hat{o}_t^S - \sigma_S\sqrt{2}\,\mathrm{erf}^{-1}(\beta), \hat{o}_t^S + \sigma_S\sqrt{2}\,\mathrm{erf}^{-1}(\beta))$
    $\hat{o}_t^C \leftarrow$ denominator of $\hat{o}_t$.
    $\mathcal{I}_t^C \leftarrow (\hat{o}_t^C - \sigma_C\sqrt{2}\,\mathrm{erf}^{-1}(\beta), \hat{o}_t^C + \sigma_C\sqrt{2}\,\mathrm{erf}^{-1}(\beta))$
  **end for**
  $\mathcal{I}^L \leftarrow \inf\{\mathcal{I}_i^S / \mathcal{I}_i^C - \mathcal{I}_j^S / \mathcal{I}_j^C\}$
  $\mathcal{I}^U \leftarrow \sup\{\mathcal{I}_i^S / \mathcal{I}_i^C - \mathcal{I}_j^S / \mathcal{I}_j^C\}$
**end if**
$\mathcal{I} \leftarrow (\mathcal{I}^L, \mathcal{I}^U)$
**return** $\mathcal{I}$

process is identical to adding i.i.d. Gumbel noise[4] to each score and releasing the top-$k$ predicates by the noisy scores (i.e., there is no need to remove predicates after sampling) . We, therefore, use this result to devise a similar solution that is presented in Algorithm 6. In line 1, we set the noise scale. In lines 2–4, we randomly sample Gumbel noise with scale $\sigma$ and add it to the influence of each explanation predicate from the space $\mathcal{P}$. In line 5, we sort the noisy scores in the descending order, and in line 6, we find the top-k explanation predicates by their noisy scores. This algorithm satisfies $\rho_{Topk}$-zCDP (as formally stated in Proposition 4.4.6), and can be applied to questions on SUM, COUNT, or AVG queries, with different score functions and sensitivity values for different aggregates.

**Algorithm 6** Noisy Top-k Predicates [63]

**Input:** An influence function I with sensitivity $\Delta_{\mathrm{I}}$, a set of explanation predicates $\mathcal{P}$, a privacy parameter $\rho_{Topk}$ and a size parameter $k$.
**Output:** Top-k explanation predicates.

1: $\sigma \leftarrow 2\Delta_{\mathrm{I}}\sqrt{k/(8\rho_{Topk})}$
2: **for** $u \leftarrow 1 \dots |\mathcal{P}|$ **do**
3: $\quad s_u \leftarrow \mathrm{I}(p_u) + Gumbel(\sigma)$
4: **end for**
5: Sort $s_1 \dots s_{|\mathcal{P}|}$ in the descending order.
6: Let $p_1, p_2, \dots, p_k$ be the top-k elements in the list.
7: **return** $p_1, p_2, \dots, p_k$

### A.2.3 Private Confidence Interval of Influence

Algorithm 7 takes a privacy budget $\rho_{Influ}$ as input. In Line 2 we divide the privacy budget $\rho_{Influ}$ into $k$ equal portions for each explanation predicate $p_u$ for $u \in \{1, \dots, k\}$. In Line 3, we calibrate the noise scale according to the sensitivity of the influence function. In Line 10, we add a Gaussian noise to the influence $\mathrm{I}(p_u)$ of explanation predicate $p_u$, and finally in Lines 11 and 12, we derive the confidence interval based

[4]For a Gumbel noise $Z \sim Gumbel(\sigma)$, its CDF is $Pr[Z \leqslant z] = \exp(-\exp(-z/\sigma))$.

on the Gaussian property [163].

**Algorithm 7** Compute Confidence Interval of Influence

**Input:** An influence function I with respect to the question $(\alpha_i, >, \alpha_j)$, $k$ explanation predicates $p_1, p_2, \ldots, p_k$, a private database $D$, a privacy budget $\rho_{Influ}$, and a confidence level $\gamma$.

**Output:** A list of $\gamma$-level confidence intervals of the influence $\mathrm{I}(p_u)/(\hat{o}_i - \hat{o}_j)$ for $u \in \{1, 2, \ldots, k\}$.

**for** $u \in \{1, 2, \ldots, k\}$ **do**
  $\rho \leftarrow \rho_{Influ}/k$
  **if** agg $= COUNT$ **then**
    $\sigma \leftarrow 4/\sqrt{2\rho}$
  **else if** agg $= SUM$ **then**
    $\sigma \leftarrow 4A_{agg}^{max}/\sqrt{2\rho}$
  **else if** agg $= AVG$ **then**
    $\sigma \leftarrow 16A_{agg}^{max}/\sqrt{2\rho}$
  **end if**
  $\hat{\mathrm{I}} \leftarrow \mathrm{I}(p_u) + N(0, \sigma^2)$
  $\mathcal{I}_u^L \leftarrow \hat{\mathrm{I}} - \sqrt{2}\sigma\,\mathrm{erf}^{-1}(\gamma)$
  $\mathcal{I}_u^U \leftarrow \hat{\mathrm{I}} + \sqrt{2}\sigma\,\mathrm{erf}^{-1}(\gamma)$
  $\mathcal{I}_u \leftarrow (\mathcal{I}_u^L, \mathcal{I}_u^U)$
**end for**
**return** $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_k$

### A.2.4 Influence Function Monotonicity

The influence function $\mathrm{I}(p)$ is not monotone w.r.t. predicate implication even without the normalizing factor in the function. We demonstrate this property in the example below.

**Example A.2.1.** *Start with a database with three binary attributes: $A, B, C$ and two tuples: (0, 0, 0), (1, 0, 1). Consider an agg $= COUNT$ query with group by on $A$, so we have $agg(g_0(D)) = 1$ and $agg(g_1(D)) = 1$ for two groups $A = 0$ and $A = 1$. Consider three explanation predicates for the user question $(\alpha_0, \alpha_1, >)$ (note that the noisy values can be different from the true values): $p_1 : B = 0$, $p_2 : B = 0 \wedge C = 0$ and*

$p_3 : B = 0 \wedge C = 1$, *which satisfy* $p_2 \Rightarrow p_1$ *and* $p_3 \Rightarrow p_1$. *However, while* $\mathrm{I}(p_1) = 0$, *we have* $\mathrm{I}(p_2) = 1$ *and* $\mathrm{I}(p_3) = -1$, *i.e.,* $\mathrm{I}(p_3) < \mathrm{I}(p_1) < \mathrm{I}(p_2)$.

### A.2.5 General User Question

In this section, we introduce a general form of user question through weighted sum, such that more groups can be involved in the question and the comparison between groups can be more flexible. This covers the cases of the original questions, since a single group difference can also be treated as a weighted sum between two groups. We also discuss how the explanation framework should be adapted to this general form. Finally, we give a use case for privately explaining a general user question.

**Definition A.2.2** (General User Question)**.** *Given a database $D$ an aggregate query $q$, a DP mechanism $\mathcal{M}$, and noisy group aggregation releases $\hat{o}_{i_1}, \hat{o}_{i_2}, \ldots, \hat{o}_{i_m}$ of the groups $\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_m}$ from the query $q$, a general user question $Q$ is represented by $m$ weights and a constant $c$: $(w_{i_1}, w_{i_2}, \ldots, w_{i_m}, c)$. Intuitively, the question is interpreted as "Why $\sum_{j=i_1}^{i_m} w_j \hat{o}_j \geqslant c$".*

Definition A.2.2 allows more interesting questions, such as "Why the total salary of group A and B is larger than the total salary of group C and D?" or "Why the average salary of group A is 10 times larger than the one of group B?". Next we illustrate how the algorithms for each problem related to our framework should be adapted in the case of general user question.

*Private Confidence Interval of Question.* Given a general user question $(w_{i_1}, w_{i_2}, \ldots, w_{i_m}, c)$, we discuss how to derive the confidence interval of $\sum_{j=i_1}^{i_m} w_j o_j - c$. Comparing to the case of a simple user question $(\alpha_i, >, \alpha_j)$, where the target of confidence interval is $o_i - o_j$, here we have a weighted sum of multiple group results. Therefore, when $agg$ is $CNT$ or $SUM$, the noisy weighted sum follows the Gaussian

distribution with scale $\sqrt{\sum_{j=i_1}^{i_m} w_j^2}\sigma$, where $\sigma$ is the noise scale used in query answering. When $agg$ is $AVG$, the noisy weighted sum can also be viewed as a combination of multiple Gaussian variables. In conclusion, we consider the adaptaions as follows:

1. For $agg = CNT$ or $agg = SUM$, update the margin $\sqrt{2}(\sqrt{2}\sigma)\,\mathrm{erf}^{-1}(\gamma)$ as $\sqrt{2}(\sqrt{\sum_{j=i_1}^{i_m} w_j^2}\sigma)\,\mathrm{erf}^{-1}(\gamma)$.

2. For $agg = AVG$, update the sub confidence level $\beta$ to be $(\gamma - 1)/(2m) + 1$ , and the image of sub confidence intervals to be $\sum_{j=i_1}^{i_m} \mathcal{I}_j^S / \mathcal{I}_j^C - c$ .

*Private Top-k Explanation Predicates.*

Since the user question has a new form, the influence function and its corresponding score function should also be adapted. We consider their natural extensions as follows:

**Definition A.2.3** (General Influence Function)**.** *Given a database $D$ and a general user question $Q = (w_{i_1}, w_{i_2}, \ldots, w_{i_m}, c)$ with respect to the query* SELECT $A_{gb}$, agg($A_{agg}$) FROM R WHERE $\phi$ GROUP BY $A_{gb}$*, the influence of an explanation predicate $p$ is defined follows:*

$$\mathrm{I}(p; Q, D) = \left( \sum_{j=i_1}^{i_m} w_j q(g_j(D)) - \sum_{j=i_1}^{i_m} w_j q(g_j(\neg p(D))) \right) \times \begin{cases} \dfrac{\min\limits_{t \in \{i_1, i_2, \ldots, i_m\}} |g_t(\neg p(D))|}{\max\limits_{t \in \{i_1, i_2, \ldots, i_m\}} |g_t(D)| + 1} & , agg \in \{COUNT, SUM\} \\ \min\limits_{t \in \{i_1, i_2, \ldots, i_m\}} |g_t(\neg p(D))| & , agg = AVG \end{cases}$$

We can plug-in the new influence function into algorithm 6 to find the noisy top-k explanation predicates. The corresponding sensitivity of the new influence function is given as follows:

**Theorem A.2.4.** *[General Influence Function Sensitivity] Given an explanation predicate $p$ and a general user question $Q = (w_{i_1}, w_{i_2}, \ldots, w_{i_m}, c)$ with respect to a group-by query with aggregation $agg$, the following holds:*

1. *If $agg = CNT$, the sensitivity of $\mathrm{I}(p; Q, D)$ is $2\sum_{j=i_1}^{i_m} |w_j|$.*

2. *If $agg = SUM$, the sensitivity of $\mathrm{I}(p; Q, D)$ is $2\sum_{j=i_1}^{i_m} |w_j| A_{agg}^{max}$.*

3. *If $agg = AVG$, the sensitivity of $\mathrm{I}(p; Q, D)$ is $8\sum_{j=i_1}^{i_m} |w_j| A_{agg}^{max}$.*

*Proof.* It is a weighted version of Proposition 4.4.4. □

We also allow explanation predicates to include disjunction and allow the framework to specify a specific set of explanation predicates by enumeration.

*Private Confidence Interval of Influence.* We can plug-in the new influence function and their sensitivities into the original algorithm to find the confidence interval of influence.

*Private Confidence Interval of Rank.* We can plug-in the new influence function and their sensitivities into algorithm 4 to find the confidence interval of rank.

*Use Case: Taxi-Imbalance.* We consider the New York City taxi trips dataset [2] in January and February, 2019, as a use case. We preprocessed the dataset such that it includes 4 columns: PU_Zone, PU_Borough, DO_Zone, DO_Borough. In this case we analyze the traffic volume between boroughs. With privacy budget $\rho_q = 0.1$, the framework answers the user query as "SELECT PU_Borough, DO_Borough, CNT(*) FROM R GROUP BY PU_Borough, DO_Borough". There are in total 49 groups, and among the query answers we have *(Brooklyn, Queens): 11,431* and *(Queens, Brooklyn): 121,934*. User then asks "Why Queens to Brooklyn has more than 10 times the number of trips from Brooklyn to Queens?" This corresponds to the question "why $q_1 - 10q_2 \geqslant 0$", or in the form of weights $(1, -10, 0)$. The confidence

interval of the question is $(7580, 7668)$, which validates the question. To explain the question, we consider a predicate space of the form "PU_Zone = ¡zone¿ $\vee$ DO_Zone = ¡zone¿" with in total 127 different zones. With $\rho_{Topk} = 0.025$, $\rho_{Influ} = 0.025$, and $\rho_{Rank} = 0.95$, we have the explanation table as shown in fig. A.1. The relative influence is relative to the noisy difference $\hat{o}_1 - 10\hat{o}_2 = 7624$. From this table, we can find that two airports, JFK and LaGuardia airports that are located in Queens, are the major reasons for why there are more traffic volume from Queens to Brooklyn since there are more incoming taxi traffic to the airports instead of outgoing taxi traffic.

| explanation predicate | Rel Influ 95%-CI | | Rank 95%-CI | |
|---|---|---|---|---|
| | L | U | L | U |
| zone = "JFK Airport" | 55.21% | 72.18% | 1 | 1 |
| zone = "LaGuardia Airport" | 28.75% | 45.72% | 1 | 3 |
| zone = "Bay Ridge" | -6.64% | 23.60% | 3 | 127 |
| zone = "Queensboro Hill" | -10.75% | 6.22% | 3 | 127 |
| zone = "Flushing" | -12.52% | 4.25% | 3 | 127 |

**Figure A.1**: Top-5 explanations for Taxi-Imbalance.

### A.2.6 Finding Top-k by Arbitrary Influence Function

In the noisy binary search of algorithm 4, we use the difference between $\mathrm{I}(p; D)$ and $\mathrm{I}(\sigma_{top((,t)}; D, \mathcal{P}, I); D)$ as an indicator for each branch. The utility of this algorithm depends on the global sensitivity of the influence function. However, given a complex influence function, the sensitivity of influence could be unbounded. For example, given a target query as $q_f(D) = q_1(D)/q_2(D)$ and $dir = high$, our influence function is $I(p; D) = (1 - |p(D)|/|D|)(q_1(D)/q_2(D) - q_1(\neg p(D))/q_2(\neg p(D)))$. In this case, one can always find $p$ and $D$ and $D'$ such that the absolute difference between $I(p; D)$

and $I(p; D')$ is arbitrary high. A typical work around is to bound the ranges of basic queries; however, it introduces bias and may destroy the ranking order. Moreover, the bound needs to be chosen without looking the data, which makes it even more impossible.

On the other hand, the difference between $I(p; D)$ and $I(\sigma_{top((,t)}; D, \mathcal{P}, I); D)$ is not the only choice of branch indicator. Denote $S$ as a function of the form $S(p, t; D, \mathcal{P}, I)$. In general, if this function satisfies three properties as listed in the theorem below, which are also the only properties that the proof of Theorem 4.4.10 requires, using this function as the branch indicator in algorithm 4 still allows this algorithm to satisfy $\rho$-zCDP and the guarantee of confidence interval of rank.

**Theorem A.2.5.** *Substituting* $\mathrm{I}(p) - \mathrm{I}(\mathrm{rank}^{-1}(t))$ *by* $S(p, t; D, \mathcal{P}, I)$ *and* $\sigma$ *by* $\Delta_S/\sqrt{2(\rho/N)}$ *for algorithm 4, the new algorithm satisfies* $\rho$*-zCDP and outputs correct confidence intervals of rank if the following holds for* $S$ *and* $\Delta_S$*:*

- *Center Zero.* $S(p, \mathrm{rank}(p; D, \mathcal{P}, I); D, \mathcal{P}, I) = 0$
- *Non-Decreasing. For any* $i < j$, $S(p, i; D, \mathcal{P}, I) \leqslant S(p, j; D, \mathcal{P}, I)$.
- *Stable. For any two neighboring datasets* $D \approx D'$, $|S(p, t; D, \mathcal{P}, I) - S(p, t; D', \mathcal{P}, I)| \leqslant \Delta_S$.

A natural choice of $S$ is to define $S(p, t; D, \mathcal{P}, I) = I(p; D) - I(\sigma_{top((,t)}; D, \mathcal{P}, I); D)$, the difference between the influence of $p$ and the $t$-th largest influence. With the "Center Zero" and "Non-Decreasing" properties, the indicator function $S$ can tell that a number $t$ is a rank bound of $\mathrm{rank}(p; D, \mathcal{P}, I)$ if $S(p, t; D, \mathcal{P}, I) > 0$. If $i$ and $j$ are both rank bound of $\mathrm{rank}(p; D, \mathcal{P}, I)$ and $i$ is closer to the target rank than $j$, $S(p, i; D, \mathcal{P}, I)$ is also closer to 0 than $S(p, j; D, \mathcal{P}, I)$. However, for the natural choice of $S$, sensitivity $\Delta_S = 2\Delta_I$ and $\Delta_I$ could be unbounded for some $I$, which results in

poor utility. Instead, we can define $S$ in a way such that it still reflects the difference between the influence of $p$ and the $t$-th largest influence, but has low sensitivity.

Inspired by inverse sensitivity and other techniques that share the same spirit [16, 15, 49, 149], we present a stable branch indicator function $S(p, t; D, \mathcal{P}, I)$ such that it is approximately the least number of tuples that need to be changed to move the rank of $p$ beyond $t$. Specially, when $t \leqslant \text{rank}(p; D, \mathcal{P}, I)$, $S(p, t; D, \mathcal{P}, I) = 0$.

Denote $D \triangle D'$ as the symmetric difference between two datasets $D$ and $D'$. Denote influence lower bound $ILB(p, d; D, I) = \inf\{I(p; D') \mid |D' \triangle D| \leqslant d\}$ the least influence of $p$ and influence upper bound $IUB(p, d; D, I) = \sup\{I(p; D') \mid |D' \triangle D| \leqslant d\}$ the largest influence of $p$ within distance $d$ to $D$. Given two predicates $p$ and $\tilde{p}$, if $IUB(\tilde{p}, d; D, I) < ILB(p, d; D, I)$, it indicates that there is no dataset $D'$ within distance $d$ to $D$ such that the influence of $\tilde{p}$ is higher than or equal to the one of $p$. Denote the complementary size of such predicate $\tilde{P}$ in $\mathcal{P}$ as $B(p, d; D, \mathcal{P}, I) = |\mathcal{P}| - |\{\tilde{p} \in \mathcal{P} \mid IUB(\tilde{p}, d; D, I) < ILB(p, d; D, I)\}|$. This gives a rank bound of $\text{rank}(p; D', \mathcal{P}, I)$ for any dataset $D'$ such that $|D \Delta D'| \leqslant d$.

**Example A.2.6.** *Suppose $\mathcal{P}$ has 5 predicates $p_1$, $p_2$, $p_3$, $p_4$ and $p_5$. Now we show $B(p_3, 2)$. Suppose at distance 2, $IUB(p_1, 2) = 1$, $IUB(p_2, 2) = 3$, $IUB(p_3, 2) = 6$, $IUB(p_4, 2) = 7$, $IUB(p_5, 2) = 10$, and $ILB(p_3, 2) = 4$. In this case, $B(p_3, 2) = 5 - 2 = 3$ since predicate $p_1$ and $p_2$ have lower $IUB$ than the $ILB$ of $p_3$. This indicates, by adding or removing 2 tuples from $D$, the rank of $p_3$ cannot be beyond 3.*

**Lemma A.2.7.** *Given a predicate $p$, a dataset $D$, a set of predicates $\mathcal{P}$ and an influence function $I$, for any dataset $D'$ such that $|D \Delta D'| \leqslant 1$ and any distance $d$, we have:*

$$B(p, d; D, \mathcal{P}, I) \leqslant B(p, d + 1; D', \mathcal{P}, I) \tag{A.16}$$

*Proof.* Denote $\mathcal{D}_1 = \{D'' || D'' \Delta D| = d\}$ and $\mathcal{D}_2 = \{D'' || D'' \Delta D'| = d + 1\}$. Notice that $B(p, d; D, \mathcal{P}, I)$ (or $B(p, d + 1; D', \mathcal{P}, I)$) is counting the complementary size

of predicate $\tilde{p}$ in $\mathcal{P}$ such that no dataset $D''$ in $\mathcal{D}_1$ (or $\mathcal{D}_2$) satisfies $I(\tilde{p}; D'') \geqslant I(p; D'')$. Since $|D \Delta D'| \leqslant 1$, we have $\mathcal{D}_1 \subseteq \mathcal{D}_2$, therefore $B(p, d; D, \mathcal{P}, I) \leqslant B(p, d + 1; D', \mathcal{P}, I)$. □

If $IUB$ is a loose influence upper bound and $ILB$ is a loose influence lower bound, the lemma above still holds.

**Table A.1**: Example of $B$

| d | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $B(p, d; D)$ | 2 | 2 | 4 | 6 | 10 |
| $B(p, d; D')$ | 2 | 3 | 5 | 7 | 8 |

**Definition A.2.8.** *Given a predicate p, a dataset $D$, a set of predicates $\mathcal{P}$ and an influence function $I$, $\omega$ is a stable branch indicator function as*

$$\omega(p, t; D, \mathcal{P}, I) = \min\{d \geqslant 0 \mid B(p, d; D, \mathcal{P}, I) \geqslant t\}$$

**Theorem A.2.9.** *Given a predicate p, a dataset $D$, a set of predicates $\mathcal{P}$ and an influence function $I$, $\omega$, the three conditions of theorem A.2.5 is satisfied if function $S = \omega$ and sensitivity $\Delta_S = 1$.*

*Proof.*

**Center Zero.** $B(p, 0; D, \mathcal{P}, I) = \text{rank}(p; D, \mathcal{P}, I)$.

**Non-Decreasing.** Since $B$ is non-decreasing in terms of $d$ given $D, \mathcal{P}, I$, $\omega_t(D)$ is also non-decreasing.

**Stable.** Drop $\mathcal{P}, I$ for simplicity. Suppose t is fixed. Denote $d^* = \omega(p, t; D)$. By definition, we have $B(p, d^*; D) \geqslant t$. For any neighboring dataset $D' \sim D$, since $B(p, d^*; D) \leqslant B(p, d^* + 1; D')$, it indicates $B(p, d^* + 1; D') \geqslant t$ and thus we have $\omega(p, t; D') \leqslant d^* + 1$.

When $d^* < 2$, which means $\omega(p, t; D) < 2$, it is impossible to have $\omega(p, t; D') - \omega(p, t; D) < -1$ since $\omega(p, t; D') \geqslant 0$ is always true. When $d^* \geqslant 1$, we show that

it is impossible to have $B(p, d^* - 2; D') \geqslant t$. If $B(p, d^* - 2; D') \geqslant t$, we have $B(p, d^* - 1; D) \geqslant B(p, d^* - 2; D') \geqslant t$, which indicates $\omega(p, t; D) \leqslant d^* - 1$ and leads to a contradiction. Therefore, we have $B(p, d^* - 2; D') < t$, which indicates the impossibility of $\omega(p, t; D') \leqslant d^* - 2$. Therefore, we have $\omega(p, t; D') \geqslant d^* - 1$.

Since $d^* - 1 \leqslant \omega(p, t; D') \leqslant d^* + 1$, we have $|\omega(p, t; D) - \omega(p, t; D')| \leqslant 1$. □

**Table A.2**: Example of $\omega$

| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ... |
|---|---|---|---|---|---|---|---|---|
| $\omega(p, t)$ | 0 | 0 | 2 | 2 | 3 | 3 | 10 | ... |

The branch indicator function $\omega$ finds the minimum $d$ such that $B(p, d; D, \mathcal{P}, I) \geqslant t$. If we add a constraint $d \geqslant C$ with some constant $C$, the theorem above still holds.

### A.2.7 Large Domain Private Top-k Selection

Algorithm 8 gives a practical version for find top-k elements given an score function from a large domain. It assumes that the domain $\mathcal{P}$ is partitioned into an active domain $\mathcal{P}_{act}$ and an idle domain $\mathcal{P}_{idle}$, such that the elements in the idle domain all have the same score $C$. We assume a random draw from the idle domain $\mathcal{P}_{idle}$ could be done in $O(1)$, so the runtime of the algorithm only depends on the size of the active domain $\mathcal{P}_{act}$ as $O(k|\mathcal{P}_{act}|)$. This algorithm satisfies $k\epsilon^2/8$-zCDP or $k\epsilon$-DP.

### A.2.8 Computing Confidence Interval of General Arithmetic Combinations

Formally, a query $q$ can be expressed as an arithmetic combination of queries if it can be expressed as $q(D) = f(q_1(D), q_2(D), \ldots, q_\ell(D))$ where function $f$ includes the operators in $\{+, -, *, /, \exp, \log\}$ and for each sub-query $q_i$, a noisy answer $\hat{s}_i =$

**Algorithm 8** Report Noisy Top-k Elements from a Large Domain

**Input:** A private dataset $D$, an active domain $\mathcal{P}_{act}$ of predicate class $\mathcal{P}$, an idle domain $\mathcal{P}_{idle}$ of predicate class $\mathcal{P}$, a score function $u$, global sensitivity of $u$ as $\Delta_u$, a constant $C$ as the score for any element from the idle domain $\mathcal{P}_{idle}$, and a privacy parameter $\epsilon$.

**Output:** Top-k predicates ordered by scores.

Compute $u(p, D)$ for every $p \in \mathcal{P}_{act}$ without releasing the results.
**for** $i \leftarrow 1 \dots k$ **do**
    $s \leftarrow -\infty$
    **for** $p \leftarrow$ iterate the space of $\mathcal{P}_{act}$ **do**
        $s' \leftarrow u(p, D) + Gumbel(2\Delta_u/\epsilon)$
        **if** $s' > s$ **then**
            $p_{r_i} \leftarrow p$
            $s \leftarrow s'$
        **end if**
    **end for**
    $s' \leftarrow \frac{2\Delta_u}{\epsilon} \ln(|\mathcal{P}_{idle}) + C + Gumbel(2\Delta_u/\epsilon)$
    **if** $s' > s$ **then**
        $\hat{p}_i \leftarrow$ a random draw from $\mathcal{P}_{idle}$
    **end if**
    **if** $\hat{p}_i \in \mathcal{P}_{act}$ **then**
        $\mathcal{P}_{act} \leftarrow \mathcal{P}_{act} \backslash \{\hat{p}_i\}$
    **else**
        $\mathcal{P}_{idle} \leftarrow \mathcal{P}_{idle} \backslash \{\hat{p}_i\}$
    **end if**
**end for**
**return** $(\hat{p}_1, \hat{p}_2, \dots, \hat{p}_k)$.

$N(q_i(D), \sigma_i^2)$ is released under $\rho$-zCDP [5], where $\sigma_i = \Delta_{q_i}/\sqrt{2\rho/\ell}$ and $\Delta_{q_i}$ is the sensitivity for sub-query $q_i$. The rest of this sub section discusses how to derive the confidence interval for $q(D)$ based on the noisy releases $\hat{s}_1, \hat{s}_2, \dots, \hat{s}_\ell$ and function $f$.

Given the noisy releases $\hat{s}_1$, $\hat{s}_2$, ..., $\hat{s}_\ell$ through the Gaussian mechanism, the confidence intervals of $q_1(D)$, $q_2(D)$, ..., $q_\ell(D)$ can be derived by Gaussian confidence interval , and there is a clear connection between these queries to $q(D)$ through function $f$. Therefore, we can compute the image of these confidence intervals through

[5] Although user doesn't see the intermediate differentially private query result, we assume the framework stores them.

the function $f$, which is also a valid confidence interval for $q_f(D)$. Given a function $f : X \to Y$, denote by $f[A]$ the image of $f$ under $A \subseteq X$ i.e. $f[A] = \{f(a) : a \in A\}$.

**Theorem A.2.10.** *Given a database $D$ and a query $q$ that can be expressed as $q_f(D)$ $= f(q_1(D), q_2(D), \ldots, q_\ell(D))$, where $f$ includes the operators in $\{+, -, *, /, \exp, \log\}$, and confidence intervals at confidence level $\beta$ for $q_1(D), q_2(D), \ldots, q_\ell(D)$ as $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell$. Let $\mathcal{I} = f[\mathcal{I}_1 \times \mathcal{I}_2 \times \ldots \times \mathcal{I}_\ell]$ be the image of $\mathcal{I}_1 \times \mathcal{I}_2 \times \ldots \times \mathcal{I}_\ell$ under $f$, i.e., the set of numbers composed of each mapping of $f$ for a combination of values from $\mathcal{I}_1 \times \mathcal{I}_2 \times \ldots \times \mathcal{I}_\ell$. Also assume that $f$ is defined for each vector in $\mathcal{I}_1 \times \mathcal{I}_2 \times \ldots \times \mathcal{I}_\ell$. Let $\mathcal{I}^L = \inf I$ and $\mathcal{I}^U = \sup I$. We have*

$$Pr[\mathcal{I}^L \leqslant q_f(D) \leqslant \mathcal{I}^U] \geqslant \ell(\beta - 1) + 1$$

*Proof.* Since the event $\bigwedge_{i=1}^{\ell} q_i(D) \in \mathcal{I}_i$ implies $q_f(D) \in \mathcal{I}$ due to $\mathcal{I}$ is the image of $\mathcal{I}_1 \times \mathcal{I}_2 \times \ldots \times \mathcal{I}_\ell$ under $f$, we have $Pr[q_f(D) \in \mathcal{I}] \geqslant Pr[\bigwedge_{i=1}^{\ell}(q_i(D) \in \mathcal{I}_i)]$. Secondly, by Lemma A.1.7 and by definition about $Pr[q_i(D) \in \mathcal{I}_i] \geqslant \beta$ for $\forall i$, we have $Pr[\bigwedge_{i=1}^{\ell}(q_i(D) \in \mathcal{I}_i)] \geqslant \sum_{i=1}^{\ell} Pr[q_i(D) \in \mathcal{I}_i] - (\ell - 1) \geqslant \ell(\beta - 1) + 1$. Thirdly, since $\mathcal{I}^L = \inf I$ and $\mathcal{I}^U = \sup I$, we have $I \subseteq [\mathcal{I}^L, \mathcal{I}^U]$, and therefore $Pr[\mathcal{I}^L \leqslant q_f(D) \leqslant \mathcal{I}^U] \geqslant Pr[q_f(D) \in \mathcal{I}]$. Together, we have $Pr[\mathcal{I}^L \leqslant q_f(D) \leqslant \mathcal{I}^U] \geqslant Pr[q_f(D) \in \mathcal{I}] \geqslant Pr[\bigwedge_{i=1}^{\ell}(q_i(D) \in \mathcal{I}_i)] \geqslant \ell(\beta - 1) + 1$.

□

Although it might not be obvious to find the analytical form of the image, we can use numerical methods to find the approximations of the supremum and infimum of the image. The width of such a interval is not determined.

Given a query $q$ decomposed by function $f$ and the noisy answers, algorithm 9 summarizes the approach for deriving the confidence interval for $q(D)$. We first derive the confidence interval of each sub-query (line 2 to 4) with confidence level $\beta = 1 - \frac{1-\gamma}{\ell}$ (line 1) and finally compute the confidence interval for $q(D)$ (line 5 and 6).

**Algorithm 9** Image-based Confidence Interval

**Input:** A query $q$ such that $q(D) = f(q_1(D), q_2(D), \ldots, q_\ell(D))$, noisy answers $\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_\ell$ of queries $q_1, q_2, \ldots, q_\ell$ using Gaussian mechanisms with scales $\sigma_1$, $\sigma_2$, $\ldots$, $\sigma_\ell$, confidence level $\gamma$.

$\beta \leftarrow 1 - (1-\gamma)/\ell$
**for** $i \in 1 \ldots \ell$ **do**
    $\mathcal{I}_i = (\hat{s}_i - \sigma_i\sqrt{2}\,\mathrm{erf}^{-1}(\beta), \hat{s}_i + \sigma_i\sqrt{2}\,\mathrm{erf}^{-1}(\beta))$
**end for**
$\mathcal{I}^L \leftarrow \inf_{x_i \in \mathcal{I}_i\, \forall x_i} f(x_1, x_2, \ldots, x_\ell)$
$\mathcal{I}^U \leftarrow \sup_{x_i \in \mathcal{I}_i\, \forall x_i} f(x_1, x_2, \ldots, x_\ell)$
$\mathcal{I}_q = (\mathcal{I}^L, \mathcal{I}^U)$
**return** $\mathcal{I}_q$.

### A.2.9 Bootstrap Confidence Interval

**Algorithm 10** Bootstrap Confidence Interval [73]

**Input:** A query $q$ such that $q(D) = f(q_1(D), q_2(D), \ldots, q_\ell(D))$, noisy answers $\hat{o}_1, \hat{o}_2, \ldots, \hat{o}_\ell$ of queries $q_1, q_2, \ldots, q_\ell$ using Gaussian mechanisms with scales $\sigma_1$, $\sigma_2$, $\ldots$, $\sigma_\ell$, confidence level $\gamma$, and a bootstrap step size $B$.

**Output:** A confidence interval for $q_f(D)$ at confidence level $\gamma$.

**for** $b \leftarrow 1 \ldots B$ **do**
    **for** $i \leftarrow 1 \ldots \ell$ **do**
        $o_i^* \leftarrow \hat{o}_i + N(0, \sigma_i^2)$
    **end for**
    $\theta_b^* \leftarrow f(o_1^*, o_2^*, \ldots, o_\ell^*)$
**end for**
$\hat{\theta} \leftarrow f(\hat{o}_1, \hat{o}_2, \ldots, \hat{o}_\ell)$
$z_0 = \Phi^{-1}(\frac{1}{B}\sum_{b=1}^{B} \mathbb{1}_{\theta_b^* < \hat{\theta}})$
$\mathcal{I}^L \leftarrow \min\{s | \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}_{\theta_b^* < s} \geqslant \Phi(2z_0 + \Phi^{-1}(\frac{1-\gamma}{2})\}$
$\mathcal{I}^U \leftarrow \max\{s | \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}_{\theta_b^* < s} \leqslant \Phi(2z_0 + \Phi^{-1}(\frac{1+\gamma}{2})\}$
$\mathcal{I}_q = (\mathcal{I}^L, \mathcal{I}^U)$
**return** $\mathcal{I}_q$.

Bootstrap is an old yet powerful technique started by Bradley Efron in late 70's [72] which can be used for computing the confidence interval of an unknown statistic. Although it is not an exact confidence interval, it enjoys a theoretical guarantee on the correctness of the approximation. Traditional bootstrap assumes there are multiple

samples that is samples from a unknown distribution with the parameters of interest. Here the parameters of interest is $q_1(D)$ to $q_\ell(D)$, the distribution is a multivariate Gaussian distribution, and we only observe one sample from it. Therefore, we apply the method introduced from [73] to construct an confidence interval for $q(D)$, which consider the similar problem of finding the confidence interval of $q(D)$ with one observation of noisy $q_1(D)$ to $q_\ell(D)$.

In section 5 of [73], it describes a parametric bootstrap. The main idea is to assume the observation is from a parametric distribution, use the observation to infer the parameters using maximum likelihood estimate, resample from the estimated distribution, and use bias-corrected percentile method to construct a confidence interval. Algorithm 10 illustrates an application of parametric bootstrap confidence interval for $q(D)$.

The most related work to this private confidence interval problem that are also based on bootstrap are [77, 44], which construct the CI that encodes the randomness from both sampling and noise, while we only consider the randomness from noise. Traditional CI is closely related to sampling, which assumes some population parameters and data is sampled according to those parameters, therefore the population parameter can be inferred from the sampled data. However, for differential privacy, the setting is totally different. Data is considered as fixed and the statistics of interest is only based on the fixed data. Therefore, there is no randomness in sampling a dataset.

## A.3 Supplementary Experiment

### A.3.1 Full Ranking

To further understand the performance of the top-k selection in DPXPLAIN, we set $k$ to be the maximum size to have a full ranking of all the explanation predicates

and stops DPXPlain at the step of top-k selection. We measure the quality of the full ranking by Kendall-Tau [102]. From Figure A.2, we find that for question G1 its Kendall-Tau is always above 0.4 for privacy budget of topk $\rho_{Topk} \geqslant 0.001$, while for question I1 its Kendall-Tau starts to be above 0.4 when $\rho_{Topk} \geqslant 0.1$. Though the interpretation of Kendall-Tau is not unified, a correlation coefficient above 0.4 indicates a moderate rank association to the true ranking and above 0.7 indicates a strong rank association [9]. However, when we increase the number of conjuncts $l$ from 1 to 2 to 3, the correlation coefficient drops significantly: for I1, it drops from 0.513 to 0.029 to 0.001, and for G1, it drops from 0.947 to 0.466 to 0.060 . This is because increasing the number of conjuncts $l$ will exponentially increase the number of explanation predicates and thus increase the difficulty of a full ranking.

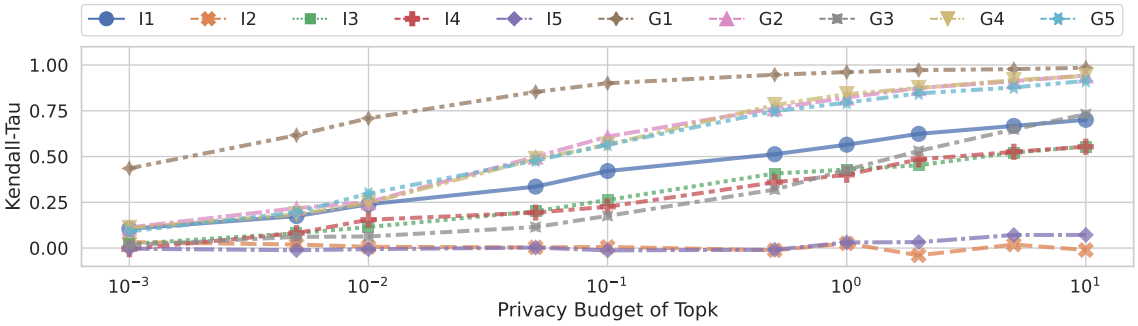



**Figure A.2**: Kendall-Tau of top-k selection by DPXPlain.

# Bibliography


[1] Dpxplain: Explaining query results under differential privacy. https://gitlab.cs.duke.edu/yuchao.tao/Private-Explanation-System. Accessed: 2022-04-26.

[2] New york city taxi and limousine commission (tlc) trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page. Accessed: 2022-01-01.

[3] Tpc benchmark h. https://http://www.tpc.org/tpch/.

[4] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[5] S. Abiteboul, R. Hull, and V. Vianu, editors. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1995.

[6] J. M. Abowd. The us census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2867–2867, 2018.

[7] I. Adler, G. Gottlob, and M. Grohe. Hypertree width and related hypergraph invariants. *European Journal of Combinatorics*, 28(8):2167–2181, 2007.

[8] Y. Ahmad, O. Kennedy, C. Koch, and M. Nikolic. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. *PVLDB*, 5(10):968–979, 2012.

[9] H. Akoglu. User's guide to correlation coefficients. *Turkish journal of emergency medicine*, 18(3):91–93, 2018.

[10] D. Alabi, O. Ben-Eliezer, and A. Chaturvedi. Bounded space differentially private quantiles. *CoRR*, abs/2201.03380, 2022.

[11] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 153–164, 2011.

[12] P. Anderson and J. Cheney. Toward provenance-based security for configuration languages. In U. A. Acar and T. J. Green, editors, *4th Workshop on the Theory and Practice of Provenance, TaPP*, 2012.

[13] A. Ankan and A. Panda. pgmpy: Probabilistic graphical models using python. In *Proceedings of the 14th python in science conference (scipy 2015)*, pages 6–11. Citeseer, 2015.

[14] M. Arapinis, D. Figueira, and M. Gaboardi. Sensitivity of counting queries. *43rd International Colloquium on Automata, Languages, and Programming*, 2016.

[15] H. Asi and J. C. Duchi. Instance-optimality in differential privacy via approximate inverse sensitivity mechanisms. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[16] H. Asi and J. C. Duchi. Near instance-optimality in differential privacy. *arXiv preprint arXiv:2005.10630*, 2020.

[17] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 739–748. IEEE, 2008.

[18] M. Bafna and J. Ullman. The price of selection in differential privacy. In *Conference on Learning Theory*, pages 151–168. PMLR, 2017.

[19] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: A holistic solution to contingency table release. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 273–282. Association for Computing Machinery, Inc., June 2007.

[20] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, July 1983.

[21] G. Bernstein, R. McKenna, T. Sun, D. Sheldon, M. Hay, and G. Miklau. Differentially private learning of undirected graphical models using collective graphical models. In *International Conference on Machine Learning*, pages 478–487. PMLR, 2017.

[22] E. Bertino, G. Ghinita, M. Kantarcioglu, D. Nguyen, J. Park, R. S. Sandhu, S. Sultana, B. M. Thuraisingham, and S. Xu. A roadmap for privacy-enhanced secure data provenance. *J. Intell. Inf. Syst.*, 43(3):481–501, 2014.

[23] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 503–512, 2010.

[24] A. Bittau, U. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 441–459, 2017.

[25] 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*, pages 87–96. ACM, 2013.

[26] A. Blum, K. Ligett, and A. Roth. A learning theory approach to noninteractive database privacy. *Journal of the ACM (JACM)*, 60(2):1–25, 2013.

[27] L. Bonomi and L. Xiong. Mining frequent patterns with differential privacy. *Proceedings of the VLDB Endowment*, 6(12):1422–1427, 2013.

[28] T. Brawner and J. Honaker. Bootstrap inference and differential privacy: Standard errors for free. *Unpublished Manuscript*, 2018.

[29] J. Brownlee. *XGBoost With python: Gradient boosted trees with XGBoost and scikit-learn*. Machine Learning Mastery, 2016.

[30] M. Bun and T. Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, pages 635–658. Springer, 2016.

[31] P. Buneman, S. Khanna, and W.-C. Tan. On propagation of deletions and annotations through views. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 150–158, 2002.

[32] R. S. Carvalho, K. Wang, L. Gondara, and C. Miao. Differentially private top-k selection via stability on unknown domain. In *Conference on Uncertainty in Artificial Intelligence*, pages 1109–1118. PMLR, 2020.

[33] M. Cesar and R. Rogers. Bounding, concentrating, and truncating: Unifying privacy loss composition for data analytics. In *Algorithmic Learning Theory*, pages 421–457. PMLR, 2021.

[34] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.

[35] K. Chaudhuri, D. Hsu, and S. Song. The large margin mechanism for differentially private maximization. *arXiv preprint arXiv:1409.2177*, 2014.

[36] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving programs robust. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 102–112, New York, NY, USA, 2011. ACM.

[37] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. In *Proceedings of the 6th International Conference on Database Theory*, ICDT '97, pages 56–70, 1997.

[38] Q. Chen, C. Xiang, M. Xue, B. Li, N. Borisov, D. Kaarfar, and H. Zhu. Differentially private data generative models. *arXiv preprint arXiv:1812.02274*, 2018.

[39] R. Chen, Q. Xiao, Y. Zhang, and J. Xu. Differentially private high-dimensional data publication via sampling-based inference. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 129–138, 2015.

[40] S. Chen and S. Zhou. Recursive mechanism: towards node differential privacy and unrestricted joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, 2013.

[41] 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*, pages 653–664, 2013.

[42] J. Cheney. A formal framework for provenance security. In *CSF*, pages 281–293, 2011.

[43] G. Cormode, T. Kulkarni, and D. Srivastava. Answering range queries under local differential privacy. *Proceedings of the VLDB Endowment*, 12(10):1126–1138, 2019.

[44] C. Covington, X. He, J. Honaker, and G. Kamath. Unbiased statistical estimation and valid confidence intervals under differential privacy. *arXiv preprint arXiv:2110.14465*, 2021.

[45] S. B. Davidson, S. Khanna, T. Milo, D. Panigrahi, and S. Roy. Provenance views for module privacy. In *PODS*, pages 175–186, 2011.

[46] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich, V. Tannen, and Y. Chen. On provenance and privacy. In *ICDT*, pages 3–10, 2011.

[47] S. B. Davidson, S. Khanna, V. Tannen, S. Roy, Y. Chen, T. Milo, and J. Stoyanovich. Enabling privacy in provenance-aware workflow systems. In *CIDR*, pages 215–218, 2011.

[48] W.-Y. Day, N. Li, and M. Lyu. Publishing graph degree distribution with node differential privacy. In *Proceedings of the 2016 International Conference on Management of Data*, pages 123–138, 2016.

[49] V. A. E. de Farias, F. T. Brito, C. Flynn, J. C. Machado, S. Majumdar, and D. Srivastava. Local dampening: Differential privacy for non-numeric queries via local sensitivity. *Proc. VLDB Endow.*, 14(4):521–533, 2020.

[50] D. Deutch, A. Frankenthal, A. Gilad, and Y. Moskovitch. On optimizing the trade-off between privacy and utility in data provenance. In *SIGMOD*, pages 379–391, 2021.

[51] D. Deutch, N. Frost, and A. Gilad. Explaining natural language query results. *VLDB J.*, 29(1):485–508, 2020.

[52] D. Deutch, N. Frost, A. Gilad, and T. Haimovich. Explaining missing query results in natural language. In *EDBT*, pages 427–430, 2020.

[53] D. Deutch and A. Gilad. Reverse-engineering conjunctive queries from provenance examples. In *EDBT*, pages 277–288, 2019.

[54] A. Differential Privacy Team. Learning with privacy at scale, 2017.

[55] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. *Advances in Neural Information Processing Systems*, 30, 2017.

[56] X. Ding, X. Zhang, Z. Bao, and H. Jin. Privacy-preserving triangle counting in large graphs. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1283–1292, 2018.

[57] Z. Ding, D. Kifer, T. Steinke, Y. Wang, Y. Xiao, D. Zhang, et al. The permute-and-flip mechanism is identical to report-noisy-max with exponential noise. *arXiv preprint arXiv:2105.07260*, 2021.

[58] I. Dinur and K. Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210, 2003.

[59] J. Dong, D. Durfee, and R. Rogers. Optimal differential privacy composition for exponential mechanisms. In *International Conference on Machine Learning*, pages 2597–2606. PMLR, 2020.

[60] W. Dong, J. Fang, K. Yi, Y. Tao, and A. Machanavajjhala. R2t: Instance-optimal truncation for differentially private query evaluation with foreign keys. In *Proc. ACM SIGMOD International Conference on Management of Data*, 2022.

[61] D. Dua and C. Graff. UCI machine learning repository, 2017.

[62] D. Durfee and R. Rogers. One-shot dp top-k mechanisms. DifferentialPrivacy.org, 08 2021. https://differentialprivacy.org/one-shot-top-k/.

[63] D. Durfee and R. M. Rogers. Practical differentially private top-k selection with pay-what-you-get composition. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3527–3537, 2019.

[64] C. Dwork. Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'06, pages 1–12, Berlin, Heidelberg, 2006. Springer-Verlag.

[65] C. Dwork. Differential privacy and the us census. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 1–1, 2019.

[66] C. Dwork and J. Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371–380, 2009.

[67] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[68] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, Aug. 2014.

[69] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[70] C. Dwork and G. N. Rothblum. Concentrated differential privacy. *arXiv preprint arXiv:1603.01887*, 2016.

[71] H. Ebadi and D. Sands. Featherweight PINQ. *J. Priv. Confidentiality*, 7(2), 2016.

[72] B. Efron. *The jackknife, the bootstrap and other resampling plans*. SIAM, 1982.

[73] B. Efron. Bootstrap confidence intervals for a class of parametric problems. *Biometrika*, 72(1):45–58, 1985.

[74] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *Proc. VLDB Endow.*, 8(1):61–72, sep 2014.

[75] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[76] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, July 1983.

[77] C. Ferrando, S. Wang, and D. Sheldon. General-purpose differentially-private confidence intervals. *arXiv preprint arXiv:2006.07749*, 2020.

[78] C. Ferrando, S. Wang, and D. Sheldon. Parametric bootstrap for differentially private confidence intervals, 2021.

[79] S. Flood, M. King, R. Rodgers, S. Ruggles, J. R. Warren, and M. Westberry. Integrated public use microdata series, current population survey: Version 9.0 [dataset]. *Minneapolis, MN: IPUMS*, 2021. https://doi.org/10.18128/D030.V9.0.

[80] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. *SIGPLAN Not.*, 48(1):357–370, Jan. 2013.

[81] J. Gillenwater, M. Joseph, and A. Kulesza. Differentially private quantiles. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 3713–3722. PMLR, 2021.

[82] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[83] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pages 657–666, 2001.

[84] U. Grömping. South german credit data: Correcting a widely used data set. *Rep. Math., Phys. Chem., Berlin, Germany, Tech. Rep*, 4:2019, 2019.

[85] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. *arXiv preprint arXiv:1012.4763*, 2010.

[86] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[87] M. Hay, L. Elagina, and G. Miklau. Differentially private rank aggregation. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 669–677. SIAM, 2017.

[88] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, pages 139–154, 2016.

[89] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[90] M. Herschel and M. A. Hernández. Explaining missing answers to SPJUA queries. *PVLDB*, 3(1):185–196, 2010.

[91] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1):736–747, 2008.

[92] G. Iyer, D. Soberman, and J. M. Villas-Boas. The targeting of advertising. *Marketing Science*, 24(3):461–476, 2005.

[93] B. Jiang, X. Zhang, and T. Cai. Estimating the confidence interval for prediction errors of support vector machine classifiers. *J. Mach. Learn. Res.*, 9:521–540, 2008.

[94] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for sql queries. *Proceedings of the VLDB Endowment*, 11(5):526–539, 2018.

[95] N. M. Johnson, J. P. Near, and D. X. Song. Practical differential privacy for SQL queries using elastic sensitivity. *CoRR*, abs/1706.09479, 2017.

[96] J. Jordon, J. Yoon, and M. Van Der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2018.

[97] B. Kanagal, J. Li, and A. Deshpande. Sensitivity analysis and explanations for robust query evaluation in probabilistic databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 841–852, New York, NY, USA, 2011. ACM.

[98] H. Kaplan, S. Schnapp, and U. Stemmer. Differentially private approximate quantiles. *CoRR*, abs/2110.05429, 2021.

[99] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, 2011.

[100] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In A. Sahai, editor, *Theory of Cryptography*, pages 457–476, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[101] S. P. Kasiviswanathan, K. Nissim, S. Raskhodnikova, and A. Smith. Analyzing graphs with node differential privacy. In *Theory of Cryptography Conference*, pages 457–476. Springer, 2013.

[102] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[103] C. T. Kenny, S. Kuriwaki, C. McCartan, E. T. Rosenman, T. Simko, and K. Imai. The use of differential privacy for census data and its impact on redistricting: The case of the 2020 us census. *Science advances*, 7(41):eabk3283, 2021.

[104] A. Korolova. Privacy violations using microtargeted ads: A case study. In *2010 IEEE International Conference on Data Mining Workshops*, pages 474–482. IEEE, 2010.

[105] I. Kotsogiannis, A. Machanavajjhala, M. Hay, and G. Miklau. Pythia: Data dependent differentially private algorithm selection. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1323–1337, 2017.

[106] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment*, 12(11):1371–1384, 2019.

[107] I. Kotsogiannis, Y. Tao, A. Machanavajjhala, G. Miklau, and M. Hay. Architecting a differentially private sql engine. In *CIDR*, 2019.

[108] P. Laud, M. Pettai, and J. Randmets. Sensitivity analysis of sql queries. In *PLAS@CCS*, 2018.

[109] J. Lee and C. W. Clifton. Top-k frequent itemsets via differentially private fp-trees. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 931–940, 2014.

[110] S. Lee, S. Köhler, B. Ludäscher, and B. Glavic. A sql-middleware unifying why and why-not provenance for first-order queries. In *ICDE*, pages 485–496, 2017.

[111] S. Lee, B. Ludäscher, and B. Glavic. PUG: a framework and practical implementation for why and why-not provenance. *VLDB J.*, 28(1):47–71, 2019.

[112] J. Lei. Differentially private m-estimators. *Advances in Neural Information Processing Systems*, 24, 2011.

[113] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[114] J. Leskovec and J. J. Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547, 2012.

[115] C. Li, Z. Miao, Q. Zeng, B. Glavic, and S. Roy. Putting things into context: Rich explanations for query answers using join graphs. In *SIGMOD*, pages 1051–1063, 2021.

[116] N. Li, W. H. Qardaji, D. Su, and J. Cao. Priv'basis: Frequent itemset mining with differential privacy. *Proc. VLDB Endow.*, 5(11):1340–1351, 2012.

[117] E. Livshits, L. E. Bertossi, B. Kimelfeld, and M. Sebag. The shapley value of tuples in query answering. In *ICDT*, volume 155, pages 20:1–20:19, 2020.

[118] M. Lyu, D. Su, and N. Li. Understanding the sparse vector technique for differential privacy. *Proc. VLDB Endow.*, 10(6):637–648, Feb. 2017.

[119] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. *L*-diversity: Privacy beyond *k*-anonymity. *TKDD*, 1(1):3, 2007.

[120] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[121] I. Masi, Y. Wu, T. Hassner, and P. Natarajan. Deep face recognition: A survey. In *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, pages 471–478. IEEE, 2018.

[122] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *Proc. VLDB Endow.*, 11(10):1206–1219, 2018.

[123] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *PVLDB*, 11(10), 2018.

[124] R. McKenna, D. Sheldon, and G. Miklau. Graphical-model based estimation and inference for differential privacy. In *International Conference on Machine Learning*, pages 4435–4444. PMLR, 2019.

[125] R. McKenna and D. R. Sheldon. Permute-and-flip: A new mechanism for differentially private selection. *Advances in Neural Information Processing Systems*, 33:193–203, 2020.

[126] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 94–103. IEEE, 2007.

[127] 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*, pages 19–30. ACM, 2009.

[128] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[129] A. Meliou, S. Roy, and D. Suciu. Causality and explanations in databases. *Proc. VLDB Endow.*, 7(13):1715–1716, 2014.

[130] Z. Miao, Q. Zeng, B. Glavic, and S. Roy. Going beyond provenance: Explaining query answers with pattern-based counterbalances. In *SIGMOD*, pages 485–502, 2019.

[131] I. Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275. IEEE, 2017.

[132] 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*, pages 75–84, 2007.

[133] K. Nissim, S. Raskhodnikova, and A. D. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, 2007.

[134] C. Palamidessi and M. Stronati. Differential privacy for relational algebra: Improving the sensitivity bounds via constraint systems. In *Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2012, Tallinn, Estonia, 31 March and 1 April 2012.*, pages 92–105, 2012.

[135] D. Phillips. tpch-dbgen. https://github.com/electrum/tpch-dbgen.

[136] W. Qardaji, W. Yang, and N. Li. Priview: practical differentially private release of marginal contingency tables. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1435–1446, 2014.

[137] G. Qiao, W. J. Su, and L. Zhang. Oneshot differentially private top-k selection. *arXiv preprint arXiv:2105.08233*, 2021.

[138] S. Raskhodnikova and A. Smith. Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 495–504. IEEE, 2016.

[139] J. Reed and B. C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, pages 157–168, New York, NY, USA, 2010. ACM.

[140] A. Reshef, B. Kimelfeld, and E. Livshits. The impact of negation on the complexity of the shapley value in conjunctive queries. In D. Suciu, Y. Tao, and Z. Wei, editors, *PODS*, pages 285–297, 2020.

[141] R. Rogers and T. Steinke. A better privacy analysis of the exponential mechanism. DifferentialPrivacy.org, 07 2021. https://differentialprivacy.org/exponential-mechanism-bounded-range/.

[142] L. Rosenblatt, X. Liu, S. Pouyanfar, E. de Leon, A. Desai, and J. Allen. Differentially private synthetic data: Applied evaluations and enhancements. *arXiv preprint arXiv:2011.05537*, 2020.

[143] S. Roy, L. J. Orr, and D. Suciu. Explaining query answers with explanation-ready databases. *Proc. VLDB Endow.*, 9(4):348–359, 2015.

[144] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In C. E. Dyreson, F. Li, and M. T. Özsu, editors, *SIGMOD*, pages 1579–1590, 2014.

[145] P. Ruan, G. Chen, A. Dinh, Q. Lin, B. C. Ooi, and M. Zhang. Fine-grained, secure and efficient data provenance for blockchain. *Proc. VLDB Endow.*, 12(9):975–988, 2019.

[146] S. Ruggles, C. Fitch, D. Magnuson, and J. Schroeder. Differential privacy and census data: Implications for social and economic research. In *AEA papers and proceedings*, volume 109, pages 403–08, 2019.

[147] J. L. C. Sanchez, J. B. Bernabé, and A. F. Skarmeta. Towards privacy preserving data provenance for the internet of things. In *WF-IoT*, pages 41–46, 2018.

[148] S. Shang, T. Wang, P. Cuff, and S. Kulkarni. The application of differential privacy for rank aggregation: Privacy and accuracy. In *17th International Conference on Information Fusion (FUSION)*, pages 1–7. IEEE, 2014.

[149] A. Smith. Privacy-preserving statistical estimation with optimal convergence rates. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 813–822, 2011.

[150] D.-E. Spanos, P. Stavrou, and N. Mitrou. Bringing relational databases into the semantic web: A survey. *Semantic Web*, 3(2):169–209, 2012.

[151] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[152] L. Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, 2002.

[153] Y. S. Tan, R. K. L. Ko, and G. Holmes. Security and data accountability in distributed systems: A provenance survey. In *HPCC/EUC*, pages 1571–1578, 2013.

[154] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang. Privacy loss in apple's implementation of differential privacy on macos 10.12. *arXiv preprint arXiv:1709.02753*, 2017.

[155] Y. Tao, X. He, A. Machanavajjhala, and S. Roy. Computing local sensitivities of counting queries with joins. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 479–494, 2020.

[156] A. G. Thakurta and A. Smith. Differentially private feature selection via stability arguments, and the robustness of the lasso. In *Conference on Learning Theory*, pages 819–850. PMLR, 2013.

[157] The pandas development team. pandas-dev/pandas: Pandas, Feb. 2020.

[158] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD*, pages 15–26, 2010.

[159] J. Vaidya, B. Shafiq, A. Basu, and Y. Hong. Differentially private naive bayes classification. In *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 571–576. IEEE, 2013.

[160] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 137–146, New York, NY, USA, 1982. ACM.

[161] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[162] P. Voigt and A. Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 10(3152676):10–5555, 2017.

[163] L. Wasserman. *All of statistics: a concise course in statistical inference*, volume 26. Springer, 2004.

[164] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson. Differentially private sql with bounded user contribution. *arXiv preprint arXiv:1909.01917*, 2019.

[165] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson. Differentially private sql with bounded user contribution. *ArXiv*, abs/1909.01917, 2019.

[166] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *Proc. VLDB Endow.*, 6(8):553–564, 2013.

[167] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[168] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. *IEEE Transactions on knowledge and data engineering*, 23(8):1200–1214, 2010.

[169] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.

[170] Z. Yan, G. Li, and J. Liu. Private rank aggregation under local differential privacy. *International Journal of Intelligent Systems*, 35(10):1492–1519, 2020.

[171] M. Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB '81, pages 82–94, 1981.

[172] H. Zhang, G. Kamath, J. Kulkarni, and S. Wu. Privately learning markov random fields. In *International Conference on Machine Learning*, pages 11129–11140. PMLR, 2020.

[173] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Private release of graph statistics using ladder functions. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 731–745, New York, NY, USA, 2015. ACM.

[174] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Private release of graph statistics using ladder functions. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 731–745, 2015.

[175] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. Privbayes: Private data release via bayesian networks. *ACM Transactions on Database Systems (TODS)*, 42(4):1–41, 2017.

# Biography

Yuchao Tao was born in Shanghai, China. He received his bachelor degree in computer science from Shanghai Jiao Tong University in 2017. At the same year, he joined Duke as a master student in the computer science department. In 2018, he switched to the Ph.D. program and was advised by Prof. Machanavajjhala on database and privacy. He also had a close collaboration with Prof. Roy. He has published in SIGMOD and VLDB, and won the second place in the NIST 2020 Differential Privacy Temporal Map Challenge.